MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO DE WEB ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Operadores del lenguaje y bucles

04

ÍNDICE

© MEDAC 978-84-18983-24-5
Reservados todos los derechos. Queda rigurosamente prohibida, sin la autorización escrita de los titulares del copyright, bajo las sanciones establecidas en las leyes, la reproducción, transmisión y distribución total o parcial de esta obra por cualquier medio o procedimiento, incluidos la reprografía y el tratamiento informático.

ÍNDICE

OBJETIVOS

- *Conocer los principales operadores del lenguaje*
- *Saber crear expresiones en el lenguaje*
- *Saber qué son las sentencias de código*
- *Conocer los mecanismos para realizar bucles*
- *Utilizar herramientas de desarrollo*
- *Crear bloques de código complejos*

/ 1. Introducción y contextualización práctica

JavaScript es un lenguaje que nos permite crear códigos de cierta complejidad para solucionar los problemas que nos plantee el cliente. En el lenguaje podemos encontrar estructuras realmente complejas como la sentencia o las expresiones, entre otras. Al igual que sucede con otros lenguajes de programación, JavaScript ofrece diferentes mecanismos para crear comportamientos iterativos y de toma de decisiones.

A lo largo de este tema veremos las principales estructuras que ofrece JavaScript para poder crear un código más elaborado que permite realizar tareas de control. Estas tareas se suelen denominar «control de flujo». Como en todos los lenguajes de programación, JavaScript ofrece diferentes operadores que requieren un orden específico de ejecución, lo que se conoce como precedencia de operadores.

Fig. 1. JavaScript como lenguaje de cliente

Escucha el siguiente audio que describe el caso práctico que iremos resolviendo a lo largo de la unidad:

Audio Intro. JavaScript: Sistemas condicionales
https://bit.ly/2XNt6Xw

/ 2. Palabras reservadas

En todo lenguaje de programación se dispone de palabras reservadas que son propias de cada lenguaje y que no pueden ser modificadas. Gracias a las palabras podemos utilizar el lenguaje de programación en cuestión. En este sentido, JavaScript dispone de un conjunto de palabra reservadas bastante sencillo mediante el que podemos desarrollar nuestra aplicación. La mayor diferencia entre las palabras reservadas y los identificadores es que las primeras no pueden contener guiones, mayúsculas, ni números. Al tratarse palabras reservadas, no podemos crear identificadores con ese nombre. Algunas de las palabras reservadas de JavaScript se muestran a continuación:

Palabra reservada	Descripción
function	Permite declarar una función
for	Permite crear un bloque que se repita tantas veces como la condición sea cierta
break	Permite salir de un bloque ejecución
switch	Permite elegir qué bloque de sentencias ejecutar a partir de una condición
do...while	Ejecuta un bloque de sentencias y lo repite tantas veces como la condición sea cierta
If...else	Permite ejecutar un bloque de sentencias u otro dependiendo de una condición

Tabla 1. Principales palabras clave de JavaScript

/ 3. Sintaxis del lenguaje

La sintaxis de cualquier lenguaje de programación es el conjunto de reglas que nos permite crear código en dicho lenguaje. Para aprender un lenguaje de programación es necesario conocer tanto la sintaxis como las palabras reservadas del propio lenguaje. Se dice que un lenguaje puede ser más fácil o difícil de aprender dependiendo de su sintaxis.

Esta sintaxis permite determinar cómo declarar variables, asignar tipo a las variables, ejecutar operaciones, etc. JavaScript es un lenguaje sensible a mayúsculas y minúsculas que sigue una notación Lower Camel Case. Además, en JavaScript existen diferentes tipos de valores: fijos y variables. Los valores fijos no pueden ser modificados y se denominan **literales.**

3.1. Sentencias

Utilizando la sintaxis del lenguaje y conociendo sus palabras reservadas, podemos crear sentencias. Desde un punto de vista algorítmico, un programa informático consiste en un conjunto de instrucciones que un computador debe ejecutar. En un lenguaje de programación, este conjunto de instrucciones recibe el nombre de sentencias. Al igual que sucede en el resto de los lenguajes de programación, el código de JavaScript se organiza en sentencias. Las sentencias en JavaScript pueden estar formadas por: valores, operadores, expresiones, palabras claves y comentarios. En JavaScript las sentencias se ejecutan una a continuación de otra en el orden que han sido escritas. Las sentencias van separadas por ";", lo que permite escribir en la misma línea en varias sentencias, siempre y cuando vayan separadas correctamente.

