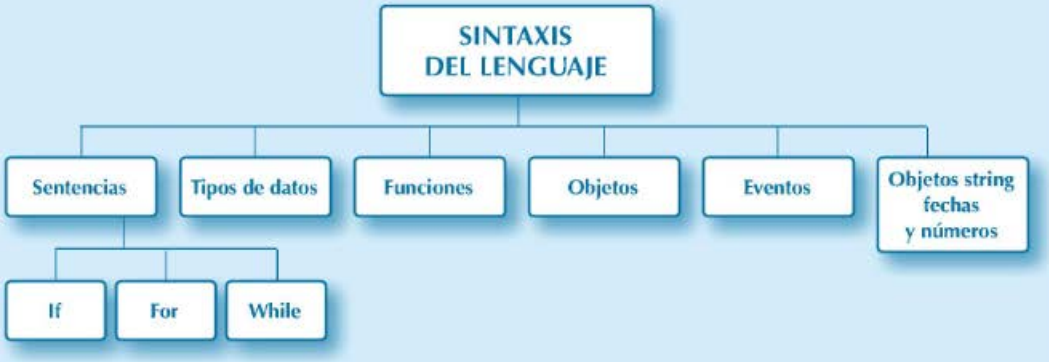
2

Sintaxis del lenguaje JavaScript

Objetivos

- ✓ Comprender la sintaxis básica de JavaScript para poder realizar pequeños scripts funcionales.
- ✓ Entender las reglas básicas del lenguaje JavaScript.
- ✓ Conocer la utilización de las sentencias básicas de JavaScript.
- ✓ Descubrir elementos propios de JavaScript como los eventos.

Mapa conceptual



Glosario

Case-sensitive. Expresión informática que se aplica a los textos en los que tiene relevancia escribir un carácter en mayúsculas o minúsculas y que significa "sensible a mayúsculas y minúsculas".

Consola de JavaScript. Herramienta del programador para comunicar su programa JavaScript con el exterior. Se invoca pulsando generalmente la tecla F12.

ES6 o ES2015 (ECMAScript). Especificación de lenguaje JavaScript publicada por ECMA International. Los navegadores utilizan una implementación de ECMAScript y acceso al DOM para manipular las páginas web.

GMT (Greenwich Mean Time). Estándar de tiempo que se refería al tiempo solar medio en el Real Observatorio de Greenwich y que dejó de ser utilizado por la comunidad científica en 1972, cuando se reemplazó por el UTC.

Listener. Código responsable de controlar los eventos. Están a la escucha (de ahí su nombre) y, cuando ocurre un evento, se ejecuta el código que el programador haya implementado.

Operador ternario. Operador que toma tres argumentos. La ventaja que ofrece es que puede reducir varias líneas de código en una sola.

UTC (Coordinated Universal Time). Principal estándar de tiempo, que casi siempre es sinónimo de GMT, y se obtiene a través del tiempo atómico internacional.

2.1. Sintaxis del lenguaje. Operadores y palabras reservadas

La sintaxis de JavaScript es muy parecida a Java y C++. Cualquier programador que sepa programar en Java, PHP u otro lenguaje con sintaxis similar será capaz de comprender la sintaxis de JavaScript de una manera rápida. No obstante, al ser JavaScript un lenguaje de programación del lado del cliente, es importante conocer sus aspectos básicos como pueden ser los eventos, el control de los elementos HTML, etc.

RECUERDA

✓ Cualquier programa JavaScript puede comunicarse con la consola del navegador mediante la sentencia:

```
console.log("información para la consola");
```

En la consola, se pueden escribir literales y también valores de variables. Todos los navegadores tienen consola y es muy utilizada por los programadores con fines de depuración.

2.1.1. Las 10 reglas básicas de la sintaxis del lenguaje

A continuación, se presentan las diez reglas básicas del lenguaje JavaScript:

- *Regla 1.* Las instrucciones en JavaScript terminan en un punto y coma. Ejemplo:

```
var s = "hola";
```

- *Regla 2.* Uso de decimales en JavaScript. Los números en JavaScript que tengan decimales utilizarán el punto como separador de las unidades con la parte decimal. Ejemplos de números:

```
var x = 4;
var pi = 3.14;
```

- *Regla 3.* Los literales se pueden escribir entre comillas dobles o simples. Ejemplo:

```
var s1 = "hola";
var s2 = 'hola';
```

- *Regla 4.* Cuando sea necesario declarar una variable, se utilizará la palabra reservada *var*.
- *Regla 5.* El operador de asignación, al igual que en la mayoría de lenguajes, es el símbolo igual (=).
- *Regla 6.* Se pueden utilizar los siguientes operadores aritméticos: (+ − * /). Ejemplo:

```
var x = (5*4)/2+1;
```

- *Regla 7.* En las expresiones, también se pueden utilizar variables. Ejemplo:

```
var t = 4;
var x = (5*t)/2+1;
var y;
y = x * 2;
```

- *Regla 8.* Comentarios en JavaScript. Existen dos opciones para comentar el código:

  a) // cuando se desea comentar el resto de la línea a partir de estas dos barras invertidas.
  b) /* y */. todo lo contenido entre ambas etiquetas quedará comentado.

- *Regla 9.* Los identificadores en JavaScript comienzan por una letra o la barra baja (_) o el símbolo del dólar ($).
- *Regla 10.* JavaScript es sensible a las mayúsculas y minúsculas (case-sensitive). Ejemplo:

```
var nombre = "Julio";
var Nombre = "Ramón";
```

TOMA NOTA

- *Nombre* y *nombre* son dos variables diferentes.
- Ten cuidado al escribir la palabra reservada *var*. Si escribes *Var* o *VAR*, tu código no funcionará.

