# PROGRAMACIÓN EN JAVASCRIPT

HERNAN PEÑARANDA V.

materias-hpv.comli.com

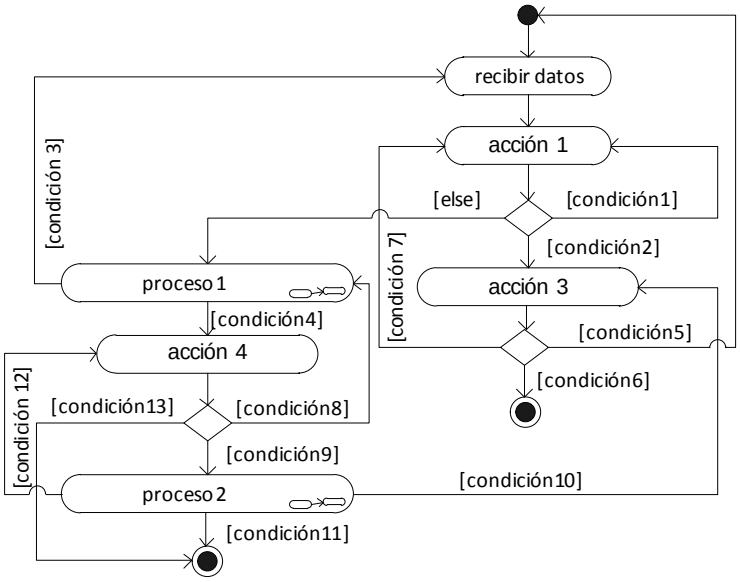
Sucre, 1/2013

# 1. INTRODUCCIÓN

En la asignatura se busca inculcar en el estudiante el hábito de resolver problemas, siguiendo el razonamiento y los principios de la programación estructurada.

La metodología de programación estructurada surgió, como otras metodologías, ante la necesidad de elaborar programas que pudieran ser fácilmente entendidos por un grupo de desarrolladores.

Para comprender la necesidad de contar con una metodología de programación, se debe tomar en cuenta que en los primeros días de la informática los programas eran elaborados directamente en lenguaje de máquina (o en ensamblador). El lenguaje de máquina incentiva y prácticamente obliga al uso de saltos dentro del programa (hacia adelante y hacia atrás). Esta práctica suele da lugar a una lógica confusa que al ser representada en forma de un diagrama revela una gran cantidad de líneas que se cruzan y entrelazan, por lo que esta lógica se conoce con el nombre de "lógica tipo espagueti":




Puesto que la mayoría de los primeros programadores surgieron de ese ambiente, los primeros lenguajes de alto nivel (como Fortran, Basic y C) tenían (y aún tienen) un comando para realizar este tipo de saltos: el comando "goto".

El problema surge cuando es necesario corregir, modificar y/o ampliar un programa, actividades inevitables en la producción de software y que, si no han sido convenientemente planificadas, consumen más tiempo y más recursos que la elaboración del producto original.

Para un mantenimiento eficiente es imprescindible que la lógica de los diferentes programas sea clara, pues de lo contrario resulta más práctico elaborar un nuevo programa que tratar de entender y luego corregir el programa existente.

Es con este fin, que surge la Programación Estructurada (PE), la cual recoge las prácticas de programación que habían demostrado ser exitosas en la elaboración y mantenimiento de programas y que se resumen en los siguientes principios:

- ***Dividir un problema complejo en problemas más sencillos.***
- ***Emplear estructuras estándar para construir la totalidad del programa.***
- ***Emplear tipos de datos a la medida.***

Si bien hoy en día los ambientes de desarrollo son en su mayoría orientados a objetos, los principios de la programación estructurada siguen siendo válidos dentro de los nuevos paradigmas y de hecho soportados por la mayoría de los lenguajes actuales. Pero además esta metodología sigue siendo la metodología dominante en el desarrollo de sistemas operativos (el núcleo de todos los sistemas operativos actuales está escrito en C, un lenguaje estructurado) y el en los sistemas de bases de datos (más del 80% de las bases de datos en actual funcionamiento son relacionales, esto es estructuradas).

A continuación se profundiza un poco más en estos principios.

## 1.1. LA PROGRAMACIÓN MODULAR (DESCENDENTE)

El primer principio, de la programación estructurada es conocido también como programación modular o programación descendente. Su aplicación es universal y expresa la misma intención que la frase "divide y vencerás". Básicamente nos dice que se debe dividir un problema en problemas más pequeños y estos a su vez en otros, hasta que los problemas sean lo suficientemente sencillos como para ser resueltos independientemente.

Cada uno de los problemas en los que se divide el problema principal se conoce como módulo y es la razón por la cual este principio se conoce también con el nombre de "programación modular".

Es importante aclarar que un módulo es considerado como tal sólo si resuelve el problema de manera independiente, es decir sin importar de donde vengan los datos ni donde o como vayan a ser empleados los resultados devueltos.

Este principio se conoce también como programación descendente (top-down) porque se parte de un problema complejo que se divide en problemas más sencillos, estos en otros más sencillos y así sucesivamente, descomponiendo el problema de arriba (del problema más complejo) hacia abajo (al problema más sencillo). No obstante, la solución por lo general es ascendente, pues se comienza por resolver los problemas más sencillos y se va subiendo en dirección a los problemas más complejos.

El dividir el problema en problemas más sencillos, facilita considerablemente el mantenimiento de los programas y e incentiva la reutilización de código. Un módulo que resuelve un problema específico, al ser independiente, puede ser empleado no sólo en el software para el cual fue elaborado, sino también en cualquier otro que deba resolver el mismo problema, es de esa forma que surgen las librerías que contienen módulos que pueden ser reutilizados en varios programas, reduciendo considerablemente el tiempo de desarrollo.

Además los módulos facilitan el mantenimiento y detección de errores: Si en la elaboración de un programa se emplean módulos previamente probados y se produce un error, se sabe que el error no está en dichos módulos, sino en los nuevos módulos añadidos, pero incluso es relativamente sencillo identificar el módulo problemático, con lo que la búsqueda del error se limita, por lo general a un solo módulo.

## 1.2. LAS ESTRUCTURAS ESTÁNDAR

El objetivo del segundo principio es el de promover la elaboración de programas que puedan ser entendidos por todos los programadores. Si en la construcción de un programa se emplean sólo estructuras estándar, resulta más fácil de comprender y en consecuencia mantener, los programas.

Las tres estructuras que deberían emplearse para construir cualquier programa (de acuerdo al teorema de la programación estructurada) son: a) la secuencia (instrucciones consecutivas), b) la selección (*if-else*) y c) la iteración (*while*).

En ocasiones, no obstante, el uso exclusivo de estas tres estructuras puede dar lugar a programas con una lógica innecesariamente complicada, lo que va en contra de la filosofía de este principio, que es la de promover un código claro y sencillo. Por esta razón la mayoría de los lenguajes de programación incorporan estructuras adicionales, pero además permiten modificar, hasta cierto punto, las estructuras estándar.

En esta materia se asumirá que *una estructura es estándar si tiene un solo flujo de entrada y un solo flujo de salida*.

## 1.3. TIPOS DE DATOS A LA MEDIDA

El propósito de este principio es reducir posibles fuentes de error cuando se leen datos introducidos por el usuario o reciben datos en un módulo, así como evitar errores en el programa al devolver resultados con los tipos de datos erróneos.

En la práctica este principio se aplica mediante la validación de datos y resultados, es decir impidiendo, en lo posible, que el usuario introduzca datos del tipo erróneo o que un módulo trabaje con datos erróneos, así como evitando devolver resultados con el tipo incorrecto.

La validación de datos es una de las actividades más importantes en la elaboración de programas, sobre todo al momento de dar funcionalidad a las interfaz del software.

## 1.4. LENGUAJE DE PROGRAMACIÓN A EMPLEAR EN LA MATERIA

Para que una metodología de programación sea de utilidad práctica, tiene que ser empleada en la resolución de problemas y para ello, se debe recurrir a algún lenguaje de programación. El lenguaje de programación que se empleará en la materia es Javascript.

Se ha elegido este lenguaje porque soporta todas las características que se exige a un lenguaje de programación estructurado (incluida la creación de sub-módulos). Además (y con mucho) es el lenguaje más empleado para el desarrollo de páginas y aplicaciones WEB dinámicas.

Hoy en día se está desarrollando un gran número de aplicaciones para el entorno WEB y se prevé que esa sea la norma en un futuro cercano, por lo que, con seguridad, los futuros profesionales del área de la informática, tendrán que recurrir a Javascript en algún momento de su vida profesional.

## 1.5. BREVE INTRODUCCIÓN A JAVASCRIPT

Aunque JavaScript es un lenguaje creado con el propósito de darle funcionalidad a las páginas WEB (crear páginas WEB dinámicas), es en realidad un lenguaje de propósito general, que puede y es empleado, en diferentes campos de la actividad y conocimiento humano.

Quizá la principal ventaja es su universalidad: para hacer correr un programa Javascript sólo se requiere un navegador Internet y los navegadores Internet están disponibles en todos los sistemas operativos actuales, así como en los sistemas operativos y dispositivos programables actuales.

Como lenguaje soporta tanto la programación estructurada como la orientada a objetos y es lo suficientemente potente y general como para permitir la creación de aplicaciones complejas en todos los campos del conocimiento humano.

Para escribir un programa en JavaScript lo único que se requiere es un editor de textos, como el block de notas de Windows (Notepad), aunque se puede recurrir a editores más especializados como Notepad++ (http://notepad-plus-plus.org) para Windows y DroidEdit o WebMaster's HTML (disponibles en la tienda Android: https://play.google.com/ store/) para sistemas Android. Existen también, entornos completos de desarrollo como eclipse (www.eclipse.org/) y NetBeans (http://netbeans.org/).

Como ya se dijo, para ejecutar un programa JavaScript lo único que se requiere es un navegador Internet. Por esta razón, los programas Javascript pueden ser ejecutados en cualquier sistema operativo siempre y cuando cuente con un navegador Internet. Esto significa que un programa escrito en un celular funcionará igualmente en Windows, Linux, Free BSD, OS System (Macintosh), Solaris, Androide, Symbian y en cualquier otra computadora y/o dispositivo programable que cuente con un navegador Internet.

## 1.6. FUNDAMENTOS BÁSICOS DEL LENGUAJE

Javascript es un lenguaje interpretado, es decir que para ejecutar un programa, Javascript interpreta (traduce) la instrucción a lenguaje de máquina (en memoria) ejecuta dicha instrucción, devuelve el resultado de la instrucción y pasa a la siguiente instrucción, donde se repite todo el proceso. Traducir cada instrucción, cada vez que se ejecuta, consume tiempo, por lo que los lenguajes interpretados, como Javascript, son más lentos que los lenguajes compilados, como C++. En estos últimos el programa es traducido a lenguaje de máquina en su integridad y es guardado en un archivo que luego se ejecuta (sin necesidad de traducción pues ya se encuentra en lenguaje de máquina).

En contrapartida, los lenguajes interpretados son más flexibles que los compilados, lo que por una parte hace más fácil su programación y por otra permite crear, evaluar y usar elementos que se crean mientras el programa se ejecuta (algo que es difícil de conseguir en los lenguajes compilados).

En cuanto a su sintaxis, Javascript se parece mucho a C, así al igual que dicho lenguaje **diferencia entre mayúsculas y minúsculas**, por lo tanto "sqrt" es diferente de "Sqrt" o de "sQrt", "SQRT", etc. Igualmente Javascript no toma en cuenta los espacios adicionales (incluido los saltos de línea y tabuladores), aspecto que se aprovecha, al igual que en muchos otros lenguajes, para el escribir el código de forma más legible para el ser humano (mediante la identación, es decir, mediante la práctica de escribir unos espacios más hacia la derecha los bloques de código que estén dentro de una estructura (y sólo dichos bloques).

En este acápite se hace un breve repaso de los fundamentos básicos del lenguaje, que como ya se dijo es muy similar a C, por lo que debe ser estudiado por comparación con dicho lenguaje (***recuerde que, al ser las eva-***

***luaciones con consulta, no es necesario memorizar la información que se proporciona, pero sí comprenderla***).

### *1.6.1. Variables*

Como ocurre con la mayoría de los lenguajes interpretados, las variables en JavaScript pueden almacenar cualquier tipo de información. Las variables toman el tipo de dato en función al valor que almacenan.

Los nombres de las variables pueden estar formados por letras, números y los símbolos "$" y "_". Los nombres de las variables no pueden comenzar con números.

Para declarar una variable se emplea la palabra "**var**" seguida del nombre de la variable (o del nombre de las variables separados con compas, si se quiere declarar más de una variable).

Por ejemplo, las siguientes instrucciones declaran (crean) las variables a, b, c y d:

```
var a;
var b, c, d;
```

Estas variables, sin embargo, no tienen un valor asignado. Una vez declaradas se les puede asignar valores con el operador de asignación "**=**", por ejemplo:

```
a = 4.23;
b = "Javascript";
c = 125;
d = [1,2,3,4];
```

Donde a la variable "a" se le ha asignado un número real, a la variable "b" un texto, a la variable "c" un entero y a la variable "d" un vector. Observe también que después de cada instrucción se ha escrito un punto y coma (";"). En este ejemplo en particular el punto y coma no es imprescindible (porque cada instrucción se encuentra en una línea diferente), sin embargo, para evitar errores, ***se debe escribir un punto y coma al final de cada instrucción***.

Es posible declarar una variable y asignarle un valor inicial al mismo tiempo (variables inicializadas), por ejemplo las anteriores declaraciones y asignaciones se pueden hacer al mismo tiempo con:

```
var a = 4.23, b = "Javascript", c = 125, d = [1,2,3,4];
```

Si no se declara una variable con la palabra "**var**", Javascript crea una variable global, lo que puede dar lugar a errores difíciles de encontrar dentro de los programas, por eso **se recomienda que las variables en Javascript sean declaradas siempre con la palabra "var"**.

### *1.6.2. Operadores*

En JavaScript se pueden emplear los siguientes operadores aritméticos:

**+** : Suma

**-** : Resta

***** : Multiplicación

**/** : División

`%` : Residuo de la división

`++` : Incremento

`--` : Decremento

Los cuatro primeros permiten realizar las operaciones básicas, el quinto "%" devuelve el residuo de la división (entera o real), el operador "++" incrementa en uno el valor de una variable numérica, mientras que "--" disminuye en uno el valor de una variable numérica.

Los dos últimos operadores ("++" y "--") tienen un comportamiento que varía en función a si son escritos como prefijo (antes) o sufijo (después) de la variable: si se encuentran como prefijo primero incrementan (o disminuyen) el valor de la variable y luego realiza las operaciones con el nuevo valor de la variable, mientras que como sufijo primero realiza las operaciones con el valor original de la variable y luego incrementa o disminuye su valor.

A más del operador de asignación ("="), en JavaScript se cuentan con los operadores de asignación compuestos:

`+=` : Suma y asignación (x+=5; equivale a x=x+5;)

`-=` : Resta y asignación (x-=5; equivale a x=x-5;)

`*=` : Multiplicación y asignación (x*=5; equivale a x=x*5;)

`/=` : División y asignación (x/=5; equivale a x=x/5;)

`%=` : Residuo y asignación (x%=5; equivale a x=x%5;)

### 1.6.3. Estructuras "if-else"

La estructura ***if-else*** tiene la siguiente sintaxis:

```
if (condición) {
  instrucciones 1;
} else {
  instrucciones 2;
}
```

Si la condición es verdadera se ejecutan las "instrucciones 1" y si es falsa "las instrucciones 2". Como en otros lenguajes el caso contrario ("else") es opcional. Además, si en las "instrucciones" sólo se tiene una instrucción, no es necesario escribir las llaves, aunque el escribirlas no constituye un error.

### 1.6.4. Operadores relacionales y lógicos

Como en otros lenguajes, la condición se construye empleando operadores relacionales y lógicos. Los operadores lógicos disponibles en Javascript son:

`>` : Mayor que

`<` : Menor que

`==` : Igual a

`===` : Igual y del mismo tipo que

`<=` : Menor o igual a

`>=` : Mayor o igual a

`!=` : Diferente de

**!==** : Diferente o de otro tipo que

Y los operadores lógicos son:

**!** : **No** lógico

**&&** : **Y** lógico

**||** : **O** lógico

El operador "**!**" devuelve verdadero si el valor al que precede es falso y falso si es verdadero. El operador "**&&**" devuelve verdadero sólo si los dos valores que compara son verdaderos, en cualquier otro caso devuelve falso. El operador "**||**" devuelve falso sólo si los dos valores que compara son falsos, en cualquier otro caso devuelve verdadero.

En Javascript cualquier valor diferente de 0 es interpretado como verdadero, mientras que 0 es interpretado como falso, igualmente el texto vacío ("") es interpretado como falso y cualquier otro texto como verdadero. Sin embargo las expresiones relacionales y/o lógicas devuelven el valor lógico "true" (verdadero) o "false" (falso), que si se requiere son convertidos automáticamente por Javascript en 1 o 0.

### *1.6.5. La estructura "switch"*

Si existen dos o más condiciones consecutivas y en las mismas se compara un mismo valor con otros, resulta más eficiente emplear la estructura "switch":

```
switch (n) {
  case valor 1:
    instrucciones 1;
    break;
  case valor 2:
    instrucciones 2;
    break;
  case valor 3:
    instrucciones 3;
    break;
  …
  default:
    instrucciones por defecto;
}
```

Donde "n" es el valor a comparar y "valor 1", "valor 2", etc., son los valores con los que se compara. Si "n" es igual a uno de estos valores se ejecutan las instrucciones que se encuentran dentro de ese caso ("case") y el programa continúa con la instrucción que sigue a "switch" (después de las llaves). Si el caso no tiene un "break", entonces se ejecutan las instrucciones de ese caso, luego las instrucciones del caso siguiente y luego del caso subsiguiente, hasta que se encuentre un "break" o hasta que termina la estructura.

### *1.6.6. El operador "?"*

Si las instrucciones que se ejecutan en base a una condición son simples, se puede emplear el operador "?", en lugar de "if-else". La sintaxis de este operador es la siguiente:

```
r = (condición) ? instrucción_1 : instrucción_2;
```

Donde se ejecuta la "instrucción_1" si la "condición" es verdadera y la "instrucción_2" si es falsa. El resultado devuelto por la "instrucción_1"

o "instrucción_2", es asignado a la variable "r" (o puede ser empleado directamente como argumento de una función o dentro de una expresión).

En realidad es posible escribir más de una instrucción, si se separan con comas y encierran entre paréntesis, en cuyo caso el valor devuelto es el resultado de la última instrucción ejecutada, sin embargo, si se tiene más de una instrucción generalmente es preferible emplear la estructura "if-else" en lugar del operador "?".

### 1.6.7. La estructura "while"

La estructura "while" tiene la siguiente sintaxis:

```
while (condición) {
  instrucciones;
}
```

En esta estructura las "instrucciones" se ejecutan, repetidamente, mientras la "condición" es verdadera. Si se trata de una sola instrucción no son necesarias las llaves, pero el escribirlas tampoco constituye un error.

### 1.6.8. La estructura "do-while"

Esta estructura tiene la siguiente sintaxis:

```
do {
  instrucciones;
} while (condición);
```

Esta estructura tiene la misma lógica que la anterior, excepto que la condición está al final del ciclo, razón por la cual las "instrucciones" se ejecutan por lo menos una vez (mientras que con "while" es posible que no se ejecuten nunca).

### 1.6.9. La estructura "for"

La estructura "for" tiene la siguiente sintaxis:

```
for (inicialización; condición; incremento) {
  instrucciones;
}
```

En esta estructura las "instrucciones" se repiten mientras la "condición" es verdadera, sin embargo, a diferencia de "while", primero se ejecutan las instrucciones escritas en "inicialización" (una sola vez) y antes de repetir el ciclo se ejecutan las instrucciones escritas en "incremento" (cada vez que se repite el ciclo). Normalmente en "inicialización" se escriben las instrucciones que inicializan la o las variables (separadas con comas), mientras que en "incremento" se escriben las instrucciones que incrementan o disminuyen el valor de los contadores.

### 1.6.10. La estructura "for-in"

La sintaxis de esta estructura es:

```
for (atributo in objeto) {
  instrucciones;
```

```
}
```

Esta estructura realiza una iteración por cada elemento que existe en el "objeto", recuperando en la variable "atributo", una a una, las propiedades del objeto. Si el objeto es un vector o texto, los atributos son los índices del vector, es decir números enteros consecutivos, comenzando en 0.

### 1.6.11. Los comandos "break" y "continue"

Para salir del interior de un ciclo, sin que se cumpla la condición del mismo, se escribe el comando "break". Cuando Javascript encuentra este comando, termina inmediatamente la ejecución del ciclo y pasa a ejecutar la siguiente instrucción del programa (la que está después del ciclo).

Para continuar con el siguiente ciclo, sin terminar el ciclo actual, se escribe el comando "continue". Cuando Javascript encuentra el comando "continue" pasa inmediatamente al siguiente ciclo, dejando sin ejecutar las instrucciones restantes.

### 1.6.12. Ciclos infinitos

La aparición de un ciclo infinito en un programa indica la existencia un error: una condición errónea. No obstante, cuando debido a la lógica del problema, se debe salir del ciclo en algún punto intermedio del mismo (no al principio ni al final) se crea un ciclo infinito (escribiendo "true" en lugar de la condición) entonces se sale del ciclo, desde cualquier punto intermedio del mismo, empleando el comando "break".

La sintaxis de un ciclo que termina en algún punto intermedio del mismo es aproximadamente la siguiente:

```
while (true) {
  instrucciones;
  if (condición) break;
  instrucciones;
}
```

Que puede ser implementada también con las estructuras "do-while" y "for":

```
do {
  instrucciones;
  if (condición) break;
  instrucciones;
} while (true);

for (;;) {
  instrucciones;
  if (condición) break;
  instrucciones;
}
```

Observe que en el caso de la estructura "for", para crear un ciclo infinito, simplemente se dejan en blanco los sectores de inicialización, condición e incremento.

### 1.6.13. Funciones

En Javascript las funciones pueden ser creadas de tres formas. La forma más usada en la mayoría de los casos es la forma estándar:

```
function nombre_de_la_funcion(lista_de_parámetros) {
  instrucciones;
}
```

Donde "nombre_de_la_función" es el nombre que se da a la función y cuya denominación sigue las mismas reglas que las variables.

La "lista_de_parámetros" son los nombres de las variables donde se guardan los datos que se mandan a la función. Dichos nombres deben estar separados por comas.

Para que devuelver un resultado desde una función se emplea el comando "return", de acuerdo a la sintaxis:

```
return valor_o_expresión;
```

Cuando Javascript encuentra este comando termina la ejecución de la función y devuelve el "valor_o_expresión". El "valor_o_expresión" puede ser un valor literal (por ejemplo un número o texto), una variable (cuyo valor es devuelto) o una expresión (que es evaluada y cuyo resultado es devuelto).

Si en una función no se escribe el comando "return", la función termina al ejecutarse su última línea de código sin devolver ningún resultado.

Como se dijo, en la calculadora Javascript no se pueden crear funciones en la forma estándar, por lo que deben ser creadas en la forma literal, de acuerdo a la siguiente sintaxis:

```
var variable=function nombre_de_la_funcion(lista_de_parámetros) {
  instrucciones;
}
```

Como se puede ver, la única diferencia con relación a la forma estándar, es que la función es asignada a una variable (se le da un alias), entonces la función puede ser llamada indistintamente con el "nombre_de_la_función" o el "nombre_de_la_variable" (el alias). Por eso, en esta forma, casi siempre se omite el "nombre_de_la_función" (es decir que se crea una función anónima).

```
var variable=function (lista_de_parámetros) {
  instrucciones;
}
```

Finalmente una función puede ser creada también de la siguiente forma:

```
var variable = new Function("par1","par2",…,"parn", "instrucciones");
```

Donde "par1", "par2",…,"parn" son los parámetros de la función e "instrucciones" son las instrucciones Javascript, es decir el código de la función. Esta forma permite crear funciones en tiempo de ejecución: funciones dinámicas, las cuales, a diferencia de las formas estándar y literal, son siempre globales. Esta forma de crear funciones Javascript no se utiliza muy frecuentemente.

En Javascript (a diferencia de C) las funciones pueden ser anidadas, es decir se puede crear una función dentro de otra (excepto, por supuesto con la última forma, que siempre crea funciones globales).

## 1.7. ESCRITURA DE CÓDIGO JAVASCRIPT

Una vez repasados los fundamentos del lenguaje, se pueden elaborar algunos programas básicos en Javascript.

Como ya se dijo, Javascript es un lenguaje que corre en los navegadores

Internet, por lo tanto, un programa Javascript, debe ser escrito y/o importado en una página HTML. Ello implica aprender también algo de HTML (el lenguaje de las páginas WEB).

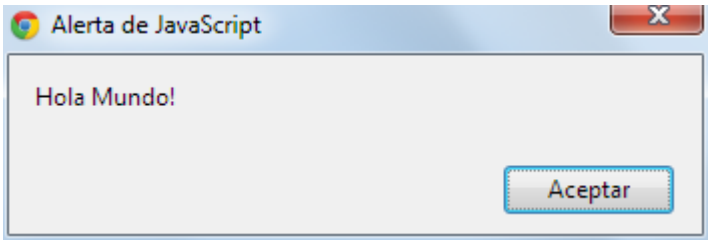
Las instrucciones en HTML se escriben dentro de etiquetas. Las etiquetas son palabras reservadas que se encierran entre los símbolos "< >". La mayoría de las etiquetas tienen una etiqueta de apertura y otra de cierre. La etiqueta de cierre se diferencia de la de apertura porque está precedida por el símbolo "/", por ejemplo todas las etiquetas de un página HTML se encierran dentro de la etiqueta "html":

```
<html>
. . .
</html>
```

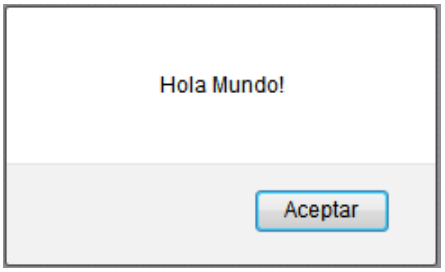
La etiqueta dentro la cual se escribe (o importa) código Javascript es "script". En la mayoría de los navegadores es suficiente escribir esta etiqueta para que sea reconocido como código (instrucciones) Javascript. Así por ejemplo si se crea el archivo "hola.html", en Notepad, Notepad++, DroidEdit o cualquier otro editor de textos y se escribe dentro del mismo lo siguiente:

```
<script>
  alert("Hola Mundo!");
</script>
```

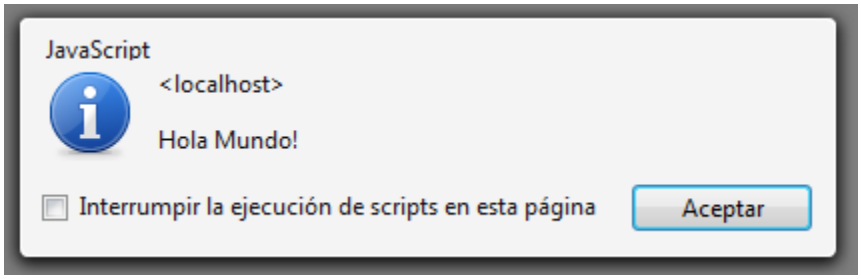
Al abrir el archivo, en un navegador, aparece una ventana de alerta con el mensaje "Hola Mundo!" (la instrucción "alert" de Javascript, sirve justamente para ese fin). La apariencia de esta ventana varía de un navegador a otro, pero no su funcionalidad, así en Google Chrome es:




En Firefox:

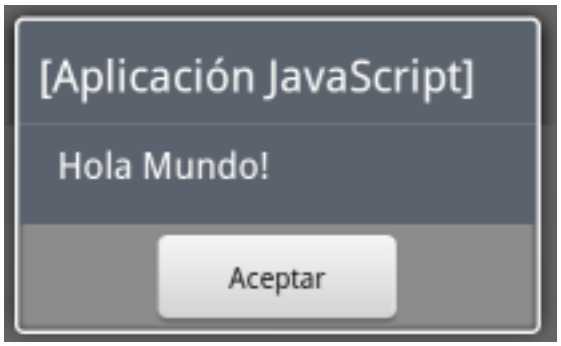



Y en Opera:



Por supuesto, la apariencia varía también en los dispositivos móviles, así en Dolphin (en los dispositivos Android) es:

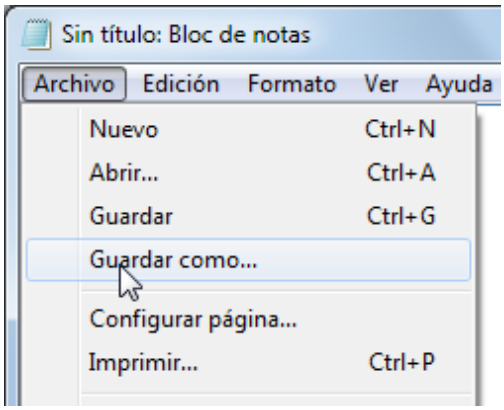

En Firefox:



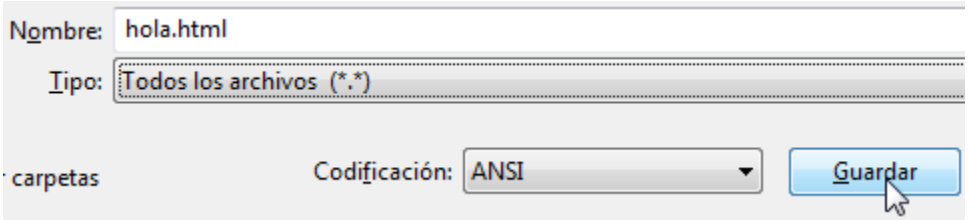

Y en Opera:


UC Browser tiene algún defecto (un bug) con las ventanas emergentes pues no las muestra ni genera ningún mensaje de error.

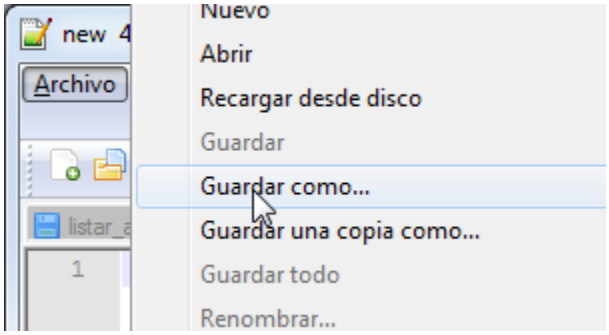
Ahora se verá con más detalle cómo crear un archivo HTML. Una vez abierto el editor, simplemente se elige la opción archivo->nuevo y se guarda el archivo con la extensión ".html". Así en Notepad (en el block de notas), se elige la opción "Guardar como…":



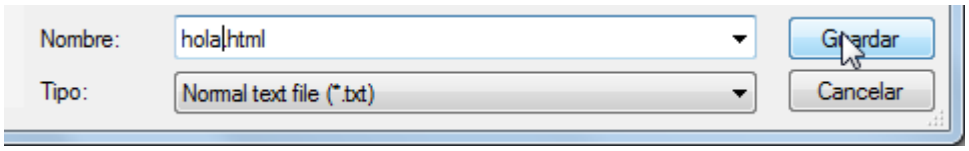
En la ventana que aparece se elige la carpeta en la cual se quiere guardar el archivo y se guarda el archivo con el nombre "hola.html":



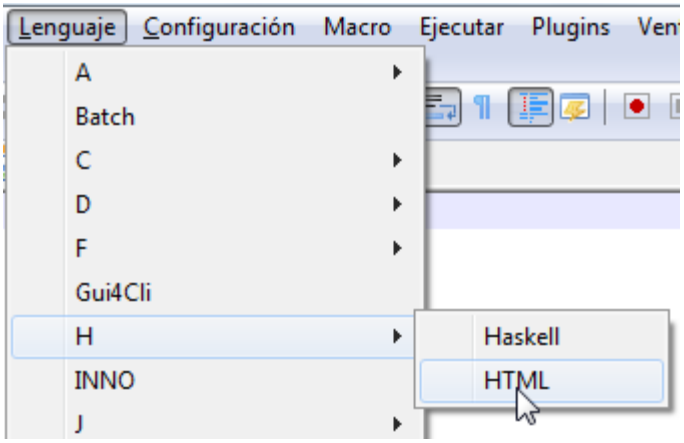

Igualmente, en Notepad++, después de crear una nueva página ("Ctrl+N" o menú Archivo->Nuevo), se elige la opción "Guardar como…"



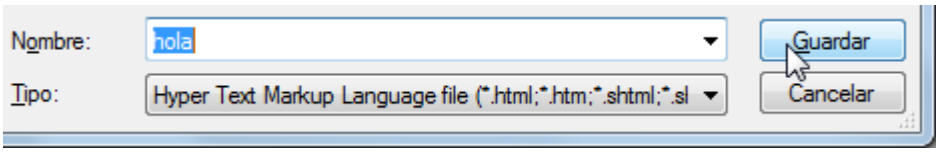

Y se guarda el archivo con la extensión ".html":




Alternativamente, en Notepad++, se puede elegir primero el lenguaje:




Y luego guardar el archivo (Archivo->guardar) con el nombre "hola" (la extensión "html") se añade automáticamente:

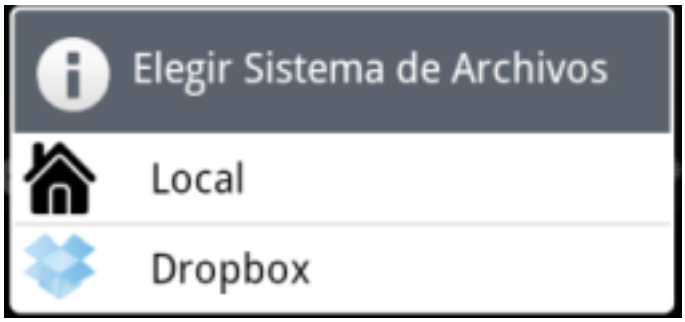



En los dispositivos móviles con sistema Android, si se está trabajando con DroidEdit, se abre el editor, haciendo tap (clic) sobre el icono de la aplicación:


Por supuesto, y como ya se dijo, se pueden emplear otros editores como WebMaster's HTML, DreamEdit, Syntax Highlighted, etc. Una vez en el navegador se crea un nuevo archivo haciendo tap (clic) en el icono respectivo, que en el caso de DroidEdit es:

Entonces se escribe algún carácter (para que se habilite el icono "guardar") y se guarda el archivo haciendo tap en el icono guardar, que en el caso de DroidEdit es:

Entonces en la ventana que aparece se hace tap en "Local" (es decir que se elige guardar el archivo en el dispositivo móvil):




Y en la nueva ventana que aparece se elige la carpeta en la que se quiere guardar el archivo y se crea el archivo con el nombre "hola.html":

Como se ha visto, es posible escribir código Javascript simplemente con la etiqueta "script", sin embargo, para crear programas realmente útiles, es necesario emplear algunas otras etiquetas más. La estructura HTML básica que se empleará para escribir programas en Javascript, es:

```
<html>
<head>
  <title>Título de la página</title>
  <script type="text/javascript">
    código Javascript
  </script>
</head>
<body>
  Etiquetas HTML
  <script type="text/javascript">
    código Javascript
  </script>
  Etiquetas HTML
</body>
</html>
```

Como ya se explicó, la etiqueta "html" le indica al navegador que su contenido es HTML.

La etiqueta "head" identifica el encabezado de la página. El código Javascript escrito en esta parte, así como otros elementos que se incluyen en esta parte, se cargan en primer lugar.

La etiqueta "title" sirve para darle un título a la página, este título es el que aparece en la pestaña del navegador, ayudando a identificar la página (por supuesto no es imprescindible).

La etiqueta "body" identifica el cuerpo del documento. Es en esta parte donde se escribe la mayoría de las instrucciones HTML.

Como se puede observar, las instrucciones Javascript pueden ser escritas tanto en el encabezado como en el cuerpo del documento. Es posible también escribir varios bloques de código en diferentes partes del documento (dentro de etiquetas "script") y en ese caso son ejecutados secuencialmente, en el orden en que fueron escritos.

Sin embargo, es aconsejable escribir las instrucciones en un solo bloque: en el encabezado del documento (head). De esa manera se garantiza que las funciones, estén disponibles para su uso desde un principio, además se facilita su mantenimiento y se evita entremezclar código Javascript con instrucciones HTML.

En realidad, lo más aconsejable es escribir el código Javascript en un archivo independiente (con la extensión ".js") e importar el mismo en la página HTML con la siguiente etiqueta:

```
<script type="text/javascript" src="nombre_del_archivo_Javascript.js"></script>
```

Donde se reemplaza "nombre_del_archivo_Javascript" por el nombre real del archivo que contiene el código Javascript.

El atributo "type", que se ha escrito en todas las etiquetas "script", no es realmente necesario. La mayoría de los navegadores actuales asumen que, si no existe el atributo "type", se trata de código Javascript, pero, por razones de claridad y universalidad, es conveniente escribirlo.

Con estas consideraciones se reescribe el programa "Hola Mundo!", teniendo en Notepad++ la siguiente apariencia:

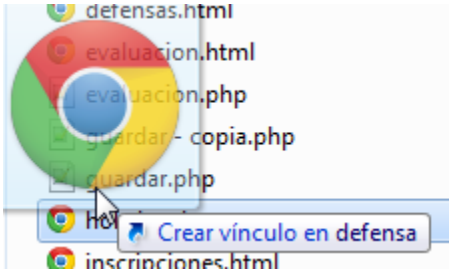
```
1  <html>
2  <head>
3    <title>Hola Mundo</title>
4    <script type="text/javascript">
5      alert("Hola Mundo!");
6    </script>
7  </head>
8  <body>
9  </body>
10 </html>
```

Y en DroidEdit:

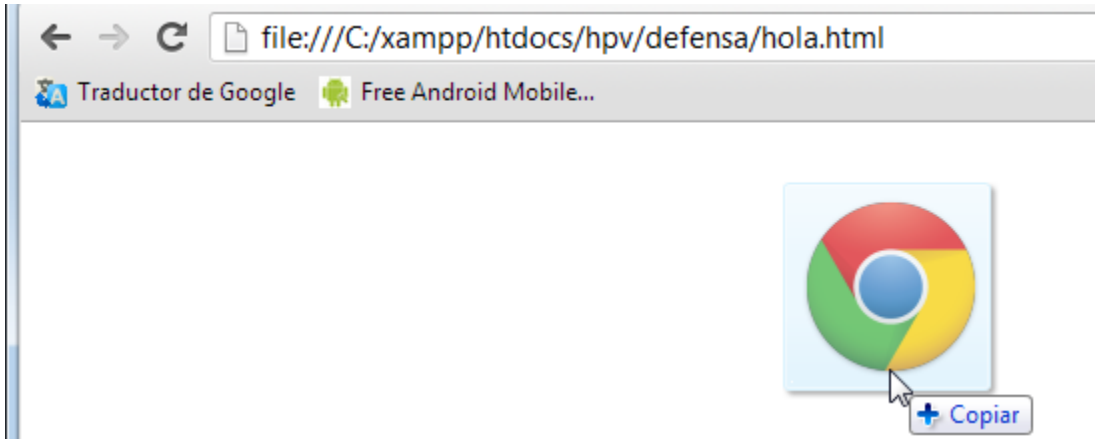
```
1  <html>
2  <head>
3    <title>Hola Mundo</title>
4    <script type="text/javascript">
5      alert("Hola Mundo!");
6    </script>
7  </head>
8  <body>
9  </body>
10 </html>
```

Por supuesto este programa hace lo mismo que el anterior (con excepción del título de la página). ***No debe olvidar "guardar" el archivo una vez escrito el código***.

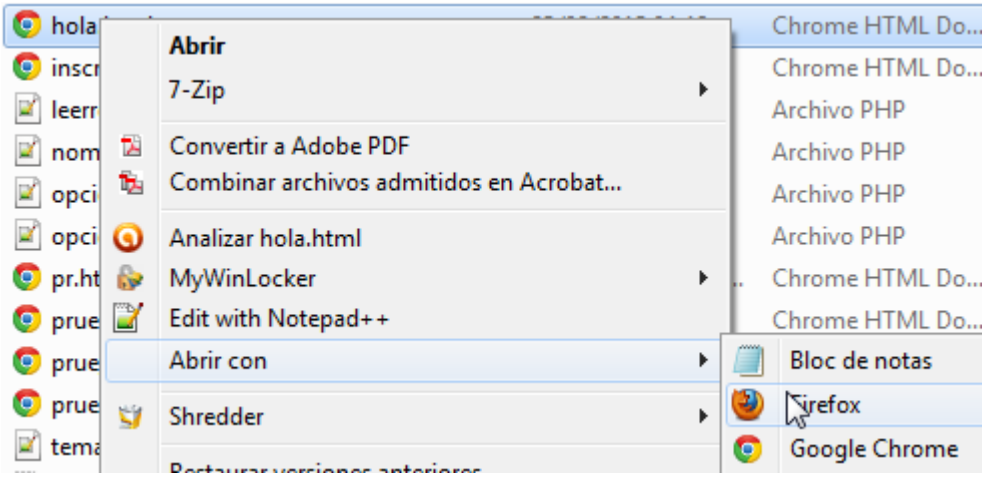
Una vez creado el archivo HTML debe ser abierto en un navegador. Y para ello existen varias alternativas, la más sencilla (en Windows) consiste en arrastrar el icono del archivo:



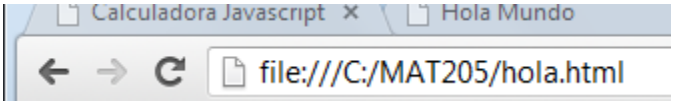

Y soltarlo sobre la ventana del navegador:



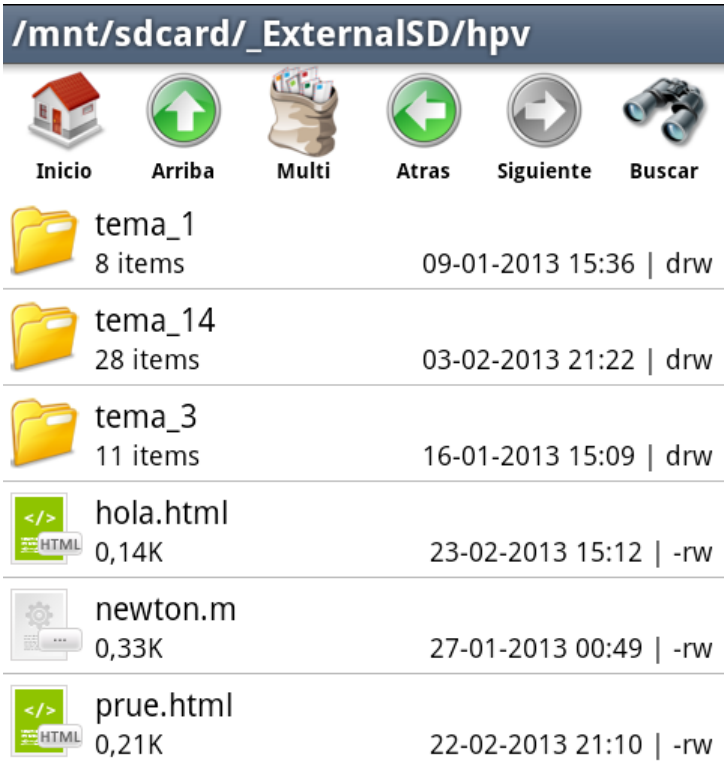
También se puede hacer clic sobre el archivo, con el botón derecho del mouse y en el menú emergente se elige "abrir con..", seleccionando luego el navegador con el cual se quiere abrir la página:



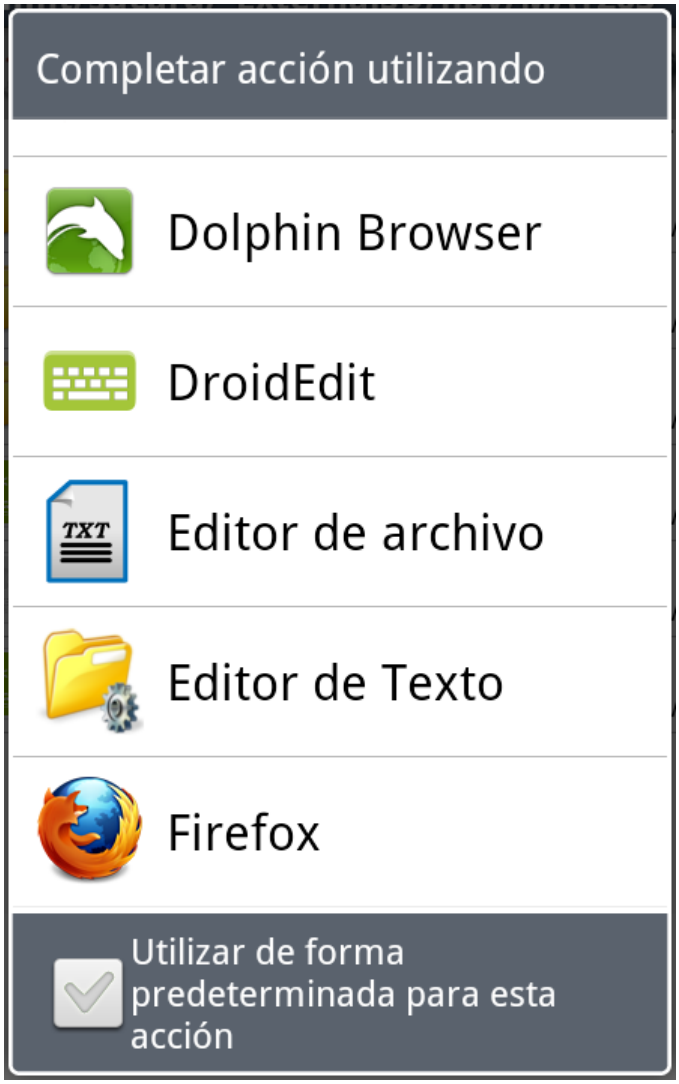

Es posible también escribir directamente, en el campo de direcciones del navegador, el camino donde se encuentra el archivo, precediendo dicho camino con la palabra file:///, así si el archivo se encuentra en C:\MAT205\hola.html, se escribe:




En los dispositivos móviles con sistema Android, se abre la página ubicando el mismo con un administrador de archivos (como File Manager):




Entonces se hace tap en el archivo (en "hola.html") y en el menú que aparece se hace tap en el navegador con el cual se quiere abrir la página:




Es posible también escribir en el navegador el camino donde se encuentra el archivo (de manera similar a como se procede en Windows).

A más del editor de textos, ***en los dispositivos Android se recomienda instalar el teclado "Multiling"*** (bajándolo del play store). Este teclado está disponible en varios idiomas y tiene una configuración que facilita considerablemente la escritura de código.

## 1.8. EJERCICIOS

1. Cree la página "saludo.html" que muestre un mensaje de alerta con su nombre completo.
2. Cree la página "info.html" que muestre un primer mensaje de alerta (escrito en el encabezado de la página) con el número de su carnet de identidad y un segundo mensaje (escrito en el cuerpo del documento) con el número de su carnet universitario.

## 1.9. RESOLUCIÓN DE PROBLEMAS EN JAVASCRIPT

Una vez que se conocen los fundamentos tanto de la programación estructurada como del lenguaje, se puede comenzar a resolver problemas en Javascript.

En primer lugar, es necesario tomar en cuenta que como Javascript soporta la programación orientada a objetos (a más de la programación estructurada), la mayoría de los recursos que posee han sido implementados en forma de objetos, ahora bien, para acceder (utilizar) las funciones y/o datos de un objeto, se debe escribir el nombre del objeto, un punto y el nombre de la función, es decir:

```
Nombre_del_objeto.Nombre_de_la_función_o_dato
```

Así por ejemplo las funciones matemáticas han sido definidas dentro del objeto "Math", por tanto, si se quiere calcular la raíz cuadrada de 4.52 se debe escribir:

```
Math.sqrt(4.52)
```

A propósito, las funciones matemáticas disponibles en el objeto "Math" son:

| | |
|---|---|
| sin(x): | Seno de en ángulo en radianes "x". |
| cos(x): | Coseno del ángulo en radianes "x". |
| tan(x): | Tangente del ángulo en radianes "x". |
| asin(x): | Arco seno del valor x, valor devuelto en radianes. |
| acos(x): | Arco coseno del valor x, valor devuelto en radianes. |
| atan(x): | Arco tangente del valor x, valor devuelto en radianes. |
| atan2(x,y): | Arco tangente del cociente de "x" y "y". |
| log(x): | Logaritmo natural de "x" ($\log_e(x)$) |
| exp(x): | Exponente de "x" ($e^x$). |
| sqrt(x): | Raíz cuadrada de "x". |
| pow(x,y): | Resultado de elevar "x" a "y" ($x^y$). |
| abs(x): | Valor absoluto de "x" (x sin signo). |
| round(x): | Valor de "x" redondeado al entero más cercano (el entero mayor si la parte fraccionaria es mayor o igual a 0.5). |
| ceil(x): | Valor de "x" redondeado al entero mayor inmediato. |
| floor(x): | Valor de "x" redondeado al entero menor inmediato. |
| random(): | Número real aleatorio comprendido entre 0 y 1. |

Donde también se ha definido las siguientes constantes matemáticas:

| | |
|---|---|
| E: | Número de Euler. |
| LN2: | Logaritmo natural de 2. |
| LN10: | Logaritmo natural de 10. |
| LOG2E: | Logaritmo del número de Euler en base 2. |
| LOG10E: | Logaritmo de Euler en base 10. |
| PI: | Número pi. |
| SQRT1_2: | Raíz cuadrada de 1/2. |
| SQRT2: | Raíz cuadrada de 2. |

En programación orientada a objetos las funciones implementadas dentro de los métodos se conocen con el nombre de "métodos", mientras que los datos declarados en el objeto se conocen con el nombre de "propiedades" o "atributos". Entonces, más propiamente, las funciones matemáticas como "sin", "cos", etc., son métodos del objeto "Math", mientras que las constantes como "E", "LN2" son los "atributos" de dicho objeto.

En algunos casos, sobre todo cuando se trabaja con expresiones matemáticas, resulta tedioso tener que escribir, una y otra vez el nombre del objeto. Afortunadamente, en Javascript es muy simple asignar otro nombre (un alias) a un método o atributo. Por ejemplo, si se quiere calcular la raíz cuadrada con "sqrt" en lugar de "Math.sqrt", se escribe:

```
sqrt = Math.sqrt;
```

Después de esta asignación, la raíz cuadrada de 4.52 se calcula simplemente con:

```
sqrt(4.52)
```

Sin embargo se debe tomar en cuenta que este alias es local, es decir que sólo existe y por lo tanto puede ser empleado, en la página donde ha sido creado.

Si se piensa emplear frecuentemente un conjunto de alias (como por ejemplo alias para todas las funciones matemáticas), lo más eficiente es crear dichos alias en una librería Javascript. Una librería Javascript simplemente es un archivo de texto, guardado con la extensión ".js", donde se escriben (sin la etiqueta "script") todas las instrucciones Javascript que se quiere estén disponibles para las páginas que así lo requieran.

Generalmente en una librería Javascript se crean objetos, funciones, constantes, alias, etc., sin embargo, es posible escribir cualquier instrucción Javascript válida, como "alert('Hola Mundo!')" por ejemplo.

Existe un gran número de librerías Javascript disponibles (a través de Internet) para diferentes propósitos. En la materia se emplearán algunas de dichas librerías, además, se ha creado la librería "sis101.js" (que se distribuye con el material del tema) donde en primera instancia, se han escrito alias para todas las funciones matemáticas y se han añadido además las siguientes funciones matemáticas no disponibles en el objeto "Math".

| | |
|---|---|
| csc(x) | Cosecante de x. |
| sec(x) | Secante de x. |
| cot(x) | Cotangente de x. |
| sinh(x) | Seno hiperbólico de x. |
| cosh(x) | Coseno hiperbólico de x. |
| tanh(x) | Tangente hiperbólica de x. |
| csch(x) | Cosecante hiperbólica de x. |
| sech(x) | Secante hiperbólica de x. |
| coth(x) | Cotangente hiperbólica de x. |
| asinh(x) | Arco seno hiperbólico (seno hiperbólico inverso) de x. |
| acosh(x) | Arco coseno hiperbólico (coseno hiperbólico inverso) de x. |
| atanh(x) | Arco tangente hiperbólico de x. |
| sqr(x) | Cuadrado de x. |
| sign(x) | Signo de x: 1 positivo, -1 negativo, 0 cero. |
| cbrt(x) | Raíz cúbica de x. |
| odd(x) | Impar. Verdadero si x es impar, falso en caso contrario. |
| even(x) | Par. Verdadero si x es par, falso en caso contrario. |
| xroot(x,n) | Raíz enésima (n) de x. |
| quot(x,y) | Cociente de la división entre "x" y "y". |
| frac(x) | Parte fraccionaria de x. |
| isInteger(x) | Es entero. Veradero si x es un entero. |
| toDeg(x) | Convierte x de radianes a grados. |
| toRad(x) | Convierte x de grados a radianes. |
| log10(x) | Logaritmo en base 10 de x. |
| log2(x) | Logaritmo en base 2 de x. |
| logb(b,x) | Logaritmo en base "b" de x. |
| fact(n) | Factorial de n. |

Esta librería cuenta además con otras funciones y a medida que se avance en la materia, se le irán añadiendo otras más, por lo que la misma será distribuida conjuntamente el material de los diferentes temas (pues se modificará frecuentemente).

Como ya se explicó, para emplear una librería en una página HTML, simplemente se la importa en la cabecera de la página HTML, dentro de la etiqueta "script":

```
<script type="text/javascript" src="mat205.js"></script>
```

Por ejemplo, si el problema es calcular el valor de la siguiente expresión matemática para para valores de "y" iguales a 3.1, 4.2 y 7.3:

$$\sqrt[3]{\frac{y+4!+y^{3.2}}{e^{y}+\cos(y)}}$$

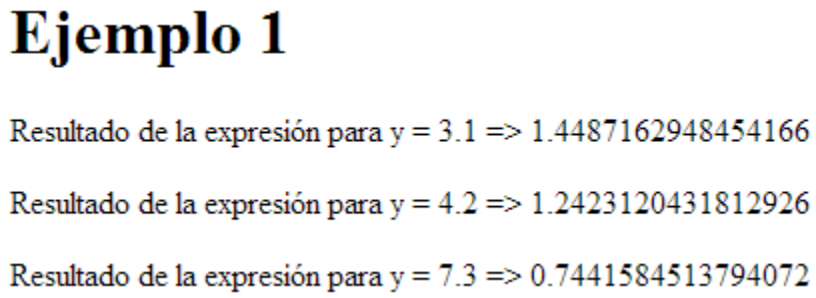
Lo primero que se debe hacer es un análisis del problema. En este caso se debe evaluar la expresión 3 veces, por lo tanto (aplicando el primer principio de la programación estructurada), se deduce que es necesario un módulo (una función) que reciba como dato el valor de "y" y devuelva como resultado el valor de la expresión.

Por otra parte, dado que no existe ninguna condición, no se requiere ninguna estructura estándar (segundo principio) y como los datos son constantes, tampoco es necesario validarlos (tercer principio).

Tomando en cuenta las anteriores consideraciones se resuelve el problema creando la siguiente página HTML (ejemplo1.html):

```
1  <html>
2  <head>
3     <title>Ejemplo 1</title>
4     <script type="text/javascript" src="sis101.js"></script>
5     <script type="text/javascript">
6        function fy(y){
7           return cbrt((y+fact(4)+pow(y,3.2))/(exp(y)+cos(y)));
8        }
9     </script>
10 </head>
11 <body>
12    <h1>Ejemplo 1</h1>
13    <p>Resultado de la expresión para y = 3.1 =>
14       <script type="text/javascript">
15          document.write(fy(3.1));
16       </script>
17    </p>
18    <p>Resultado de la expresión para y = 4.2 =>
19       <script type="text/javascript">
20          document.write(fy(4.2));
21       </script>
22    </p>
23    <p>Resultado de la expresión para y = 7.3 =>
24       <script type="text/javascript">
25          document.write(fy(7.3));
26       </script>
27    </p>
28 </body>
29 </html>
```

Que al ser abierta en el navegador muestra lo siguiente:




Si en el navegador no aparecen los resultados esperados, lo más probable es que se haya cometido algún error al escribir el código Javascript, por lo que debe ser revisado. Por defecto los navegadores Internet no muestran ningún mensaje cuando se produce algún error1. en la ejecución del código Javascript, por lo tanto, ***si no aparecen los resultados buscados, se debe asumir que se ha cometido algún error y revisar el código***. El cometer errores de escritura (sintaxis) es lo más frecuente cuando se programa y es más frecuente en los primeros programas, por eso es necesario acostumbrarse a revisar el código, encontrar los errores y corregirlos, tratando de evitar volver a cometerlos.

En la solución del problema se han empleado dos nuevas etiquetas HTML: "h1" y "p".

La etiqueta "h1" muestra, el texto que se escribe en su interior, como un encabezado de primer nivel (un título). En HTML se tienen etiquetas tipo encabezado hasta un sexto nivel: "h2", "h3", "h4", "h5" y "h6".

La etiqueta "p", por otro lado, muestra el texto que se escribe en su interior como un párrafo, es decir que añade un salto de línea (y cierto espaciamiento) antes y después del texto.

Observe que, como era de esperar, la expresión matemática ha sido evaluada sin necesidad de escribir "Math.". Es importante, para que esto suceda, es necesario que ***el archivo "sis101.js" (la librería) se encuentre en el mismo directorio que la página HTML***.

En esta página se ha empleado también una nueva instrucción Javascript: "document.write". Esta instrucción inserta en la página, el texto que se escribe, como código HTML. El objeto "document" representa a la página HTML y a más de "write" cuenta con otras propiedades y métodos, algunos de los cuales serán estudiados posteriormente.

