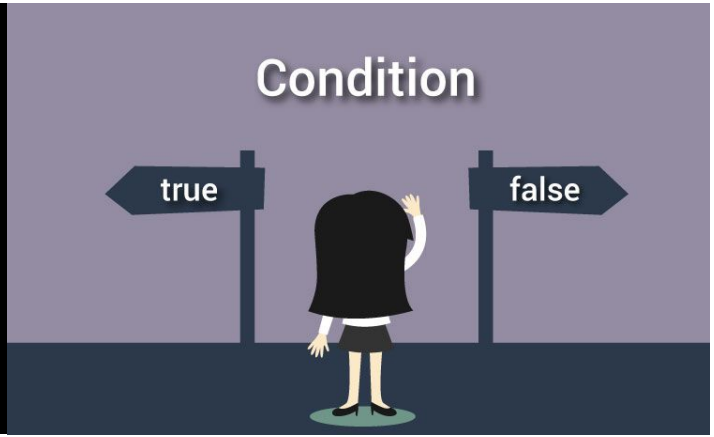

# Estructuras condicionales

# Declaraciones condicionales

- Las declaraciones condicionales nos ayudan cuando deseamos ejecutar diferentes acciones basadas en diferentes condiciones.
- Puedes utilizar **if** para especificar un bloque de código que será ejecutado si una condición específica es verdadera.

```
if (condición)
{
    statements
}
```



# Declaraciones condicionales

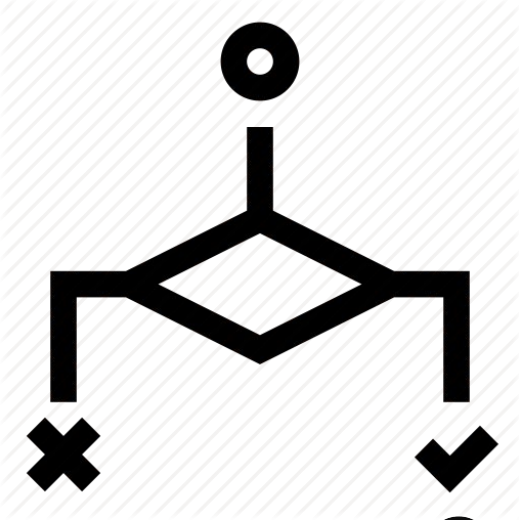
Las declaraciones serán ejecutadas sólo si la condición específica es **verdadera**

```
let num = 8;
if (num < 10) {
  console.log("El número es menor a 10");
}
```

# Condicionales dobles

- Para que una condicional sea doble utilizaremos el campo **else**, este nos permite definir el código que se debe ejecutar si el **if** no se cumple, es decir si la condición evalúa a falso.

```
if (condicion)
{
    statements
}
else
{
    statements
}
```



# Condicionales dobles

Ejemplo

```
let num = 8;
if (num < 10) {
  console.log("El número es menor a 10");
} else {
  console.log("El número es igual o mayor a 10");
}
```

# Condicionales múltiples o anidados

- En JavaScript (y en la mayoría de lenguajes de programación) es posible anidar condicionales, así que una posible solución al ej. anterior  sería la siguiente:

```
let num = 8;
if (num < 10) {
  console.log("El número es menor a 10");
} else {
  if (num > 10) {
    console.log("El número es mayor a 10");
  } else {
    console.log("El número es igual a 10");
  }
}
```

# Condicionales múltiples - “De lo contrario, si” (else if)

- Es preferible no tener que anidar condicionales porque son difíciles de leer y entender. Un atajo que nos ofrece JavaScript para los condicionales es el **else if**, que significa "De lo contrario, si ..." en Inglés.

```
if (<primera condición>) {
  // código que se ejecuta si <primera condición> se cumple
} else if (<segunda condición>) {
  // código si <primera condición> NO se cumple, pero <segunda condición> se cumple
} else if (<tercera condición>) {
  // código si <primera condición> y <segunda condición> NO se cumplen, pero <tercera condición> sí se cumple
} else {
  // código si ninguna de las condiciones se cumple
}
```

# Condicionales múltiples - “De lo contrario, si” (else if)

- Ejemplo anterior

```
let num = 8;
if (num < 10) {
  console.log("El número es menor a 10");
} else if (num > 10) {
  console.log("El número es mayor a 10");
} else {
  console.log("El número es igual a 10");
}
```

# Condicionales múltiples - según sea , el caso de

- El condicional  switch , se parece mucho y puede ser utilizado en lugar del  if en ciertas ocasiones.

```
switch (expresión) {
  case valor:
    // bloque de código
    break;
  case valor:
    // bloque de código
    break;
  default:
    // bloque de código
}
```

# Condicionales múltiples - según sea , el caso de

- Cada **case** , se interpreta de la siguiente forma: "si la expresión es igual al valor, ejecuta el bloque de código siguiente"
- La palabra **break** hace que la ejecución salte fuera del código switch para que no se ejecute los casos que vienen a continuación.
- La palabra **default** es como el else de los condicionales if, indica qué hacer si ninguno de los casos anteriores son iguales a la expresión.