2.1.2. Las palabras reservadas de JavaScript

Algunas de estas palabras reservadas se han visto ya y otras se trabajarán a lo largo del libro. En el cuadro 2.1, se muestran las más utilizadas.

CUADRO 2.1
Palabras reservadas más comunes de JavaScript

Palabra	Descripción
`var`	Utilizada para declarar una variable.
`if … else`	Estructura condicional.
`for`	Estructura de repetición. Se ejecutará mientras la condición sea verdadera.
`do … while`	Estructura de repetición. Se ejecutará mientras la condición sea verdadera.
`switch`	Serie de sentencias que van a ser ejecutadas dependiendo de diferentes circunstancias.

[.../...]

CUADRO 2.1 (CONT.)

`break`	Termina un switch o un bucle.
`continue`	Sale del bucle y se coloca al comienzo de este.
`function`	Declara una función.
`return`	Sale de una función.
`try … catch`	Utilizadas para el manejo de excepciones.

2.1.3. Uso de variables en JavaScript

Las variables son la base de la programación. Una variable es una posición de memoria donde se pueden alojar datos. El nombre de la variable permitirá a JavaScript poder ubicar y localizar dicho espacio cuando interprete los scripts. En este apartado, se mostrará el uso de las variables en JavaScript.

FUNDAMENTAL

Constantes en JavaScript

En JavaScript, a partir de ES6 (ES2015), se puede utilizar la palabra reservada *const*. Por lo tanto, en vez de utilizar:

```
var pi = 3.141592;
```

se aconseja emplear:

```
const PI = 3.141592;
```

dado que pi es una constante (valor invariable). Por convención, se suele utilizar mayúsculas cuando se definen constantes.

En JavaScript, se pueden ejecutar las siguientes órdenes:

```
var x;
x = 2 * x + 1;
var pi = 3.141592;
var paginaweb = "Myfpschool";
var pregunta = '¿cuantos años tienes?', respuesta="veinte";
paginaweb="Myfpschool.com";
paginaweb="Myfpschool" + "." + "com";
```

RECUERDA

✓ El resultado a la derecha de una asignación se almacena en la variable del lado izquierdo de esta.

2.1.4. Operadores en JavaScript

A continuación, se muestran distintos tipos de operadores utilizados en JavaScript.

A) Operadores aritméticos

Todos los lenguajes de programación tienen operadores y, generalmente, suelen coincidir (salvo los operadores incremento y decremento que aparecieron con C++, siendo uno de sus rasgos más significativos).

CUADRO 2.2
Operadores aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
++	Incremento
--	Decremento

B) Operadores de asignación

Otros operadores básicos son los operadores de asignación. En el cuadro 2.3, se muestran los operadores de asignación de JavaScript.

CUADRO 2.3
Operadores de asignación

Operador	Ejemplo de uso
=	x = y;
+=	x += y; // igual que (x = x + y)
-=	x -= y; // igual que (x = x – y)
*=	x *= y; // igual que (x = x * y)
/=	x /= y; // igual que (x = x / y)
%=	x %= y; // igual que (x = x % y)

C) Operadores de manejo de strings

En JavaScript, se utilizan los operadores + y += para concatenar strings. Véase un ejemplo de uso:

```
var = "hola" + " " + "mundo";
```

O lo que sería igual:

```
var = "hola";
var += " ";
var += "mundo";
```

D) Operadores lógicos y de comparación

Tanto los operadores lógicos como los de comparación son profusamente utilizados por los programadores. En el cuadro 2.4, se muestran los operadores tanto lógicos como de comparación para los programadores JavaScript.

CUADRO 2.4
Operadores lógicos y de comparación

Operador	Descripción
==	Igual que
===	Igual valor y tipo
!=	Distinto
!==	Distinto valor o distinto tipo
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
?	Operador ternario

E) Operadores de tipo

Los operadores de tipo permiten conocer si un objeto es una instancia de un tipo concreto o bien conocer el tipo de una variable. A continuación, se muestran los operadores de tipo disponibles en JavaScript:

- *typeof.* Devuelve el tipo de una variable.
- *instanceof.* Devuelve true si un objeto es una instancia de un tipo de objeto.

2.2. Tipos de datos. Asignaciones y expresiones

JavaScript es un lenguaje bastante simplificado en lo que a tipos de datos se refiere y, por ello, solamente tiene cinco tipos de datos:

1. String.
2. Number.
3. Boolean.
4. Array.
5. Object.

Véase un ejemplo de utilización de cada uno de estos tipos:

```
var edad = 25; // Number
var nombre = "Dimas"; // String
var asignatura = ["lengua", "mate", "cono"]; // Array
var persona = {nombre:"Dimas", apellido:"Moreno"}; // Objeto
```

"Dimas" o "lengua" son literales y, por eso, aparecen entrecomillados.
Para JavaScript, las siguientes dos líneas de código son iguales:

```
var dato = "Ronaldo " + 10;
var dato = "Ronaldo " + "10";
```

JavaScript interpretará el número como un string.

2.2.1. El valor null

Null para los lenguajes de programación es *nada* o algo que no existe. Generalmente, cuando se asigna null a una variable, es porque es o será un objeto.

Véase un ejemplo de uso del null:

```
var persona = null;
persona = {nombre:"Dimas", apellido:"Moreno"};
```

2.2.2. Conversiones entre tipos de datos en JavaScript

JavaScript es un lenguaje interpretado, por lo tanto, la conversión entre un tipo de dato y otro es transparente al programador. JavaScript asignará el tipo de datos más acertado a la variable que el programador esté utilizando.

No obstante, existen funciones de conversión como String() y Number() para convertir a cadenas de caracteres y números, respectivamente.

