

Agencia de
Aprendizaje
a lo largo
de la vida

Desarrollo Fullstack



Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 12

Javascript

- ¿Qué es Javascript?
- Variables
- Tipos de datos
- Prompt, alert y console

Clase 13

Javascript

- Operadores Aritméticos
- Operadores Relacionales
- Operadores Lógicos
- Condicionales
- Bucles

Clase 14

Javascript

- Funciones
- Expresadas
- Declaradas
- Arrow
- Callbacks
- Scopes

JS JAVASCRIPT

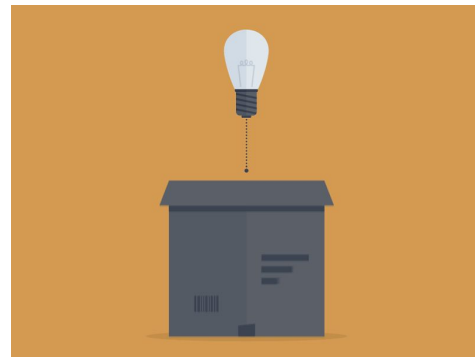
Lógica a su servicio

En el capítulo anterior...

Variable y declaración de variables

Variables

- **Una variable** en programación es utilizada de forma habitual.
- Se trata de un elemento fundamental para **gestionar la información** que se decida incorporar en el código y básicamente **funciona para guardar datos**.
- **Una variable almacena un valor único.**



Características de una variable en programación

1. Una variable **reserva espacio de memoria.**
2. Una variable debe ser **declarada para que exista.**
3. Una variable tiene un **alcance global** es decir puede ser utilizada en todo el proyecto o **local** (ámbito de bloque).



Declaración de variables: **var**

1. **Ámbito de función:** Las variables declaradas con var tienen un ámbito de función, lo que significa que son accesibles en toda la función en la que se declaran, independientemente de bloques de código. Puede ser asimilada a una variable global. **Actualmente están en desuso.**
2. **Hoisting (elevar):** Las declaraciones var son "elevadas" al principio de su contexto de ejecución. Esto significa que puedes usar la variable antes de su declaración sin generar un error. Sin embargo, el valor de la variable será undefined hasta que se le asigne un valor.



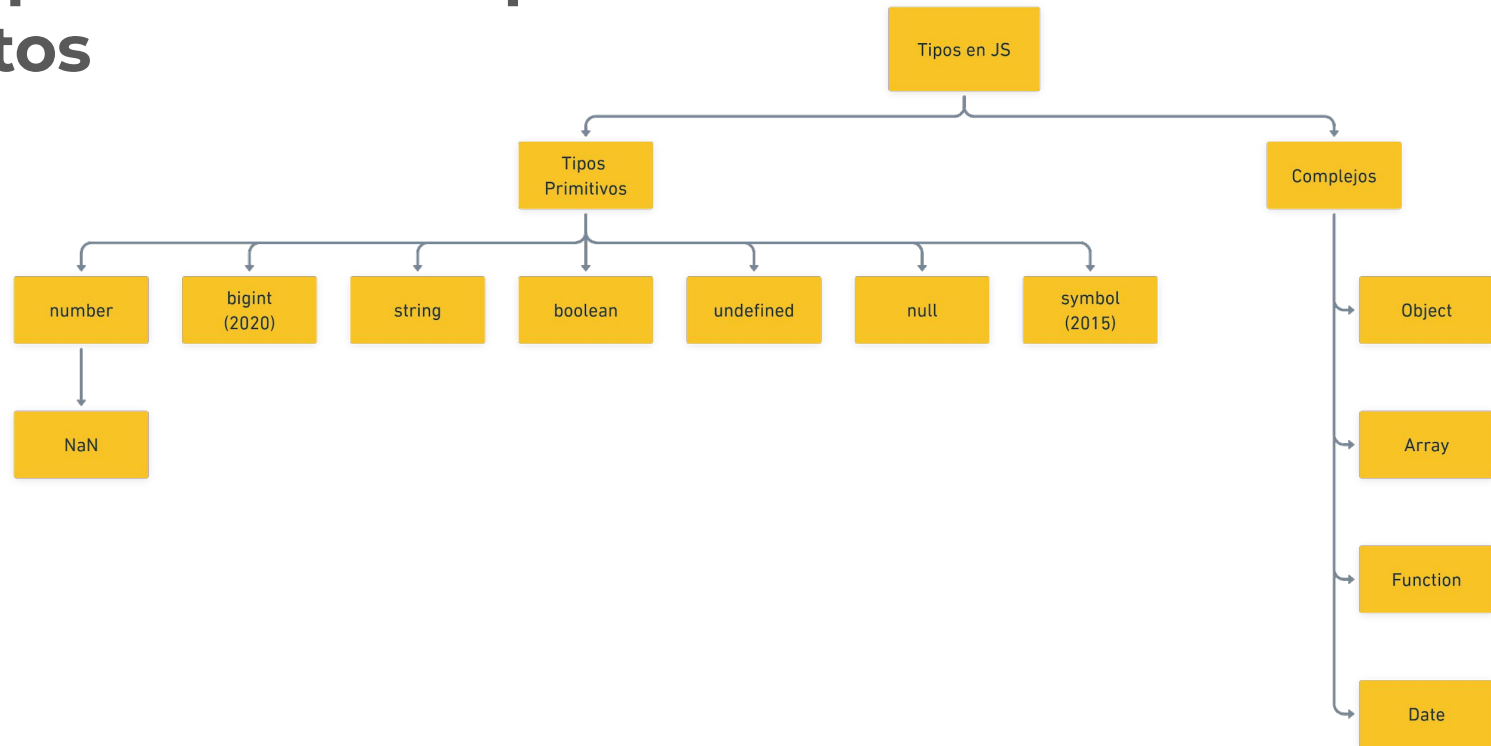
Declaración de variables: **let**

1. **Ámbito de bloque:** Las variables declaradas con `let` tienen un ámbito de bloque, lo que significa que son válidas solo dentro del bloque en el que se declaran. De esta manera existe mayor control sobre ellas.
2. **No hoisting:** Las variables declaradas con `let` **no son "elevadas"** al principio de su contexto de ejecución. **Deben ser declaradas antes de ser utilizadas.** Lo cual otorga orden al código.



