

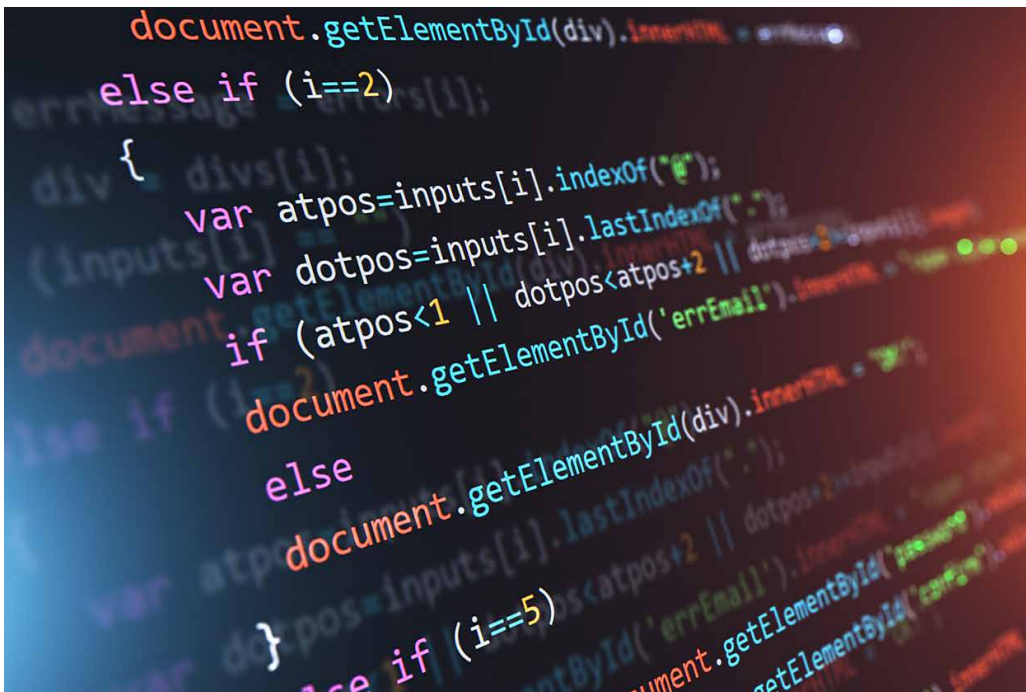
Programación con JavaScript I

# Controles de flujo

El flujo en la ejecución de un programa elaborado en lenguajes de programación como JavaScript es lineal, es decir, se va ejecutando desde la primera línea hasta la última, a menos que este orden sea alterado por algún bloque o instrucción específica que cambie este flujo.

A los bloques de código que tienen la capacidad de modificar el flujo de ejecución se les llama estructuras de control de flujo. Dichas estructuras son instrucciones del tipo “si ocurre esto, entonces haz esto, de otra manera, realiza esta otra actividad”. Otro tipo sería “repite esto mientras se cumpla alguna condición”.

Estas modificaciones que cambian el flujo de un programa permiten crear programas que pueden tomar decisiones dependiendo de los valores de las variables. Con esto, los programas dejan de ser una simple sucesión lineal de instrucciones.



```
document.getElementById(div).innerHTML = "Error";  
else if (i==2)  
{  
    var atpos=inputs[i].indexOf("@");  
    var dotpos=inputs[i].lastIndexOf(".");  
    if (atpos<1 || dotpos<atpos+2 || dotpos>inputs[i].length-1)  
        document.getElementById('errEmail').innerHTML = "Error";  
    else  
        document.getElementById(div).innerHTML = "OK";  
}  
if (i==5)
```

Una expresión es una frase de JavaScript que puede ser evaluada para generar un valor. “Las condicionales son expresiones que nos permiten ejecutar una secuencia de instrucciones u otra diferente dependiendo de lo que estemos comprobando” (Azaustre, 2016). Para poder realizar estas comparaciones, se utilizan los operadores de comparación:

Operador	Descripción	Ejemplo
==	<b>Igualdad:</b> devuelve true si ambos operandos son iguales.	3 == 3 3 == '3'
!=	<b>Desigualdad/diferente:</b> devuelve true si los operandos no son iguales.	3 != 4
===	<b>Estrictamente iguales:</b> devuelve true si los operandos son iguales y tienen el mismo tipo.	3 === 3
!==	<b>Estrictamente desiguales:</b> devuelve true si los operandos no son iguales o no son del mismo tipo.	3 !== '3'
>	<b>Mayor que:</b> devuelve true si el operando de la izquierda es mayor que el operando de la derecha.	4 > 3
>=	<b>Mayor o igual que:</b> devuelve true si el operando izquierdo es mayor o igual que el operando derecho.	4 >= 4
<	<b>Menor que:</b> devuelve true si el operando izquierdo es menor que el operando derecho.	12 < 15
<=	<b>Menor o igual que:</b> devuelve true si el operando izquierdo es menor o igual que el operando derecho.	12 <= 12

También se utilizan operadores lógicos, los cuales se emplean generalmente con valores booleanos. Sin embargo, los operadores and (&&) y or (||), en realidad devuelven el valor de uno de los operandos especificados, por lo que si alguno de esos operadores se usa con valores no booleanos, puede devolver un valor no booleano.