También existen las funciones:

- *parseInt()*. Que convierte una cadena a un número entero.
- *parseFloat()*. Que convierte una cadena a un número decimal.

Véase un ejemplo de uso de estas funciones:

```
alert("Entero: " + parseInt("9.99"));  //mostrará Entero: 9
alert("Float: " + parseFloat("9.99")); //mostrará Float: 9.99
```

Es posible también convertir variables de tipo fecha a string con la función toDateString(). A continuación, se muestra un ejemplo de dicha conversión:

```
var fecha = new Date();
var cadena = fecha.toDateString(); // Ahora cadena contendrá la fecha
actual "Sun, 13 Jan 2019".
```

Si lo que se desea es convertir la hora en formato UTC, se puede utilizar alternativamente la función toUTCString(). Se muestra un ejemplo de dicha conversión:

```
var fecha = new Date();
var cadena = fecha.toUTCString(); // Ahora cadena contendrá la fecha ac-
tual en formato UTC "Sun, 13 Jan 2019 07:45:49 GMT".
```

2.3. Introducción a las funciones

Las funciones son uno de los elementos de la base de la programación estructurada. Reutilizar el código es algo básico en cualquier lenguaje de programación porque la regla que se debe seguir es "escribe una vez y utilízala tantas veces como puedas". La variación del resultado de las funciones dependerá de sus argumentos.

Un ejemplo de estructura básica de una función es el siguiente:

```
function suma(a, b) {
    return a + b;
}
```

Como se puede ver, se tiene la palabra reservada *function* para que JavaScript interprete que se encuentra frente a una función seguida de su nombre (en el ejemplo anterior, *suma*). Esta función aceptará dos parámetros *a* y *b* y devolverá (return) la suma de ambos.

TOMA NOTA

- Una función puede tener cero, uno o varios parámetros.
- El código de la función estará dentro de las llaves { y }.
- La función devolverá el resultado con la sentencia return.

Véase un ejemplo de utilización de la función anterior:

```
var c = suma(3,3); // c valdrá 6
c = suma(c,3); // ahora c valdrá 9
var text = "El valor de c será ahora: "+c;
```

Actividad propuesta 2.1

Vista la función suma, realiza la función resta, multiplicación y división.

2.4. Introducción a los objetos en JavaScript

Desde hace mucho tiempo, la programación estructurada era el único paradigma efectivo y eficiente en programación, pero las cosas cambiaron y nació la programación orientada a objetos, que rompía con lo establecido. No era una evolución, era una nueva filosofía que no tenía nada que ver con lo anterior.

Hoy en día, es prácticamente imposible programar sin utilizar programación orientada a objetos. Esta forma de programar es más cercana a cómo se expresarían las cosas en la vida real que en otros tipos de programación clásicos.

Analistas y programadores piensan y describen las realidades y el entorno de una manera distinta para plasmar dichos conceptos en programas en términos de objetos, atributos y métodos.

Véase cómo se definiría un objeto en JavaScript:

```
var perro = {
    raza: "Podenco",
    peso: 12,
    altura: 58,
    color: "negro"
};
```

Como se puede observar, un objeto, al igual que en la vida real, puede tener muchas características o atributos, y cada atributo se verá reflejado en una variable dentro del objeto. Un objeto se diferenciará de otros por el valor de sus atributos.

También se pueden encontrar definiciones de objetos en una sola línea (aunque la forma anterior parece más legible):

```
var perro = {raza:"Podenco", peso:12, altura:58,color:"negro"};
```

Para acceder a los atributos de un objeto, se puede hacer de la siguiente manera:

```
perro.raza;
```

O bien:

```
perro[ "raza" ];
```

Actividad propuesta 2.2

Crea una página web con un script que contenga un objeto de la clase persona, la cual tendrá los siguientes atributos:

- Nombre: Dimas.
- Edad: 28.
- Altura: 185.

Una vez creado el objeto, se deberá mostrar por consola: "El usuario Dimas tiene 28 años y mide 185 centímetros".

Todo ello utilizando los atributos del objeto.

2.5. Variables y ámbitos de utilización

A continuación, se estudiará la diferencia que existe entre variables globales y locales en JavaScript. Véase un ejemplo:

```
var a; // Esta es una variable global
a=10;

function suma() {
    var b = 5;
    return a + b; // la función devolverá 15
}
```

La diferencia entre las variables *a* y *b* es que *a* puede utilizarse en otras funciones que se puedan crear, mientras que *b* solamente existirá dentro de la función suma.

¿Qué es lo que ocurre cuando no se declara una variable como la variable *d* en el siguiente código?

```
var a; // Esta es una variable global
a=10;
function suma() {
    var b = 5;
    c = 20;
    return a + b; // la función devolverá 15
}
```

Que será global a todo el JavaScript de esa página.

RECUERDA

✓ Las variables, y obviamente sus valores, desaparecen cuando se cierra la página web.

2.5.1. Las diferencias entre var y let

JavaScript permite declarar las variables con var y let. La diferencia entre ambas palabras reservadas es que, cuando se declara una variable con var, la variable puede utilizarse fuera del bloque donde fue declarada. Véase un ejemplo de esto:

```
var a = 10; // x vale 10
{
   let a = 5; // a vale 5
}
// aquí a vale 10
{
   let b=3;
}
// aquí no se puede utilizar b
{
   var c=7;
}
// aquí sí se puede utilizar c y valdrá 7.
```

> **FUNDAMENTAL**
>
> *Las palabras reservadas* let *y* const
>
> El estándar ES2015 añadió las palabras clave *let* y *const*. La primera se utiliza para declarar variables en un bloque determinado y la segunda, para definir constantes. Anteriormente, en JavaScript, las variables estaban definidas a nivel global o a nivel de función.