Tipos de datos

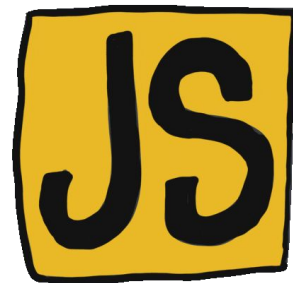
Esquema de los tipos de datos



Instrucciones de entrada - salida

Instrucción de entrada

- Una **instrucción de entrada** nos permite interactuar con la aplicación.
- En JavaScript, las instrucciones de entrada típicamente se realizan para **obtener datos del usuario o del entorno** en el que se ejecuta el código.
- Estos datos **serán almacenados en una variable**.
- Posteriormente estos datos **serán procesados para producir un resultado**.

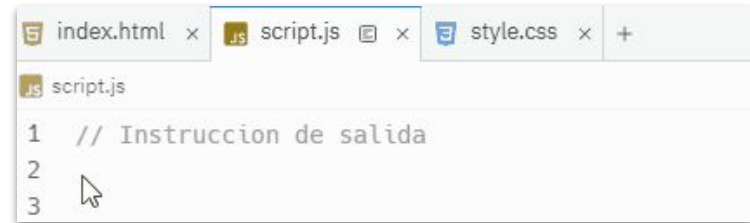


Instrucción de salida alert()

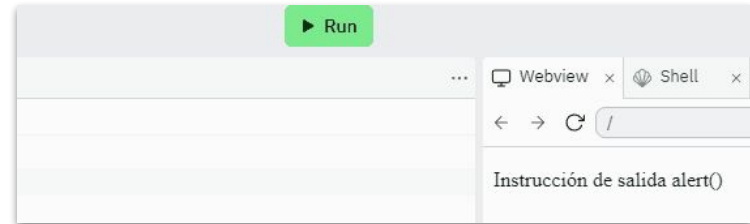
alert(): despliega un mensaje en la ventana del navegador con el texto que reciba por parámetro.

La sintaxis **en JS** es la siguiente:

```
alert("Mensaje_de_alerta");
```



```
index.html x script.js x style.css x +
script.js
1 // Instruccion de salida
2
3
```

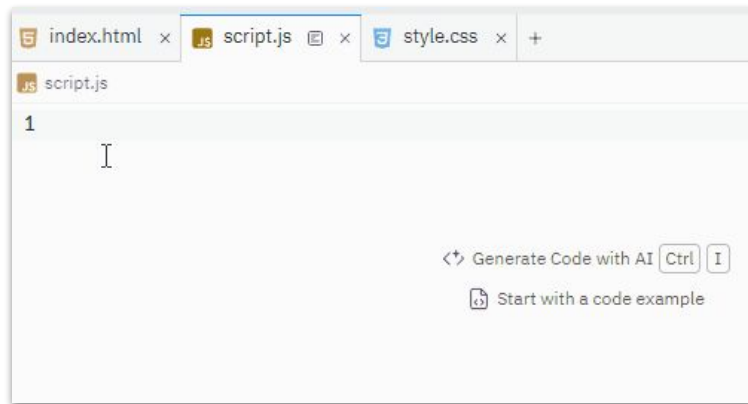


Instrucción de salida console.log()

console.log(): imprime mensajes o lo que se le pase por parámetro, en la consola del navegador.

La sintaxis **en JS** es la siguiente:

```
console.log(<mensaje_expresion>);
```



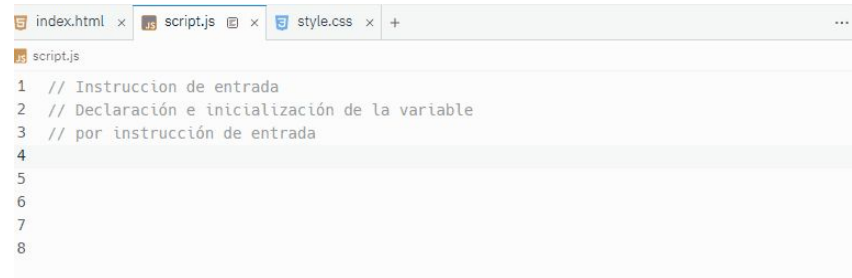
Instrucción de entrada: prompt()

prompt(): función que muestra un cuadro de diálogo que solicita al usuario que ingrese información. Retorna el valor ingresado como una cadena (string).

La sintaxis **en JS** es la siguiente:

```
let <identificador> =  
    prompt("Mensaje");
```

- **let** declara una variable.
- **<identificador>** nombre de la variable.



```
index.html x script.js x style.css x +  
script.js  
1 // Instruccion de entrada  
2 // Declaración e inicialización de la variable  
3 // por instrucción de entrada  
4  
5  
6  
7  
8
```

Operadores

Operadores

- Los **operadores** nos ayudan a **modificar, reasignar y comprobar** el valor de las variables con una **sintaxis más sencilla** y acotada.