Operador	Descripción	Ejemplo
&&	<b>And:</b> devuelve true si ambos operandos son true.	const a = 'Gato' && 'Perro'; // t && t devuelve Perro
	<b>Or:</b> devuelve true si alguno de los operandos es true.	const o = 'Gato'    'Perro'; // t    t devuelve Gato
!	<b>Not:</b> devuelve false si su único operando puede convertirse a true.	const n = !'Gato'; // !t devuelve false
??	<b>Nullish:</b> devuelve la parte derecha del operador cuando la parte izquierda es null o undefined.	const x = null ?? 'Cat'; // null ?? t devuelve Cat

Ahora que conoces todos los operadores que se utilizan en las expresiones, se nombrarán los condicionales más útiles en JavaScript:

1. El tipo de condición más común es **if**. Dependiendo del resultado de la condición, obtenemos un valor u otro. Se puede utilizar de tres maneras:

Condicional	Sintaxis
if Simple:	<pre>if (condición){   //bloque_de_código }</pre>
if/else	<pre>if (condición) {   //bloque de código 1 } else {   //bloque de código 2 }</pre>
if/elseif	<pre>if (condición 1) {   //bloque de código 1 } else if (condición 2) {   //bloque de código 2 } else {   //bloque de código 3 }</pre>

Ejemplo:

```
const hora = 13;  
let saludo;  
  
if (hora < 12) {  
  saludo = "Buenos días";  
}  
  
if (hora < 20) {  
  saludo = "Buenas tardes";  
}  
  
if (hora >= 20) {  
  saludo = "Buenas noches";  
}  
  
console.log(saludo) // Buenas tardes
```

Ahora utilizando else if:

```
const hora = 13;
let saludo;

if (hora < 12) {
  saludo = "Buenos días";
} else if (hora < 20) {
  saludo = "Buenas tardes";
} else (hora >= 20) {
  saludo = "Buenas noches";
}

console.log(saludo) // Buenas tardes
```

2. **Switch:** con esta sentencia podemos sustituir las sentencias **if-else** y podemos hacer que nuestro código sea más legible y elegante. Utiliza la palabra reservada **break** para poder separar los bloques. Ejemplo:

```
switch (condición) {
  case condición_1:
    //bloque de código 1
    break;
  case condición_2:
    //bloque de código 2
    break;
  default:
    //bloque de código 3
}
```

Ejemplo:

```
switch (dia) {
  case 1:
    text = "Domingo";
    break;
  case 2:
    text = "Lunes";
    break;
  case 3:
    text = "Martes";
    break;
  case 4:
    text = "Miércoles";
    break;
  case 5:
    text = "Jueves";
    break;
  case 6:
    text = "Viernes";
    break;
  case 7:
    text = "Sábado";
    break;
  default:
    text = "Error";
}

console.log(text); // Martes
```



En JavaScript, el código se organiza mediante **bloques** que están delimitados por llaves, una de apertura "{" y otra de cierre "}". Estos bloques pueden servir en la definición de objetos o para la utilización de alguna funcionalidad. Por ejemplo:

```
if (condición){ //inicio de bloque
    //código del bloque
} //fin de bloque
```

La **asignación condicional**, también conocida como operador ternario, "sirve para asignar en una sola línea un valor determinado si la condición que se evalúa es verdadera u otro si es falsa" (Azaustre, 2016). La sintaxis es la siguiente:

```
condición ? valor_si_verdadero : valor_si_falso
```

Y se lee de la siguiente manera: si la condición devuelve verdadero, tomará el valor de *valor\_si\_verdadero*, en caso contrario, tomará el de *valor\_si\_falso*

Ejemplo:

Si se utilizara un if/else, la instrucción sería la siguiente:

```
const speed = 120
let message

if(speed > 100) {
    message = "Estas yendo muy rápido!"
} else {
    message = "Bien, vas debajo del límite"
}

console.log(message) // Estas yendo muy rápido!
```

Cuando se utiliza el operador ternario, la instrucción se ve de la siguiente manera:

```
const speed = 120
const message = speed > 100 ? "Estas yendo muy rápido!" : "Bien, vas debajo del límite"
console.log(message) // Estas yendo muy rápido!
```

Por otro lado, todas las variables en JavaScript contienen intrínsecamente un valor booleano, este valor es conocido como **truthy** o **falsy**.

