

Agencia de  
Aprendizaje  
a lo largo  
de la vida

# FULL STACK PYTHON

## Clase 14

Javascript 2

# Condicionales y Ciclos



JS

# Les damos la bienvenida

Vamos a comenzar a grabar la clase

## Clase 13

**Introducción a Javascript**

- ¿Qué es y para qué se usa?
- Conceptos generales. Sintaxis básica.
- Variable, ¿qué es y cómo declararla? Tipos.
- Asignación y cambio del valor.
- Operadores aritméticos.
- Conversión a entero y flotante.

## Clase 14

**Condicionales y Ciclos**

- Control de flujos.
- Condicional. ¿Qué es?
- Operadores lógicos y de comparación.
- Ciclos. ¿Qué son? Tipos y diferencias entre sí.
- Cómo combinar operadores lógicos y ciclos.

## Clase 15

**Programación modular con funciones**

- Funciones. ¿Qué son? Parámetros de entrada y de salida.
- Scope global y local.
- Programación modular vs. Funciones.
- Función anónima y función flecha.

# Operadores lógicos y relacionales

# Operadores de comparación

Un operador de comparación (o relacional) compara sus operandos y devuelve un **valor lógico** en función de si la comparación es verdadera (**true**) o falsa (**false**). Los operandos pueden ser valores numéricos, de cadena, lógicos u objetos. Las cadenas se comparan según el orden lexicográfico estándar. En la mayoría de los casos, si los dos operadores no son del mismo tipo, **JavaScript** intenta convertirlos a un tipo apropiado para la comparación. Este comportamiento generalmente resulta en comparar los operadores numéricamente.

| Operador | Descripción         |
|----------|---------------------|
| ==       | igual a             |
| ===      | igual valor y tipo  |
| !=       | no igual a          |
| !==      | igual valor no tipo |
| >        | mayor que           |
| <        | menor que           |
| >=       | mayor o igual que   |
| <=       | menor o igual que   |
| ?        | operador ternario   |

# Operadores lógicos

Los operadores lógicos se utilizan normalmente con valores booleanos (lógicos).

| Operador          | Descripción           |
|-------------------|-----------------------|
| <b>&amp;&amp;</b> | Y lógico (Conjunción) |
| <b>  </b>         | O lógico (Disyunción) |
| <b>!</b>          | NO lógico (Negación)  |

**! (Negación)**

| Prop A | Resultado    |
|--------|--------------|
| !True  | <b>False</b> |
| !False | <b>True</b>  |

**&& (Conjunción)**

| Prop A | Prop B | Resultado    |
|--------|--------|--------------|
| True   | True   | <b>True</b>  |
| True   | False  | <b>False</b> |
| False  | True   | <b>False</b> |
| False  | False  | <b>False</b> |

**|| (Disyunción)**

| Prop A | Prop B | Resultado    |
|--------|--------|--------------|
| True   | True   | <b>True</b>  |
| True   | False  | <b>True</b>  |
| False  | True   | <b>True</b>  |
| False  | False  | <b>False</b> |

# Operadores prefijo y posfijo

Los **afijos** se anteponen o se posponen en un nombre de una variable. Cuando hablamos de **prefijo** nos referimos a que se antepone a la variable y el **posfijo** se pospone. Se utilizan para realizar operaciones aritméticas, tanto para incrementar como para decrementar el valor de una variable.

| Operador     | Descripción        | Ejemplo                               |
|--------------|--------------------|---------------------------------------|
| <b>i++</b>   | incremento posfijo | a=i++ primero a=i y después i=i +1    |
| <b>++ i</b>  | incremento prefijo | a=++i primero i=i +1 y después a=i    |
| <b>i - -</b> | decremento posfijo | a=i - - primero a=i y después i=i - 1 |
| <b>- - i</b> | decremento prefijo | a=- - i primero i=i - 1 y después a=i |



# Operadores de asignación

No solamente el `=` (igual) es un operador de asignación. Existen otras variantes:

