Hack Academy®

Curso de Front-End

Clase 09

Agenda de la clase

Agenda

- Repaso.
- Comparadores.
- Condicionales (`if/else`).
- *Truthy* & *Falsy*.
- Operadores `&&` y `||`.
- `for` *loops*.
- Recorrer un *array* (usando `for`, `forEach` y `for..of`).
- Ejercicios.

Repaso

Tips generales

¡Practiquen mucho!

👉 Sugerimos que practiquen aprox. **6 horas semanales**.

⚠️ Es la única forma de realmente aprovechar el curso.

😢 Además, a medida que transcurre el curso, la diferencia entre los alumnos que practican y los que no, empieza a ser cada vez más grande.

REPASO

Booleanos

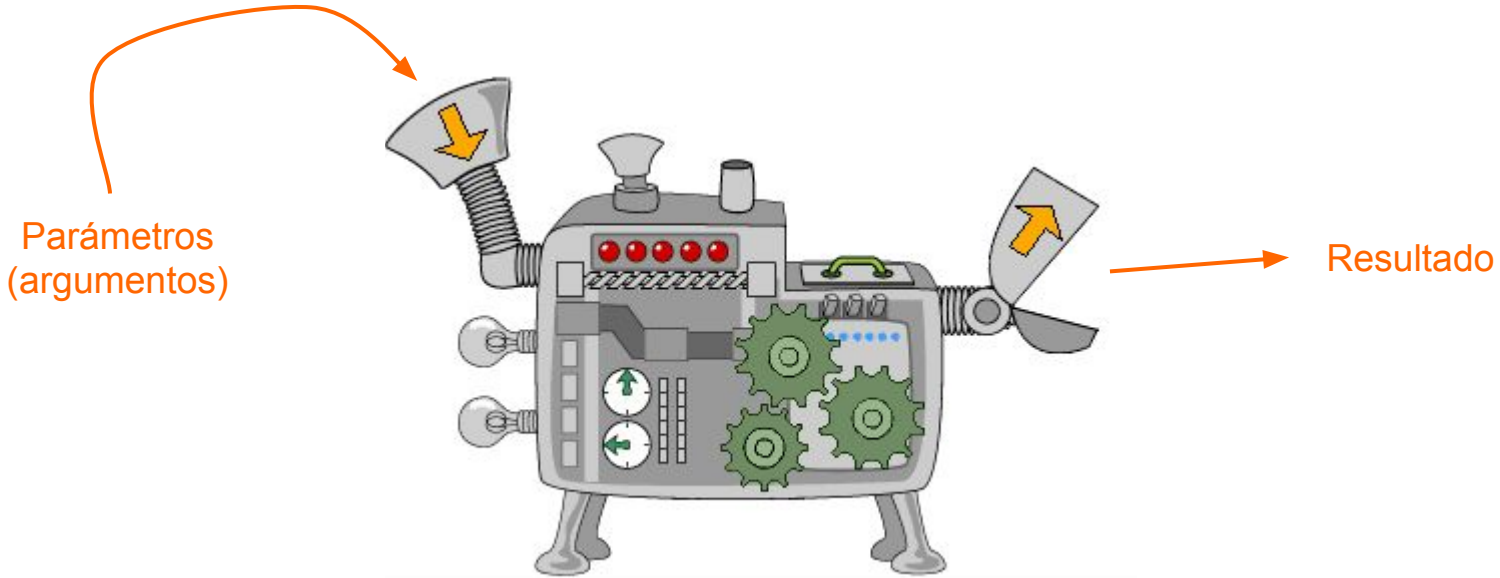
Boolean es otro tipo de dato (como lo es *String* y *Number*).

Un dato booleano sólo puede tomar 2 valores: `true` o `false`.

```
var mariaEsPeruana = true;
var hoyEsFeriado = false;

!mariaEsPeruana; // ¿Qué hace esto?
```

¿Qué es una función?

Pueden pensar en una función como en una máquina que recibe materia prima (parámetros o argumentos), los procesa y luego retorna un producto final (resultado).



