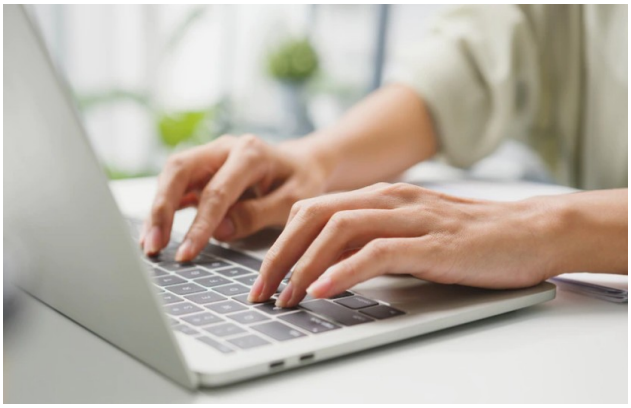
Desarrollo web en entorno cliente

Unidad 2: Manejo de la sintaxis del lenguaje

ÍNDICE

INTRODUCCIÓN

Una vez tenemos claras las diferentes **tecnologías** y **modelos** de programación web, es hora de ponernos manos a la obra.



Ya hemos escogido un lenguaje de programación que nos acompañará durante todo el módulo, **JavaScript**. Ahora pues, es el momento de empezar a estudiarlo en profundidad.

Veremos todo lo relativo a la **sintaxis básica del lenguaje**, variables, estructuras de control, funciones… Dejando la parte más compleja para unidades posteriores.

Para el correcto entendimiento de esta unidad será de vital importancia seguir el resto de asignaturas de programación del ciclo, ya que es mucho más sencillo aprender un **lenguaje de programación** cuando ya se domina alguno otro.

Así pues, en esta unidad podemos decir que estamos sentando las bases de lo que será la **programación web** que seguiremos durante el resto del ciclo.

OBJETIVOS / CAPACIDADES

En esta unidad de aprendizaje, las capacidades que más se van a trabajar son:

- ✔ **Utilizar lenguajes, objetos y herramientas, interpretando las especificaciones para desarrollar aplicaciones Web con acceso a bases de datos.**

PROYECTO DE LA UNIDAD

Antes de empezar a trabajar el contenido, te presentamos la **actividad** que está relacionada con esta unidad de aprendizaje. Se trata de un **caso práctico** basado en una **situación real** con la que te puedes encontrar en tu puesto de trabajo. Con esta actividad se evaluará la puesta en práctica de los **criterios de evaluación** vinculados al resultado de aprendizaje que se trabaja en esta unidad. Para realizarla deberás hacer lo siguiente: lee el enunciado que te presentamos a continuación, dirígete al área general del módulo profesional, concretamente a la actividad de evaluación que se encuentra dentro de esta unidad, allí encontrarás todos los detalles sobre fecha y forma de entrega, objetivos... A lo largo de la unidad irás adquiriendo los conocimientos necesarios para ir elaborando este proyecto.

Enunciado:

Adivina el número secreto

Gracias a nuestra evolución durante el curso, alguna empresa de prácticas ya se ha empezado a fijar en nosotros, pero antes de decidirse a hacernos un contrato, nos lanzan un reto a modo de prueba.

Se trata de crear un pequeño programa en JavaScript para adivinar un número escogido al azar. Deberás hacer lo siguiente.

- Crear un pequeño menú, donde el usuario escoja si quiere JUGAR, ver las INSTRUCCIONES o SALIR.
- La entrada de datos del usuario se hará mediante la instrucción prompt.
- Crear una función que devuelva un número aleatorio entre 1 y 10.
- Preguntar al usuario el número. Si acierta: felicitar y volver al menú, si no volver preguntar, hasta un máximo de 3 intentos.
- Comentar adecuadamente el programa.

1. ETIQUETAS Y UBICACIÓN DE CÓDIGO

A partir de este capítulo empezamos nuestra andadura con un nuevo lenguaje de programación: **JavaScript**. Por eso antes que nada es conveniente hacer algunas aclaraciones sobre JavaScript.

➔ **¿Qué es JavaScript?**:

- ✔ Es un lenguaje interpretado, lo podemos considerar un dialecto de ECMAScript.
- ✔ Es un lenguaje de script, y entre sus atributos destacan:
  - Polimorfismo.
  - Herencia por delegación (entre objetos y no entre clases).
  - Abstracción.
  - Encapsulación.
- ✔ Se utiliza generalmente en el lado del cliente, pese a que en contadas ocasiones existen implementaciones para el lado del servidor.

➔ **¿Qué no es JavaScript?**:

- ✔ No es un lenguaje orientado a objetos.
- ✔ JavaScript no es Java, pese a que se parezca el nombre.

Una vez que conocemos mejor que es **JavaScript**, podemos adentrarnos en su manera de **programar**.

Podemos decir que **JavaScript** es un **lenguaje** que, generalmente, requiere de un **código anfitrión** para poder ser ejecutado. Esto suele ser alguna **página HTML**, que funcionan mediante etiquetas.

*Como vimos en la unidad anterior, cuando queramos utilizar código JavaScript en una página HTML bastará con insertarlo dentro de una **etiqueta script**, ya sea como código en línea o mediante fichero externo.*

Vídeo: ¿qué es JavaScript?