# Condicionales múltiples - según sea , el caso de

```
let estacion = prompt("¿Cuál es tu estación del año preferida?");
switch (estacion) {
 case "primavera":
   // si la variable estacion contiene la cadena de texto "primavera"  se ejecutará este bloque de código
   console.log('la primavera');
   break;
 case "verano":
   // si la variable estacion contiene la cadena de texto "verano"
   // se ejecutará este bloque de código
   console.log('el verano');
   break;
 case "otoño":
   console.log('el otoño');
   break;
 case "invierno":
      console.log('el invierno');
   break;
 default:
   // si la variable estacion no contiene ningún nombre válido
   // se ejecutará este bloque de código
   console.log('no es una estación del año');
}
```

# Condiciones compuestas

- Imaginemos que queremos escribir un programa que imprima "El número está entre 10 y 20" si el valor de una variable está efectivamente entre 10 y 20.
- Una opción es usar condiciones anidadas, de esta forma:

```
let num = 15;
if (num >= 10) {
  if (num <= 20) {
    console.log("El número está entre 10 y 20");
  }
}
```

# Condiciones compuestas

- Sin embargo, como decíamos antes, leer condiciones anidadas es difícil y, en lo posible, es mejor evitarlas. En cambio, podemos utilizar los operadores lógicos y (&&) y o (||) para crear condiciones compuestas.

```
let num = 15;
if (num >= 10 && num <= 20) {
  console.log("El número está entre 10 y 20");
}
```

# Evaluación de expresiones booleanas

| Expresión | Resultado |
| --- | --- |
| true && true | true |
| true && false | false |
| false && true | false |
| false && false | false |

| Expresión | Resultado |
| --- | --- |
| true \|\| true | true |
| true \|\| false | true |
| false \|\| true | true |
| false \|\| false | false |


# Estructuras repetitivas - Bucles

# Bucles

- Los bucles nos permiten repetir la ejecución de un código varias veces.
- Imaginemos que quisiéramos repetir la frase "Hola mundo" 5 veces.

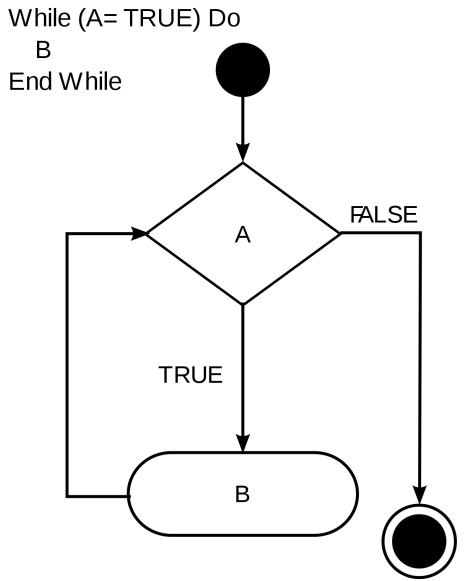
```
console.log("Hola Mundo");
console.log("Hola Mundo");
console.log("Hola Mundo");
console.log("Hola Mundo");
console.log("Hola Mundo");
```

- Ahora imagina que quisiéramos repetir 1000 veces. Ya no sería tan divertido copiar todo ese número de líneas en el archivo. Podemos entonces utilizar un ciclo.

# While (Mientras)

Se crea utilizando la palabra clave while seguido de una condición, que va a definir el número de veces que se va a repetir ese ciclo.

```
let i = 1;
while (i < 1000) {
  console.log("Hola mundo");
  i = i + 1;
}
```



# Do While (Hacer Mientras)

Es muy similar al ciclo while. Este ciclo crea un bucle que ejecuta una sentencia especificada, hasta que la condición se evalúa como falsa. La condición se evalúa después de ejecutar el cuerpo del ciclo, dando como resultado que el cuerpo especificado se ejecute al menos una vez.

```
do{
// acá va el cuerpo del ciclo, el código que se va a repetir al menos una vez y mientras la condición se cumpla
}while (<condicion>)
```