Operadores de asignación

Inicialización de variables

Operador de asignación

- Cuando las variables **no están inicializadas**, su valor es indefinido (*undefined*).
- Para iniciarlas en un valor, se utiliza el **operador de asignación** en **JS** es el símbolo **=**

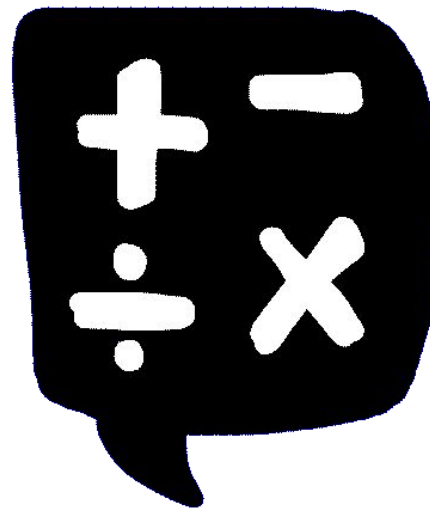


Operadores Aritméticos

Operaciones matemáticas

Operadores aritméticos

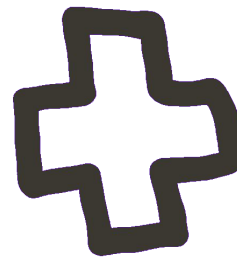
Consta de los operadores fundamentales de la aritmética con el agregado de la potencia y el módulo o residuo, que devuelve el resto entero que se produce al realizar un cociente entre dos números enteros.



Operador	Nombre	Ejemplo	Resultado	Descripción
+	Suma	$12 + 3$	15	Devuelve la suma de dos expresiones.
-	Resta	$12 - 3$	9	Devuelve la resta de dos expresiones.
*	Multiplicación	$12 * 3$	36	Devuelve la resta de dos expresiones.
/	División	$12 / 3$	4	Devuelve el cociente de dos expresiones.
**	Potenciación	$12 ** 3$	1728	Devuelve la potencia entre dos expresiones.
%	Módulo o Residuo	$12 \% 3$	0	Devuelve el resto del cociente entre dos enteros.

Operador de concatenación

- Para hacer más atractivas las salidas, se utiliza el operador de concatenación, en Js existe dos:
 - **El signo mas “+” en general el mas utilizado.**
 - **La coma “,”.**
- Este permite unir cadenas de textos unas con otras o con contenidos de variables **para una mejor lectura en la salida de datos.**
- La **coma aporta un espacio** entre elementos concatenados, **el signo + no lo hace.**
- En ocasiones, por cuestiones de legibilidad del código, será más recomendable la utilización de uno sobre otro.



Algoritmos

Algoritmos

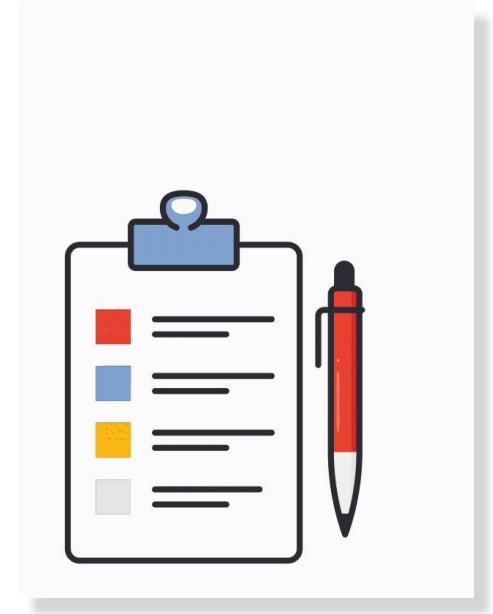
La **importancia de los algoritmos en la informática** es trascendental, dado que **la programación tiene como objetivo la implementación de ellos en una computadora**, para que sea ésta quien los **ejecute y resuelva determinado problema**, sin embargo, éstos trascienden la disciplina informática, pudiendo encontrarlos en la matemática o la lógica.

Prof. Carlos Cimino

Características de los algoritmos

Las características que debe cumplir un algoritmo son:

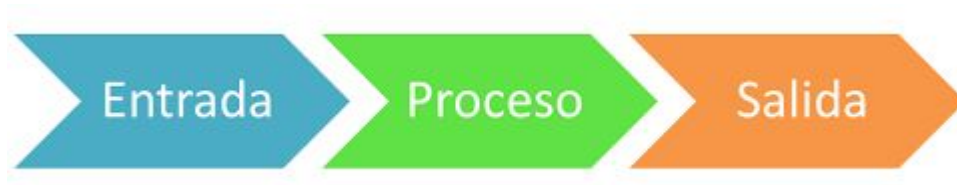
- **Precisos:** sin ambigüedades, no dejar algo al azar.
- **Ordenados:** presentan una **secuencia de pasos** para llegar a la solución.
- **Finitos:** tienen **principio y un fin**; un número determinado de pasos.



Algoritmo de una suma

1. Inicio del algoritmo
2. Necesito un número a
3. Necesito otro número b
4. Sumo $a+b$
5. Obtengo el resultado
6. Fin del algoritmo

Componentes del Algoritmo



- **Entrada:** Información que damos al algoritmo con la que va a trabajar para ofrecer la solución esperada.
- **Proceso:** Conjunto de pasos para que, a partir de los datos de entrada, llegue a la solución del problema.
- **Salida:** Resultados, a partir de la transformación de los valores de entrada durante el proceso.