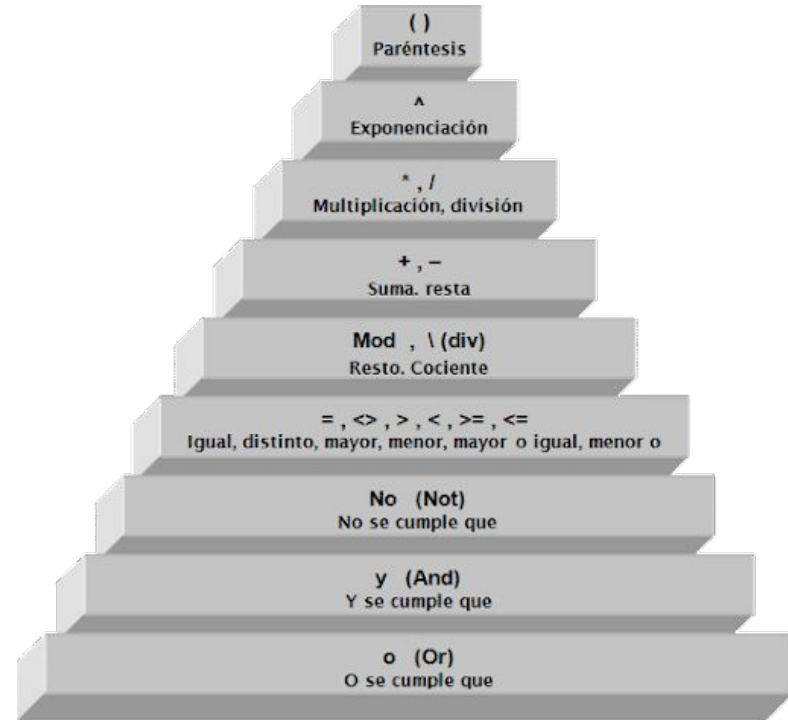
| Operador         | Descripción          | Equivale a              |
|------------------|----------------------|-------------------------|
| <code>=</code>   | <code>x = 3</code>   | <code>x = 3</code>      |
| <code>+=</code>  | <code>x += y</code>  | <code>x = x + y</code>  |
| <code>-=</code>  | <code>x -= y</code>  | <code>x = x - y</code>  |
| <code>*=</code>  | <code>x *= y</code>  | <code>x = x * y</code>  |
| <code>/=</code>  | <code>x /= y</code>  | <code>x = x / y</code>  |
| <code>%=</code>  | <code>x %= y</code>  | <code>x = x % y</code>  |
| <code>**=</code> | <code>x **= y</code> | <code>x = x ** y</code> |

# Jerarquía de los operadores

Al igual que ocurre en las matemáticas, los operadores en programación tienen un orden de prioridad.

Este orden es el siguiente (de menos prioritario a más prioritario): operadores booleanos; operadores de comparación, y finalmente los aritméticos (con el mismo orden de prioridad que en las matemáticas).

Este orden de prioridad se puede alterar con el uso de los paréntesis ().



# Estructuras de control

Las **estructuras de control de flujo**, son instrucciones que nos permiten evaluar si se puede cumplir una condición o no, o si debe ser evaluada **n** veces.

Los **condicionales** nos permiten evaluar si una condición cumple o no. Su sintaxis es muy sencilla: podemos evaluar si la condición es **verdadera** o **falsa**. Incluso podemos añadir opciones en el caso de que no se cumpla la primera condición y se deban evaluar más.

Además, existen otras estructuras de control, a las que se les suele denominar **ciclos, bucles** o **loops**. En ellos se evalúa una condición **n** veces hasta que ésta se cumpla. Son estructuras existentes en casi todos los lenguajes de programación, como los bucles **for** y **while**, entre otros.

# Estructuras de control

Para controlar el flujo de la ejecución estableciendo alternativas, es decir, que una serie de enunciados se ejecuten en algunas ocasiones y en otras no, existen las estructuras condicionales. En **JS** disponemos de las siguientes:

| Estructura de control | Descripción   |
|-----------------------|---|
| <b>If</b>             | <b>Condición simple:</b> Si ocurre algo, haz lo siguiente...                          |
| <b>If/else</b>        | <b>Condición con alternativa:</b> Si ocurre algo, haz esto, sino, haz lo esto otro... |
| <b>?:</b>             | <b>Operador ternario:</b> Equivalente a <b>If/else</b> , método abreviado.            |
| <b>Switch</b>         | <b>Estructura para casos específicos:</b> Similar a varios <b>If/else</b> anidados.   |

# Estructuras de control | Condicionales

Existe un orden para el desarrollo de un programa, y se lo conoce como “flujo del programa”.