Visualiza este vídeo en el que podrás ver una explicación sobre qué es JavaScript.
https://www.youtube.com/embed/riZbwRFMFuw

2. VARIABLES Y CONSTANTES. ÁMBITOS DE UTILIZACIÓN. DECLARACIÓN Y CONVENCIONES. INICIALIZACIÓN

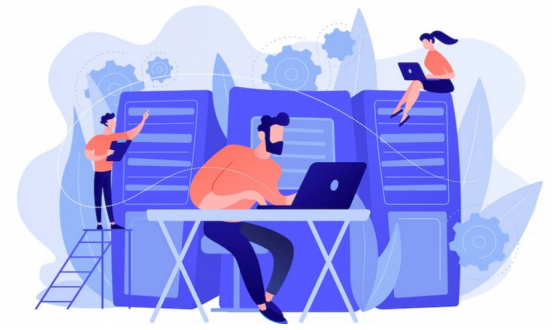
Como cualquier lenguaje de programación, **JavaScript** utiliza **variables** para poder almacenar datos y trabajar con ellos.

Podemos definir una variable como un lugar en memoria dónde se almacenará información, para posteriormente, en cualquier momento del programa, acceder a esos datos.

Las variables tienen un nombre, que identifica qué información se está guardando dentro.

Este nombre, por convenciones establecidas suele empezar con letra minúscula, y tiene que ser ni muy corto ni muy largo, lo suficiente como para ser auto explicativo del nombre de la variable.

Para nombres de variables con varias palabras, podemos usar dos **técnicas básicas de separación**:



- **Técnica del camello**: consiste en poner la primera letra de cada palabra (excepto de la primera) en mayúsculas. A modo de las jorobas de un camello:

  `diasTranscurridos.`

- **Técnica de la serpiente**: consiste en separar palabras mediante el guión bajo:

  `dias_transcurridos.`

Además, deberíamos evitar caracteres no estándar en nombres de variables, como tildes, la eñe…

Veamos ahora algunos **aspectos relevantes sobre las variables**:

➔ **Definir variable**: las variables se definen mediante la palabra **var**. Así podemos tener una variable que almacene texto:

```
var saludo = "Hola"
```

➔ **Valor inmutable**: si por el contrario queremos que ese valor sea inmutable, usaremos las constantes, esta vez con la palabra **const**:

```
const saludo = "Hola"
```

➔ **Variables de bloque**: destacar, por último, que, desde hace algunos años, se permiten lo que se denomina variables de bloque. Usando la palabra **let**, conseguimos que el ámbito de una variable sea su propio bloque y no la función que lo contiene.

```
if (a < 3) {
   let num = 3;
}
```

➔ **Inicializar variable**: para inicializar una variable, basta con darle un valor con el símbolo **"=",** tal y como hemos visto en los ejemplos anteriores.

3. TIPOS DE DATOS. TIPOS ESPECIALES Y DEFINIDOS POR EL USUARIO

Hemos visto que una **variable** sirve para guardar datos, pero no hemos visto qué tipos de datos se pueden almacenar en esas variables.

DESTACADO

JavaScript es un lenguaje de los que se denomina de tipado dinámico, al estilo de PHP, esto significa que una variable puede cambiar de tipo durante la ejecución del programa. Cosa que no pasa, por ejemplo, con lenguajes tipados como Java.

Si declaramos una variable, pero no le asignamos ningún valor, JavaScript la considerará **undefined**. Si declaramos una variable, y posteriormente, la volvemos a declarar, perderá el primer valor que habíamos asignado.

Cuando declaramos una variable, no necesitamos indicarle el tipo de datos, JavaScript lo infiere implícitamente. Esto no significa que no haya tipos de datos. De tal manera que podemos distinguir estos **tipos de datos principales**:

- **String**. Indicado con comillas simples o dobles.

```
var saludo = "Hola";
var saludo = 'Hola;
```

- **Number**. En JavaScript no hay enteros ni decimales, todo se considera un número.

```
var numero = 11;
var peso = 95.8;
```

➔ **Boolean**. Cierto o falso.

```
var ocupado = true;
```

➔ **Object**. Es una colección no ordenada de elementos.

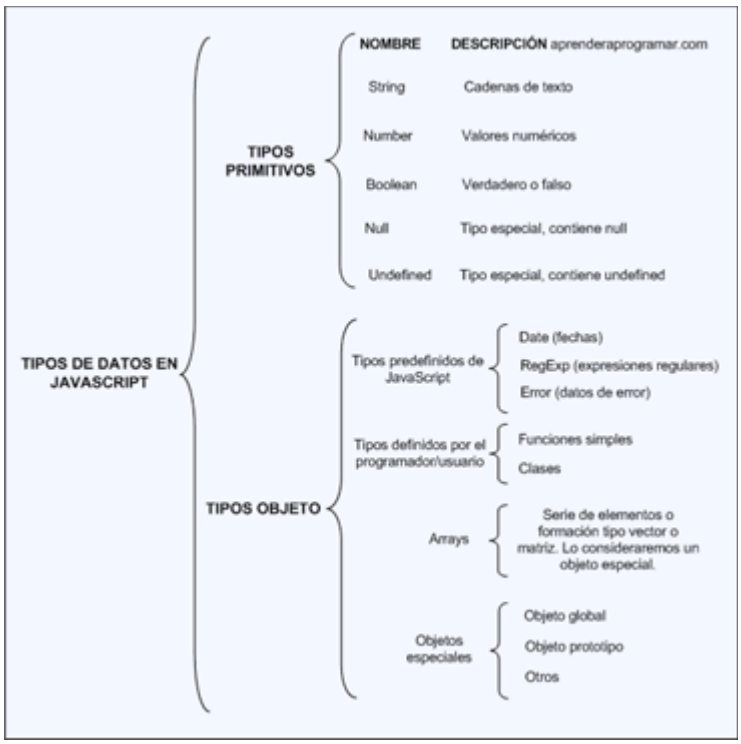
```
var coches = ["Mazda", "Toyota", "BMW"];
```