Desafío II de clase

Utilizamos whimsical para realizar el diagrama de flujo:

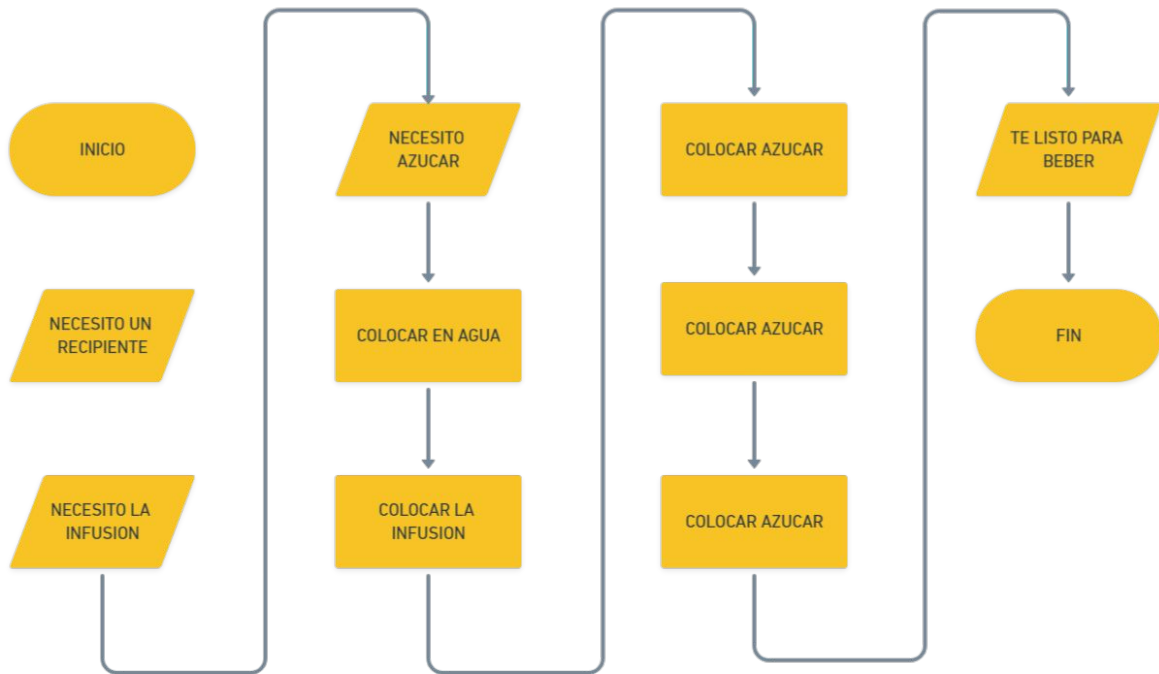
Objetivo:

- Preparar una taza de té para obtener un té listo para tomar.

Preparar una taza de té

1. Inicio;
2. Necesito una taza;
3. Necesito agua;
4. Necesito te;
5. Necesito una cucharita;
6. Coloco agua caliente;
7. Selecciono el tipo de infusión;
8. Coloco la infusión en el agua;
9. Coloco una cucharadita de azúcar;
10. Coloco una cucharadita de azúcar;
11. Coloco una cucharadita de azúcar;
12. Espero 5 minutos;
13. Obtengo el té listo para beber;
14. Fin;

Ejemplo final: Algoritmo para preparar té

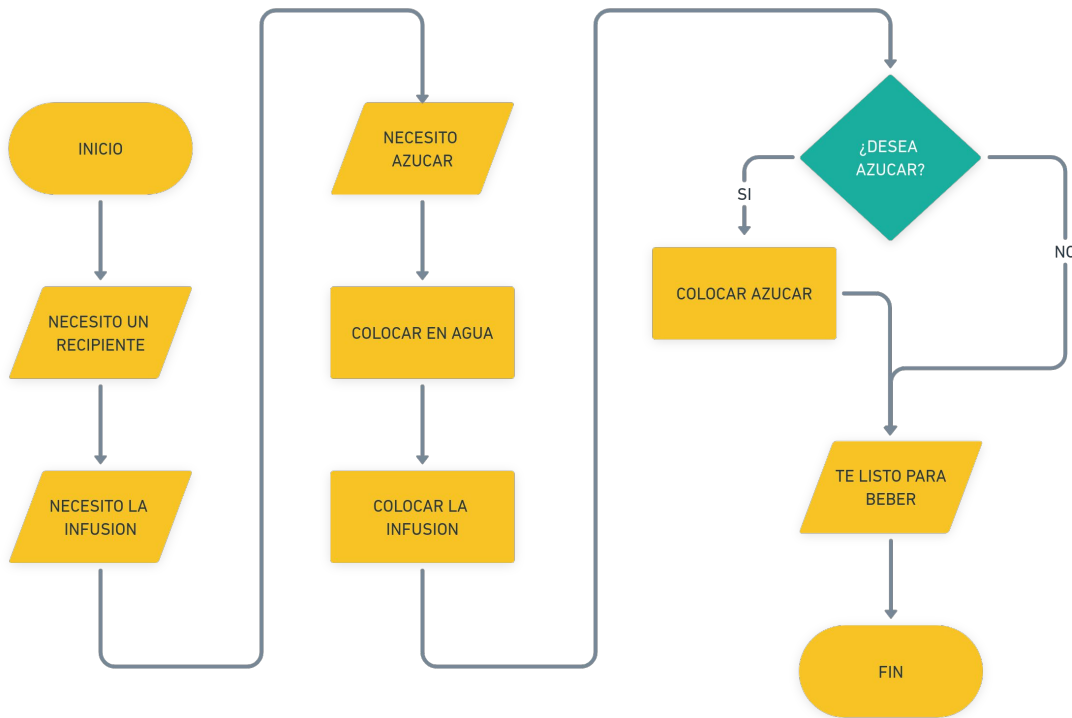


Made with Whimsical

Estructuras Condicionales

Flujos de control y de selección

Ejemplo final: Algoritmo para preparar té



Estructuras de control, flujos de selección

- **Estas estructuras** básicamente, **nos permiten decidir un camino a tomar** de acuerdo a una evaluación booleana o lógica y ejecutar, en consecuencia, un bloque de código.