REPASO

¿Qué es una función? (en JavaScript)

- Es un bloque de código (re-utilizable) que cumple determinada tarea.
  Al igual que un programa, una función está compuesta por una secuencia de sentencias llamadas “cuerpo” de la función.
- Re-utilizable: La función se puede usar y re-usar sin límites.
- Cuando se utiliza una función, se dice que se la *llama* (*call a function*).
  También se puede decir que se “invoca” una función.
- Toda función retorna (devuelve) un resultado.
  Si no se especifica el resultado, se retorna (casi siempre): `undefined`. Por eso en la consola muchas veces aparece escrito el texto “undefined”, aparentemente sin sentido.
- Opcionalmente, una función puede recibir parámetros (argumentos) cuando se la llama.

Nota técnica: una función en JS es un *objeto*. Lo veremos más adelante.

REPASO

¿Cómo se **declara** una función? – Ejemplo

```
function calcularSueldoLiquido(sueldoBruto) {
  /**
   * Nota: la siguiente es una simplificación del
   * cálculo del sueldo líquido de un empleado.
   */
  var descuento = 0.80;
  return sueldoBruto * descuento;
}
```

Esta forma de definir una función se llama “Function Declaration”.

REPASO

¿Cómo se **llama** a una función? – Ejemplo

```
calcularSueldoLiquido(30000); // Retorna 24000.

calcularSueldoLiquido(65000); // Retorna 52000.

calcularSueldoLiquido(100000); // Retorna 80000.
```

REPASO

¿Qué es un *Array*? (en JavaScript)

- Es una **estructura de datos** que permite almacenar un conjunto de valores en formato lista. Se lo puede pensar como una "lista de elementos".
- En lugar manipular los valores por separado, se manipula el conjunto.
- El largo de los *arrays* es variable.
- Los valores contenidos en un *array* no tienen por qué ser del mismo tipo.
  Pueden ser *strings*, *numbers*, *booleans*, funciones, otros *arrays*, etc (todo mezclado).
  De todas maneras, lo más usual es que los elementos de un *array* sean del mismo tipo.
- En español se les llama "Arreglos" (o incluso "Vectores").

Nota técnica: un *array* en JS es un *objeto*. Lo veremos más adelante.

REPASO

¿Cómo se crea un *Array*?

Se utilizan corchetes `[]` y se pasan los valores separados por comas `","`.

```
var marcas = ["BMW", "Peugeot", "Chevrolet", "Subaru", "Nissan"];
```

También se puede crear un *array* vacío:

```
var marcas = [];
```

REPASO

Índices en un *Array*

A cada elemento dentro del *array* le corresponde un índice.

```
var marcas = ["BMW", "Peugeot", "Chevrolet", "Subaru", "Nissan"];
```

Nota: tener en cuenta que el índice del *array* comienza en 0, no en 1.
Esto varía según cada lenguaje. Los que comienzan en 0 suelen conocerse como *zero-index array*.

Acceder a un elemento de un *Array*

```
marcas[2];
```

Se accede al elemento de la posición 2.

Agregar un elemento a un *Array*

```
marcas[5] = "Fiat";
```

Se agrega el elemento “Fiat” a la posición 5.

¿Qué pasa si ya había un elemento en la posición 5?

REPASO

Largo de un *Array*

```
marcas.length; // Esto devuelve el largo del array "marcas".
```

Métodos (funciones) de un *Array*

```
marcas.push("Fiat"); // Agrega "Fiat" al final del array (sin especificar el índice).
marcas.pop(); // Elimina el último elemento del array.
marcas.shift(); // Elimina el primer elemento del array.
marcas.unshift("VW"); // Agrega "VW" al inicio del array (mueve los otros elementos).
marcas.splice(1,2); // Elimina 2 elementos del array, desde la posición 1 inclusive.
marcas.toString(); // Retorna un string con los elementos del array separados por comas.
marcas.join(" - "); // Similar a toString, pero permite especificar separador.
```

Documentación: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Methods_2

Comparadores

(Operadores de comparación)

Comparadores (1/5)

Operador	Significado
==	Igual a
===	Estrictamente igual a
!=	Distinto a
!==	Estrictamente distinto a
>	Mayor a
<	Menor a
>=	Mayor o igual a
<=	Menor o igual a

La recomendación es evitar el uso del doble igual (“==”). Por defecto, siempre se sugiere usar el triple igual (“===”).