Por defecto, el navegador leerá el script de forma secuencial, una línea luego de otra, desde arriba hacia abajo. Normalmente, la ejecución de la línea 5 nunca ocurrirá antes de la línea 3.

Al escribir un programa necesitamos establecer **condiciones** o **decisiones**, a partir de las cuales el navegador realiza una acción “A” si se cumple una condición o una acción “B” si no se cumple. Este es el primer tipo de estructura de control que analizaremos.

# Estructuras de control | If

El más conocido de estos mecanismos de estructura de control es el **if** (si condicional). Podemos indicar que se tome un camino sólo si se cumple la condición que establezcamos. Si no se cumple no se ejecuta nada y el programa sigue su curso:

```
var nota = 7;
console.log("Nota: ", nota);
// Condición (si nota es mayor o igual a 5)
if (nota >= 5) {
    console.log("¡Estoy aprobado!");
}
```

Cómo el valor de nota es superior a 5, nos aparece en la consola el mensaje «¡Estoy aprobado!». Sin embargo, si modificamos en la primera línea el valor de nota a un valor inferior a 5, no aparecerá el mensaje.

# Estructuras de control | If / else

Si utilizamos **if** seguido de un **else** podemos establecer una acción “A” si se cumple la condición, y una acción “B”.

Modificamos el ejemplo anterior para mostrar también un mensaje cuando estamos suspendidos, pero en este caso, en lugar de mostrar el mensaje directamente con **console.log** guardamos el texto en una nueva variable llamada **calificación**:

```
var nota = 7;  
console.log("El examen a resultado:");  
// Condición  
if (nota < 5) {  
  // Acción A (nota es menor que 5)  
  calificacion = "suspendido";  
} else {  
  // Acción B: (nota es mayor o igual que 5)  
  calificacion = "aprobado";  
}  
console.log("Estoy", calificacion);
```

# Estructuras de control | Operador ternario

El **operador ternario** es una alternativa de condicional **if/else** con una sintaxis más corta y, en muchos casos, más legible. Los dos scripts siguientes hacen lo mismo. El primero usa if/else, el segundo el operador ternario:

```
if (nota < 5) {  
  calificacion = "suspendido";  
} else {  
  calificacion = "aprobado";  
}  
console.log("Estoy", calificacion)  
;
```

```
// Operador ternario: (condición ? verdadero : falso)  
var calificacion = nota < 5 ? "suspendido" : "aprobado";  
console.log("Estoy", calificacion);
```



# Estructuras de control | If múltiple

Para analizar más de 2 condiciones podemos anidar varios **if/else** uno dentro de otro, de la siguiente forma:

```
var nota = 7;
console.log("He realizado mi examen.");
// Condición
if (nota < 5) {
  calificacion = "Insuficiente";
} else if (nota < 6) {
  calificación = "Suficiente";
} else if (nota < 8) {
  calificacion = "Bien";
} else {
  calificacion = "Sobresaliente";
}
console.log("He obtenido un", calificacion);
```

# Estructuras de control | Switch

La estructura de control **switch** permite definir casos específicos a realizar en el caso de que la variable expuesta como condición **sea igual** a los valores que se especifican a continuación mediante los **case**.

Con los **if** múltiples podemos controlar valores comprendidos en un rango. Con switch esto no es posible, ya que solo permite valores concretos y específicos.