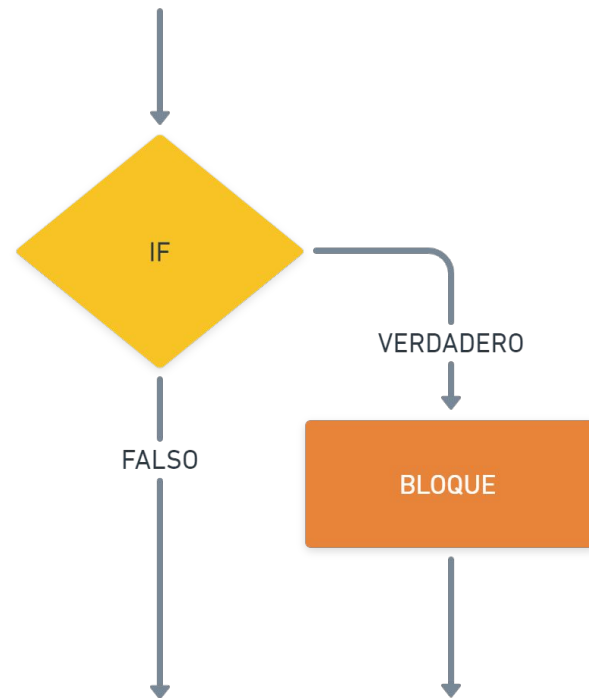
Elas son:

- Flujo **IF (Si)**
 - Flujo **IF - ELSE (Si-Sino)**
 - Flujo **SWITCH (Segun-Hacer-Caso)**
- Estas estructuras, desde ya, pueden trabajar anidadas y combinadas, esto dependerá del análisis previo del problema y de **la decisión del programador de tomar tal o cual camino.**

Estructura if

Estructura if

- La estructura if se denomina **estructura de selección única** porque ejecuta un bloque de sentencias **solo cuando se cumple la condición** del if.
- Si la condición es **verdadera se ejecuta el bloque de sentencias**.
- Si la condición es **falsa**, el flujo del programa **continúa en la sentencia inmediatamente posterior al if**.



Sintaxis

```
if (condicion) {  
  bloque-de-sentencias  
}
```

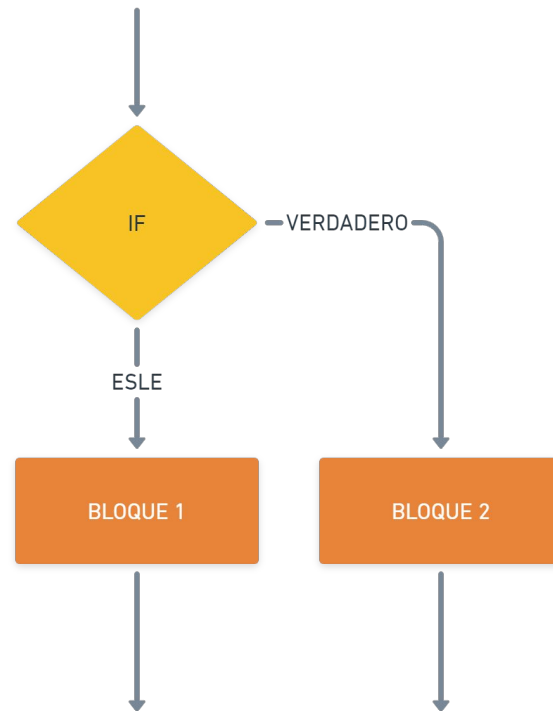
Ejemplo

```
if (calificacion == 10) {  
  console.log("Matrícula de Honor");  
}
```

Estructura if-else

Estructura if-else

- La **estructura if-else** se denomina **de selección doble** porque selecciona entre dos bloques de sentencias mutuamente excluyentes.
- **Si se cumple la condición**, se ejecuta el bloque de sentencias asociado al if.
- **Si la condición no se cumple**, entonces se ejecuta el bloque de sentencias asociado al else.



Sintaxis

Una sentencia if-else tiene la siguiente sintaxis:

```
if (condicion) {  
bloque-de-sentencias-if  
} else {  
bloque-de-sentencias-else  
}
```

Desafío I. Evaluar el número como par o impar

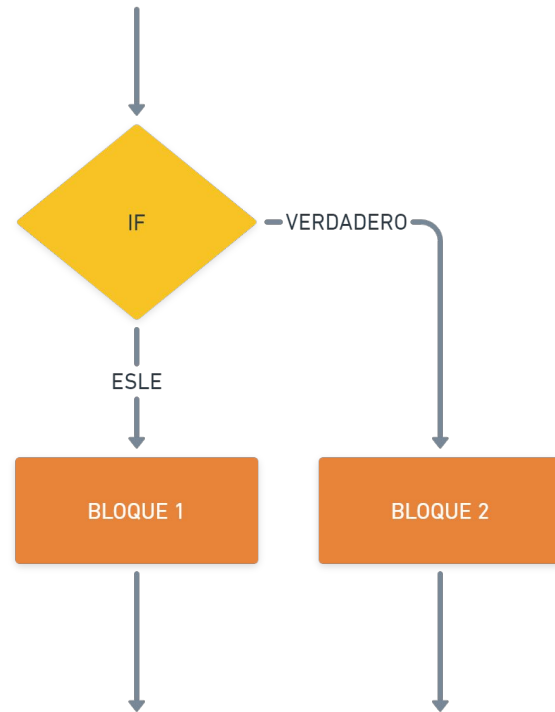
```
if (numero % 2 == 0) {  
  console.log("El número es par");  
}else{  
  console.log("El número es impar");  
}
```