En el ejemplo, la instrucción "document.write" transforma, automáticamente, el resultado devuelto por la función "fy" en texto e inserta dicho texto en la página (dentro de las etiquetas "p").

Puesto que "document.write" permite insertar instrucciones HTML en la página, es posible crear todo el contenido de la página directamente en Javascript, evitando así entremezclar etiquetas HTML con código Javascript.

Por ejemplo, si el problema es encontrar el valor de la siguiente expresión:

$$\frac{\log\left(x+\sqrt{y+z}\right)}{\sqrt[5]{x+y^{z}}}$$

Para los siguientes valores de "x", "y" y "z": x=1.1, y=1.4, z=1.9;

x=2.3, y=5.2, z=8.3; x=1.9, y=1.2, z=3.1. Se puede resolver el problema (el análisis es el mismo que el anterior problema) creando la siguiente página HTML (ejemplo2.html)

```
<html>
<head>
  <script type="text/javascript" src="sis101.js"></script>
  <script type="text/javascript">
    function fxyz(x,y,z){
      return log10(x+sqrt(y+z))/xroot(x+pow(y,z),5);
    }
    document.write("<h1>Ejemplo 2</h1>");
    var x=1.1, y=1.4, z=1.9;
    document.write("<p>Valor de la expresión para x="+x+
      ", y="+y+", z="+z+" => "+fxyz(x,y,z)+"</p>");
    x=2.3, y=5.2, z=8.3;
    document.write("<p>Valor de la expresión para x="+x+
      ", y="+y+", z="+z+" => "+fxyz(x,y,z)+"</p>");
    x=1.9, y=1.2, z=3.1;
    document.write("<p>Valor de la expresión para x="+x+
      ", y="+y+", z="+z+" => "+fxyz(x,y,z)+"</p>");

  </script>
</head>
<body>
</body>
</html>
```

Que al ser abierta en un navegador muestra:

# Ejemplo 2

Valor de la expresión para x=1.1, y=1.4, z=1.9 => 0.373295627961028

Valor de la expresión para x=2.3, y=5.2, z=8.3 => 0.0502867873527808

Valor de la expresión para x=1.9, y=1.2, z=3.1 => 0.4622452740293394

Como se puede observar en este programa, para concatenar (unir) texto se emplea el operador de suma ("+"), además, Javascript convierte automáticamente los valores numéricos a texto.

Como ya se explicó, Javascript ignora los espacios en blanco adicionales (incluyendo saltos de línea y tabulaciones). En este programa se ha aprovechado ese hecho para dividir, en dos líneas, las instrucciones "document.write" de manera que quepan en la pantalla, sin embargo, el programa funciona igual si dichas instrucciones son escritas en una sola línea.

Observe también que las variables "x", "y" y "z" han sido declaradas una sola vez (con "var"), pues una vez declaradas se les puede asignar nuevos valores, sin necesidad de volver a declararlas, las veces que se requiera.

Como es lógico suponer y al igual que en el anterior programa, al crear este programa las instrucciones han sido escritas una sola vez: para la primera serie de valores y luego simplemente copiadas y modificadas para las restantes.

Supongamos ahora que el problema consiste en calcular la sumatoria de los primeros 15, 30 y 50 números impares.

Para resolver el problema se requiere un ciclo iterativo que se repita 15, 30 y 50 veces respectivamente, por lo tanto se requiere una estructura iterativa (segundo principio) y dado que se conoce el número de repeticiones, la estructura iterativa más adecuada es "for". En cada una de las repeticiones, el contador del ciclo debe ser incrementado en 2 y su valor añadido al acumulador (el acumulador es la variable donde se guarda la sumatoria y cuyo valor comienza en cero). El ciclo debe repetirse hasta que el contador sea igual a 2*n-1 (que es el último número impar de los primeros "n" número simpares).

Como el proceso se repite 3 veces, para los 3 valores dados, debe ser programado en un módulo (primer principio).

Al igual que en los anteriores ejemplos, no se realiza ninguna validación (tercer principio), porque los datos son constantes.

Con las anteriores condiciones se crea la siguiente página HTML (ejemplo3.html) para resolver el problema:

```
1  <html>
2  <head>
3    <title>Ejemplo 3</title>
4    <script type="text/javascript">
5      function sumai(n){
6        var s=0;
7        for (var i=1;i<2*n;i+=2)
8          s+=i;
9        return s;
10     }
11     document.write("<h1>Ejemplo 3</h1>");
12     document.write("<p>Sumatoria de los primeros 15 números "+
13       "impares => "+sumai(15)+"</p>");
14     document.write("<p>Sumatoria de los primeros 30 números "+
15       "impares => "+sumai(30)+"</p>");
16     document.write("<p>Sumatoria de los primeros 50 números "+
17       "impares => "+sumai(50)+"</p>");
18   </script>
19 </head>
20 <body>
21 </body>
22 </html>
```

Que al ser abierto en un navegador, muestra los siguientes resultados:

# Ejemplo 3

Sumatoria de los primeros 15 números impares => 225

Sumatoria de los primeros 30 números impares => 900

Sumatoria de los primeros 50 números impares => 2500

## 1.10. EJERCICIOS

Para presentar los ejemplos y ejercicios del tema, debe crear una carpeta con el nombre "tema1" y guardar en la misma todos los archivos HTML creados.

3. Calcule el valor de la siguiente expresión, para valores de "x" iguales a 2.3, 4,5 y 7.2.

$$\sin^{-1}\left(\frac{\sqrt{x + 2x^{0.3} + 1.2}}{2x + 0.5}\right)$$

4. Calcule el valor de la siguiente expresión, para los siguientes valores de "x" y "y": x=9.2, y=5.6; x=2.3, y=8.4; x=7.1, y=6.3.

$$\left(\sin(x) - \frac{\sec(y)}{\sinh(x)}\right)^{0.34}$$

5. Determine si los siguientes números son positivos, negativos o cero: 23, 0, 12.2, -5.42, 12, -20.3.

6. Muestre el equivalente en días de la semana (siendo el primer día "domingo") de los siguientes números: 2, 5, 3, 4, 7, 1, 6.

7. Encuentre la sumatoria de los primeros 20, 40 y 55 números pares.

8. Para cada uno de los siguientes números muestre el primer cociente que no es divisible entre 2: 66, 128, 1832, 3830, 1234, 227, 2680.

# 2. VALIDACIÓN

En el anterior tema se ha aplicado el primer y segundo principio de la programación estructurada: el primero a través de las funciones (módulos) y el segundo con las estructuras de control (estructuras estándar).

En los módulos creados hasta ahora se ha asumido que los datos proporcionados son siempre del tipo y valor correctos. Así si el módulo recibe un entero positivo, se asume que siempre se le manda un dato de tipo entero y con un valor positivo.

En la práctica, no obstante, el realizar dicha asunción constituye un error, pues con seguridad, en algún momento durante su uso, el módulo recibirá un dato del tipo y/o valor incorrectos. Así al módulo que recibe un número entero positivo, en algún momento se le mandará un número real, un texto u otro tipo de dato (tipo de dato incorrecto) o si se le manda el tipo correcto (un número entero) en algún momento el mismo será negativo (valor incorrecto).

La probabilidad de que tales errores ocurran incrementa exponencialmente si los datos son introducidos directamente por los usuarios finales (no programadores). Es un hecho comprobado que si a un usuario final se le pide que haga algo, con seguridad no lo hará, así si se le pide que introduzca sólo números enteros y positivos, con seguridad introducirá números negativos, reales, texto y otro tipo de datos y si se le pide que no haga algo, con seguridad lo hará, así si se le pide que no introduzca texto, seguro que introducirá texto.

Por eso, en toda aplicación real se debe aplicar el tercer principio de la programación estructurada (además de los dos primeros) y más aún, si dichos datos son introducidos directamente por los usuarios finales.

Al proceso de controlar y/o evitar que se introduzcan o empleen datos del tipo y valores incorrectos se conoce como validación. Normalmente la validación se la hace en dos puntos: a) en los módulos, donde antes de realizar las operaciones y/o acciones propias del módulo se verifica que los datos recibidos sean del tipo y valores correctos y b) en la interfaz del usuario, donde se hace lo posible para evitar que el usuario pueda introducir datos del tipo y valor incorrectos.

A partir de este tema se aplicará también el tercer principio de la programación estructurada en la resolución de los problemas.

## 2.1. DIAGRAMAS DE ACTIVIDADES

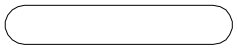
Para expresar gráficamente un algoritmo se emplearán los diagramas los *diagramas de actividades.* Como se sabe, los algoritmos describen la secuencia lógica de acciones que deben llevarse a cabo para resolver un problema.

Los *diagramas de actividades* constituyen una de las varias clases de diagramas que forman parte de *UML*, el *Lenguaje de Modelado Unificado*, que actualmente es el lenguaje estándar para el modelado de sistemas de software. Se ha elegido este tipo de diagramas no sólo por ser un estándar, sino porque permiten expresar los algoritmos de manera clara, ordenada, sencilla y porque además son muy versátiles, permitiendo representar diferentes tipos de estructuras.

El inicio de un diagrama de actividades se representa con un círculo relleno:

El final del diagrama de actividades se representa con un círculo que contiene un círculo relleno en su interior:

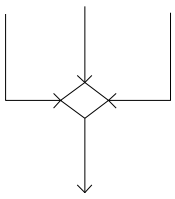
Una acción (o sentencia) se representa con un rectángulo con sus bordes redondeados:



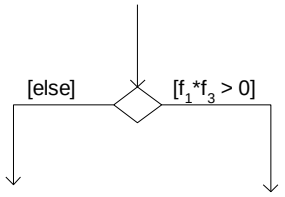
Un flujo, que es el símbolo empleado para unir los elementos del diagrama y señalar la dirección en que se deben seguir las acciones, se representa por una flecha continua abierta:

Una unión o bifurcación se representa por un pequeño rombo:

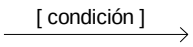
Cuando se emplea como unión llegan dos o más flujos y sale uno.



Cuando se emplea como bifurcación ingresan uno o más flujos y salen dos o más flujos acompañados de alguna condición.




Una condición, tal como se puede observar en la anterior figura, se representa como texto encerrado entre corchetes y siempre está relacionado a algún flujo:




El texto de la condición puede ser una expresión matemática, una expresión relacional, una condición textual, etc. Para la condición por defecto se puede emplear *[else]* (caso contrario).

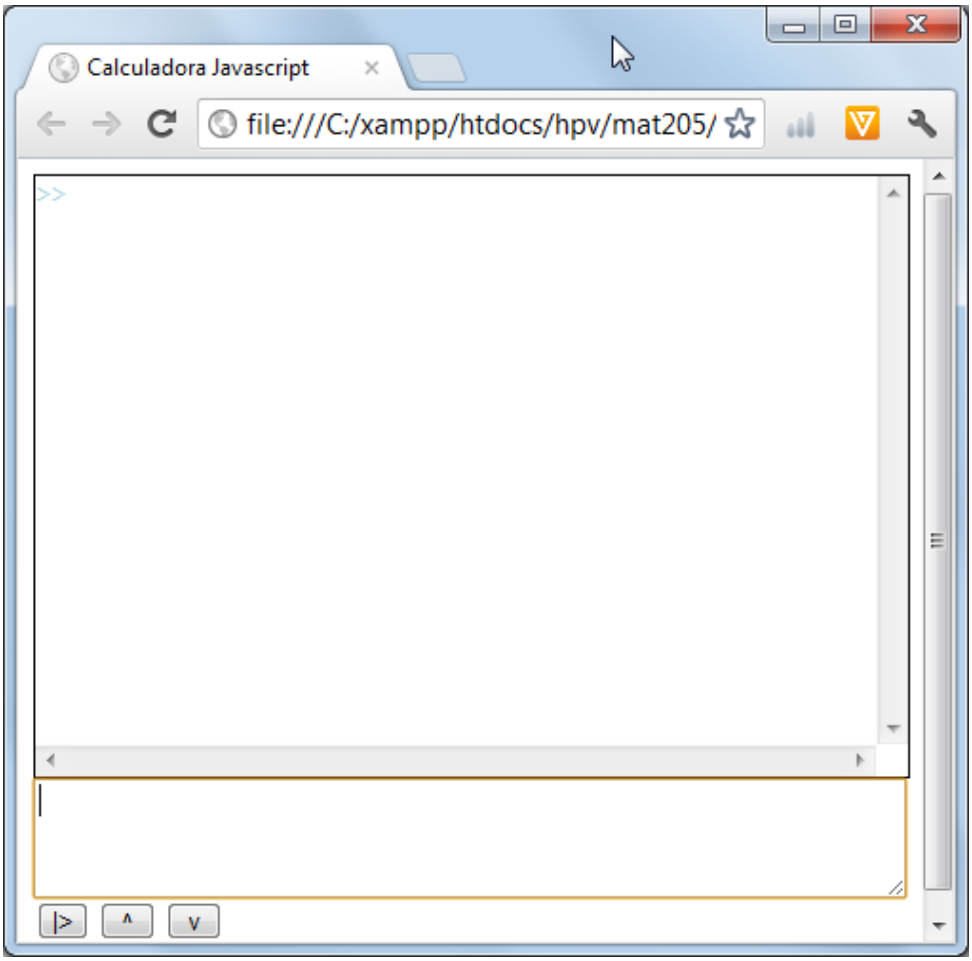
Una actividad representa un conjunto de acciones (o un subprograma) que se detalla luego por separado empleando otro diagrama de actividades. Se representa como una acción con un icono de una acción llamando a otra en la parte inferior derecha:

Las notas se emplean para comentar y documentar los diagramas de actividades. Se representan como una hoja con una esquina doblada y asociada, mediante una línea discontinua, a cualquiera de los elementos de un diagrama de actividades:

Al elaborar un diagrama de actividades se debe tener presente que es un lenguaje y como tal es necesario respetar su sintaxis (es decir los símbolos) pues cada uno de ellos tiene su propio significado y si se cambian, cambia también la lógica del algoritmo. Por ejemplo, para representar un flujo siempre se debe emplear la flecha continua abierta, no una flecha cerrada y rellena: ——►, una flecha cerrada no rellena: ——▷, o una flecha discontinua abierta: ┈┈┈> pues cada una de ellas tienen un significado muy diferente (mensaje, herencia y dependencia respectivamente).

## 2.2. LA CALCULADORA JAVASCRIPT

Con el propósito de facilitar algunos cálculos rápidos, así como probar de manera inmediata operadores, instrucciones y otras características del lenguaje Javascript, se ha creado una página: "calculadora.html" (suministrada con el material del tema):



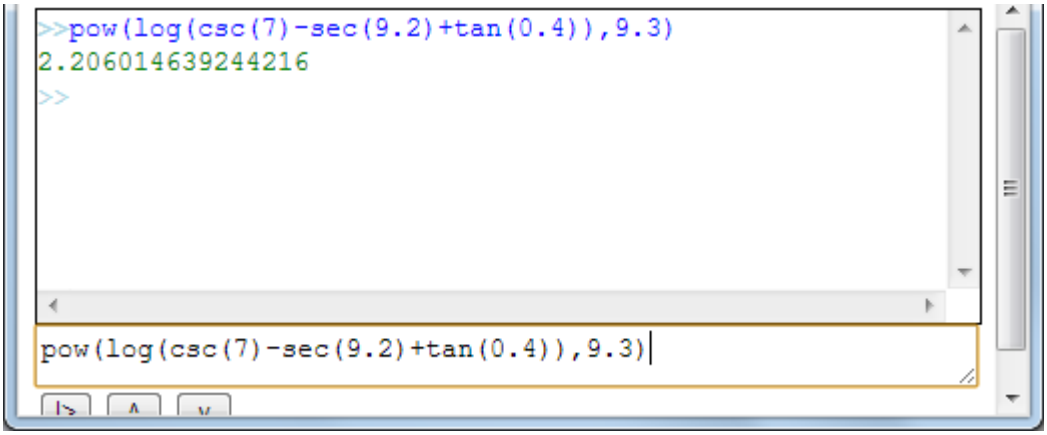

Esta página hace uso de la librería "sis101.js" y la librería "diagram.js" (Javascript Diagram Builder: 3.3: www.lutanho.net/diagram/), modificada para adaptarla a los requerimientos de la materia (ambas librerías se proporcionan conjuntamente el material del tema).

Como toda página, para utilizarla simplemente se abre en un navegador (verificando que tanto "mat205.js" como "diagram.js" se encuentren en el mismo directorio que "calculadora.html").

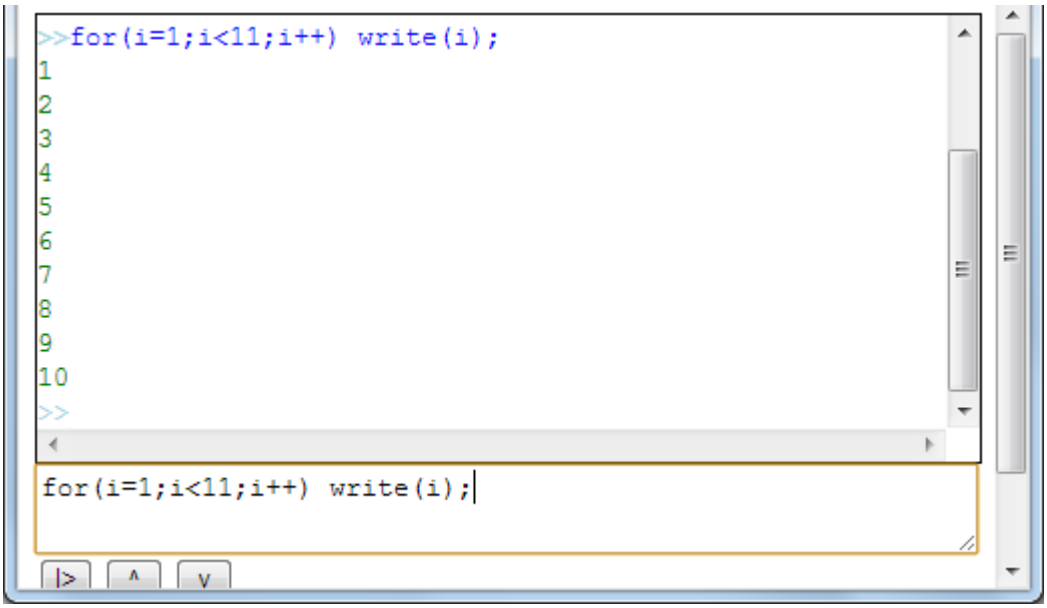
Para ejecutar una instrucción, simplemente se la escribe en el recuadro inferior y se pulsa la tecla "Enter" (o se hace click en el botón [|>]). Por ejemplo, para calcular el valor de la siguiente expresión:

$$\ln(\csc(7) - \sec(9.2) + \tan(0.4))^{9.3}$$

Se escribe la expresión en el recuadro inferior y al pulsar "Enter" se obtiene:




En la calculadora se pueden ejecutar prácticamente todas las instrucciones Javascript, así como las de la librería "mat205.js". Por ejemplo, se puede crear un ciclo "for" con contador que vaya del 1 al 10 y que en cada repetición del ciclo muestre el valor del contador:




La función "write" es una función propia de la calculadora (no de Javascript) y como se puede ver, sirve para mostrar en el recuadro de resultados la expresión que se escribe en ella.

La calculadora almacena las instrucciones a medida que se ejecutan y pueden ser recuperadas (para ser modificadas o vueltas a ejecutar) con las teclas de cursores, o con los botones [^] y [v]. En los celulares con sistema Android, el teclado **Multiling** cuenta con dichas teclas, mismas

que funcionan correctamente en **UCBrowser** y **Firefox**.

En la calculadora se pueden crear funciones empleando la forma literal (no la forma estándar). Por ejemplo, en las siguientes instrucciones se crea una función para calcular el cubo de un número y luego se la emplea para calcular el cubo de 3 y de 7.89:

```
>>cubo = function(x){return x*x*x;}
function()
>>cubo(3)
27
>>cubo(7.89)
491.1690689999999
```

Sin embargo, ***las funciones creadas en la calculadora sólo existen mientras la página está abierta en el navegador*** y desaparecen tan pronto como se cierra. Por esa razón, en la calculadora sólo se deben crear funciones que no vayan a ser reutilizadas y/o que sean muy fáciles de programar.

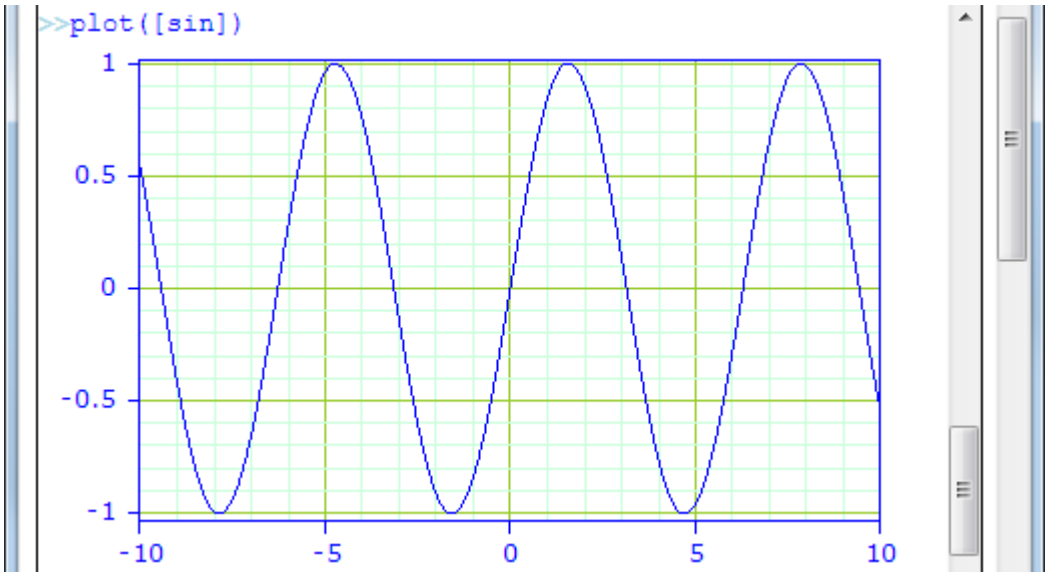
En las computadoras es posible insertar saltos de línea al escribir una función (u otra instrucción) con las teclas "Shift+Enter", no obstante, si la función es compleja, no es recomendable resolverla en la calculadora, sino más bien en una página HTML.

Por otra parte, a la librería "diagram.js" se le ha añadido la función "plot" (la misma que puede ser empleada también desde la calculadora):

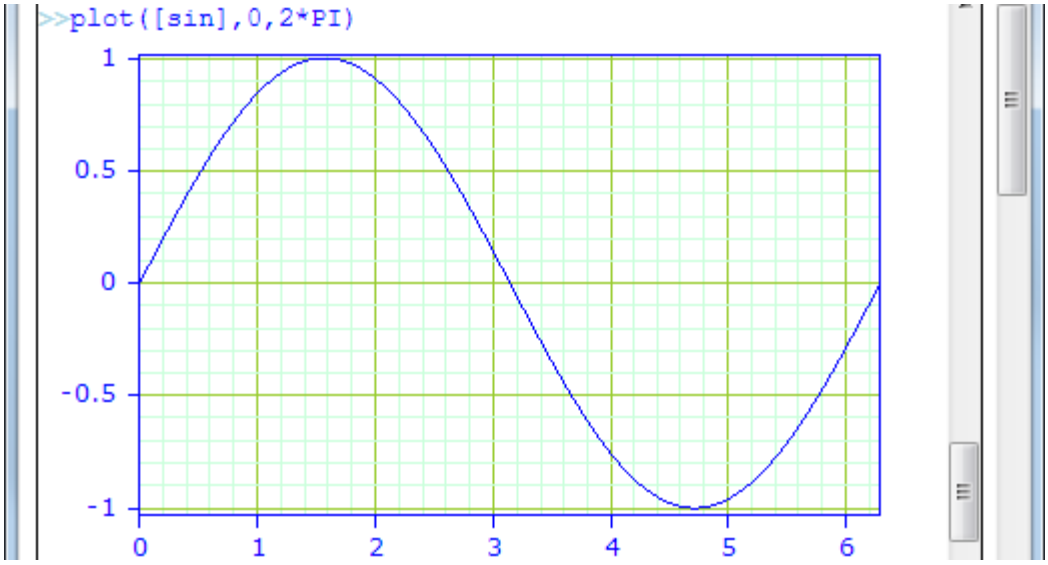
```
plot([lista_de_funciones],xi,xf,w,h,yi,yf);
```

Donde "lista_de_funciones" son los nombres de las funciones a graficar (separadas con comas), "xi" y "xf" son los límites inicial y final de la variable independiente (valores por defecto -10 y 10); "w" y "h" son el ancho y alto de la gráfica en pixeles (valores por defecto 420 y 240); "yi" y "yf" son los valores inicial y final de la variable dependiente (los valores por defecto se calculan automáticamente).

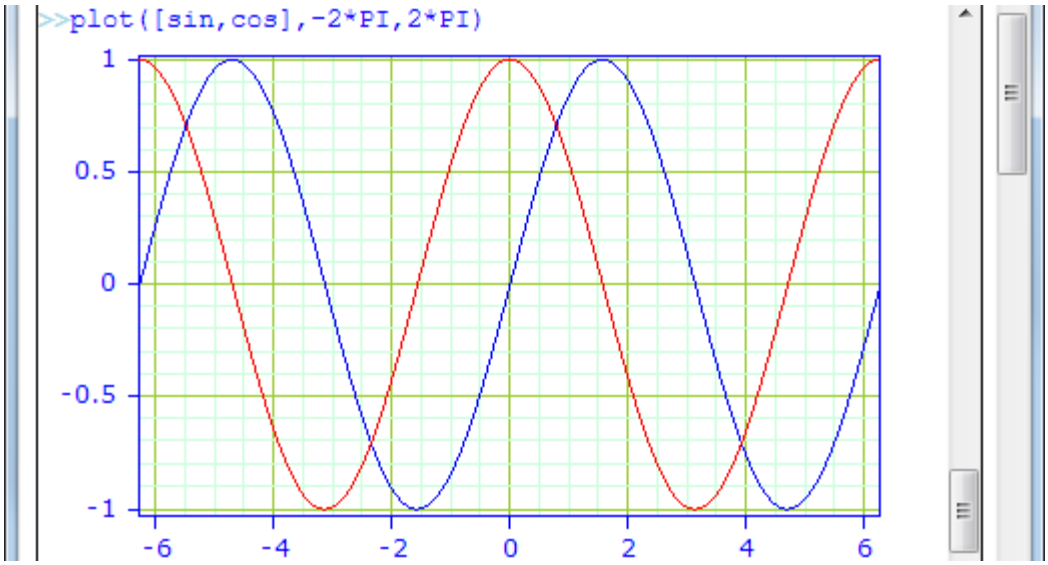
Por lo tanto sólo es imprescindible el nombre, o nombres, de las funciones, así para graficar la función "sin" se puede escribir:




Que es la gráfica de la función seno entre -10 y 10. En la práctica casi siempre se dan los límites de la variable independiente, así, con la siguiente instrucción se grafica la función seno entre 0 y 2π:




Y con la siguiente se grafican las funciones seno y coseno entre -2π y 2π.




Como se dijo, la utilidad principal de la calculadora es la de permitir realizar algunos cálculos rápidos, pero sobre todo probar, interactivamente, las estructuras, características y funciones del lenguaje y/o de las librerías.

Para probar las funciones de otra u otras librerías (aparte de "mat205.js" y "diagrama.js") simplemente se edita la página "calculadora.html" y se le añade las etiquetas "script" para importar las librerías correspondientes. Por supuesto, si no se requiere las funciones de las librerías "mat205.js" y/o "diagram.js", simplemente no se las incluye.

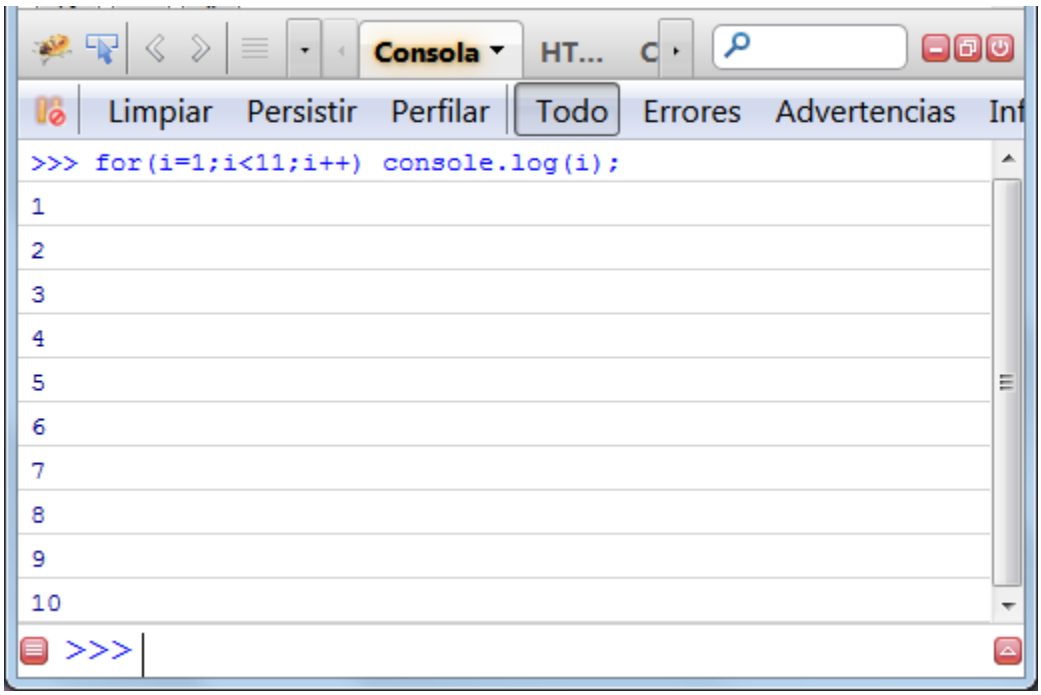
## 2.3. LAS CONSOLAS DE LOS NAVEGADORES

Actualmente, debido a la creciente importancia de Javascript en el desarrollo de aplicaciones, los navegadores modernos cuentan con consolas para ayudar a desarrollar aplicaciones. Dichas consolas (entre otras cosas) permiten ejecutar instrucciones Javascript, de manera similar a como lo hace la calculadora.

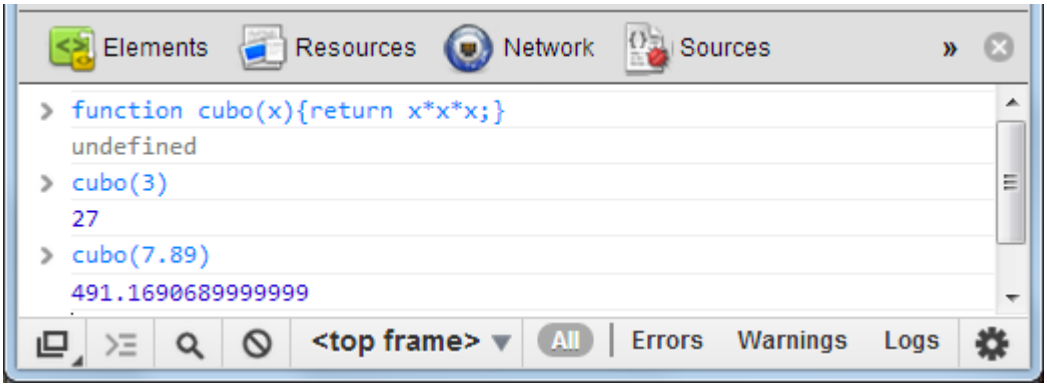
Para emplear la consola, simplemente se abre la página donde se encuentra el código Javascript y/o donde se han importado (con "script") las librerías Javascript. Entonces se accede a la consola pulsando la tecla F12 (si, en la ventana que aparece, no está por defecto la consola, se la abre haciendo clic en el menú que aparece).

Al igual que sucede con la calculadora, en las consolas se pueden probar prácticamente todas las características del lenguaje y/o librerías. Es posible también imprimir valores pero empleando la instrucción "console.log" en lugar de "write".

Por ejemplo la instrucción que imprime los números del 1 al 10, escrito en la consola de Mozilla Firefox, es:




También se pueden crear funciones, pero en las consolas es posible crearlas en la forma estándar, así, la función "cubo" (creada como ejemplo en la calculadora) se crea y utiliza en Google Chrome de la siguiente forma:




Como en la calculadora, las funciones creadas en las consolas sólo existen mientras la página está abierta y dejan de existir tan pronto se cierra.

La mayoría de los navegadores, proporcionan, además de la consola, otras herramientas, tales como autocompletado, depuración, análisis de la estructura HTML, etc. Lamentablemente, no todos los navegadores proporcionan las mismas herramientas y no todos los navegadores poseen dichas herramientas.

Las herramientas más avanzadas las proporcionan los navegadores Google Chrome y Mozilla Firefox. En este último navegador, para acceder a la consola y las herramientas de depuración, se debe instalar el complemento "Firebug" (disponible en: http://getfirebug.com/), siendo conveniente instalar también "Acebug" (disponible en: https://addons.mozilla.org/en-US/firefox/addon/acebug/).

Si se dispone de conexión a Internet la instalación de estos complementos es automática, de no ser así, se deben bajar los complementos, guardarlos en un directorio e instalarlos de la siguiente forma: se abre el menú principal del navegador (botón del extremo superior izquierdo del navegador):


En el menú que aparece se elige la opción complementos. Dentro de complementos se hace clic en el botón:

Se elige la opción "Instalar complemento desde archivo...", se busca el archivo en el lugar donde fue guardado y se abre. De esa manera el complemento queda instalado.

## 2.4. EJERCICIOS

Los siguientes ejercicios deben ser resueltos en la calculadora Javascript (o en una de las consolas) y presentados en la misma, por lo que debe copiar las instrucciones que empleó para resolverlos en cualquier editor de texto y volverlos a copiar a la calculadora, uno por uno, al momento de la presentación.

1. Empleando la estructura "for" muestre los números del 20 al 1 (en orden descendente).
2. Empleando la estructura "while" muestre los cocientes resultantes de dividir el número 123456789 entre 10.
3. Grafique las funciones seno y coseno entre 0 y 4π.

## 2.5. LA FUNCIÓN MAP

Cuando se trabaja con series de más de 2 datos puede resultar de utilidad la función "map" (implementada en la librería "mat205.js"):

```
map(función, vector_1, vector_2, vector_3,…)
```

Donde vector_1, vector_2, etc., son los datos de los parámetros 1, 2, etc., de la "función". Map evalúa la "función" para cada uno de los elementos de los vectores devolviendo los resultados igualmente en un vector.

Por ejemplo, para calcular los valores del seno para: 0.2, 0.8, 1.2, 1.4, 1.8, 2.1, 2.5, 3.4, 3.6, 4.2, en lugar de evaluar 10 veces la función "sin", cambiando su valor en cada ejecución, se pueden calcular todos los resultados a la vez (como se puede probar en la calculadora):

```
>>map(sin,[0.2,0.8,1.2,1.4,1.8,2.1,2.5,3.4,3.6,4.2])
[0.19866933079506122, 0.7173560908995228,
0.9320390859672263, 0.9854497299884601,
0.9738476308781951, 0.8632093666488737,
0.5984721441039565, -0.2555411020268312,
-0.44252044329485246, -0.8715757724135882]
```

## 2.6. EJERCICIO

4. Calcule (empleando la calculadora Javascript) la raíz cuadrada de: 1, 2, 2.5, 3, 3.5, 4, 5, 6, 7.1, 8, 10, 12.1, 12.3, 14.5, 16.6, 17, 18, 18.5 y 19.

## 2.7. LA INTERFAZ DE USUARIO

La interfaz de usuario es la apariencia visual que tiene el programa y es la forma en la que el usuario final percibe el programa (o aplicación). Para el usuario esa interfaz "es" el programa o aplicación. Por esa razón, en la resolución de problemas reales, es importante diseñar interfaces de usuario que sean fáciles de utilizar (intuitivas) y que además sean agradables a la vista.

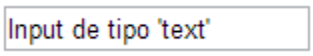
HTML cuenta con los elementos necesarios para crear interfaces de usuario funcionales y para lograr una apariencia más atractiva se tiene a disposición diferentes técnicas, siendo la más empleada (y la recomendada) las hojas de estilo en cascada (Cascading Style Sheets: CSS).

En la asignatura se empezará creando interfaces de usuario simples, empleando unos cuantos elementos (sin adornos) y luego, a medida que se vaya adquiriendo experiencia en la creación de interfaces, se irán añadiendo otros elementos, adornos y librerías.

Para introducir (o mostrar) información en forma de texto se emplea la etiqueta <input> con el atributo "type" igual a "text", como se muestra en el siguiente ejemplo:

```
<input type="text" value="Input de tipo 'text'">
```

El atributo "value" establece (o recupera) el texto que aparece en el elemento, así la anterior instrucción genera un cuadro con la siguiente apariencia y contenido:

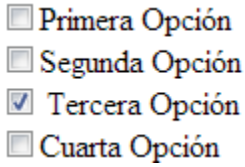



Para permitir que el usuario seleccione una o más opciones, de un conjunto de posibles opciones, se emplea la etiqueta <input> con el atributo "type" igual a "checkbox", como en el siguiente ejemplo:

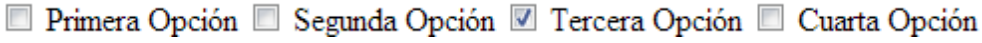
```
<input type="checkbox" name="cb1" value="opcion1">
  Primera Opción<br>
<input type="checkbox" name="cb1" value="opcion2">
  Segunda Opción<br>
<input type="checkbox" name="cb1" value="opcion3" checked>
  Tercera Opción<br>
<input type="checkbox" name="cb1" value="opcion4">
  Cuarta Opción<br>
```

Cuando se crean grupos de opciones (como ocurre en este caso) se los agrupa dando a todos los elementos el mismo nombre ("name"), en este

ejemplo el nombre asignado es "cb1". El atributo "value", establece y devuelve el valor asociado a la opción. El atributo "checked" (empleado en la tercera opción) hace que la opción aparezca seleccionada, tal como se puede ver en la salida que generan estas instrucciones:




La etiqueta "<br>", empleada en el ejemplo después de cada <input>, sirve para insertar saltos de línea en la página HTML. Sin este etiqueta la salida generada es:

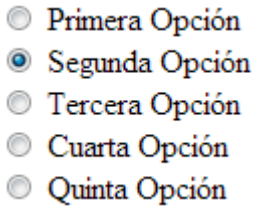



Al igual que <input>, <br>, no tiene etiqueta de cierre.

Cuando el usuario debe seleccionar solo una opción, de un grupo de posibles opciones, se emplea la etiqueta <input> con el atributo "type" igual a "radio":

```
<input type="radio" name="r1" value="opcion1">
  Primera Opción<br>
<input type="radio" name="r1" value="opcion2" checked>
  Segunda Opción<br>
<input type="radio" name="r1" value="opcion3">
  Tercera Opción<br>
<input type="radio" name="r1" value="opcion4">
  Cuarta Opción<br>
<input type="radio" name="r1" value="opcion5">
  Quinta Opción<br>
```

Que produce la siguiente salida:

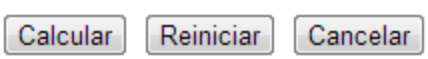



Como se puede ver el tipo "radio" es muy similar al tipo "checkbox", excepto que con "radio" sólo se puede elegir una de las opciones.

Para que el usuario instruya al programa realizar una acción o tarea, se puede emplear el tipo <input> con el atributo "type" igual a "button":

```
<input type="button" value="Calcular">
<input type="button" value="Reiniciar">
<input type="button" value="Cancelar">
```

Que produce la siguiente salida:




Alternativamente se puede emplear la etiqueta <button>:

```
<button type="button">Calcular</button>
<button type="button">Reiniciar</button>
```

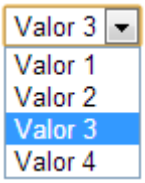
```
<button type="button">Cancelar</button>
```

Que genera la misma salida. Al contar <button> con etiqueta de cierre, el texto del botón se escribe entre las etiquetas de apertura y cierre, pudiendo emplearse también una imagen en lugar de texto.

Para permitir que el usuario seleccione un valor (o más de un valor) de una lista de valores posibles, se emplea la etiqueta <select>, como en el siguiente ejemplo:

```
<select>
  <option value="uno">Valor 1</option>
  <option value="dos">Valor 2</option>
  <option value="tres" selected>Valor 3</option>
  <option value="cuatro">Valor 4</option>
</select>
```

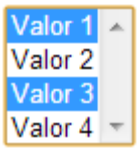
Que genera la siguiente salida (cuando se abren las opciones):




El atributo "selected", empleado en la tercera opción, hace que dicha opción aparezca seleccionada por defecto. Para que sea posible seleccionar más de una opción se debe emplear el atributo "multiple" en la etiqueta <select>:

```
<select multiple>
  <option value="uno" selected>Valor 1</option>
  <option value="dos">Valor 2</option>
  <option value="tres" selected>Valor 3</option>
  <option value="cuatro">Valor 4</option>
</select>
```

Que genera la siguiente salida:




Para seleccionar más de un valor (en Windows) se debe mantener presionada la tecla "Ctrl" (en Android, aparece una ventana donde se pueden seleccionar los valores).

En HTML5, la etiqueta <input> cuenta con otros tipos ("type") que facilitan la introducción y/o selección de información. Lamentablemente no todos los navegadores soportan aún todos estos tipos (sobre todo en los dispositivos móviles). Por esta razón no se emplearán, por el momento, en la creación de interfaces.

A más de los atributos antes descritos, todos los elementos cuentan con el atributo "id", con el que se le asigna un nombre único (identificador) al elemento. Es mediante este identificador que se pueden leer y modificar, desde Javascript, los atributos y contenido de los elementos. A diferencia de "name", no pueden existir "id's" repetidos, es decir que no se le puede asignar un mismo "id" a dos o más elementos.

## 2.8. EVENTOS

Los elementos de una página HTML pueden responder a eventos. Un evento es un suceso generado por el usuario o por el sistema, así por ejemplo, el hacer clic en un elemento constituye un objeto, mover el mouse constituye otro evento, pulsar una tecla otro, etc.

En respuesta a los eventos se ejecuta código Javascript (casi siempre en forma de una función).

Existen varias formas en las que se puede asociar código al o a los eventos de un elemento (un elemento puede disparar dos o más eventos). La más simple consiste en escribirla como un atributo del elemento, en la forma:

```
<nombre_del_elemento nombre_del_evento="código y/o función Javascript">
```

Algunos de los eventos más frecuentes ("nombre_del_evento") son:

| Evento | Ocurre cuando: |
|---|---|
| onclick | Se hace click con el mouse. |
| ondblclick | Se hace doble click con el mouse. |
| onmouseover | El puntero se mueve sobre el elemento. |
| onmouseout | El puntero se mueve fuera del elemento. |
| onmousemove | El puntero se mueve en el elemento. |
| onkeydown | Se está presionando una tecla. |
| onkeyup | Se deja de presionar una tecla. |
| onkeypress | Se pulsa (se presiona y suelta) una tecla |
| onblur | El elemento pierde el foco (ya no es el elemento activo). |
| onfocus | El elemento gana el foco (es el elemento activo). |
| onchange | El valor del elemento ha sido cambiado. |
| onselect | Se selecciona un texto en el elemento. |
| onload | La página ha terminado de ser cargada. |

## 2.9. EJEMPLOS

**1. Cree una página HTML en la que dado un número comprendido entre 1 y 12 devuelva el mes equivalente.**

La lógica del módulo (función) que resuelve el problema es directa: si el número es 1, la función devuelve "enero", si es 2 devuelve "febrero" y así sucesivamente.

En la interfaz de usuario se empleará un <select> para seleccionar el número del mes. De esa manera se evita un posible error al impedir que el usuario introduzca datos diferentes al conjunto de valores proporcionados (los números del 1 al 12).

El nombre del mes se mostrará en un <input> de tipo "text", con el atributo readonly (lectura solamente). Esto porque en el <input> solo se mostrará el nombre del mes (sin permitir que se escriba nada).

El código Javascript que encuentra el mes equivalente y muestra el nombre en el elemento <input>, se activará (se disparará) cuando el usuario elije un nuevo número de mes (cuando ocurre el evento "onchange").

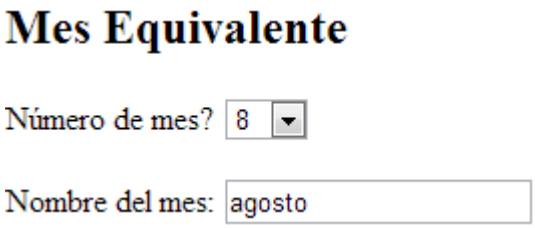
La página HTML creada para resolver el problema es:

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale:1">
```

```
    <title>Ejemplo 1</title>
    <script src="sis101.js"></script>
    <script>
      //Devuelve el nombre del mes equivalente al número "n"
      function nombreMes(n){
        if (isNaN(n)) throw new Error(
          "nombreMes: El argumento debe ser un número entero.");
        if (frac(n)!=0) throw new Error(
          "nombreMes: El número debe ser entero.");
        if (n<1 || n>12) throw new Error(
          "nombreMes: El número debe estar comprendido entre 1 y 12");
        switch (n){
          case 1: return "enero";
          case 2: return "febrero";
          case 3: return "marzo";
          case 4: return "abril";
          case 5: return "mayo";
          case 6: return "junio";
          case 7: return "julio";
          case 8: return "agosto";
          case 9: return "septiembre";
          case 10: return "octubre";
          case 11: return "noviembre";
          default: return "diciembre";
        }
      }
      //Muestra el mes equivalente al número elegido "n"
      function mostrarMes(n){
        document.getElementById("nom_mes").value=nombreMes(n);
      }
    </script>
  </head>
  <body>
    <h2>Mes Equivalente</h2>
    <p>Número de mes?
      <select id="num_mes" onchange="mostrarMes(this.selectedIndex+1);">
        <option value="1" selected>1</option>
        <option value="2">2</option>
        <option value="3">3</option>
        <option value="4">4</option>
        <option value="5">5</option>
        <option value="6">6</option>
        <option value="7">7</option>
        <option value="8">8</option>
        <option value="9">9</option>
        <option value="10">10</option>
        <option value="11">11</option>
        <option value="12">12</option>
      </select>
```

```
    </p>
    <p>Nombre del mes:
      <input id="nom_mes" type="text" value="enero" readonly>
    </p>
  </body>
  </html>
```

Que al ser abierto en un navegador y seleccionar el mes "8" muestra:




En esta solución hay algunas cosas nuevas que se deben aclarar. En primer lugar, antes de la etiqueta <html>, se ha empleado una nueva etiqueta <!DOCTYPE html>. Esta etiqueta le informa al navegador que el contenido debe ser formateado de acuerdo a lo establecido en la especificación HTML5, que es la especificación actual para el lenguaje HTML (aunque todavía no es un estándar).

Se ha empleado también una nueva etiqueta <meta>, con el atributo "name" igual a "viewport". Esta etiqueta y atributo, le instruyen al navegador que en caso de tratarse de un dispositivo móvil, emplee el conjunto de pares "atributo=valor" que se le dan en "content". En este ejemplo se le instruye que el ancho del navegador sea el ancho del dispositivo (width=device-width) y que la escala inicial sea 1 (initial-scale=1) es decir que inicialmente la página esté en el tamaño normal (sin ampliación).

En cuanto al código Javascript, se ha dividido el problema en dos módulos (primer principio de la programación estructurada), pues en realidad se tienen dos problemas: a) convertir un número comprendido entre 1 y 12 en el nombre del mes equivalente (nombreMes) y b) Leer el número seleccionado del elemento <select> y mostrar el nombre del mes equivalente en el elemento <input> (mostrarMes).

En cuanto al segundo principio, en "nombreMes" se han empleado las estructuras "if" y "switch", mientras que en "mostrarMes" sólo se requiere una instrucción.

En lo referente al tercer principio, la validación ha comenzado con el uso del elemento <select>, con el cual se impide que el usuario introduzca datos erróneos.

Como por definición un módulo debe ser independiente, es decir que debe funcionar igual sin importar de donde o quien le envíe los datos, se ha hecho la validación respectiva en el módulo "nombreMes", comprobando primero que el dato sea realmente un número, empleando para ello la función "isNaN" que devuelve verdadero (true) si el dato que se le pasa no es un número y falso en caso contrario. Si el dato es un número se verifica que no sea un número entero, es decir que no tenga parte fraccionaria (empleando para ello la función "frac"), finalmente, se verifica que el número entero esté comprendido entre 1 y 12.

En caso de no cumplirse alguna de las condiciones de validación, la función genera un error (con "throw new Error(…)") y no devuelve ningún

resultado (posteriormente se estudiará cómo capturar y tratar estos errores).

En el módulo "mostrarMes", se recibe el número seleccionado (n), se transforma ese número en el nombre del mes (con la función "nombreMes") y se asigna el resultado al atributo "value" del elemento <input>. Para trabajar con el elemento <input>, se busca el mismo en el documento ("document"), con el método "getElementById" (empleando el "id" asignado al elemento: "nom_mes"). El método "getElementById" devuelve el objeto equivalente al elemento y en el mismo se pueden modificar y/o leer tanto sus atributos como su contenido.

En este ejemplo en particular, no se han empleado los valores asignados a la propiedad "value" de las opciones (<option>), esto porque el índice de la opción seleccionada ("selectedIndex") es dicho valor menos 1 (tenga presente que en Javascript el primer índice es siempre 0). Por lo tanto para obtener el número de mes, lo más sencillo en este caso es sumar 1 a "selectedIndex", tal como se ha hecho al definir el evento "onchange":

```
<select id="num_mes" onchange="mostrarMes(this.selectedIndex+1);">
```

En esta instrucción la palabra "this" hace referencia al elemento donde se está generando el evento, es decir "this", en este caso, es el elemento <select>. En otras palabra, en este ejemplo, "this" es equivalente al objeto que se obtendría con la instrucción "document.getElementById ('nun_mes')".

Si no existiera la relación entre el índice seleccionado (selectedIndex) y el número del mes, habría sido necesario escribir la siguiente instrucción:

```
<select id="num_mes" onchange=
  "mostrarMes(parseInt(this.options[this.selectedIndex].value));">
```

Donde la función "parseInt" se emplea para convertir el texto en número entero, "options" es un vector con las opciones (<options>) existentes en <select> y "value" es el valor de la opción seleccionada (en forma de texto).

**2. Elabore una página HTML que dado un número, calcule, en función a la o las opciones elegidas, el logaritmo natural, el logaritmo en base 2, el logaritmo en base 5 y/o el logaritmo en base 10 dicho número.**

Resolver el problema en si es muy sencillo: simplemente se debe calcular el resultado con las funciones logarítmicas ya existentes.

El problema radica en el diseño y funcionalidad de la interfaz del usuario. Como el número puede ser entero o real y existe un número infinito de posibles números reales, no se puede emplear <select> para evitar que el usuario escriba datos erróneos, por esta razón, en este caso no queda otra opción que emplear el elemento <input> con el tipo "text".

Por otra parte, para permitir elegir la o las funciones a calcular se emplearán elementos <input> del tipo "checkbox" y los resultados serán mostrados en elementos de tipo <input> pero con el atributo "readonly" (de manera que no puedan ser modificados).

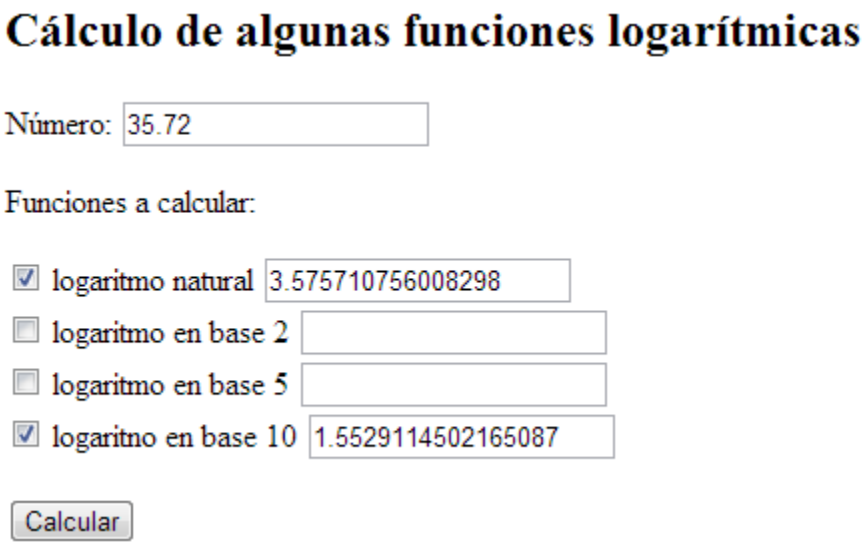
Finalmente, para que los cálculos se lleven a cabo, se empleará el evento "onclick" de un botón. Dicho evento llamará a la función que se encargará de leer los datos, realizar los cálculos y mostrar los resultados obtenidos.

La página HTML creada para resolver el problema es:

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ejemplo 2</title>
  <script src="sis101.js"></script>
  <script>
    function calcular(){
      var num = document.getElementById("num");
      var n = parseFloat(num.value);
      if(isNaN(n)){
        alert("El dato introducido debe ser un número");
        num.focus();
        throw new Error("Dato incorrecto");
      }
      var funs = document.getElementsByName("fun");
      var res = document.getElementsByName("res");
      if (funs[0].checked) {
        res[0].value=log(n);
      } else
        res[0].value="";
      if (funs[1].checked) {
        res[1].value=log2(n);
      } else
        res[1].value="";
      if (funs[2].checked) {
        res[2].value=logb(n,5);
      } else
        res[2].value="";
      if (funs[3].checked) {
        res[3].value=log10(n);
      } else
        res[3].value="";
    }
  </script>
</head>
<body>
  <h2>Cálculo de algunas funciones logarítmicas</h2>
  <p>Número: <input id="num" type="text" value="0"></p>
  <p>Funciones a calcular:</p>
  <input name="fun" type="checkbox" checked>
    logaritmo natural
    <input name="res" type="text" readonly><br>
  <input name="fun" type="checkbox">
    logaritmo en base 2
    <input name="res" type="text" readonly><br>
  <input name="fun" type="checkbox" value="log5">
    logaritmo en base 5
    <input name="res" type="text" readonly><br>
```

```
    <input name="fun" type="checkbox" value="log10">
      logaritno en base 10
      <input name="res" type="text" readonly><br>
    <p>
      <input type="button" value="Calcular" onclick="calcular()">
    </p>
  </body>
  </html>
```

Abriendo la página, introduciendo algún dato y seleccionado algunas funciones se obtiene:




En este ejemplo existen también algunas cosas que aclarar. El elemento que contiene el número se obtiene, como en el anterior ejemplo, con "document.getElementById" y su valor (num.value) se convierte en un número real con la función "parseFloat".

Se verifica (se valida) que el número escrito sea realmente un número (con isNaN) y de no ser así, se muestra un mensaje, se pasa el foco (el control) al elemento <input> (para que se corrija el error) y se genera un error (con "new throw Error()") para que la ejecución de la función se detenga (como ya se dijo, posteriormente se verá como atrapar y tratar estos errores).

Los elementos que contienen las opciones elegidas, se obtienen en forma de un vector con "document.getElementsByName('fun')", se procede de igual forma con los elementos que muestran los resultados (se debe tener en mente siempre que en Javascript, al igual que C/C++, el primer índice de un vector es siempre 0).

Una vez obtenidos los checkbox (en forma de vectores) se analiza la propiedad "checked" de cada uno de ellos y si está seleccionada (si es verdadera), se hace el cálculo y se muestra el resultado respectivo, caso contrario se deja el resultado en blanco. Esto se logra con cuatro estructuras "if-else".

El proceso puede ser simplificado empleando una estructura "for" y una estructura "if-else" (dentro de la estructura "for"), sin embargo ello implica el uso de otras matrices y/o la función "eval", razón por la cual por el momento se deja la solución tal como se la ha presentado.

## 2.10. EJERCICIOS

Para presentar los ejemplos y ejercicios del tema, debe crear una carpeta con el nombre "tema2" y guardar en la misma todos los archivos HTML creados.

5. Cree una página HTML que dado un día de la semana (por ejemplo "miércoles") devuelva el número equivalente respectivo (siendo 1 el equivalente al día "domingo").
6. Cree una página HTML que dado un número entero comprendido entre 1 y 15, muestre su equivalente en números romanos.
7. Cree una página HTML que dado un número, calcule, en función a la o las opciones elegidas, la raíz cuadrada, cúbica, quinta y/o séptima del mismo.
8. Cree una página HTML que dado un ángulo en grados, calcule, en función a la o las opciones elegidas, el seno, coseno, secante y cosecante del mismo.

# 3. RECURSIVIDAD

En este tema se estudia otra forma de resolver problemas repetitivos: la recursividad. En general, un concepto es recursivo cuando en su definición se emplea el concepto que se está definiendo. Desde el punto de vista de la programación, una función es recursiva cuando se llama a sí misma.

La recursividad no es imprescindible en la solución de problemas iterativos, sin embargo, algunos problemas pueden ser resueltos de manera más clara y sencilla empleando este método.

Puesto que una solución más clara y sencilla es también más fácil de comprender, el empleo de estos métodos puede facilitar el mantenimiento y actualización de los programas, que es justamente el propósito de la programación estructurada.

Es importante comprender que la recursividad se trata sobre todo de otra forma de pensar, otra forma de abordar los problemas repetitivos. Mientras que con las estructuras estándar se piensa en el número de veces que se debe repetir un proceso o la condición que debe cumplirse para que deje de repetirse, en la recursividad se piensa en la forma de encontrar la solución aprovechando una solución previa.