También se los llama operadores de comparación.

Comparadores (2/5)

Ejemplos:

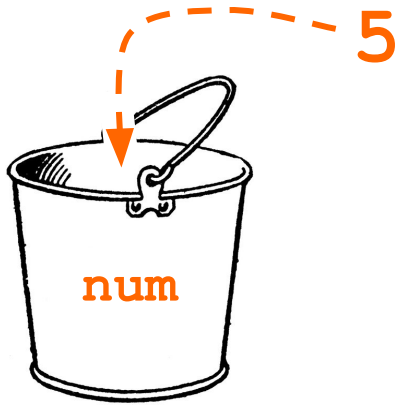
```
1 === 5;   // Retorna false.

1 !== 5;   // Retorna true.

2 < 4;    // Retorna true.

4 < 4;    // Retorna false.

9 <= 9;   // Retorna true.

9 >= 9;   // Retorna true.
```

Comparadores (3/5)

Preguntas:

```
"hola" === "HOLA";  // ¿Qué resultado retorna esto?
```

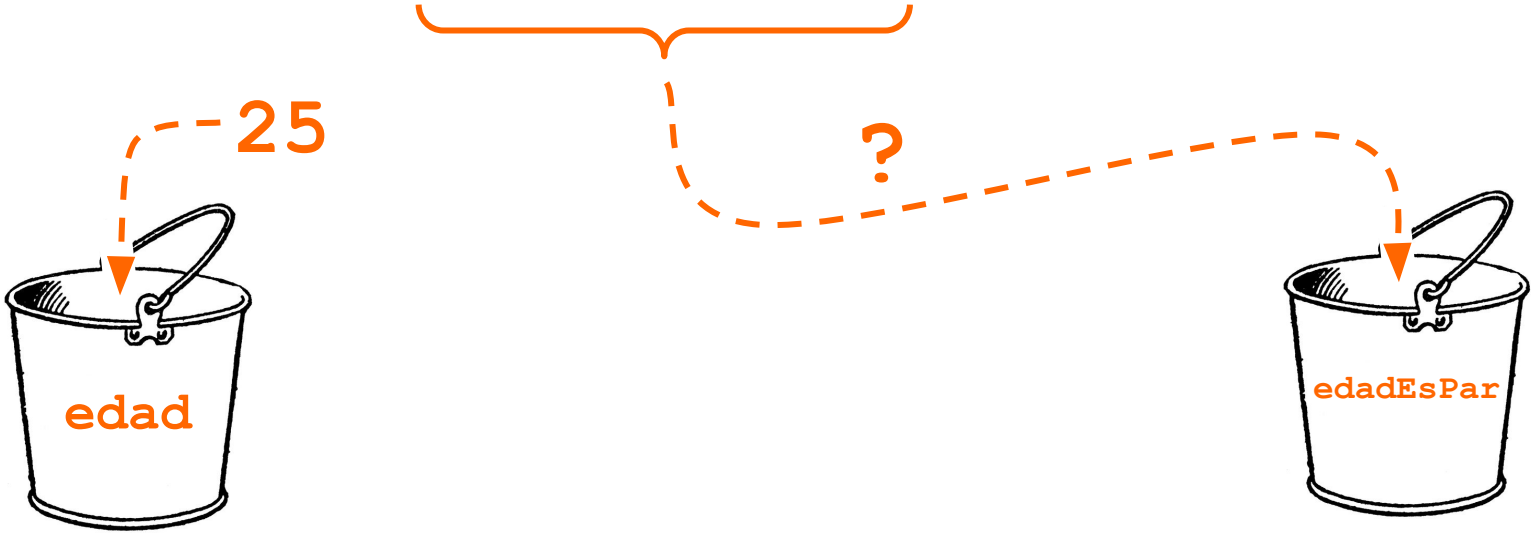
```
var num = 5;

num !== 6; // ¿Qué resultado retorna esto?
```



