Buenos Aires aprende
Agencia de Habilidades para el Futuro

Talento Tech

Fron-End JS

Clase 10 - “Condicionales y ciclos”

Buenos Aires Ciudad

¡Les damos la bienvenida!

 Vamos a comenzar a grabar la clase

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Talento Tech

Clase 09.

JS 1 - Introducción a JavaScript

1. ¿Qué es y para qué se usa JavaScript?
2. Conceptos generales. Sintaxis básica
3. Variable: ¿qué es y cómo declararla? Tipos
4. Asignación y cambio del valor
5. Operadores aritméticos
6. Conversión a entero y flotante

Clase 10.

JS 2 - Condicionales y ciclos

1. Diagrama de flujo
2. Condicional: ¿Qué es?
3. Operadores lógicos y de comparación: ¿Qué son y cuál es su uso en los condicionales?
4. Bucles: ¿Qué son? Tipos y diferencias entre sí
5. Cómo combinar operadores lógicos y ciclos

Clase 11.

JS 3 - Programación modular con funciones

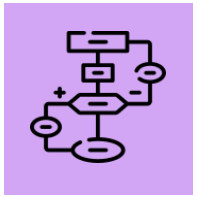
1. Funciones: ¿Que son? Parámetros de entrada y de salida
2. Scope global y local
3. Programación modular vs. Funciones
4. Ejercitación de funciones
5. Parámetros.
6. Funciones nativas.

Buenos Aires aprende
Agencia de Habilidades para el Futuro
Buenos Aires Ciudad

s condicionales?
é son? Tipos y
diferencias entre sí
10. Cómo combinar operadores

Talento Tech

JavaScript



Diagramas de flujo

Diagramas de flujo

¿Qué es un diagrama de flujo?

Un diagrama de flujo es una representación visual de los pasos secuenciales que se siguen para resolver un problema o realizar una tarea. Se utiliza mucho en programación y en otros procesos para ilustrar cómo se avanza de un paso al siguiente.