**Algo que es muy importante al momento de plantear una solución recursiva es no considerar como parte del problema el proceso recursivo en sí**, es decir que no se debe considerar como parte del problema el cómo la función se llama a sí misma. **En el planteamiento recursivo sólo interesa que efectivamente devuelva el resultado buscado, no como se llevan a cabo las diferentes llamadas recursivas**.

**En toda solución recursiva es importante que exista una condición de finalización** pues de lo contrario el planteamiento recursivo resultaría en un proceso infinito: la función se llamaría a sí misma indefinidamente o hasta que se consuma toda la memoria disponible.

Algo que también se debe tener en mente es que las soluciones recursivas son menos eficientes que las iterativas: consumen más memoria y requieren más tiempo. Esto porque cada vez que una función se llama a sí misma crea una copia de las variables y parámetros de la función, lo que consume memoria y requiere tiempo, pero además cuando devuelven el resultado (cuando terminan) se destruyen las variables y parámetros de la función, lo que también consume tiempo.

## 3.1. EL OBJETO "EVENT"

Hasta el momento las funciones encargadas de responder a los eventos, denominadas **"manejadores de eventos"**, han realizado su tarea sin recibir información con relación al evento al cual están respondiendo.

En muchos casos, sin embargo, los manejadores de eventos requieren información con relación al evento que están manejando, tal como el elemento en el que se ha producido, el tipo de evento, la tecla o botón del mouse que ha sido pulsada, el lugar de la pantalla donde se encontraba el puntero, etc. En Javascript, dicha información está disponible a través del objeto "event".

El objeto "event" es creado automáticamente por el navegador cada vez que se produce un evento y en los navegadores que siguen el estándar (que prácticamente son todos con excepción de Internet Explorer) debe ser el único parámetro que deben recibir los manejadores de eventos.

Algunas de las propiedades (datos) que están disponibles a través de este objeto son:

| Propiedad | Descripción |
|---|---|
| type | Nombre del evento (sin la palabra "on" por delante, por ejemplo "click"). |
| target | Elemento en el cual se ha producido el evento (el elemento que ha disparado el evento). |
| which | Código (número) de la tecla pulsada (eventos onkeydown y onkeypress). |
| keyCode | Código (número) de la tecla pulsada (evento onkeydown). |
| charCode | Código (número) de la tecla pulsada (evento onkeypress). |
| altKey | Verdadero si la tecla "Alt" está presionada. |
| ctrlKey | Verdadero si la tecla "Ctrl" está presionada. |
| shiftKey | Verdadero si la tecla "Shift" está presionada. |
| isChar | Verdadero si la tecla pulsada es un carácter. |
| button | El botón (número) del mouse que ha sido pulsado. |
| detail | Número de veces que se han pulsado los botones del mouse. |
| clientX | Coordenada horizontal del mouse respecto a la ventana actual. |
| clientY | Coordenada vertical del mouse respecto a la ventana actual. |
| pageX | Coordenada horizontal del mouse respecto a la página. |
| pageY | Coordenada vertical del mouse respecto a la página. |
| screenX | Coordenada horizontal del mouse respecto a la pantalla. |
| screenY | Coordenada vertical del mouse respecto a la pantalla. |

Algunos de los métodos (funciones) del objeto "event" son:

| Método | Descripción |
|---|---|
| stopPropagation() | Detiene la propagación del evento. El evento no pasa a los elementos dentro del cual se encuentra el elemento donde se produjo el evento. |
| preventDefault() | Cancela la acción que por defecto genera el evento. |

Cuando se enlaza, mediante código Javscript, el manejador de eventos al elemento (como se verá posteriormente), el objeto "event" es pasado automáticamente, pero cuando se realiza una asignación manual, como se ha procedido hasta ahora, el objeto "event" debe ser pasado explícitamente, por ejemplo, la siguiente instrucción crea un <input> de tipo "text" que responde al evento "onkeydown" mediante el manejador "validar" al que se le manda el objeto "event".

```
<input type="text" id="input1" onkeydown="validar(event)">
```

El manejador de eventos por su parte (la función) recibe el objeto asignándole cualquier nombre, aunque casi siempre dicho nombre es "event" o "e":

```
function validar(event){
  ...
}

function validar(e){
  ...
}
```

En la función también se puede acceder al evento mediante la variable "arguments". Esta variable se crea automáticamente para cada función y es

una especie de array, donde cada uno de sus elementos corresponde a cada uno de los argumentos (datos) que se pasan a la función, puesto que en un manejador de eventos el primer (y único) argumento es el objeto "event", se accede al mismo con "arguments[0]":

```
function validar(){
  var event=arguments[0];
  ...
}
```

Una vez recibido el evento se pueden acceder a sus propiedades y/o métodos empleando la notación de objetos, así el código de la tecla pulsada se obtiene con: "event.keyCode".

## 3.2. TRATAMIENTO DE ERRORES

En el anterior tema se ha aprendido a generar errores en Javascript (con throw) y como se vio, cuando se genera un error, el programa se detiene inmediatamente, sin devolver ningún resultado ni mostrar realmente ningún mensaje.

En este tema tema se comienza a tratar dichos errores de manera que muestren información más útil o se pueda y para ello se recurre a la estructura "*try-catch-finally*":

```
try {
  Instrucciones a ejecutar (código normal)
} catch (variable que captura el error) {
  Instrucciones a ejecutar si ocurre un error en el bloque try
} finally {
  Intrucciones a ejecutar sea que ocurra o no un error (opcional)
}
```

Dentro del bloque *try* se escriben las instrucciones que se quieren ejecutar, es decir las instrucciones normales. Este es el código que se escribiría usualmente si no se hiciera el tratamiento del error.

Si en el bloque *try* se produce algún error la ejecución del código se detiene y el programa salta al bloque *catch*, donde el error (un objeto) es asignado a la variable escrita entre los paréntesis. Entonces se ejecutan las instrucciones escritas dentro del bloque *catch*.

Por otra parte, el bloque *finally*, se ejecuta siempre, ya sea que se produzca o no un error. Normalmente se emplea para liberar memoria y otros recursos reservados por el programa.

La estructura "try" puede estar conformada por los tres bloques: *try-catch-finally* o por sólo dos *try-catch* o *try-finally*, siendo el bloque *try-catch* el más frecuente.

A más del objeto "Error", generado en el anterior tema (con *throw new Error(…)*) JavaScript cuenta con otros tipos de error: ***EvalError, RangeError, ReferenceError, SyntaxError, TypeError y URIError***. Estos errores pueden ser generados también con "throw, así por ejemplo para generar un erro de rango se escribe:

```
throw new RangeError("Mensaje del error");
```

## 3.3. EJEMPLOS

**1. Cree una página HTML que permita introducir un número entero positivo y que, mediante una función recursiva, muestre el factorial del mismo.**

Este es el ejemplo clásico de recursividad. Desde el punto de vista iterativo se define como una productoria:

$$n! = \prod_{i=1}^{n} i$$

Que se resuelve con ciclo "for" que va desde 2 hasta "n" (no 1 porque el acumulador comienza en 1):

```
function factorial(n){
  var p=1;
  for (var i=2;i<=n;i++){p*=i;}
  return p;
}
```

Desde el punto de vista recursivo, la misma función se define como:

$$n! = (n-1)! \cdot n$$

Es decir que, el factorial de un número, es igual al factorial del número anterior multiplicado por dicho número. Por ejemplo el factorial de 5 es igual al factorial de 4 multiplicado por 5, es decir:

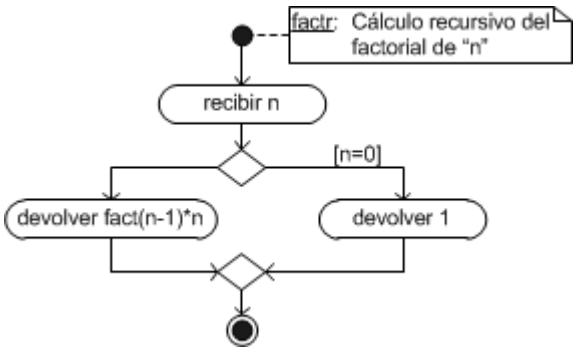
$$5! = 4! \cdot 5 = 24 \cdot 5 = 120$$

Qué efectivamente es el resultado correcto, por lo tanto el planteamiento recursivo (en este caso la fórmula recursiva) es correcto.

Como ya se dijo **en el planteamiento recursivo sólo interesa que el planteamiento recursivo devuelva el resultado buscado,** en este caso por ejemplo, no es parte del problema averiguar cómo se calcula el factorial de 4, sólo interesa verificar que al multiplicar el factorial de 4 por 5 se obtiene el factorial de 5.

Como también se dijo, **en toda solución recursiva se debe contar además con una condición de finalización,** pues de lo contrario el proceso sería infinito. En este caso se sabe (por definición) que el factorial de 0 es 1, por lo tanto esa es la condición de finalización natural del problema, es decir que el proceso debe concluir cuando "n" sea 0 devolviendo el resultado 1.

El algoritmo, en forma de diagrama de actividades, es:




Sólo con el propósito de ilustrar lo que ocurre en el proceso recursivo y comprender así mejor en qué consiste, se muestra a continuación los pa-

sos que lleva a cabo el lenguaje que se está empleando (no el programador) para resolver el problema de forma recursiva.

La primera vez que se llama a *fact* el parámetro "*n*" toma el valor 5 y como no es igual a 0, se ejecuta la instrucción:

*fact(5-1)*5 = fact(4)*5*

Ahora, para calcular el factorial de 4 (fact(4)) la función se llama a sí misma y en esta llamada (n=4), se ejecuta la instrucción:

*fact(4-1)*4 = fact(3)*4*

Una vez más, para calcular el factorial de 3, la función se llama a sí misma y sigue así hasta que "n" es igual a 0:

*fact(3-1)*3 = fact(2)*3*

*fact(2-1)*2 = fact(1)*2*

*fact(1-1)*1 = fact(0)*1*

Ahora la función se llama a sí misma con "n" igual a 0 y como se cumple la condición de finalización (n es igual a 0) devuelve el resultado 1. Este resultado es devuelto a la función que hizo la llamada (la copia de la función para "n" igual a 1), donde "fact(0)" es reemplazado por el resultado devuelto (1), valor con el cual se lleva a cabo la multiplicación:

*fact(0)*1 = 1*1 = 1*

Este resultado (1) es devuelto a la función que hizo la llamada (la copia de la función para "n" igual a 2), e igual que el caso anterior, con este resultado se puede calcular ahora el factorial de 2:

*fact(1)*2 = 1*2 = 2*

Y continúa así hasta llegar a la primera llamada recursiva:

*fact(2)*3 = 2*3 = 6*
*fact(3)*4 = 6*4 = 24*
*fact(4)*5 = 24*5 = 120*

Siendo este el resultado devuelto por la función recursiva. **Como se puede ver el proceso recursivo en si es moroso y hasta confuso, pero como se dijo, dicho proceso no es parte del problema,** del mismo se encarga automáticamente el lenguaje de programación.

Observe también que en el algoritmo se ha resuelto exclusivamente el problema sin validar el dato recibido. Esto es algo que siempre se debe hacer al resolver un problema de manera recursiva: **la función recursiva sólo debe resolver el problema en sí, la validación debe ser hecha en una función no recursiva desde la cual se llame a la recursiva.**

El hacer la anterior división es muy importante cuando se resuelven problemas de forma recursiva, porque si los casos especiales se tratan en la función recursiva, dicho tratamiento ocurre en cada llamada recursiva, lo que constituye un error porque sólo deben ser tratados una vez: la primera vez que se llama a la función.

En cuanto a la interfaz de usuario, en este caso el dato será leído en un <input> de tipo texto. Esto porque existe un número infinito de números naturales (enteros positivos) y no es posible crear un <select> con un número infinito de opciones. Se podría emplear uno de los nuevos atributos HTML5 (como "number"), pero como ya se dijo, no son aún soportadas por todos los navegadores. La validación se logrará mediante un manejador de eventos y el objeto "event".

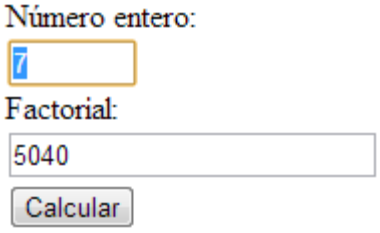
El resultado será mostrado también en un <input> de tipo "text", pero con la propiedad "readonly" (sólo lectura), pues dicho elemento sólo debe mostrar los resultados, sin permitir modificarlos.

El código escrito para resolver el problema es:

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo 1</title>
  <script src="sis101.js"></script>
  <script>
    //Cálculo recursivo del factorial
    function factr(n){
      if (n==0) return 1;
      return factr(n-1)*n;
    }
    //Validación del número y llamada a la función recursiva
    function factorial(n){
      if(isNaN(n))
        throw new Error("fact: El dato debe ser numérico.");
      if(frac(n)!=0)
        throw new Error("fact: El número debe ser entero.");
      if(n<0)
        throw new Error("fact: El número debe ser positivo.");
      return factr(n);
    }
    //Cálculo y presentación del factorial
    function calcular(){
      try{
        var n=parseInt(document.getElementById("num").value);
        document.getElementById("res").value=factorial(n);
        document.getElementById("num").focus();
        document.getElementById("num").select();
      } catch(e){
        alert("factorial: "+e.message);
      }
    }
    //Valida el dato escrito en el input "num"
    function vDato(){
      var n=document.getElementById("num").value;
      try{
        if(isNaN(n)) throw new Error("El dato debe ser un número");
        if(frac(n)!=0) throw new Error("El número debe ser entero");
        if(n<0) throw new Error("El entero debe ser positivo");
        return true;
      } catch(e){
          alert(e.message);
          document.getElementById("num").select();
          document.getElementById("num").focus();
```

```
        return false;
      }
    }
    //Valida la escritura de números naturales (enteros positivos)
    function vNatural(event){
      //tecla pulsada
      var key=event.which;
      //Si la tecla pulsada está entre 0 y 9
      if (key>=48 && key<=57) return true;
      //Si la tecla es retroceso (8), tab (9), fin (35), inicio (36),
      //cursor izquierdo (37), cursor derecho (39) o suprimir (46)
      if (key==8 || key==9 || key==35 || key==36 || key==37 ||
        key==39 || key==46) return true;
      //Si la tecla pulsada es Enter y el valor escrito es un número
      if (key==13 && vDato()){
        document.getElementById("btn1").onclick();
        return true;
      }
      return false;
    }
    //Selecciona el input "num" al cargar la página
    function iniciar(){
      document.getElementById("num").focus();
      document.getElementById("num").select();
    }
  </script>
</head>
<body onload="iniciar()">
  <label for="num">Número entero:<br>
  <input type="text" id="num" value="0" size="5"
    onkeydown="return vNatural(event)"><br>
  <label for="res">Factorial:<br>
  <input type="text" id="res" value="1" size="25" readonly><br>
  <button type="button" id="btn1" onclick="calcular()">Calcular
  </button>
</body>
</html>
```

Que al ser abierta en el navegador tiene la siguiente apariencia:




Como una vez más se hace evidente en este ejemplo, el problema en sí (calcular el factorial) constituye sólo un 5% del código escrito, la mayor parte del código tiene que ver con la validación de datos y la interfaz de usuario. Esto es algo que ocurre en la mayoría de las aplicaciones

prácticas, y no es raro, porque las aplicaciones son creadas justamente para los usuarios y como se dijo, para los usuarios la aplicación es la interfaz de la aplicación.

Sin embargo, sin ese 5% y sin importar cuan atractiva y bien diseñada sea la interfaz, la aplicación sería inútil, por lo que es crucial que ese 5% (que para el usuario no existe) funcione no sólo correctamente, sino también de forma eficiente.

El cálculo recursivo del factorial es realizado por la función "factr". Esta función es llamada desde "factorial", que es donde se valida el dato recibido. En esta aplicación en particular, debido a que en la interfaz se valida el dato introducido por el usuario (con las funciones "vNatural" y "vDato"), es prácticamente imposible que no sea un número entero positivo (un número natural), por lo que se podría pensar que esta validación es inútil, pero no es así, porque las funciones (los módulos) deben ser independientes (como señala el primer principio de la programación estructurada) y ello es necesario porque los módulos deben funcionar correctamente sin importar de donde provenga él o los datos que recibe. En esta aplicación los datos han sido validados previamente, pero el módulo debe funcionar correctamente en cualquier aplicación, sin importar de donde provengan los datos.

En esta aplicación el dato escrito por el usuario se valida en dos módulos "vNatural" y "vDato".

En "vNatural" se verifica que la tecla pulsada sea un número o una de las teclas permitidas y de ser así devuelve el valor lógico verdadero (true). En cualquier otro caso devuelve el valor lógico falso (false). El valor devuelto por esta función es empleado en la etiqueta <input id="num"> con la instrucción 'onkeydown="return vNatural()"', de esta manera si el valor devuelto por "vNatural" es "true" el elemento <input> se comporta de forma normal, es decir muestra la tecla pulsada, mueve el cursor o borra un carácter, pero si es "false" ignora la tecla pulsada y es como si la misma no hubiese sido pulsada. De esta manera se consigue que en este <input> sólo sea posible escribir números.

Como se puede ver la tecla "Enter" (código 13) se trata de forma diferente. Cuando la tecla pulsada es "Enter", se llama al evento "onclick" del botón (document.getElementById("btn1").onclick()), de esta manera al pulsar la tecla "Enter" se llama a la misma función que al hacer clic en el botón, por lo tanto, el cálculo del factorial ocurre tanto al hacer clic en el botón como al pulsar la tecla "Enter" (después de escribir el número).

Con la anterior validación se asegura que el dato introducido por el usuario sea un número, pues no se permite escribir nada más y si todos los navegadores respondieran de la misma forma no sería necesaria ninguna validación adicional, lamentablemente no es así y en algunos navegadores (sobre todo en los dispositivos móviles) los <input> no responden correctamente a los eventos de teclado por lo que la anterior validación no funciona en los mismos. Por eso es necesario contar con un segundo módulo de validación "vDato".

En "vDato" básicamente se verifica que el número introducido sea un número entero positivo. De no ser así se muestra un mensaje (aprovechando la estructura *try-catch*), se devuelve el foco al <input> y se selecciona el dato escrito (con "select()").

Observe que los errores generados por la función "factorial" no son tratados (con *try*) en la misma función, sino que son tratados en la función "calcular" que es donde se llama a (se emplea) la función "facto-

rial". Esto último es el procedimiento más frecuente: los errores no son tratados en el módulo que los genera sino en el módulo que llama (utiliza) dicho módulo.

Para que al iniciar la aplicación el foco se encuentre en el <input> y el valor del mismo esté seleccionado se ha asignado al evento "onload" de <body> el manejador "iniciar". El evento "onload" de <body> ocurre cuando la página termina de ser cargada, es decir cuando todos los elementos de la página han sido cargados por el navegador.

**2. Elabore una página HTML que dado un número natural, calcule, mediante una función recursiva, el número de dígitos del mismo.**

Se puede determinar el número de dígitos de un número sumando uno al número de dígitos de dicho número sin su último dígito. Por ejemplo el número de dígitos de 345212, se calcula con:

numdig(345212) = numdig(34521)+1 = 5+1 = 6

Que es el resultado correcto, por lo tanto el planteamiento recursivo es correcto. Una vez más se recalca que no es parte del problema calcular el número de dígitos del número sin su último dígito, sólo se debe comprobar que sumando 1 a ese número de dígitos se obtiene el resultado correcto (lo que efectivamente ocurre).

Para completar el planteamiento recursivo se requiere además una condición de finalización y como el número de dígitos de 0, es 0, esa será la condición de finalización a emplear.

Para obtener el número sin su último dígito simplemente se calcula el cociente de su división entre 10.

En cuanto a la interfaz del usuario (y por las mismas razones) será prácticamente la misma que la del anterior ejemplo.

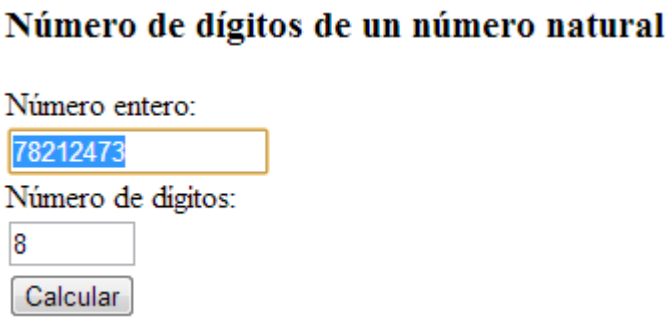
El código de la página HTML que resuelve el problema es:

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo 2</title>
  <script src="sis101.js"></script>
  <script>
    //Cálculo recursivo del número de dígitos
    function numdigr(n){
      if (n!=0) return numdig(quot(n,10))+1;
      return 0;
    }
    //Validación del número y llamada a la función recursiva
    function numdig(n){
      if(isNaN(n))
        throw new Error("fact: El dato debe ser numérico.");
      if(frac(n)!=0)
        throw new Error("fact: El número debe ser entero.");
      if(n<0)
        throw new Error("fact: El número debe ser positivo.");
      return numdigr(n);
    }
    //Cálculo y presentación del número de dígitos
```

```
    function calcular(){
      try{
        var n=parseInt(document.getElementById("num").value);
        document.getElementById("res").value=numdig(n);
        document.getElementById("num").focus();
        document.getElementById("num").select();
      } catch(e){
        alert("factorial: "+e.message);
      }
    }
    //Valida el dato escrito en el input "num"
    function vDato(){
      var n=document.getElementById("num").value;
      try{
        if(isNaN(n)) throw new Error("El dato debe ser un número");
        if(frac(n)!=0) throw new Error("El número debe ser entero");
        if(n<0) throw new Error("El entero debe ser positivo");
        return true;
      } catch(e){
          alert(e.message);
          document.getElementById("num").select();
          document.getElementById("num").focus();
          return false;
      }
    }
    //Valida la escritura de números naturales (enteros positivos)
    function vNatural(event){
      //tecla pulsada
      var key=event.which;
      //Si la tecla pulsada está entre 0 y 9
      if (key>=48 && key<=57) return true;
      //Si la tecla es retroceso (8), tab (9), fin (35), inicio (36),
      //cursor izquierdo (37), cursor derecho (39) o suprimir (46)
      if (key==8 || key==9 || key==35 || key==36 || key==37 ||
        key==39 || key==46) return true;
      //Si la tecla pulsada es Enter y el valor escrito es un número
      if (key==13 && vDato()){
        document.getElementById("btn1").onclick();
        return true;
      }
      return false;
    }
    //Selecciona el input "num" al cargar la página
    function iniciar(){
      document.getElementById("num").focus();
      document.getElementById("num").select();
    }
  </script>
</head>
```

```
<body onload="iniciar()">
  <h3>Número de dígitos de un número natural</h3>
  <label for="num">Número entero:<br>
  <input type="text" id="num" value="0" size="16"
    onkeydown="return vNatural(event)"><br>
  <label for="res">Número de dígitos:<br>
  <input type="text" id="res" value="1" size="5" readonly><br>
  <button type="button" id="btn1" onclick="calcular()">Calcular
  </button>
</body>
</html>
```

Que al ser abierta en un navegador tiene la apariencia:




Como se puede ver, con excepción del módulo recursivo, el código es esencialmente el mismo que el del problema anterior.

**3. Elabore una página HTML que calcule, mediante una función recursiva, la sumatoria de los primeros "n" números impares.**

Se puede calcular la suma de los primeros "n" números impares si a la suma de los primeros "n-1" números impares se le suma el último número impar. Por ejemplo la sumatoria de los primeros 10 números impares es:

```
sumai(10) = sumai(9)+(2*10-1) = 81+19 = 100
```

Que es el resultado correcto, por lo tanto el planteamiento recursivo es correcto. Como siempre, no es parte del problema el cómo se calcula la sumatoria de los "n-1" números impares, sólo se debe comprobar que sumando el último número impar a esa sumatoria se obtiene el resultado correcto (lo que efectivamente sucede).

Para completar el planteamiento recursivo se requiere además una condición de finalización y como la sumatoria de 0 números impares es 0, esa será la condición de finalización.

En cuanto a la interfaz del usuario (y por las mismas razones) será prácticamente la misma que la del anterior ejemplo.

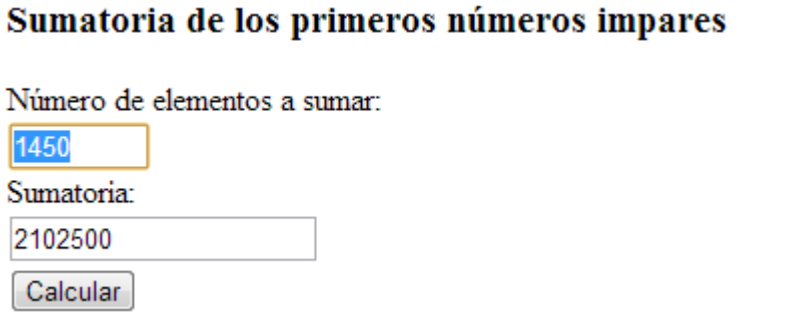
El código de la página HTML que resuelve el problema es:

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo 3</title>
  <script src="sis101.js"></script>
  <script>
    //Cálculo recursivo del número de dígitos
```

```
function sumir(n){
  if (n!=0) return sumi(n-1)+2*n-1;
  return 0;
}
//Validación del número y llamada a la función recursiva
function sumi(n){
  if(isNaN(n))
    throw new Error("fact: El dato debe ser numérico.");
  if(frac(n)!=0)
    throw new Error("fact: El número debe ser entero.");
  if(n<0)
    throw new Error("fact: El número debe ser positivo.");
  return sumir(n);
}
//Cálculo y presentación del número de dígitos
function calcular(){
  try{
    var n=parseInt(document.getElementById("num").value);
    document.getElementById("res").value=sumi(n);
    document.getElementById("num").focus();
    document.getElementById("num").select();
  } catch(e){
    alert("factorial: "+e.message);
  }
}
//Valida el dato escrito en el input "num"
function vDato(){
  var n=document.getElementById("num").value;
  try{
    if(isNaN(n)) throw new Error("El dato debe ser un número");
    if(frac(n)!=0) throw new Error("El número debe ser entero");
    if(n<0) throw new Error("El entero debe ser positivo");
    return true;
  } catch(e){
      alert(e.message);
      document.getElementById("num").select();
      document.getElementById("num").focus();
      return false;
  }
}
//Valida la escritura de números naturales (enteros positivos)
function vNatural(event){
  //tecla pulsada
  var key=event.which;
  //Si la tecla pulsada está entre 0 y 9
  if (key>=48 && key<=57) return true;
  //Si la tecla es retroceso (8), tab (9), fin (35), inicio (36),
  //cursor izquierdo (37), cursor derecho (39) o suprimir (46)
  if (key==8 || key==9 || key==35 || key==36 || key==37 ||
```

```
        key==39 || key==46) return true;
      //Si la tecla pulsada es Enter y el valor escrito es un número
      if (key==13 && vDato()){
        document.getElementById("btn1").onclick();
        return true;
      }
      return false;
    }
    //Selecciona el input "num" al cargar la página
    function iniciar(){
      document.getElementById("num").focus();
      document.getElementById("num").select();
    }
  </script>
</head>
<body onload="iniciar()">
  <h3>Sumatoria de los primeros números impares</h3>
  <label for="num">Número de elementos a sumar:<br>
  <input type="text" id="num" value="0" size="6"
    onkeydown="return vNatural(event)"><br>
  <label for="res">Sumatoria:<br>
  <input type="text" id="res" value="1" size="20" readonly><br>
  <button type="button" id="btn1" onclick="calcular()">Calcular
  </button>
</body>
</html>
```

Que al ser abiertas en el navegador tiene la apariencia:




El atributo "size" empleado en las etiquetas <input> (en este y en los anteriores ejemplos) determina el ancho del elemento.

## 3.4. EJERCICIOS

Para presentar los ejemplos y ejercicios del tema, debe crear una carpeta con el nombre "tema3" y guardar en la misma todos los archivos HTML creados (ejemplos y ejercicios).

1. Cree una página HTML que, mediante una función recursiva, calcule el valor de $x^n$, siendo "x" un número real y "n" un número entero positivo.
2. Cree una página HTML que, mediante una función recursiva, calcule la sumatoria de los primeros "n" números enteros positivos.

3. Cree una página HTML que reciba un número entero positivo y mediante una función recursiva, devuelva dicho número sin su primer dígito.

# 4. MATRICES

En este tema comienza el estudio de los datos estructurados que, conjuntamente las estructuras estándar, constituyen la base de la programación estructurada. En general, se dice que ***un dato es estructurado si está conformado por dos o más datos***.

Las matrices (vectores o arreglos) constituyen la primera estructura estándar porque están conformadas por dos o más datos del mismo tipo. Estrictamente hablando una matriz sólo puede guardar datos del mismo tipo, sin embargo, la mayoría de los lenguajes actuales, sobre todo los interpretados como Javascript, permiten almacenar, en una misma matriz, diferentes tipos de datos.

Esto es posible porque las matrices en Javascript no guardan los datos propiamente, sino sólo *punteros*. En ese sentido cumple con la condición de que las matrices contengan un solo tipo de dato (pues el único tipo de dato que contienen son los punteros).

**Un puntero simplemente es una dirección de memoria** y como tal, puede apuntar (tener la dirección de memoria) de cualquier dato (incluido otro puntero o una función), esa es la razón por la cual una lista aparentemente contiene diferentes tipos de datos, sin embargo (internamente), sólo contiene punteros (elementos del mismo tipo).

Aun cuando el manejo y uso de punteros es transparente en la mayoría de los lenguajes modernos (es decir que se crean, actualizan y destruyen automáticamente), **es importante tener en mente** (para evitar comportamientos inesperados), **que los punteros son sólo direcciones de memoria y no él o los datos en sí**.

La mayoría de los elementos, en los lenguajes actuales, son en realidad punteros (un ejemplo son los vectores y otro los objetos).

Para comprender mejor el por qué es importante recordar que los punteros son sólo direcciones de memoria y comprobar que los mismos son creados transparentemente por el lenguaje de programación, escriba las siguientes instrucciones en la calculadora Javascript:

```
>>a=[1,2,3]
[1, 2, 3]
>>b=a;
[1, 2, 3]
>>a.push(4)
4
>>a
[1, 2, 3, 4]
>>b
[1, 2, 3, 4]
```

Como se puede observar, se crea la variable "a" (un vector), luego se asigna la variable "a" a la variable "b", con lo que se esperaría se cree una copia, sin embargo, al añadir un elemento (el número 4) al final del vector "a" (con "a.push(4)") no solo cambia el vector "a" (como se esperaría) sino también el vector "b". Ocurre exactamente lo mismo cuando se modifica la variable "b", por ejemplo si se añade un elemento (el número 0) al principio del vector "b" (con "b.unshift(0)"), lo que cambia no sólo el vector "b", sino también el vector "a":

```
>>b.unshift(0)
5
```

```
>>a
[0, 1, 2, 3, 4]
```

Este comportamiento se comprende si se toma en cuenta que "a" y "b" son punteros, es decir que "a" no contiene los números "1,2,3", sino solo la dirección donde se encuentran dichos números, por lo tanto cuando "b" toma el valor de "a", lo que se guarda en "b" no son los números "1,2,3", sino la dirección de memoria donde se encuentran dichos números. En consecuencia, cualquier modificación a los valores de "a" o "b" es una modificación en esa dirección de memoria y como ambas variables muestran (automáticamente) el contenido de dicha dirección, cualquier cambio hecho en "a" es también un cambio en "b" y viceversa.

## 4.1. DECLARACIÓN DE VECTORES

En JavaScript en realidad sólo existen vectores y como ya se vio, la forma más sencilla de crear un vector (o lista) es escribir sus elementos entre corchetes (separados con comas), por ejemplo, con las siguientes instrucciones se crean tres vectores (en la calculadora Javascript):

```
>>a=[5.1,2.2,3.5,6.8]
[5.1, 2.2, 3.5, 6.8]
>>b=["a","b","c","d"]
[a, b, c, d]
>>c=[1,3,4,"a","hola",6,"mundo",5.7,8]
[1, 3, 4, a, hola, 6, mundo, 5.7, 8]
```

El tercer vector "c", aparentemente contiene diferentes tipos de datos, pero como ya se explicó en el anterior tema, en realidad lo único que contiene son punteros y como ya se dijo, los punteros pueden apuntar a diferentes tipos de datos.

Alternativamente (y formalmente más correcto) se pueden crear los vectores con "new Array":

```
>>a=new Array(5.1,2.2,3.5,6.8)
[5.1, 2.2, 3.5, 6.8]
>>b=new Array("a","b","c","d")
[a, b, c, d]
>>c=new Array(1,3,4,"a","hola",6,"mundo",5.7,8)
[1, 3, 4, a, hola, 6, mundo, 5.7, 8]
```

Sin embargo, si se quiere crear un vector con un solo elemento numérico, ese elemento es interpretado como la dimensión del vector (el número de elementos del vector), por ejemplo, la instrucción:

```
>>d=new Array(10)
[undefined, undefined, undefined, undefined, undefined, undefined,
undefined, undefined, undefined, undefined]
```

Crea un vector con 10 elementos inicialmente indefinidos (sin valor), no, como se podría esperar, un vector con el elemento número 10.

Para crear un vector con un elemento (con este método), se crea primero un vector vacío y luego se le asigna el valor al primer elemento:

```
>>d=new Array(); d[0]=10
10
>>d
[10]
```

Como se puede ver en este ejemplo, Javascript añade automáticamente

elementos al vector en la posición que se le indica. Si existen elementos sin definir entre la nueva posición y el último elemento añadido, los elementos intermedios quedan sin definir, por ejemplo si al vector "d" se le añade un elemento en la décima posición (recuerde que el primer índice es 0) se obtiene:

```
>>d[9]=12
12
>>d
[undefined, undefined, undefined, undefined, undefined, undefined,
undefined, undefined, undefined, 12]
```

Con "new Array", la declaración de los vectores "a", "b" y "c", es:

```
>>a=new Array(); a[0]=5.1; a[1]=2.2; a[2]=3.5; a[3]=6.8; a
[5.1, 2.2, 3.5, 6.8]
>>b=new Array(); b[0]="a"; b[1]="b"; b[2]="c"; b[3]="d"; b
[a, b, c, d]
>>c=new Array(); c[0]=1; c[1]=3; c[2]=4; c[3]="a"; c[4]="hola"; c[5]=6;
c[6]="mundo"; c[7]=5; c[8]=7; c[9]=8; c
[1, 3, 4, a, hola, 6, mundo, 5, 7, 8]
```

Donde se ha escrito "a", "b" y "c", al final de cada instrucción, simplemente para mostrar el vector creado. Se obtiene el mismo resultado si se especifica previamente el número de elementos:

```
>>a=new Array(4); a[0]=5.1; a[1]=2.2; a[2]=3.5; a[3]=6.8; a
[5.1, 2.2, 3.5, 6.8]
>>b=new Array(4); b[0]="a"; b[1]="b"; b[2]="c"; b[3]="d"; b
[a, b, c, d]
>>c=new Array(10); c[0]=1; c[1]=3; c[2]=4; c[3]="a"; c[4]="hola";
c[5]=6; c[6]="mundo"; c[7]=5; c[8]=7; c[9]=8; c
[1, 3, 4, a, hola, 6, mundo, 5, 7, 8]
```

Las cuatro formas de declarar matrices (más propiamente vectores) son equivalentes, pero desde el punto de vista práctico la primera (escribir los elementos entre corchetes) es la más sencilla, por lo que se la usa siempre que sea posible.

## 4.2. DECLARACIÓN DE MATRICES

En Javascript sólo existen vectores, entonces ¿cómo es posible trabajar con matrices en Javascript? La respuesta es sencilla si se recuerda que en JavaScript todos los elementos son punteros y que, como ya se sabe, un puntero puede apuntar a cualquier tipo de dato, incluido otro vector. Ahora bien, una matriz es simplemente un vector de vectores, por lo tanto si cada uno de los elementos de un vector es a su vez otro vector, se tiene una matriz.

Entonces, para declarar una matriz en Javascript, simplemente se crea un vector de vectores, por ejemplo, para crear una variable con los elementos de la siguiente matriz:

$$\begin{vmatrix} 1 & 3 & 5 & 6 \\ 3 & 5 & 7 & 2 \\ 2 & 8 & 9 & 4 \end{vmatrix}$$

Se escribe:

```
>>a=[[1,3,5,6],[3,5,7,2],[2,8,9,4]]
[[1, 3, 5, 6], [3, 5, 7, 2], [2, 8, 9, 4]]
```

Al ser simplemente un vector de vectores, las matrices pueden ser declaradas en cualquiera de formas vistas previamente, o mediante de una combinación de las mismas:

```
>>a=new Array([1,3,5,6],[3,5,7,2],[2,8,9,4]); a
[[1, 3, 5, 6], [3, 5, 7, 2], [2, 8, 9, 4]]
```

```
>>a=new Array(); a[0]=[1,3,5,6]; a[1]=[3,5,7,2]; a[2]=[2,8,9,4]; a
[[1, 3, 5, 6], [3, 5, 7, 2], [2, 8, 9, 4]]
```

```
>>a=new Array(); a[0]=new Array(1,3,5,6); a[1]=new Array(3,5,7,2);
a[2]=new Array(2,8,94); a
[[1, 3, 5, 6], [3, 5, 7, 2], [2, 8, 94]]
```

```
>>a=new Array(3);
[undefined, undefined, undefined]
>>a[0]=new Array(4); a[0][0]=1; a[0][1]=3; a[0][2]=5; a[0][3]=6; a[0]
[1, 3, 5, 6]
>>a[1]=new Array(4); a[1][0]=3; a[1][1]=5; a[1][2]=7; a[1][3]=2; a[1]
[3, 5, 7, 2]
>>a[2]=new Array(4); a[2][0]=2; a[2][1]=8; a[2][2]=9; a[2][3]=4; a[2]
[2, 8, 9, 4]
>>a
[[1, 3, 5, 6], [3, 5, 7, 2], [2, 8, 9, 4]]
```

Una vez más la primera forma es la más sencilla (y con mucho), sin embargo, en la elaboración de programas por lo general se emplea la última, porque es la forma en que se puede acceder a los elementos individualmente.

Se debe tener presente que al ser una matriz un vector de vectores, cada fila puede tener diferente número de elementos. Esa puede ser una fuente de errores cuando se quiere trabaja con matrices rectangulares, pero puede ser de utilidad para trabajar con matrices irregulares o especiales como la siguiente matriz triangular inferior:

$$\begin{vmatrix} 1 & & & & \\ 2 & 3 & & & \\ 4 & 5 & 6 & & \\ 7 & 8 & 9 & 10 & \\ 11 & 12 & 13 & 14 & 15 \end{vmatrix}$$

Que puede ser creada con:

```
>>b=[[1],[2,3],[4,5,6],[7,8,9,10],[11,12,13,14,15]]
[[1], [2, 3], [4, 5, 6], [7, 8, 9, 10], [11, 12, 13, 14, 15]]
```

Que mostrada fila por fila (con "mshow"), es:

```
>>mshow(b)
[[1]
 [2, 3]
 [4, 5, 6]
 [7, 8, 9, 10]
 [11, 12, 13, 14, 15]]
```

¿Cuál es la ventaja? Que no es necesario almacenar ceros para los elementos inexistentes. De esa manera se emplea menos memoria que en una ma-

triz rectangular estándar.

## 4.3. PROPIEDADES Y MÉTODOS

Los vectores en Javascript son objetos y como tales tienen una serie de propiedades (atributos) y métodos. En este acápite se verán algunas de dichas propiedades y métodos.

La propiedad "**length**", devuelve el número de elementos del vector, así, para el siguiente vector:

```
>>v=[3,2,4,6,2,3,1,7]
[3, 2, 4, 6, 2, 3, 1, 7]
```

Que tiene 8 elementos, la propiedad "length" devuelve dicho número:

```
>>v.length
8
```

En el caso de las matrices, al ser cada elemento un vector y representar cada elemento una fila, la propiedad "length" devuelve el número de filas, así, para la siguiente matriz:

```
>>m=[[1,2,3,4],[2,3,4,5],[3,4,5,6]]
[[1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6]]
```

La propiedad "length" devuelve el número de filas (3):

```
>>m.length
3
```

Cuando se trata de una matriz rectangular, como en este caso, el número de columnas es el número de elementos de cualquiera de sus filas, por lo tanto, para averiguar el número de columnas de una matriz rectangular, simplemente se averigua el número de elementos de la primera fila:

```
>>m[0].length
4
```

En las matrices irregulares esta operación debe ser repetida para cada fila (porque cada fila puede tener diferente número de columnas).

El método "**concat(a1,a2,…)**" une los elementos de una, dos o más matrices a los elementos de la matriz desde la cual se llama al método. Por ejemplo, si se crean las siguientes matrices:

```
>>a=[1,2,3,4]
[1, 2, 3, 4]
>>b=[5,6,7]
[5, 6, 7]
>>c=[8,9,10]
[8, 9, 10]
```

Y se quiere unir los elementos de las matrices "b" y "c" a la matriz "a", se escribe:

```
>>d=a.concat(b,c)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Con lo que se crea un nuevo vector, que en este ejemplo ha sido guardado en la variable "d". El vector original "a", desde el cual se llama al método "concat", no es modificado:

```
>>a
[1, 2, 3, 4]
```

El método "**join(s)**", devuelve una cadena (String) con los elementos del vector desde el cual se llama al método. Para separar los elementos "join" emplea la cadena de caracteres "s". Por ejemplo, con la siguiente instrucción se crea una cadena de caracteres con los elementos del vector "d", quedando los elementos separados con una coma y un espacio ", ":

```
>>s=d.join(", ")
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

El método "**push(e1,e2,…)**" añade él o los elementos "e1", "e2",…, al final del vector desde el cual se llama al método. El método devuelve el nuevo número de elementos del vector. Por ejemplo, dado el vector:

```
>>x=[1,2,3,4,5,6]
[1, 2, 3, 4, 5, 6]
```

La siguiente instrucción añade 6 elementos al final del mismo:

```
>>x.push(7,8,9,10,11,12)
12
```

Y como se puede ver devuelve el número 12, que es el nuevo número de elementos del vector:

```
>>x
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Al ser las matrices un vector de vectores, se puede emplear este método para añadir filas a una matriz, por ejemplo, dada la matriz:

```
>>y=[[1,2,3],[4,5,6]]
[[1, 2, 3], [4, 5, 6]]
```

La siguiente instrucción, añade tres filas a la matriz:

```
>>y.push([7,8,9],[10,11,12],[13,14,15])
5
```

Donde el número 5, es el nuevo número de filas de la matriz:

```
>>y
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15]]
```

El método "**pop()**", remueve el último elemento del vector, devolviendo el elemento removido. Por ejemplo, la siguiente instrucción quita el último elemento (el número 12) de la matriz "x":

```
>>x.pop()
12
```

Con lo que ahora los elementos del vector "x" son:

```
>>x
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Por supuesto "pop" se puede emplearse también remover la última fila de una matriz, así aplicando este método a la matriz "y", se obtiene:

```
>>y.pop()
[13, 14, 15]
```

Con lo que la matriz "y" queda con las filas:

```
>>y
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
```

El método "**shift()**" es similar a "pop", sólo que en lugar de remover el

último elemento del vector, remueve el primero, por ejemplo, aplicando este método a la matriz "x", se obtiene:

```
>>x.shift()
1
```

Con lo que el vector "x" queda con los elementos:

```
>>x
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

El método "**unshift(e1,e2,…)**" hace lo contrario que "shift", es decir añade él o los elementos "e1", "e2",…, al principio del vector y devuelve el nuevo número de elementos. Por ejemplo con la siguiente instrucción se añaden los elementos -3, -2, -1, 0 y 1 al vector "x":

```
>>x.unshift(-3,-2,-1,0,1)
15
>>x
[-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Como de costumbre, estos métodos pueden ser empleados también con las matrices (pues simplemente son vectores de vectores), así la siguiente instrucción añade dos filas al principio de la matriz "y":

```
>>y.unshift([-5,-4,-3],[-2,-1,0])
6
>>y
[[-5, -4, -3], [-2, -1, 0], [1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11,
12]]
```

El método "**slice(i,f)**" crea un nuevo vector con los elementos existentes entre el índice "i" y el índice "f" (sin incluir este último). Si no se especifica "f" el vector se crea con todos los elementos a partir de "i". Por ejemplo, dado el siguiente vector:

```
>>v=[1,2,3,4,5,6]
[1, 2, 3, 4, 5, 6]
```

La siguiente instrucción crea el vector "u" con los elementos de "v" que se encuentra entre la posición 2 (el segundo elemento) y 5 (el último elemento, sin incluir dicho elemento):

```
>>u=v.slice(1,5)
[2, 3, 4, 5]
```

La siguiente instrucción crea el vector "w" con los elementos comprendidos entre el elemento 3 (el cuarto) y el último:

```
>>w=v.slice(3)
[4, 5, 6]
```

Cuando el índice es negativo, se crea un vector con ese número de elementos extraídos contando desde el último, por ejemplo, la siguiente instrucción crea el vector "r" con los dos últimos elementos del vector "v":

```
>>r=v.slice(-2)
[5, 6]
```

El método "slice" no modifica la matriz original:

```
>>v
[1, 2, 3, 4, 5, 6]
```

El método "**splice(i,n,e1,e2,e3,…)**", remueve, inserta y/o reemplaza elementos en la matriz a partir del índice (posición) "i". Cuando se extraen elementos "n" especifica el número de elementos a extraer, los cuales son devueltos por el método. Cuando se insertan elementos (n=0) "e1,e2,e3,…" son los elementos a insertar. Cuando se reemplazan elementos (n igual al número de elementos a reemplazar) "e1,e2,e3",…", son los elementos que reemplazan a los "n" elementos que se extraen (los cuales son devueltos por el método).

Por ejemplo la siguiente instrucción inserta los números 8, 9, 10 y 11, en la quinta posición (índice 4) de la matriz "v":

```
>>v.splice(4,0,8,9,10,11)
[]
>>v
[1, 2, 3, 4, 8, 9, 10, 11, 5, 6]
```

Mientras que la siguiente remueve tres elementos a partir de la cuarta posición (los números 4, 8 y 9):

```
>>v.splice(3,3)
[4, 8, 9]
>>v
[1, 2, 3, 10, 11, 5, 6]
```

Mientras que la siguiente reemplaza los números 10 y 11 por 20 y 30:

```
>>v.splice(3,2,20,30)
[10, 11]
>>v
[1, 2, 3, 20, 30, 5, 6]
```

Al igual que "slice", si el índice es negativo las posiciones se cuentan comenzando del último elemento (que viene a ser el elemento -1) hacia atrás, por ejemplo, si el vector es:

```
>>v=[1,2,3,4,5,6,7,8,9,10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

La siguiente instrucción extrae los elementos 5, 6, 7 y 8

```
>>v.splice(-6,4)
[5, 6, 7, 8]
>>v
[1, 2, 3, 4, 9, 10]
```

Y la siguiente los vuelve a insertar:

```
>>v.splice(-2,0,5,6,7,8)
[]
>>v
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

El método "**sort(f)**" ordena los elementos del vector desde el cual se llama al método, el parámetro "f", que es una función, es opcional. Por defecto los elementos son ordenados ascendentemente tratándolos como texto. Por ejemplo, dado el siguiente vector:

```
>>a=["x","a","r","b","z","c"]
[x, a, r, b, z, c]
```

Al llamar al método "sort" se obtiene:

```
>>a.sort()
[a, b, c, r, x, z]
```

No obstante si el vector es numérico, como el siguiente:

```
>>b=[10,2,4,20,5,6,40,70,9]
[10, 2, 4, 20, 5, 6, 40, 70, 9]
```

Al llamar al método "sort" se obtiene:

```
>>b.sort()
[10, 2, 20, 4, 40, 5, 6, 70, 9]
```

Que no es el orden correcto. Esto se debe a que "sort", como se dijo, trata los elementos como texto (no como números). En estos casos, para obtener el orden correcto, se debe mandar también la función "f", normalmente como una función anónima que recibe dos parámetros y devuelve un valor positivo cuando el primer parámetro es mayor al segundo, cero si son iguales y negativo si el primer parámetro es menor que el segundo. Así en el caso de los números la función simplemente tiene que devolver la resta de los dos números, pues la resta es positiva (mayor a cero) cuando el primer número es mayor que el segundo, es cero si son iguales y es negativa si el primer número es menor al segundo:

```
>>b.sort(function(x,y){return x-y;})
[2, 4, 5, 6, 9, 10, 20, 40, 70]
```

Es importante tomar en cuenta que "sort" no devuelve un nuevo vector ordenado, sino que modifica el vector original:

```
>>b
[2, 4, 5, 6, 9, 10, 20, 40, 70]
```

Para ordenar el vector en orden inverso (descendente) simplemente se cambia el orden de la resta en la función:

```
>>b.sort(function(x,y){return y-x;})
[70, 40, 20, 10, 9, 6, 5, 4, 2]
```

Sin embargo si lo que se quiere es invertir el orden en que se encuentran los elementos (no ordenarlos), se debe emplear el método "**reverse()**", así para invertir los elementos del vector:

```
>>c=[8,1,20,4,8,54,12,9,20]
[8, 1, 20, 4, 8, 54, 12, 9, 20]
```

Se escribe:

```
>>c.reverse()
[20, 9, 12, 54, 8, 4, 20, 1, 8]
```

Al igual que "sort", "reverse" no devuelve un nuevo vector, sino que cambia el orden del vector original:

```
>>c
[20, 9, 12, 54, 8, 4, 20, 1, 8]
```

Finalmente algunos métodos y propiedades que están presentes en todos los objetos (no solo en los vectores) son: "**toString()**" convierte el objeto en texto. Este método es llamado automáticamente por Javascript cuando realiza operaciones donde es necesario convertir el objeto en texto, como cuando se muestra el objeto en pantalla. Este método ha sido modificado para el objeto Array en la librería "sis101.js" y por ello, cuando se emplea dicha librería, los vectores se muestran encerrados entre corchetes.

El método "**valueOf()**" devuelve el valor del objeto. Este método es llamado automáticamente por JavaScript cada vez que se requiere el valor de

un vector o matriz para realizar operaciones.

La propiedad "**constructor**" devuelve la función que se ha empleado para crear el objeto, así en el caso de un vector (o matriz) devuelve:

```
>>x=[1,2,3,4,5,6,7,8]
[1, 2, 3, 4, 5, 6, 7, 8]
>>x.constructor
function Array() { [native code] }
```

Es decir que la función empleada para crear un vector es "Array". Desde el punto de vista práctico esta propiedad permite averiguar el tipo de objeto con el que se está trabajando. Por ejemplo la siguiente instrucción devuelve "Es un array", porque "x" es un vector (es decir un array):

```
>>if (x.constructor==Array) "Es un Array"; else "No es un Array"
Es un Array
```

Por último la propiedad "**prototype**" hace posible añadir propiedades y/o métodos a los objetos. Más adelante se estudiará la aplicación de esta propiedad.

## 4.4. TRABAJO CON MATRICES

Una de las estructuras que más se emplea cuando se trabaja con matrices es la estructura "for", esto porque por lo general en dichas operaciones se conocen los límites y/o el número de veces que se debe repetir el proceso y, como se sabe, la estructura iterativa más adecuada para esos casos es la estructura "for".

Por ejemplo, dada la siguiente matriz:

```
>>x=[4,7,8,9,11,11,56,74,3,8,12,15]
[4, 7, 8, 9, 11, 11, 56, 74, 3, 8, 12, 15]
```

Para sumar sus elementos se puede escribir la siguiente instrucción:

```
>>s=0; for (var i=0;i<x.length;i++) s+=x[i]; s
218
```

O de forma más sencilla empleando la forma "for-in":

```
>>s=0; for (i in x) s+=x[i]; s
218
```

Lamentablemente, esta última forma no está correctamente implementada en todos los navegadores, sobre todo en los dispositivos móviles, donde suele generar información adicional. Por esta razón, cuando se quiere escribir código que sea compatible con la mayoría de los navegadores, es preferible emplear "for" en lugar de "for-in".

Se aplican procedimientos similares cuando se trabaja con matrices, así dada la matriz:

```
>>a=[[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7]]
[[1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6], [4, 5, 6, 7]]
```

La siguiente instrucción calcula la sumatoria de todos sus elementos:

```
>>s=0; for(var i=0;i<a.length;i++) for(var j=0;j<a[0].length;j++) s+=a[i][j]; s
64
```

Qué por supuesto puede ser calculada también con "for-in":

```
>>s=0; for(var i in a) for(var j in a[0]) s+=a[i][j]; s
64
```

## 4.5. FUNCIONES DE VALIDACIÓN

Como ya se vio en el anterior tema, una parte importante del código lo constituyen las instrucciones de validación. Puesto que la validación es un proceso imprescindible en toda aplicación práctica es conveniente crear algunas funciones con el fin de evitar la reescritura de código y disminuir al mismo tiempo la posibilidad de error.

En ese sentido se han creado algunas funciones (que han sido añadidas a la librería "sis101.js". En primer lugar, para determinar si un dato es o no un número real, entero o entero positivo (natural) se han creado las siguientes funciones:

```
//Función que devuelve verdadero si el dato "d" es un número real
//valido, falso en caso contrario
function isReal(d){
  if (isNaN(d)) return false;
  return true;
}

//Funcion que devuelve verdadero si el dato "d" es un número
//entero positivo o negativo.
function isInteger(d){
  if (isReal(d) && frac(d)==0) return true;
  return false;
}

//Función que devuelve verdadero si el dato "d" es un número
//entero positivo, falso en caso contrario
function isNatural(d){
  if (isInteger(d) && d>=0) return true;
  return false;
}
```

Observe que, siguiendo los principios de la programación estructurada (para reducir la escritura de código), la segunda función llama a la primera y la tercera a la segunda.

Una vez que se cuenta con estas funciones, para determinar, por ejemplo, si un número es natural simplemente se escribe algo así:

```
if (isNatural(x)) {...
```

Lo que reduce considerablemente el código necesario, no obstante, es recomendable tener en mente el cómo operan estas funciones para aplicarlas correctamente y/o para modificarlas en caso de ser necesario.

Por otra parte para leer el código de la tecla pulsada (incluso en Internet Explorer), se ha creado la siguiente función (añadida, igualmente, a la librería "sis101.js"):

```
//Devuelve el códito de la tecla pulsada cuando se ha disparado
//el evento onkeydown
function tecla(event) {
  return event.which || event.keyCode;
}
```

Para comprender por qué esta función devuelve el resultado correcto, es

necesario aclarar que en Javascript los operadores lógicos (como ||) no devuelven necesariamente un resultado lógico (true o false) sino el primer valor con el cual se puede determinar, sin lugar a dudas, el resultado de la expresión. Por ejemplo la siguiente instrucción devuelve 23:

```
>>23 || 7
23
```

Y es así porque 23 se interpreta como verdadero (pues es diferente de 0) y como el operador || devuelve verdadero si uno de los valores que compara es verdadero, no es necesario evaluar ya el otro valor. En cambio, la siguiente expresión devuelve 57:

```
>>0 || 57
57
```

Porque al ser el primer valor 0 (falso) se evalúa el segundo y como este es diferente de cero (verdadero) se devuelve dicho valor.

En la función "tecla" para todos los navegadores (con excepción de Internet Explorer), "event.which" devuelve un valor diferente de 0 y en consecuencia la función devuelve dicho valor, pero cuando el navegador es Internet Explorer, "event.which" devuelve "undefined" (falso) por lo que devuelve el valor de "event.keyCode".

Para determinar si una tecla es un carácter válido cuando se escribe un número, se han creado las siguientes funciones (añadidas también a la librería "sis101.js"):

```
//Función que devuelve verdadero si la tecla pulsada "key" es un
//carácter válido cuando se escribe un número entero positivo
function isNaturalKey(key){
  //Si la tecla pulsada está entre los caracteres 0 y 9
  //incluido el teclado numérico
  if ((key>=48 && key<=57) || (key>=96 && key<=105)) return true;
  //Si la tecla es retroceso (8), tab (9), enter (13), fin (35),
  //inicio (36), cursor izquierdo (37), cursor derecho (39) o
  //suprimir (46)
  if (key==8 || key==9 || key==13 || key==35 || key==36 ||
    key==37 || key==39 || key==46) return true;
  return false;
}

//Función que devuelve verdadero si la tecla pulsada "key" es un
//carácter válido cuando se escribe un número entero
function isIntegerKey(key){
  if (isNaturalKey(key)) return true;
  //Si la tecla es el signo menos "-"
  if (key==109 || key==189) return true;
  return false;
}

//Función que devuelve verdadero si la tecla pulsada "key" es un
//carácter válido cuando se escribe un número real
function isRealKey(key){
  if (isIntegerKey(key)) return true;
```

```
    //Si la tecla es la letra "e" (69), el signo menos "+" (107
    //o 189) o el punto (110 o 190)
    if (key==69 || key==107 || key==187 || key==110 || key==190)
      return true;
    return false;
}
```

Donde, una vez más (para no reescribir código), la segunda función llama a la primera y la tercera a la segunda.

Con estas funciones, para que por ejemplo un input text, sólo permita escribir números enteros, se puede escribir:

```
<input type="text" onkeydown="return isIntegerKey(tecla(event));">
```

## 4.6. GENERACIÓN DE DATOS ALEATORIOS

Cuando se trabaja con matrices es conveniente contar con funciones que permitan generar conjuntos de datos aleatorios, mismos que puedan ser empleados como valores de prueba de las funciones. El introducir manualmente decenas, centenas o miles de datos, para probar una función, no es una opción viable, más aún si se toma en cuenta que en dicho proceso muy probable la generación de errores, por lo que es frecuente que dichos datos deban ser introducidos no una, sino varias veces.

Para generar un número aleatorio se emplea la función "random()" (más propiamente "Math.random()") que, como se sabe, genera un número real aleatorio comprendido entre 0 y 1.

Si al número que devuelve "random" se le multiplica un número cualquiera "d", se obtiene un número aleatorio comprendido entre 0 y "d" (d*1=d). Ahora si a ese resultado se le suma un límite inferior "li", se obtiene un número aleatorio comprendido entre ese límite y "li+d". Así si "d" es 10 y "li" es 5, se obtiene un número aleatorio comprendido entre 5 y 15 (5+10).

Por lo tanto, para generar un número aleatorio cualquiera, comprendido entre dos límite: inferior "li" y superior "ls", se aplica la fórmula:

**N° aleatorio comprendido entre "li" y "ls"= random*(ls-li)+li**

Y para que dicho número sea entero (en lugar de un número real) simplemente se redondea el resultado obtenido (con "round").

Entonces, para generar una matriz con números aleatorios, se aplica la anterior fórmula para cada uno de sus elementos (recorriendo la matriz con la estructura "for").

Una de las funciones añadidas a la librería "sis101.js" para la generación de números es:

```
random(li,ls)
```

Sin argumentos, genera como de costumbre un número real comprendido entre 0 y 1. Con argumentos, genera un número real comprendido entre "li" y "ls". Por ejemplo la siguiente instrucción genera un número real comprendido entre -10 y 10:

```
>>random(-10,10)
9.050663751550019
```

Para que una función responda de una u otra forma, en función a los argumentos (datos) que recibe, se emplean los argumentos por defecto. Ja-

vascript no tiene argumentos por defecto como tales, sin embargo, cuenta con herramientas que permiten emularlos. Así si una función no recibe el número de argumentos que espera, los argumentos restantes quedan indefinidos y se puede averiguar ese hecho comparándolos con "null" (o "undifined").

Por ejemplo la siguiente función suma 3 números, pero si no se le manda el último toma por defecto el valor 3, si no se le mandan los dos últimos toman por defecto los valores 2 y 3, y si no se le manda ninguno de los números toman por defecto los valores 1, 2 y 3:

```
>>sum3=function(x1,x2,x3){ if (x1==null) x1=1; if (x2==null) x2=2; if
(x3==null) x3=3; return x1+x2+x3; }
function (x1,x2,x3){
  if (x1==null) x1=1;
  if (x2==null) x2=2;
  if (x3==null) x3=3;
  return x1+x2+x3;
}
```

Así al llamar a esta función sin valores se obtiene:

```
>>sum3()
6
```

Que es la suma de los valores por defecto. Si sólo se manda un valor, se suma ese valor y los otros valores por defecto:

```
>>sum3(5)
10
```

Si se mandan dos valores se suman dichos valores y el tercer valor por defecto:

```
>>sum3(5,6)
14
```

Y si se mandan todos los valores se suman dichos valores:

```
>>sum3(5,6,7)
18
```

Es posible también mandar algunos valores y dejar otros por defecto mandando "null" para los valores por defecto que se quieren conservar, así en la siguiente instrucción se mandan el primer y último valor, conservando el segundo por defecto:

```
>>sum3(10,null,20)
32
```

Otra forma (más flexible) de crear parámetros por defecto en Javascript es mediante la variable "arguments". Esta variable es creada automáticamente por Javascript cuando se llama a una función y es un vector que contiene los valores (argumentos) recibidos por la función. Así, con "arguments", se puede crear una función que sume no sólo 3, sino todos los números que se le manden:

```
>>sumn=function(){ var s=0; for (var i=0;i<arguments.length;i++)
s+=arguments[i]; return s; }
function (){
  var s=0;
  for (var i=0;i<arguments.length;i++) s+=arguments[i];
  return s;
}
```

Por ejemplo con las siguientes instrucciones se suman 3 y 10 números respectivamente:

```
>>sumn(1,2,3)
6
>>sumn(1,2,3,4,5,6,7,8,9,10)
55
```

Recuerde que en la calculadora Javascript para añadir un salto de línea, al escribir una instrucción, se deben pulsar las teclas "shift+enter" y que si se está editando una instrucción (si se está escribiendo o modificando una instrucción) se deben pulsar las teclas "ctrl+arriba" o "ctrl+abajo" para pasar a la instrucción anterior o posterior.

En las funciones añadidas a la librería (cuyo código debe ser estudiado como parte de los ejemplos) se han empleado ambas formas para crear parámetros por defecto, pero además se ha empleado una segunda forma de recursividad: "las llamadas circulares o recíprocas", en las que una función llama a otra y esta a su vez llama a la primera.

Las funciones añadidas la librería para la generación de números o conjuntos de números aleatorios (aparte de "random") son:

```
randomc(li,ls)
```

Sin argumentos genera un número complejo con las partes real e imaginaria comprendidas entre 0 y 1. Con argumentos genera un número complejo con las partes real e imaginaria comprendidas entre los límites dados:

```
>>randomc()
0.7189483027905226+0.40197217208333313i
>>randomc(5,10)
7.5963250431232l5+9.590400715824217i
```

La función "rand", genera un vector o una matriz:

```
rand(n,m,li,ls)
```

Con un argumento genera un vector con ese número de elementos aleatorios comprendidos entre 0 y 1:

```
>>rand(7)
[0.11070959037169814, 0.7048113474156708, 0.19553812872618437,
0.27860282524488866, 0.5007058342453092, 0.833899108460173,
0.5555433630943298]
```

Con dos argumentos genera una matriz con ese número de filas y columnas de elementos aleatorios comprendidos entre 0 y 1:

```
>>rand(3,4)
[[0.6238643436226994, 0.8650215738452971, 0.5996474497951567,
0.19420748576521873], [0.20076548820361495, 0.5637048308271915,
0.7273495462723076, 0.656704256311059], [0.1661558833438903,
0.32016177102923393, 0.799623871454969, 0.7091619167476892]]
```

Para ver en la calculadora cada fila de la matriz en una línea diferense te puede emplear la función "mshow":

```
>>mshow(ans)
[[0.6238643436226994, 0.8650215738452971, 0.5996474497951567,
0.19420748576521873]
 [0.20076548820361495, 0.5637048308271915, 0.7273495462723076,
0.656704256311059]
 [0.1661558833438903, 0.32016177102923393, 0.799623871454969,
```

```
0.7091619167476892]]
```

Con 3 argumentos genera un vector con el número de elementos dado (primer argumento) comprendidos entre los límites especificados (segundo y tercer argumento):

```
>>rand(5,10,15)
[12.867830061586574, 14.249553692061454, 14.613155242986977,
11.461944980546832, 12.407131311483681]
```

Y con 4 argumentos genera una matriz con el número de filas y columnas dado (primer y segundo argumentos) comprendidos ente los límites especificados (tercer y cuarto argumento):

```
>>mshow(rand(4,3,6,9))
[[7.639992176089436, 7.8939549929928035, 7.504247563658282]
 [8.765059168450534, 7.260654717218131, 8.729129368672147]
 [6.444568110862747, 8.527120805578306, 8.70455662929453]
 [7.612514588283375, 6.337499512592331, 8.408573352266103]]
```

Para generar vectores y matrices con números complejos se tiene la función "randc" que tiene los mismos parámetros que "rand" sólo que devuelve un vector (o matriz) de números complejos en lugar de reales:

```
randomc(n,m,li,ls)
```

En este tema se generan exclusivamente datos de tipo numérico. Otros tipos de datos serán generados en temas posteriores.

## 4.7. FUNCIONES DIVERSAS

La librería "sis101.js" cuenta con muchas otras funciones para el trabajo con matrices, algunas de las cuales son:

```
range(li,inc,ls)
```

Genera un vector con elementos comprendidos entre "li" y "ls", separados con el incremento "inc":

```
>>range(1,0.5,8)
[1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8]
```

Si no se especifica el incremento se emplea 1 por defecto:

```
>>range(1,10)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Si sólo se manda un número, se genera un vector con elementos comprendidos entre 0 y dicho número:

```
>>range(7)
[0, 1, 2, 3, 4, 5, 6, 7]
```

La función "linspace":

```
linspace(li,ls,n)
```

Genera un vector con "n" elementos, igualmente espaciados, comprendidos entre "li"  "ls":

```
>>linspace(1,5,10)
[1, 1.4444444444444444, 1.8888888888888888, 2.333333333333333,
2.7777777777777777, 3.2222222222222223, 3.666666666666667,
4.111111111111112, 4.555555555555556, 5.000000000000001]
```

Si no se especifica el número de elementos se genera un vector con 100 elementos (redondeados, en el siguiente ejemplo, al segundo decimal):

```
>>round(linspace(1,10),2)
[1, 1.09, 1.18, 1.27, 1.36, 1.45, 1.55, 1.64, 1.73, 1.82, 1.91, 2, 2.09,
2.18, 2.27, 2.36, 2.45, 2.55, 2.64, 2.73, 2.82, 2.91, 3, 3.09, 3.18, 3.27,
3.36, 3.45, 3.55, 3.64, 3.73, 3.82, 3.91, 4, 4.09, 4.18, 4.27, 4.36, 4.45,
4.55, 4.64, 4.73, 4.82, 4.91, 5, 5.09, 5.18, 5.27, 5.36, 5.45, 5.55, 5.64,
5.73, 5.82, 5.91, 6, 6.09, 6.18, 6.27, 6.36, 6.45, 6.55, 6.64, 6.73, 6.82,
6.91, 7, 7.09, 7.18, 7.27, 7.36, 7.45, 7.55, 7.64, 7.73, 7.82, 7.91, 8,
8.09, 8.18, 8.27, 8.36, 8.45, 8.55, 8.64, 8.73, 8.82, 8.91, 9, 9.09, 9.18,
9.27, 9.36, 9.45, 9.55, 9.64, 9.73, 9.82, 9.91, 10]
```

Para crear una matriz (o vector) con todos sus elementos iguales a cero, se cuenta con la fución "zeros":

```
zeros(n,m)
```

Así, la siguiente instrucción crea una matriz de ceros, con 3 filas y cuatro columnas:

```
>>zeros(3,4)
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Si sólo se manda un número, crea un vector con ese número de elementos:

```
>>zeros(5)
[0, 0, 0, 0, 0]
```

Si no se manda ningún dato, crea una matriz de ceros con tres filas y tres columnas:

```
>>zeros()
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Para crear una matriz (o vector) con todos sus elementos iguales a una constante dada, se cuenta con la función "mconst":

```
mconst(n,m,c)
```

Así la siguiente instrucción crea una matriz de 3 filas y 4 columnas, con todos sus elementos iguales a 5:

```
>>mconst(3,4,5)
[[5, 5, 5, 5], [5, 5, 5, 5], [5, 5, 5, 5]]
```

Si sólo se mandan dos datos, se crea un vector con el número de elementos especificado (primer argumento) iguales a la constante dada (segundo argumento):

```
>>mconst(5,1)
[1, 1, 1, 1, 1]
```

Como ya se ha explicado previamente, cuando se trabaja con matrices, si se asigna una variable a otra, sólo se asigna la dirección de memoria donde se encuentra la matriz, no la matriz en sí. Para realizar una copia real de la matriz, se cuenta con la función "copy":

```
y = copy(x)
```

Que hace una copia de los datos a los que apunta "x" en la variable "y". El dato a copiar ("x") puede ser un número real, un número complejo, un vector o una matriz.

Por ejemplo, dada la siguiente matriz:

```
>>a=mconst(3,4,7)
[[7, 7, 7, 7], [7, 7, 7, 7], [7, 7, 7, 7]]
```

Con la siguiente instrucción se hace una copia de dichos elementos en la variable "b":

```
>>b=copy(a)
[[7, 7, 7, 7], [7, 7, 7, 7], [7, 7, 7, 7]]
```

Se puede comprobar que es una copia de la matriz, y no sólo de su dirección de memoria, cambiando el valor de alguno de sus elementos:

```
>>b[2][1]=10; b
[[7, 7, 7, 7], [7, 7, 7, 7], [7, 10, 7, 7]]
```

Ahora si se ven los elementos de la matriz original "a", se puede comprobar que los mismos no han sido modificados:

```
>>a
[[7, 7, 7, 7], [7, 7, 7, 7], [7, 7, 7, 7]]
```

Para determinar si un valor dado es o no un vector se cuenta con la función "isArray"

```
isArray(a)
```

Que devuelve verdadero si "a" es un vector (un Array) y falso en caso contrario. Igualmente se cuenta con las funciones "isMatrix":

```
isMatrix(a)
```

Que devuelve verdadero si "a" es una matriz y la función "isMatrixr":

```
isMatrixr(a)
```

Que devuelve verdadero si "a" es una matriz regular (con igual número de columnas en todas sus filas).

Otras funciones disponibles en la librería se las irá presentando y empleando a medida que se requieran.

## 4.8. EJEMPLOS

1. **Cree una página HTML que dado un vector de números reales o enteros, calcule la media aritmética de los mismos. La aplicación debe dar la opción de introducir el vector manualmente o de generarlo automáticamente y de ser así generar números enteros comprendidos entre 1 y 100.**

   La media aritmética se calcula con la expresión:

$$\bar{x} = \frac{\sum_{i=1}^{i=n} x_i}{n}$$

   Donde "n" es el número de elementos a promediar. La codificación de esta expresión es muy sencilla: en un ciclo "for" que va desde 1 hasta "n" se añade (en cada repetición del ciclo) el valor del elemento "$x_i$" a un contador, luego se divide el acumulador entre el número de elementos.

   En cuanto a la interfaz, lo ideal sería introducir y mostrar los datos en tablas, por el momento, sin embargo, los datos se introducirán y mostrarán en cuadros de texto "textarea", que son similares a los "input text" excepto que permite escribir y mostrar varias líneas en lugar de una sola. La presentación e introducción de datos en tablas (que requiere algo más de programación) se verá posteriormente.

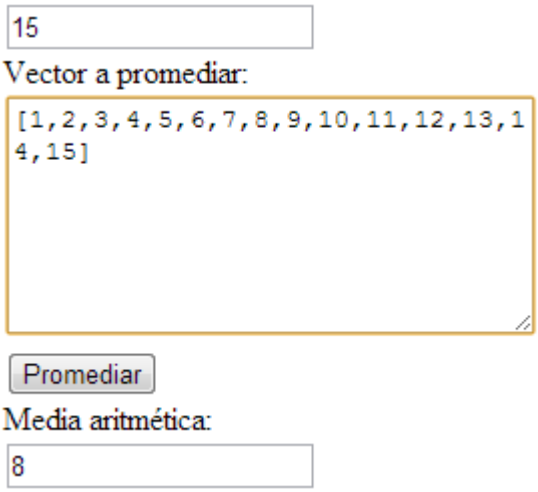
En la interfaz se empleará entonces un "textarea" para introducir (o mostrar) el vector con los elementos a promediar. Para presentar el resultado se empleará un "input text" de sólo lectura, el cálculo se llevará a cabo al hacer clic en un botón y para permitir generar el vector (en lugar de escribirlo) se empleará otro "input text" en el que se podrá introducir el número de elementos, generándose un vector con dicho número de elementos al pulsar la tecla "enter".

El código HTML que resuelve el problema es el siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ejemplo 1</title>
  <script src="sis101.js"></script>
  <script>

    //Cálculo de la media aritmética
    function media(x){
      var s=0,n=x.length;
      for (var i=0;i<n;i++) s+=x[i];
      return s/n;
    }

    //Calcula y muestra el promedio
    function promediar(){
      //Se lee el texto escrito en el "textarea" y se convierte en
      //vector con la función "eval"
      var x=eval(document.getElementById("vector").value);
      //Si el vector ha sido bien escrito, se calcula y muestra el
      //promedio, se actualiza el "input text" numElementos y se pasa
      //el foco al "input text" numElementos
      if (isArray(x)) {
         document.getElementById("resultado").value=media(x);
         document.getElementById("numElementos").value=x.length;
         document.getElementById("numElementos").focus();
      } else {
        alert("El vector está mal escrito");
        document.getElementById("vector").focus();
        document.getElementById("vector").select();
      }
    }

    //Validación del número de elementos a generar
    function validar1(event){
      var key=tecla(event);
      //Si la tecla es "enter" y el número es correcto, se genera y
      //muestra el vector y se pasa el foco al botón calcular
      if (key==13) {
         var n=document.getElementById("numElementos").value;
        if (isNatural(n)) {
          document.getElementById("vector").value=round(rand(n,1,100));
          document.getElementById("calcular").focus();
          return true;
        } else {
          alert("El número debe ser entero positivo.");
          document.getElementById("numElementos").focus();
          document.getElementById("numElementos").select();
```

```
        }
      }
      //si no, se comprueba que sea una tecla válida
      return isNaturalKey(key);
    }

    //Validación del vector a promediar
    function validar2(event){
      key=tecla(event);
      //si la tecla es "enter" se calcula la media aritmética
      if (key==13){
        document.getElementById("calcular").onclick();
        return false;
      }
      //Si no, se verifica que sea: " " (32), cursor arriba (38),
      //cursor abajo (40), "[" (186), "]" (187) o "," (188).
      if (key==32 || key==38 || key==40 || key==186 || key==187 ||
          key==188) return true;
      //o una tecla válida para un número real
      return isRealKey(key);
    }

    //Selecciona el "input text" numElementos al cargar la página
    function iniciar(){
      document.getElementById("numElementos").focus();
      document.getElementById("numElementos").select();
    }
  </script>
</head>
<body onload="iniciar()">
  <h3>Media Aritmética</h3>

  <!--input text para mostrar o introducir el número de elementos-->
  <label for="numel">Número de elementos:</label><br>
  <input type="text" id="numElementos" value="10" onkeydown=
    "return validar1(event)"><br>

  <!--textarea para mostrar o introducir el vector-->
  <label for="vector">Vector a promediar:</label><br>
  <textarea id="vector" rows="7" cols="30" onkeydown=
    "return validar2(event)"></textarea><br>

  <!--botón para calcular el promedio-->
  <button id="calcular" onclick="promediar()" >Promediar</button><br>

  <!--input text para mostrar el resultado calculado-->
  <label for="resultado">Media aritmética:</label><br>
  <input type="text" id="resultado" readonly><br>

</body>
</html>
```

Abriendo la página en un navegador se obtiene algo parecido a:




Como se puede ver en este ejemplo, se han fijado las dimensiones del "textarea" en 7 filas de alto y 30 columnas de ancho (valores que, por supuesto, pueden ser cambiados).

Por otra parte y como está explicado en el código, se ha empleado la función "eval" para convertir el texto introducido en el "textarea" en un vector. La función "eval" trata el texto que se le manda como instrucciones Javascript, es la función que se emplea en la calculadora Javascript para evaluar las instrucciones que se escriben en la misma, por lo tanto, en este ejemplo "eval", al recibir el texto, en forma de vector, lo trata como tal, con lo que se consigue la conversión (siempre y cuando el vector haya sido correctamente escrito).

Otro aspecto que se debe tomar en cuenta es que los código (números) empleados para las teclas "[", "]" y "," no son los mismos en todos los teclados, pues depende de la configuración del mismo, en este caso específico corresponden a un teclado en español. Este problema puede ser resuelto con el evento "onkeypress", aunque en ese caso se deben tomar en cuenta otros aspectos, como se verá posteriormente.

**2. Elabore una página HTML que dadas dos matrices de números reales o enteros, calcule el producto de los mismos. La aplicación debe dar la opción de escribir las matrices manualmente o de generarlas con números enteros comprendidos entre 1 y 10.**

Como se sabe en la multiplicación de matrices se multiplican las filas de la primera matriz (a) con las columnas de la segunda (b), realizando la sumatoria respectiva. La ecuación para calcular los elementos de la matriz resultante "c" es la siguiente:

$$c_{i,j} = \sum_{k=1}^{k=nca} a_{i,k} b_{k,j} \quad \begin{cases} i = 1 \rightarrow nfa \\ j = 1 \rightarrow ncb \end{cases}$$

En esencia, para resolver el problema se debe programar esta ecuación y puesto que se trata de una sumatoria (con límites definidos) la estructura más adecuada para este fin es "for". Se requieren tres estructuras for, una donde "i" va desde 1 hasta el número de filas de la matriz "a" (nfa), otro (dentro del anterior) que va desde "j" igual a 1 hasta el número de columnas de "b" (ncb) y un tercero (dentro del último ciclo "for") que va desde "k" igual a 1 hasta el número de columnas de "a" (nca).

En el último ciclo, puesto que se trata de una sumatoria, se deben inicializar los elementos $c_{ij}$ en cero (o emplear una variable auxiliar) para

que almacene el valor de la sumatoria.

Por otra parte, es necesario verificar que el número de columnas de la matriz "a" (nca) sea igual al número de filas de la matriz "b" (nfb), pues de lo contrario las matrices no podrían ser multiplicadas.

En cuanto a la interfaz de usuario, se emplearán cuatro "input text" para introducir las dimensiones de las matrices, tres "textbox", dos para mostrar o escribir las matrices y uno (de solo lectura) para mostrar la matriz resultante. Para generar las matrices y para calcular la multiplicación se emplearán 3 botones.

El código de la página HTML que resuelve el problema (explicado mediante comentarios) es:

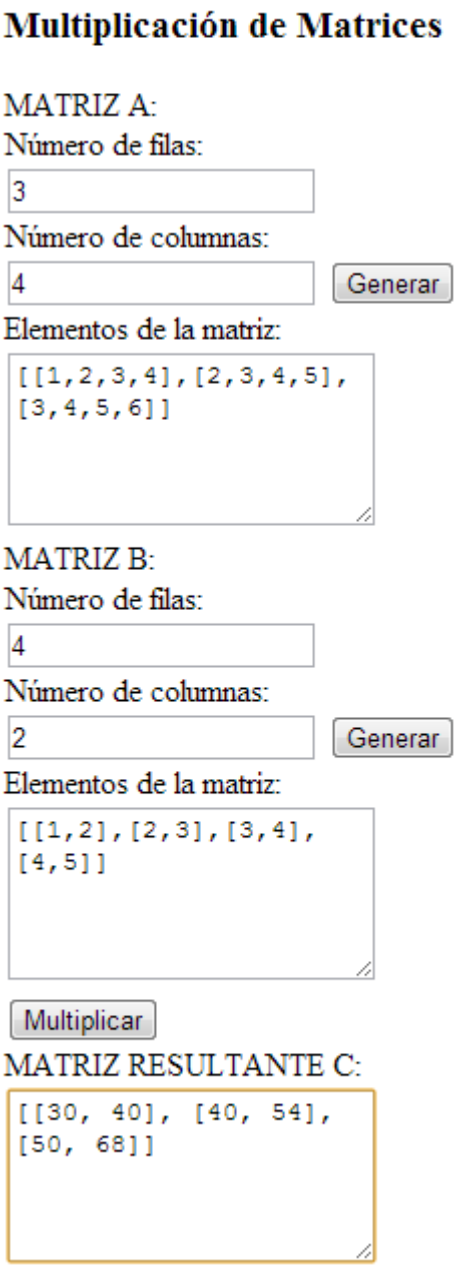
```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ejemplo 2</title>
  <script src="sis101.js"></script>
  <script>

    //Multiplicación de matrices
    function mulmat(a,b){
      var nfa=a.length, nca=a[0].length;
      var nfb=b.length, ncb=b[0].length;
      //Si las matrices no son compatibles se genera un error
      if (nca!=nfb)
        throw new Error("mulmat: matrices no compatibles.");
      var c=new Array(nfa);
      for (var i=0;i<nfa;i++){
        c[i]=new Array(ncb);
        for (var j=0;j<ncb;j++){
          c[i][j]=0;
          for (var k=0;k<nca;k++) c[i][j]+=a[i][k]*b[k][j];
        }
      }
      return c;
    }

    //Cálculo y presentación de la matriz
    function calcular(){
      //Si los elementos de la matriz A no conforman una matriz
      //regular se devuelve el foco al textarea matriza
      var ma = document.getElementById("matriza");
      var a=eval(ma.value);
      if (!isMatrixr(a)){
        alert("A no es una matriz regular");
        ma.focus();
      }
      //Si los elementos de la matriz B no conforman una matriz
      //regular se devuelve el foco al textarea matrizb
      var mb=document.getElementById("matrizb");
      var b=eval(mb.value);
      if (!isMatrix(b)){
        alert("B no es una matriz regular");
        mb.focus();
      }
      //Se multiplican las matrices atrapando el error en caso de
```

```
    //producirse (matrices incompatibles).
    try {
      document.getElementById("matrizc").value=mulmat(a,b);
      document.getElementById("matrizc").focus();
    } catch(e){
      alert(e.message);
      document.getElementById("nfa").focus();
    }
  }

  //Generación de los elementos de la matriz a
  function generarMatrizA(){
    var nfa=parseInt(document.getElementById("nfa").value);
    var nca=parseInt(document.getElementById("nca").value);
    try {
      document.getElementById("matriza").value=round(rand(nfa,nca,1,
          10));
      //si la matriz es generada, se pasa el foco al textarea matriza
      document.getElementById("matriza").focus();
    } catch(e) {
      //si no, se muestra el error y el foco pasa al input text nfa
      alert(e.message);
      document.getElementById("nfa").focus();
    }
  }

  //Generación de los elementos de la matriz b
  function generarMatrizB(){
    var nfb=parseInt(document.getElementById("nfb").value);
    var ncb=parseInt(document.getElementById("ncb").value);
    try {
      document.getElementById("matrizb").value=round(rand(nfb,ncb,1,
          10));
      //si la matriz es generada, se pasa el foco al textarea matrizb
      document.getElementById("matrizb").focus();
    } catch(e) {
      //si no, se muestra el error y el foco pasa al input text nfb
      alert(e.message);
      document.getElementById("nfb").focus();
    }
  }

  //Validación de las matrices A y B
  function vmatriz(event){
    key=tecla(event);
    //Se verifica que la tecla sea: " " (32), cursor arriba (38),
    //cursor abajo (40), "[" (186), "]" (187) o "," (188).
    if (key==32 || key==38 || key==40 || key==186 || key==187 ||
        key==188) return true;
    //o una tecla válida para un número real
    return isRealKey(key);
  }

  //Actualiza el número de filas y columnas de la matriz A en
  //función a los datos del textarea matriza
  function actualizarfca(){
    var ma=eval(document.getElementById("matriza").value);
    if (isMatrixr(ma)){
```

```
          document.getElementById("nfa").value=ma.length;
          document.getElementById("nca").value=ma[0].length;
        }
      }

      //Actualiza el número de filas y columnas de la matriz B en
      //función a los datos del textarea matrizb
      function actualizarfcb(){
        var mb=eval(document.getElementById("matrizb").value);
        if (isMatrixr(mb)){
          document.getElementById("nfb").value=mb.length;
          document.getElementById("ncb").value=mb[0].length;
        }
      }

      function iniciar(){
        document.getElementById("nfa").focus();
        document.getElementById("nfa").select();
      }

    </script>
  </head>
  <body onload="iniciar()">
    <h3>Multiplicación de Matrices</h3>

    <!-- Dimensiones y elementos de de la matriz a -->
    MATRIZ A:<br>
    <label for="nfa">Número de filas:</label><br>
    <input type="text" id="nfa" value="3" onkeydown=
      "return isNaturalKey(tecla(event))"><br>
    <label for="nca">Número de columnas:</label><br>
    <input type="text" id="nca" value="4" onkeydown=
      "return isNaturalKey(tecla(event))">
    <button id="bmatriza" onclick="generarMatrizA()">Generar</button><br>
    <label for="matriza">Elementos de la matriz:</label><br>
    <textarea id="matriza" rows="5" cols="20" onkeydown=
      "return vmatriz(event)" onblur="actualizarfca()"></textarea><br>

    <!-- Dimensiones y elementos de la matriz b -->
    MATRIZ B:<br>
    <label for="nfb">Número de filas:</label><br>
    <input type="text" id="nfb" value="4" onkeydown=
      "return isNaturalKey(tecla(event))"><br>
    <label for="ncb">Número de columnas:</label><br>
    <input type="text" id="ncb" value="3" onkeydown=
      "return isNaturalKey(tecla(event))">
    <button id="bmatrizb" onclick="generarMatrizB()">Generar</button><br>
    <label for="matrizb">Elementos de la matriz:</label><br>
    <textarea id="matrizb" rows="5" cols="20" onkeydown=
      "return vmatriz(event)" onblur="actualizarfcb()"></textarea><br>

    <!-- Botón que activa el cálculo de la matriz resultante -->
    <button id="multiplicar" onclick="calcular()">Multiplicar</button><br>

    <!-- Elementos de la matriz resultante -->
    MATRIZ RESULTANTE C:<br>
    <textarea id="matrizc" rows="5" cols="20" readonly></textarea><br>
```

```
</body>
</html>
```

Abriendo la página en un navegador se obtiene algo parecido a lo siguiente:




Como se puede ver en este ejemplo, los dos primeros "textarea" responden a dos eventos, el evento "onkeydown", para permitir escribir sólo los caracteres válidos y el evento "onblur" para actualizar los números de fila y columna cuando las matrices se escriben (como ocurre en el ejemplo) en lugar de generarlas.

3. **Elabore una página HTML que dada una matriz, permita intercambiar los elementos de dos columnas. La aplicación debe dar la opción de escribir la matriz manualmente o generarla con números reales comprendidos entre -10 y 10 y redondeados al primer dígito.**

Para intercambiar los elementos de dos columnas simplemente se recorre la matriz fila por fila (con un ciclo "for") y en cada repetición del ciclo se intercambian los elementos de ambas columnas.

En cuanto a la interfaz de usuario, se emplearán dos "input text" (con la validación respectiva) para introducir el número de filas y columnas, un "textarea" (con la validación respectiva) para mostrar o introducir los elementos de la matriz, un botón para generar la matriz, otro para intercambiar las columnas y dos "select" (inicialmente en vacíos) para seleccionar las columnas a intercambiar.

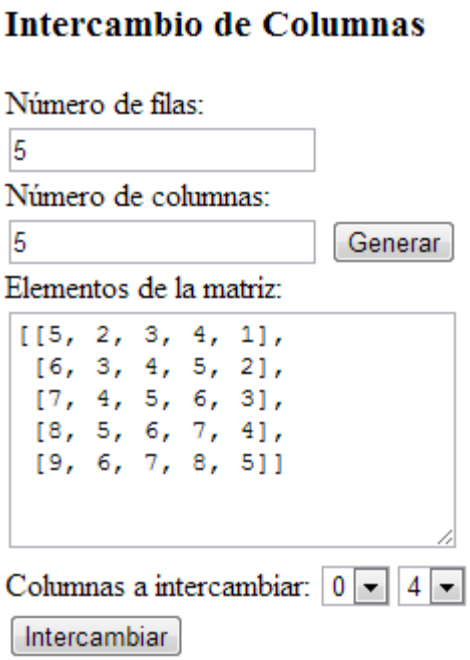
El código de la página HTML que resuelve el problema es:

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ejemplo 3</title>
  <script src="sis101.js"></script>
  <script>

    //Intercambia las columas c1 y c2 de la matriz a
    function intCol(a,c1,c2){
      //Si la matriz no es regular se genera un error
      if (!isMatrixr(a))
        throw new Error("intCol: No es una matriz regular.");
      //Si los números de columna no son enterpos positivos se
      //genera un error
      if (!isNatural(c1) || !isNatural(c2)) throw new Error(
          "intCol: Los números de columna deben ser enteros.");
      //Si las columnas a intercambiar son las mismas, no se hace nada
      if (c1==c2) return;
      var aux;
      for (var i=0;i<a.length;i++){
        aux=a[i][c1]; a[i][c1]=a[i][c2]; a[i][c2]=aux;
      }
    }

    //Intercambia y muestra las columnas intercambiadas
    function intercambiar(){
      try {
        var c1=parseInt(document.getElementById("c1").value);
        var c2=parseInt(document.getElementById("c2").value);
        var a=eval(document.getElementById("matriz").value);
        intCol(a,c1,c2);
        document.getElementById("matriz").value=mshow(a);
      } catch(e) {
        alert(e.message);
        document.getElementById("nf").focus();
      }
    }

    //Validación de la escritura de elementos en la matriz
    function vMatriz(event){
      key=tecla(event);
      //Se verifica que la tecla sea: " " (32), cursor arriba (38),
      //cursor abajo (40), "[" (186) "]" (187), "," (188)
      if (key==32 || key==38 || key==40 || key==186 || key==187 ||
          key==188) return true;
      //o si es una tecla válida para un número real
```

```
      return isRealKey(key);
    }

    //Genera la matriz con el número de filas y columnas introducido
    function generarMatriz(){
      try {
        var nf=parseInt(document.getElementById("nf").value);
        var nc=parseInt(document.getElementById("nc").value);
        document.getElementById("matriz").value=
          mshow(round(rand(nf,nc,-10,10),1));
        document.getElementById("matriz").focus();
      } catch(e) {
        alert(e.message);
        document.getElementById("nf").focus();
      }
    }

    //Llena los select c1 y c2 con las columnas disponibles
    function llenarSelects(){
      var n=parseInt(document.getElementById("nc").value);
      var t="";
      for (var i=0;i<n;i++) t+="<option>"+i+"</option>"
      document.getElementById("c1").innerHTML=t;
      document.getElementById("c2").innerHTML=t;
    }

    //Actualiza el número de filas y columnas en función a la matriz
    //introducida
    function actualizarfc(){
      var a=eval(document.getElementById("matriz").value);
      if (!isMatrixr(a)){
        alert("actualizarfc: No es una matriz regular.");
        document.getElementById("matriz").focus();
      }
      //Si es una matriz regular se actualiza el número de filas y
      //columnas y las opciones de los selects
      document.getElementById("nf").value=a.length;
      document.getElementById("nc").value=a[0].length;
      llenarSelects();
    }

    //Cuando la página carga, selecciona el input "nf"
    function iniciar(){
      document.getElementById("nf").focus();
      document.getElementById("nf").select();
    }

  </script>
</head>
<body onload="iniciar()">
  <h3>Intercambio de Columnas</h3>

  <!-- Input para el número de filas de la matriz-->
  <label for="nf">Número de filas:</label><br>
  <input type="text" id="nf" value="3" onkeydown=
    "return isNaturalKey(tecla(event))"><br>

  <!-- Input para el número de columnas de la matriz -->
```

```
    <label for="nc">Número de columnas:</label><br>
    <input type="text" id="nc" value="4" onkeydown=
      "return isNaturalKey(tecla(event))" onblur="llenarSelects()">

    <!-- Botón para generar los elementos de la matriz -->
    <button id="bgenerar" onclick="generarMatriz()">Generar</button><br>

    <!-- Textarea para mostrar o escribir los elementos de la matriz-->
    <label for="matriz">Elementos de la matriz:</label><br>
    <textarea id="matriz" onkeydown="return vMatriz(event)" rows="7"
     cols="25" onblur="actualizarfc()"></textarea><br>

    <!-- Selects para las columnas a intercambiar -->
    <label for="c1">Columnas a intercambiar:</label>
    <select id="c1"></select><select id="c2"></select><br>

    <!-- Botón para intercambiar las columnas -->
    <button id="bintercambiar" onclick="intercambiar()">Intercambiar
    </button>


</body>
</html>
```

Abriendo la página en un navegador se obtiene algo parecido a:




Como se puede ver, para mostrar las filas de la matriz en líneas separadas se ha empleado la función "mshow".

Los elementos "select" (para elegir los números de columnas a intercambiar) están inicialmente vacíos (no tiene opciones: <option></option>). Dichas opciones son añadidas en tiempo de ejecución mediante la función "llenarSelects". En esta función se lee el número de columnas de la matriz y mediante un ciclo "for" que va desde 0 hasta el número de columnas menos uno, se añade al texto "t", en cada repetición del ciclo, una opción, con el número de columna respectivo. Luego este texto (con las opciones añadidas) se asigna a la propiedad "innerHTML" del "textarea". La

propiedad "innerHTML" es el código HTML que normalmente se escribe cuando se crea el elemento directamente en la página, con la ventaja de que al ser creada desde Javascript, se pueden automatizar algunos procesos (como ocurre en este ejemplo).

Tome en cuenta que en este caso no es posible crear los "select" con el número de opciones correctas pues el número de columnas cambia cada vez que se cambia el número de columnas de la matriz.

## 4.9. EJERCICIOS

Para presentar los ejemplos y ejercicios del tema, debe crear una carpeta con el nombre "tema4" y guardar en la misma todos los archivos HTML creados (ejemplos y ejercicios).

1. Cree una página HTML que dado un vector de números reales o enteros, calcule la desviación estándar de los mismos, empleando la ecuación que se muestra al pie de la pregunta. La aplicación debe dar la opción de introducir el vector manualmente o de generarlo automáticamente y de ser así generar números enteros comprendidos entre -25 y 25.

$$d_{std} = \sqrt{\frac{\sum_{i=1}^{i=n}(\bar{x} - x_i)^2}{n}}$$

$$\bar{x} = \frac{\sum_{i=1}^{i=n} x_i}{n}$$

2. Elabore una página HTML que dada una matriz de números reales o enteros, calcule la transpuesta de la misma. La aplicación debe dar la opción de escribir la matriz manualmente o de generarlas con números enteros comprendidos entre 10 y 50.

3. Elabore una página HTML que dada una matriz de números reales o enteros, permita intercambiar los elementos de dos filas. La aplicación debe dar la opción de escribir la matriz manualmente o generarla con números reales comprendidos entre 1 y 9 y redondeados al segundo dígito.

# 5. CADENAS (STRINGS)

Las cadenas de caracteres o simplemente "cadenas" o "strings" (por su denominación en inglés), básicamente son vectores especializados en el manejo de caracteres, es decir en el manejo de texto, razón por la cual también se las conoce como datos de tipo texto.

Este tipo de dato está presente en todos los lenguajes de programación porque el manejo de texto es crucial en toda aplicación práctica. La mayor parte de la información que una aplicación presenta a o recibe del usuario está en forma de texto (inclusive los números).

Ya en temas anteriores se ha empleado cadenas (texto) y como se ha visto, para crear una variable de tipo cadena sólo se debe encerrar la información entre comillas o entre apóstrofos, por ejemplo las siguientes instrucciones crean dos variables de tipo cadena:

```
>>a="Programación en Javascript";
Programación en Javascript
>>b='Metodología de la Programación I';
Metodología de la Programación I
```

En Javascript las cadenas, al igual que los vectores, son objetos, por lo que formalmente, una variable de tipo cadena (de tipo texto) se crea de con la instrucción "new":

```
>>c=new String("Una cadena en Javascript");
Una cadena en Javascript
>>d=new String('Es un objeto');
Es un objeto
```

Por supuesto es más sencillo crearla directamente escribiendo el texto entre comillas o apóstrofos, por lo que esa es la forma más empleada (y con mucho) en la práctica.

## 5.1. PROPIEDADES, MÉTODOS Y OPERACIONES CON CADENAS

Al ser las cadenas un tipo de dato especializado en el manejo de matrices, tiene propiedades y métodos específicos para el manejo de matrices, además, responde al operador de suma "+", concatenando (uniendo) las cadenas, por ejemplo, si se suman las variables "a" y "b" (creadas en el acápite anterior) se obtiene:

```
>>a+b
Programación en JavascriptMetodología de la Programación I
```

Esta característica ha sido empleada ya en temas anteriores cuando se ha concatenado (sumado) texto para mostrar resultados y/o crear instrucciones HTML. Como ya se ha visto, cuando en Javascript se suma una cadena a un valor numérico (o de otro tipo), dicho valore es convertido previamente en una cadena y el resultado sumado a la otra cadena para formar la cadena resultante, por ejemplo si se tiene la siguiente variable:

```
>>x=2.34
2.34
```

Se puede crear un mensaje de texto mostrando la raíz cuadrada de ese número:

```
>>"La raíz cuadrada de "+x+" es: "+sqrt(x)
La raíz cuadrada de 2.34 es: 1.5297058540778354
```

Como se puede ver, las conversiones necesarias son hechas automáticamente por Javascript para dar un resultado de tipo texto. Se puede averiguar el tipo de dato de un valor dado con la función "typeof", así aplicando esta función al anterior resultado se obtiene:

```
>>typeof(ans)
string
```

Corroborándose así que el resultado es de tipo "string" (cadena).

Aparte del operador de suma, que es de mucha utilidad práctica, el objeto "string" cuenta con varios métodos y algunas propiedades, siendo la propiedad más utilizada "length" que, como en un vector, devuelve el número de caracteres que tiene la cadena, así, si la cadena es:

```
>>s="Hola Mundo!";
Hola Mundo!
```

La propiedad "length" devuelve 11:

```
>>s.length
11
```

El método "concat(cadena 1, cadena 2, …)" une la cadena de caracteres desde la cual se llama al método, a otras cadenas, de forma similar al operador "+". Así si se tienen las siguientes variables:

```
>>s1="Bienvenido a"
Bienvenido a
>>s2=" Javascript "
Javascript
>>s3=" un lenguaje interpretado "
un lenguaje interpretado
```

Se pueden concatenar las mismas y una cadena adicional, con la siguiente instrucción:

```
>>s1.concat(s2,s3," y ubicuo")
Bienvenido a Javascript un lenguaje interpretado y ubicuo
```

Este método crea una nueva cadena con el resultado, pero no modifica la cadena original, por lo que el valor de "s1" sigue siendo:

```
>>s1
Bienvenido a
```

El método "indexOf(cadena, índice_inicial)" devuelve la posición (el índice) del primer carácter que coincide con la "cadena", así, si la variable es:

```
>>s="Las cadenas son vectores especializados en caracteres"
Las cadenas son vectores especializados en caracteres
```

La siguiente instrucción devuelve 16:

```
>>s.indexOf("vectores")
16
```

Porque 16 es el índice (posición 17) del primer carácter de la palabra "vectores" dentro la cadena, mientras que la siguiente instrucción devuelve 4:

```
>>s.indexOf("ca")
4
```

Porque 4 es el índice de la primera aparición de "ca" dentro de la cadena. Para encontrar el índice de la segunda aparición de "ca" en la cadena, se debe comenzar a buscar después del índice 4, es decir desde el índice 5:

```
>>s.indexOf("ca",5)
43
```

Para encontrar el índice comenzando a buscar desde el último carácter (en lugar del primero) se emplea el método "lastIndexOf(cadena,índice_inicial)", así para encontrar la última aparición de "ca" en la variable "s" se escribe:

```
>>s.lastIndexOf("ca")
43
```

Y para encontrar la aparición anterior de "ca", se inicia la búsqueda en el índice anterior al encontrado, es decir:

```
>>s.lastIndexOf("ca",42)
4
```

Para obtener el carácter que se encuentra en un determinado índice se emplea la función "charAt(índice)", por ejemplo, la siguiente instrucción devuelve el carácter que se encuentra en el índice 12 (posición 13) de la variable "s":

```
>>s.charAt(12)
s
```

Si lo que se quiere es el código ASCII del carácter que se encuentra en un determinado índice (en lugar del carácter en sí) se emplea el método "charCodeAt(índice)", así el código del carácter en el índice 12 (el código de la letra "s") es:

```
>>s.charCodeAt(12)
115
```

Si se quiere obtener el carácter correspondiente a un código ASCII dado se puede emplear el método "String.fromCharCode(código)", así por ejemplo para obtener el carácter correspondiente al código ASCII 65 (la letra "A") se escribe:

```
>>String.fromCharCode(65)
A
```

Este método puede devolver una cadena conformada con los caracteres equivalentes a la lista de códigos que se le manda, por ejemplo, la siguiente instrucción devuelve "HOLA":

```
>>String.fromCharCode(72,79,76,65)
HOLA
```

Para copiar una parte de una cadena (una subcadena) se puede emplear el método "substr(inicio,longitud)", que copia "longitud" caracteres de la cadena, comenzando en el índice "inicio", por ejemplo, si la cadena es:

```
>>b='Los métodos son las funciones de un objeto'
Los métodos son las funciones de un objeto
```

Para copiar la palabra funciones se escribe:

```
>>b.substr(20,9)
funciones
```

Por supuesto se puede emplear el método "indexOf" para obtener el índice de la palabra a extraer y la propiedad "length" para su longitud:

```
>>v='funciones'
funciones
>>b.substr(b.indexOf(v),v.length)
funciones
```

Este método no modifica la cadena original, por lo que el valor de "b" sigue siendo:

```
>>b
Los métodos son las funciones de un objeto
```

Alternativamente, para copiar una subcadena se puede emplear el método "substring(inicio,fin)", que copia la cadena de caracteres comprendida entre el índice "inicio" y el índice "fin-1", por ejemplo para copiar la palabra "funciones" con este método, se escribe:

```
>>b.substring(20,29)
funciones
```

Si no se especifica el índice "fin", se copian todos los caracteres que existen desde el índice "inicio" hasta el final del texto (lo mismo ocurre con "substr" si no se especifica la longitud). Por ejemplo las siguientes instrucciones copian todos los caracteres que existen desde el índice 20 (la letra "f") hasta el final:

```
>>b.substring(20)
funciones de un objeto
>>b.substr(20)
funciones de un objeto
```

Al igual que "substr", este método no modifica la cadena original.

Un método similar a "substring" es "slice(inicio,fin)" que igualmente extrae la cadena de caracteres comprendidas entre el índice "inicio" y el índice "fin-1", básicamente la única diferencia con relación a "substring" es que si a "slice" se le pasa un número negativo, copia ese número de elementos comenzando desde el último carácter de la cadena, así para copiar la palabra "objeto" se escribe:

```
>>b.slice(-6)
objeto
```

Para dividir una cadena en un vector de subcadenas, se emplea el método "split(separador,límite)" (que viene a ser el complemento del método "join" de vectores). Este método divide la cadena en todos los lugares donde encuentra la cadena "separador", devolviendo un vector con estas subcadenas. Si se especifica el "límite" el método sólo devuelve ese número de elementos. Por ejemplo para obtener un vector conformado por todas las palabras de la cadena "b", se escribe:

```
>>b.split(" ")
[Los, métodos, son, las, funciones, de, un, objeto]
```

Por supuesto, se puede emplear el método "join" para volver a construir la cadena:

```
>>ans.join(" ")
Los métodos son las funciones de un objeto
```

Para obtener un vector conformado solo por las tres primeras palabras, se escribe:

```
>>b.split(" ",3)
[Los, métodos, son]
```

Para convertir los caracteres alfabéticos de una cadena a mayúsculas se puede emplear el método "toUpperCase()" y para la operación inversa, es decir convertir los caracteres alfabéticos a minúsculas se emplea el método "toLowerCase()". Así para convertir la cadena "b" a mayúsculas se escribe:

```
>>b.toUpperCase()
LOS MÉTODOS SON LAS FUNCIONES DE UN OBJETO
```

Y para convertir este resultado a minúsculas se escribe:

```
>>ans.toLowerCase()
los métodos son las funciones de un objeto
```

Para reemplazar un subcadena por otra, se emplea el método "replace(subcadena, nueva_cadena)", por ejemplo, si el texto es:

```
>>s="Javascript es un lenguaje interpretado"
Javascript es un lenguaje interpretado
```

Se puede reemplazar la palabra "interpretado" por "compilado" con la siguiente instrucción:

```
>>s.replace("interpretado","compilado")
Javascript es un lenguaje compilado
```

Al igual que los otros métodos, "replace" no modifica la cadena original, por lo que el valor de "s" sigue siendo:

```
>>s
Javascript es un lenguaje interpretado
```

Sin embargo, la verdadera potencia de este método y de los métodos "match" y "search" se logra cuando se emplea con expresiones regulares (en lugar de cadenas simples) tal como se ve a continuación.

## 5.2. EXPRESIONES REGULARES

Las expresiones regulares son secuencias de caracteres que pueden ser empleadas para encontrar patrones en una cadena.

En Javascript las expresiones regulares son objetos y como tales se crean formalmente con "new RegExp(patrón,modificadores)". Por ejemplo la siguiente instrucción crea una expresión regular que ubica cualquier letra minúscula (no acentuada) en una cadena:

```
>>r=new RegExp("[a-z]","g")
/[a-z]/g
```

Tal como ocurre con los vectores y las cadenas, las expresiones regulares tienen una forma más sencilla de ser creadas y es la forma que se ve en el resultado de la instrucción, es decir que una expresión regular puede ser creada escribiendo el patrón entre quebrados ("/") y los modificadores después del último quebrado. Por lo tanto, la anterior expresión regular puede ser creada también con:

```
>>r=/[a-z]/g
/[a-z]/g
```

Ambas formas son equivalentes, pero por su sencillez casi siempre se emplea la segunda.

Si se tiene la siguiente cadena:

```
>>s="programación en JAVASCRIPT"
programación en JAVASCRIPT
```

Al aplicar a esta cadena la expresión regular creada, con el método "match(expresión regular)", se obtiene un vector con todas las letras minúsculas (no acentuadas) existentes en la cadena:

```
>>s.match(r)
[p, r, o, g, r, a, m, a, c, i, n, e, n]
```

Los modificadores, en las expresiones regulares, permiten determinar el alcance de la búsqueda y son los siguientes:

| Modificador | Descripción |
|---|---|
| **i** | Si se emplea este modificador, la búsqueda no distingue entre mayúsculas y minúsculas. |
| **g** | Si se emplea este modificador, la búsqueda es global, es decir que encuentra todas las coincidencias y no sólo la primera. |
| **m** | Si se emplea este modificador la búsqueda se realiza en todas las líneas y no sólo la primera. |

Por otra parte, el patrón de una expresión regular, puede estar conformado por texto simple, en cuyo caso encuentra el texto especificado. Por ejemplo con la siguiente instrucción se encuentran todas las letras "a" en la cadena "s" (sin importar que estén escritas en mayúsculas o minúsculas):

```
>>s.match(/a/gi)
[a, a, A, A]
```

El patrón puede estar encerrado también entre corchetes, en cuyo caso encuentra el rango de caracteres especificado o el conjunto de caracteres dado:

| Forma | Descripción |
|---|---|
| **[caracteres]** | Encuentra cualquiera de los caracteres escritos entre los corchetes. Por ejemplo /[aefg]/ encuentra los caracteres "a", "e", "f" o "g". |
| **[^caracteres]** | Encuentra cualquier carácter, excepto los escritos a continuación del circunflejo ("^"). Por ejemplo /[^begxz]/ encuentra todos los caracteres que no sean "b", "e", "g", "x" o "z". |
| **[inicial-final]** | Encuentra todos los caracteres comprendidos entre el carácter "inicial" y el carácter "final". Por ejemplo /[A-Z]/ encuentra todas las letras mayúsculas. |
| **[opc1\|opc2\|opc3]** | Encuentra los caracteres especificados en "opc1" o en "opc2" o en "opc3". Por ejemplo /[código\|javascript\|lenguaje]/ encuentra "código" o "javascript" o "lenguaje". |

En el patrón se pueden emplear también los siguientes metacaracteres:

| Metacarácter | Descripción |
|---|---|
| **.** | Encuentra un carácter (cualquier carácter, excepto saltos de línea: \n). |
| **\w** | Encuentra cualquier carácter alfanumérico, incluido el carácter de subrayado "_". |
| **\W** | Complemento de "\w". Encuentra cualquier carácter que no sea alfanumérico o "_". |
| **\d** | Encuentra un dígito. |

| | |
|---|---|
| **\D** | Complemento de "\d". Encuentra cualquier carácter que no sea un dígito. |
| **\s** | Encuentra un espacio. |
| **\S** | Complemento de "\s". Encuentra cualquier carácter que no sea un espacio. |
| **\b** | Encuentra los caracteres que se encuentran al principio o al final de una palabra. Por ejemplo /\bt/ encuentra los caracteres "t" que se encuentran al principio de una palabra, mientras que /t\b/ encuentra los caracteres "t" que se encuentran al final de una palabra. |
| **\B** | Complemento de "\b". Encuentra coincidencias que no estén al principio o al final de una palabra. |
| **\0** | Encuentra un carácter nulo. |
| **\n** | Encuentra un carácter de salto de línea. |
| **\t** | Encuentra un carácter de tabulación. |
| **\r** | Encuentra un carácter de retorno de carro. |
| **\xxx** | Encuentra el carácter que tiene el código octal xxx. |
| **\xdd** | Encuentra el carácter que tiene el código decimal dd. |
| **\uxxxx** | Encuentra el carácter que tiene el código hexadecimal xxxx. |

Es posible emplear también los siguientes metacaracteres cuantificadores:

| **Metacarácter** | **Descripción** |
|---|---|
| **n+** | Encuentra cualquier cadena que contenga uno o más caracteres o patrones "n". Por ejemplo /Ja+vascript/ encuentra palabras como "Javascript", "JaaavaScript", pero no "Jvascript". |
| **n*** | Encuentra cualquier cadena que contenga 0 o más caracteres o patrones "n". Por ejemplo /co*dificar/ encuentra palabras como "coooodificar", "cdificar", "coodificar", pero no "oodificar". |
| **n?** | Encuentra cualquier cadena que contenga 0 o 1 caracteres o patrones "n". Por ejemplo /pro*grama/ encuentra palabras como "prgrama" y "programa", pero no "proograma". |
| **n{x}** | Encuentra cualquier cadena que contenga una secuencia de "x" caracteres o patrones "n". Por ejemplo /\d{4}/ encuentra cadenas como "1234" "6032" pero no "12" o "344". |
| **n{x,y}** | Encuentra cualquier cadena que contenga una secuencia de entre "x" y "y" caracteres o patrones "n". Por ejemplo /Ja{2,4}vascript/ encuentra cadenas como "Jaavascript", "Jaavavascript", "Jaaaavascript", pero no "Javascript" o "Jaaaavascript". |
| **n{x,}** | Encuentra cualquier cadena que contenga una secuencia de "x" o más caracteres o patrones "n". Por ejemplo /ca(de){2,}na/ encuentra cadenas como "cadedena", "cadedededena" pero no "cadena" o "cana". |

Además una expresión regular puede incluir metacaracteres de posición:

| **Metacarácter** | **Descripción** |
|---|---|
| **^n** | Encuentra una cadena que comience con el carácter o patrón |

| | |
|---|---|
| | "n". Si se especifica el modificador "m" encuentra todas las líneas que comiencen con el carácter o patrón "n". |
| **n$** | Es el complemento de "^n". Encuentra una cadena que termine con el carácter o patrón "n". |
| **(?=n)** | Encuentra cualquier cadena que esté seguida por el carácter o patrón "n". |
| **(?!n)** | Es el complemento de "?=n". Encuentra cualquier cadena que no esté seguida por el carácter o patrón "n". |

Finalmente, dado que las expresiones regulares son objetos, tienen propiedades y métodos propios.

Las propiedades de una expresión regular informan sobre su contenido, así "global", "ignoreCase", "multiline" devuelven verdadero si se han establecido los modificadores "g", "i" o "m" y falso en caso contrario. La propiedad "source" devuelve el patrón de la expresión regular.

Dos de los métodos más empleados en una expresión regular son "exec(cadena)" y "test(cadena)". El primero "exec" devuelve la primera coincidencia del patrón en la cadena o nulo (null) si no hay coincidencias. El segundo devuelve verdadero (true) si hay por lo menos una coincidencia y falso (false) en caso contrario. Si la expresión regular tiene el modificador "g" se puede emplear la propiedad "lastIndex" que devuelve el índice posterior a la última coincidencia encontrada.

Como se dijo la verdadera potencia de métodos como "replace", "match" y "search" (e inclusive "splite") se consigue cuando se emplea en conjunción con expresiones regulares. Por ejemplo si se tiene la siguiente cadena:

```
>>s="El correo electrónico del usuario 1 es: usuario_1@gmail.com, del
usuario 2 es: usuario_2@gmail.com, del usuario 3: usuario_3@gmail.com"
El correo electrónico del usuario 1 es: usuario_1@gmail.com, del usuario
2 es: usuario_2@gmail.com, del usuario 3: usuario_3@gmail.com
```

Y se quiere extraer (en un vector) los correos electrónicos que se encuentran en el texto, se puede emplear el método "match(expresión regular)" con la siguiente expresión:

```
>>s.match(/\w+@\w+\.\w{3}/g)
[usuario_1@gmail.com, usuario_2@gmail.com, usuario_3@gmail.com]
```

Es importante notar que para ubicar las cadenas que tienen un punto seguido de tres caracteres (como ".com", ".net", ".org", etc.), no se ha escrito directamente el punto, sino que se ha precedido el mismo con el quebrado invertido "\", esto porque el punto en una expresión regular (como ya se ha visto) es un metacarácter que representa a cualquier carácter (excepto el salto de línea "\n").

El anteceder un carácter con el quebrado invertido "\" hace que dicho carácter sea tratado como tal y no con el significado especial que pudiera tener. Al par quebrado invertido carácter (o quebrado invertido código) se conoce también como secuencia de escape y se puede emplea en cualquier texto para introducir caracteres especiales como saltos de línea (\n), tabulaciones (\t), comillas (\"), apóstrofos (\'), etc.

Si en la anterior expresión regular no se emplea una secuencia de escape para el punto y el texto en el que se está buscando el correo es:

```
>>t=s.replace(/\./g,"*")
El correo electrónico del usuario 1 es: usuario_1@gmail*com, del usuario
2 es: usuario_2@gmail*com, del usuario 3: usuario_3@gmail*com
```

Devolvería lo siguiente:

```
>>t.match(/\w+@\w+.\w{3}/g)
[usuario_1@gmail*com, usuario_2@gmail*com, usuario_3@gmail*com]
```

Lo que sería incorrecto porque no son correos electrónicos propiamente, en cambio con la secuencia de escape devuelve:

```
>>t.match(/\w+@\w+\.\w{3}/g)
null
```

Que es correcto, pues en la cadena "t" no existe realmente un correo electrónico.