Estructura if-else

Operador condicional

Operador condicional o ternario

- El operador condicional (`_ ? _ : _`) se relaciona con la estructura if-else.
- Es el único operador de Java que utiliza tres operandos.
- El primer operando es una condición lógica, el segundo es el valor que toma la expresión cuando la condición es true y el tercero es el valor que toma la expresión cuando la condición es false.



Sintaxis

El operador ternario tiene la siguiente sintaxis:

```
variable = condicion_logica ? valor_si_true : valor_si_false;
```

//Ejemplo quiero evaluar la mayoría de edad

```
int edad = 16;
```

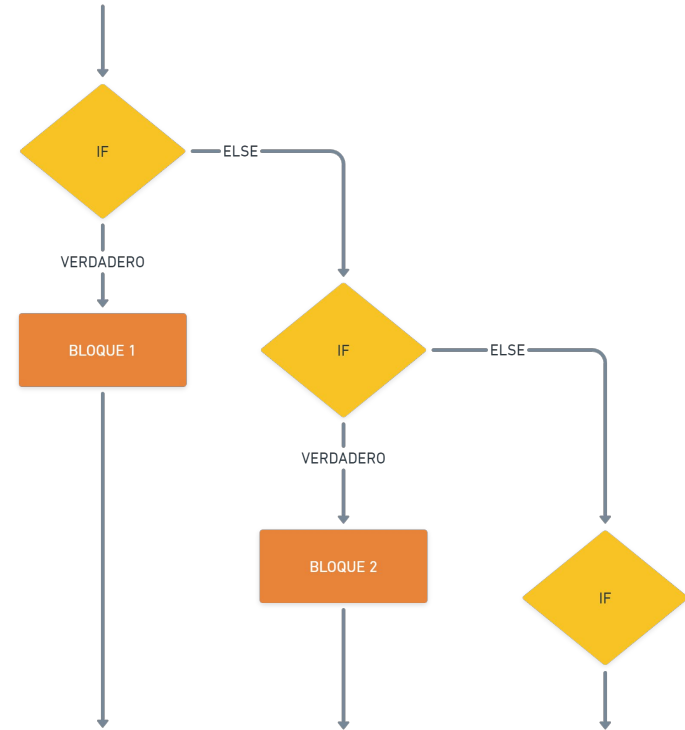
```
let txt = (edad >= 18) ? "Mayor de edad" : "Menor de edad";
```

```
console.log(txt);
```

Estructura if - else if - else

Estructura if else - if

- La estructura **if-else-if** representa un **if-else anidado**, es decir permite escribir de forma abreviada las condiciones de un if-else anidado.
- En el caso de esta estructura, los bloques de sentencias van encerrados entre corchetes { }.



Sintaxis

Una sentencia **if-else** tiene la siguiente sintaxis:

```
if (condicion-1) {  
bloque-de-sentencias-condicion-1  
} else if (condicion-2) {  
bloque-de-sentencias-condicion-2  
} else {  
bloque-de-sentencias-else  
}
```

Repo de clases

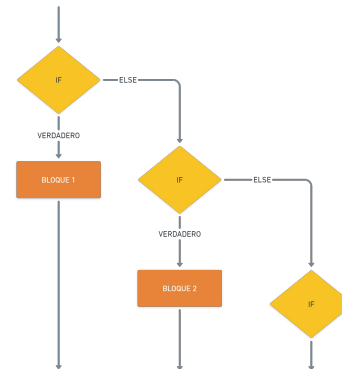
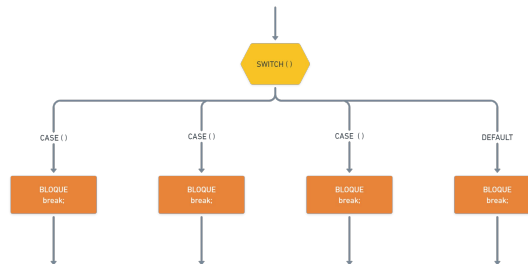
realizar un repo de clases

Estructuras de selección en JS

Switch

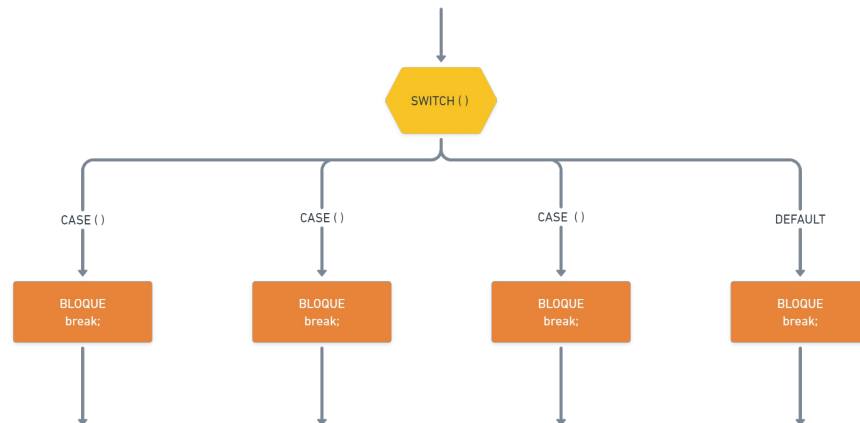
Estructura switch

- La estructura switch es una estructura de selección múltiple que permite seleccionar un bloque de sentencias entre varios casos.
- Es **parecido a una estructura de if-else anidados**.