Un usuario también puede definir sus propios tipos de objetos, cosa que ampliaremos en próximos capítulos.

```
var coche = {marca: "Mazda", modelo: "3"};
```

Finalmente, JavaScript tiene algunos **tipos especiales** que nos permiten trabajar, por ej, con Fechas, como es el caso de Date.

```
var cumple = new Date(1979, 8, 18);
```



4. CONVERSIONES ENTRE TIPOS DE DATOS

En ciertas ocasiones, tenemos una variable con un cierto tipo de datos, que por algún motivo nos interesaría que fuera de otro tipo de datos diferente.

Este es uno de los motivos para usar **conversiones** entre tipos de datos. En JavaScript, podemos distinguir entre conversión de datos **implícita** o **explícita**.

- **Conversión de datos implícita**: como en muchos otros lenguajes, **JavaScript** utiliza en muchas ocasiones conversión de datos automática. Lo cual significa que es capaz, por ejemplo, de convertir un número a **String** para mostrarlo por pantalla, como en el siguiente ejemplo.

```
var numero = 11;
document.write("Mi número favorito es el " + numero);
```

Otro caso de conversión implícita se da cuando usamos operaciones matemáticas, que convierten los valores de tipos **String** a **números** para poder operar.

```
var a = "12";
var b = "3";
var division = a / b;
```

En el ejemplo anterior, la variable division tendrá por valor 4, convirtiendo previamente los dos **String** a **Number**, antes de realizar la operación matemática.

- **Conversión de datos explícita**: si, por el contrario, lo que queremos es convertir algún valor a un tipo diferente de manera manual, usaremos la

conversión explícita. Esto significa que tenemos que indicar a qué tipo de datos queremos convertir.

Es lo que se conoce en otros lenguajes como **cast**.

Las tres **conversiones** más utilizadas son:

- ✔ **ToString**. No tiene demasiada complicación, básicamente convierte cualquier cosa a String para poder representarla como texto.

```
var num = 11; // 11
var numTexto = String(num); // "11"
```

- ✔ **ToNumber**. Intenta convertir un valor a número.

```
var numTexto = "11"; // "11"
var num = Number(numTexto); // 11
```

  En caso de no poder hacerse la conversión, el resultado será NaN.

```
var texto = "Hola"; // "Hola"
var num = Number(texto); // NaN
```

- ✔ **ToBoolean**. Los valores vacíos se convierten a false, el resto a true.

```
var a = Boolean(1); // true
var b = Boolean(0); // false
var c = Boolean("Hola"); // true
var d = Boolean(""); // false
```

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Cuál es el precursor de JavaScript?

a. Java.

b. HTML.

c. ECMAScript.

Lee el enunciado e indica la opción correcta:

¿Cómo se denomina a la conversión automática de datos?

a. Implícita.

b. Explícita.

c. No existe.

Lee el enunciado e indica la opción correcta:

¿Qué instrucción no sirve para declarar una variable?

a. var.

b. const.

c. let.

5. LITERALES

En JavaScript, podemos definir un literal como un valor, no modificable, que podemos asignar a una variable o usar en cualquier expresión.

```
var saludo = "Hola";
var numero = 11;
```

En el anterior ejemplo, tanto "**Hola**", como 11 son **literales**.

Entre los literales nos podemos encontrar diferentes **tipos principales**:

- **Array de literales**:cuando declaramos un array con unos determinados valores iniciales, estamos creando un array de literales.

  ```
  var coches = ["Mazda", "Toyota", "BMW"];
  ```

  La **longitud del array** viene especificada, implícitamente, por la cantidad de literales que contenga. En el anterior ejemplo, la longitud del array es 3.

Podemos dejar posiciones del array vacías, simplemente dejando un espacio entre las comas.

  Así, si queremos crear un array con 4 coches, pero sin especificar la marca de uno de ellos, haríamos:

  ```
  var coches = ["Mazda", , "Toyota", "BMW"];
  ```

  En este caso el array tendrá 4 **posiciones**, teniendo la segunda un valor undefined.

- **Literales booleanos**: son básicamente los **valores true** y **false**, para indicar cierto y falso respectivamente.
- **Literales de tipo String**: los podemos representar mediante **comillas simples o dobles**. Existen algunos caracteres especiales, que se pueden expresar mediante la "\", como en la mayoría de lenguajes. Por ejemplo:
  - **\n** -> Salto de línea.

- ✔ **\t** -> Tabulador.
- ✔ **\" ->** Comillas dobles.