Como otro ejemplo, si se tiene la siguiente cadena:

```
>>s2="Dirección: Calle Ravelo 3456, celular: 74589123, ci: 1028178"
Dirección: Calle Ravelo 3456, celular: 74589123, ci: 1028178
```

Y se quiere ubicar la posición del número de celular en el mismo, se puede emplear el método "search(expresión regular)" con la siguiente expresión:

```
>>i=s2.search(/\d{8}/)
39
```

Luego, una vez conocida la posición del texto, se puede extraer el mismo con "substr" o "substring":

```
>>s2.substr(i,8)
74589123
>>s2.substring(i,i+8)
74589123
```

Por supuesto, se puede extraer también directamente el texto, en forma de vector, con "match":

```
>>s2.match(/\d{8}/)
[74589123]
```

O también con el método "exec" de la expresión regular:

```
>>/\d{8}/.exec(s2)
[74589123]
```

De forma similar, se pueden extraer las fechas de la siguiente cadena:

```
>>s3="reunión en oficina: 7/4/2013; cita con el médico: 15/8/2013; reunión
colegio: 23/11/13"
reunión en oficina: 7/4/2013; cita con el médico: 15/8/2013; reunión
colegio: 23/11/13
>>s3.match(/\d\d?\/\d\d?\/\d{2,4}/g)
[7/4/2013, 15/8/2013, 23/11/13]
```

Tres funciones que se emplean frecuentemente con cadenas, pero que no forman parte de los método estándar del objeto "String" y que por lo tanto no están disponibles en todos los navegadores, son "trim()", que elimina los espacios en blanco al principio y al final de la cadena; "trimLeft()" que elimina los espacios en blanco al principio de la cadena y "trimRight()" que elimina los espacios en blanco al final de la cadena. La mayoría de los navegadores actuales, como Google Chrome y Mozilla Firefox, cuentan con estas funciones, algunos otros, como Opera, sólo implementan alguna de ellas ("trim") y otros como Internet Explorer no implementan ninguna.

Dado que son de utilidad, es conveniente que estén disponibles en todos los navegadores. Afortunadamente, en Javascript, es posible añadir métodos y propiedades a un objeto a través de la propiedad "prototype" y para ello

simplemente se añade el método (función) o propiedad (dato) a prototype. Así para "trimLeft()" se puede escribir el siguiente código:

```
try {
  "x".trimLeft();
} catch(e) {
    String.prototype.trimLeft=function(){
      return this.replace(/^\s+/g,"");
    }
}
```

Como se puede ver, el método se añade dentro de un bloque "try-catch", donde primero se intenta emplear el método "trimLeft" y si ese intento falla (si se produce un error) se crea el método trimLeft. El método en sí, simplemente devuelve el resultado de reemplazar (mediante una expresión regular) los espacios en blanco al principio de la cadena con caracteres nulos (""). De esta manera el método se añade sólo si no existe.

De forma similar se pueden añadir los métodos "trimRight" y "trim":

```
try {
  "x".trimRight();
} catch(e) {
    String.prototype.trimRight=function(){
      return this.replace(/\s+$/g,"");
    }
}
try {
  "x".trim();
} catch(e) {
  String.prototype.trim=function(){
    return this.trimLeft().trimRight();
  }
}
```

Como se puede ver, en el último método, simplemente se llama a los dos anteriores para eliminar los espacios en blanco al principio y al final de la cadena: primero se eliminan los espacios en blanco al principio de la cadena (con "trimLeft") y a la cadena resultante (que ya no tiene espacios en blanco al principio) se le quitan los espacios en blanco al final (con "trimRight").

En estos métodos la palabra "this" (este) hace referencia al objeto al cual se están añadiendo los métodos, es decir a "String" (la cadena). Por lo tanto "this", en estos métodos es la cadena de caracteres en sí.

Como ya se ha visto, para que una función (en este caso un método) esté disponible en cualquier aplicación, es necesario que sea guardado en una librería, en nuestro caso en la librería "sis101.js".

Empleando estos métodos desde la calculadora con la siguiente cadena:

```
>>s="       Hola Mundo       "
       Hola Mundo
```

Con "trimLeft" se obtiene:

```
>>s.trimLeft()
Hola Mundo
```

Como en el resultado no se puede apreciar si se han eliminado o no los espacios en blanco a la derecha, se puede ver la longitud de la cadena resultante:

```
>>ans.length
17
```

Que al compararla con la longitud original de la cadena:

```
>>s.length
24
```

Revela que sólo se han eliminado los espacios en blanco a la izquierda (pues de lo contrario la longitud de la cadena sería 10, no 17). Del mismo modo se prueban los otros dos métodos:

```
>>s.trimRight()
       Hola Mundo
>>ans.length
17
>>s.trim()
Hola Mundo
>>ans.length
10
```

Estos ejemplos dan una idea de la potencia y flexibilidad de las expresiones regulares. Y esta característica hace que sean una de las opciones más viables al momento de validar los datos introducidos por los usuarios. Por ejemplo, para verificar si el número de teléfono introducido por el usuario está en el formato "###-##-#######", donde los "#" representan a los números, se puede emplear una expresión regular. Así si el número introducido por el usuario es:

```
>>s="591-04-6453234"
591-04-6453234
```

La prueba (test) con la siguiente expresión regular devuelve verdadero:

```
>>/\d{3}-\d{2}-\d{7}/.test(s)
true
```

Y es así porque el número está bien escrito, sin embargo, si el número introducido por el usuario es:

```
>>s2="59-04-6418372"
59-04-6418372
```

Donde falta un dígito en el primer grupo, la prueba con la expresión regular devuelve falso:

```
>>/\d{3}-\d{2}-\d{7}/.test(s2)
false
```

Por supuesto, si como en este caso, se emplea la misma expresión regular en más de una ocasión, no debe volver a ser escrita, sino guardada en una variable:

```
>>r=/\d{3}-\d{2}-\d{7}/
/\d{3}-\d{2}-\d{7}/
>>r.test(s)
true
```

```
>>r.test(s2)
false
```

La validación de teclas se puede simplificar también con las expresiones regulares, por ejemplo si el código de la tecla leída está en la variable "key" y por ejemplo es igual a 50:

```
>>key=50
50
```

Se puede convertir este código en el carácter respectivo:

```
>>c=String.fromCharCode(key)
2
```

Y verificar que es un número empleando una expresión regular:

```
>>/\d/.test(c)
true
```

Si además de números se debe permitir los caracteres "e", "E", "+", "-" y "." (números reales), la expresión regular cambia a:

```
>>/[0-9.eE+\-]/.test(c)
true
```

Que probada con otros caracteres resulta:

```
>>/[0-9.eE+\-]/.test("-")
true
>>/[0-9.eE+\-]/.test("*")
false
>>/[0-9.eE+\-]/.test("E")
true
>>/[0-9.eE+\-]/.test("h")
false
```

Observe que el signo menos ("-") ha sido escrito como una secuencia de escape, esto porque el signo menos dentro de los corchetes especifica un rango de caracteres (tal como ocurre con "0-9").

De esta forma se pueden emplear las expresiones regulares para validar, de manera más sencilla, tanto las teclas pulsadas como la información introducida.

Sin embargo, si la validación se la hace en función a los caracteres en lugar de códigos, resulta más práctico trabajar con el evento "onkeypress" en lugar del evento "onkeydown", pues el evento "onkeypress" devuelve el código del carácter pulsado en lugar del código de la tecla pulsada.

## 5.3. TABLAS

Con frecuencia, en una página HTML, es necesario presentar información en forma tabular, por ejemplo cuando se trabaja con matrices o se presentan listas con dos o más columnas. Para ese fin HTML cuenta con las tablas.

Las tablas en una página HTML se crean con la etiqueta <table></table>. Dentro de una tabla las filas se crean con la etiqueta <tr></tr> y dentro de una fila cada celda se crea con la etiqueta <td></td> o <th></th> cuando se quiere que el contenido esté centrado y en negrita.

Además, las filas de una tabla pueden ser agrupadas en tres segmentos: el encabezado con <thead></thead>, el cuerpo con <tbody></tbody> y el pie con

<tfoot></tfoot>. Adicionalmente se puede dar un título a la tabla con <caption></caption>.

Si se emplea la etiqueta <table>, sin ningún atributo, la tabla resultante no tiene líneas de división, por ejemplo con la siguiente instrucción se crea una tabla con dos filas y dos columnas, pero que no tiene líneas de división:

```
>>write("<table><tr><td>Dato 1</td><td>Dato 2</td></tr><tr><td>10</td>
<td>20</td></tr></table>")
Dato 1 Dato 2
10     20
```

Para que las líneas de división sean visibles se debe establecer la propiedad "border" en algún ancho, por ejemplo la anterior tabla con un borde igual a "1" produce el siguiente resultado:

```
>>write("<table border='1'><tr><td>Dato 1</td><td>Dato 2</td></tr><tr>
<td>10</td><td>20</td></tr></table>")
```

| Dato 1 | Dato 2 |
|---|---|
| 10 | 20 |

Observe que el ancho del borde (1), ha sido encerrado entre apóstrofos, esto porque toda la instrucción está encerrada entre comillas, por lo que no se pueden emplear otras comillas dentro la misma (a no ser que esté como una secuencia de escape '\"').

Lamentablemente las tablas sólo permiten mostrar datos, no introducirlo, por lo que para lograr dicha funcionalidad es necesario escribir código Javascript.

## 5.4. HOJAS DE ESTILO EN CASCADA (CSS)

El aspecto visual de los elementos HTML se modifica, en los navegadores actuales, mediante la aplicación de estilos, más propiamente las hojas de estilo en cascada: CSSS, por su siglas en inglés: Cascading Style Sheets.

CSS define un conjunto de propiedades que permiten modificar la apariencia y comportamiento de cualquier elemento HTML. Existen tres formas en las que se pueden aplicar dichas instrucciones: a) escribiendo los estilos en un archivo separado e importándolo en la página con la etiqueta <link>; b) escribiendo los estilos en la página HTML dentro de las etiquetas <style></style>, normalmente en el encabezado de la página y c) escribiendo los estilos en el atributo "style" de los elementos.

La más sencilla, aunque no la más recomendada, es la última forma. Por ejemplo con la siguiente instrucción se muestra el texto del párrafo ("Hola Mundo") en color rojo sobre un fondo amarillo:

```
>>write("<p style='color:red;background-
color:yellow'>Javascript es un lenguaje flexible</p>")

Javascript es un lenguaje flexible
```

Como ocurre con todos los atributos HTML, el valor del atributo debe estar encerrado entre comillas o entre apóstrofos, en este ejemplo se han empleado apóstrofos porque la instrucción en su conjunto está encerrada entre comillas.

Como se puede ver en el ejemplo, cada propiedad CSS está conformada por el par "nombre_de_la_propiedad:valor" y las propiedades se separan unas de

otras con puntos y comas. Las propiedades empleadas en este ejemplo son "color", que permite cambiar el color del texto y "background-color" que permite cambiar el color de fondo del elemento.

En CSS el color se establece en una de 4 formas: a) Con el nombre del color en inglés, pero se debe tomar en cuenta que solo tienen nombre un limitado conjunto de colores; b) Con la función rgb(r,g,b), donde "r" es el código del color rojo (un número comprendido entre 0 y 255, siendo 0 la ausencia de color y 255 el color puro), "g" es el código del color verde y "b" es el código del color azul; c) Con #rrggbb, donde "rr" es el código del color rojo (en hexadecimal, donde el valor mínimo es "00" y máximo "FF"), "gg" es el código del color verde y "bb" es el código del color azul y d) Con rgb(r%,g%,b%), donde "r%" es el porcentaje de color rojo (comprendido entre 0% y 100%), "g%" es el porcentaje de color verde y "b%" es el porcentaje de color azul.

Cuando una propiedad debe ser fijada como alguna medida, por ejemplo para el ancho de un elemento, dicha medida puede tener las siguientes unidades: % = porcentaje del elemento contenedor; in = pulgadas; cm = centímetros; mm = milímetros; em = tamaño en puntos de la fuente actual (normalmente 12 pt); ex = alto, en puntos, de la fuente actual; pt = puntos (1 pt = 1/72 pulgadas); pc = pica (1 pc = 12 pts) y px = pixeles (un punto en la pantalla de la computadora).

En cuanto a las imágenes, se asignan mediante al función "url('URL')", donde "URL" es la dirección Internet donde se encuentra la imagen o el nombre del archivo en el disco duro (incluyendo el camino si la imagen no se encuentra en el mismo directorio que el archivo HTML). Por ejemplo url('scroll.gif'), asigna como imagen el archivo "scroll.gif' que debe estar en el mismo directorio que la página, de no ser así, se debe escribir el camino completo para llegar al archivo, por ejemplo: url('C:/Downloads/images/scroll.gif'), observe que la separación entre directorios se la hace con el quebrado normal ("/"), no el quebrado invertido ("\") como ocurre normalmente en Windows.

Algunas de las propiedades CSS son:

| Propiedad | Descripción |
|---|---|
| background-color | Establece el color de fondo del elemento. |
| background-image | Establece la imagen de fondo del elemento: url('url o dirección de la imágen') |
| background-position | Establece la posición de la imágen: left top, left center, left bottom, right top, right center, right bottom, center top, center, center bottom o posiciónx posicióny |
| border-color | Establece el color del borde del elemento. |
| border-style | Establece el estilo del borde: hidden, dotted, dashed, solid, doublé, Groove, ridge, inset, outset. |
| border-width | Establece el ancho del borde. |
| height | Establece el alto del elemento. |
| width | Establece el ancho del elemento. |
| font-family | Especifica la familia de caracteres a emplear. Si el nombre de la familia tiene espacios, debe estar encerrada entre comillas, p.e. "Times New Roman". Cuando se especifica más de una familia deben estar separadas con comas. |
| font-size | Especifica el tamaño del texto: xx-small, x-small, small, médium, large, x-large, xx-large, smaller, |

| | |
|---|---|
| | larger o tamaño en unidades o porcentaje. |
| font-style | Especifica el estilo del texto: normal, italic, oblique. |
| font-weight | Especifica el espesor del texto: normal, bold, bolder, lighter. |
| margin | Establece los márgenes del elemento (el espacio alrededor del elemento). |
| padding | Establece el espacio entre el contenido del elemento y el borde del mismo. |
| position | Establece la posición de un elemento: absolute, fixed, relative, static (el valor por defecto). |
| left | Fija el margen izquierdo para un elemento con posición absoluta o relativa. |
| right | Fija el margen derecho para un elemento con posición absoluta o relativa. |
| top | Fija el margen superior para un elemento con posición absoluta o relativa. |
| bottom | Fija el margen inferior para un elemento con posición absoluta o relativa. |
| overflow | Especifica qué sucede si el contenido sobrepasa al elemento: visible (el contenido que excede al elemento está visible, valor por defecto), hidden (el contenido que excede al elemento se oculta), scroll (aparecen barras de desplazamiento para ver el contenido excedente), auto. |
| cursor | Determina el tipo de cursor a ser empleado cuando el puntero está sobre el elemento: auto, crosshair, default, e-resize, help, move, n-resize, ne-resize, nw-resize, pointer, progress, s-resize, se-resize, sw-resize, text, w-resize, wait. |
| display | Determina el tipo de caja para el elemento: none (sin caja), block (con saltos de línea antes y después), inline (sin saltos de línea), table (como una tabla), table-caption (como el título de una tabla), table-cell (como la celda de una tabla), table-column (como la columna de una tabla), table-row (como la fila de una tabla). |
| float | Determina si el elemento flotará o no: left (flota hacia la izquierda), right (flota a la derecha), none (no flota, por defecto). |
| visibility | Especifica si el elemento es o no visible: visible, hidden, collapse (sólo para tablas). |
| z-index | Establece el orden en que se apilan los elementos: auto o un número (mientras menor sea el número, puede ser negativo, más atrás se encuentra de los otros elementos). |
| border-collapse | Hace que los bordes de las tablas se muestren como uno solo (collapse) o como dos (separate) que es el valor por defecto: collapse, separate. |
| Color | Establece el color del texto. |
| text-align | Establece la alineación horizontal del texto: left (por defecto), right, center, justify. |
| text-decoration | Establece la decoración del texto: none (por defecto), underline, overline, line-through, blink. |
| text-transform | Transforma el texto a mayúsculas (uppercase), a minúsculas (lowercase) y a primeras letras en mayúsculas (capitalize). |

| | |
|---|---|
| text-indent | Establece la identación de la primera línea de un bloque de texto: unidad de medida o porcentaje. |
| vertical-align | Establece la alineación vertical del texto: baseline, sub, super, top, text-top, middle, bottom, text-bottom o unidad de medida o porcentaje. |
| white-space | Determina como se manejan los espacios en blanco en un elemento: normal (por defecto: dos o más espacios en blanco son tratados como uno), nowrap (el texto continúa en la misma línea hasta encontrar un <br>), pre (se conservan los espacios en blanco), pre-wrap (se conservan los espacios en blanco y el texto pasa a la siguiente línea cuando ya no hay espacio en la actual). |

Esta no sólo es una lista parcial, sino que además no se han considerado casos específicos y generales de algunas propiedades. Por ejemplo la propiedad border-color, puede ser aplicada específicamente a uno de los bordes con: border-left-color, border-right-color, border-top-color y border-bottom-color. Por el contrario es posible asignar todas las propiedades del borde en una sola instrucción con: border: ancho estilo color. Lo mismo es válido para otras propiedades como “margin” y “padding”.

A más de las propiedades, CSS cuenta con las siguientes pseudo-clases y pseudo-elementos:

| Propiedad | Descripción |
|---|---|
| :active | Aplica un estilo al elemento que está activo. |
| :after | Añade contenido después de un elemento. |
| :before | Añade contenido antes de un elemento. |
| :first-child | Aplica un estilo al primer elemento dentro del elemento. |
| :first-letter | Aplica un estilo al primer carácter de un texto. |
| :first-line | Aplica un estilo a la primera línea de un texto. |
| :focus | Aplica un estilo al elemento que tiene el foco. |
| :hover | Aplica un estilo a un elemento cuando el mouse se mueve sobre el mismo. |
| :lang | Aplica un estilo a un elemento con un atributo de lenguaje específico. |
| :link | Aplica un estilo a un enlace no visitado. |
| :visited | Aplica un estilo a un enlace visitado. |

Adicionalmente, existen muchas otras propiedades que han sido incluidas en la especificación CSS3, lamentablemente no todas ellas son soportadas aún por todos los navegadores. Algunas de dichas propiedades se emplearán en temas posteriores según se requiera.

En las primeras aplicaciones de los estilos CSS se empleará la tercera forma, es decir serán escritos directamente como un atributo del elemento, tal como se ha mostrado en el ejemplo. Luego, se emplearán la segunda y primera forma, que es la empleada en las aplicaciones WEB profesionales.

## 5.5. EL ELEMENTO <DIV>

Otro elemento que se emplea frecuentemente en la elaboración de aplicaciones WEB es . Este elemento permite crear una división o sección dentro de una página HTML y en su interior se puede incluir cualquier elemento HTML (excepto por supuesto <body>), por esta razón  suele ser empleado para agrupar elementos HTML en secciones de la página.

Por sí solo  es simplemente un rectángulo en blanco (sin bordes), por lo que en realidad no es visible a menos que se modifique alguna de sus propiedades (como el color de fondo y/o el borde del elemento).

Por ejemplo la siguiente instrucción crea un elemento  de 250px de ancho, con texto en color azul, fondo amarillo claro y borde de color verde:

```
>>write("<div style='width:250px; color:blue; background-
color:lightyellow; border:1px solid green;'>Este texto  está dentro
del elemento 'div'</div>")
Este texto está dentro del
elemento 'div'
```

Estos nuevos elementos, así como los estilos CSS, se comenzarán a aplicar a partir del presente tema.

## 5.6. EJEMPLOS

**1. Elabore una aplicación que permita introducir texto, en una o varias líneas, conteniendo únicamente letras en minúsculas (incluido acentos, espacios puntos, comas y la tecla enter). La aplicación debe mostrar el número total de caracteres escritos, el número de palabras escritas, el número de palabras con 3 o más letras, convertir las letras iniciales de cada palabra a mayúsculas y mostrar el texto resultante en un elemento  con las letras convertidas resaltadas en color rojo siendo el fondo del elemento amarillo claro.**

Para la introducción del texto se empleará un elemento <textarea>, porque permite introducir texto en una o más líneas y dado que sólo debe ser posible escribir letras en minúsculas, se validará el teclado con el evento "onkeypress". En este caso resulta más práctico emplear este evento porque devuelve directamente el código del carácter pulsado (no de la tecla), por lo tanto diferencia entre mayúsculas y minúsculas. Se puede emplear también el evento "onkeydown", pero en ese caso se debe controlar además si la tecla "shift" (o "caps lock") ha sido pulsada.

Sin embargo, cuando se emplea el evento "onkeypress" se debe tomar encuenta que el mismo sólo responde a las teclas "normales", es decir que prácticamente sólo responde a las teclas alfanuméricas (y la tecla "enter"), por lo tanto con este evento no se pueden validar teclas como las teclas de función (F1 a F12), los cursores,  teclado numérico, etc.

Para mostrar el número de caracteres, el número palabras y el número de palabras con 3 o más letras, se empleará un elemento <table> y para mostrar el texto resultante se empleará un elemento  (tal como dicta el enunciado).

El código de la página HTML donde se resuelve el problema es:

```
<DOCYTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ejemplo 1</title>
  <script src="sis101.js"></script>
  <script>
    //Función que valida la pulsación de teclas en el elemento
    //textarea1
    function validarTecla(event){
```

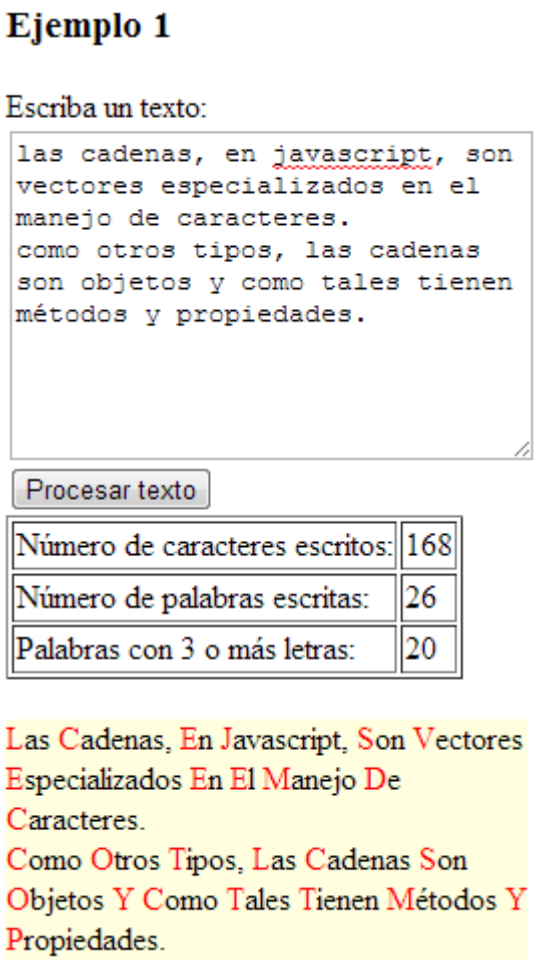
```
  var key=tecla(event);
  //Si la tecla es 'enter' se le asigna su secuencia de escape
  var c= (key!=13) ? String.fromCharCode(key) : "\n";
  //Se verifica que la tecla sea una letra, una vocal acentuada,
  //la letra "ñ", el espacio " " o la tecla enter "\n"
  if (/[a-záéíóúñ,. \n]/.test(c)) return true;
  else return false;
}
//Función que procesa el texto escrito para obtener los
//resultados pedidos
function procesarTexto(texto){
  //Número de caracteres escritos
  var nce=texto.length;
  document.getElementById("nce").innerHTML=nce;
  //Palabras escritas = cadenas que tienen una o más letras
  //seguidas por un espacio (\s), enter (\n) o una letra
  var npe=texto.match(/[\wáéíóúñ,\.]+(?=[\s,\n\w])/g).length
  document.getElementById("npe").innerHTML=npe;
  //Palabras con 3 o más letras
  var pm3=texto.match(/[\wáéíóúñ,\.]{3,}(?=[\s,\n\w])/g).length;
  document.getElementById("pm3").innerHTML=pm3;
  //Conversión a mayúsculas de las primeras letras
  var j=0, s="", tr="", np="true", cb="false";
  for (var i=0;i<texto.length;i++){
    //si el carácter es un espacio en blanco el siguiente
    //carácter no blanco es una palabra (np=true)
    if (texto.charAt(i)==" ") {
      np=true;
      //Si hay más de un espacio en blanco se reemplaza el segundo
      //y posterior espacios blanco por su equivalente HTML
      tr+= (cb) ? " " : " ";
      cb=true; continue;}
    //Si el carácter es un salto de línea el siguiente
    //carácter no blanco es una palabra (np=true)
    if (texto.charAt(i)=="\n"){
      np=true; cb=false;
      //se reemplaza el salto de línea por su equivalente HTML
      tr+="<br>"
      continue;
    }
    if (np) {
      //si es una nueva palabra se escribe en color rojo y en
      //mayúsculas
      tr+="<span style='color:red;'>"+texto.charAt(i).
        toUpperCase()+"</span>"; np=false;}
    else
      //si no se añade el carácter al texto resultante
      tr+=texto.charAt(i);
```

```
            cb=false;
          }
          document.getElementById("div1").innerHTML=tr;
        }
      </script>
    </head>
    <body>
      <h3>Ejemplo 1</h3>
      <label for="textarea1">Escriba un texto:</label><br>
      <textarea id="textarea1" cols="30" rows="10" onkeypress=
        "return validarTecla(event)">
      </textarea><br>
      <button onclick="procesarTexto(textarea1.value);">Procesar texto
      </button>
      <table id="table1" border="1">
        <tr><td>Número de caracteres escritos:</td><td id="nce">0</td></tr>
        <tr><td>Número de palabras escritas:</td><td id="npe">0</td></tr>
        <tr><td>Palabras con 3 o más letras:</td><td id="pm3">0</td></tr>
      </table><br>
      <div id="div1" style="width:260px;background-color:lightyellow;">
      </div>
    </body>
    </html>
```

Como se puede ver, el número de palabras escritas se determina empleando expresiones regulares, también se emplean expresiones regulares en la validación del teclado, pero para la conversión de las letras iniciales a mayúsculas se emplean estructuras estándar. En este proceso en lugar de "texto.charAt(i)" se puede emplear directamente texto[i] (pues como se ha dicho las cadenas de caracteres son vectores especializados en caracteres).

Para mostrar el texto en color rojo se ha empleado un nuevo elemento HTML: , este elemento no tiene representación visual y se emplea principalmente para añadir estilos a una parte del texto (como se ha hecho en este ejemplo). Observe también que para modificar el contenido de un elemento se asigna el texto HTML creado a la propiedad "innerHTML" del mismo.

Abriendo esta página en un navegador e introduciendo algún texto se obtiene algo parecido a lo siguiente:




Si lo que se quiere es mostrar la primera letra de cada palabra en mayúsculas, pero no realmente convertirla, se puede conseguir dicho dicho resultado de manera más sencilla con una expresión regular y estilos CSS, por ejemplo, si el texto es:

```
>>s1="En un lugar de la mancha de cuyo nombre no quiero acordarme";
En un lugar de la mancha de cuyo nombre no quiero acordarme
```

Se puede obtener el resultado buscado (y ver el texto transformado en la calculadora) con la siguiente instrucción:

```
>>write(s1.replace(/(\b\w)/g,"<span style='color:red;text-
transform:uppercase;'>$1</span>"))
En Un Lugar De La Mancha De Cuyo Nombre No Quiero Acordarme
```

En esta expresión "$1" representa a las cadenas que concuerdan con la expresión regular "\b\w" (es decir las primeras letras de cada palabra), pero para que "$1" represente a esas coincidencias, la parte de la expresión regular a la que representa debe estar entre paréntesis (como se muestra en el ejemplo). De esta forma es posible agrupar y trabajar con las coincidencias de hasta 99 grupos (de $1 a $99) dentro de una expresión regular.

**2. Elabore una aplicación que permita introducir texto alfanumérico, en una o varias líneas, conteniendo letras tanto mayúsculas como minúsculas (incluido acentos, espacios puntos, comas y la tecla enter), así como siglas de materias en el formato CCCDDD (3 caracteres y 3 dígitos) y números de carnet universitario en el formato "##-#####". La aplicación**

**debe encontrar el número de siglas escrito, el número de carnets universitarios escritos, el número total de dígitos escritos, mostrar en una tabla de dos columnas las dos primeros dígitos del carnet en la primera columna y los restantes en la segunda y mostrar en un elemento  las siglas en color verde, los dos primeros números del carnet en color rojo y los números restante en azul (todo sobre un fondo cian claro)**

Como en el anterior ejemplo el texto será introducido en un <textarea> y las teclas pulsadas se validarán en el evento "onkeypress". El número de siglas y el número de carnets escritos se mostrarán en una tabla y como dicta el enunciado los otros datos se mostrarán en una tabla y en un elemento .

El código de la página HTML que resuelve el problema es:

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ejemplo 2</title>
  <script src="sis101.js"></script>
  <script>
    //Función que valida la pulsación de teclas en el elemento
    //textarea1
    function validarTecla(event){
      var key=tecla(event);
      //Si la tecla es 'enter' se le asigna su secuencia de escape
      var c= (key!=13) ? String.fromCharCode(key) : "\n";
      //Se verifica que sea un carácter válido
      if (/[A-Za-z\d\-ÁÉÍÓÚÑáéióúñ,.:\(\) \n]/.test(c)) return true;
      else return false;
    }
    //Procesamiento el texto escrito para obtener los resulados
    function procesarTexto(texto){
      //Número de siglas escritas
      var nse=texto.match(/[A-Z]{3}\d{3}/g).length;
      document.getElementById("nse").innerHTML=nse;
      //Número de carnets universitarios escritos
      var nce=texto.match(/\d{1,2}-\d{4,5}/g).length
      document.getElementById("nce").innerHTML=nce;
      //Número de dígitos escritos
      var nde=texto.match(/\d/g).length;
      document.getElementById("nde").innerHTML=nde;
      //Creación del texto para la tabla "table1"
      //Vector con los números de carnet
      var vnc=texto.match(/\d{1,2}-\d{4,5}/g);
      var tr="";
      for (var i=0;i<vnc.length;i++)
        tr+="<tr><td>"+vnc[i].substr(0,vnc[i].indexOf("-"))+"</td>"+
```

```
            "<td>"+vnc[i].substr(vnc[i].indexOf("-")+1)+"</td></tr>";
        document.getElementById("table1").innerHTML=tr;
        //Creación del texto para el elemento div1
        var tr=texto.replace(/([A-Z]{3}\d{3})/g,
            '<span style="color:green;">$1</span>');
        var tr=tr.replace(/(\d{2}(?=-))/g,
            '<span style="color:red;">$1</span>');
        var tr=tr.replace(/-(\d{4,5})/g,
            '-<span style="color:blue;">$1</span>');
        document.getElementById("div1").innerHTML=tr;
      }
    </script>
  </head>
  <body>
    <!-- Elemento para la introducción del texto -->

    <label for="ta1">Escriba texto con siglas y CUs:<label><br>
    <textarea  id="ta1" cols="30" rows="10"
      onkeypress="return validarTecla(event);">
    </textarea><br>
    <!-- Botón para iniciar el procesamiento del texto-->
    <button onclick="procesarTexto(ta1.value)">Procesar texto
    </button><br>
    <!--Elemento para mostrar las estadísticas-->
    <table border="1">
      <tr><td>Número de siglas escritos:</td><td id="nse">0</td></tr>
      <tr><td>Número de carnets escritos:</td><td id="nce">0</td></tr>
      <tr><td>Número de dígitos escritos:</td><td id="nde">0</td></tr>
    </table><br>
    <!--Elemento para mostrar los números de carnet separados en
        dos columnas-->
    <table id="table1" border="1">
    </table><br>
    <!--Elemento para mostrar el texto resaltado-->
    <div id="div1" style="width:260px;background-color:lightcyan;">
    </div>
  </body>
  </html>
```

Abriendo la página en un navegador, se obtiene algo parecido a:

Escriba texto con siglas y CUs:

```
Algunas de las materias son:
cálculo I MAT101, cálculo II
MAT102, física I FIS100, física
II FIS102, física 3 FIS200.
Algunos números de carnet son:
32-3456, 56-7784, 35-12345, 35-
3245, 26-54234, 46-1093.
```

Procesar texto

| | |
|---|---|
| Número de siglas escritos: | 5 |
| Número de carnets escritos: | 6 |
| Número de dígitos escritos: | 54 |

| | |
|---|---|
| 32 | 3456 |
| 56 | 7784 |
| 35 | 12345 |
| 35 | 3245 |

| | |
|---|---|
| 26 | 54234 |
| 46 | 1093 |

Algunas de las materias son: cálculo I MAT101, cálculo II MAT102, física I FIS100, física II FIS102, física 3 FIS200. Algunos números de carnet son: 32-3456, 56-7784, 35-12345, 35-3245, 26-54234, 46-1093.

## 5.7. EJERCICIOS

Para presentar los ejemplos y ejercicios del tema, debe crear una carpeta con el nombre "tema5" y guardar en la misma todos los archivos HTML creados (ejemplos y ejercicios).

1. Elabore una aplicación que permita introducir texto, en una o varias líneas, conteniendo únicamente letras en mayúsculas (incluido acentos, espacios puntos, comas y la tecla enter). La aplicación debe mostrar el número total de vocales escritas, el número total de saltos de línea, el número de palabras con 3 letras o menos y convertir la letra final de cada palabra a mayúsculas, mostrando el texto resultante en un elemento  con las letras convertidas en color azul, siendo el fondo del elemento verde claro.

2. Elabore una aplicación que permita introducir texto alfanumérico, en una o varias líneas, incluido acentos, espacios puntos, comas y la tecla enter. La aplicación debe contar todos los dígitos escritos, contar todas las vocales acentuadas, contar todos los grupos de números, encontrar los números telefónicos que se encuentre en el formato: ###-##### y mostrarlos en un  en color rojo sobre un fondo azul claro.

3. Elabore una aplicación que permita introducir texto alfanumérico, en una o varias líneas, incluido direcciones de correo electrónico. La aplicación debe contar todos los correos electrónicos escritos, mostrar en una tabla de dos columnas, los nombres de usuario en la primera (la cadena que se encuentra antes de @) y los servidores en la segunda (la cadena que se encuentra después de @). Finalmente debe mostrar el texto en un elemento  con los correos electrónicos en color verde oscuro, siendo el fondo del documento verde claro.

4. Elabore una aplicación que permita introducir texto alfanumérico, en una o varias líneas, incluido fechas de nacimiento en el formato dd/mm/aaaa (donde los días y los meses pueden tener uno o dos dígitos y los años dos o cuatro dígitos). La aplicación debe contar todas las fechas escritas, mostrar en una tabla con tres columnas el día en la primera columna, el mes en la segunda y el año en la tercera. Finalmente debe mostrar el texto en un elemento  con las fechas resaltadas de la siguiente forma: días en color verde, meses en color azul, años en color café y quebrados de división (“/) en color rojo, siendo el fondo amarillo claro.

# 6. ESTRUCTURAS DE DATOS

En los anteriores temas se han estudiado dos datos estructurados predefinidos por el lenguaje: las matrices y las cadenas. En este tema se estudia la forma de crear y emplear estructuras de datos propias.

Como sucede en la mayoría de los lenguajes actuales, las estructuras no sólo permiten guardar datos, sino también funciones, lo que en principio constituye la definición de un objeto: "un objeto es un elemento que contiene tanto datos como las funciones que operan sobre dichos datos". Por esta razón en este tema se verán también algunos conceptos básicos relativos a objetos, sin embargo, el estudio se centra sobre todo en las estructuras de datos más que en los objetos como tales.

## 6.1. LA NOTACIÓN JSON

La notación JSON: JavaScript Object Notation (Notación de Objetos de JavaScript), es un formato para el intercambio de información y es a la vez la forma más sencilla de crear estructuras de datos en Javascript.

Ya se ha empleado esta notación al trabajar con matrices y vectores, siendo la forma más empleada en la práctica. La notación JSON para matrices especifica que sus elementos deben estar encerrados entre corchetes ("[ ]") y separados con comas (tal como se ha hecho en el anterior tema). Así la siguiente instrucción crea un vector (un Array) cuyos elementos son los días de la semana:

```
var x=["domingo","lunes","martes","miércoles","jueves","viernes","sábado"]
```

Como ya se ha visto, sin la notación JSON, el Array debe ser creado con:

```
var x = new Array("domingo","lunes","martes","miércoles","jueves",
"viernes","sábado");
```

O con:

```
x = new Array()
x[0]="domingo"; x[1]="lunes"; x[2]="martes"; x[3]="miércoles";
x[4]="jueves"; x[5]="viernes"; x[6]="sábado";
```

Que obviamente es más moroso que con la notación JSON.

Las estructuras en Javascript son igualmente Objetos y la notación JSON para crear un objeto es la siguiente:

```
var nombre_objeto = {propiedad1:valor1, propiedad2:valor2, ...,
propiedadn:valorn};
```

Es decir que los pares propiedad/valor (o método/definición) deben estar separados unos de otros con comas y la propiedad debe estar separada de su valor con dos puntos (":"). Los nombres de las propiedades pueden tener espacios, pero en ese caso deben estar encerrados entre comillas o apóstrofos, igualmente se debe encerrar entre comillas o apóstrofos palabras reservadas o nombres que emplean caracteres no validos (tales como "*", "/", etc.). Como el encerrar los nombres de las propiedades entre comillas no constituye un error en ningún caso, se recomienda encerrarlos siempre que se tenga duda.

Así, para crear una estructura con los datos de un empleado se puede escribir:

```
>>empleado1={nombre:"Juan Pérez", direccion:"Calle Junín 345",
celular:73242361,fecha_ingreso:"1/4/2005"}
```

```
{nombre:Juan Pérez, direccion:Calle Junín 345, celular:73242361,
fecha_ingreso:1/4/2005}
```

Pero es igualmente correcto escribir:

```
>>empleado1={"nombre":"Juan Pérez", "direccion":"Calle Junín 345",
'celular':73242361,'fecha_ingreso':"1/4/2005"}
{nombre:Juan Pérez, direccion:Calle Junín 345, celular:73242361,
fecha_ingreso:1/4/2005}
```

En la programación estructurada, las propiedades de una estructura se conocen como "campos", razón por la cual se hará referencia a ellos también con ese nombre. Por otra parte se recuerda que en la calculadora las variables globales se crean sin la palabra "var", siendo esta la razón por la cual no se ha empleado la misma en estas instrucciones.

Al igual que sucede con otras estructuras, los espacios, tabulaciones y saltos de línea son ignorados al momento de interpretar la notación JSON, lo que permite escribir estas instrucciones en un formato más legible. Así la anterior estructura puede ser escrita de forma más clara de la siguiente forma:

```
empleado1 = {
  nombre:"Juan Pérez",
  direccion:"Calle Junín 345",
  celular:73242361,
  fecha_ingreso:"1/4/2005"
}
```

Por supuesto, este tipo de formato es de mayor utilidad en el código de un página HTML que en la calculadora Javascript.

Para acceder a las propiedades (o métodos) de esta estructura, se procede como con las propiedades de cualquier objeto en Javascript, es decir se escribe el nombre del objeto, un punto y el nombre de la propiedad. Por ejemplo para obtener la dirección de "empleado1", se escribe:

```
>>empleado1.direccion
Calle Junín 345
```

Y para cambiar el número del celular a 62876543, se escribe:

```
>>empleado1.celular=62876543
62876543
```

Empleando esta notación es posible añadir una nueva propiedad al objeto (después de que ha sido creado) así para añadir el carnet de identidad del empleado se escribe:

```
>>empleado1.ci=1092883
1092883
```

Con lo que ahora el objeto tiene las siguientes propiedades:

```
>>empleado1
{nombre:Juan Pérez, direccion:Calle Junín 345, celular:62876543,
fecha_ingreso:1/4/2005, ci:1092883}
```

A estas estructuras se las conoce también como Array asociativos, porque es posible acceder a sus elementos empleando la notación de los Array, sólo que en lugar de escribir el índice se escribe la propiedad (o método). Así por ejemplo, para obtener el nombre del empleado se puede escribir:

```
>>empleado1['nombre']
Juan Pérez
```

O lo que es lo mismo:

```
>>empleado1["nombre"]
Juan Pérez
```

Igualmente, se puede cambiar el valor de una propiedad con esta notación, así para cambiar la fecha de ingreso a 1/7/2003, se escribe:

```
>>empleado1["fecha_ingreso"]="1/7/2003"
1/7/2003
```

Por supuesto es posible también añadir una nueva propiedad con esta notación, así por ejemplo para añadir la propiedad fecha de nacimiento se escribe:

```
>>empleado1['fecha_nacimiento']='17/10/1900'
17/10/1900
```

Con lo que la estructura tiene ahora los siguientes campos (propiedades):

```
>>empleado1
{nombre:Juan Pérez, direccion:Calle Junín 345, celular:73242361,
fecha_ingreso:1/7/2003, ci:1092883, fecha_nacimiento:17/10/1900}
```

En general es más sencillo y claro emplear la notación de punto, sin embargo, cuando el nombre de la propiedad tiene espacios o es una palabra reservada, la única forma de acceder a la misma es mediante la notación de Arrays.

Para eliminar una propiedad (campo) se procede como con cualquier variable, es decir empleando el comando "delete". Por ejemplo para eliminar la propiedad "fecha_nacimiento" se escribe:

```
>>delete empleado1.fecha_nacimiento
true
```

Con lo que la estructura "empleado1" tiene ahora los campos:

```
>>empleado1
{nombre:Juan Pérez, direccion:Calle Junín 345, celular:73242361,
fecha_ingreso:1/7/2003, ci:1092883}
```

Sin la notación JSON, es posible también crear estructuras (array asociativos), pero el proceso es más moroso. Por ejemplo, para crear otro objeto empleado sin JSON, se escribe:

```
>>empleado2=new Object()
{}
>>empleado2.nombre="Carlos Cardozo"
Carlos Cardozo
>>empleado2.direccion="Calle Loa 532"
Calle Loa 532
>>empleado2.celular=58347832
58347832
>>empleado2.fecha_ingreso="1/3/2006"
1/3/2006
>>empleado2.ci=10238932
10238932
```

Con lo que la estructura (el objeto) empleado2 tiene los campos:

```
>>empleado2
{nombre:Carlos Cardozo, direccion:Calle Loa 532, celular:58347832,
fecha_ingreso:1/3/2006, ci:10238932}
```

Por supuesto, el mismo resultado se puede conseguir empleando la notación de Arrays:

```
>>empleado3=new Object()
{}
>>empleado3["nombre"]="Juan Mendez"
Juan Mendez
>>empleado3["direccion"]="Calle Junín 654"
Calle Junín 654
>>empleado3["celular"]=72847832
72847832
>>empleado3["fecha_ingreso"]="1/5/2001"
1/5/2001
>>empleado3["ci"]=10238932
10238932
>>empleado3
{nombre:Juan Mendez, direccion:Calle Junín 654, celular:72847832,
ci:10238932, fecha_ingreso:1/5/2001}
```

Es necesario aclarar que la posibilidad de añadir y quitar campos en una estructura, después de haber sido creada, es una característica especial de Javascript. La mayoría de los lenguajes (como C, C++, Java, Hecl, RPL, Pascal, etc.) no permiten añadir ni quitar campos de una estructura, una vez que ha sido creada.

## 6.2. FUNCIONES CONSTRUCTORAS (CONSTRUCTORES)

Cuando se quiere guardar un conjunto de datos que tienen la misma estructura, es decir que tienen los mismos campos, no resulta práctico tener que escribir repetidamente los nombres de los campos para cada nuevo dato y no sólo porque es moroso, sino también porque es un proceso propenso a errores.

Para evitar ello Javascript permite crear funciones constructoras (o simplemente "constructores"). Dichas funciones son las encargadas de asignar los valores que se le mandan a las propiedades del objeto (así como reservar memoria, asignar valores por defecto, validar datos y otras actividades que garanticen la consistencia de los datos).

Por ejemplo, para guardar los campos: nombre, dirección, curso y celular, de los alumnos de un colegio, se puede crear el siguiente constructor:

```
>>Alumno=function (nombre,direccion,curso,celular){  this.nombre = nombre,
 this.direccion = direccion,  this.curso = curso,  this.celular = celular }
function (nombre,direccion,curso,celular){
 this.nombre = nombre,
 this.direccion = direccion,
 this.curso = curso,
 this.celular = celular
}
```

La palabra "this" hace referencia al objeto (la estructura) que se está creando. Es importante entender que "this.nombre" es una variable diferente al parámetro "nombre": "this.nombre" es el campo "nombre" del objeto que se está creando, mientras que "nombre" es el dato que se manda a la función. No es necesario que los parámetros del constructor tengan los mismos nombres que los campos, es más, algunos autores recomiendan dar nombres diferentes a los parámetros para evitar confusión, mientras que otros recomiendan dar los mismos nombres y justamente por la misma razón.

Por lo tanto el constructor puede ser creado también con parámetros como los siguientes:

```
>>Alumno = function (_nombre, _direccion, _curso, _celular){  this.nombre =
_nombre,  this.direccion = _direccion,  this.curso = _curso,  this.celular
= _celular }
```

```
function (_nombre, _direccion, _curso, _celular){
 this.nombre = _nombre,
 this.direccion = _direccion,
 this.curso = _curso,
 this.celular = _celular
}
```

Otro detalle que es importante notar es que el nombre del constructor (de la función) comienza con una letra mayúscula. Esa no es una norma, sin embargo, es una práctica frecuente y recomendable pues de esa manera se puede distinguir entre una función normal y un constructor.

Una vez que creado el constructor puede ser empleado para crear registros con diferentes conjuntos de datos, como por ejemplo:

```
>>e1 = new Alumno("Juan Pérez","Arenales 345",5,783198347)
{nombre:Juan Pérez, direccion:Arenales 345, curso:5, celular:783198347}
>>e2 = new Alumno("Carlos Romero","Urcullo 478",7,48923482)
{nombre:Carlos Romero, direccion:Urcullo 478, curso:7, celular:48923482}
>>e3 = new Alumno("Amanda Rivas","España 38",8,63294235)
{nombre:Amanda Rivas, direccion:España 38, curso:8, celular:63294235}
```

En programación estructurada, a cada uno de estos datos se conoce con el nombre de "registro". La agrupación de dos o más registros constituye una "tabla" y la agrupación de dos o más tablas constituye una "base de datos".

Como se dijo, los constructores no sólo facilitan la creación de estructuras (registros), sino que al ser funciones, pueden ser empleadas para validar los datos y/o asignar valores por defecto. Por ejemplo, se puede modificar el constructor "Alumno" de manera que asigne valores por defecto en caso de no ser enviados:

```
>>Alumno=function (nombre,direccion,curso,celular){  this.nombre = nombre
|| "sin nombre",  this.direccion = direccion || "sin dirección",
 this.curso = curso || 1,  this.celular = celular || 0 }
function (nombre,direccion,curso,celular){
 this.nombre = nombre || "sin nombre",
 this.direccion = direccion || "sin dirección",
 this.curso = curso || 1,
 this.celular = celular || 0
}
```

Ahora, con este constructor, si no se mandan datos, se crea un registro con los valores por defecto:

```
>>e4 = new Alumno()
{nombre:sin nombre, direccion:sin dirección, curso:1, celular:0}
```

Como se recordará este constructor funciona porque en Javascript el operador "||" devuelve el primer valor que es interpretado como verdadero (es decir cualquier valor diferente de null, 0, undefined o "").

## 6.3. CAMPOS CONSTANTES

En ocasiones uno o más campos (propiedades) tienen los mismos valores para todos los registros (objetos). En esos casos no tiene sentido escribir dichos valores para cada registro, siendo una mejor alternativa crearlas como constantes.

En Javascript existe dos formas en las que se puede crear una propiedad constante: a) como un campo del constructor y b) como un campo de la propiedad "prototype".

La primera forma es la preferida y es la que se considera propiamente como constante en Javascript. Un ejemplo de este tipo de campo es el número PI, que como se recordará se encuentra en el objeto "Math": Math.PI.

Para crear un campo constante, simplemente se añade el mismo al constructor. Por ejemplo, dado que todos los alumnos pertenecen al mismo colegio, es conveniente crear un campo constante para el nombre del colegio:

```
>>Alumno.COLEGIO = "Franz Tamayo"
Franz Tamayo
```

Luego, para acceder a este valor, se escribe:

```
>>Alumno.COLEGIO
Franz Tamayo
```

La otra forma de crear un campo constante es mediante la propiedad "prototype", que a su vez es un objeto (un registro). Con "prototype", al igual que con el constructor el campo se crea una sola vez, pero se accede al mismo como si fuera un campo propio del registro. Por ejemplo, dado que la dirección del colegio es la misma para todos los alumnos, se puede crear una propiedad para el mismo:

```
>>Alumno.prototype.dircolegio = "San Alberto 423"
San Alberto 423
```

Entonces, para acceder a este campo desde cualquiera de los registros (por ejemplo, el registro "e1"), se escribe:

```
>>e1.dircolegio
San Alberto 423
```

Pero, aún cuando se accede a este campo igual que a cualquier otro, ha sido escrito una sola vez y sólo existe una copia del mismo. Observe también que ha sido posible acceder a esta propiedad desde un registro que ha sido creado cuando el constructor no tenía aún dicha propiedad. Esta es una característica importante de "prototype", cualquier propiedad (o método) que se añade a "prototype", queda inmediatamente disponible para todos los registros (objetos) creados con ese constructor, así como para todos los registros que se crean posteriormente con el mismo.

## 6.4. REGISTROS JERÁRQUICOS

Los campos de una estructura pueden ser de cualquier tipo válido, incluido otro registro. Cuando esto ocurre, es decir cuando uno o más campos son a su vez registros, se dice que son jerárquicos (los registros que no cuentan con este tipo de campos se denominan registros planos).

Un mismo conjunto de datos puede ser visto tanto como un registro plano como uno jerárquico, por ejemplo el nombre de una persona puede ser visto como un dato simple (de tipo texto) o como un registro con tres campos: apellido paterno, apellido materno y nombres, igualmente la fecha de nacimiento puede ser vista como un registro compuesto por tres campos: el día, el mes y el año, algo similar ocurre con la dirección, que puede ser vista como un registro compuesto por dos campos: el nombre de la calle y el número de la casa.

Así para guardar los datos de una persona en un registro jerárquico se crean primero los constructores para los registros internos:

```
>>Nombre = function(a_paterno,a_materno,nombres){  this.a_paterno =
a_paterno,  this.a_materno = a_materno,  this.nombres = nombres }
function (a_paterno,a_materno,nombres){
 this.a_paterno = a_paterno,
```

```
 this.a_materno = a_materno,
 this.nombres = nombres
}
>>Direccion = function(calle,numero){  this.calle = calle,  this.numero =
numero }
function (calle,numero){
 this.calle = calle,
 this.numero = numero
}
>>Fecha = function(dia,mes,año){  this.dia = dia,  this.mes = mes,
 this.año = año }
function (dia,mes,año){
 this.dia = dia,
 this.mes = mes,
 this.año = año
}
```

Con estos constructores es posible ahora crear el registro jerárquico:

```
>>Persona = function(ci,nombre,direccion,f_nacimiento){  this.ci = ci,
 this.nombre = nombre,  this.direccion = direccion,  this.f_nacimiento =
f_nacimiento }
function (ci,nombre,direccion,f_nacimiento){
 this.ci = ci,
 this.nombre = nombre,
 this.direccion = direccion,
 this.f_nacimiento = f_nacimiento
}
```

Con este constructor se pueden crear registros jerárquico, como por ejemplo:

```
>>nombre = new Nombre("Juan","Perez","Romero")
{a_paterno:Juan, a_materno:Perez, nombres:Romero}
>>direccion = new Direccion("Avenida Jaime Mendoza",678)
{calle:Avenida Jaime Mendoza, numero:678}
>>f_nacimiento = new Fecha(4,10,1997)
{dia:4, mes:10, año:1997}
>>p1 = new Persona(10827374,nombre,direccion,f_nacimiento)
{ci:10827374, nombre:{a_paterno:Juan, a_materno:Perez, nombres:Romero}, di-
reccion:{calle:Avenida Jaime Mendoza, numero:678}, f_nacimiento:{dia:4,
mes:10, año:1997}}
```

Como se puede ver los campos nombre, dirección y f_nacimiento, muestran la información que contienen entre llaves, lo que confirma que se tratan de registros (objetos). Para acceder a uno de los campos de este registro se procede de la forma habitual, así por ejemplo, para obtener la dirección de "p1" se escribe:

```
>>p1.direccion
{calle:Avenida Jaime Mendoza, numero:678}
```

Y para acceder a uno de los subcampos, simplemente se extiende esta nomenclatura, por ejemplo, para obtener el año de nacimiento de "p1", se escribe:

```
>>p1.f_nacimiento.año
1997
```

Que se interpreta como de costumbre, de derecha a izquierda, es decir es el "año" del campo "f_nacimiento" de la persona "p1".

Es posible crear directamente el constructor para una persona, sin crear previamente los constructores para los campos internos:

```
>>Persona_ =
function(ci,a_paterno,a_materno,nombres,calle,numero,dia,mes,año){  this.ci
= ci,  this.nombre ={a_paterno:a_paterno, a_materno:a_materno, nombres:nom-
bres},  this.direccion = {calle:calle, numero:numero}  this.f_nacimiento =
{dia:dia, mes:mes, año:año} }
function (ci,a_paterno,a_materno,nombres,calle,numero,dia,mes,año){
 this.ci = ci,
 this.nombre ={a_paterno:a_paterno, a_materno:a_materno, nombres:nombres},
 this.direccion = {calle:calle, numero:numero}
 this.f_nacimiento = {dia:dia, mes:mes, año:año}
}
```

Como se puede ver, en este caso los registros internos no tienen la palabra "this" para diferenciarlos de los nombres de los parámetros y no es necesario, porque Javascript sabe que el valor antes de los dos puntos es el nombre del campo y que el valor después de los dos puntos es su valor.

Con este constructor se crean los mismos campos que con "Persona", pero mandando los datos de todos los campos (incluidos subcampos) por separado, como por ejemplo:

```
>>new
Persona_(1082834,"Gómez","Salazar","Mariela","Estudiantes",67,23,5,1991)
{ci:1082834, nombre:{a_paterno:Gómez, a_materno:Salazar, nombres:Mariela},
direccion:{calle:Estudiantes, numero:67}, f_nacimiento:{dia:23, mes:5,
año:1991}}
```

Si bien en primera instancia los registros jerárquicos pueden ser vistos como una forma más eficiente y ordenada de organizar la información, en la práctica dificultan el acceso a la misma y constituyen estructuras rígidas que complican innecesariamente su expansión y/o modificación. Es por esta razón que en la práctica la mayoría de las bases de datos (más del 80%) están constituidas por registros planos. Los registros jerárquicos funcionan bien en pequeñas aplicaciones y/o aplicaciones donde es poco probable que la estructura de información sea ampliada o modificada.

## 6.5. OBJETOS

Como se ha apuntado en el anterior capítulo, los campos de una estructura pueden ser de cualquier tipo, incluido funciones. Cuando esto ocurre, es decir cuando uno o más campos de una estructura son funciones, la estructura se conoce como una "clase" y las variables que se crean con esa estructura se conocen como "objetos".

Aun cuando no es obligatorio, dichas funciones (denominadas métodos) deberían operar sobre los datos (propiedades) del objeto.

Por ejemplo se puede definir una estructura para guardar los datos de un rectángulo. Los datos propiamente son los puntos correspondientes a la esquina superior izquierda y a la esquina inferior derecha (ambas a su vez registros). Las otras "propiedades" como la longitud de los lados, el perímetro y el área se pueden calcular a partir de los dos puntos, por lo que se implementan como métodos:

```
>>Rectangulo = function(x1,y1,x2,y2){  this.p1 = {x:x1,y:y1},  this.p2 =
{x:x2,y:y2},  this.alto = function(){return this.p2.y-this.p1.y;},
 this.ancho = function(){return this.p2.x-this.p1.x;},  this.perimetro =
function(){return 2*(this.ancho()+this.alto());},  this.area = function()
{return this.ancho()*this.alto();} }
function (x1,y1,x2,y2){
 this.p1 = {x:x1,y:y1},
```

```
 this.p2 = {x:x2,y:y2},
 this.alto = function(){return this.p2.y-this.p1.y;},
 this.ancho = function(){return this.p2.x-this.p1.x;},
 this.perimetro = function(){return 2*(this.ancho()+this.alto());},
 this.area = function(){return this.ancho()*this.alto();}
}
```

Ahora, si se crea un objeto con este constructor:

```
>>r1 = new Rectangulo(10,10,30,20)
{p1:{x:10, y:10}, p2:{x:30, y:20}, alto:function (){return this.p2.y-
this.p1.y;}, ancho:function (){return this.p2.x-this.p1.x;},
perimetro:function (){return 2*(this.ancho()+this.alto());}, area:function
(){return this.ancho()*this.alto();}}
```

Se puede averiguar el alto, ancho, perímetro y área del mismo llamando a los métodos del objeto:

```
>>r1.alto()
10
>>r1.ancho()
20
>>r1.perimetro()
60
>>r1.area()
200
```

Sin embargo, esta solución es ineficiente pues cada vez que se crea un nuevo objeto, se crean también nuevas funciones para el mismo, es decir que si se crea un nuevo objeto, como el siguiente:

```
>>r2 = new Rectangulo(10,20,100,200)
{p1:{x:10, y:20}, p2:{x:100, y:200}, alto:function (){return this.p2.y-
this.p1.y;}, ancho:function (){return this.p2.x-this.p1.x;},
perimetro:function (){return 2*(this.ancho()+this.alto());}, area:function
(){return this.ancho()*this.alto();}}
```

No sólo se crean nuevos campos (con los nuevos valores), sino también nuevos métodos ("ancho", "alto", "perimetro" y "area") a pesar de ser exactamente los mismos que para el objeto "r1". Para evitar este comportamiento ilógico los métodos deben ser añadidos a la propiedad "prototype" del constructor (tal como sucede con los campos constantes). De esa manera se crea una sola copia de la función la cual es compartida (heredada) por todos los objectos creados con ese constructor.

Procediendo de esa forma, la versión optimizada del constructor se crea con:

```
>>Rectangulo = function(x1,y1,x2,y2){  this.p1 = {x:x1,y:y1},  this.p2 =
{x:x2,y:y2} }
function (x1,y1,x2,y2){
 this.p1 = {x:x1,y:y1},
 this.p2 = {x:x2,y:y2}
}
```

Al cual se añaden las funciones (métodos) en la propiedad prototype:

```
>>Rectangulo.prototype.alto = function(){return this.p2.y-this.p1.y;}
function (){return this.p2.y-this.p1.y;}
>>Rectangulo.prototype.ancho = function(){return this.p2.x-this.p1.x;}
function (){return this.p2.x-this.p1.x;}
>>Rectangulo.prototype.perimetro = function(){return 2*(this.ancho()
+this.alto());}
function (){return 2*(this.ancho()+this.alto());}
```

```
>>Rectangulo.prototype.area = function(){return this.ancho()*this.alto();}
function (){return this.ancho()*this.alto();}
```

Ahora, cuando se crea un nuevo objeto con este constructor:

```
>>r4
{p1:{x:1, y:2}, p2:{x:3, y:4}, alto:function (){return this.p2.y-
this.p1.y;}, ancho:function (){return this.p2.x-this.p1.x;},
perimetro:function (){return 2*(this.ancho()+this.alto());}}
```

Aparentemente se han creado nuevamente las funciones (pues aparecen en el resultado), pero en realidad no es así, estas funciones (métodos) son las añadidas a la propiedad "prototype", las cuales, al ser heredadas, son mostradas como si pertenecieran al objeto.

Esto hecho se puede comprobar añadiendo un nuevo método a la propiedad "prototype" del constructor. Por ejemplo se puede añadir un método para calcular la diagonal del rectángulo:

```
>>Rectangulo.prototype.diagonal = function(){return sqrt(sqr(this.alto())
+sqr(this.ancho()));}
function (){return sqrt(sqr(this.alto())+sqr(this.ancho()));}
```

Y como este nuevo método pertenece a la propiedad "prototype" del constructor es heredado automáticamente por todos los objetos creados con el mismo, por lo tanto puede ser empleado desde el objeto "r4" (a pesar de haber sido añadido después de crear el objeto "r4"):

```
>>r4.diagonal()
2.8284271247461903
```

Cuando se ven los elementos del objeto "r4":

```
>>r4
{p1:{x:1, y:2}, p2:{x:3, y:4}, alto:function (){return this.p2.y-
this.p1.y;}, ancho:function (){return this.p2.x-this.p1.x;},
perimetro:function (){return 2*(this.ancho()+this.alto());},
diagonal:function (){return sqrt(sqr(this.alto())+sqr(this.ancho()));}}
```

Se muestra también el método diagonal como parte del mismo, con lo que se demuestra que los métodos añadidos a la propiedad "prototype" son reportados como parte de los objetos creados con el constructor, pero en realidad sólo existe una copia de dichos métodos y la misma se encuentra en la propiedad "prototype" del constructor.

## 6.6. ESTRUCTURAS Y FUNCIONES (PROGRAMACIÓN ESTRUCTURADA)

En la programación estructurada, a diferencia de la orientada a objetos, las estructuras de datos y las funciones no constituyen una unidad. Las estructuras se crean por un lado y las funciones que operan con dichos datos por otro.

Ambas metodologías permiten llegar a los mismos resultados, pero analizando y resolviendo el problema desde diferentes puntos de vista. En la programación estructurada se analiza el problema pensando por un lado en las funciones (tareas) que debe llevar a cabo la aplicación y por otro en los datos que se requieren para realizar dichas funciones. En la programación orientada a objetos en cambio, se piensa primero en los objetos que se requieren para resolver el problema y luego, para cada objeto, se piensa en las propiedades (datos) y métodos (funciones) que deben operar sobre dichas propiedades.

Por ejemplo la solución que se ha dado en el anterior acápite para el problema del rectángulo, puede ser encontrada también siguiendo la metodología estructurada.

En el enfoque estructurado se piensa primero en las funciones que se requieren al trabajar con rectángulos. Dichas funciones serían las encargadas de calcular el ancho, alto, perímetro, área y la diagonal de un rectángulo. Luego se piensa en los datos que se requieren para llevar a cabo dichos cálculos.

Dado que un rectángulo queda completamente definido con los puntos que se encuentran en la esquina superior izquierda y la esquina inferior derecha se requiere un dato estructurado compuesto por estos dos puntos, es decir:

```
>>Rectangle = function(x1,y1,x2,y2){  this.p1 = {x:x1, y:y1},  this.p2 =
{x:x2, y:y2} }
function (x1,y1,x2,y2){
 this.p1 = {x:x1, y:y1},
 this.p2 = {x:x2, y:y2}
}
```

Que en primera instancia es prácticamente el mismo constructor empleado para crear los objetos en el anterior acápite, sin embargo, en este caso este constructor sólo tiene datos (campos), por lo que es un constructor de registros y no de objetos.

Ahora (por separado) se crean las funciones que se encargan de llevar a cabo las tareas antes mencionadas:

```
>>ancho = function(r){  return r.p2.x-r.p1.x; }
function (r){
 return r.p2.x-r.p1.x;
}
>>alto = function(r){  return r.p2.y-r.p1.y; }
function (r){
 return r.p2.y-r.p1.y;
}
>>perimetro = function(r){  return 2*ancho(r)*alto(r); }
function (r){
 return 2*ancho(r)*alto(r);
}
>>area = function(r){  return ancho(r)*alto(r); }
function (r){
 return ancho(r)*alto(r);
}
>>diagonal = function(r){  return sqrt(sqr(ancho(r))+sqr(alto(r))); }
function (r){
 return sqrt(sqr(ancho(r))+sqr(alto(r)));
}
```

Con lo que el problema estaría resuelto. Para probar esta solución se crea un rectángulo con los mismos puntos que el objeto "r1" del anterior acápite:

```
>>r0 = new Rectangle(10,10,30,20)
{p1:{x:10, y:10}, p2:{x:30, y:20}}
```

Ahora, con "r0" y las funciones creadas se calcula el alto, ancho, perímetro, área y diagonal de este rectángulo:

```
 >>alto(r0)
10
>>ancho(r0)
20
```

```
>>perimetro(r0)
60
>>area(r0)
200
>>diagonal(r0)
22.360679774997898
```

Obteniéndose, como era de esperar, los mismos resultados que en la solución con objetos.

Como se ha comprobado en el anterior ejemplo, el mismo problema puede se resuelto tanto desde el punto de vista estructurado como orientado a objetos y obteniendo en ambos casos los mismos resultados.

Entonces ¿qué enfoque se debe emplear para resolver un problema?, la respuesta a esta pregunta depende de varios factores: a) La naturaleza del problema: existen problemas que por su naturaleza se resuelven de forma más eficiente con el enfoque estructurado, mientras que otros con el enfoque orientado a objetos; b) Las herramientas disponibles: algunas herramientas sólo soportan uno de los enfoques en cuyo caso, si se está empleando una de esas herramientas no queda otra alternativa que resolver el problema con el enfoque soportado por la misma; c) El tamaño del proyecto: aunque en teoría la programación orientada a objetos facilita la elaboración de grandes proyectos, en la práctica el esfuerzo requerido en su mantenimiento incrementa exponencialmente a medida que el proyecto crece, por lo que para proyectos grandes (como los sistemas operativos) suele ser una mejor opción la programación estructurada; d) La preferencia del desarrollador: este es en última instancia el factor determinante, porque es el desarrollador quien elegirá una u otro enfoque en función a su preferencia y experiencia con uno u otro enfoque.

Son varias las ventajas y desventajas que los defensores de uno y otro enfoque mencionan, pero como ya se dijo la elección final es sobre todo una cuestión de preferencia, pues con ambas metodologías se pueden resolver los mismos problemas.

Una de las desventajas que se señala con relación a la programación estructurada es que las funciones pasan a forma parte del espacio de nombres de las variables globales, con lo que incrementa la probabilidad de que los nombres de dos o más variables colisionen. Este problema, sin embargo, puede ser minimizado declarando las funciones en un espacio de nombres.

Un espacio de nombres es simplemente una estructura, inicialmente vacía, a la cual se le añaden las funciones, estructuras, variables, etc., que se quiere pertenezcan a este espacio de nombres. De esa manera el único nombre que pasa al espacio de las variables globales es el nombre de la estructura.

Por ejemplo, el constructor "Rectangle" y sus funciones pueden ser colocados en un espacio de nombres, por ejemplo un espacio de nombres denominado "geometria", que se crea con:

```
>>gemoetria = {}
{}
```

O también con:

```
>>geometria = new Object()
{}
```

Una vez creado el espacio de nombres se añaden al mismo el constructor y las funciones:

```
>>geometria.Rectangle = function(x1,y1,x2,y2){  this.p1 = {x:x1, y:y1},
```

```
 this.p2 = {x:x2, y:y2} }
function (x1,y1,x2,y2){
 this.p1 = {x:x1, y:y1},
 this.p2 = {x:x2, y:y2}
}
>>geometria.ancho = function(r){  return r.p2.x-r.p1.x; }
function (r){
 return r.p2.x-r.p1.x;
}
>>geometria.alto = function(r){  return r.p2.y-r.p1.y; }
function (r){
 return r.p2.y-r.p1.y;
}
>>geometria.perimetro = function (r){  return 2*ancho(r)*alto(r); }
function (r){
 return 2*ancho(r)*alto(r);
}
>>geometria.area = function (r){  return ancho(r)*alto(r); }
function (r){
 return ancho(r)*alto(r);
}
>>geometria.diagonal = function (r){  return sqrt(sqr(ancho(r))
+sqr(alto(r))); }
function (r){
 return sqrt(sqr(ancho(r))+sqr(alto(r)));
}
```

Por supuesto, para acceder a estas funciones, se debe preceder su nombre con el nombre del espacio de nombres. Por ejemplo con la siguiente instrucción se crea un nuevo rectángulo y se calcula su área:

```
>>r5 = new geometria.Rectangle(50,50,200,200)
{p1:{x:50, y:50}, p2:{x:200, y:200}}
>>geometria.area(r5)
22500
```

Se puede evitar el reescribir el nombre del espacio de nombres para cada variable y/o función empleando la estructura "with":

```
with (nombre_del_espacio) {
  instrucciones
}
```

Así con esta estructura, se puede crear otro rectángulo y calcular la altura, perímetro, área y diagonal del mismo:

```
>>with (geometria) {  r6 = new Rectangle(20,30,200,250);  write(alto(r6));
 write(perimetro(r6));  write(area(r6));  write(diagonal(r6)); }
220
79200
39600
284.2534080710379
```

Como se puede ver, al interior de la estructura "width" no se requiere ahora el nombre del espacio de nombres ("geometria") para acceder a sus elementos (Rectangle, alto, area, etc.). La función "write" nada tiene que ver con los espacios de nombre, ha sido empleada simplemente para imprimir los resultados intermedios.

Se debe aclarar que la estructura "with" funciona no sólo con espacios de nombres, sino con cualquier objeto, no obstante es importante tomar en cuenta que su uso disminuye la velocidad de ejecución de los programas, por lo que debe ser empleada con moderación.

Aunque, como se dijo, la selección del enfoque es sobre todo una cuestión de preferencia, cuando el problema es por su naturaleza estructurado u orientado a objetos, lo recomendable es resolver el problema con ese enfoque.

Un ejemplo lo constituyen las expresiones matemáticas, que por su naturaleza son estructuradas: en matemáticas se tienen por un lado los datos (números naturales, enteros, reales, complejos, etc.) y por otro las funciones que operan sobre los mismos (suma, resta, multiplicación, etc.) es por ello que prácticamente el 100% de las soluciones profesionales en este campo son estructuradas. Mientras que las soluciones orientadas a objetos, en este campo, no han tenido éxito y no porque no sea posible resolver el problema con ese enfoque, sino porque la solución (al no ser natural) es difícil de aplicar en los problemas prácticos: por ejemplo en la suma, sin importar qué se sume, se escribe el primer dato, el operador de suma ("+") y el segundo dato, no (como requiere la programación orientada a objetos) el dato, un punto, el nombre del operador y el segundo dato entre paréntesis y la situación se complica más aún si los datos no son del mismo tipo, pues entonces se debe recurrir además al moldeo de tipos.

## 6.7. GENERACIÓN ALEATORIA DE DATOS NO NUMÉRICOS

Para probar las aplicaciones que trabajan con registros, con frecuencia se requieren conjuntos de datos no numéricos como apellidos, nombres de personas, nombres de artículos, nombres de calles, etc. El introducir estos datos manualmente no es una alternativa viable, pues por una parte consume demasiado tiempo y por otra, cuando se produce un error y eso es algo que ocurre en prácticamente el 100% de las aplicaciones en desarrollo, sería necesario volver a introducirlos. Por esta razón, la alternativa más viables consiste en generarlos aleatoriamente.

Para generar un conjunto de datos no numéricos de forma aleatoria se debe crear primero un domino del cual se extraigan al azar los datos.

La forma más sencilla y práctica de crear un dominio es empleando Arrays, por ejemplo con el siguiente Array, se crea un dominio con 45 apellidos:

```
>>d_apellidos_=["Abad","Ajata","Alsina","Cadenas","Aragón","Arteaga","Amor"
,"Barberá","Belda","Bonilla","Cadenas","Cantuta","Casado","Cervantes","Choq
uemamani","Comas","Cuervo","Elorza","Ferrán","Galiano","Gisbert","Guijarro"
,"Huguet","Janco","Ledesma","Lucas","Marí","Menéndez","Morales","Nieto","Pa
csipati","Paucarmayta","Pilco","Portero","Quevedo","Rico","Rosell","Salvà",
"Sevilla","Sureda","Tomé","Uyardo","Verdugo","Yáñez","Zurita"]
[Abad, Ajata, Alsina, Cadenas, Aragón, Arteaga, Amor, Barberá, Belda,
Bonilla, Cadenas, Cantuta, Casado, Cervantes, Choquemamani, Comas, Cuervo,
Elorza, Ferrán, Galiano, Gisbert, Guijarro, Huguet, Janco, Ledesma, Lucas,
Marí, Menéndez, Morales, Nieto, Pacsipati, Paucarmayta, Pilco, Portero,
Quevedo, Rico, Rosell, Salvà, Sevilla, Sureda, Tomé, Uyardo, Verdugo,
Yáñez, Zurita]
```

Con este dominio se pueden crear una función que genere al azar uno de estos apellidos. Para ello simplemente se genera un número entero aleatorio comprendido entre 0 y el número de elementos del dominio menos 1 devolviendo el elemento (el apellido) correspondiente a este número generado:

```
>>generarApellido=function(){  return
d_apellidos_[round(random(0,d_apellidos_.length-1))]; }
function (){
 return d_apellidos_[round(random(0,d_apellidos_.length-1))];
}
```

Llamando a la función unas cuantas veces se obtienen algunos apellidos aleatorios:

```
>>generarApellido()
Verdugo
>>generarApellido()
Sevilla
>>generarApellido()
Bonilla
>>generarApellido()
Marí
>>generarApellido()
Cantuta
```

Mientras mayor sea el número de elementos en el dominio más cercanos a los valores serán los elementos generados.

De la misma forma se pueden generar datos aleatorios para otros dominios como: nombres de artículos, marcas de automóviles, nombres de países, etc.

En la librería SIS101.js se han añadido dominios y funciones generadoras para apellidos: generarApellido(); nombres femeninos: generarNombreF(); nombres masculinos: generarNombreM() y nombres de calles: generarCalle(), los mismos que pueden ser empleados para generar registros al azar. En esta librería (o en otra propia) se pueden añadir los dominios y funciones generadoras que se requieran.

## 6.8. FORMULARIOS

Cuando se trabaja con estructuras de datos, es frecuente que dichos datos sean introducidos en formularios.

Los formularos en una página HTML son simplemente contenedores (similares a los elementos ) en cuyo interior se incluyen los elementos que permiten introducir (o seleccionar) los campos del registro. Se emplean porque cuentan con algunos métodos que facilitan el paso de datos desde y hacia las bases de datos ubicadas en los servidores, no obstante, con el advenimiento de la tecnología asíncrona (AJAX), que permite una comunicación interactiva con los servidores, la utilidad de dichos métodos ha disminuido, razón por la cual actualmente no es un requisito que los datos de un registro se encuentren dentro de un formulario, pudiendo estar en cualquier otro contenedor o en ninguno.

En ocasiones inclusive se evita el uso de formularios para que los métodos automáticos asociados a los mismos no interfieran con los métodos elaborados.

Para crear un formulario se emplean las etiquetas "<form>...</form>". Por ejemplo, en la siguiente página HTML, se crea un formulario básico para introducir los datos correspondiente a un directorio telefónico. Se hace notar que el formulario de este ejemplo es sólo la interfaz de usuario de la aplicación, no tiene integrada ninguna funcionalidad, por lo que en realidad no hace nada:

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Formulario</title>
  <style type="text/css">
    form {
      width:250px;
```

```
        height:145px;
        border-style:ridge;
        border-width:5px;
        border-color:lightblue;
        background-color:lightblue;
        padding: 10px;
      }
      label {
        position:absolute;
        width:70px;
        color:blue;
        font-weight:bold;
        text-align:right;
      }
      input {
        position:absolute;
        text-color:green;
        font-weight:bold;
        left:100px;
      }
    </style>
  </head>
  <body>
    <form id="forma1">
      <!--Campo para introducir el nombre-->
      <label for="nombre" style="top:35px;">Nombre:</label>
      <input type="text" id="nombre" style="top:31px;"><br>
      <!--Campo para introducir la dirección-->
      <label for="direccion" style="top:60px;">Direccion:</label>
      <input type="text" id="direccion" style="top:56px;"></input><br>
      <!--Campo para introducir el número de teléfono-->
      <label for="telefono" style="top:85px;">Telefono:</label>
      <input type="text" id="telefono" style="top:81px;width:70px;"><br>
      <!-- Campo para seleccionar el sexo-->
      <label style="width:45px;top:110px;">Sexo:</label>
      <input type="radio" name="sexo" id="sexo1" checked
        style="top:110px;left:70px;cursor:pointer;">
      <label for="sexo1" style="top:110px;left:95px;cursor:pointer">
        Masculino</label>
      <input type="radio" name="radio" id="sexo2"
        style="top:110px;left:165px;cursor:pointer">
      <label for="sexo2" style="top:110px;left:180px;cursor:pointer;">
        Femenino</label><br>
      <!--Botón para mandar los datos al servidor-->
      <input type="submit" id="guardar" value="Guardar"
  style="top:140px;left:115px;cursor:pointer;">
    </form>
  </body>
  </html>
```

En este caso los estilos CSS se han aplicado en dos formas: a) Algunas propiedades han sido fijadas dentro de la etiqueta <style> del encabezado de la página y b) las propiedades específicas de cada elemento han sido asignadas en la etiqueta del elemento (con el atributo "style").

Observe que las propiedades comunes a los elementos "label" e "input", así como todas las propiedades del elemento "form" (que es el único elemento "form" en esta página) han sido asignadas dentro de la etiqueta <style>.

En realidad, dentro de la etiqueta <style> es posible asignar todas la propiedades de todos los elementos de la página, pues CSS cuenta con varias mecanismos para seleccionar elementos específico, por ejemplo, si un elemento tiene asignado un identificador (se le ha asignado el atributo "id"), puede ser seleccionar con el símbolo "#" seguido de dicho "id". Esta es la forma que recomiendan varios autores (pues de esa manera se separan las instrucciones HTML de los estilos CSS).

En este ejemplo, sin embargo, sólo se asignan las propiedades comunes a todos los elementos dentro de la etiqueta "style", mientras que las propiedades específicas de cada elemento se asignan dentro de las etiquetas de los elementos.

Hasta ahora los elementos HTML han sido dejados tal como los presenta el navegador. Ello es suficiente para una aplicación de uso personal, pero no para una aplicación comercial o institucional, donde los usuarios esperan ver los elementos alineados y correctamente distribuidos.

Para alinear los elementos en una página existe, como casi siempre ocurre en programación, diferentes alternativas, en este ejemplo se ha fijado la propiedad "position" de los objetos "label" e "input" en "absolute". Cuando un elemento está en posición absoluta, puede ser colocado en una posición específica con relación a su contenedor. Para ello se emplean principalmente las propiedades "left" (izquierda), "top" (arriba), "height" (alto) y "width" (ancho) (tal como se puede ver en el ejemplo).

Abriendo la página HTML en un navegador se obtiene algo parecido a:


Observe también que el puntero cambia a una mano con un dedo apuntando cuando se encuentra sobre el botón o las opciones "Masculino" o "Femenino". Esto se debe a que se ha asignado el valor "pointer" a la propiedad "cursor" de dichos elementos. Existen varios tipos de cursores que se pueden emplear en esta propiedad (crosshair, e-resize, w-resize, n-resize, s-resize, help, move, progress, text, wait, etc.) mismos que constituyen una ayuda visual para los usuarios, por lo que es conveniente emplearlos en las aplicaciones.

Por otra parte, el botón que aparece en el formulario es de tipo "submit", este botón envía los valores de los campos al servidor Internet (algo que se estudiará en temas posteriores).

## 6.9. EJEMPLO

**1. Elabore una aplicación que genere 300 registros aleatorios con los siguientes campos: nombre completo, dirección, teléfono, carnet de identidad y fecha de nacimiento. La aplicación debe mostrar los registros en un formulario y permitir recorrerlos hacia adelante o hacia atrás uno por uno, así como ir al primer registro o al último y a un número de re-**

**gistro específico.**

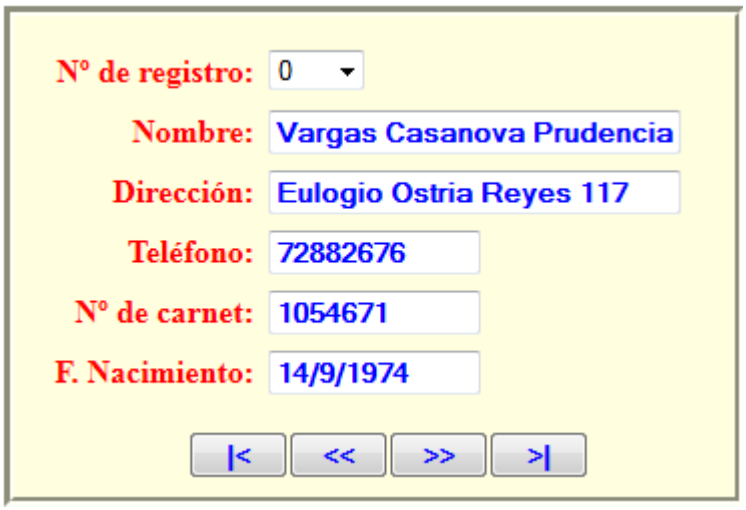
El código elaborado para resolver el problema es el siguiente:

```
<!DOCTYPE html>
<html>
<head>

  <meta name="vieport" content="width=device-width, initial-scale=1">

  <title>Ejemplo 1</title>

  <script src="sis101.js"></script>
  <script>

    function Registro(nombre,direccion,telefono,ci,f_nacimiento){
      this.nombre=nombre;
      this.direccion=direccion;
      this.telefono=telefono;
      this.ci=ci;
      this.f_nacimiento=f_nacimiento;
    }

    function generarRegistro(){
      var apellido1=generarApellido();
      var apellido2=generarApellido();
      var nombre1= round(random(1)) ? generarNombreF():generarNombreM();
      var nombre=apellido1+" "+apellido2+" "+nombre1;
      var direccion=generarCalle()+" "+round(random(1,1000));
      var telefono=round(random(30000,90000))+72800000;
      var ci=round(random(10000,99999))+1000000;
      var f_nacimiento=round(random(1,28))+"/"+round(random(1,12))+
        "/"+round(random(1970,2000));
      return new Registro(nombre,direccion,telefono,ci,f_nacimiento);
    }

    function mostrarRegistro(r){
      document.getElementById("numreg").value=ra;
      document.getElementById("nombre").value=r.nombre;
      document.getElementById("direccion").value=r.direccion;
      document.getElementById("telefono").value=r.telefono;
      document.getElementById("ci").value=r.ci;
      document.getElementById("f_nacimiento").value=r.f_nacimiento;
    }

    function inicializar(){
      nr=300;//Número de registros
      var txt="";
      for (var i=0;i<nr;i++){
        txt+="<option value="+i+">"+i+"</option>"
      }
      document.getElementById("numreg").innerHTML=txt;
      registro = new Array(300);//Registros generados
      for (var i=0;i<nr;i++)
        registro[i]=generarRegistro();
      ra=0;//Número de registro actual
      mostrarRegistro(registro[ra]);
    }
```

```
    function primerRegistro(){
      ra=0;
      mostrarRegistro(registro[ra]);
    }

    function siguienteRegistro(){
      ra = (ra<nr-1) ? ra+1 : 1;
      mostrarRegistro(registro[ra]);
    }

    function anteriorRegistro(){
      ra = (ra>0) ? ra-1 : nr-1;
      mostrarRegistro(registro[ra]);
    }

    function ultimoRegistro(){
      ra=nr-1;
      mostrarRegistro(registro[ra]);
    }

    function enesimoRegistro(n){
      ra=n;
      mostrarRegistro(registro[ra]);
    }

  </script>

  <style type="text/css">
    form {
      width:340px;
      height:220px;
      background-color:lightyellow;
      border-width:5px;
      border-style:groove;
      border-color:lightyellow;
      padding:10px;
    }
    label {
      position:absolute;
      color:red;
      font-weight:bold;
      width:110px;
      text-align:right;
    }
    input {
      position:absolute;
      color:blue;
      font-weight:bold;
      left:140px;
    }
    input[type=button]{
      top:220px;
      width:50px;
      cursor:pointer;
    }
  </style>
```

```
</head>
<body onload="inicializar()">
  <form id="form1">

    <label for="numreg" style="top:30px;">N° de registro:</label>
    <select id="numreg" style="position:absolute;top:30px;left:140px;"
      onchange="enesimoRegistro(parseInt(this.value))">
    </select><br>

    <label for="nombre" style="top:60px;">Nombre:</label>
    <input type="text" id="nombre" style="top:60px;width:200px;"><br>

    <label for="direccion" style="top:90px;">Dirección:</label>
    <input type="text" id="direccion"
      style="top:90px;width:200px;"><br>

    <label for="telefono" style="top:120px;">Teléfono:</label>
    <input type="text" id="telefono" style="top:120px;width:100px;"><br>

    <label for="ci" style="top:150px;">N° de carnet:</label>
    <input type="text" id="ci" style="top:150px;width:100px;"><br>

    <label for="f_nacimiento" style="top:180px;">F. Nacimiento:</label>
    <input type="text" id="f_nacimiento"
      style="top:180px;width:100px;"><br>

    <input type="button" value="|<" style="left:100px;"
      onclick="primerRegistro()">

    <input type="button" value="<<" style="left:150px;"
      onclick="anteriorRegistro()">

    <input type="button" value=">>" style="left:200px;"
      onclick="siguienteRegistro()">

    <input type="button" value=">|" style="left:250px;"
      onclick="ultimoRegistro()">

  </form>
</body>
</html>
```

Que al ser abierta en un navegador, tiene la siguiente apariencia:




Como se puede ver, para navegar a través de los registros se han empleado cuatro botones.

Para mostrar el número de registro actual, como para ir a un registro específico se ha empleado un "select" en cuyo evento "onchange" se ha programado el salto al registro respectivo. Las opciones de este elemento (las 300 opciones) se generan en la función "inicializa" que es llamada cuando la página termina de ser cargada (evento onload de "body").

Igualmente, los 300 registros son generados en la función "inicializar" llamando a la función "generarRegistro", la cual a su vez hace uso del constructor "Registro" y de las funciones "generarApellido", "generarNombreF", "generarNombreM" y "generarCalle" de la librerías "sis101.js". Los números de teléfono, carnet de identidad y la fecha de nacimiento, al ser números, se generan con la función "random".

Finalmente, para mostrar los resultados en el formulario se llama a la función "mostrarRegistro" que recibe como dato el registro a mostrar.

Observe que las variables "nr" (número de registros), "ra" (registro actual) y "registro" (el vector con los registros generados) son variables globales (no han sido declaradas con "var"), se ha procedido así para que dichas variables puedan ser empleadas desde otras funciones (como "primerRegistro", "siguienteRegistro", etc.)

En cuanto a los estilos, con la intención de demostrar uno de los mecanismos de selección de CSS, se ha empleado "input[type=button]" con esta instrucción se crea un estilo para los elementos "input" que tengan el atributo "type" igual a "button" (que en este ejemplo son los botones de navegación). Este mecanismo de selección se puede emplear con cualquier atributo y puede tener un valor asignado (como en el ejemplo) o simplemente el nombre del atributo, por ejemplo si se escribe "label[for]" el estilo se aplica a todos los elementos "label" que tienen el atributo "for" (sin importar el valor que tengan asignado), es posible también seleccionar elementos que tengan un atributo con un valor diferente al designado, por ejemplo "input[size~=20]" aplica el estilo a todos los elementos "input" cuyo atributo "size" sea diferente de 20.

En este ejemplo no se ha validado la introducción de datos porque los campos son generados (no introducidos manualmente). En una aplicación real se tienen que validar los datos introducidos.

## 6.10. EJERCICIOS

Para presentar los ejemplos y ejercicios del tema, debe crear una carpeta con el nombre "tema5" y guardar en la misma todos los archivos HTML creados (ejemplos y ejercicios).

1. Elabore una aplicación que genere 500 registros aleatorios, correspondientes a estudiantes de la Facultad de Tecnología, con los siguientes campos: apellido paterno, apellido materno, nombres (1 o 2), dirección y carnet universitario con los "id" de hasta 3 carreras (32, 35, 37). La aplicación debe mostrar los registros en un formulario y permitir recorrerlos hacia adelante o hacia atrás uno por uno, así como ir al primer registro o al último y a un número de registro específico.

2. Elabore una aplicación que genere 400 registros aleatorios, correspondientes a las calificaciones de hasta 5 asignaturas, con los siguientes campos: nombre completo, sigla de la asignatura, nota (entre 1 y 100) y estado (aprobado o reprobado). La aplicación debe mostrar los registros en un formulario y permitir recorrerlos hacia adelante o hacia atrás uno por uno, así como ir al primer registro o al último y a un número de registro específico.

3. Elabore una aplicación que genere 350 registros aleatorios, correspondientes a la nómina de trabajadores de una empresa, con los siguientes campos: nombre completo (con uno o dos apellidos y uno o dos nombres), sexo (masculino/femenino), dirección y fecha de nacimiento (entre 1960 y 1990). La aplicación debe mostrar los registros generados en una tabla donde cada campo corresponda a una columna y cada fila a un registro. En la tabla la primera fila debe mostrar los nombres de los campos en azul, negrita, centrado, sobre fondo verde claro, y los registros deben ser mostrados en verde, intercalando el color del fondo de las filas entre amarillo claro y cian claro.

4. Elabore una aplicación que genere 600 registros aleatorios, correspondientes a los artículos de una ferretería, con los siguientes campos: nombre del artículo, costo unitario, precio unitario y cantidad. La aplicación debe mostrar los registros generados en un formulario y permitir recorrerlos hacia adelante o hacia atrás uno por uno, así como ir al primer registro o al último y a un número de registro específico. Para generar los nombres de los artículos debe crear un dominio (y su correspondiente función generadora) con por lo menos 50 artículos.

5. Elabore una aplicación que genere 550 registros aleatorios, correspondientes a los artículos de una farmacia, con los siguientes campos: nombre del producto, tipo de venta (con receta o sin receta), costo unitario, precio unitario y cantidad. La aplicación debe mostrar los registros generados en una tabla donde cada campo corresponda a una columna y cada fila a un registro. En la tabla la primera fila debe mostrar los nombres de los campos en rojo, negrita, centrado sobre fondo azul claro, y los registros deben ser mostrados en azul oscuro, intercalando el color del fondo de las filas entre verde claro y azul claro. Para generar los nombres de los productos debe crear un dominio (y su correspondiente función generadora) con por lo menos 50 productos.

# 7. ORDENACIÓN

Como una aplicación de las estructuras estudiadas en los anteriores temas, en este se estudian algunos métodos de ordenación. Es conveniente comenzar con la definición de este término: ordenar es arreglar los elementos de una lista de acuerdo a un determinado criterio.

Por ejemplo para ordenar ascendentemente números reales, el criterio es que todos los elementos anteriores a un número dado sean menores o iguales al dicho número y que los posteriores sean mayores o iguales al mismo. Si el objetivo es ordenar descendentemente los elementos de un vector con nombres, el criterio sería que todos los elementos anteriores a un determinado nombre sean alfabéticamente, mayores o iguales a dicho elemento y que todos los posteriores sean menores o iguales al mismo.

En la mayoría de los casos prácticos los elementos de un vector son ordenados para que sus elementos estén en orden ascendente o descendente. Es por ello que los métodos que se estudiarán en este capítulo permiten ordenar los vectores en una de estas dos formas.

En general los métodos de ordenación se clasifican en dos grupos: a) Los *métodos directos*, que comparan elementos consecutivos y b) los *métodos indirectos*, que comparan elementos no consecutivos.

En este tema se estudian tres métodos directos: *Burbuja*, *Selección* e *Inserción* y dos indirectos *Shell* y *Quick-Sort*.

Con listas pequeñas (de hasta unos 1000 elementos) no existe mayor diferencia entre un y otro método, pero a medida que incrementa el número de elementos la diferencia entre el tiempo consumido por los métodos directos e indirectos incrementa y con listas de cientos de miles o millones de elementos, las diferencias son tan grandes que los métodos directos pueden requerir días y hasta semanas para ordenar una lista mientras que los métodos indirectos llevan a cabo la misma labor en minutos o segundos.

En todos los métodos se explica el procedimiento que se sigue para ordenar ascendentemente números enteros, sin embargo los mismos principios se aplican para ordenar cualquier tipo de elemento y para cambiar el orden simplemente cambia el criterio de ordenación.

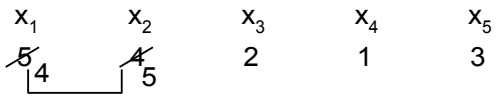
## 7.1. MÉTODO DE BURBUJA

El fundamento de este método consiste en ***llevar el mayor valor de la lista a la última posición***. Para ello se comparan pares consecutivos de elementos (comenzando con los dos primeros) y se realiza un intercambio si el primer elemento es mayor que el segundo, luego se comparan los dos siguientes (el segundo con el tercero) y así hasta comparar los dos últimos.

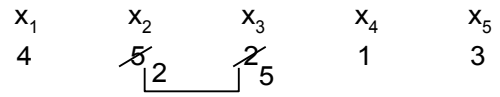
Una vez que se aplica el anterior procedimiento el mayor valor queda en la última posición, por lo tanto el último elemento queda ordenado. Entonces se repite el proceso con los "n-1" elementos restantes (con lo que se queda ordenado el penúltimo elemento) y se continúa así hasta que sólo queda un elemento en la lista. Por ejemplo para ordenar el siguiente vector:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| 5 | 4 | 2 | 1 | 3 |

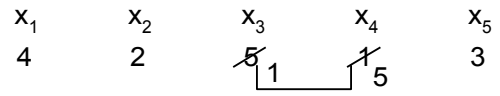
Se comienza comparando los dos primeros valores (5 con 4) y como el primero (5) es mayor que el segundo (4), se realiza un intercambio de variables:



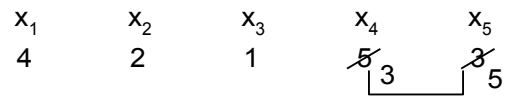

Ahora se compara el segundo con el tercer elemento (5 con 2) y como el primero es mayor al segundo se realiza otro intercambio de variables:



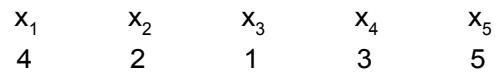

Se continúa entonces con el tercer y cuarto elemento (5 y 1) y como una vez más el primero es mayor que el segundo se realiza un intercambio:



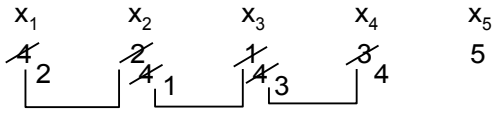

Finalmente se compara el cuarto y quinto elemento (5 con 3) y se realiza otro intercambio:



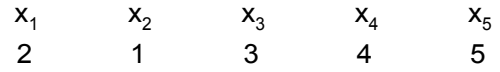

Con ello termina el proceso y, como se puede observar, el mayor valor queda en la última posición:



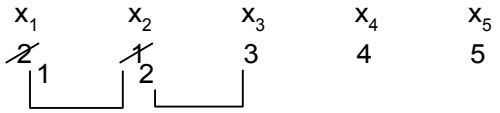

Ahora, para ordenar los elementos restantes, simplemente se repite el anterior proceso sin tomar en cuenta el último elemento, pues ya está ordenado:



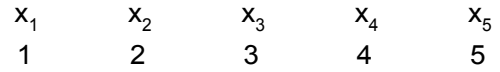

Con lo que el penúltimo elemento queda ordenado:



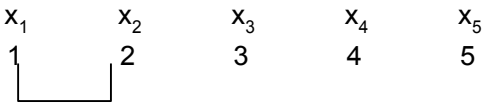

Se repite una vez más el proceso (sin tomar en cuenta los dos últimos elementos que ya están ordenados):



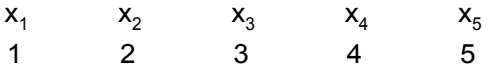

Con lo que el tercer elemento queda ordenado:




Finalmente se comparan los dos elementos restantes (el primer con el segundo):



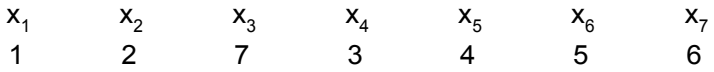

Y como no hay intercambios se sabe que el segundo elemento está ordenado:



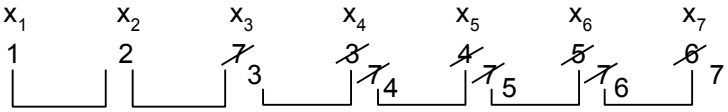

Ahora, como sólo queda un elemento, la lista ya está ordenada.

Al aplicar el método de burbuja puede suceder que la lista quede ordenada antes de repetir el proceso *n-1* veces (inclusive la lista puede estar ordenada desde un principio). En el método de burbuja se sabe que la lista está ordenada cuando al aplicar el proceso no se produce ningún intercambio.

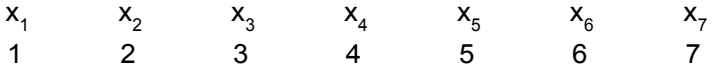
Por ejemplo si la lista a ordenar es la siguiente:



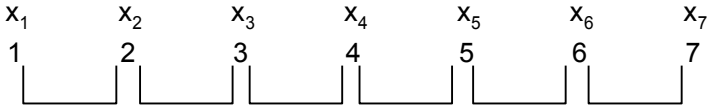

Aplicando el procedimiento de burbuja una vez se tiene:




Con lo que el vector queda ordenado:




Si se vuelve a aplicar el método a esta lista:



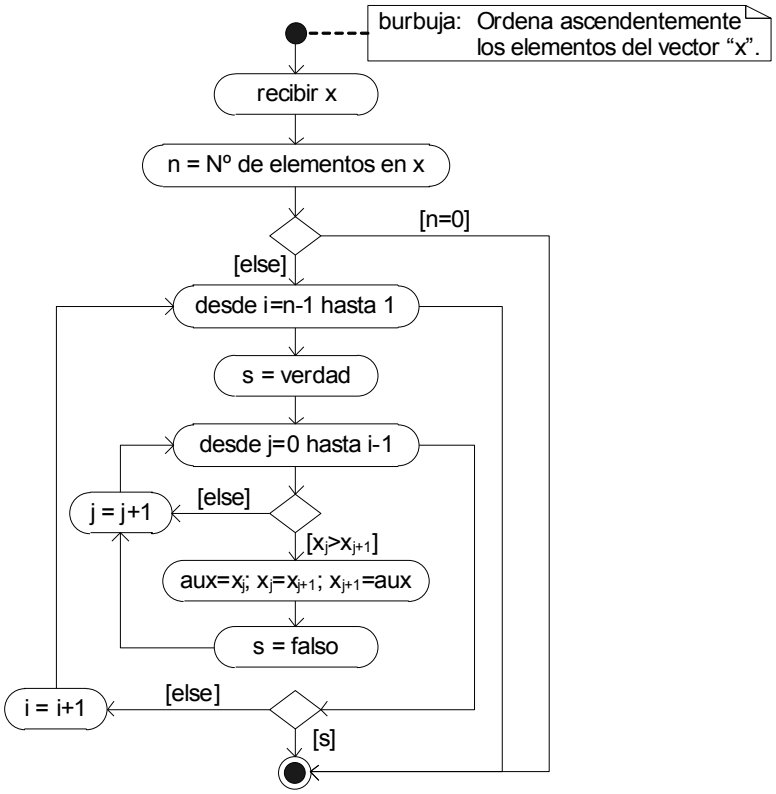

No se produce ningún intercambio, lo que indica que la lista ya está ordenada. En consecuencia si al repetir el método de burbuja no se produce ningún intercambio se sabe que la lista ya está ordenada.

### *7.1.1. Algoritmo*

El algoritmo para ordenar de forma ascendente los elementos de una lista, se deduce fácilmente de la anterior explicación y se presenta en forma de diagrama de actividades en la siguiente página.

Para probar este algoritmo, se ha creado una página que ordena una lista de 1000 números enteros, generados al azar y comprendidos entre 1 y 2000.

```
<!DOCTYPE html>
<html>

<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <title>Burbuja</title>
```

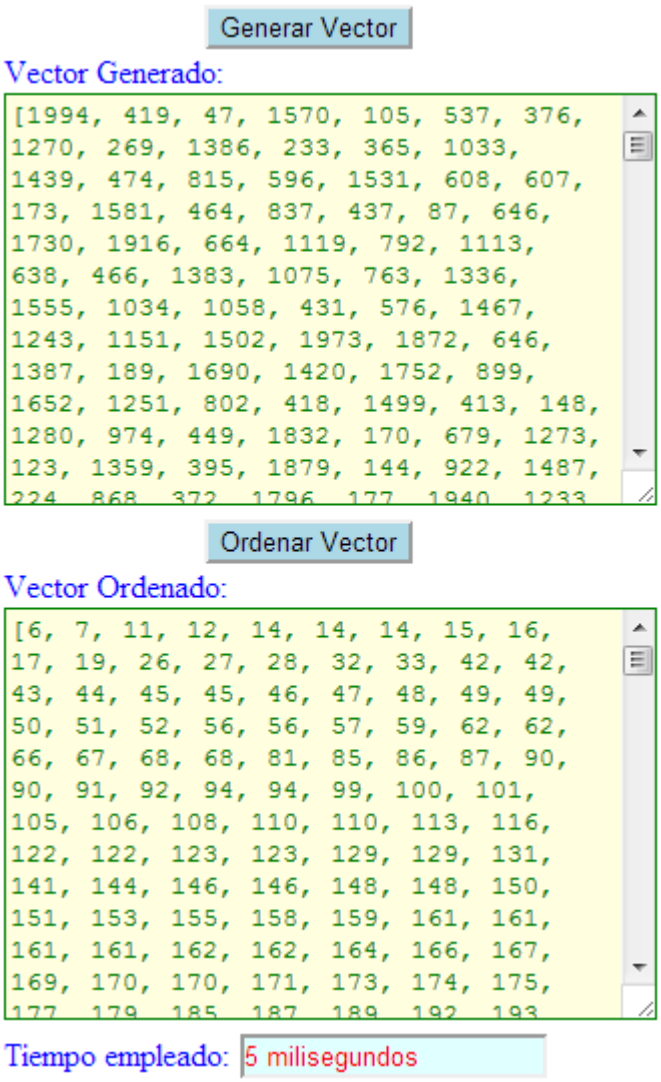



```
<script src="sis101.js"></script>

<script>

  function burbuja(x){
    var i,j,aux,n=x.length;
    if (n==0) return;
    for (i=n-1;i>0;i--){
      s=true;
      for (j=0;j<i;j++)
        if (x[j]>x[j+1]){
          aux=x[j]; x[j]=x[j+1]; x[j+1]=aux;
          s=false;
        }
      if (s) break;
    }
  }

  function generar(){
    x = round(rand(1000,1,2000));
    document.getElementById("vect1").innerHTML = x;
  }

  function ordenar(){
    var t1=new Date();
    burbuja(x);
    var t2=new Date();
    var dt = t2.getTime()-t1.getTime();
    document.getElementById("vect2").innerHTML=x;
    document.getElementById("tiempo").value=dt+" milisegundos";
  }
```

```
    </script>

    <style>
      input[type="button"] {
        background-color:lightblue;
        margin-left:100px;
        cursor:pointer;
      }
      input[type="text"] {
        color:red;
        background-color:lightcyan;
      }
      label {
        color:blue;
      }
      textarea {
        color: green;
        width:320px;
        height:200px;
        background-color:lightyellow;
      }
    </style>

  </head>

  <body onload="generar()">
    <input type="button" value="Generar Vector" onclick="generar()"><br>
    <label for="vect1">Vector Generado:</label><br>
    <textarea id="vect1"></textarea><br>
    <input type="button" value="Ordenar Vector" onclick="ordenar()"><br>
    <label for="vect2">Vector Ordenado:</label><br>
    <textarea id="vect2"></textarea><br>
    <label for="tiempo">Tiempo empleado:</label>
    <input type="text" id="tiempo">
  </body>

  </html>
```

Al abrir esta página en un navegador, se puede ver una interfaz con una apariencia similar a la que se muestra en la siguiente página. Observe que el vector se genera y guarda en la variable global "x" (es global porque no ha sido declarada con la palabra "var" por delante), al ser global, puede ser empleada desde cualquiera de las funciones, siendo esa la razón por la cual puede ser empleada directamente en la función "ordenar".

Para determinar el tiempo que demora el método en ordenar los elementos, se han empleado objetos "Date". Estos objetos permiten trabajar con valores de tiempo y/o fecha en Javascript. Si se crea un objeto "Date" sin ningún argumento (como se ha hecho en el ejemplo), contiene la fecha y hora actuales, por esa razón en el ejemplo se crea el objeto "t1" antes de llamar al método de ordenación y se crea el objeto "t2" cuando el método devuelve la lista ordenada, entonces, para obtener el tiempo transcurrido entre la llamada y la devolución del resultado, se emplea el método "getTime()" que devuelve los milisegundos transcurridos entre el primero de enero de 1970 y la fecha del objeto. Por lo tanto, al restar los milisegundos de "t1" a los milisegundos de "t2", se obtiene el tiempo empleado por el método en ordenar la lista.




Como se puede ver, en listas pequeñas como esta (con solo 1000 elementos) el tiempo requerido es también pequeño, sin embargo a medida que incrementa el número de elementos el tiempo incrementa pero no en forma lineal, sino en forma geométrica, así para ordenar 10000 elementos se requiere alrededor de 370 milisegundos, es decir ¡74 veces lo que requiere ordenar 1000 elementos!. Si la relación fuera lineal, ordenar 10000 elementos debería tomar unos 50 milisegundos, no 370 (por supuesto el tiempo real empleado varía de computadora a computadora).

Por otra parte, cuando se generan números aleatorios, por lo general se quiere que los mismos sean realmente aleatorios, es decir impredecibles (como ha ocurrido hasta ahora), sin embargo, para comparar la eficiencia de dos o más métodos, es necesario contar con la misma serie de valores, pues de lo contrario la comparación no sería correcta y se podría llegar a conclusiones erróneas.

Con ese fin se deben generar números pseudoaleatorios, es decir secuencias de números que pueden ser reproducidas. Javascript no cuenta con un generador pseudoaleatorio, por lo que se ha añadido uno a la librería "sis101.js". Para activar este generador, se debe iniciar la variable global "randseed" asignándole un valor diferente de nulo (null), dicho número, conocido como semilla (seed), se emplea como base para generar la secuen-

cia de números pseudoaleatorios, por lo que es posible reproducir dicha secuencia volviendo a asignar el mismo número (la misma semilla) a la variable "randseed".

Por ejemplo, en las siguientes instrucciones se inicia la semilla en 5 y se generan 10 número aleatorios comprendidos entre 1 y 10 y redondeados al cuarto dígito después del punto, se obtiene:

```
>>randseed=5
5
>>round(rand(10,1,10),4)
[4.6961, 9.0007, 5.7198, 8.3253, 5.9564, 4.4185, 1.1838, 2.3912, 9.8464,
5.0214]
```

Para volver a reproducir esta serie de números, se vuelve a iniciar la semilla en 5:

```
>>randseed=5
5
>>round(rand(10,1,10),4)
[4.6961, 9.0007, 5.7198, 8.3253, 5.9564, 4.4185, 1.1838, 2.3912, 9.8464,
5.0214]
```

Con lo que se obtiene, como se puede ver, la misma serie.

Para desactivar el generador de números pseudoaleatorios, se vuelve a asignar el valor nulo a la semilla:

```
>>randseed=null
null
>>round(rand(10,1,10),4)
[4.8337, 7.1327, 6.1085, 3.1623, 9.6492, 5.5345, 6.0016, 4.2365, 6.3835,
3.5756]
```

Con lo que, como se puede ver, la serie no se repite. En este tema, para comparar los diferentes métodos se iniciará la semilla en 10.

## 7.2. MÉTODO DE SELECCIÓN

El fundamento de este método consiste en ***seleccionar el mayor valor de la lista e intercambiarla con el último elemento***, de esta manera (al igual que sucede con el método de *Burbuja*), el mayor valor queda en la última posición.

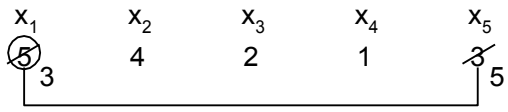
Luego, al igual que en el método de burbuja, el proceso se repite sin tomar en cuenta el último elemento (que ya está ordenado) y se continúa así (sin tomar en cuenta los elementos ya ordenados) hasta que en la lista queda un elemento.

Por lo general, este método es más eficiente que el de burbuja porque implica menos intercambios, sin embargo, en este método no hay modo de saber si en algún momento la lista ya estaba ordenada, por lo que el mismo se repite "n-1" veces inclusive, si la lista queda ordenada en el proceso o ya estaba ordenada desde un principio.

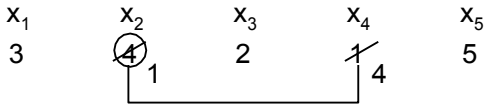
Para comprender mejor el método se ordenará la siguiente lista aplicando el método.

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| 5 | 4 | 2 | 1 | 3 |

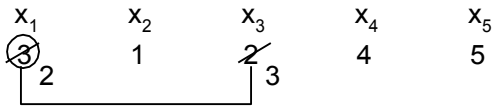
Primero se selecciona el mayor de valor de la lista (el número 5) y se intercambia con el último elemento:



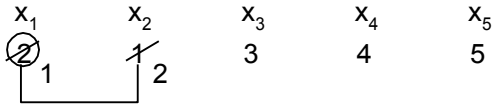

Entonces se repite el procedimiento sin tomar en cuenta el último elemento (pues ya está ordenado):



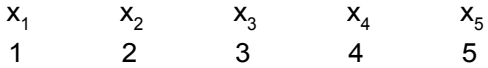

Ahora los dos últimos elementos ya están ordenados, por lo que se repite el proceso con los 3 elementos restantes:




Finalmente se repite el proceso con los dos elementos restantes:



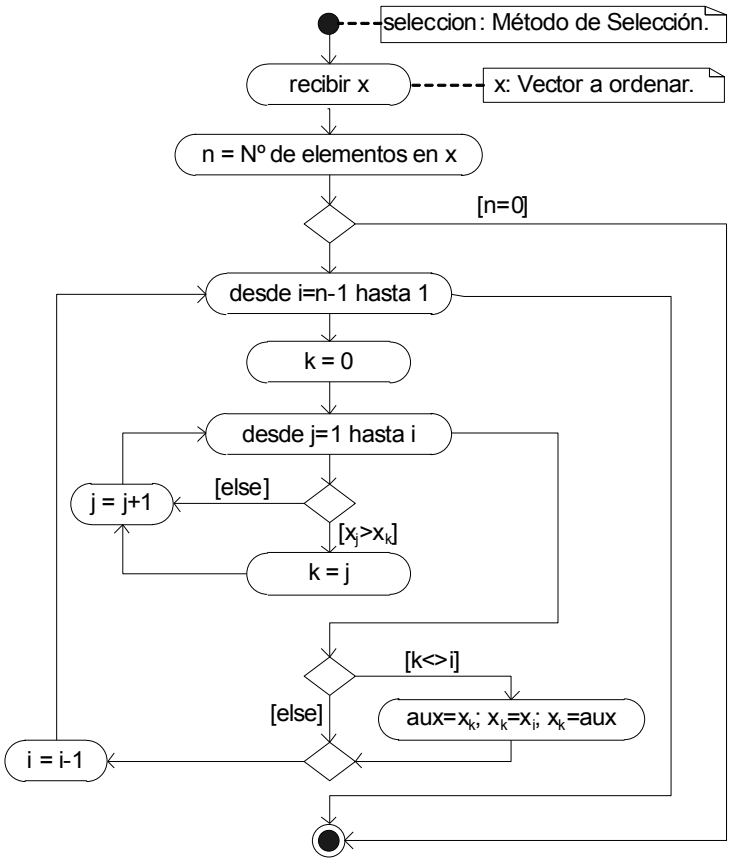

Ahora al quedar un sólo elemento, la lista ya está ordendada:




### *7.2.1. Algoritmo*

El algoritmo para ordenar ascendentemente los elementos de una lista se muestran en la siguiente página. La página escrita para probar este algoritmo, con una lista de números enteros, es esencialmente la misma que para el método de Burbuja, cambiando obviamente el método, la llamada al método y la generación de números, que como se dijo, será de 10000 números pseudo-aleatorios (con una semilla de 10). Dichos cambios son los siguientes:

```
    function seleccion(x){
     var i,j,aux,n=x.length;
     if (n==0) return;
     for (i=n-1;i>0;i--){
       k=0;
       for (j=1;j<=i;j++)
         if (x[j]>x[k])k=j;
       if (k!=i) {
         aux=x[k]; x[k]=x[i]; x[i]=aux;
       }
     }
    }

    function generar(){
      randseed=10;
      x = round(rand(10000,1,10000));
      document.getElementById("vect1").innerHTML = x;
```

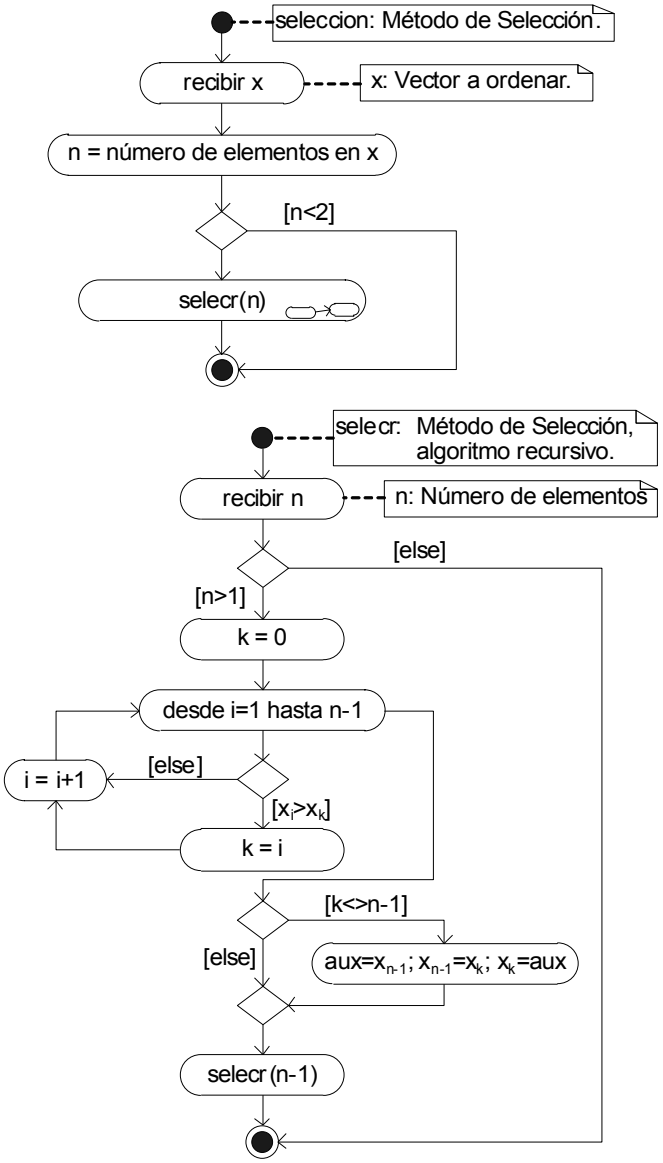



```
    document.getElementById("vect2").innerHTML = "";
    document.getElementById("tiempo").value = "";
}

function ordenar(){
  var t1=new Date();
  seleccion(x);
  var t2=new Date();
  var dt = t2.getTime()-t1.getTime();
  document.getElementById("vect2").innerHTML = x;
  document.getElementById("tiempo").value=dt+" milisegundos";
}
```

Abriendo la página y ordenando la lista, el tiempo requerido es de aproximadamente 210 milisengudos, es decir casi 2 veces menos (1.7 veces menos) que el método de Burbuja (aunque, como se ha explicado, esta diferencia no siempre es a favor del método de selcción).