Al final de cada caso es necesario indicar un **break** para salir del switch. Si no se hace esto, el programa pasa automáticamente al siguiente **case**, aunque no se cumpla la condición específica.

```
var nota = 7;
console.log("He realizado mi examen. Mi resultado es el siguiente:");
switch (nota) {
  case 10:
    calificacion = "Sobresaliente";
    break;
  case 9:
  case 8:
    calificacion = "Notable";
    break;
  case 7:
  case 6:
    calificacion = "Bien";
    break;
  case 5:
    calificacion = "Suficiente";
    break;
  case 4:
  case 3:
  case 2:
  case 1:
  case 0:
    calificacion = "Insuficiente";
    break;
  default:
    // Cualquier otro caso
    calificacion = "Nota errónea";
    break;
}
console.log("He obtenido un", calificacion);
```

# Estructuras de control | If con && (AND)

Podemos combinar el **if** con los operadores lógicos **&&** (AND) y **||** (OR) para describir condiciones más complejas.

Utilizando **&&** (AND) deben cumplirse todas las condiciones para que la proposición sea verdadera. Caso contrario, será falsa.

```
var altura = 0;
var edad = 0;
altura = parseFloat(prompt("Ingrese la altura"));
edad = parseInt(prompt("Ingrese la edad"));
if (altura > 1.30 && edad > 14) {
    console.log("Cumple con los requisitos");
} else{
    console.log("No cumple con los requisitos");
}
```

# Estructuras de control | If con || (OR)

Utilizando || (OR) basta con que se cumpla una de las condiciones para que la proposición sea verdadera. En caso de que todas las condiciones sean falsas, la proposición será falsa.

```
var color;  
color = prompt("Ingrese el color del auto");  
if (color == "Rojo" || color == "Verde") {  
    console.log("El auto pertenece a la categoría A");  
} else{  
    console.log("El auto pertenece a la categoría B");  
}
```

# Estructuras de control | Bucles e Iteraciones

Una de las principales ventajas de la programación es la posibilidad de crear bucles y repeticiones para tareas específicas, evitando realizarlas varias veces de forma manual. Existen muchas formas de realizar bucles, y analizaremos los más básicos, similares en otros lenguajes de programación:

| Tipo de bucle          | Descripción   |
|------------------------|---|
| <code>while</code>     | Bucles simples.   |
| <code>for</code>       | Bucles clásicos por excelencia.                             |
| <code>do..while</code> | Bucles simples que se realizan siempre como mínimo una vez. |

# Estructuras de control | Bucles e Iteraciones

Conceptos básicos sobre bucles:

- **Condición:** Al igual que en los **if**, en los bucles se va a **evaluar una condición** para saber si se debe repetir el bucle o finalizarlo. Generalmente, si la condición es **verdadera**, se repite. Si es **falsa**, se finaliza.
- **Iteración:** Se llama así a cada **repetición** de un bucle. Por ejemplo, si un bucle repite una acción 10 veces, se dice que realiza 10 iteraciones.
- **Contador:** Muchas veces, los bucles tienen una variable que se denomina **contador**, porque cuenta el número de repeticiones que ha hecho, hasta llegar a un número concreto y finalizar. Dicha variable debe ser inicializada (crearla y darle un valor) antes de comenzar el bucle.

# Estructuras de control | Bucles e Iteraciones

- **Incremento:** Cada vez que un bucle finaliza, se suele realizar el incremento (o decremento) de una variable, generalmente de la denominada variable **contador**.
- **Bucle infinito:** Se trata de la situación que tiene lugar cuando en un bucle no se modifica (incrementando o decrementando) la variable contador, o cuando escribimos una condición que nunca tiene lugar. En esos casos, el bucle se repite eternamente, sin que el flujo del programa pueda continuar. Cuando esto ocurre, se suele decir que “*el programa se queda colgado*”.

# Estructuras de control | While (mientras)

El bucle **while** se usa cuando el fin de la repetición de ciclos depende de una condición (\*). Analicemos el siguiente ejemplo y todas sus partes, para comprender qué ocurre en cada iteración del bucle:

```
i = 0; // Inicialización de la variable contador
// Condición: Mientras la variable contador sea menor de 5
while (i < 5) {
    console.log("Valor de i:", i);
    i = i + 1; // Incrementamos el valor de i
}
```

Valor de i: 0

Valor de i: 1

Valor de i: 2

Valor de i: 3

Valor de i: 4

(\*) Es muy importante que esa condición en un momento deje de ser verdadera, para evitar que ocurra un loop infinito.