Los siguientes valores siempre serán falsy:

- false
- 0 (cero numérico)
- "" o "" - string vacío
- Null (valor nulo)
- Undefined (valor indefinido)
- NaN (no es un numérico)

Todo lo demás es **truthy**:

- '0' - string conteniendo cero
- 'false' - string con el texto false
- [] - arreglo vacío
- {} - objeto vacío
- Function (){} - función vacía

Esto te será de mucha utilidad cuando desees utilizar condicionales, ya que se puede evaluar un solo valor sin la necesidad de operadores lógicos.

En ocasiones se requiere que un bloque de código sea ejecutado cierto número de veces, para ello, se utilizan los **bucles**, dado que nos permiten, como dice Flanagan (2020), recorrer el mismo código una y otra vez, cada vez con un valor diferente y esto sucederá mientras se cumpla una condición.

En la programación, en cualquier lenguaje, existen tres elementos que nos ayudan a controlar el flujo del ciclo o bucle (loop). El primero es la **inicialización**, que fija los valores con los que va a iniciar el bucle, el segundo es la **condición de permanencia** y el tercero es la **actualización** de las variables de control, es decir, el incremento al ejecutarse la iteración.

JavaScript cuenta con cinco tipos de bucles:

1. **While:** el bloque de código dentro de este bucle se ejecutará **mientras** se cumpla la condición.

Ejemplo:

```
var i = 1; //inicialización

while(i < 11) { // condición de permanencia
  console.log(i);
  i++; //actualización de la variable de control
}
```

2. **Do/While:** este es similar al anterior, pero con la diferencia de que se ejecutará el bloque de código dentro de **do** por primera vez y después se comprueba la condición de permanencia en el bucle. De esta manera se asegura que **al menos se ejecute una vez el código** del bucle.

```
var x = 0;
do {
  console.log(x);
  x++;
} while (x < 10); // Regresa: 0 1 2 3 4 5 6 7 8 9
```

3. **For:** la siguiente sentencia es muy potente, pues nos permite resumir en una línea la forma de un bucle **while**. Su sintaxis es la siguiente:

```
for(inicialización; condición de permanencia; actualización){
  bloque de código
}
```

Ejemplo de cómo se verían los bucles anteriores utilizando for:

```
for (var x = 0; x < 10; x++) {
  console.log(x)
} // Regresa 0 1 2 3 4 5 6 7 8 9
```

4. **For/of:** este es definido por ES6. Este tipo de bucle utiliza la sentencia for, pero trabaja con objetos iterables como *arrays*, *strings*, *sets* y mapas. Por ejemplo, podemos usar *for/of* para ciclar a través de los elementos de un array de números y computar la suma.

```
let datos = [1, 2, 3, 4, 5, 6, 7, 8, 9], sum = 0;
for (let element of datos) {
  sum += element;
}
sum // Regresa 45
```

5. **For/in:** con este tipo de bucle podemos recorrer los nombres de propiedades de un objeto especificado de una manera más sencilla. La sintaxis es **for(variable in object)**, siendo **variable** el nombre de la propiedad y **object** el valor de la propiedad. Este sería un ejemplo:

```
var libro = {
  titulo: "Javascript",
  autor: "David Flanagan",
  numPaginas: 1245,
  editorial = "O'Really",
  precio: "10.45"
};

for (let prop in libro) {
  console.log("la propiedad " + prop + " contiene: " + libro[prop]);
} //Regresa:
//La propiedad título contiene: Javascript
//la propiedad autor contiene: David Flanagan
...
//la propiedad precio contiene: 11.45
```



Además del uso que se puede dar en la sentencia **switch** de la palabra reservada **break**, también se puede utilizar para salir de un bucle. Ejemplo:

```
for (let i = 0; i < 10; i++) {  
  if (i === 3) break  
  console.log('Number: ${i}')
```

En este ejemplo, **break** termina el bucle cuando el contador **i** es igual a 3. Por otro lado, **continue** no detiene el bucle, solo se brinca una iteración y continúa con el ciclo.

```
for (let i = 0; i < 10; i++) {  
  if (i === 3) continue  
  console.log('Number: ${i}')
```

Como has observado, en JavaScript se puede tener un programa lineal que va ejecutando línea tras línea hasta que encuentra alguna expresión de control de flujo. En este tema aprendiste que las expresiones permitirán ejecutar un conjunto de instrucciones cuando se cumpla una condición y si no se cumple, ejecutarán un conjunto de instrucciones diferente. Por otro lado, conociste que los ciclos permiten que un bloque de código se ejecute hasta que se deje de cumplir cierta condición.