Este método, el de Burbuja y todos los métodos de ordenación que se estudiarán en este tema pueden ser implementados de forma recursiva. El razonamiento recursivo para ordenar una lista por el método de selección es el siguiente: Una lista puede ser ordenada buscando su mayor valor e intercambiándolo con el último elemento, mandando luego la lista restante (sin su último elemento) a ser ordenada por el método de Selección. En el caso del método de Burbuja el razonamiento recursivo es el siguiente: Una lista puede ser ordenada llevando su mayor valor a la última posición comparando elementos sucesivos e intercambiándolos si el primero es mayor que el segundo, mandando luego la lista restante (sin su último elemento) a ser ordenando por el método de Burbuja. Note que en ambos métodos el razonamiento implica una llamada al mismo método.

El algoritmo recursivo para ordenar ascendentemente listas por el método de Selección es el siguiente:




Observe que en el algoritmo recursivo "selecr" es una sub-función del módulo principal "seleccion", como se recordará, el símbolo ⬭→⬭, informa que se trata de una función anidada. La capacidad de crear una función dentro de otra (funciones anidadas) es una característica propia de los lenguajes que soportan la metodología de Programación Estructurada y es una de las herramientas con las que se logra el encapsulamiento en esta metodología, es decir evitar que variables externas interfieran con variables locales y evitar interferir con variables externas.

Una función anidada puede acceder a todas las variables (y parámetros) de la función que la contiene, pero la función contenedora no puede acceder a las variables de la función anidada, en contrapartida, las variables de la función anidada no interfieren con las variables de la función contenedora.

Es gracias a estas características que la función recursiva "selecr" puede trabajar con el vector "x" sin que sea un parámetro de la misma (sin necesidad de que se le mande el vector "x"), "selecr" puede acceder al vector

"x" porque al ser una función anidada tiene acceso a todas las variables de la función contenedora. De esa manera, la función recursiva sólo recibe el único dato que realmente varía en el proceso recursivo: el número de elementos "n".

Para probar el algoritmo recursivo y como una forma de demostrar que este algoritmo (como cualquier otro) no está limitado a números, sino que puede ser aplicado a cualquier tipo de dato, se creará una página HTML en la que se genera una lista con 1000 registros conteniendo los campos nombre, dirección y teléfono, ordenando luego dichos registros por el campo nombre:

```
<!DOCTYPE html>
<html>

<head>

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <script src="sis101.js"></script>

  <title>Seleccion - Ejemplo 2</title>

  <script>

   function seleccion(x){
     var i,k,aux;
     function selecr(n){
       if (n>1){
         var k=0;
         for (i=1;i<n;i++)
           if (x[i].nombre>x[k].nombre) k=i;
        if (k!=n-1){
          aux=x[n-1]; x[n-1]=x[k]; x[k]=aux;
        }
        selecr(n-1);
       }
     }
     var n=x.length;
     if (n>1) selecr(n);
   }

    function Registro(nombre,direccion,telefono){
      this.nombre = nombre;
      this.direccion = direccion;
      this.telefono = telefono;
    }

    function generarNombre(){
      var s=generarApellido();
      if (random(1)) s+=" "+generarApellido();
      if (random(1)) s+=" "+generarNombreF()
      else s+=" "+generarNombreM();
      return s;
    }

    function generarDireccion(){
      var s=generarCalle();
      s+=" N° "+round(random(1,1000));
      return s;
    }
```

```
  function generarTelefono(){
    return 6400000+round(random(1,99999));
  }

  function generarRegistros(){
    randseed = 10;
    r = new Array(1000)
    for (var i=0;i<1000;i++){
      r[i] = new Registro(generarNombre(),generarDireccion(),
          generarTelefono());
    }
  }

  function llenarTabla(){
    generarRegistros();
    var texto="<tr><th>Nombre</th><th>Dirección</th>"+
      "<th>Teléfono</th></tr>";
    for (var i=0;i<r.length;i++){
      texto+= odd(i) ? "<tr fila='impar'>" : "<tr fila='par'>";
      texto+="<td>"+r[i].nombre+"</td>"+
        "<td>"+r[i].direccion+"</td>"+
        "<td>"+r[i].telefono+"</td></tr>";
    }
    document.getElementById("tabla1").innerHTML=texto;
  }

  function ordenarTabla(){
    seleccion(r);
    var texto="<tr><th>Nombre</th><th>Dirección</th>"+
    "<th>Teléfono</th></tr>";
                        for (var i=0;i<r.length;i++){
          texto+= odd(i) ? "<tr fila='impar'>" : "<tr fila='par'>";
                      texto+="<td>"+r[i].nombre+"</td>"+
                         "<td>"+r[i].direccion+"</td>"+
                       "<td>"+r[i].telefono+"</td></tr>";
                                      }
          document.getElementById("tabla1").innerHTML=texto;
  }

</script>

<style type="text/css">
  button {
    color:mediumblue;
    font-weight:bold;
    background-color:PowderBlue;
  }
  table {
    color:ForestGreen;
    border-style:double;
    border-width:3px;
    border-collapse:collapse;
  }
  td {
    border-style:solid;
    border-width:1px;
```

```
      }
      th {
        color:chocolate;
        font-weight:bold;
        background-color:#99CC00;
        border-style:solid;
        border-width:1px;
        border-color:black;
      }
      tr[fila="impar"] {
        background-color:#FFFFCC;
      }
      tr[fila="par"] {
        background-color:#CCFF99;
      }
    </style>

  </head>

  <body onload="llenarTabla()">
    <button onclick="llenarTabla()">Generar</button>
    <button onclick="ordenarTabla()">Ordenar</button><br><br>
    <table id="tabla1" >
    </table>

  </body>

  </html>
```

Como se puede ver, el algoritmo en sí no cambia, lo único que es necesario tomar en cuenta cuando se ordenan registros, en lugar de datos simples, es que al comparar los valores sólo se compara él o los campos, por los que se ordenan (en este caso el campo "nombre").

Al abrir la página se llena la tabla con los 1000 registros generados (el cual puede volver a ser generado haciendo clic en el botón "generar"), para ordenar estos registros se hace clic en el botón "ordenar" con lo que los mismos quedan ordenados por el campo "nombre":


| Nombre | Dirección | Teléfono |
|---|---|---|
| Abascal Lasa Naudina | Destacamento 317 Nº 946 | 6417117 |
| Abella Agustí Irmaris | Rosendo Villa Nº 995 | 6403153 |
| Abella Ariño Maricela | Los Sauces Nº 514 | 6472915 |
| Abellán Chachajaque Marion | Regimiento Campos 6 de infantería Nº 170 | 6454277 |

En realidad, el resultado no es del todo correcto debido a que, por defecto, la comparación de cadenas en Javascript no funciona correctamente con letras acentuadas y caracteres especiales como la letra "ñ".

## 7.3. MÉTODO DE INSERCIÓN

El fundamento de este método consiste en ***insertar un elemento en su posición correcta con relación a los elementos que se encuentran antes del mis-***

***mo.*** Cuando la lista está totalmente desordenada, el método comienza con el segundo elemento y continúa insertando elementos, en su posición correcta, hasta llegar al último elemento.

En la práctica este método se emplea principalmente para insertar nuevos elementos en listas ya ordenadas, no obstante, en este tema se implementa el método de manera que permita ordenar listas completas.

Para comprender mejor el método se ordena el siguiente vector:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| 5 | 14 | -2 | 10 | 3 |

Se comienza con el segundo elemento (que tiene el valor 14) y como es mayor al primero se encuentra ya en su posición correcta con relación al elemento anterior, por lo que es insertado directamente en la segunda posición:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| 5 | (14) | -2 | 10 | 3 |

Ahora se pasa al tercer elemento (que tiene el valor -2) y como es menor a los dos anteriores, es insertado en la primera posición, desplazando los elementos anteriores una posición hacia la derecha:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| 5 ⟵ | 14 ⟵ | (-2) | 10 | 3 |

Con ello los tres primeros elementos quedan ordenados:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| -2 | 5 | 14 | 10 | 3 |

Ahora se pasa al cuarto elemento (que tiene el valor 10) y como este elemento es menor al tercero, pero mayor al segundo, debe ser insertado en la tercera posición:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| -2 | 5 | 14 ⟵ | (10) | 3 |

Quedando los cuatro primeros elementos ordenados:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| -2 | 5 | 10 | 14 | 3 |

Finalmente se pasa al quinto elemento (que tiene el valor 3) y como este elemento es menor al segundo, pero mayor al primero, debe ser insertado en la segunda posición:

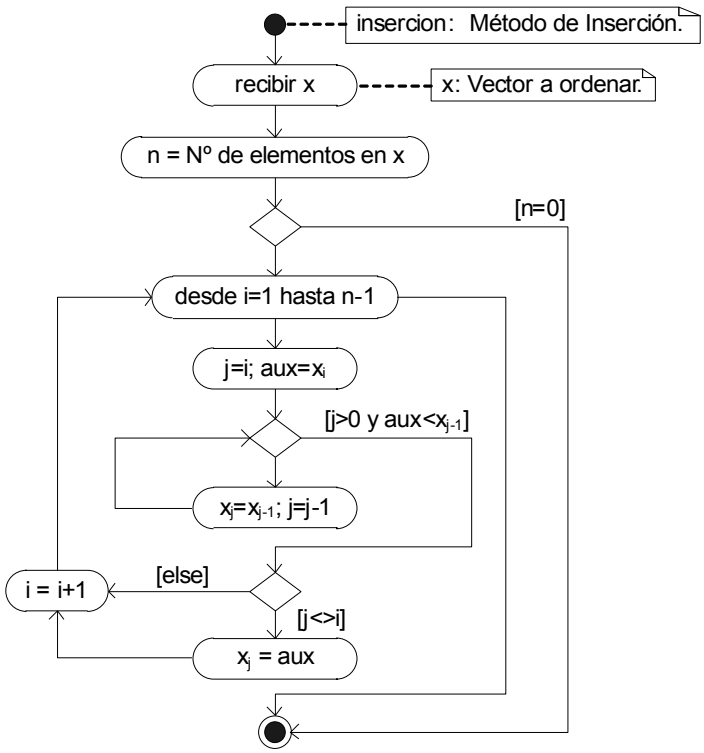
| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| -2 | 5 ⟵ | 10 ⟵ | 14 ⟵ | (3) |

Quedando con ello los 5 elementos ordenados:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|
| -2 | 3 | 5 | 10 | 14 |

### *7.3.1. Algoritmo*

El algoritmo del método, en forma de diagrama de actividades, es el siguiente:




La página creada para probar este método, con excepción del método en sí, es esencialmente la misma que la del anterior, las partes que varían con relación a dicho código son:

```
function insercion(x){
  var i,j,aux,n=x.length;
  if (n==0) return;
  for (i=1;i<n;i++){
    j=i; aux=x[i];
    while (j>0 && aux<x[j-1]){
      x[j]=x[j-1]; j--;
    }
    if (j!=i) x[j]=aux;
  }
}

function ordenar(){
  var t1=new Date();
  insercion(x);
  var t2=new Date();
  var dt = t2.getTime()-t1.getTime();
  document.getElementById("vect2").innerHTML=x;
  document.getElementById("tiempo").value=dt+" milisegundos";
}
```

El método de inserción requiere aproximadamente 100 milisegundos para ordenar los 10000 números, es decir 3.7 veces menos que el método de burbuja y 2.1 veces menos que el método de selección.

Al igual que los métodos anteriores, este método puede ser implementado de manera recursiva y el algoritmo puede ser empleado para ordenar registros en lugar de datos numéricos.

## 7.4. MÉTODO SHELL

En el método *Shell*, denominado así en honor a su creador *D. L. Shell*, **se comparan valores que están separados entre sí por un determinado número de elementos** y al igual que en el método de burbuja si el primer valor es mayor que el segundo se intercambian.

Al número de elementos que separan los valores a comparar se conoce con el nombre de *salto* (o *paso*) e inicialmente es igual a la mitad (entera) del número de elementos existentes.

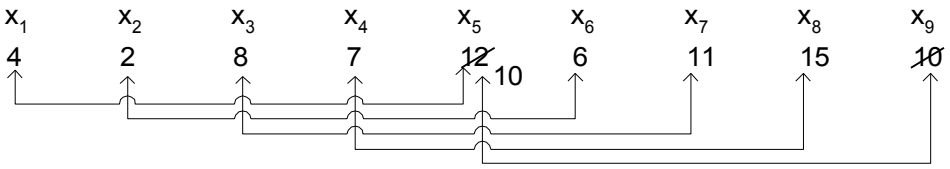
Si *k* es el paso o salto, en el proceso se comparan el primer elemento con el elemento que se encuentra *k+1* elementos más adelante, luego se compara el segundo elemento con el elemento que se encuentra *k+2* elementos más adelante y así sucesivamente hasta que se compara el último elemento de la lista, es decir se realiza un total de *k-p* comparaciones.

El anterior procedimiento se repite hasta que no ocurre ningún intercambio, entonces el *paso* o *salto* es reducido a la mitad y el procedimiento se vuelve a repetir (con el nuevo *paso*) hasta que, una vez más, no existen más intercambios. Cuando esto ocurre se vuelve a reducir el valor del *paso* a la mitad y se continúa de esta manera hasta que el *paso* se reduce a 1.

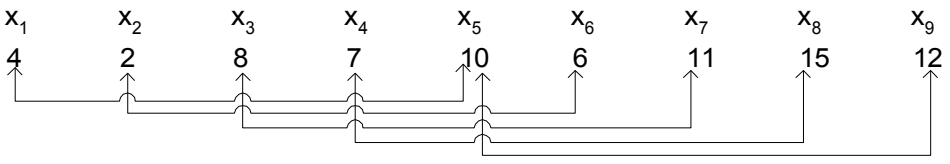
Para comprender mejor el procedimiento se ordenará el siguiente vector siguiendo el método Shell:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ |
|---|---|---|---|---|---|---|---|---|
| 4 | 2 | 8 | 7 | 12 | 6 | 11 | 15 | 10 |

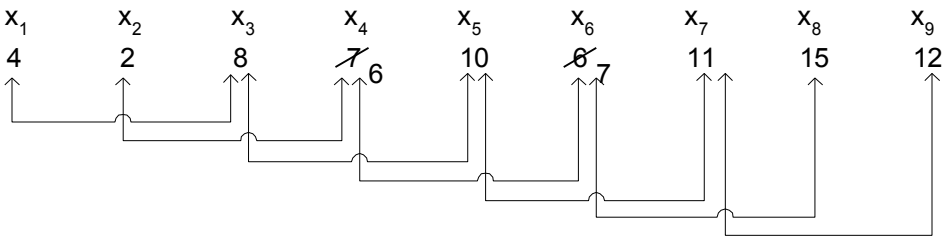
Como el número de elementos es 9, el paso inicial es 9/2=4 (división entera). Entonces se compara el primer elemento con el quinto, el segundo con el sexto y así sucesivamente:



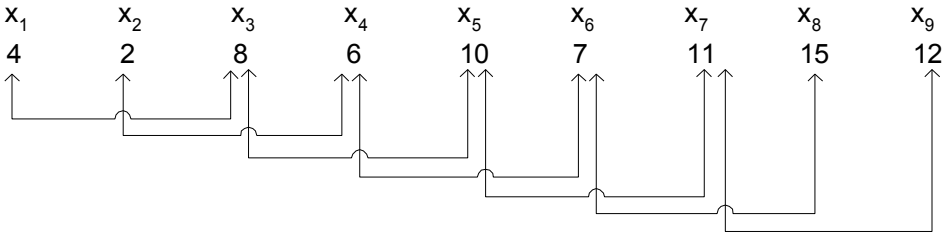

Y como se ha producido un intercambio se repite el procedimiento:



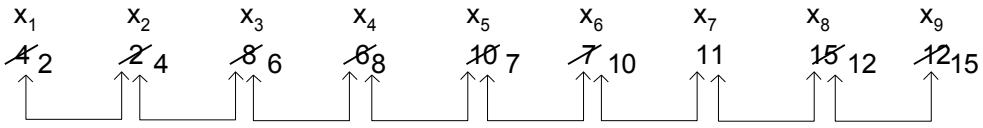

Ahora no se produce ningún intercambio, por lo tanto se reduce el paso a la mitad: 4/2=2, y se vuelve a repetir el proceso empleando este nuevo paso (es decir se compara $x_1$ con $x_3$, $x_2$ con $x_4$ y así sucesivamente):



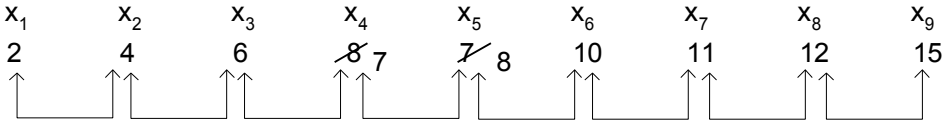

Como hay intercambios se repite el proceso:



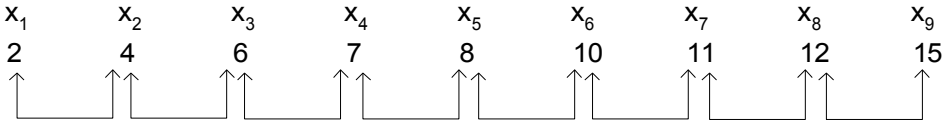

Ahora no se producen intercambios, por lo que el paso se reduce a la mitad: 2/2=1 y se repite el proceso con este nuevo paso (se comparan elementos consecutivos: $x_1$ con $x_2$, $x_2$ con $x_3$, etc.):




Como hay intercambios, se repite el proceso:




Una vez más hay intercambios, por lo que se vuelve a repetir el proceso:

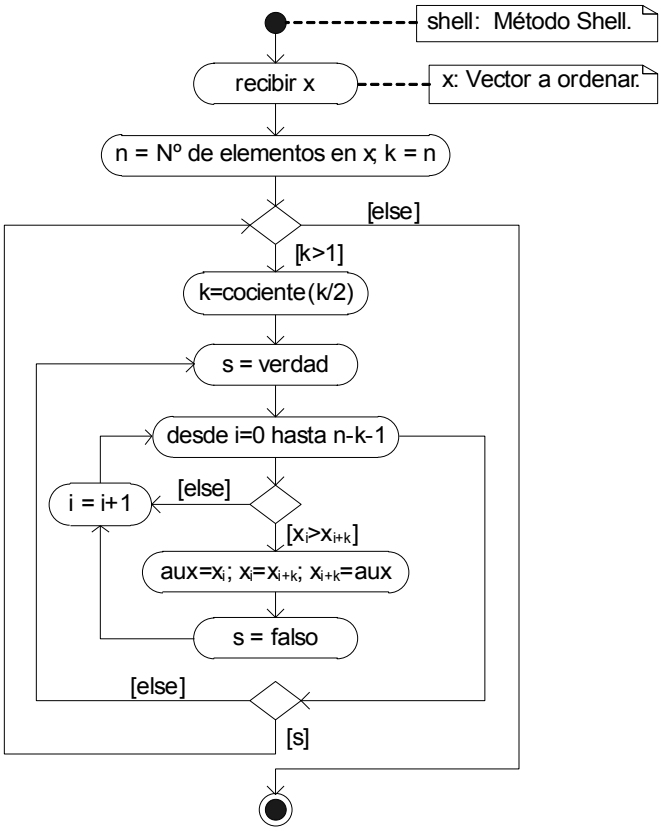



Ahora no hay intercambios y dado que el paso es 1, el proceso concluye y como se puede ver la lista ya está ordenada.

Como ya se dijo, estos métodos son más eficientes que los directos porque reducen el número de intercambios. Así en el ejemplo se requieren 7 intercambios, mientras que con *Burbuja* se habrían requerido 9 y esta diferencia incrementa geométricamente con el número de elementos.

### *7.4.1. Algoritmo*

El algoritmo para ordenar listas de forma ascendente se presenta en la siguiente página. El código escrito para probar el mismo, con excepción del método en sí, es esencialmente el mismo que para los anteriores métodos, siendo las partes que cambian las siguientes:




```
function shell(x){
  var i,j,aux,n=x.length,k=n;
  while (k>1) {
    k = quot(k,2);
    while (true) {
      s = true;
      for (i=0;i<n-k-1;i++)
        if (x[i]>x[i+k]){
          aux=x[i]; x[i]=x[i+k]; x[i+k]=aux;
          s = false;
        }
      if (s) break;
    }
  }
}

function ordenar(){
  var t1=new Date();
  shell(x);
  var t2=new Date();
  var dt = t2.getTime()-t1.getTime();
  document.getElementById("vect2").innerHTML=x;
  document.getElementById("tiempo").value=dt+" milisegundos";
}
```

El método Shell requiere aproximadamente 9 milisegundos para ordenar los 10000 números, es decir que en esta lista es 41 veces más rápido que el método de Burbuja, 23 veces más rápido que el método de Selección y 11 veces más rápido que el método de inserción. Como todos los métodos puede ser implementado también en forma recursiva.

## 7.5. MÉTODO QUICK-SORT (HOARE)

Este método fue propuesto por *C. R. Hoare* y es el método de ordenación más eficiente de todos los que se han estudiado hasta ahora. ***El método consiste en dividir el vector en dos: uno con elementos menores o iguales a un valor de referencia denominado pivote y otro con elementos mayores o iguales a dicho valor***.

El anterior procedimiento se repite con cada uno de los dos vectores resultantes y luego con cada uno de los vectores resultantes de estos y así sucesivamente hasta que cada vector queda con un solo elemento, momento en el cual el vector original queda ordenado.

El valor de referencia, el valor *pivote*, puede ser cualquiera de los elementos del vector, aunque frecuentemente se elige el elemento central, el primer elemento o el último elemento. Al finalizar la división, el *pivote* queda en uno de los dos vectores resultantes o al medio de los dos, en cuyo caso se encuentra ya en la posición correcta.

Para comprender mejor el procedimiento se ordenará la siguiente lista empleando el método *Quick Sort*:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ |
|---|---|---|---|---|---|---|---|---|
| 15 | 21 | 30 | 7 | 20 | 18 | 10 | 14 | 28 |

Para este ejemplo, se elige como *pivote* el primer elemento, por lo tanto el *pivote* será 15 (tome en cuenta que el pivote es el valor del primer elemento, no el elemento en sí, porque en el proceso su valor puede ir cambiando).

Ahora se debe dividir la lista en 2, una con elementos mayores a 15 y otra con elementos menores a 15. Para ello se emplean dos contadores: uno que comienza en el índice más bajo del vector (1) y otro en el índice más alto (9):

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | |
|---|---|---|---|---|---|---|---|---|---|
| 15 | 21 | 30 | 7 | 20 | 18 | 10 | 14 | 28 | pivote =15 |
| i=1 | | | | | | | | j=9 | |

Ahora se incrementa el valor del contador *i* mientras $x_i$ sea menor al valor pivote. Como en este caso $x_i$ es igual al pivote, no incrementa su valor.

Se pasa entonces a disminuir el valor del contador *j*, en este caso mientras $x_j$ sea mayor al pivote, por lo tanto el contador *j* disminuye hasta $x_8$, donde el valor (14) es menor al pivote:

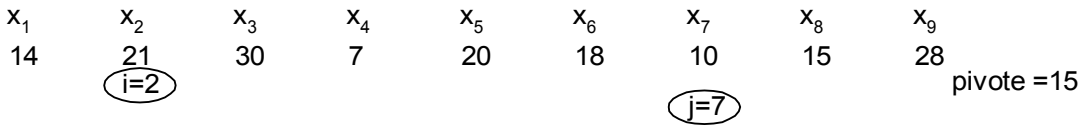
| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | |
|---|---|---|---|---|---|---|---|---|---|
| 15 | 21 | 30 | 7 | 20 | 18 | 10 | 14 | 28 | pivote =15 |
| i=1 | | | | | | | j=8 | ← j=9 | |

Entonces se intercambian $x_i$ con $x_j$, ($x_1$ con $x_8$), luego se incrementa el contador *i* en uno y se disminuye el contador *j* en uno:

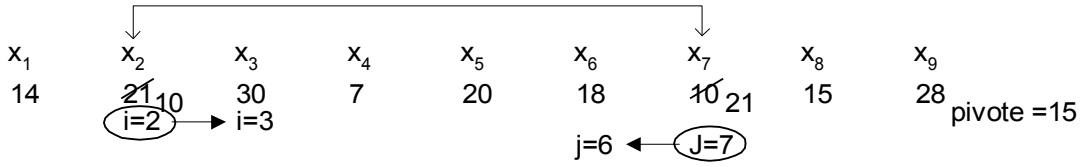
| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | |
|---|---|---|---|---|---|---|---|---|---|
| ~~15~~ 14 | 21 | 30 | 7 | 20 | 18 | 10 | ~~14~~ 15 | 28 | pivote =15 |
| i=1 → | i=2 | | | | | j=7 ← | j=8 | | |

Ahora se vuelve a repetir el proceso, es decir se incrementa el contador *i* mientras $x_i$ sea menor al pivote y se disminuye *j* mientras $x_j$ sea mayor al pivote.

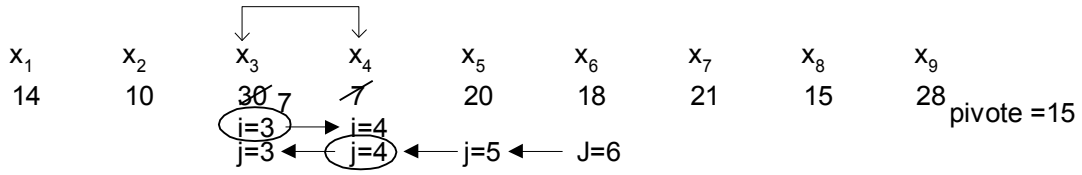
En este caso $x_i$ ya es mayor al pivote y $x_j$ menor, por lo tanto no incrementan ni se disminuyen sus valores:




Ahora que los contadores han quedado fijos, se intercambia $x_i$ con $x_j$, luego se incrementa *i* en uno y se disminuye *j* en uno:




Ahora se repetir el procedimiento: se incrementar *i* mientras $x_i$ sea menor al pivote, se disminuye *j* mientras $x_j$ sea mayor al pivote, se intercambian variables, se incrementa *i* y se disminuye *j:*



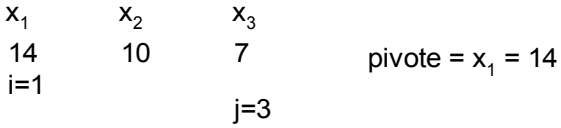

Como se puede ver en este caso el contador *i* queda con un valor mayor que el contador *j*.

Cuando esto sucede el proceso concluye y la lista queda dividida en dos: la lista izquierda que va desde el primer elemento hasta *j* (es decir desde 1 hasta 3) y la derecha que va desde *i* hasta el último elemento (es decir desde 4 hasta 9):

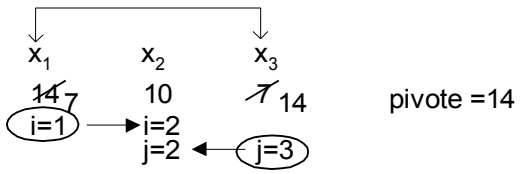
| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ |
|---|---|---|---|---|---|---|---|---|
| 14 | 10 | 7 | 30 | 20 | 18 | 21 | 15 | 28 |

De esta manera se consigue dividir el vector en dos: el izquierdo con los elementos menores al pivote y el derecho con los elementos mayores o iguales al pivote.

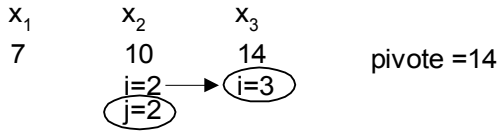
Ahora se vuelve a repetir el procedimiento con las listas resultantes. Los contadores de la lista izquierda son:



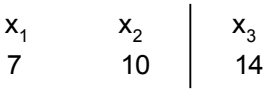

Como se ve ahora los contadores comienzan en 1 y 3 respectivamente y el pivote es el número 14. Aplicando el procedimiento una vez resulta:




Y repitiendo el procedimiento una vez más se tiene:

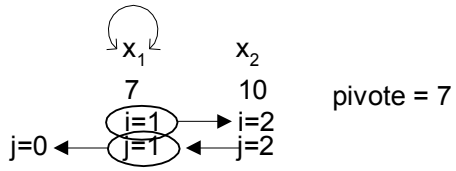



Ahora que los contadores han quedado fijos se debería intercambiar los valores de $x_i$ y $x_j$, pero como el contador *i* es mayor al contador *j*, no se realiza el intercambio y la lista queda dividida en dos: la izquierda (hasta j=2) con los elementos menores al pivote y la derecha (desde i=3) con un sólo elemento igual al pivote:




Cuando, como en este caso, una de las listas queda con un solo elemento, dicho elemento ya está ordenado, es decir tiene el valor correcto.

El vector izquierdo sin embargo tiene dos elementos por lo que se debe aplicar el procedimiento al mismo:

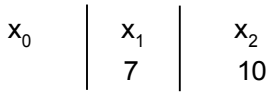



En este caso los contadores *i* y *j* llegan al mismo valor y se intercambia el elemento consigo mismo (algo que no se hace en la práctica).

Posteriormente (como siempre sucede después de un intercambio) el valor de i incrementa en uno (a 2) y el de j disminuye en uno (a 0).

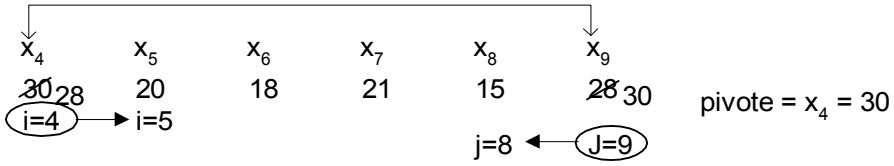
Entonces como *i* tiene un valor mayor a *j* el proceso concluye quedando la lista dividida en dos: el vector izquierdo (hasta j=0) que como vemos en realidad no tiene elementos y el derecho (desde i=2) que queda con un elemento.

Al medio queda una lista con un solo elemento y como es único, ese elemento ya está ordenado.

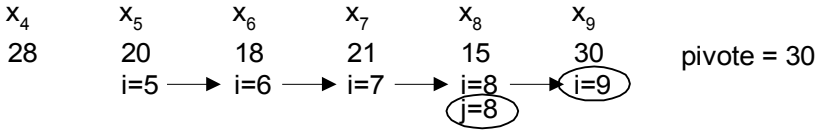



Como todos los vectores del lado izquierdo de la primera división han quedado con un solo elemento (o ninguno), estos elementos están ordenados.

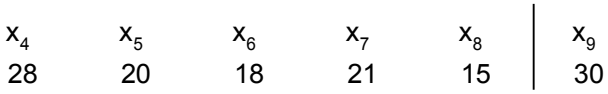
Ahora se aplica el procedimiento al vector derecho resultante de la primera división:




Puesto que *i* sigue siendo menor a *j* se vuelve a aplicar el procedimiento:

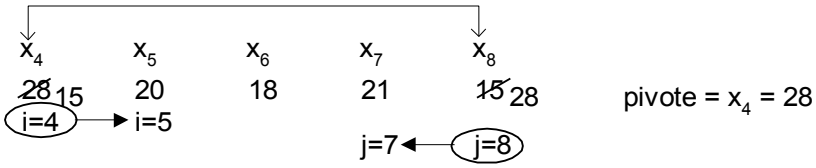



Donde no se realiza ningún intercambio pues *i* es ya mayor a *j*, por lo tanto el proceso concluye y la lista queda dividida en 2:

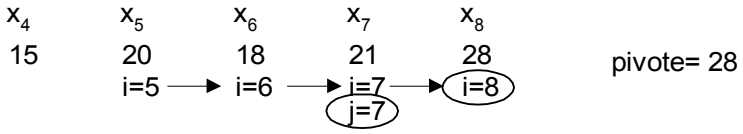



Como la lista derecha tiene un solo elemento, está ya con el valor correcto.

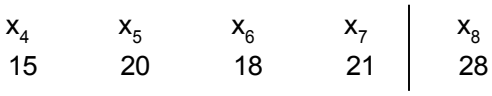
La lista izquierda, sin embargo, tiene más de un elemento por lo que se aplica el procedimiento a la misma:



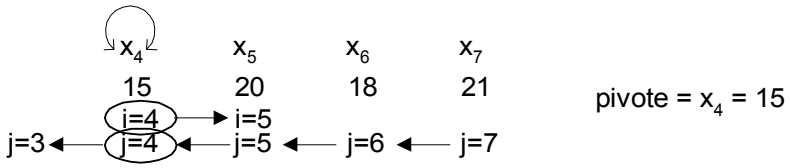

Puesto que *i* sigue siendo menor a *j* se vuelve a aplicar el procedimiento:




Una vez más no se realiza ningún intercambio pues el contador i es mayor al contador j. Por lo tanto el proceso concluye y la lista queda dividida en dos:



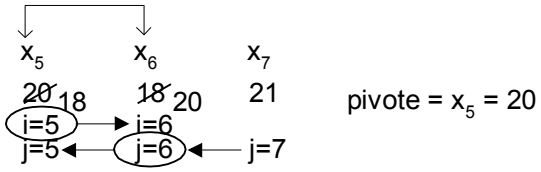

Como la lista derecha tiene un solo elemento ($x_8$), está ya ordenada. La lista izquierda, sin embargo, tiene más de un elemento, por lo que se vuelve a aplicar el procedimiento:



Ahora *i* es mayor a *j*, por lo que la lista queda dividida en 2: la lista izquierda que no tiene elementos y la lista derecha que tiene tres elementos. Al medio queda un elemento y como es único está ya con el valor correcto:

| $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|
| | 15 | 20 | 18 | 21 |

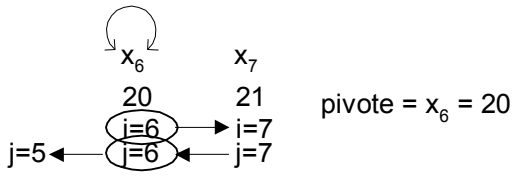
Como la lista derecha tiene más de un elemento, se vuelve a aplicar el procedimiento a la misma:




Entonces la lista queda dividida en 2:

| $x_5$ | $x_6$ | $x_7$ |
|---|---|---|
| 18 | 20 | 21 |

Como la lista izquierda tiene un solo elemento ya está ordenada. Pero como la lista derecha tiene aún 2 elementos se debe aplicar el procedimiento a la misma:




Una vez más la lista queda dividida en 3:

| $x_5$ | $x_6$ | $x_7$ |
|---|---|---|
| | 20 | 21 |

Puesto que la lista izquierda no tiene elementos, la derecha sólo uno y al medio queda un elemento, todos vectores tienen un solo elemento y en consecuencia están ya ordenadas y como no quedan listas con más de un elemento, la lista original está ya ordenada:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ |
|---|---|---|---|---|---|---|---|---|
| 7 | 10 | 14 | 15 | 18 | 20 | 21 | 28 | 30 |

Al estudiar este ejemplo, no resulta raro preguntarse si no ha existido un error al presentar este método como el más eficiente de todos, porque desde el punto de vista del esfuerzo humano parecería ser todo lo contrario, sin embargo, en todos los casos se repite siempre el mismo procedimiento, para lo cual los humanos no somos buenos pero las computadora si.

Por el contrario (y como ya se dijo) algo que consume tiempo de computación, y que en consecuencia es conveniente reducir, son los intercambios de variables, y en ese sentido el método Quick - Sort reduce considerablemente el número de intercambios. Aún en una lista tan pequeña como la del ejem-

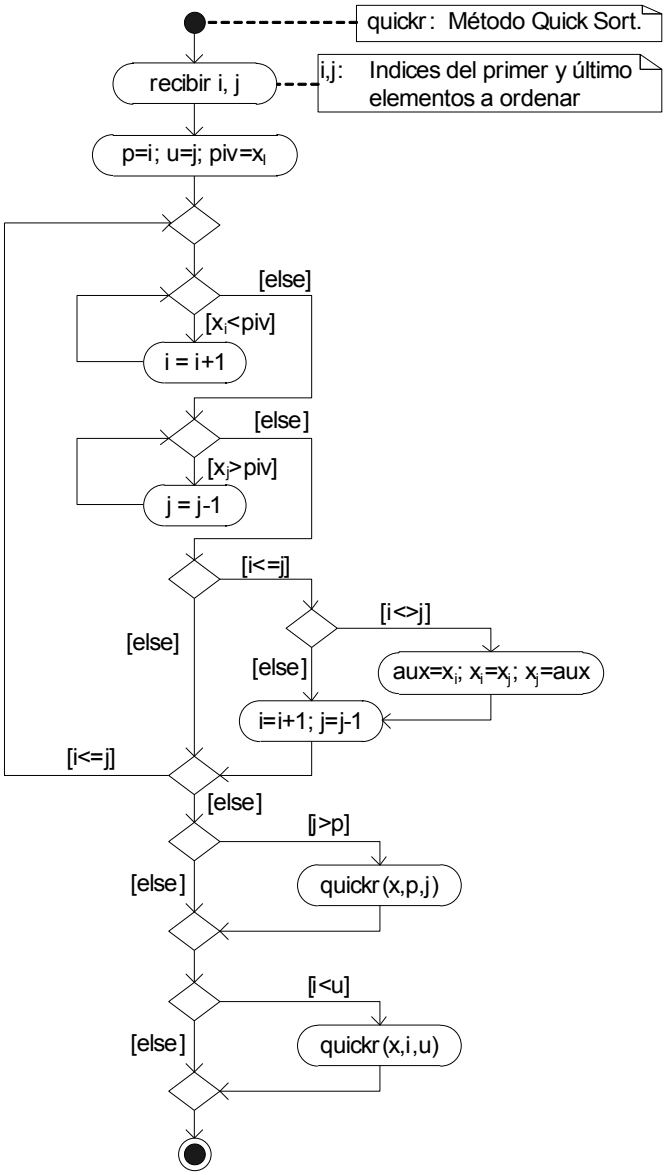
plo, con Quick Sort se requieren 7 intercambios, comparado con los 19 que serían necesarios con el método de *Burbuja*.

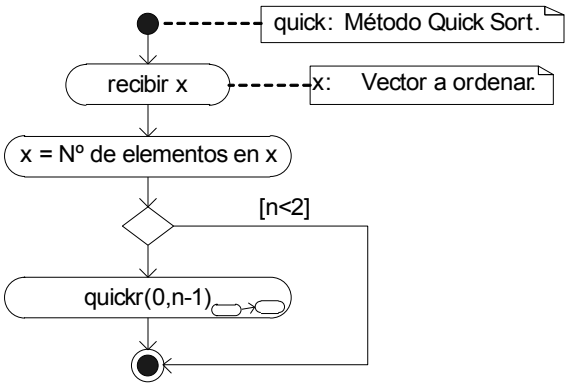
A medida que incrementa el número de elementos la diferencia se incrementa y cuando la lista es muy grande (con millones o cientos de millones de elementos) lo que a Burbuja le toma días a Quick Sort le toma minutos.

### 7.5.1. Algoritmos

El método Quick-Sort es por su naturaleza recursivo, por lo que la forma más sencilla de implementarlo es en esa forma: se aplica el procedimiento antes explicado para dividir la lista en dos sublistas. Luego el método se llama a sí mismo para ordenar los elementos que quedan en las dos listas resultantes.

El algoritmo, en forma de diagrama de actividades, se presenta a continuación y en el mismo, al igual que se hizo con el método de selección, la función recursiva (quickr) es una función anidada dentro de la función contenedora (quick):






El código que cambia, con relación a los otros métodos es el siguiente:

```
function quick(x){
  var aux,n=x.length,k=n;
  function quickr(i,j){
    var p=i,u=j,piv=x[i];
    do {
      while (x[i]<piv) i++;
      while (x[j]>piv) j--;
      if (i<=j){
        if (i!=j){
          aux=x[i]; x[i]=x[j]; x[j]=aux;
        }
        i++; j--;
      }
    } while(i<=j);
    if (j>p) quickr(p,j);
    if (i<u) quickr(i,u);
  }
  n = x.length;
  if (n>1) quickr(0,n-1);
}

function ordenar(){
  var t1=new Date();
  quick(x);
  var t2=new Date();
  var dt = t2.getTime()-t1.getTime();
  document.getElementById("vect2").innerHTML=x;
  document.getElementById("tiempo").value=dt+" milisegundos";
}
```

Para ordenar los 10000 números el método Quick-Sort requiere 2 milisegundos, es decir 185 veces menos que Burbuja, 105 veces menos que Selección, 50 veces menos que inserción y 4.5 veces menos que Shell, con lo que se comprueba es el método más rápido de los estudiados en este tema.

Simplemente para demostrar que este algoritmo (como todos los otros) puede ser empleado para ordenar otros tipos de datos, tales como registros, se ordenará el vector de 1000 elementos del método de selección, pero ahora por el campo dirección. Las partes del código que cambian con relación a dicho ejemplo son las siguientes:

```
function quick(x){
  var aux,n=x.length,k=n;
  function quickr(i,j){
    var p=i,u=j,piv=x[i].direccion;
```

```
        do {
          while (x[i].direccion<piv) i++;
          while (x[j].direccion>piv) j--;
        if (i<=j){
          if (i!=j){
            aux=x[i]; x[i]=x[j]; x[j]=aux;
          }
          i++; j--;
        }
        } while(i<=j);
        if (j>p) quickr(p,j);
        if (i<u) quickr(i,u);
      }
      n = x.length;
      if (n>1) quickr(0,n-1);
    }

  function ordenarTabla(){
      quick(r);
      var texto="<tr><th>Nombre</th><th>Dirección</th>"+
      "<th>Teléfono</th></tr>";
      for (var i=0;i<r.length;i++){
        texto+= odd(i) ? "<tr fila='impar'>" : "<tr fila='par'>";
        texto+="<td>"+r[i].nombre+"</td>"+
          "<td>"+r[i].direccion+"</td>"+
          "<td>"+r[i].telefono+"</td></tr>";
      }
      document.getElementById("tabla1").innerHTML=texto;
  }
```

Como se puede ver y al igual que sucedió con el método de Selección, la única modificación que se debe hacer al algoritmo es la de comparar los registros por el campo en que son ordenados. Como en Quick-Sort el pivote forma parte de esa comparación, su valor debe ser tomado también de dicho campo. Ordenando la tabla por el campo dirección se obtiene:

Generar Ordenar

| Nombre | Dirección | Teléfono |
|---|---|---|
| Ariza Pachacuti Helga | 29 de Septiembre Nº 449 | 6423143 |
| Rueda Benavides Benilde | 29 de Septiembre Nº 612 | 6428612 |
| Múgica Cabo Bibiana | 29 de Septiembre Nº 785 | 6403880 |
| Padilla Maita Geralidine | 29 de Septiembre Nº 931 | 6497339 |
| Pla Barco Mora | 29 de Septiembre Nº 940 | 6467137 |
| Rico Mercader Yazmin | A. Quijarro Nº 27 | 6499599 |
| Tamarit Arenas Inmaculada | A. Quijarro Nº 940 | 6411967 |

## 7.6. EJERCICIOS

Para presentar los ejemplos y ejercicios del tema, debe crear una carpeta con el nombre "tema7" y guardar en la misma todos los archivos HTML creados (ejemplos y ejercicios).

1. Cree una página HTML que genere una lista con 15000 números enteros aleatorios comprendidos entre 0 y 30000 y los ordene de forma descendente por el método de Burbuja. El programa debe mostrar el tiempo que requiere el método para encontrar los resultados.

2. Cree una página HTML que genere una lista con 2000 registros con los campos: nombre, correo electrónico, fecha de nacimiento y carnet de identidad. La página debe ordenar ascendentemente los registros por el campo nombre, empleando el método de Burbuja implementado de forma recursiva.

3. Cree una página HTML que genere una lista con 15000 números enteros aleatorios comprendidos entre 0 y 30000 y los ordene de forma descendente por el método de Selección. El programa debe mostrar el tiempo que requiere el método para encontrar los resultados.

4. Cree una página HTML que genere una lista con 2000 registros con los campos: nombre, correo electrónico, fecha de nacimiento y carnet de identidad. La página debe ordenar ascendentemente los registros por el campo correo electrónico, empleando el método de Selección.

5. Cree una página HTML que genere una lista con 15000 números enteros aleatorios comprendidos entre 0 y 30000 y los ordene de forma ascendente por el método de Inserción implementado en forma recuriva. El programa debe mostrar el tiempo que requiere el método para encontrar los resultados.

6. Cree una página HTML que genere una lista con 15000 números enteros aleatorios comprendidos entre 0 y 30000 y los ordene de forma ascendente por el método Shell implementado en forma recursiva. El programa debe mostrar el tiempo que requiere el método para encontrar los resultados.

7. Cree una página HTML que genere una lista con 15000 números enteros aleatorios comprendidos entre 0 y 30000 y los ordene de forma descendente por el método Quick-Sort. El programa debe mostrar el tiempo que requiere el método para encontrar los resultados.

8. Cree una página HTML que genere una lista con 2000 registros con los campos: nombre, correo electrónico, fecha de nacimiento y carnet de identidad. La página debe ordenar de forma descendente los registros por el campo fecha de nacimiento, empleando el método Quick-Sort.

# 8. INTERCALACIÓN Y BÚSQUEDA

En este tema se estudia la intercalación, un método que permite obtener una lista ordenada a partir de 2 listas más pequeñas ordenadas. Además, se estudian métodos de búsqueda que permiten ubicar información en listas estén ordenadas o no.

## 8.1. INTERCALACIÓN

***La intercalación es la operación por la cual se unen dos listas ordenadas para formar una tercera, igualmente ordenada, con los elementos de ambas***. Por supuesto las listas podrían ser simplemente añadidas y posteriormente ordenadas con uno de los métodos de ordenación estudiados en el anterior tema, sin embargo, el proceso de ordenación consume más tiempo que el de intercalación, además dicha operación implica la repetición innecesaria de operaciones (se vuelven a ordenar listas que ya estaban ordenadas).

La intercalación puede ser empleada también para ordenar listas muy grandes. En ese caso se ordenan segmentos de la lista empleando uno de los métodos de ordenación y luego se intercalan los segmentos ordenados, consiguiendo así la lista ordenada.

Este mismo procedimiento puede ser empleado también para mejorar la eficiencia de métodos como burbuja y selección, que como se ha visto, disminuyen su eficiencia de manera geométrica con el número de elementos, entonces, para evitar dicha disminución, se divide el proceso de ordenación en listas pequeñas (de unos 50 elementos) y a medida que las mismas son ordenadas con el método, se intercalan en el vector resultante, que finalmente contendrá toda la lista ordenada.

En la intercalación los elementos de las listas se van añadiendo a la lista resultante de forma tal que la misma siempre queda ordenada. Para ello simplemente se compara un elemento de una de las listas con otro elemento de la otra (comenzando con los primeros elementos de cada una de las listas) y se añade a la lista resultante el menor de ellos. Esta operación se repite (sin tomar en cuenta los elementos añadidos) hasta que no quedan más elementos en una de las listas. Cuando esto sucede, los elementos de la otra lista (la que quedó con elementos) se añaden al final de la lista resultante.

Por ejemplo para intercalar las siguientes listas:

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| x | 6 | 15 | 23 | 30 | 34 | 40 |
| y | 5 | 10 | 25 | 26 | 30 | |

Se comienza comparando los primeros elementos de ambas listas (6 con 5) y puesto que 5 es menor que 6, se añade este valor a la lista resultante (z):

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| x | 6 | 15 | 23 | 30 | 34 | 40 |
| y | ~~5~~ | 10 | 25 | 26 | 30 | |
| z | 5 | | | | | |

Entonces se compara 6 con 10 y puesto que 6 es menor que 10, dicho valor es añadido a la lista:

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| x | ~~6~~ | 15 | 23 | 30 | 34 | 40 |
| y | ~~5~~ | 10 | 25 | 26 | 30 | |
| z | 5 | 6 | | | | |

Ahora se compara 15 con 10 y dado que 10 es menor a 15 se añade 10 a la lista resultante:

| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| x | ~~6~~ | 15 | 23 | 30 | 34 | 40 |
| y | ~~5~~ | ~~10~~ | 25 | 26 | 30 | |
| z | 5 | 6 | 10 | | | |

Proseguimos de esa manera: se compara 15 con 25 (se añade el número 15), luego 23 con 25 (se añade el número 23), 30 con 25 (se añade el número 25), 30 con 26 (se añade el número 26), 30 con 30 (como 30 no es menor a 30, se añade el número 30 de la segunda lista).

En este punto la lista "y" se queda sin elementos:

| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| x | ~~6~~ | ~~15~~ | ~~23~~ | 30 | 34 | 40 | | | | | |
| Y | ~~5~~ | ~~10~~ | ~~25~~ | ~~26~~ | ~~30~~ | | | | | | |
| z | 5 | 6 | 10 | 15 | 23 | 25 | 26 | 30 | | | |

Entonces los elementos restantes de la lista "x" (desde $x_4$ hasta $x_6$) son añadidos al final de la lista resultante, con lo que se obtiene la lista intercalada *("z")*, que como se ve está ordenada:

| I | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| X | ~~6~~ | ~~15~~ | ~~23~~ | ~~30~~ | ~~34~~ | ~~40~~ | | | | | |
| Y | ~~5~~ | ~~10~~ | ~~25~~ | ~~26~~ | ~~30~~ | | | | | | |
| z | 5 | 6 | 10 | 15 | 23 | 25 | 26 | 30 | 30 | 34 | 40 |

### *8.1.1. Algoritmo y código*

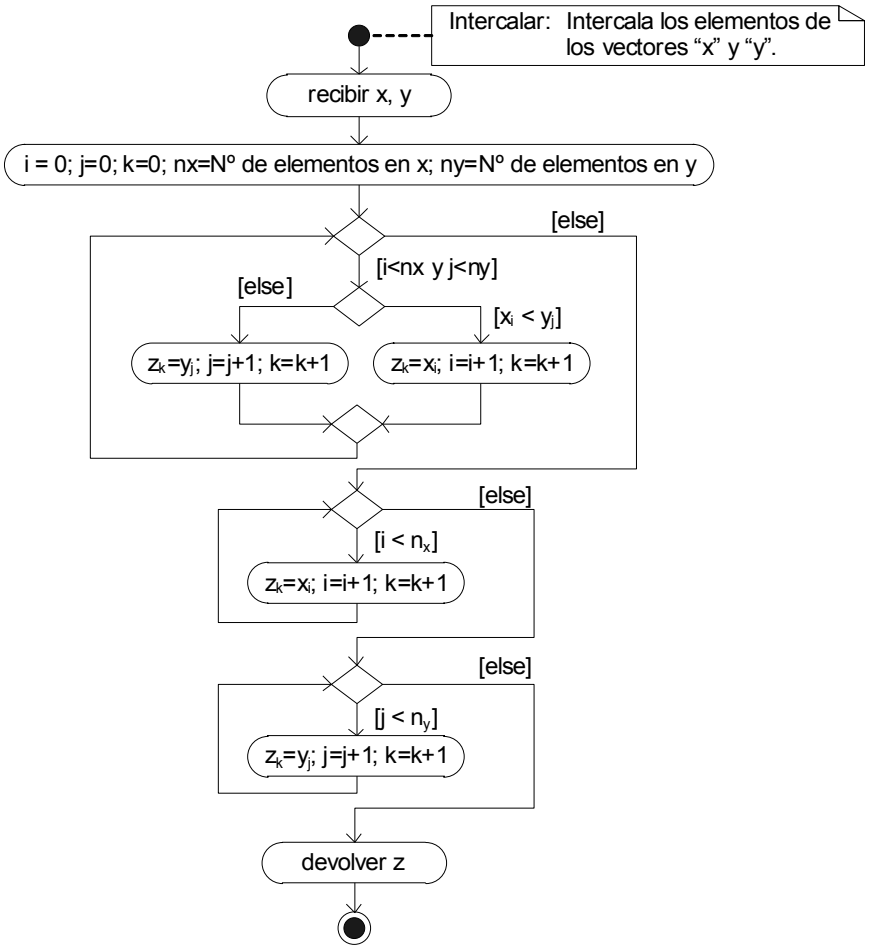
El algoritmo para intercalar ascendentemente dos listas de datos simples se muestra en el diagrama de actividades de la siguiente página.

Para que el algoritmo pueda intercalar vectores con diferentes tipos de datos y pueda intercalarlos no sólo de forma ascendente sino también descendente, debe recibir no sólo los vectores a intercalar, sino también la función de comparación. La función de comparación (al igual que en el método "sort"), es una función, casi siempre anónima, que recibe dos argumentos y devuelve un valor positivo si el primer argumento es mayor que el segundo, cero si son iguales y un valor negativo si el primer argumento es menor que el segundo.

La librería "sis101.js" ha sido renombrada a "hpvlib.js" y en la misma se ha añadido una función para el método de intercalación ("intercalar"), que recibe los vectores a intercalar y la función de comparación ("fc"):

```
function intercalar(x,y,fc){
  if (fc==null) fc=function(a,b){return a==b?0:a>b?1:-1;};
  var nx=x.length,ny=y.length,nz=nx+ny,i=0,j=0,k=0,z=new Array(nz);
  while (i<nx && j<ny)
    if (fc(x[i],y[j])<0) z[k++]=x[i++]; else z[k++]=y[j++];
  while (i<nx) z[k++]=x[i++];
  while (j<ny) z[k++]=y[j++];
  return z;
}
```

Como se puede ver, si no se manda la función de comparación (si "fc" es nulo), se crea una función de comparación que devuelve 0 si los valores son iguales, 1 si el primero es mayor que el segundo y -1 si es menor, es decir que por defecto se intercalan los vectores en forma ascendente.




Por ejemplo, dada las siguientes dos listas, generadas aleatoriamente y ordenadas con “sort”:

```
>>randseed=0;
>>x=round(rand(15,1,50)).sort(function(a,b){return a-b;})
[4, 11, 11, 16, 19, 27, 28, 30, 32, 34, 36, 37, 41, 42, 48]
>>y=round(rand(20,1,50)).sort(function(a,b){return a-b;})
[6, 6, 7, 9, 14, 18, 20, 21, 27, 29, 31, 38, 39, 39, 41, 42, 43, 44, 48,
48]
```

Se obtiene una tercera lista con los elementos intercalados de ambas, llamando a “intercalar”:

```
>>z=intercalar(x,y,function(a,b){return a-b;})
[4, 6, 6, 7, 9, 11, 11, 14, 16, 18, 19, 20, 21, 27, 27, 28, 29, 30, 31, 32,
34, 36, 37, 38, 39, 39, 41, 41, 42, 42, 43, 44, 48, 48, 48]
```

En este ejemplo no es imprescindible mandar la función de comparación, porque la función que se manda es la función por defecto del método. Sin embargo, si no se comparan los elementos, sino sólo algunos de sus campos, o se intercala según otro criterio, por ejemplo de forma descendente, la función de comparación es necesaria.

Así, el anterior ejemplo, con listas ordenadas de forma descendente, es:

```
>>randseed=0;
>>x=round(rand(15,1,50)).sort(function(a,b){return b-a;})
[48, 42, 41, 37, 36, 34, 32, 30, 28, 27, 19, 16, 11, 11, 4]
```

```
>>y=round(rand(20,1,50)).sort(function(a,b){return b-a;})
[48, 48, 44, 43, 42, 41, 39, 39, 38, 31, 29, 27, 21, 20, 18, 14, 9, 7, 6,
6]
>>z=intercalar(x,y,function(a,b){return b-a;})
[48, 48, 48, 44, 43, 42, 42, 41, 41, 39, 39, 38, 37, 36, 34, 32, 31, 30,
29, 28, 27, 27, 21, 20, 19, 18, 16, 14, 11, 11, 9, 7, 6, 6, 4]
```

Las funciones de comparación son útiles no sólo en la intercalación, sino principalmente en los métodos de ordenación, pues con las mismas se pueden crear módulos de carácter más general. Empleando estas funciones se han añadido, a la librería "hpvlib.js", funciones para los métodos de burbuja ("burbuja"), selección ("seleccion"), inserción ("insercion"), shell ("shell") y quick-sort ("quick"). Todas ellas pueden recibir el vector a ordenar y la función de comparación.

Al igual que con "intercalar", si no se manda la función de comparación, los métodos de ordenación generan una función por defecto, la misma que en la función "intercalar". En otras palabras, si no se manda la función de comparación, los elementos se ordenan de forma ascendente.