2.6. Eventos

Los eventos son cualquier suceso que le pueda ocurrir a un elemento HTML. Como se puede suponer, JavaScript puede darse cuenta de ese evento y reaccionar a este ejecutando el código que se haya programado.

Algunos eventos que pueden ocurrir en una página web son, entre otros:

- Pulsar un botón.
- Modificar un campo de texto.
- Pulsar una tecla.
- La página ha terminado de cargarse.
- Hacer clic sobre un elemento HTML.

El formato de programar un evento en JavaScript sería el siguiente:

```
<Elemento_HTML evento='código_JavaScript'>
```

También es posible cambiar las comillas simples por comillas dobles. Se pueden utilizar de forma indistinta.

Véase un ejemplo de evento en JavaScript programado para un botón:

```
<button onclick='this.innerHTML=Date()'>Pulsa para saber la hora</button>
```

2.6.1. Tipos de eventos en JavaScript

En un navegador, hay muchos tipos de eventos que pueden ser controlados por JavaScript. En los siguientes cuadros, se muestra una clasificación de los distintos tipos de eventos clasificados por campos de texto, ratón y carga:

CUADRO 2.5
Eventos en campos de texto

Evento	Ejecutado
onblur	Cuando se abandona un campo de entrada.
onchange	Cuando se cambia el contenido de un campo de entrada o cuando un usuario selecciona un valor de una lista desplegable.
onfocus	Cuando obtiene el foco un campo de entrada.
onselect	Cuando se selecciona el texto de entrada.
onsubmit	Cuando se hace clic en el botón de enviar.
onreset	Cuando se hace clic en el botón de reinicio.
onkeydown onkeypress	Cuando se presiona o mantiene presionada una tecla.
onkeyup	Cuando se deja de presionar una tecla.

CUADRO 2.6
Eventos relativos al ratón

Evento	Ejecutado
onclick	Cuando se hace clic en un botón.
ondblclick	Cuando se hace doble clic en un texto.
onmouseover onmouseout	Cuando el ratón pasa sobre un elemento.
onmousedown onmouseup	Al presionar o soltar un botón del ratón.
onmousemove	Al mover el puntero del ratón sobre de una imagen.
onmouseout	Al mover el puntero del ratón fuera de una imagen.
onmouseover	Al mover el ratón sobre una imagen.
onmouseout	Al mover el ratón fuera de una imagen.

CUADRO 2.7
Eventos de carga

Evento	Ejecutado
onload	Cuando se carga una imagen o una página.
onerror	Cuando se produce un error al cargar una imagen.
onunload	Cuando el navegador cierra el documento (página web).
onresize	Cuando se cambia el tamaño de la ventana del navegador.

2.6.2. Los listener

Otra forma de añadir los eventos a los elementos u objetos HTML del DOM (listeners):

```
<!DOCTYPE html>
<html>
<body>

<p id="myFPtext">texto a modificar</p>
<button id="myFPboton">Dale fuerte!</button>

<script>

function dale(){
var x = document.getElementById("myFPtext");
x.innerHTML = "hola";
}

document.getElementById("myFPboton").addEventListener("click", dale,
false);

</script>

</body>
</html>
```

Como se puede observar, en el código anterior se utiliza el método addEventListener para vincular un evento a una función JavaScript. Esta forma de programar los listeners es más limpia y clara.

Los parámetros de la función `addEventListener()` son los siguientes:

1. El evento (sin el "on").
2. La función por ejecutar.
3. useCapture. Se utilizará si el usuario quiere iniciar la captura para que todos los eventos de ese tipo se lancen en el mismo listener. Generalmente, se utiliza false en este parámetro.

2.7. Objeto string o cadena de caracteres

En anteriores apartados, se han visto ejemplos de funcionamiento de las cadenas de caracteres o strings:

```
var web = "myfpschool.com";
var web = 'myfpschool.com';
```

Como ya se ha explicado, se podían utilizar comillas dobles y simples para definir valores a los string.

En el caso de que se quieran utilizar las comillas dentro de un string, se necesitará utilizar secuencias de escape. Un ejemplo de introducción de una barra invertida dentro de una cadena sería el siguiente:

```
var web = "la mejor \"web\" de tecnología";
```

El contenido del string web anterior sería: la mejor "web" de tecnología.

TOMA NOTA

Es posible incluir tabulación (\t) y saltos de línea (\n) dentro de las cadenas de caracteres. Ejemplo:

```
alert("hola \tme llamo \nCarlos");
```

2.8. Números

A continuación, se explicarán particularidades sobre los números en JavaScript.

2.8.1. Convertir un número en un string

En JavaScript, al igual que en otros lenguajes, existe el método toString(), el cual convierte un número en una cadena de caracteres.

Véase un ejemplo de su uso:

```
var num = 999;
num.toString(); // devolverá 999 pero como cadena de caracteres
(888).toString(); // devuelve 888 que lo toma del literal 888
(888 + 111).toString(); // realiza la operación y devuelve 999 pero como
cadena de caracteres
```

2.8.2. Ajuste de decimales. Número de decimales de un número

En JavaScript, el método tofixed() servirá para ajustar el número de decimales de un número.

```
var num = 9.99;
num.toFixed(0); // devolverá 10. 0 lugares decimales. Realiza redondeo.
num.toFixed(1); // devolverá 10.0 . 1 lugar decimal. Realiza redondeo.
num.toFixed(2); // devolverá 9.99. 0 lugares decimales.
num.toFixed(3); // devolverá 9.990. 0 lugares decimales.
```