3.2. Cadenas

En JavaScript las cadenas se utilizan para manejar series de caracteres. Las cadenas van definidas con comillas dobles/simples al inicio y al final.

```
var moto1 = "Yamaha TMAX";  // Dobles comillas
var moto2 = 'Yamaha TMAX';  // Comillas simples
```
Código 1. Cadenas de caracteres

Vídeo 1. Notación y cadenas
https://bit.ly/2MapvgD

3.3. Bloques de código

En JavaScript las sentencias pueden ir en bloques de código que están delimitados por llaves. La idea de agrupar el código en bloques permite su ejecución y facilita su reutilización. Existen bloques anónimos y bloques que tienen un nombre asignado. A continuación, se muestra un ejemplo de un bloque con nombre en JavaScript:

```
function test(){
var a, b, c;
a = 5;
b = 6;
c = a + b;
}
```
Código 2. Bloque de código con nombre

Como se puede observar en el ejemplo anterior, disponemos de un bloque de código en el que se declara un conjunto de variables y se realizan continuas asignaciones.

3.4. Expresiones

Una expresión es la combinación de valores, variables y operadores para generar un valor determinado. Cuando una expresión es ejecutada se dice que se ha evaluado.

3.5. Operadores

Además de las palabras reservadas, los lenguajes de programación ofrecen operadores. Gracias a los ellos, podemos realizar diferentes tareas como evaluar operaciones matemáticas y expresiones lógicas, tomar decisiones, etc. El operador por excelencia de un lenguaje de programación es la **asignación.**

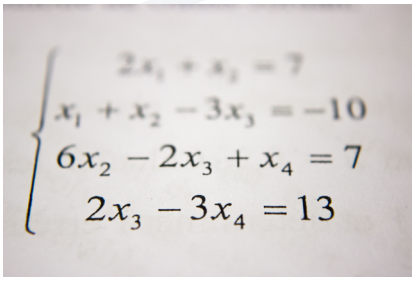

Fig. 2. JavaScript permite utilizar expresiones aritméticas

Audio 1. "JavaScript: Expresiones, operadores y precedencia".
https://bit.ly/3ciGoAi

/ 4. Caso práctico 1: "Error con palabras reservadas"

Planteamiento. Durante el desarrollo de una aplicación web vemos que el navegador nos devuelve un error al ejecutar un fragmento del código. Sabemos que el código que devuelve el error está localizado en las siguientes líneas:

```
var for = 10
var var2 = "sol"
var var = "Madrid";
```

Código 3. Error de código

Nudo. ¿Qué crees que puede estar pasando? ¿Por qué se está produciendo el error?

Fig. 3. ¿Cómo arreglamos el código?

Desenlace. Como hemos visto en el apartado de las palabras reservadas, no es posible declarar identificadores con el nombre de una palabra reservada, pues el lenguaje no lo permite. En el ejemplo anterior, se están utilizándolos palabras reservadas como si fuesen identificadores, es decir, variables. Se puede observar que se declara una variable con el nombre for, siendo este el nombre de una palabra reservada. También se declara otra variable con el nombre var, que es otra palabra reservada del lenguaje.

/ 5. Operadores aritméticos y de comparación

Los operadores aritméticos son aquellos que permiten realizar operaciones con números. Algunos de los operadores aritméticos más utilizados en JavaScript son los siguientes:

Operador	Descripción
+	Operador suma
-	Comprador resta
*	Operador de producto
**	Operador de potencia
/	Operador división
%	Operador módulo.
++	Operador de incremento
--	Operador de decremento

Tabla 2. Operadores aritméticos en JavaScript

En JavaScript, la precedencia de los operadores aritméticos es igual a la de los operadores matemáticos. Para modificar la precedencia se utiliza los paréntesis.