➔ **Literales de tipo objeto**: son una lista de parejas **clave-valor**, al estilo de otros lenguajes de programación. Van encerradas entre claves.

```
var persona = {nombre: "Jaime", edad: "42", país: "España"};
```

Podemos acceder a cada una de sus propiedades con la notación por punto:

```
var nombre = persona.nombre; // Jaime
```

6. ASIGNACIONES

Asignar un valor a una variable significa **darle valor**, o sea que la variable guarde el valor que le estamos asignando.

*Para asignar valores en JavaScript, como en la mayoría de **lenguajes modernos**, se usa el operados "=", seguido del valor que queremos asignar.*

```
var nombre = "Judit";
```

A continuación, veremos algunos **aspectos relevantes sobre la asignaciones de valores**:

- **Reemplazar**: la anterior instrucción asigna el valor "Jaime" a la variable nombre, y, además, implícitamente también está indicando que será una variable de tipo String.

Cuando asignamos un valor a una variable que ya tenía alguno, se dice que lo estamos reemplazando. Es decir, deja de tener el valor antiguo para tener el nuevo.

- **Variables de tipo primitivos**: podemos asignar también, en lugar de literales, el valor de una cierta variable a otra, con lo que la primera obtendrá el valor que tenía la segunda.

```
var nombre2 = nombre; // Judit
```

  Esto es así, mientras sean variables de tipos **primitivos**, como **String**, **Number**, **Boolean**… En cambio, si son de tipos de datos complejos (arrays, objetos o functions), la asignación se hace por referencia y no por valor.

- **Otros operadores de asignación**:

Existen, además, otros operadores de asignación que nos permiten realizar cálculos acumulativos sobre la variable en la que los utilizamos.

Funcionan para las operaciones de **suma**, **resta**, **multiplicación**, **división** y **módulo**. Se trata de los siguientes:

```
var a = 2;
a += 3; // 5 -> a = a + b;
var a = 2;
a -= 3; // -1 -> a = a - b;
var a = 2;
a *= 3; // 6 -> a = a * b;
var a = 2;
a /= 3; // 0.66666 -> a = a / b;
var a = 2;
a %= 3; // 2 -> a = a % b;
```

7. OPERADORES

Los **operadores de asignación**, que acabamos de ver son un tipo muy concreto de operador, pero existen algunos tipos más, que son los que veremos a continuación.

➔ **Operadores aritméticos**: nos permiten realizar operaciones matemáticas entre variables, literales y expresiones. Aquí encontramos:

- ✔ **Suma** (+): suma dos valores.
- ✔ **Resta** (-): resta dos valores.
- ✔ **Multiplicación** (*): multiplica dos valores.
- ✔ **División** (/): divide dos valores.
- ✔ **Módulo** (%): residuo de dividir un valor entre otro.
- ✔ **Incremento** (++): incrementa una variable en una unidad.
- ✔ **Decremento** (--): decrementa una variable en una unidad.

```
var a = 5;
var b = 7;
a++; // 6
b--; // 6
document.write(a+b); // 12
document.write(a%b);  // 0
document.write(a/3);  // 2
document.write(a*b);  // 36
```

➔ **Operadores relacionales**: permiten comparar dos valores o expresiones para saber si son iguales, diferentes, mayores o menores.

- ✔ **Igual** (==).

- ✔ **Idéntico** (===): mismo valor y mismo tipo.
- ✔ **No igual** (!=).
- ✔ **No idéntico** (!==): no son iguales o no son del mismo tipo.
- ✔ **Mayor** (>).
- ✔ **Menor** (<).
- ✔ **Mayor o igual** (>=).
- ✔ **Menor o igual** (<=).

Estos operadores tomarán vital importancia cuando lleguemos a las sentencias condicionales o iterativas.

➔ **Operadores lógicos**: permiten crear expresiones más complejas, a menudo utilizados conjuntamente con los operadores relacionales.

- ✔ **And** (&&): la expresión resultado es cierta si ambas partes lo son.
- ✔ **Or** (||): la expresión resultado es cierta si cualquiera de las partes lo es.
- ✔ **Not** (!): retorna el valor booleano contrario a la expresión.

```
var a = 5;
var b = 6;
console.log(a < b && b > a); // true
console.log(a < b && b < a); // false
console.log(a < b && b < 33); // true
console.log(a < b || b > a); // true
console.log(a < b || b < a); // true
console.log(a > b || b < 33); // true
```

Vídeo: operadores

Visualiza este vídeo para saber más sobre los operadores.
https://www.youtube.com/embed/iQiKSzAXL6U

8. EXPRESIONES

Una expresión en programación se puede entender como la combinación de valores, literales, variables y operadores para conseguir cualquier resultado.

Una expresión puede contener **diferentes tipos de valores**, y además podemos decir que dentro de una expresión puede haber más expresiones.

Para que una expresión pueda ser evaluada, tiene que ser **coherente** con todas sus partes. Es decir, tiene que estar correctamente escrita.

Una cosa a tener muy en cuenta en las expresiones es la precedencia o prioridad. Al tener expresiones complejas que mezclen operadores aritméticos, lógicos y/o relacionales hemos de tener en cuenta qué parte de la expresión se evaluará antes.