2.8.3. Precisión de un número

En JavaScript, el método toPrecision() servirá para ajustar la precisión de un número:

```
var num = 9.99;
num.toPrecision(); // devolverá 9.99.
num.toPrecision(1); // devolverá 1e+1 .
num.toPrecision(2); // devolverá 10.
num.toPrecision(3); // devolverá 9.99.
```

2.8.4. Convertir cualquier variable en un número

La función Number() permite convertir cualquier variable en un número.Véase unos ejemplos de uso:

```
Number(true); // devuelve 1
var num = false;
Number(num); // devuelve 0
```

2.9. Fechas

Las fechas en JavaScript se pueden visualizar como un número o un string.Véase un ejemplo de una fecha visualizada como cadena de caracteres:

```
<p id="fecha"></p>
<script>
document.getElementById("fecha").innerHTML = Date();
</script>
```

El resultado de visualizar la fecha actual con el script anterior sería algo similar a:

Mon Jun 27 2019 10:45:12 GMT+0200 (CEST)

2.9.1. Creación de un objeto de tipo fecha

En JavaScript, existen cuatro formas diferentes (cuatro constructores diferentes) de inicializar un objeto de tipo fecha:

```
new Date()
new Date(milisegundos)
new Date(fechaString)
new Date(año, mes, día, horas, minutos, segundos, milisegundos)
```

Véanse algunas formas de utilizar estos constructores:

```
var v = new Date(99,8,16,10,25,12,0);
var v = new Date(92500000);
var v = new Date();
```

SABÍAS QUE...

Para transformar una fecha en string, debe utilizarse el método toString().

2.9.2. Formatos de tipo fecha

En JavaScript, existen distintos formatos de tipo fecha:

- Formato corto.
- Formato largo.
- Formato completo.
- Formato ISO.

Véase un ejemplo de cada uno de estos tipos:

```
var v = new Date("08/16/2016"); // Formato corto
var v = new Date("Ago 16 2016"); // Formato largo
var v = new Date("Fri Ago 16 2019 12:15:24 GMT+0100 (W. Europe Standard
Time)"); // Formato completo
var v = new Date("2019-08-16"); // Formato ISO
```

2.9.3. Métodos del tipo fecha

De nada sirve un objeto de tipo fecha sin sus correspondientes métodos que lo hagan potente y versátil. En JavaScript, los objetos tipo fecha tienen los siguientes métodos que pueden manejarse de una manera eficiente en cualquier script:

- *getDate()*. Obtiene el día del mes (1-31).

- *getDay()*. Obtiene el día de la semana en formato numérico (0-6).
- *getFullYear()*. Obtiene el año (2016, por ejemplo).
- *getHours()*.
- *getMilliseconds()*.
- *getMinutes()*.
- *getMonth()*.
- *getSeconds()*.
- *getTime()*. Obtiene la hora en milisegundos desde el 1 de enero de 1970.

2.10. Arrays

Los arrays o vectores son una serie de elementos u objetos contiguos, a los cuales pueden accederse utilizando un subíndice. Prácticamente todos los lenguajes de programación utilizan arrays por su utilidad.

A continuación, se muestra un ejemplo de utilización de arrays dentro de una página web:

```
<p id=»aqui»></p>
<script>
var dias = [«lunes», «martes», «miércoles»,»jueves», «viernes», «sábado»,
«domingo»];
document.getElementById(«aqui»).innerHTML = dias;
</script>
```

También se podría haber creado el array de la siguiente forma:

```
var dias = new Array («lunes», «martes», «miércoles»,»jueves», «viernes»,
«sábado», «domingo»);
```

> **FUNDAMENTAL**
>
> *¿Se pueden tener objetos de diferentes tipos en un array?*
>
> Para comprobar que funciona, basta con intentar hacer lo siguiente en el ejemplo anterior:
>
> ```
> dias[1]=Date();
> ```

Actividad propuesta 2.3

En caso de que funcione el ejemplo de este apartado, cambia el formato de la fecha para que aparezca como el siguiente: dd/mm/yyyy.

2.10.1. Operaciones con arrays

Existen múltiples operaciones en JavaScript para manipular arrays. A continuación, se mostrarán las distintas posibilidades del lenguaje:

- Conocer su longitud:

```
longitud = dias.length;
```

- Ordenarlo en alfabéticamente:

```
dias = dias.sort();
```

- Añadir un nuevo elemento:

```
dias = dias.push("juernes");
```

- Extraer el último elemento:

```
ultimo = dias.pop();
```

El último elemento se eliminará del array. *Extraer* significa "retirar".

- Eliminar el primer elemento:

```
dias.shift();
```

- Eliminar un elemento concreto:

```
delete dias[2];
```

2.10.2. Arrays multidimensionales

Los arrays multidimensionales consisten en tener en una posición de un array otro array de elementos. Se pueden tener arrays bidimensionales, tridimensionales, etc. Lo más común en programación es tener tablas, también llamados *arrays bidimensionales.*

A continuación, se muestra un ejemplo de creación y acceso a un array bidimensional. Como se puede ver, se han separado los días laborables del fin de semana en los dos elementos del array principal:

```
var dias = [["lunes", "martes", "miércoles","jueves", "viernes"], ["sá-
bado", "domingo"]];
console.log(dias[0][2]);//mostrará por consola miércoles
console.log(dias[1][1]);//mostrará por consola domingo
console.log(dias[1]);//mostrará por consola sábado,domingo
```

RECUERDA

✓ El acceso y recorrido de un array con varias dimensiones es igual a un array con solo una. Únicamente hay que tener en cuenta que hay que detallar la posición en todas y cada una de las dimensiones para acceder al elemento o elementos pretendidos.