# Estructuras de control | While (mientras)

En el ejemplo anterior:

- Antes de entrar en el bucle **while**, se inicializa la variable **i** a **0**.
- Antes de realizar la primera **iteración** del bucle, comprobamos la **condición**.
- Si la condición es **verdadera**, hacemos lo que está dentro del bucle.
- Mostramos por pantalla el valor de **i** y luego incrementamos el valor actual de **i en 1**.
- Volvemos al inicio del bucle para hacer una **nueva iteración**. Comprobamos de nuevo la **condición** del bucle.
- Cuando la condición sea **falsa**, salimos del bucle y continuamos el programa.

# Estructuras de control | While (mientras)

Detalle paso a paso de las iteraciones del ejemplo:

| Iteración del bucle    | Valor de i    | Descripción                                    | Incremento |
|------------------------|---------------|--|------------|
| <b>Antes del bucle</b> | i = undefined | Antes de comenzar el programa.                 |            |
| <b>Iteración #1</b>    | i = 0         | ¿(0 < 5)? Verdadero. Mostramos 0 por pantalla. | i = 0 + 1  |
| <b>Iteración #2</b>    | i = 1         | ¿(1 < 5)? Verdadero. Mostramos 1 por pantalla. | i = 1 + 1  |
| <b>Iteración #3</b>    | i = 2         | ¿(2 < 5)? Verdadero. Mostramos 2 por pantalla. | i = 2 + 1  |
| <b>Iteración #4</b>    | i = 3         | ¿(3 < 5)? Verdadero. Mostramos 3 por pantalla. | i = 3 + 1  |
| <b>Iteración #5</b>    | i = 4         | ¿(4 < 5)? Verdadero. Mostramos 4 por pantalla. | i = 4 + 1  |
| <b>Iteración #6</b>    | i = 5         | ¿(5 < 5)? Falso. Salimos del bucle.            |            |

# Estructuras de control | For (para)

La sintaxis de un bucle **for**, uno de los más usados, es más compacta y rápida de escribir que la de un bucle **while**. Requiere inicializar la variable, determinar la condición y definir el incremento al comienzo del bucle. Se suele usar cuando se conoce de antemano cuantas repeticiones se tienen que hacer.

**Ejemplo:** Mostrar por pantalla los números enteros del 1 a 10.

```
for (var i=1; i<=10; i++) {  
    console.log(i);  
}
```

**Ejemplo:** Mostrar por pantalla los múltiplos de 2 hasta 100.

```
for (var i=2; i<=100; i+=2) {  
    console.log(i);  
}
```

# Estructuras de control | For (para)

El **bucle for** es uno de los más utilizados en la programación. Veamos el ejemplo anterior utilizando este bucle:

```
// for (inicialización; condición; incremento)
for (i = 0; i < 5; i++) {
  console.log("Valor de i:", i);
}
```

Valor de i: 0

Valor de i: 1

Valor de i: 2

Valor de i: 3

Valor de i: 4

En programación es muy habitual empezar a contar desde cero. Mientras que habitualmente contamos de 1 a 10, en programación de 10 elementos se cuentan de 0 a 9.

# Material extra

# Artículos de interés

- [Expresiones y operadores en JavaScript](#), incluyendo los de asignación, comparación, aritméticos, bit a bit, lógicos, ternarios, de cadena y otros.
- [Tomando decisiones en tu código - condicionales](#), en developer.mozilla.org
- [¿Cómo utilizar bucles en JavaScript?](#)
- [Bucle For](#) en W3Schools.com
- [Bucle While](#) en W3Schools.com

# Actividades prácticas

- Del archivo “**Actividad Práctica - JavaScript Unidad 1**” están en condiciones de hacer los ejercicios: 6 a 25

# No te olvides de dar el presente



# **Recordá:**

- **Revisar la Cartelera de Novedades.**
- **Hacer tus consultas en el Foro.**
- **Realizar los Ejercicios obligatorios.**

**Todo en el Aula Virtual.**

**Muchas gracias por tu atención.**

**Nos vemos pronto**