Por su parte los operadores de comparación permiten realizar comparaciones entre diferentes expresiones o variables. A continuación, se muestran los operadores de comparación más utilizados en JavaScript:

Operador	Descripción
==	Operador de igualdad
===	Igualdad de valor y de tipo
!=	Operador distinto
!==	Operador distinto valor o tipo
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Tabla 3. Operadores de comparación en JavaScript

/ 6. Operadores de asignación y lógicos

La asignación se puede utilizar también para realizar varias operaciones en una única sentencia. Para ello, se utilizan los operadores de asignación que proporciona JavaScript. A continuación, se muestra un ejemplo de operadores de asignación:

Operador abreviado	Ejemplo	Equivalencia
=	A = B	A = B
+=	A += B	A = A + B
-=	A -= B	A = A – B
*=	A *= B	A = A * B
/=	A /= B	A = A / B
%=	A %= B	A = A % B
**=	A **= B	A = A ** B

Tabla 4. Operadores de asignación en JavaScript

Los operadores lógicos son aquellos que permiten realizar operaciones que devuelven siempre un valor de «verdad». Los operadores lógicos más utilizados en JavaScript son los siguientes:

Operador	Descripción
&&	Y lógico
||	O lógico
!	Negación lógica

Tabla 5. Operadores lógicos en JavaScript

Estos se pueden utilizar para comparar el valor de verdad de expresiones que devuelvan otro valor de verdad (verdadero o falso).

/ 7. Operadores a nivel de bits

Los operadores a nivel de bits permiten realizar cualquier operación con valores que puedan ser representados con 32 bits. A continuación, se muestra un ejemplo de ello:

Operador	Descripción	Ejemplo
&	'Y' lógico a nivel de bit	2 & 0 equivalente a 10 & 00 = 00
|	'O' lógico a nivel de bit	2 | 0 equivalente a 10 | 00 = 10
~	Negación	~ 2 equivalente a ~10 = 01

Tabla 6. Operadores lógicos a nivel de bits en JavaScript

En JavaScript, una vez que se computa el valor de verdad de una expresión, se devuelve el valor decimal.

/ 8. Condicionales

Cualquier solución software requiere de la utilización de comparaciones, pues se trata de una acción básica. Se pueden utilizar sentencias condicionales para ello. En JavaScript se presentan diferentes tipos de sentencias condicionales. En este apartado estudiaremos todas ellas.

8.1. Sentencia if-else

Esta sentencia condicional tienes dos partes: una cuando se cumple la condición y otra en caso de no cumplirse que es opcional.

```
var result = 0
if (today > 10) {
 result++ ;
}else{
 result--;
}
```

Código 4. Sentencia if-else

En el ejemplo anterior podemos observar una sentencia condicional que tiene dos partes; result valdrá uno si se cumple la condición. En caso contrario, tomará un valor negativo. Es importante destacar que la segunda parte de la expresión es una opción posible. La sentencia condicional se puede anidar, teniendo varias condiciones unas tras otras.

8.2. Sentencia switch

Esta sentencia permite realizar diferentes acciones acordes a diferentes condiciones. Es aconsejable utilizar la sentencia switch cuando el nivel de anidamiento en sentencias condicionales simples es muy elevado.

```
switch (today) {
  case 0:
    day = “Lunes”;
    break;
  case 1:
    day = “Martes”;
    break;
  default: {
        day =”error”;
        break;
  }
}
```
Código 5. Ejemplo de funcionamiento sentencia switch

Dependiendo del valor que tenga la variable del bloque switch se ejecutará el caso asociado. Están definidas las opciones 0 y 1 y, en caso contrario, el valor del día será error.

/ 9. Bucles

Los bucles permiten ejecutar un bloque de sentencias en un número determinado de repeticiones. En todo lenguaje de programación esto es una tarea común. Dependiendo del lenguaje de programación, existen diferentes tipos de bucles. JavaScript ofrece al menos tres formas de bucles que veremos a continuación.

9.1. Bucle for

Es el tipo de bucle más común en todos los lenguajes de programación.

```
for (i = 0; i < 5; i++) {
  text += “El número es “ + i + “<br>”;
}
```
Código 6. Bucle for

En el ejemplo anterior, se va a llevar acabo el siguiente flujo de ejecución. Se ejecuta la primera sentencia que fija el contador a cero (i = 0). A continuación, se comprueba que “i” < cinco. Cada vez que se ejecute el bloque dentro del bucle se incrementará en 1 el contador (i++).