Estructura switch

- La diferencia está en que la **selección del bloque de sentencias depende de la evaluación de una expresión** que se compara por igualdad con cada uno de los casos.
- La estructura switch consta de una expresión y una serie de **etiquetas case y una opción default**.
- **La sentencia break** indica el final de la ejecución del switch.

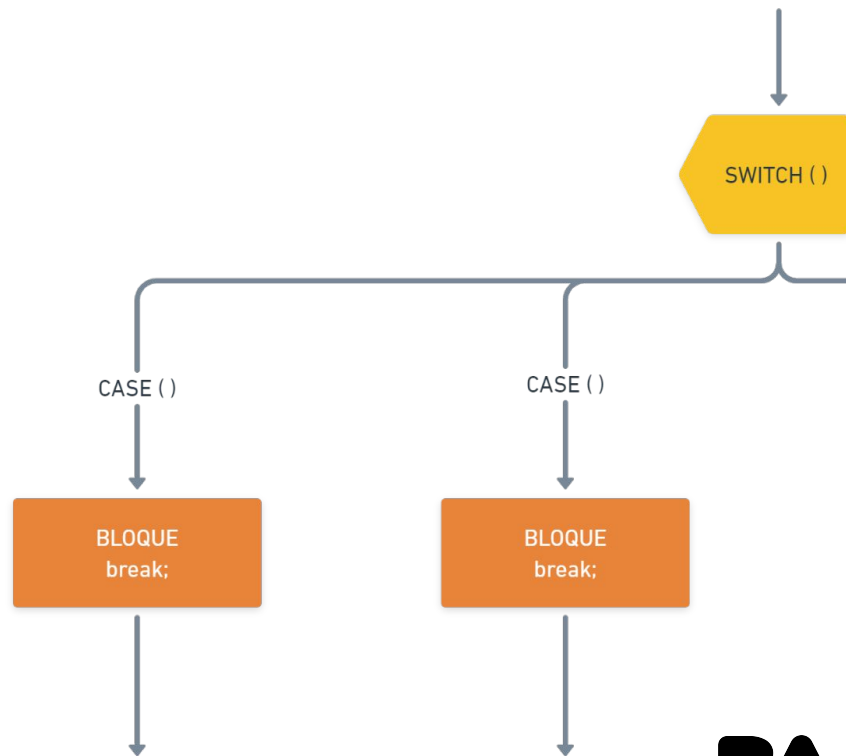


Sintaxis

```
switch (expresion) {  
  case valor-1:  
    bloque-de-sentencias-1;  
    break;  
  case valor-2:  
    bloque-de-sentencias-2;  
    break;  
  case valor-3:  
    bloque-de-sentencias-3;  
    break;  
  case valor-4:  
    bloque-de-sentencias-4;  
    break;  
  default:  
    bloque-de-sentencias-default;  
    break;  
}
```

Características del switch

- La condición de acceso al switch **debe devolver un valor de tipo entero (int) o caracter (char)** y es obligatorio que la expresión se escriba entre paréntesis.
- A continuación de cada **case** aparece uno o más **valores constantes del mismo tipo** que el valor de la expresión del switch.
- Para **interrumpir** la ejecución de las sentencias del switch **se utiliza la sentencia break** que provoca la finalización del switch y evita las caídas fall-through.



Instrucción break;

- Para asegurar el correcto flujo de ejecución de un programa durante la evaluación de una sentencia switch, **es recomendable incluir una sentencia break** al final del bloque de instrucciones de cada case, incluido el correspondiente a la etiqueta default.
- **Esto es importante**, porque si se omite la sentencia break, cuando finaliza la ejecución del bloque de sentencias de un case, **el flujo del programa continúa ejecutando los case siguientes** y esto **puede provocar un comportamiento erróneo del programa**.
- La instrucción break también **se la conoce como estructura de salto** (por el comportamiento que provoca en el código) y lo veremos en las siguientes clases.



Buenas prácticas

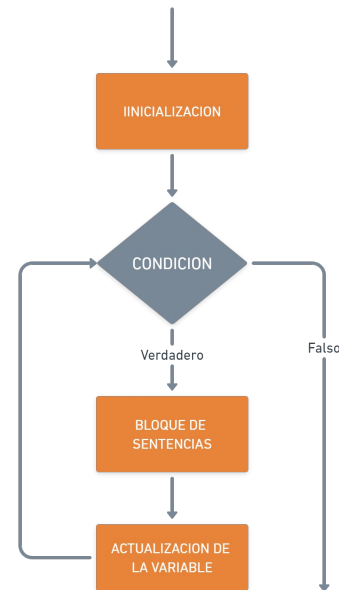
- Los valores duplicados de los **case o Casos** no están permitidos.
- Se pueden agrupar varios case para un solo bloque de código.
- El valor para un **case o Caso** debe ser del mismo **tipo** de datos que la variable en el switch o **Segun**.
- El valor para un **case o Caso** debe ser una **constante o un literal es decir una palabra**. Las **variables no están permitidas**.
- La instrucción **default o De Otro Modo** debe aparecer al final del **switch, para no dar un cierre abrupto al programa**.