Para eso podemos utilizar los **paréntesis**, al igual que lo haríamos en matemáticas, indicando que la parte entre paréntesis tiene preferencia a la hora de evaluar.

Puede haber paréntesis anidados, es decir, paréntesis dentro de otros paréntesis, que afectarán a la prioridad.

Otra manera de saber qué parte de la expresión se evaluará antes es la prioridad de los propios operadores, tanto los aritméticos como los lógicos.

➔ **Prioridad de operadores aritméticos**:

Sigue el orden siguiente:

- ✔ **Negativo** (-).
- ✔ **Multiplicación, división y módulo** (*, /, %).
- ✔ **Suma y resta** (+, -).

Los operadores de dentro del mismo grupo anterior, tienen la misma prioridad. Es decir, da igual primero sumar y luego restar, que, al revés, por ejemplo.

➔ **Prioridad de operadores lógicos**:

Sigue el siguiente orden:

- ✔ **Not** (!).
- ✔ **And** (&&).
- ✔ **Or** (||).

Los operadores aritméticos, además, tienen prioridad sobre los operadores lógicos.

```
var a = 15 + 59 * 75 / 9 < 2 // false
```

9. COMENTARIOS AL CÓDIGO

Cuando programamos en cualquier lenguaje es muy importante, no sólo que el código sea lo más **eficiente** posible, si no, que, además, intentemos hacerlo lo más **comprensible** posible.

PRESTA ATENCIÓN

Para ese propósito son muy importantes los comentarios de código. Podemos decir que un comentario es un texto, escrito por el desarrollador que ayuda a comprender el código o ciertas partes de él.

Estos comentarios no son **compilados** o **interpretados** antes de ser ejecutado el código, por lo que podemos escribir con **lenguaje natural** para expresarnos de la mejor manera.

Los comentarios son especialmente potentes cuando **trabajamos en equipo**, ya que para cualquier persona es más sencillo leer lo que hace el código que tener que entender realmente cómo lo hace.

Si posteriormente necesita profundizar en la implementación se puede revisar el código, si no, muchas veces, basta con leer los comentarios.

TOMA NOTA

Podemos encontrarnos comentarios de una línea o de varias líneas.

- **Comentarios de una línea**: empiezan con el símbolo **//**. A partir de ahí, el resto de la línea donde esté ubicado se entiende como un comentario, y por tanto no se considera código ejecutable.

  Se pueden poner al **principio de la línea**, o al **final** de ella.

```
// Esto es un comentario de una línea

var a = 2; // Esto también es un comentario de una línea
```

Pese a que se llamen comentarios de una línea, podemos poner varias líneas seguidas comentadas de esta manera. Eso sí, cada una habrá de empezar con **//**.

```
// Comentario de varias
// líneas
```

➔ **Comentario de más de una línea**: se usan los símbolos **/*** y ***/** para indicar el principio y fin de un comentario de varias líneas.

Puede ser útil si queremos comentar varias líneas a la vez de manera más sencilla, y, sobre todo, se usan mucho cuando estamos creando la documentación de un programa.

```
/* Esto es un
comentario de
varias líneas */
```

Actividad de aprendizaje 1: comentar un código

Ahora que ya habéis empezado a estudiar el lenguaje JavaScript, tu compañero y tú os veis con ganas de empezar a escribir algún programa propio, pero lamentablemente no sabéis por dónde empezar.

Habéis encontrado un código por internet que parece interesante, así que para tener las cosas más claras creéis que lo mejor sería comentarlo para ir entendiendo cada línea.

```
var letras = "TRWAGMYFPDXBNJZSQVHLCKET";
var dni = 12345678;

var posicion = dni % 23;
var letraCalculada = letras.substring(posicion, posicion + 1);

document.write(dni + "-" + letraCalculada);
```

Entrega el archivo .js en el espacio habilitado para ello.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Con qué símbolos declaramos un array?

a. {}.

b. [].

c. ().

Lee el enunciado e indica la opción correcta:

¿Qué operador usamos para acumular un valor en una variable?

a. +=

b. =

c. ==

Lee el enunciado e indica la opción correcta:

¿Qué operador es más prioritario?

a. &&

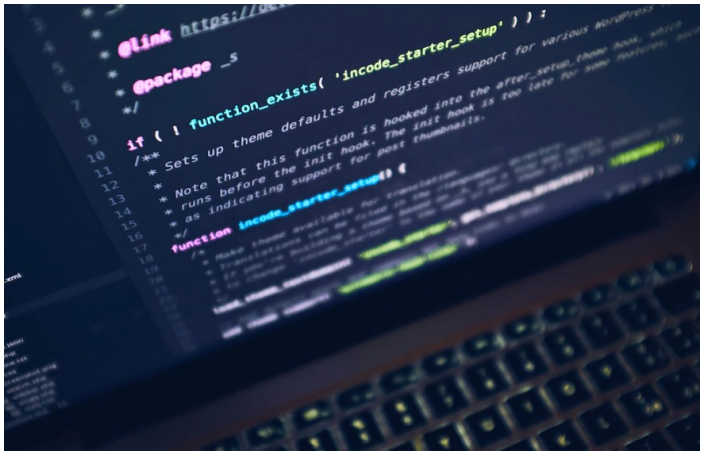
b. !

c. ||

10. SENTENCIAS

En programación, podemos entender como sentencias cada una de las líneas de código que realizan una cierta labor dentro del programa. También podemos denominarlas instrucciones.