9.2. Bucle while

Con este tipo de bucle se repiten las sentencias mientras la condición sea verdadera.

```
while (i < 10) {
  text += “El número es “ + i;
  i++;
}
```
Código 7. Ejemplo texto de tipo de bucle

En este ejemplo se va a ejecutar el bloque de código mientras se cumple la condición, o sea, mientras i<10. Como se puede observar, no existe una declaración previa de la variable contador, es decir, “i” no está definida previamente.

9.3. Bucle do-while

Es una variante del anterior, en este bloque de código se ejecuta siempre al menos una vez, pues la condición se comprueba una vez ejecutado el bloque. El resto del funcionamiento es exactamente igual, cambiando únicamente la sintaxis. Veamos un ejemplo:

```
do {
 text += “El número es “ + i;
 i++;
}
while (i < 10);
```

Código 8. Ejemplo bucle do-while

9.4. Visual Studio Code

Se trata de un editor de código multiplataforma para múltiples lenguajes, entre ellos, HTML y JavaScript. Posee un amplio número de complementos para el desarrollo en diferentes plataformas. Es un entorno bastante sencillo de usar y permite aprender con relativa facilidad sobre todo en entorno web. Proporciona integración con gestores de código como Git y SVN.

Vídeo 2. “Bucles en JavaScript con Visual Studio Code”.
https://bit.ly/2XXtiUH

/ 10. Caso práctico 2

Planteamiento. Tras analizar un fragmento del código en nuestra aplicación web, vemos que parte del código se puede mejorar sintácticamente según el lenguaje JavaScript. El fragmento que se podría mejorar se muestra a continuación:

```
for (iCont =0; iCont < 5; iCont++) {
 console.log(“Valor:” + iCont);
}
```

Código 9. Mejora de código caso práctico 2

Según nos dice el departamento de sistemas, este código debería ejecutarse al menos una vez.

Nudo. ¿Cómo mejorarías el código anterior?

Desenlace. Como hemos visto en el apartado sobre bucles, al utilizar un bucle for, el bloque de sentencia se ejecutará tantas veces como se indique en la condición. Si la condición no se cumple, no se ejecutará en ningún caso. Una forma de mejorar esto, teniendo en cuenta los requisitos que nos piden, sería cambiar el bucle for por un bucle do-while. Con el nuevo tipo de bucle propuesto, se garantiza que el bloque de sentencias se ejecutará al menos una vez antes de evaluar la condición.

/ 11. Resumen y resolución del caso práctico de la unidad

En este tema nos hemos centrado en estudiar gran parte de la sintaxis de JavaScript. Hemos presentado los operadores aritméticos, lógicos y de asignación. En cada uno de ellos hemos visto algún ejemplo en el que se muestran los usos de los operadores.

También hemos presentado las sentencias y los bloques de código que nos han permitido estudiar las diferentes formas condicionales e iterativas del lenguaje JavaScript. Cabe destacar que las expresiones aritméticas y lógicas se pueden utilizar como condiciones en las sentencias condicionales. Además, hemos presentado la herramienta desarrollo que vamos a utilizar durante la asignatura que también nos va a servir para depurar el código que generemos: Visual Studio Code.

Resolución del caso práctico de la unidad
Como hemos visto en este tema, JavaScript es un lenguaje de programación que ofrece diferentes sentencias para introducir condiciones en el código. Cuando el número de condiciones es limitado, es recomendable utilizar sentencias if anidadas, pues se pueden seguir con cierta facilidad. Sin embargo, cuando el número de condiciones aumenta, como sucede en el caso planteado, es mejor utilizar sentencias switch. Un aspecto que debemos considerar es el tipo de datos de la variable condición que se utilizará en switch ya que, en algunos lenguajes, esta variable solo puede ser de tipo entero.

/ 12. Bibliografía

Flanagan, D. (2020). JavaScript: The Definitive Guide. (7th edition). Sebastopol: O'Reilly Media, Inc.

Haverbeke, M. (2019). Eloquent JavaScript: a modern introduction to programming. San Francisco: No Starch Press.

MEDAC