Flujos de Repetición.

Estructuras basadas en condición

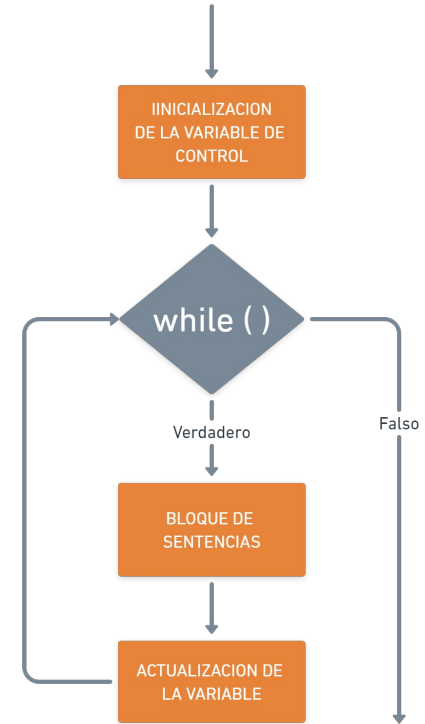
Composición de la estructura de repetición

- Las estructuras de repetición se componen de **cuatro** partes:
 - La inicialización,
 - La condición,
 - El bloque de sentencias
 - La actualización.



Estructura while

- Un while se ejecuta cero o más veces.
- La estructura de repetición **while repite el bloque** de sentencias mientras **la condición del while es verdadera**.
- El diagrama de flujo de una estructura while muestra que **la condición se verifica justo después de inicializar la variable de control**.
- Si el resultado de evaluar **la condición es verdadero**, entonces **se ejecuta el bloque de sentencias y actualiza la variable hasta que la condición ya no se cumpla**.
- Cuando **la condición no se cumpla**, el flujo continúa con la ejecución de los **bloques fuera del bucle**.



Sintaxis

Un while tiene la siguiente sintaxis:

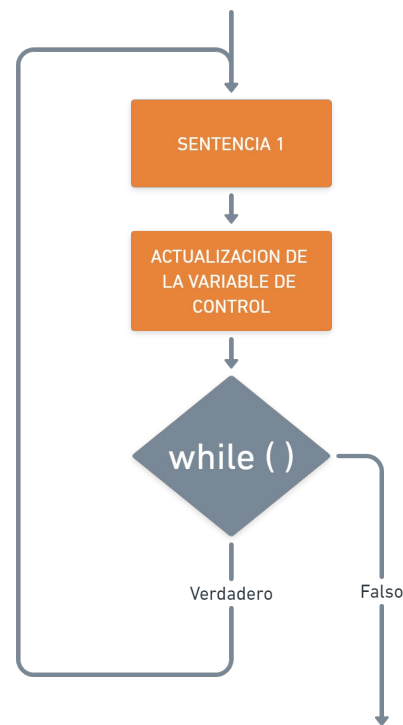
```
while (condición) {  
bloque-de-sentencias;  
actualizacion;  
}
```

Ejemplo de clase

```
/*  
* Ejemplo de clase bucle while  
* Contamos del 0 al 100  
* */  
  
//Inicialización de la variable  
let c = 0;  
//Bucle while  
while (c < 101){  
  //Bloque de sentencia  
  console.log(c);  
  //Actualización de la variable  
  c++;  
}
```

Estructura do-while

- La estructura de repetición do-while **ejecuta el bloque de sentencias al menos una vez.**
- **Después comprueba la condición y repite** el bloque de sentencias mientras la **condición es verdadera.**
- Cuando la **condición** resulte **falsa finalizará el bucle.**



Sintaxis

Un **do-while** tiene la siguiente sintaxis:

```
do {  
bloque-de-sentencias;  
actualizacion;  
} while (condición);
```


Ejemplo de clase

```
/*Realizaremos el factorial del número 5  
* Recordemos que  $5! = 5*4*3*2*1$ ;  
*/
```

```
//Declaración de variables
```

```
let n = 5; //El número a hacer factorial  
let f = 1; //El acumulador f lo inicio en 1  
let i = 1; //El conador inicia en 1
```

```
//Bloque del bucle do-while
```

```
do{  
  f *= i;  
  i++;  
}while(i<=5);
```

```
console.log("El factorial de "+n+" es " +f);
```

El bucle for

Estructura basada en cantidad definida de repeticiones

Estructura for

- La estructura de repetición **for repite el bloque de sentencias** mientras la **condición del for es verdadera**.
- Un **for** es un **caso particular** de la estructura **while**.
- **Solo se debe utilizar** cuando **se sabe el número de veces que se debe repetir** el bloque de sentencias.



Sintaxis

Un **for** tiene la siguiente sintaxis:

```
for (inicialización; condición; actualización){  
// bloque-de-sentencias;  
}
```

Ejemplo de clase

```
/*Realizaremos el factorial del número 5 recorriendo con el ciclo for  
* Recordemos que  $5! = 5*4*3*2*1$ ;  
*/
```

```
//Declaracion de variables
```

```
let n = 5; //El número a hacer factorial  
let f = 1; //El acumulador f lo inicio en 1
```

```
//Bloque for
```

```
for (i = 1; i<=5; i++){  
f*=i;  
}
```

```
console.log("El factorial de "+n+" es " +f);
```

Herramientas que utilizamos en clases



VSCode+plugins

No te olvides de dar el presente

Recordá:

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

Todo en el Aula Virtual.