Dependiendo del propósito de la sentencia podrá ser de un tipo u otro. Así, por ejemplo, podemos encontrarnos con:



- Sentencias de **asignación**.
- Sentencias de **cálculos**.
- Sentencias que llaman a **funciones**.
- Sentencias **condicionales**.
- Sentencias **iterativas**.

Más adelante en la unidad veremos en profundidad algunas de estas, como las condicionales e iterativas.

A continuación, veremos algunos **aspectos relevantes sobre las sentencias**:

- ➔ Una sentencia contiene una **instrucción** y puede ser ejecutada de manera independiente. No obstante, las sentencias suelen estar **relacionadas** entre sí, por lo que al ejecutar una instrucción el resto del programa a continuación se puede ver afectado.
- ➔ Como hemos visto, gracias al **depurador**, podemos ejecutar el código sentencia a sentencia, para buscar posibles errores que pueda surgir en nuestro programa.

➔ **En JavaScript**, las instrucciones o sentencias acaban con un punto y coma, excepto las líneas que empiezan o finalizan un bloque, que contienen claves.

```
var a = 11;
a = a + 3;
document.write(a);
```

En el siguiente capítulo veremos ampliamente lo que son los bloques de sentencias.

11. BLOQUES DE CÓDIGO Y BLOQUES DE SENTENCIAS. ESTRUCTURAS DE CONTROL DE FLUJO

En ciertas estructuras de programación nos encontraremos con que un número determinado de sentencias guardan una cierta relación entre sí, es ahí donde entran los **bloques** de sentencias.

En la mayoría de ocasiones estos bloques de sentencias, van escritos entre **claves**, que especifican el inicio y fin del bloque.

```
var a = 25;

if (a > b) {
    a = 3;
    document.write(a);
}
```

Asociado a esto, también existe el concepto de ámbito de las variables, en inglés scope, que especifica que una variable sólo puede ser utilizada dentro del bloque en que se ha creado.

A continuación, veremos algunos **aspectos relevantes sobre los bloques**:

- **Si estamos fuera del bloque**, es como si esa variable no existiera. Una manera de verlo es imaginar los bloques como si fueran cajas opacas. Desde fuera de la caja no podemos ver lo que hay dentro, pero desde dentro de la caja sí podemos verlo.
- Hay que destacar también que, al igual que los paréntesis anidados de los operadores, los bloques de sentencias también **se pueden anidar**, abriendo y cerrando nuevos bloques dentro de uno ya existente.
- Los bloques de código más comúnmente utilizados son los que corresponde a las **estructuras** de **flujo**, que veremos en los siguientes capítulos.
  - ✔ Estructuras **condicionales**.

- ✔ Estructuras **iterativas**.

Podemos decir que una estructura de control de flujo nos permite modificar el **flujo secuencial** normal de un programa.

➔ Es decir, las instrucciones no se ejecutarán una detrás de otra si no que puede haber saltos entre ellas en base a una serie de **condiciones**.

Dentro de un bloque de instrucciones encerrado en claves, estas instrucciones irán tabuladas hacia la derecha, respecto al código que está fuera del bloque.

12. DECISIONES. SENTENCIAS CONDICIONALES. SENTENCIAS DE RUPTURAS Y CONTINUACIÓN

En cualquier lenguaje de programación necesitamos formas de tomar **decisiones** en base a alguna **condición**. Esto nos permite alterar el flujo normal de un programa, haciendo que ciertas instrucciones se ejecuten o no.

A continuación, veremos el concepto de **condición** y el de **estructura condicional**:

- **Condición**:

Una condición, por lo tanto, es una expresión que nos devolverá un valor de cierto o falso (true o false), una vez evaluada. Es lo que se denomina, de otra manera, expresiones booleanas.

- **Estructura condicional**: estas decisiones se utilizan, por ejemplo, en las sentencias condicionales. Pero, **¿Qué es una estructura condicional?**

Una estructura condicional es aquella que nos permite ejecutar o no una serie de instrucciones basándose en una cierta condición.

A continuación veremos la **sintaxis** de una de las variantes más completas de la estructura condicional y también conoceremos la **estructura switch**:

- **Sintaxis**: hay variantes de esta **estructura**, siendo la más completa la que tiene la siguiente sintaxis:

```
if (condición 1) {
   // instrucciones si se cumple la condición 1

} else if (condición 2) {
   // instrucciones si se cumple la condición 2

} else {
     // instrucciones si no se ha cumplido ninguna condición
anterior

}
```

Si queremos, por ejemplo, saber la calificación de un alumno en función de su nota numérica podríamos utilizar el siguiente código:

```
if (nota < 5) {
    alert("No apto");
} else if (nota < 9) {
    alert("Apto");
} else {
    alert("Excelente");
}
```

➔ **Estructura switch**: otra estructura condicional es la que se denomina switch. Esto nos permite ejecutar ciertas instrucciones, no en base a una condición, si no en **base al valor** de una expresión.