Por ejemplo si se genera una lista con 1000 elementos enteros aleatorios comprendidos entre 1 y 5000:

```
>>randseed=1; x=round(rand(1000,1,5000));
```

Puede ser ordenada de forma descendente, con el método Shell, escribiendo la siguiente instrucción:

```
>>shell(x,function(a,b){return b-a;});
```

Con lo que los elementos del vector "x" quedan ordenados descendentemente:

```
>>x
[5000, 4998, 4993, 4989, 4987, 4966, 4964, 4957, 4953, 4949, 4942, 4928,
4916, 4916, 4911, 4909, 4907, 4905, 4901, 4900, 4899, 4896, 4895, 4891,
4887, 4877, 4875, 4873, 4869, 4859, 4858, 4858, 4853, 4853, 4841, 4838,
4837, 4834, 4831, 4829, 4812, 4807, 4796, 4793, 4793, 4787, 4780, 4774,
4772, 4770, 4769, 4762, 4753, 4751, 4741, 4738, 4730, 4729, 4719, 4717,
4709, 4709, 4706, 4698, 4698, 4694, 4684, 4679, 4676, 4674, 4671, 4670,
4669, 4669, 4666, 4666, 4666, 4656, 4654, 4645, 4644, 4638, 4638, 4637,
4633, 4633, 4632, 4630, 4626, 4623, 4622, 4622, 4620, 4618, 4614, 4613,
4611, 4608, 4587, 4583, 4582, 4579, 4574, 4564, 4561, 4549, 4548, 4525,
4522, 4521, 4514, 4513, 4507, 4502, 4501, 4496, 4495, 4490, 4488, 4482,
4480, 4478, 4477, 4471, 4441, 4437, 4434, 4427, 4425, 4423, 4422, 4415,
4410, 4405, 4399, 4395, 4394, 4385, 4374, 4368, 4367, 4362, 4358, 4357,
4348, 4338, 4334, 4328, 4328, 4326, 4322, 4318, 4317, 4312, 4310, 4306,
4303, 4303, 4289, 4270, 4262, 4253, 4234, 4230, 4219, 4216, 4214, 4205,
4204, 4203, 4202, 4197, 4195, 4190, 4182, 4180, 4180, 4176, 4176, 4175,
4167, 4163, 4162, 4151, 4143, 4142, 4142, 4139, 4139, 4138, 4133, 4131,
4123, 4122, 4106, 4102, 4096, 4095, 4093, 4085, 4083, 4070, 4069, 4066,
4064, 4062, 4057, 4054, 4054, 4045, 4045, 4045, 4045, 4044, 4042, 4036,
4035, 4023, 4022, 4019, 4017, 3996, 3996, 3991, 3990, 3985, 3979, 3975,
3973, 3969, 3966, 3961, 3947, 3944, 3943, 3937, 3936, 3935, 3929, 3925,
3924, 3915, 3911, 3897, 3896, 3894, 3889, 3872, 3857, 3850, 3839, 3836,
3829, 3824, 3821, 3816, 3815, 3812, 3799, 3797, 3769, 3763, 3757, 3757,
3754, 3751, 3751, 3739, 3739, 3737, 3731, 3729, 3701, 3700, 3694, 3685,
3677, 3677, 3674, 3670, 3665, 3657, 3657, 3649, 3644, 3644, 3639, 3637,
3636, 3631, 3630, 3619, 3617, 3613, 3610, 3610, 3600, 3597, 3595, 3592,
3581, 3580, 3579, 3576, 3571, 3568, 3568, 3567, 3567, 3566, 3562, 3561,
3550, 3547, 3541, 3536, 3536, 3533, 3532, 3530, 3528, 3520, 3494, 3493,
3486, 3484, 3470, 3463, 3461, 3460, 3453, 3447, 3443, 3440, 3439, 3435,
3424, 3424, 3418, 3418, 3401, 3363, 3355, 3349, 3348, 3338, 3337, 3336,
```

```
3330, 3324, 3324, 3309, 3298, 3294, 3294, 3291, 3290, 3269, 3267, 3260,
3260, 3253, 3251, 3245, 3242, 3240, 3233, 3217, 3214, 3207, 3200, 3195,
3189, 3188, 3185, 3180, 3173, 3170, 3168, 3162, 3156, 3153, 3149, 3145,
3136, 3134, 3132, 3125, 3124, 3117, 3116, 3115, 3110, 3100, 3097, 3087,
3086, 3076, 3076, 3067, 3059, 3053, 3045, 3043, 3040, 3036, 3036, 3027,
3021, 3018, 3013, 2994, 2988, 2988, 2981, 2970, 2966, 2962, 2956, 2953,
2941, 2939, 2934, 2933, 2927, 2923, 2919, 2914, 2893, 2893, 2891, 2887,
2883, 2883, 2882, 2880, 2875, 2869, 2869, 2868, 2864, 2864, 2859, 2858,
2852, 2849, 2848, 2848, 2842, 2835, 2831, 2826, 2826, 2815, 2810, 2803,
2800, 2799, 2794, 2794, 2783, 2777, 2776, 2771, 2767, 2761, 2759, 2751,
2750, 2750, 2749, 2747, 2745, 2740, 2738, 2732, 2727, 2721, 2720, 2719,
2701, 2680, 2680, 2677, 2676, 2672, 2667, 2663, 2655, 2649, 2639, 2633,
2621, 2613, 2611, 2604, 2604, 2603, 2595, 2593, 2579, 2577, 2571, 2565,
2564, 2562, 2562, 2555, 2548, 2542, 2539, 2538, 2534, 2525, 2525, 2522,
2513, 2508, 2501, 2490, 2488, 2484, 2483, 2482, 2481, 2459, 2458, 2456,
2455, 2452, 2440, 2436, 2436, 2432, 2426, 2423, 2422, 2418, 2417, 2417,
2411, 2410, 2404, 2391, 2383, 2379, 2372, 2370, 2370, 2370, 2367, 2345,
2340, 2339, 2332, 2326, 2321, 2309, 2307, 2305, 2304, 2302, 2294, 2288,
2281, 2280, 2274, 2267, 2259, 2258, 2257, 2253, 2247, 2241, 2236, 2235,
2228, 2226, 2221, 2219, 2215, 2215, 2215, 2211, 2204, 2199, 2196, 2193,
2182, 2167, 2164, 2162, 2161, 2161, 2161, 2157, 2155, 2150, 2147, 2147,
2133, 2133, 2128, 2118, 2113, 2112, 2100, 2084, 2070, 2070, 2069, 2066,
2065, 2062, 2060, 2052, 2048, 2042, 2040, 2040, 2031, 2030, 2029, 2019,
1991, 1985, 1984, 1977, 1957, 1956, 1952, 1949, 1948, 1947, 1942, 1921,
1919, 1918, 1918, 1912, 1906, 1905, 1899, 1891, 1888, 1885, 1883, 1871,
1870, 1867, 1864, 1859, 1856, 1853, 1852, 1851, 1850, 1847, 1838, 1832,
1829, 1823, 1821, 1812, 1803, 1802, 1801, 1796, 1790, 1789, 1779, 1776,
1766, 1765, 1765, 1762, 1761, 1755, 1754, 1750, 1746, 1737, 1737, 1719,
1712, 1709, 1692, 1690, 1673, 1672, 1659, 1646, 1634, 1625, 1622, 1619,
1609, 1608, 1603, 1595, 1594, 1573, 1571, 1570, 1569, 1565, 1561, 1560,
1548, 1525, 1522, 1513, 1512, 1502, 1498, 1498, 1492, 1492, 1490, 1471,
1470, 1467, 1444, 1430, 1428, 1411, 1411, 1409, 1403, 1401, 1397, 1393,
1389, 1383, 1371, 1370, 1369, 1368, 1365, 1362, 1354, 1354, 1348, 1338,
1330, 1324, 1324, 1321, 1320, 1318, 1313, 1311, 1305, 1282, 1280, 1272,
1271, 1264, 1257, 1251, 1249, 1247, 1242, 1239, 1236, 1230, 1221, 1219,
1212, 1211, 1210, 1203, 1202, 1200, 1199, 1187, 1173, 1171, 1170, 1168,
1154, 1153, 1150, 1149, 1138, 1138, 1137, 1134, 1133, 1133, 1131, 1128,
1125, 1122, 1117, 1117, 1112, 1106, 1104, 1101, 1100, 1099, 1098, 1090,
1080, 1073, 1073, 1069, 1060, 1060, 1058, 1047, 1046, 1042, 1039, 1037,
1030, 1027, 1011, 1006, 1005, 1002, 986, 981, 975, 964, 962, 956, 954, 953,
947, 945, 940, 936, 936, 931, 928, 922, 908, 906, 898, 888, 885, 883, 878,
872, 858, 851, 847, 847, 832, 825, 822, 810, 807, 802, 788, 777, 777, 777,
770, 753, 746, 737, 728, 728, 726, 724, 722, 719, 713, 712, 708, 705, 705,
705, 705, 697, 690, 682, 679, 657, 655, 654, 652, 635, 627, 626, 619, 617,
613, 610, 593, 592, 590, 586, 572, 570, 570, 566, 564, 561, 557, 549, 540,
538, 531, 528, 516, 514, 514, 505, 497, 495, 494, 489, 481, 473, 473, 468,
464, 459, 458, 452, 441, 439, 432, 420, 416, 414, 396, 392, 391, 371, 369,
369, 365, 359, 358, 355, 350, 346, 345, 342, 332, 297, 282, 273, 268, 265,
260, 255, 253, 245, 242, 236, 227, 227, 225, 223, 222, 220, 216, 212, 200,
192, 190, 187, 185, 184, 173, 171, 169, 167, 165, 163, 153, 143, 132, 128,
124, 119, 117, 105, 103, 102, 96, 92, 80, 75, 66, 58, 58, 56, 54, 54, 32,
27, 16, 7]
```

Como ya se dijo, la intercalación puede ser empleada conjuntamente los métodos de ordenación (principalmente los métodos directos) para optimizar el proceso de ordenación. En ese caso, sin embargo, las listas a intercalar así como la lista resultante deben estar en el mismo vector por lo que es necesario modificar los algoritmos tanto del método de ordenación como de intercalación para tomar en cuenta ese hecho.

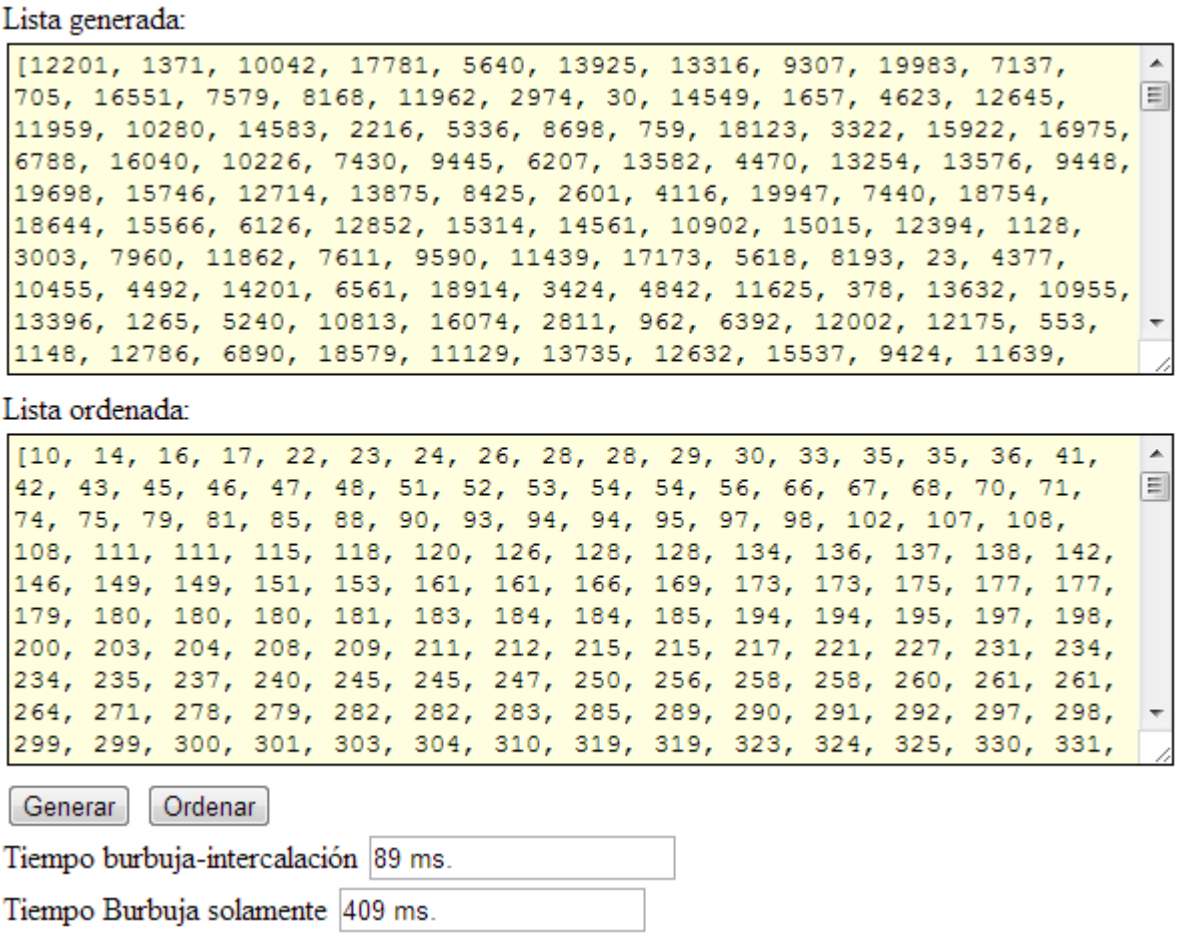
Por ejemplo en la siguiente aplicación se emplea el método de Burbuja conjuntamente el de intercalación para ordenar una lista de 10000 números enteros generados al azar y comprendidos entre 1 y 20000.

Con ese fin se ordena la lista, con el método de burbuja, en grupos de 50 elementos, los cuales se intercalan luego con el método de intercalación. Para comparar el tiempo que toma el método combinado, con relación al método de Burbuja por sí solo, se ordena también la lista con este método:

```
<!DOCTYPE html>
<html>
<head>

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Ejemplo Intercalación</title>

  <script src="hpvlib.js"></script>
  <script>
    var v;

    //Ordena el vecgtor "x" con el método de burbuja e intercalación
    function burbujai(x,fc){
      if (fc==null) fc=new function(x){return a-b;};
      var i=50,n=x.length;
      //Ordena el vector "x" desde el índice "p" hasta el úndice "u"
      function burb(p,u){
        for (var i=u;i>p;i--){
          s=true;
          for (var j=p;j<i;j++)
            if (fc(x[j],x[j+1])>0){
              aux=x[j]; x[j]=x[j+1]; x[j+1]=aux;
              s=false;
             }
          if (s) break;
        }
      }
      //Intercala los elementos del vector "x" comprendidos entre el
      //índice "p" y "u" con la lista comprendida entre 0 y "p-1"
      function inter(p,u){
                              var k=0,i=0,j=p,l,aux;
                                while (k<u && j<=u)
                              if (fc(x[k],x[j])<0) k++;
                                              else {
                 aux=x[j]; for(l=j;l>k;l--) x[l]=x[l-1]; x[k++]=aux;j++;}
      }
      if (n<=50) {burb(0,n-1);return;}
      burb(0,49);
      while (i+49<n){
        burb(i,i+49);
        inter(i,i+49);
        i+=50;
      }
      if (i<n){
        burb(i,n-1);
        inter(i,n-1);
      }
    }
```

```
      function genvec(){
        randseed=10;
        v=round(rand(10000,1,20000));
      }

      function generar(){
        genvec();
        document.getElementById("ta1").innerHTML=v;
      }

      function ordenar(){
        var t1,t2,dt;
        t1=new Date();
        burbujai(v,function(a,b){return a-b;});
        t2=new Date();
        dt=t2.getTime()-t1.getTime();
        document.getElementById("ta2").innerHTML=v;
        document.getElementById("it1").value=dt+" ms.";
        genvec();
        t1 = new Date();
        burbuja(v,function(a,b){return a-b;});
        t2 = new Date();
        dt=t2.getTime()-t1.getTime();
        document.getElementById("it2").value=dt+" ms.";
      }

    </script>

  <style type="text/css">
    textarea {
      background-color:lightyellow;
    }
  </style>

  </head>

  <body>

    <label for="ta1">Lista generada:</label>
    <textarea id="ta1" rows="10" cols="70"></textarea><br>

    <label for="ta2">Lista ordenada:</label>
    <textarea id="ta2" rows="10" cols="70"></textarea><br>

    <button id="bt1" onclick="generar()">Generar</button>
    <button id="bt2" onclick="ordenar()">Ordenar</button>

    <br><label for="it1">Tiempo burbuja-intercalación</label>
    <input type="text" id="it1"><br>
    <label for="it2">Tiempo Burbuja solamente</label>
    <input type="text" id="it2"><br>

  </body>

  </html>
```

Abriendo la página en un navegador, generando el vector y ordenando el mismo y haciendo correr el programa se obtiene algo parecido a:




Como se puede ver el método de Burbuja, combinado con el método de intercalación, es 4.6 veces más rápido que el método de Burbuja por sí solo.

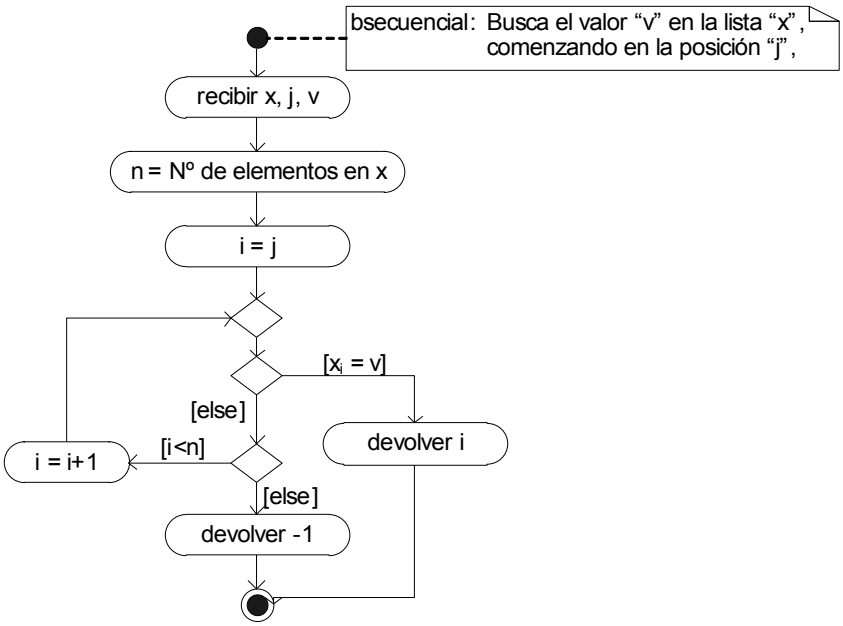
## 8.2. BÚSQUEDA SECUENCIAL

Cuando se trabaja con vectores, ubicar un valor implica determinar la posición (índice o dirección de memoria) del elemento que contiene ese valor.

En realidad sólo se puede hablar de métodos de búsqueda cuando la información está ordenada, de lo contrario, no queda otra alternativa que buscar la información de forma secuencial, es decir comparando elemento a elemento, hasta encontrar el valor buscado o hasta que ya no quedan más elementos en la lista. A este procedimiento que se conoce como búsqueda secuencial o lineal y es la única alternativa disponible cuando los elementos no están ordenados.

Como en una lista pueden existir valores duplicados, una vez ubicado un elemento, se puede continuar la búsqueda, comenzando con el elemento que se encuentra después del elemento ubicado, hasta ubicar la posición de otro de los valores o hasta que ya no queden más elementos en la lista.

A pesar de ser intuitivo y no constituir un método propiamente, la búsqueda secuencial es de utilidad práctica, porque gran parte de la información existente, como ser el texto de documentos y libros, se encuentra desordenada, por lo que la única alternativa para ubicar información en los mismos es mediante la búsqueda secuencial o lineal. El algoritmo correspondiente a la búsqueda secuencial se presenta en la siguiente página.




Al igual que con el método de intercalación y los de ordenación, para contar con un módulo de carácter más general, es conveniente realizar la búsqueda no directamente con la igualdad, sino mediante una función de comparación que devuelva 0 si los valores son iguales, positivo si el primer valor es mayor al segundo y negativo en caso contrario. Dicha función, añadida a la librería "hpvlib.js" es:

```
function bsecuencial(x,j,v,fc){
  if (fc==null) fc=function(a,b){return a==b ? 0: 1;};
  var n=x.length,i=j;
  do {
    if (!fc(v,x[i])) return i;
    i++
  } while(i<n);
  return -1;
}
```

Como se puede ver, si no se le manda una función de comparación, el método genera uno por defecto, la cual devuelve 0 si el valor buscado y el elemento comparado son iguales.

Por ejemplo, si se genera una lista con 200 números enteros comprendidos entre 1 y 300:

```
>>>randseed=1; x=round(rand(200,1,300))
[76, 164, 103, 286, 230, 39, 271, 212, 231, 156, 235, 207, 65, 124, 91,
123, 5, 7, 155, 300, 80, 286, 237, 136, 43, 240, 299, 148, 134, 96, 128,
44, 250, 107, 95, 286, 209, 112, 32, 262, 219, 208, 89, 243, 43, 115, 8,
260, 165, 238, 85, 65, 160, 242, 239, 198, 257, 212, 176, 255, 110, 221,
61, 57, 157, 99, 178, 108, 195, 298, 201, 173, 243, 259, 159, 137, 109,
278, 106, 174, 55, 273, 280, 213, 165, 252, 279, 207, 216, 79, 222, 295,
36, 38, 16, 275, 77, 102, 23, 165, 79, 3, 128, 73, 40, 91, 206, 259, 177,
7, 187, 212, 198, 247, 23, 5, 265, 33, 153, 67, 66, 167, 75, 249, 172, 116,
106, 293, 15, 63, 55, 195, 243, 183, 184, 174, 202, 106, 278, 236, 85, 83,
113, 246, 11, 254, 116, 188, 187, 118, 140, 36, 77, 290, 225, 120, 166,
281, 70, 191, 28, 52, 190, 190, 192, 288, 294, 15, 11, 164, 245, 83, 224,
109, 29, 1, 30, 274, 218, 133, 44, 102, 243, 64, 77, 187, 157, 266, 89,
149, 198, 73, 166, 295, 264, 33, 84, 188, 173, 288]
```

Se puede encontrar la posición del primer número 77 con:

```
>>bsecuencial(x,0,77)
96
```

Por lo tanto, el primer número 77 se encuentra en la posición 97 (recuerde que los vectores en Javascript el primer elemento es el elemento 0). Se puede buscar otro elemento que tenga el número 77 comenzando la búsqueda en la posición 97 (96+1):

```
>>bsecuencial(x,97,77)
152
```

Y continuar de esa manera la búsqueda de otros elementos que tengan el número 77:

```
>>bsecuencial(x,153,77)
184
>>bsecuencial(x,185,77)
-1
```

Como se puede ver existe otro elemento (en la posición 185) que tiene el número 77, luego "bsecuencial" devuelve "-1", informando que ya no existen más elementos con ese número.

Se puede realizar la búsqueda de todos los elementos que tienen el número 77 y devolver un vector con las posiciones de dichos elementos, empleando un ciclo "while":

```
>r=[]; p=0; while((p=bsecuencial(x,p,77))>0){r.push(p); p++;}; r
[96, 152, 184]
```

En estos casos, funciona bien la función de comparación por defecto, sin embargo, si por ejemplo se tiene la siguiente lista con apellidos:

```
>>randseed=0; v=new Array(100); for(i=0;i<100;i++) v[i]=generarApellido();
v
[Canquichoque, Peinado, Losada, Pocoata, Rosado, Anglada, Fabregat, Salom,
Viana, Lledó, Calzada, Paco, Navas, Coello, Mesa, Aucapoma, Manzano, Boada,
Dávila, Garriga, Quisicala, Queso, Chachajaque, Seguí, Rodríguez, Leal, Sa-
las, Sirpa, Montes, Baeza, Ferrán, Varela, Aznar, Porras, Velasco, Casares,
Figueras, Almansa, Trillo, Pereira, Moreno, Mascaró, Boada, Barón, Morell,
Barrios, Valle, Huanca, Alfonso, Ledesma, Llanquechoque, Colque, Chacón,
Casado, Duarte, Mateo, Zabala, Mateu, Tello, Caparrós, Goicoechea, Naranjo,
Crespo, Coello, Camino, Ponce, Montaña, Adadia, Solano, Rebollo, Inca, Be-
nito, Carreño, Almazán, Villalba, Esteve, Aruquipa, Quero, Montoya, Apari-
cio, Valle, Fiol, Espejo, Uriarte, Ayala, García, Bayón, Caballero, Mori-
llo, Villarroel, Albero, Gascón, Sinchiroca, Aquise, Canqui, Carvajal, Pe-
rea, Gargallo, Inca, Franch]
```

Y se quiere encontrar el primer elemento cuyo apellido comienza con "Pe" (no apellidos iguales a "Pe", sino los que comienza con "Pe"). En ese caso es necesario mandar la función de comparación para se compare sólo los dos primeros caracteres de los apellidos con el valor buscado, como sucede con la siguiente instrucción:

```
>>bsecuencial(v,0,"Pe",function(a,b){return a==b.substr(0,2) ? 0 : 1;})
1
```

Que como se puede ver, informa que el segundo elemento tiene un nombre que comienza con "Pe". Para obtener la lista de todos los elementos cuyos apellidos comienzan con "Pe", se procede como en el ejemplo anterior:

```
 >>r=[]; p=0; while((p=bsecuencial(v,p,"Pe",function(a,b){return
a==b.substr(0,2) ? 0: 1;}))>0){r.push(p); p++;}; r
[1, 39, 96]
```

Y como se puede ver los elementos con los índices 1, 39 y 96 contienen apellidos que comienzan con "Pe".

## 8.3. MÉTODO DE LA BISECCIÓN

Como ya se dijo, sólo se pueden desarrollar verdaderos métodos de búsqueda si los datos están ordenados. Uno de dichos métodos es el método de búsqueda binaria. En la explicación de este método se asumirá que los datos están ordenados ascendentemente.

***En este método se compara el valor buscado con el elemento central del vector. Si el valor buscado es igual al elemento central el proceso concluye, caso contrario se divide el vector en dos y se repite el procedimiento en la mitad donde es probable que se encuentre el valor buscado***. Este procedimiento se repite (dividiendo el vector en dos en cada iteración) hasta que el valor es encontrado o hasta que la mitad donde se realiza la búsqueda queda sin elementos.

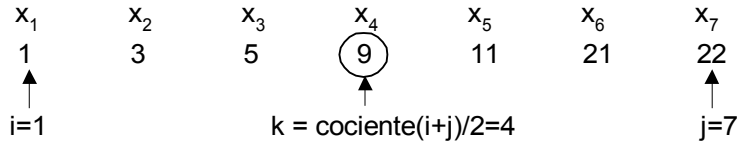
Se sabe en cuál de las mitades se encuentre el valor buscado comparándolo con el elemento central: Si es mayor al elemento central entonces se encuentra en la mitad derecha (o superior) y si es menor en la mitad izquierda (o inferior).

Cuando en el vector existen dos o más elementos con el valor buscado, el método de búsqueda binaria ubica uno de esos elementos, pero no garantiza que dicho elemento sea el primero. Por consiguiente, una vez ubicado el elemento, es necesario recorrer el vector hacia atrás hasta que el valor de un elemento es diferente al buscado.

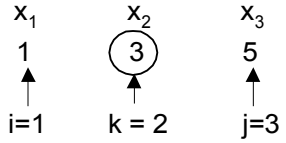
Para comprender mejor el procedimiento que se sigue en el método se ubicará el número 5 en el siguiente vector:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ |
|---|---|---|---|---|---|---|
| 1 | 3 | 5 | 9 | 11 | 21 | 22 |

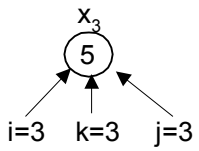
En el proceso se emplean 3 variables: *"i"* para el índice del primer elemento; *"j"* para el índice del último elemento y *"k"* para el índice del elemento central. El valor de *"k"* se calcula como el cociente de la división de *i+j* entre 2:




Como el elemento central es mayor al valor buscado (5) se repite el procedimiento en la mitad izquierda, por lo tanto, para repetir el proceso, el índice "j" toma el valor del elemento central "k" menos 1 (4-1=3):

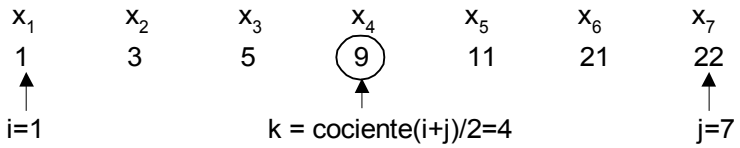



Ahora el elemento central es menor al valor buscado (5), por lo que el valor buscado debe encontrarse en la mitad derecha, entonces para repetir el proceso, el primer índice "i" toma el valor del elemento central "k" más 1 (2+1=3):

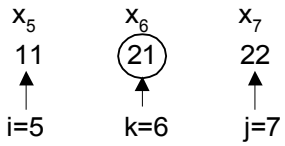



En la lista derecha queda un elemento, y como el mismo es igual al valor buscado se ha encontrado la posición de mismo, es decir el valor de "k" (3).

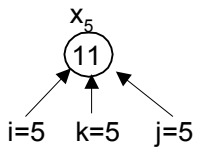
Para comprender mejor el método se volverá a aplicar el mismo para ubicar un nuevo valor, el número 11. Como de costumbre el proceso comienza tomando en cuenta el elemento central del vector:




Puesto que el elemento central es menor al valor buscado (11) se repite el procedimiento en el vector derecho (i=k+1=5):




Dado que el elemento central es mayor al valor buscado (11) se repite el procedimiento con el vector izquierdo (j=k-1=5), que tiene un solo elemento:



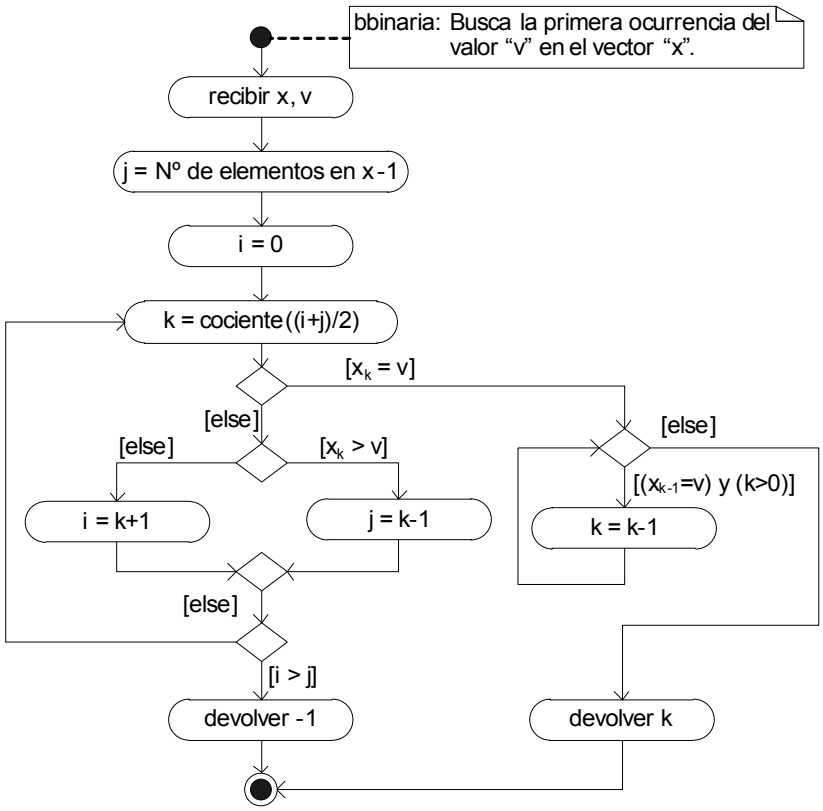

Y como el mismo es igual al valor buscado el proceso concluye habiéndose ubicado el valor en la posición 5 (el valor de *k*).

Si se busca un elemento que no existe en el vector, por ejemplo el número 12, aplicando el procedimiento se llega al mismo vector que en el caso anterior (cuando se busca el número 11). En este caso sin embargo el elemento que queda no es igual al valor buscado y al no existir más elementos, se concluye que el valor buscado no existe en la lista. Por lo tanto se sabe que el valor buscado no se encuentra en la lista cuando en la misma queda un solo elemento y el mismo no es igual al valor buscado.

### *8.3.1. Algoritmo y código*

Como ocurre con los métodos de ordenación, el método de búsqueda binaria puede ser programado tanto en forma iterativa como recursiva, el algoritmo iterativo se presenta en la siguiente página. Para su codificación y como se ha hecho con los anteriores métodos, es conveniente realizar la comparación empleando una función en lugar del operador de igualdad. Dicho código, añadido a la librería "hpvlib.js" es el siguiente:

```
function bbinaria(x,v,fc){
  if (fc==null) fc=function(a,b){return a==b?0:a>b?1:-1;}
  var j=x.length-1,i=0,k;
```




```
    do{
      k=quot(i+j,2);
      if (!fc(v,x[k])) {
        while (!fc(v,x[k-1]) && k>0) k--; return k;}
      if (fc(v,x[k])<0) j=k-1; else i=k+1;
    } while(i<=j);
    return -1;
  }
```

Como de costumbre, si no se manda una función de comparación, el método genera una por defecto, la cual funciona bien si el valor buscado se compara directamente con los valores (como un todo) de los elementos.

Por ejemplo, si se genera y ordena un vector con 100 números enteros comprendidos entre 1 y 200:

```
>>randseed=1; x=round(rand(100,1,200)).sort(function(a,b){return a-b;})
[4, 5, 6, 11, 16, 21, 25, 26, 26, 29, 29, 30, 37, 38, 41, 44, 44, 51, 52,
53, 54, 57, 60, 61, 64, 64, 66, 68, 69, 71, 71, 72, 73, 73, 75, 77, 82, 83,
85, 90, 91, 92, 99, 104, 104, 105, 106, 107, 110, 110, 110, 110, 116, 116,
118, 119, 130, 132, 134, 138, 138, 139, 140, 141, 141, 142, 144, 146, 147,
148, 153, 154, 157, 158, 159, 159, 160, 162, 162, 162, 167, 168, 170, 172,
173, 173, 174, 181, 182, 183, 185, 186, 187, 191, 191, 191, 197, 199, 199,
200]
```

Y se quiere ubicar la posición (el índice) del primer elemento que contiene el número 71, la función de comparación por defecto es suficiente:

```
>>bbinaria(x,71)
29
```

Entonces el primer número 71 se encuentra en el elemento con el índice 29. Sin embargo, si se genera y ordena una lista con 100 nombres femeninos:

```
>>randseed=1; y=[]; for(i=0;i<100;i++) y.push(generarNombreF()); y.sort()
[Adelina, Afra, Agostina, Almudena, Anelis, Argentina, Atocha, Aurora,
Aviv, Begoña, Begoña, Belinda, Carolina, Caterina, Ciara, Cleopatra,
Cleopatra, Darel, Dariana, Denise, Desdemona, Edeltrudis, Electra, Elia,
Emiliana, Enevi, Ester, Eunice, Evanan, Eyda, Fabia, Fatima, Faustina,
Felicia, Florencia, Fuencisla, Gloria, Graciela, Guiomar, Iklerk, Imelda,
Ines, Jemiel, Kariett, Karina, Karyme, Larisa, Lavinia, Libia, Lidia,
Lidia, Lidia, Loretta, Lourdes, Lucina, Luisana, Matilde, Melinda, Milca,
Monica, Monica, Naara, Nadina, Natanael, Natasha, Naudina, Nerina, Nilda,
Nolbin, Nubia, Paloma, Paola, Piedad, Presentacion, Purificacion,
Querubina, Raquel, Rene, Reyes, Reyes, Sabrina, Saritzia, Shael, Sheli,
Sibila, Silka, Simena, Teresa, Tremedal, Ursula, Ventura, Vianney,
Victoria, Yadira, Yahel, Yahel, Zafiro, Zinnia, Zoraida, Zulema]
```

Y se quiere ubicar el primer nombre que comienza con "Fa", se debe mandar además la función de comparación, porque en este caso sólo se deben comparar los dos primeros caracteres de los nombres (no todo el nombre):

```
>>bbinaria(y,"Fa",function(a,b){return a==b.substr(0,2)?0:a>b.substr(0,2)?
1:0;})
30
```

Por lo tanto el primer elemento que contiene un nombre que comienza con "Fa" es aquel con el índice 30.

Otros casos en los que es necesario mandar una función de comparación ocurre cuando los datos son estructurados. En dichos casos se deben comparar únicamente él o los campos correspondientes al valor buscado.

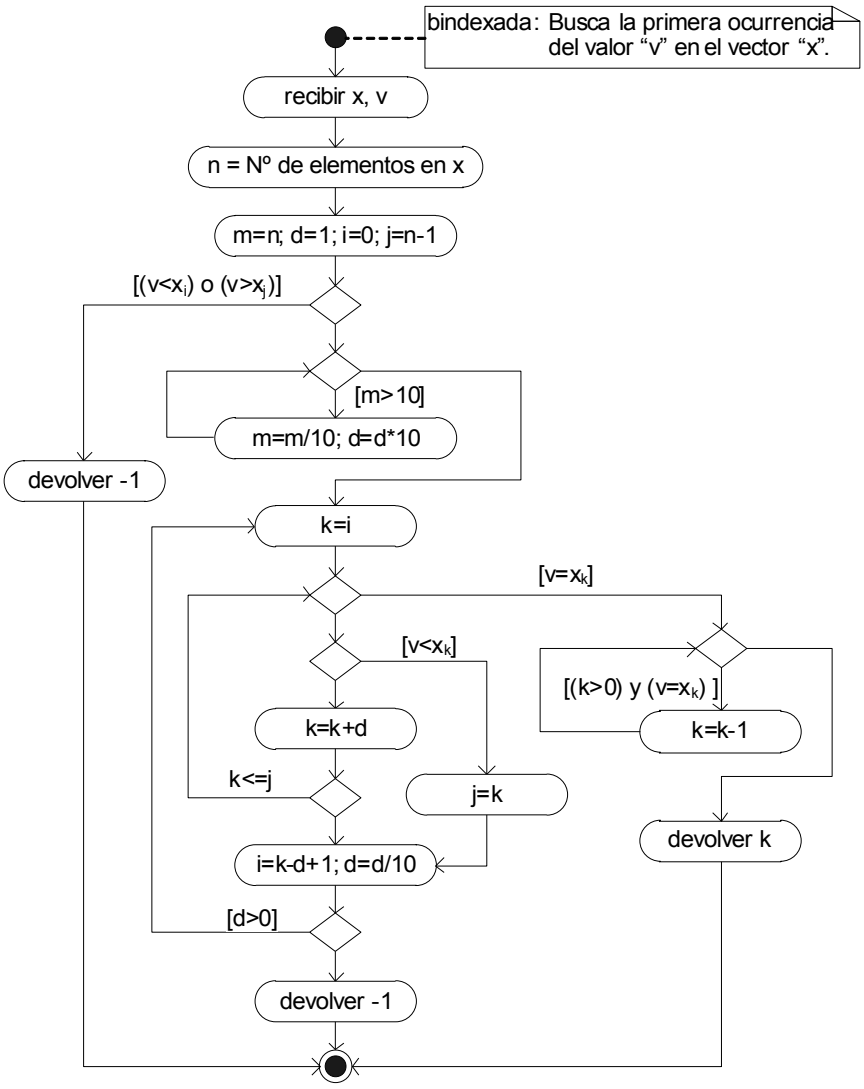
## 8.4. BÚSQUEDA SECUENCIAL INDEXADA

La búsqueda secuencial indexada, al igual que la búsqueda binaria, requiere que el vector donde se realiza la búsqueda esté ordenado. En este método se realiza una búsqueda secuencial, pero no en elementos consecutivos, sino en elementos separados generalmente por algún múltiplo de 10.

La búsqueda comienza empleando como incremento un múltiplo de 10, con la misma dimensión que el número de elementos en el vector, por ejemplo, si el vector tiene 560000 elementos, el incremento inicial es 100000, es decir que el valor buscado se compara con los elementos 0, 100000, 200000, etc. Si en esta búsqueda no se encuentra el valor, se identifica el segmento en el que se encuentra y la búsqueda continua en dicho segmento con un incremento igual a la décima parte del incremento anterior, continuando de esa manera hasta que se encuentra el valor o hasta que el incremento es uno y no quedan más elementos en el segmento, en cuyo caso se sabe que el valor buscado no existe en el vector.

El algoritmo para la búsqueda secuencial indexada se presenta en la siguiente página y el código elaborado en base al mismo, empleando una función de comparación en lugar de los operadores relacionales directos y añadido a la librería "hpvlib.js" es:

```
function bindexada(x,v,fc){
  if (fc==null) fc=function(a,b){return a==b?0:a>b?1:-1;}
  var n=x.length,m=n,d=1,i=0,j=n-1,k;
  if (fc(v,x[i])<0 || fc(v,x[j])>0) return -1;
  while (m>10) {m=quot(m,10); d*=10;}
  do{
    k=i;
    do{
      if (!fc(v,x[k])){
```




```
        while (k>0 && !fc(v,x[k-1])) k--;
        return k;
      }
      if (fc(v,x[k])<0) {j=k; break;}
      k+=d;
    } while (k<=j);
    i=k-d+1; d=quot(d,10);
  } while (d>0);
  return -1;
}
```

Por ejemplo, si se genera y ordena (con el método Quick-Sort) una lista con 230000 números enteros comprendidos entre 1 y 400000:

```
>>randseed=1; x=round(rand(23000,1,40000)); quick(x)
```

Para buscar en la misma por ejemplo el número 14571, se escribe:

```
>>bindexada(x,14571)
8323
```

Por lo tanto el elemento con el índice 8323 es el primer elemento que tiene el número buscado, lo que se puede corroborar, viendo el valor del mismo:

```
>>x[8323]
14571
```

Y verificando además que el elemento anterior, no contiene el número buscado:

```
>>x[8322]
14570
```

Como de costumbre, si la lista contiene datos estructurados o si la búsqueda es parcial, para la realizar la búsqueda (así como para ordenar la lista) se debe mandar además la función de comparación.

Por ejemplo, si con el siguiente constructor, que genera un registro aleatorio con los campos nombre, dirección y teléfono:

```
>>genreg=function(){this.nombre=generarApellido()+" " +generarApellido()+"
"+(round(random(0,1))?generarNombreF():generarNombreM()), this.direccion=
generarCalle()+" "+round(random(1,1000)), this.telefono=6400000+
round(random(1,99999))};
```

Se crea una lista con 100 registros y se la ordena por el campo nombre (con el método Quick-Sort):

```
>>randseed=1; y=[]; for(i=0;i<100;i++) y.push(new genreg()); quick(y,
function(a,b){return a.nombre==b.nombre?0:a.nombre>b.nombre?1:-1;}); y
[{nombre:Abascal Artigas Nieves, direccion:Gregorio Pacheco 146,
telefono:6433777}, {nombre:Alcázar Lobo Dora, direccion:Pasje Estrada 361,
telefono:6409438}, {nombre:Aramburu Lucas Abib, direccion:George Rouma 241,
telefono:6413086}, {nombre:Aznar Coello Deyanira, direccion:Pilinco 948,
telefono:6439524}, {nombre:Baena Iniesta Natasha, direccion:Potosí 946,
telefono:6413130}, {nombre:Balaguer Torrents Niceto, direccion:Miguel Antel
Valda 488, telefono:6403718}, {nombre:Barragán Alemán Rene,
direccion:Regimiento Azurduy 699, telefono:6487688}, {nombre:Barranco Ruano
Samantha, direccion:Warnes 414, telefono:6493413}, {nombre:Batlle
Montserrat Victoria, direccion:La Paz 531, telefono:6489559},
{nombre:Batlle Siripaca Diego, direccion:Villa Serrano 927,
telefono:6436577}, {nombre:Bernat Nebot Vladimiro, direccion:Nataniel
Aguirre 534, telefono:6442246}, {nombre:Bernat Pujol Iris, direccion:Tomás
Katari 716, telefono:6403039}, {nombre:Blanes Novoa Adelgunda,
direccion:Potosí 505, telefono:6444283}, {nombre:Blázquez Almeida Casimiro,
direccion:George Rouma 102, telefono:6428543}, {nombre:Bolaños Barrios
Recaredo, direccion:Tarija 979, telefono:6492216}, {nombre:Bonilla
Huarahuara Saverio, direccion:Guillermo Loayza 598, telefono:6444718},
{nombre:Borrell Carbajo Hina, direccion:J. Prudencio Bustillos 843,
telefono:6406911}, {nombre:Bou Mulet Marite, direccion:Uruguay 979,
telefono:6404826}, {nombre:Busquets Amo Leila, direccion:Isabel Acrelo 609,
telefono:6441985}, {nombre:Cabañas Bello Omar, direccion:Demetrio Canelas
652, telefono:6452073}, {nombre:Camps Llanque Ximena, direccion:Iturricha
143, telefono:6463745}, {nombre:Canquichoque Estévez Oberto,
direccion:Emilio Hochman 431, telefono:6436026}, {nombre:Carani Caparrós
Danae, direccion:Ravelo 572, telefono:6438342}, {nombre:Carani Garay
Aspasia, direccion:Cobija 959, telefono:6428170}, {nombre:Cardona Salcedo
Naim, direccion:Osvaldo Molina 113, telefono:6402535}, {nombre:Casares
Rosales Hermalindo, direccion:Regimiento Azurduy 348, telefono:6444288},
{nombre:Castells Huertas Tecla, direccion:Estudiantes 967,
telefono:6435792}, {nombre:Castelló Barrios Cedric, direccion:Urriolagoitia
826, telefono:6446085}, {nombre:Castrillo Lorenzo Velasco, direccion:Peru
128, telefono:6490414}, {nombre:Chana Carmona Elfida, direccion:J. A.
Martinez 974, telefono:6404413}, {nombre:Charca Mora Ivana,
direccion:Avenida del Maestro 570, telefono:6488671}, {nombre:Chico Moreno
Yajaira, direccion:Sebastian Garcia 377, telefono:6444295}, {nombre:Chiuchi
```

Chaparro Robertino, direccion:Alfredo Vargas 846, telefono:6438366}, {nombre:Chuwi Ródenas Félix, direccion:Adolfo Vilar 866, telefono:6454893}, {nombre:Cobo Gallart Adelmo, direccion:José Carrasco 1000, telefono:6426414}, {nombre:Cobo Palomares Lucrecia, direccion:Able Arduz 624, telefono:6470706}, {nombre:Collado Matas Hildebrando, direccion:Peru 444, telefono:6437383}, {nombre:Collo Iglesia Arabeli, direccion:Uruguay 146, telefono:6432056}, {nombre:Corominas Uría Oda, direccion:Paraguay 227, telefono:6461515}, {nombre:Cáceres Larrea Lisístrato, direccion:Emilio Hochman 649, telefono:6499284}, {nombre:Céspedes Pujol Reyes, direccion:Trinidad 100, telefono:6493526}, {nombre:Duarte Villalba Cesarión, direccion:Bolivar 649, telefono:6481070}, {nombre:Echevarría Castillo Inmaculada, direccion:Rosendo Villa 228, telefono:6450952}, {nombre:Escudero Manrique Milena, direccion:Pje. Setesur 513, telefono:6408373}, {nombre:Espinosa Tarqui Caterina, direccion:Chaco 399, telefono:6402842}, {nombre:Esteban Cornejo Frida, direccion:José Carrasco 89, telefono:6463393}, {nombre:Fernández Gisbert Osmundo, direccion:Buenos Aires 433, telefono:6468349}, {nombre:Ferrández Beltrán Gemma, direccion:Benavidez 942, telefono:6445040}, {nombre:Galiano Recio Nolbin, direccion:J. A. Martinez 142, telefono:6420109}, {nombre:Giner Esteban Isaura, direccion:Tomás Katari 556, telefono:6407795}, {nombre:Goñi Peña Ailen, direccion:Junín 413, telefono:6404302}, {nombre:Guaracusi Córdoba Ayax, direccion:Real Audiencia 354, telefono:6431446}, {nombre:Guerrero Mir Hina, direccion:Olañeta 903, telefono:6403366}, {nombre:Heras Madrid Arnulfo, direccion:Iturricha 210, telefono:6490023}, {nombre:Hierro Viana Tercio, direccion:Lima Pampa 941, telefono:6422736}, {nombre:Huarahuara Barba Tico, direccion:Kilómetro 7 457, telefono:6403828}, {nombre:Isern Tello Klaus, direccion:José Mostajo 245, telefono:6432360}, {nombre:Izaguirre Olivera Ladislao, direccion:Vicente Donoso 879, telefono:6410775}, {nombre:Llanque Puig Lida, direccion:Regimiento Azurduy 714, telefono:6477938}, {nombre:Lucena Sales Monica, direccion:Paraguay 263, telefono:6474000}, {nombre:Manjón Jáuregui Eloy, direccion:Destacamento 317 629, telefono:6497734}, {nombre:Marí Samper Germán, direccion:Cañada Strongest 665, telefono:6489637}, {nombre:Meléndez Vendrell Jemiel, direccion:Capitan Ustarez 935, telefono:6422220}, {nombre:Merino Pinedo Zamira, direccion:La Laguna 363, telefono:6407276}, {nombre:Moles Mollo Minerva, direccion:El Villar 927, telefono:6478712}, {nombre:Molina Checa Pascual, direccion:José Ballivian 638, telefono:6419041}, {nombre:Morcillo Morán Higinio, direccion:Avenida Juana Azurduy de Padilla 256, telefono:6496758}, {nombre:Oliver Rovira Adam, direccion:Rosendo Villa 106, telefono:6450833}, {nombre:Osuna Mariscal Sibila, direccion:José Salgueiro 457, telefono:6436047}, {nombre:Paillo Moya Emerita, direccion:Loa 956, telefono:6430228}, {nombre:Panti Calleja Anacleto, direccion:Estados Unidos 772, telefono:6455649}, {nombre:Pedraza Querol Piedad, direccion:Osvaldo Molina 215, telefono:6441298}, {nombre:Perelló Esparza Nerina, direccion:Arturo Postnaski 885, telefono:6424223}, {nombre:Perera Peña Gotardo, direccion:José Mostajo 27, telefono:6413930}, {nombre:Peñas Alegria Saturnino, direccion:Jose Antonio Arce 523, telefono:6466959}, {nombre:Portillo Franch Violane, direccion:Cañada Strongest 635, telefono:6409139}, {nombre:Pozo Castillo Heráclito, direccion:Fortún 344, telefono:6423072}, {nombre:Reina Flor Sigifredo, direccion:Demetrio Canelas 662, telefono:6482751}, {nombre:Ribas Choque José, direccion:Primero de mayo 795, telefono:6465806}, {nombre:Robledo Melero Wagner, direccion:Brasil 214, telefono:6444211}, {nombre:Rodríguez Cánovas Marcial, direccion:José Mostajo 887, telefono:6429387}, {nombre:Roma Nadal Rómulo, direccion:Real Audiencia 563, telefono:6440958}, {nombre:Rosell Garay Armentario, direccion:Pastor Sainz 843, telefono:6456049}, {nombre:Saldaña Posada Gastón, direccion:Topater 72, telefono:6417546}, {nombre:Sarmiento Pazos Shael, direccion:Emo Reyes 736, telefono:6420213}, {nombre:Seguí Palma Córdula, direccion:Vicente Donoso 91, telefono:6408753}, {nombre:Taboada Quenta Audomaro,

```
direccion:Kilómetro 7 982, telefono:6471841}, {nombre:Tejada Falcón
Leocadio, direccion:Monseñor Miguel de los Santos 968, telefono:6473397},
{nombre:Torrens Saura Flavio, direccion:A. Solares 905, telefono:6460361},
{nombre:Torrents Marcos Udolfo, direccion:Emilio Hochman 72,
telefono:6495240}, {nombre:Torres Donoso Carolina, direccion:Suipacha 933,
telefono:6470931}, {nombre:Torrijos Mita Yanina, direccion:Anastacio
Paravicini 151, telefono:6422553}, {nombre:Tudela Solera Coloma,
direccion:Pasje Estrada 729, telefono:6492433}, {nombre:Vaquero Pascual
Antonio, direccion:Ricardo Andrade 730, telefono:6469262}, {nombre:Vaquero
Reina Audomaro, direccion:Potosí 997, telefono:6449159}, {nombre:Vilalta
Flor Marquelda, direccion:Julio Pinkas 833, telefono:6478497},
{nombre:Villena Barba Alem, direccion:Tarija 255, telefono:6433834},
{nombre:Yucra Pou Candela, direccion:Raúl De Córdoba 289,
telefono:6437270}, {nombre:Zapata Lara Milca, direccion:Ayacucho 716,
telefono:6422327}, {nombre:Zaragoza Pardo Patrick, direccion:Canadá 698,
telefono:6419483}]
```

Para encontrar, por ejemplo, el primer elemento cuyo nombre comience con el apellido "Tejada", no sólo se manda el valor buscado, sino también la función de comparación de manera que se comparen sólo los primeros 6 caracteres del campo "nombre" (pues el valor buscado sólo tiene 6 caracteres):

```
>>bindexada(y,"Tejada",function(a,b){return a==b.nombre.substr(0,6)?
0:a>b.nombre.substr(0,6)?1:-1;})
87
```

Que corresponde al registro:

```
>>y[87]
{nombre:Tejada Falcón Leocadio, direccion:Monseñor Miguel de los Santos
968, telefono:6473397}
```

Una función de comparación más general, aunque un tanto más larga y menos eficiente, se construye con la propiedad length. Por ejemplo, para ubicar el primer nombre que comience con el apellido "Camps" (sin necesidad de contar el número de caracteres del valor buscado) se escribe:

```
>>bindexada(y,"Camps",function(a,b){return a==b.nombre.substr(0,a.length)?
0:a>b.nombre.substr(0,a.length)?1:-1;})
20
```

Cuando se busca un valor que no existe en la lista, por ejemplo, "Pérez", se obtiene -1 (un índice inexistente):

```
>>bindexada(y,"Pérez",function(a,b){return a==b.nombre.substr(0,a.length)?
0:a>b.nombre.substr(0,a.length)?1:-1;})
-1
```

## 8.5. EJERCICIOS

Para presentar los ejemplos y ejercicios del tema, debe crear una carpeta con el nombre "tema8" y guardar en la misma todos los archivos HTML creados (ejemplos y ejercicios).

1. Cree una página HTML con tres "textarea", dos botones y las etiquetas (label) que sean necesarias. En el primer "textarea" se deben mostrar los 3000 elementos de un vector con números aleatorios comprendidos entre 1 y 5000 ordenados ascendentemente con el método de selección. En el segundo se deben mostrar los 2350 elementos de un vector con números enteros aleatorios comprendidos entre 1 y 4800 ordenados ascendentemente con el método de burbuja. En el tercer "textarea" se deben mostrar los elementos intercalados de los vectores de los dos primeros "textarea".

Los elementos deben ser generados al hacer clic en uno de los botones y la intercalación debe tener lugar haciendo clic en el otro.

2. Cree una página HTML con dos "textarea", dos botones, dos "input text" y las etiquetas (label) que sean necesarias. En el primer "textarea" se deben mostrar los 3000 elementos de un vector con registros aleatorios con los campos apellido paterno, apellido materno y nombres (1 o 2). En el segundo se deben mostrar los registros ordenados por los campos apellido paterno + apellido materno + nombres. Los registros deben ser generados al hacer clic en uno de los botones y deben ser ordenados, con una combinación entre los métodos de selección y de intercalación, al hacer clic en el otro. En los "input text" se debe mostrar el tiempo requerido para ordenar los registros por el método combinado y el método de selección por si solo.

3. Cree una página HTML con un "textarea", dos "input text", un "button" y las etiquetas (label) que sean necesarias. En el "textarea" se deben mostrar los elementos de un vector con 10000 números enteros aleatorios comprendidos entre 0 y 20000, en uno de los "input text" se debe introducir el valor a buscar y en el otro se debe mostrar la posición del valor buscado. Se debe hacer clic en el botón tanto para comenzar la búsqueda con el método de búsqueda secuencial, como para continuar la búsqueda y ubicar el siguiente valor en el vector.

4. Cree una página HTML con un "textarea", dos "input text", un "button" y las etiquetas (label) que sean necesarias. En el "textarea" se deben mostrar los elementos de un vector con 1000 nombres masculinos aleatorios, en uno de los "input text" se debe introducir el nombre a buscar, el cual puede ser un nombre completo o parcial y en el otro se debe mostrar la posición del valor buscado. Se debe hacer clic en el botón tanto para comenzar la búsqueda con el método de búsqueda secuencial, como para continuar la búsqueda y ubicar el siguiente valor en el vector.

5. Cree una página HTML con un "textarea", dos "input text", un "button" y las etiquetas (label) que sean necesarias. En el "textarea" se deben mostrar los elementos de un vector con 10000 números enteros aleatorios comprendidos entre 1 y 15000, ordenados ascendentemente con el método Shell. En uno de los "input text" se debe introducir el valor a buscar y en el otro se debe mostrar la posición del valor buscado. Se debe hacer clic en el botón para comenzar la búsqueda con el método de búsqueda binaria.

6. Cree una página HTML con un "textarea", dos "input text", un "button" y las etiquetas (label) que sean necesarias. En el "textarea" se deben mostrar los elementos de un vector con 2000 números nombres masculinos aleatorios, ordenados descendentemente con el método Quick-Sort. En uno de los "input text" se debe introducir el valor a buscar, el cual puede ser un nombre completo o parcial y en el otro se debe mostrar la posición del valor buscado. Se debe hacer clic en el botón para comenzar la búsqueda con el método de búsqueda binaria.

7. Cree una página HTML con un "textarea", dos "input text", un "button" y las etiquetas (label) que sean necesarias. En el "textarea" se deben mostrar los elementos de un vector con 20000 números enteros comprendidos entre 1 y 30000, ordenados descendentemente con el método Shell. En uno de los "input text" se debe introducir el valor a buscar y en el otro se debe mostrar la posición del valor buscado. Se debe hacer clic en el botón para comenzar la búsqueda con el método de búsqueda secuen-

cial indexada.

8. Cree una página HTML con un “textarea”, dos “input text”, un “button” y las etiquetas (label) que sean necesarias. En el “textarea” se deben mostrar los elementos de un vector con 1000 registros aleatorios con los campos nombre (con un apellido y un nombre), teléfono y fecha de nacimiento. Los registros deben estar ordenados descendentemente, con el método Shell, por el campo nombre. En uno de los “input text” se debe introducir el nombre a buscar, el cual puede ser completo o parcial y en el otro se debe mostrar la posición del valor buscado. Se debe hacer clic en el botón para comenzar la búsqueda con el método de búsqueda secuencial indexada.