Estos dos conceptos te permitirán dar inteligencia a la aplicación y podrás controlar su flujo dependiendo de los valores de entrada.

## Referencias bibliográficas

- Azaustre, C. (2016). *Aprendiendo JavaScript*. Recuperado de [https://www.academia.edu/36052337/Aprendiendo\\_JavaScript\\_Carlos\\_Azaustre\\_FREELIBROS](https://www.academia.edu/36052337/Aprendiendo_JavaScript_Carlos_Azaustre_FREELIBROS)
- Flanagan, D. (2020). *JavaScript: the Definitive Guide* (7a ed.). Estados Unidos: O'Reilly Media, Inc.
- Haverbeke, M. (2018). *ELOQUENT JAVASCRIPT* (3ª ed.). Estados Unidos: No Starch Press.

## Para saber más

**Los siguientes enlaces son externos a la Universidad Tecmilenio, al acceder a ellos considera que debes apegarte a sus términos y condiciones.**

### Lecturas

- MDN. (s.f.). *Tomando decisiones en tu código - Condicionales*. Recuperado de [https://developer.mozilla.org/es/docs/Learn/JavaScript/Building\\_blocks/conditionals](https://developer.mozilla.org/es/docs/Learn/JavaScript/Building_blocks/conditionals)
- Kantor, I. (2022). *Operadores Lógicos*. Recuperado de <https://es.javascript.info/logical-operators>

### Videos

- jonmircha. (2020, 25 de febrero). *Curso JavaScript: 13. Tipos de Operadores - #jonmircha* [Archivo de video]. Recuperado de [https://www.youtube.com/watch?v=\\_8Z5AeGVIXE](https://www.youtube.com/watch?v=_8Z5AeGVIXE)
- jonmircha. (2020, 26 de febrero). *Curso JavaScript: 14. Condicionales - #jonmircha* [Archivo de video]. Recuperado de [https://www.youtube.com/watch?v=9h5hyh\\_wDjo](https://www.youtube.com/watch?v=9h5hyh_wDjo)
- jonmircha. (2020, 27 de febrero). *Curso JavaScript: 15. Ciclos (Loops) - #jonmircha* [Archivo de video]. Recuperado de <https://www.youtube.com/watch?v=AapgtR0Rwk0>

## Checkpoints

### Asegúrate de:

- Utilizar correctamente los operadores lógicos en el flujo de tus programas desarrollados con lenguaje JavaScript.
- Usar las sentencias de ciclos correctos en cada caso que se presente.
- Identificar correctamente cada una de las opciones para crear condicionales y poder utilizarlas en la situación adecuada.

## Requerimientos Técnicos

- Computadora con acceso a Internet.
- Editor de texto.
- Permisos de administrador y/o Git instalado previamente.

## Prework

- Haber realizado los ejercicios de la Actividad 1.
- Leer detenidamente y comprender el material explicado en este tema.
- Practicar todos los ejemplos que se describen previamente en este tema.
- Revisar cada uno de los recursos adicionales que se proponen en este tema.

Tecmilenio no guarda relación alguna con las marcas mencionadas como ejemplo. Las marcas son propiedad de sus titulares conforme a la legislación aplicable, se utilizan con fines académicos y didácticos, por lo que no existen fines de lucro, relación publicitaria o de patrocinio.

---

La obra presentada es propiedad de ENSEÑANZA E INVESTIGACIÓN SUPERIOR A.C. (UNIVERSIDAD TECMILENIO), protegida por la Ley Federal de Derecho de Autor; la alteración o deformación de una obra, así como su reproducción, exhibición o ejecución pública sin el consentimiento de su autor y titular de los derechos correspondientes es constitutivo de un delito tipificado en la Ley Federal de Derechos de Autor, así como en las Leyes Internacionales de Derecho de Autor.

El uso de imágenes, fragmentos de videos, fragmentos de eventos culturales, programas y demás material que sea objeto de protección de los derechos de autor, es exclusivamente para fines educativos e informativos, y cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por UNIVERSIDAD TECMILENIO.

Queda prohibido copiar, reproducir, distribuir, publicar, transmitir, difundir, o en cualquier modo explotar cualquier parte de esta obra sin la autorización previa por escrito de UNIVERSIDAD TECMILENIO. Sin embargo, usted podrá bajar material a su computadora personal para uso exclusivamente personal o educacional y no comercial limitado a una copia por página. No se podrá remover o alterar de la copia ninguna leyenda de Derechos de Autor o la que manifieste la autoría del material.