Por ejemplo, si queremos mostrar por pantalla cuál es el color favorito, guardado en una variable.

```
var favcolor = "red";
switch (favcolor) {
    case "red":
        alert("Your favorite color is red!");
       break;
    case "green":
        alert("Your favorite color is green!");
        break;
    default:
}
```

Cabe destacar que la estructura **switch** hace uso de la instrucción "**break**", que se es lo que se denomina una sentencia de ruptura. Las sentencias de ruptura sirven para salir de manera brusca de un bloque de código, por lo que hay que ser cuidadoso con su uso.

No se recomienda utilizarlas demasiado más allá de las estructuras switch, donde sí están completamente aceptadas.

Vídeo: switch

Visualiza el siguiente vídeo en el que hablamos más detenidamente sobre la estructura Switch.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Qué es una estructura condicional en programación?

a. Una expresión que devuelve un valor booleano.

b. Una sentencia de ruptura en un bloque de código.

c. Una construcción que permite ejecutar o no instrucciones basándose en una condición.

Lee el enunciado e indica la opción correcta:

¿Cuál es la sintaxis más completa de la estructura condicional "if-else"?

a. if (condición) { } else if (condición) { } else { }.

b. if (condición) { } else { }.

c. if (condición) { } else if (condición) { }.

Lee el enunciado e indica la opción correcta:

¿Cuál es la función de la sentencia "break" en la estructura "switch"?

a. Salir de manera brusca de un bloque de código.

b. Evaluar una condición en base al valor de una expresión.

c. Finalizar la ejecución de un programa.

13. BUCLES. ANIDACIÓN DE BUCLES

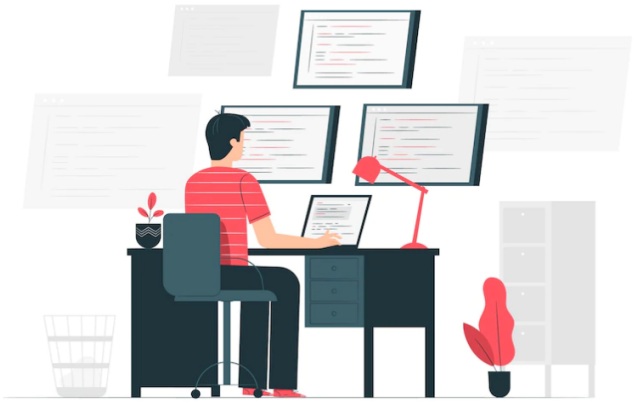
Hay ciertas veces que necesitamos ejecutar un cierto código un número **repetido** de veces. Eso es lo que se denominan **estructuras iterativas**, o simplemente **bucles**.

DESTACADO

Podemos definir un bucle como una estructura que nos permite repetir la ejecución de ciertas instrucciones mientras se dé una cierta condición, que se evaluará cada vez que acabe la ejecución del bloque de instrucciones a ejecutar.

Hay cuatro **tipos básicos de bucles** en JavaScript:

- **while**: repite un bloque de código mientras se cumpla la condición dada.
- **do while**: ejecuta un bloque de código al menos una vez, y después lo repite entras se cumpla la condición.
- **for**: repite un bloque de código un cierto número de veces.



- **for in**: repite un bloque de código para cada propiedad de un objeto.

Veamos algunos **ejemplos** con su sintaxis. Para escribir por pantalla los números del 1 al 5 usaríamos los siguientes ejemplos:

➔ **While**:

```
var x = 1;
while(x <= 5) {
    console.log("The number is: " + x);
    x++;
}
```

➔ **Do while**:

```
var x = 1;
do {
    console.log("The number is: " + x);
x++;
} while (x <= 5);
```

➔ **For**:

```
for (var i = 1; i <= 5; i++) {
   console.log("The number is: " + x);
}
```

Veamos ahora un ejemplo del **for in**, que nos servirá para recorrer un array de coches:

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
for (x in cars){
    console.log(cars[x]);
}
```

Vídeo: estructuras iterativas

VÍDEO

Visualiza este vídeo en el que hablamos sobre estructuras iterativas.

14. FUNCIONES

Una función es un bloque de código que puede ser utilizado una o más veces en cualquier punto de nuestro programa.

Su uso tiene una serie de **ventajas** como son:

- La **reutilización** de código: Permite utilizar muchas veces un cierto código que sólo escribimos una vez.
- La **organización** del código: Aunque sólo vayamos a ejecutar un código una sola vez, ponerlo dentro de una función nos permite una mejor organización del código.
- La **escalabilidad** del código: Hacer crecer el programa es mucho más sencillo si está codificado de maneras modular, mediante funciones.

- La **parametrización** del código: Podemos ejecutar la misma función con diferentes parámetros, lo que producirá, seguramente, resultados distintos.

Los objetos en JavaScript tienen ya funciones implícitas asociadas, pero también se permite la creación de nuevas funciones por parte del desarrollador.

A continuación, veremos **aspectos relevantes sobre las funciones de JavaScript**:

➔ Una función en JavaScript tiene el siguiente **aspecto**:

```
function writeMsg() {
    alert("Hello world!");
}
```

➔ Para **llamar a la función** basta con invocar a su nombre:

```
writeMsg();
```

➔ Además, una función puede tener **parámetros** que permitan que la ejecución sea distinta en función de estos:

```
function writeMsg(msg) {
    alert(msg);
}
writeMsg("Hello world");
writeMsg("Today is a beautiful day");
```

Los parámetros se consideran variables internas de la propia función. Pueden ser de cualquier tipo, y se tienen que respetar en la llamada a la función.

➔ Además, se les puede **asignar un valor** por defecto, que se usará si no se especifica ningún valor en la llamada:

```
function writeMsg(name="John", surname="Doe") {
    alert("Hello " + name + " " + surname);
}
writeMsg();
writeMsg("James");
writeMsg("James", "Bond");
```

➔ Finalmente, las funciones pueden **retornar algún valor**, como, por ejemplo, el resultado de un cálculo. Ese valor retornado puede ser usado en el momento de hacer la llamada:

```
function suma(a, b) {
    return (a + b);
}
alert(suma(5,6));
```

15. CONVENCIONES DE FORMATO Y CODIFICACIÓN

En cualquier lenguaje de programación existen una serie de **normas** escritas, y a veces no escritas, que tratan acerca de cuál es la mejor forma de programar.

Es lo que se conoce como guías de estilos. En ellas se explican las convenciones de formato para tratar de que todos los desarrolladores programen de una manera similar.

Esto se convierte en una herramienta prácticamente imprescindible para cualquier programador.

Pese a que no es obligatoria su utilización, sí que es muy recomendable, ya que, si todo el mundo utiliza JavaScript de la misma manera, será más fácil entender el código de otros desarrolladores, o incluso el nuestro propio pasado algún tiempo sin tocarlo.

Algunas de las cosas que se tienen en cuenta en la **guía de estilos** son:

- Tabulación del código.
- Uso de mayúsculas y minúsculas.
- Normas en los nombres de variables o funciones.
- Uso de los distintos operadores.
- Uso de claves o paréntesis.
- Declaración e inicialización de variables.
- Comentarios.

Resumiendo, lo ideal para ser un buen programador, a parte evidentemente de los conocimientos, es tener un estilo de programación correcto y que cumpla con los **estándares**, así el **trabajo en equipo** será mucho más sencillo.

Encontrarás el enlace a la guía de estilo de JavaScript en la bibliografía de esta unidad.

Actividad de aprendizaje 2: crea tu propio código

Ya con las ideas mucho más claras acerca de cómo funciona JavaScript, crees que definitivamente es el momento de crear tu propio código, tu primer programa.

Éste consistirá en una pequeña web que mostrará, a partir de un día, mes y año, guardados en variables, si una fecha es correcta o no.

Las tres variables serán numéricas y la página ha de mostrar si la fecha es o no correcta.

Por ejemplo:

- La fecha 8/7/2019 es correcta.
- La fecha 29/2/2022 no es correcta.

Entrega el archivo .js en el espacio habilitado para ello.

Cuestionario

Lee el enunciado e indica la opción correcta:

¿Qué palabra reservada usamos en una sentencia condicional?

a. for.

b. while.

c. if.

Lee el enunciado e indica la opción correcta:

¿Cuál es la sentencia de ruptura?

a. Switch.

b. Break.

c. For.

Lee el enunciado e indica la opción correcta:

¿Cuántas veces se ejecuta al menos un bucle do-while?

a. Al menos una.

b. Puede que ninguna.

c. Más de dos.

RESUMEN

En esta unidad hemos empezado a estudiar un nuevo **lenguaje de script**, como es **JavaScript**.

Comparándolo con lenguajes tradicionales como **Java** vemos que tiene gran parte de similitudes, aunque también sus diferencias, empezando por la manera de **compilar** o **ejecutar**.

Hemos visto qué son las **variables**, como declararlas, inicializarlas y utilizarlas. De qué tipos pueden ser, e incluso como realizar **conversiones** entre tipos.

Como en todo lenguaje de programación, hemos incidido en la importancia de los **comentarios**, y en cómo utilizarlos de la mejor manera.

Por último, hemos visto en profundidad las distintas **estructuras de control de flujo**, como son las **condicionales**, e **iterativas**, en todas sus variantes.

Resumiendo, ahora ya podemos hacer nuestro primeros **programas sencillos en JavaScript.**

RECURSOS PARA AMPLIAR

PÁGINAS WEB

- Ámbito de las variables (Introducción a JavaScript): *https://uniwebsidad.com/libros/javascript/capitulo-4/ambito-de-las-variables* [Consulta junio 2022].
- Por valor vs por referencia en JavaScript: *https://medium.com/laboratoria-developers/por-valor-vs-por-referencia-en-javascript-de3daf53a8b9* [Consulta junio 2022].

BIBLIOGRAFÍA

PÁGINAS WEB

- JavaScript and HTML DOM Reference: *https://www.w3schools.com/jsref/default.asp* [Consulta junio 2022].
- JavaScript Standard Style: *https://standardjs.com/rules.html* [Consulta junio 2022].
- JSON - JavaScript | MDN: *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON* [Consulta junio 2022].
- W3C: *https://www.w3.org/* [Consulta junio 2022].

GLOSARIO

- **Array**: conjunto de elementos del mismo tipo, con una cierta ordenación.
- **ECMAScript**: es una especificación de lenguaje web, predecesor, de alguna manera, de JavaScript.
- **NaN (Not a Number)**: se da cuando una operación no es matemáticamente posible.