Comparadores (4/5)

¿Qué se guarda en `edadEsPar`?

```
var edad = 25;

var edadEsPar = ((edad % 2) === 0);
```



Comparadores (5/5)

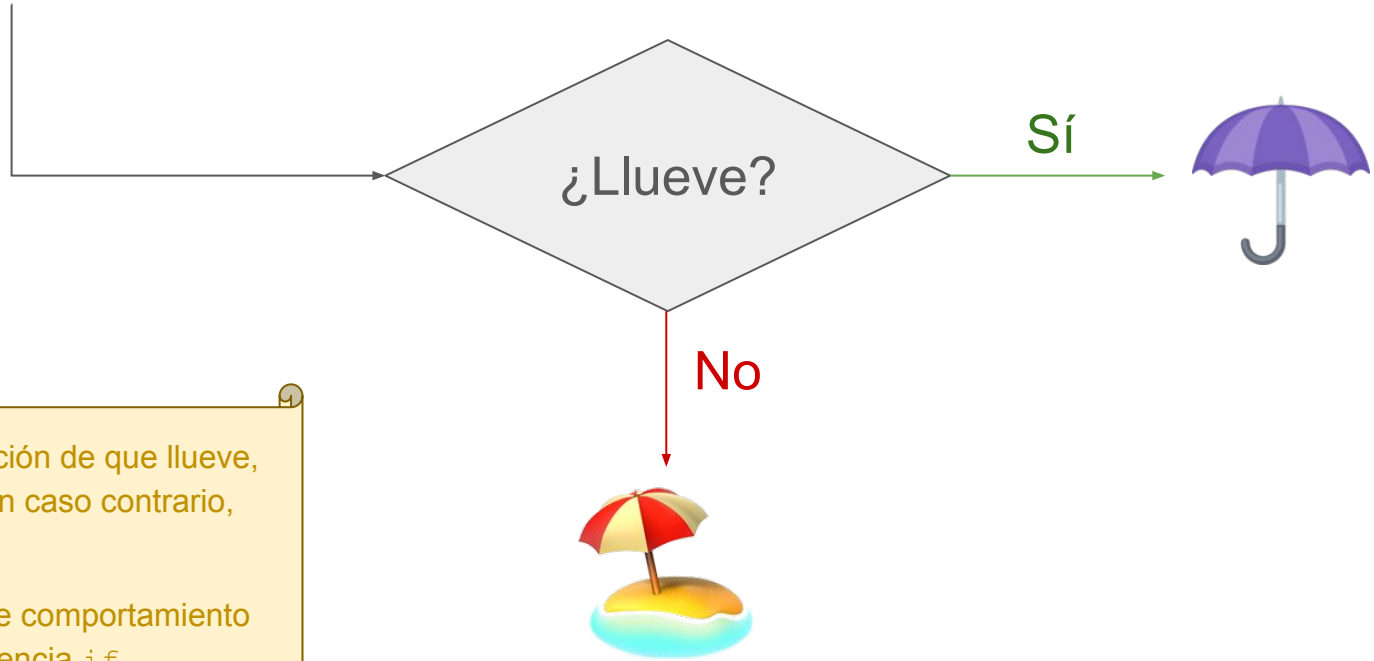
```
0 == ""; // ¿Qué resultado retorna esto?
0 === ""; // ¿Y esto?

1 == true; // ¿Qué resultado retorna esto?
1 === true; // ¿Y esto?

0 == false; // ¿Qué resultado retorna esto?
0 === false; // ¿Y esto?
```

⚠️ **Nota**: No es importante saberse todo esto de memoria. De hecho, cuando se comparan dos valores, en general siempre se comparan valores del mismo tipo.

Condicionales (`if` / `else`)

“Todo el tiempo tomamos decisiones basados en una condición”

Ejemplo de un condicional:



Si se cumple la condición de que llueve, llevamos paraguas. En caso contrario, vamos a la playa.