2.11. Sentencias condicionales

JavaScript tiene diferentes sentencias condicionales. A continuación, se citarán cada una de ellas:

- *If.* Se ejecutará el bloque de código siguiente si la condición es verdadera.
- *Else.* Se ejecutará el bloque de código siguiente si la condición es falsa.
- *Else if.* Es combinación de las anteriores y se utilizará cuando la primera condición sea falsa.
- *Switch*. Se utilizará para especificar una serie de códigos que se ejecutarán de forma alternativa.

Todo esto se puede apreciar mejor con ejemplos:

- La sentencia IF:

```
if (new Date().getHours() < 18) {
    saludo = "Buenos días";
}
```

El código anterior hace que la variable saludo valga "Buenos días" si es antes de las 18:00.

- La sentencia ELSE:

```
if (new Date().getHours() < 18) {
    saludo = "Buenos días";
}else{
    saludo = "Buenas tardes";
}
```

El código anterior hace que la variable saludo valga "Buenos días" si es antes de las 18:00 y "Buenas tardes" en caso contrario.

- La sentencia ELSE IF:

```
if (new Date().getHours() < 10) {
    saludo = "Al que madruga Dios le ayuda";
}else if (new Date().getHours() < 18) {
    saludo = "Buenos días";
}else {
    saludo = "Buenas tardes";
}
```

El código anterior hace que la variable saludo valga "Al que madruga Dios le ayuda" si la hora es antes de las 10 de la mañana, "Buenos días" si es antes de las 18:00 y "Buenas tardes" en caso contrario.

- La sentencia SWITCH:

```
var = new Date().getDay();
switch (var) {
    case 0:
    dia = "Domingo";
    break;
    case 1:
    dia = "Lunes";
    break;
    case 2:
    dia = "Martes";
    break;
    case 3:
    dia = "Miércoles";
    break;
    case 4:
    dia = "Jueves";
    break;
    case 5:
    dia = "Viernes";
    break;
    case 6:
    dia = "Sábado";
    default:
    dia = " - ";
}
```

Como se puede ver, dependiendo del valor de la variable var, la variable dia tendrá un valor u otro en función del día de la semana en el que se esté.

TOMA NOTA

- Se utiliza la sentencia break para salir de cada bloque del switch.
- La sentencia default se ejecutará si ninguna de las opciones anteriores ha sido ejecutada con las opciones case.

2.12. Bucles

En cualquier lenguaje de programación, se necesita ejecutar un trozo de código repetidas veces. Para ello, se usan los bucles y los hay de distintos tipos, los comunes a todos los lenguajes son el bucle for y while. A continuación, se hará un repaso de cada una de ellos:

a) *Bucle for.* Se utiliza esta sentencia cuando se quiere ejecutar un bloque de código una serie de veces determinada (generalmente, se conoce de antemano el número de veces).
b) *Bucle while.* Se emplea esta sentencia cuando se desea ejecutar un bloque de código una serie de veces sin conocer de antemano cuántas. El bloque de código se ejecutará mientras se cumpla una condición determinada.

A continuación, se muestran algunos ejemplos de uso:

- La sentencia o bucle FOR:

```
for (i = 0; i <= 10; i++) {
    text += "Número: " + i + " ";
}
```

En el bucle for, se inicializa la variable *i* a 0 (i=0). Dicho bucle se ejecutará mientras que la variable *i* sea menor o igual que 10 (i<=10) y la variable *i* a su vez se irá incrementando en una unidad (i++) en cada pasada del bucle.

Existe una variante del bucle for que es el bucle for in. A continuación, se muestra un ejemplo sencillo de comprender:

```
var persona = {nombre:"Dimas", apellidos:"Moreno", edad:25};
var texto = "";
var x;
for (x in persona) {
    texto += persona[x];
}
```

PARA SABER MÁS

Es posible crear un bucle decreciente estableciendo los parámetros de la cabecera con ese propósito. A continuación, se muestra una versión decreciente del bucle mostrado anteriormente:

```
for (i = 10; i >= 0; i--) {
    text += "Número: " + i + " ";
}
```

Como se puede ver en el ejemplo anterior, el valor de *i* configurado a 10, en un principio, irá decreciendo y el bucle se ejecutará siempre y cuando el valor de la variable *i* sea mayor o igual que 0.

- La sentencia o bucle WHILE:

```
i = 0;
while (i <= 10) {
    text += "Número " + i;
    i++;
}
```

Como puede observarse, es el anterior ejemplo del bucle for, pero ahora reconvertido a while.

2.12.1. Recorrido de un array

Existen dos formas de recorrer un array:

1. *Mediante el recorrido clásico.* Se utilizará un bucle para recorrer el array. Véase un ejemplo de uso:

```
for (i=0;i<dias.length;i++){
    console.log(dias[i]);
}
```

2. *Mediante el método forEach().* Véase un ejemplo de uso:

```
var dias = ['lunes','martes','miércoles','jueves','viernes','sábado',
'domingo'];
dias.forEach(muestra);
```

Donde *muestra* es una función que será invocada por cada uno de los elementos del array dias.