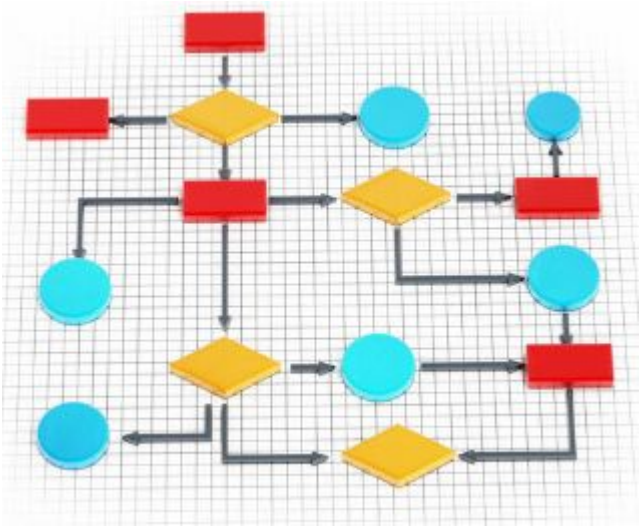


Diagrama de flujo para preparar un café

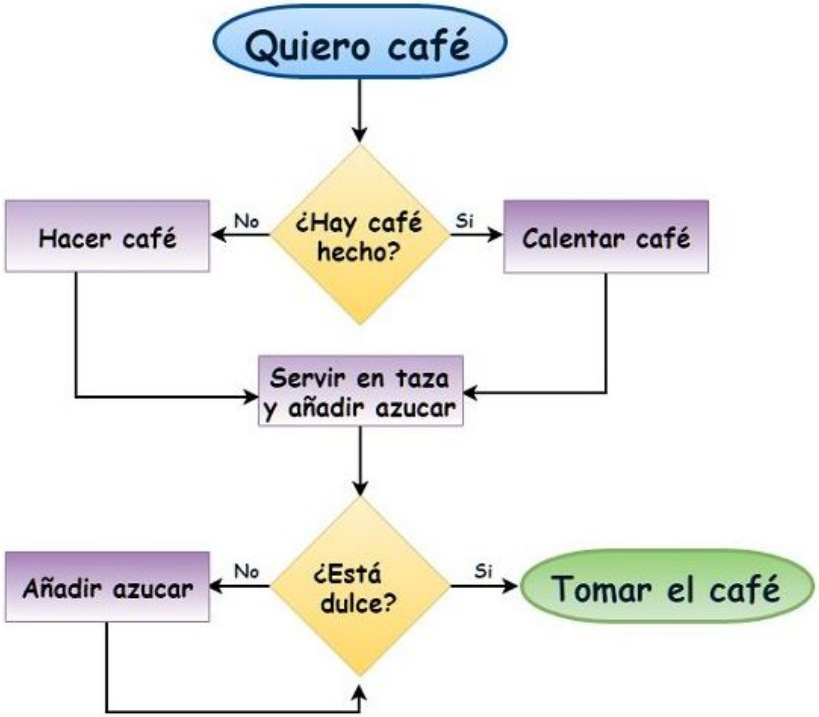


Diagrama de flujo

Elementos claves:

- **Inicio/Fin:** Representan el comienzo o la conclusión del proceso. Se muestra en forma de óvalo.
- **Acción/Proceso:** Indican una operación o tarea que se debe realizar, representada por un rectángulo.
- **Decisión:** Usado para mostrar una bifurcación o decisión en el proceso (sí o no). Se representa con un rombo.
- **Flechas:** Indican la dirección en la que fluye el proceso.

¿Para qué sirve?

- Ayuda a visualizar claramente el proceso.
- Facilita la identificación de posibles errores o puntos de mejora.
- Es una herramienta útil para planificar y organizar la lógica antes de escribir código.

Estructuras condicionales

Condicional ¿Qué es?

¿Qué es un Condicional?

Un condicional es una estructura de control en programación que permite ejecutar diferentes bloques de código dependiendo de si una condición es verdadera o falsa. Es como tomar decisiones: “si pasa esto, entonces hacé esto, si no, hacé otra cosa.”

¿Para qué sirve?

- Permite a los programas tomar decisiones según diferentes escenarios.
- Hace que el código sea flexible y capaz de reaccionar a distintos valores de entrada.

Tipos de condicionales

- **if**: Se usa para ejecutar código sólo si una condición es verdadera.

```
    if (edad >= 18) {
  console.log("Sos mayor de edad.");
}
```

- **if...else**: Permite ejecutar un bloque de código si la condición es verdadera, y otro bloque si es falsa.

```
if (edad >= 18) {
   console.log("Sos mayor de edad.");
} else {
   console.log("Sos menor de edad.");
}
```

Tipos de condicionales

- **else if**: Se usa para manejar múltiples condiciones.

```
    if (edad >= 18) {
  console.log("Sos mayor de edad.");
} else if (edad >= 13) {
  console.log("Sos un adolescente.");
} else {
  console.log("Sos un niño.");
}
```

Operadores lógicos y relacionales

Buenos Aires aprende
Agencia de Habilidades para el Futuro

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

Buenos Aires Ciudad

Talento Tech

Operadores lógicos

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

Operador	Descripción
&&	Y lógico (Conjunción)
||	O lógico (Disyunción)
!	NO lógico (Negación)

&& (Conjunción)

Prop A	Prop B	Resultado
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

! (Negación)

Prop A	Resultado
!True	**False**
!False	**True**

|| (Disyunción)

Prop A	Prop B	Resultado
True	True	**True**
True	False	**True**
False	True	**True**
False	False	**False**

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
i++	incremento posfijo	a=i++ primero a=i y después i=i +1
++ i	incremento prefijo	a=++i primero i=i +1 y después a=i
i - -	decremento posfijo	a=i - - primero a=i y después i=i – 1
- - i	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:

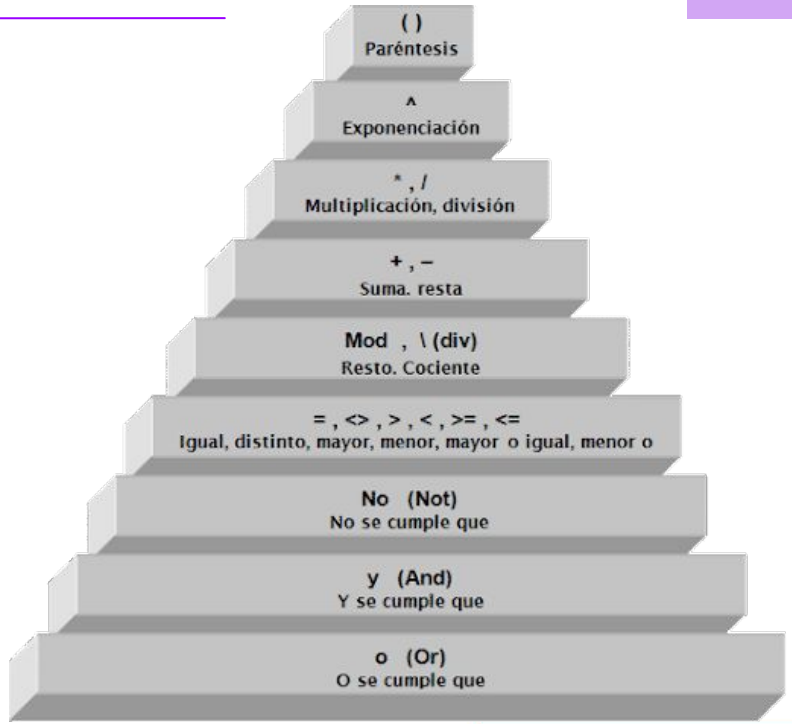
Operador	Descripción	Equivale a
=	x = 3	x = 3
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

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

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
`If`	**Condición simple**: Si ocurre algo, haz lo siguiente...
`If/else`	**Condición con alternativa**: Si ocurre algo, haz esto, sino, haz lo esto otro...
`?:`	**Operador ternario**: Equivalente a `If/else`, método abreviado.
`Switch`	**Estructura para casos específicos**: Similar a varios `If/else` 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.

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.

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 ha 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);
```

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);
```

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);
```

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");
```

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");
}
```

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);
```

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
`while`	Bucles simples.
`for`	Bucles clásicos por excelencia.
`do..while`	Bucles simples que se realizan siempre como mínimo una vez.

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”.

While

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.

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.

While

Detalle paso a paso de las iteraciones del ejemplo:

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

For

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);
}
```

For

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

¡Vamos a la práctica! 

Ejercicios prácticos

Optativos | No entregables

Evaluación de condicionales y operadores lógicos

Crear un programa que reciba la edad de una persona y si es miembro VIP. El programa deberá verificar lo siguiente:

1. Si la persona tiene 18 años o más, permitirle el acceso al evento.
2. Si además de tener 18 años o más, es miembro VIP, darle acceso al área exclusiva.
3. Si la persona tiene menos de 18 años, denegar el acceso.

Tips:

- **Validación de entradas:** Asegurate de que el usuario ingrese una edad válida. Podés usar isNaN() para verificar que el valor ingresado sea un número.
- **Uso de operadores lógicos:** Combiná el operador && para verificar que se cumplan ambas condiciones (edad >= 18 y ser miembro VIP).
- **Operador ternario:** Considerá usar un operador ternario si la lógica es sencilla.
- **Consola del navegador:** Mostrá los resultados usando console.log() para verificar si se cumplen las condiciones correctamente.

Ejercicios prácticos

Optativos | No entregables

Iterar sobre una lista de productos con bucles y condicionales

Crear un programa que reciba una lista de productos, cada uno con un precio y un indicador de descuento (true o false).
El programa debe iterar sobre la lista y:

1. Mostrar todos los productos con descuento.
2. Mostrar el total de productos sin descuento.
3. Al final, mostrar cuántos productos tienen descuento y cuántos no.

Tips:

- **Uso de bucles:** Utilizá un for para recorrer la lista de productos.
- **Condicionales:** Dentro del bucle, utilizá if para verificar si el producto tiene descuento (producto.descuento === true).
- **Validación:** Podés agregar validaciones para asegurarte de que los datos de los productos sean correctos (nombres no vacíos, precios válidos).
- **Consola del navegador:** Mostrá los resultados de los productos con descuento y el total de productos en la consola usando console.log().

¡NUEVO CUESTIONARIO EN CAMPUS!

La resolución del cuestionario es de carácter obligatorio para poder avanzar en la cursada.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

BA Buenos Aires Ciudad

Talento Tech