En programación, este comportamiento se logra con una sentencia `if`.

Sentencia `if`

👉 Aprovechar **VSC** al máximo. Usar el auto-complete al crear un `if`.

Una sentencia `if` se utiliza para especificar si un bloque de código se debe ejecutar o no dependiendo del valor de cierta condición.

```
if (CONDICIÓN) {

    // Bloque de código a ejecutar

    // si la condición se cumple...

}
```

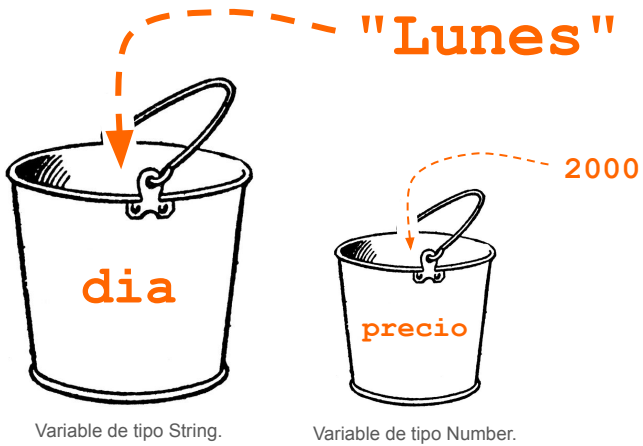
```
if (LLUEVE) {

    // Buscar el

    // paraguas...

}
```

👉 La condición es algo que puede ser `true` o `false`. En realidad, *truthy* o *falsy* (ya lo veremos).

Sentencia `if`

Ejemplo:

```
var dia = "Lunes";
var precio = 2000;
```




```
if (dia === "Martes") {
    alert("¡Hoy es martes! Por lo tanto te ofrecemos un 10% OFF");
    precio = precio * 0.90;
}
```

En este caso, la condición no se cumple y por lo tanto no se muestra el `alert`. El `precio` sigue siendo $2000. Pero si fuese martes, se le haría un 10% de descuento al comprador.

Sentencia `else`

👉 Aprovechar **VSC** al máximo. Usar el auto-complete al crear un `if-else`.

```
if (CONDICIÓN) {

    // Bloque de código a ejecutar si la condición se cumple...

} else {

    // Bloque de código a ejecutar si la condición NO se cumple...

}
```

Nota: El `else` no necesita de una condición.

Ejercicios

Ejercicios

1. Definir una función llamada `esPar` que reciba como parámetro un número cualquiera y retorne `true` si es par. En caso contrario, retorna `false`.

2. Definir una función llamada `maximoDeDos` que reciba como argumentos dos números cualesquiera y retorne el mayor. Si los números son iguales retornar dicho número.
   Nota: No utilizar la función `Math.max(...)`.

3. Definir una función llamada `maximoDeTres` que reciba como argumentos tres números cualesquiera y retorne el mayor. Si los números son iguales retornar dicho número.
   Nota: No utilizar la función `Math.max(...)`. Intentar hacerlo utilizando la función `maximoDeDos`.

EXTRA

Ejercicios (cont)

4. Definir una función llamada `validarNombre` que reciba como parámetro un *string* cualquiera y retorne `true` si el argumento recibido efectivamente es un *string* y además no coincida con el *string* vacío “”. En caso contrario, retornar `false`.

5. Definir una función llamada `validarAnioNacimiento` que reciba como parámetro un número cualquiera y retorne `true` si:
   - el argumento recibido efectivamente es un número y
   - el argumento recibido es mayor a 1920 y
   - el argumento recibido es menor o igual a 2021.

   En caso contrario, la función debe retornar `false`.