A continuación, se muestra un ejemplo en el que se recorre el array de las dos formas citadas anteriormente:

```
<!DOCTYPE html>
<html>
<body>
<h3>Recorrido de un array</h3>
<p id=»aqui»></p>
<script>
var txt = «»;
var txt2 = «»;
var dias = ['lunes','martes','miércoles','jueves','viernes','sábado',
'domingo'];
dias.forEach(muestra);
document.getElementById(«aqui»).innerHTML = txt;

function muestra(valor, index, array) {
    txt = txt + valor + «  «;
}
```

```
document.getElementById(«aqui»).innerHTML += '<br>---------<br>';
for (i=0;i<dias.length;i++){
    txt2 = txt2 + dias[i] + «  «;
}
document.getElementById(«aqui»).innerHTML += txt2;

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

El aspecto en el navegador del ejemplo anterior será el que se muestra observa en la figura 2.1.

Recorrido de un array

lunes martes miércoles jueves viernes sábado domingo
----
lunes martes miércoles jueves viernes sábado domingo

Figura 2.1
Aspecto en el navegador del recorrido de un array.

Actividad propuesta 2.4

Verifica que funciona el código empleado en este apartado en un navegador y muestra el resultado anteriormente expuesto. Al mismo tiempo, modifica el código y añade variantes de tu *cosecha* para intentar conocer mejor cómo funcionan los arrays y los bucles.

2.13. Práctica guiada

A continuación, utilizando bucles, se realiza un script que muestre en una página web lo que se observa en la figura 2.2.

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

Figura 2.2
Rombo.

Se trata de crear un rombo como el de la figura 2.2 respetando los espacios y la forma en lo posible. Recuerda que, para el espacio, hay que utilizar el símbolo HTML   y, para el salto de línea, la etiqueta
.

Se va a ir explicando la solución paso a paso. Ten en cuenta que se incluye el JavaScript dentro del HTML para una mejor comprensión del ejercicio, aunque, en un proyecto real, habría que tenerlo separado.

Para solucionar el ejercicio, se ha dividido el rombo en dos partes: una parte superior en la que va creciendo el número de asteriscos y otra parte inferior en la que el número de asteriscos es decreciente.

En las variables asterisco y espacioHTML, se van a almacenar respectivamente los asteriscos y los espacios HTML que han de mostrarse respectivamente. Si no se incluyen espacios , el rombo no tendrá la forma que se desea conseguir.

En una variable de nombre *límite,* se registra el número de filas (4) que va a tener la parte creciente (la decreciente tendrá una menos).

El código del ejercicio será el siguiente:

```
<!DOCTYPE html>

<html>

<head>
    <title>Ejercicio Rombo</title>
</head>

<body>

<div id="caja"></div>

<script>

    var caja = document.getElementById("caja");

    var limite = 4;

    //Creciente
    for (var i = 0; i < limite; i++) {

        var asterisco = "";
        var espacioHTML = "";

        for (var k = 0; k < limite - i - 1; k++) {
            espacioHTML += "  ";
        }

        for (var j = 1; j <= 2 * i + 1; j++) {
            asterisco += "*";
        }

        caja.innerHTML = caja.innerHTML + espacioHTML + asterisco+"<br>";
    }
```

```
    //Decreciente
    for (var i = limite-2; i >= 0; i--) {
        var asterisco = "";
        var espacioHTML = "";

        for (var k = 0; k < limite - i - 1; k++) {
            espacioHTML += "  ";
        }

        for (var j = 1; j <= 2 * i + 1; j++) {
            asterisco += "*";
        }

        caja.innerHTML = caja.innerHTML + espacioHTML + asterisco+"<br>";
    }

</script>

</body>

</html>
```

Ahora se pasará a analizar la parte creciente y decreciente del rombo:

- *Parte creciente.* En esta parte, se ejecuta un bucle creciente y se observa que el código más complejo son los bucles que almacenan los espacios HTML y los asteriscos en las variables espacioHTML y asterisco.

  En el caso de los espacios, estos van a ir decreciendo, dado que se establece en el bucle que la variable índice *k* sea menor que limite - i - 1 (siendo *i* el índice del bucle principal).

```
for (var k = 0; k < limite - i - 1; k++) {
    espacioHTML += "  ";
}
```

  En el caso de los asteriscos, es todo lo contrario. Al decrecer los espacios, crecen los asteriscos y, por tanto, el índice *j* del bucle será menor o igual a 2 * i + 1.

```
for (var j = 1; j <= 2 * i + 1; j++) {
    asterisco += "*";
}
```

- *Parte decreciente.* Es igual a la parte creciente, pero se observa que, en este caso, el bucle principal tiene que ser decreciente. El bucle comienza en límite-2, dado que se va a ejecutar solamente tres veces con valores del índice *i* igual a 2, 1 y 0.

```
for (var i = limite-2; i >= 0; i--) {
```

Es importante que el lector estudie con detenimiento la cabecera de los bucles, puesto que es la base y la complejidad del ejercicio.

Actividad propuesta 2.5

Utiliza el código anterior y comprueba cómo se muestra un rombo en el navegador. Si puedes, añade alguna característica complementaria, como que el rombo pueda desplazarse un número de espacios del margen izquierdo. Ese número de espacios podría ser configurable por el usuario.

Resumen

- Los programadores utilizan la sentencia console.log() para escribir información por consola.
- El separador de decimales en JavaScript es el punto y no la coma.
- JavaScript es sensible a las mayúsculas y minúsculas (case-sensitive).
- Es importante no poner a una variable el nombre *var,* puesto que es una palabra reservada.
- Los operadores en JavaScript son similares a Java.
- El operador typeof devuelve el tipo de una variable.
- JavaScript tiene solamente cinco tipos de datos: string, number, boolean, array y object.
- Se utiliza null en los objetos para indicar que no existen o que no tienen todavía un valor asignado.
- Existen varias funciones de conversión de datos como String(), Number(), parseInt(), parseFloat(), toDateString() o toUTCString().
- Las funciones en JavaScript comienzan con la palabra reservada *function*.
- Un ejemplo de objeto es el siguiente:

```
var perro = {raza:"Podenco", peso:12, altura:58,color:"negro"};
```

- Es importante el ámbito de utilización de una variable. Se recomienda no declarar funciones globales cuando solamente se van a utilizar en una función.
- Los eventos son cualquier suceso que le pueda ocurrir a un elemento HTML. El formato de programar un evento en JavaScript sería el siguiente:

```
<Elemento_HTML evento='código_JavaScript'>
```

- Con el método addEventListener, se puede vincular un evento a una función JavaScript.
- Con el método toString(), se puede convertir, por ejemplo, un número en una cadena de caracteres.
- En JavaScript, se pueden crear objetos de tipo fecha con el objeto Date, el cual tiene muchos constructores.

- El objeto Date tiene muchos métodos para obtener el día, la hora, el año, etc., de una fecha concreta.
- JavaScript tiene diferentes tipos de sentencias condicionales como if, else, else if o switch.
- En JavaScript, existen los bucles for y while.

Ejercicios propuestos

1. Realiza un programa JavaScript que calcule el área y la longitud de una circunferencia de radio 5.
2. Investiga el operador ternario y pon un ejemplo completo de uso.
3. Ejecuta las siguientes dos líneas de código y comprueba el contenido de la variable var. Interpreta y explica los resultados:

```
var dato = "Ronaldo " + 5 + 5;
var dato = 5 + 5 + " Ronaldo ";
```

4. El operador typeof permite saber el tipo de una variable. Crea un programa en el que pongas ejemplos de su utilización.
5. Crea un script con algún ejemplo de cómo se pueden sobrescribir variables y funciones globales.
6. JavaScript puede manejar eventos en una página web para conocer, por ejemplo, cuándo el ratón pasa por encima de un elemento. Pon un ejemplo de uso para cada uno de los siguientes eventos:

   - onmouseover.
   - onmouseout.
   - onblur.

7. Averigua qué resultado darán las siguientes líneas de código:

```
Number("9 9");
Number("dos");
```

8. Realiza un script que extraiga el mes y el año de una fecha concreta.
9. Romualdo, que todavía no ha terminado el ciclo de DAW, se pregunta si puede crear el siguiente array:

```
var milista = ['panadero', 456, [0, 1, 2]];
```

   También quiere saber si podría seleccionar la siguiente posición del array:

```
milista[2][2]
```

   Comprueba si puede utilizarlo e intenta hacer una pequeña web que muestre todo el contenido del array milista (si es posible).

10. Como variante de la práctica guiada, se pide realizar un script utilizando bucles que muestre en una página web el rombo de la figura 2.3. El número de filas que tiene el rombo será configurable.

Figura 2.3 Rombo hueco configurable.

Nota: aunque se puede realizar este ejercicio con todo lo aprendido en este capítulo, la función `repeat()` que se explica en el apartado 3.6.4 puede reducir y simplificar mucho el código resultante al eliminar y resumir algunos bucles.

ACTIVIDADES DE AUTOEVALUACIÓN

1. ¿Qué función se utiliza para escribir un mensaje por consola?:
   - ☐ *a)* `console.log()`
   - ☐ *b)* `console.alert()`
   - ☐ *c)* `alert()`

2. ¿Cuál de las siguientes instrucciones no es correcta?:
   - ☐ *a)* `var s = "hola";`
   - ☐ *b)* `var s = 'hola';`
   - ☐ *c)* `var pi = 3,14;`

3. Teniendo las siguientes líneas de código, la variable *x* valdrá:

```
var x = 10;
x %= 5;
```

   - ☐ *a)* 5.
   - ☐ *b)* 2.
   - ☐ *c)* 0.

4. En JavaScript, la siguiente sentencia hará que *dato* valga:

```
var dato = "Ronaldo " + 10;
```

   - ☐ *a)* "Ronaldo 10".
   - ☐ *b)* "Ronaldo".
   - ☐ *c)* Dará un error.

5. ¿Qué sintaxis se utilizará para definir una función suma en Javascript?:
   - ☐ *a)* `suma(a, b) {`
   - ☐ *b)* `function suma(a, b) {`
   - ☐ *c)* `function suma[a, b] {`

6. ¿Cuál de los siguientes sería el formato de programar un evento en JavaScript?:
   - ☐ *a)* `<Elemento_HTML evento='código_JavaScript'>`
   - ☐ *b)* `<Elemento_HTML evento={código_JavaScript}>`
   - ☐ *c)* `<Elemento_HTML evento=[código_JavaScript]>`

7. Cuando se abandona un campo de entrada, ¿cuál de los siguientes eventos se ejecuta?:
   - ☐ *a)* onleave.
   - ☐ *b)* onblur.
   - ☐ *c)* onchange.

8. Para vincular un evento clic a un objeto HTML llamado *myFPboton* para que se ejecute la función dale, ¿cuál de los siguientes listeners habrá que añadir?:
   - ☐ *a)*
   ```
   document.getElementById("myFPboton").addEventListener
   ("click", dale, false);
   ```
   - ☐ *b)*
   ```
   document.getElementById("myFPboton").addEventListener
   ("onclick", dale, false);
   ```
   - ☐ *c)*
   ```
   document.getElementById("myFPboton").addEventHandler
   ("onclick", dale, false);
   ```

9. En las siguientes líneas de código, la variable num2 valdrá:

```
var num = 9.99;
var num2 = num.toPrecision(1);
```

   - ☐ *a)* 10.
   - ☐ *b)* 1e+1.
   - ☐ *c)* 9.99.

10. ¿Cuál de las siguientes afirmaciones no es correcta?:
    - ☐ *a)* Se pueden tener objetos de diferentes tipos en un array.
    - ☐ *b)* `Dias.pop()` retira el primer elemento del array Dias.
    - ☐ *c)* `Dias.pop()` elimina el primer elemento del array Dias.

SOLUCIONES:

1. **a** b c	5. a **b** c	
2. a b **c**	6. **a** b c	9. a **b** c
3. a b **c**	7. a **b** c	10. a **b** c
4. **a** b c	8. **a** b c	