# Ciclo for

El while y do while es todo lo que necesitamos para hacer ciclos en JavaScript. Sin embargo, ese patrón que vimos en el ejemplo anterior en el que tenemos una inicialización (let i = 0), una condición (i < 1000) y un incrementador o contador (i++) es tan común, que JavaScript tiene un atajo para esto, el **for**.

```
for (<inicialización>; <condición>; <incrementador>) {
  // el cuerpo del ciclo, el código que se repite mientras que la
condición sea verdadera
}
```

# Ejemplos con while y for

Imaginemos que queremos hacer un programa que imprima los números del 10 a 20 pero saltando cada otro número, es decir, que imprima 10, 12, 14, 16, 18 y 20.

```
let i = 10;       // el inicializador
while (i <= 20) { // la condición
  console.log(i); // el cuerpo
  i = i + 2;      // el incrementador
}
```

```
for (let i=10; i <= 20; i = i + 2) {
  console.log(i);
}
```

# [ Arrays ]

# JS

# Arreglos

Un arreglo es una lista ordenada de elementos de cualquier tipo. Para crear tu primer arreglo abre la consola de navegador y escribe lo siguiente:

```
let array = [1, "Pedro", true, false, "Juan"]
```

La sintaxis de un arreglo es muy simple. Los elementos del arreglo se envuelven entre **corchetes** y se separan con coma.

# Obtener elementos del arreglo

Para obtener la primera posición del arreglo que acabamos de crear utilizamos array[0]:

```
> array = [1, "Pedro", true, false, "Juan"]
[1, "Pedro", true, false, "Juan"]
> array[0]
1
```

# Recorriendo un arreglo

Aquí simplemente usamos un ciclo for para mostrar cada uno de los elementos del arreglo.

```
const array = [1, "Pedro", true, false, "Juan"];
for (let i=0; i < array.length; i++) {
  console.log(array[i]);
}
```

# Agregar nuevos elementos

Es posible insertar nuevos elementos en un arreglo (puede estar vacío o tener elementos).

```
let array = ["Pedro"];
array.push("Germán"); // ["Pedro", "Germán"]
array.push("Diana"); // ["Pedro", "Germán", "Diana"]
```

El método push te permite agregar un elemento al final de la lista.

# Reemplazar un elemento

Es posible reemplazar el valor de cualquier elemento del arreglo. Por ejemplo:

```
let array = [1, "Pedro", true, false, "Juan"];
array[1] = "Germán"; // reemplazamos el elemento en la posición 1
// [1, "Germán", true, false, "Juan"]
```

# Agregar nuevos elementos

¿Qué pasa si queremos agregar un elemento en otra posición? Para eso sirve el método splice:

```
var array = ["Pedro", "Germán", "Diana"];
array.splice(0, 0, "Juan") // ["Juan", "Pedro", "Germán", "Diana"]
```

# Eliminar elementos

Para eliminar elementos de un arreglo utilizas el método splice. Por ejemplo:

```
let array = ["Pedro", "Germán", "Diana"];
array.splice(1, 1); // ["Pedro", "Diana"]
```

El método splice recibe uno o dos argumentos cuando se desea eliminar elementos: el índice del elemento que quieres eliminar y la cantidad de elementos a eliminar.

# Eliminar elementos

Si omites el segundo argumento se eliminarán todos los elementos después del índice que hayas especificado en el primer argumento. Por ejemplo:

```
let array = ["Pedro", "Germán", "Diana"];

array.splice(0); // []
```

En este último ejemplo el arreglo queda vacío debido a que no se indicó el segundo parámetro del método splice.

```
function nombre_funcion(parametros){
  instrucciones;
}
```

ROLLING CODE SCHOOL

# Funciones

- Frecuentemente vamos tener algunas líneas de código que necesitan ser ejecutadas varias veces y desde diferentes partes de nuestro programa. En vez de repetir el mismo código una y otra vez puedes crear una función (también conocidas como procedimientos o métodos) e invocarla cada vez que necesitemos ejecutar ese trozo de código.

```
function myFunction(p1, p2) {

  return p1 * p2;   // The function returns the product of p1 and p2

}
```

# sintaxis de una función

- Una función se define con la palabra clave **function**
- seguido va el nombre de la función
- luego los paréntesis, que pueden contener o no parámetros.
- por último entre llaves { } el código a ser ejecutado.
- Los parámetros de una función, son variables locales dentro de la función.
- Además una función puede devolver un valor, en este caso debe indicarlo con la palabra return y el resultado que devuelve.

```
function name(parameter1, parameter2, parameter3) {

  // code to be executed

}
```

# Funciones que devuelven un valor

En este ejemplo se crea la función square, la cual recibe como parámetro un número. Luego de realizar la operacion dentro de la función, está devuelve o retorna un resultado numérico.

```
function square(number) {
return number * number;
}

var cuadrado = square(10); //llamo a la función
```