Sentencia `else if`

Sentencia `else if` (1/2)

```
if (CONDICIÓN_A)
{
    // Bloque de código a ejecutar si la condición A se cumple...
}
else if (CONDICIÓN_B)
{
    // Bloque de código a ejecutar si la condición A NO se cumple
    // y se cumple la condición B...
}
```

Sentencia `else if` (2/2)

Ejemplo:

Supongamos que cierto comercio está abierto entre las 09:00 y 18:00 horas. Fuera de dicho horario, está cerrado.

```
if (hora > 18) {
    alert("Está cerrado, es tarde.");
} else if (hora < 9) {
    alert("Está cerrado, es temprano.");
} else {
    alert("Está abierto");
}
```

Truthy & Falsy

Truthy & *Falsy* (1/2)

Todo en JavaScript tiene inherentemente un valor booleano, conocido como *truthy* o *falsy*.

Un valor *truthy* es un valor que se considera `true` al ser evaluado en un contexto booleano (por ejemplo: en la condición de un `if`).

Ejemplos de valores que son *truthy*:

- `true`
- `1`
- `38`
- `"hola"`

Ejemplos de valores que siempre son *falsy*:

- `false`
- `0` (cero)
- `""` (string vacío)
- `null`
- `undefined`
- `NaN` (Not-a-Number)

Nota: Los círculos muestran los más interesantes.

Truthy & Falsy (2/2)

Ejemplos:

```
var esFeriado = 1;
if (esFeriado) {
    alert("¡El día es feriado!"); // ¿Se ejecuta esta sentencia?
}

var esVacio = 0;
if (!esVacio) {
    alert("¡No está vacío!"); // ¿Se ejecuta esta sentencia?
}
```

De todas maneras, por más de que este código sea válido, para estos casos es preferible usar `true` y `false`.

Operadores && y | |

Operadores Lógicos: && y ||

A veces es conveniente hacer varias comparaciones en una misma sentencia.

Ejemplos:

```
var precioA = 400;
var precioB = 700;
var precioC = 400;

(precioA === precioC) && (precioA < precioB);   // Retorna true;
(precioA === precioC) || (precioA === precioB);  // Retorna true;
```

JavaScript evalúa los operadores de izquierda a derecha y se detiene cuándo conoce la respuesta.

Operador Lógico: &&

Ejemplo:

```
var anioNacimiento = 1989;
```

```
if (anioNacimiento >= 1980 && anioNacimiento <= 1995) {

    alert("¡Sos un Millennial!");

}
```

¿Se ejecuta el `alert`?

Operador Lógico: ||

Ejemplo:

```
var pais = "Ecuador";
```

```
if (pais === "EEUU" || pais === "Canadá" || pais === "México") {
    alert("Tu país está en América del Norte");
} else {
    alert("Tu país NO está en América del Norte");
}
```

¿Cuál de los dos `alert` se ejecuta? ¿Se podrían ejecutar ambos a la vez? ¿Podría ser ninguno?

Tabla de Verdad

var1	var2	var1 && var2	var1 || var2
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Probablemente hayan visto esta tabla en alguna asignatura de secundaria (liceo). En Uruguay, este tema se ve en 5º de liceo, en la asignatura **Filosofía**, dentro del tema **Lógica**.

Loops

(loop = bucle)

“Muchas veces necesitamos ejecutar un bloque de código repetidamente”

Loops – Ejemplo 1

Supongamos que queremos mostrar en la consola todos los números enteros del 1 al 10, uno por línea. Con lo visto hasta ahora, tendríamos que hacer lo siguiente:

```
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
console.log(6);
console.log(7);
console.log(8);
console.log(9);
console.log(10);
```

Loops – Ejemplo 1 (cont)

¿No sería mucho mejor poder hacer algo así?

REPETIR 10 VECES, y que `i` varíe entre 1 y 10:

```
console.log(i);
```

🙌 Esto se logra usando loops.

Loops – Ejemplo 2

Supongamos que tenemos un *array* con 8 elementos (*strings*) y queremos imprimir a cada uno de ellos en una línea nueva en la consola. Se podría hacer lo siguiente:

```
console.log(marcas[0]);
console.log(marcas[1]);
console.log(marcas[2]);
console.log(marcas[3]);
console.log(marcas[4]);
console.log(marcas[5]);
console.log(marcas[6]);
console.log(marcas[7]);
```

Loops – Ejemplo 2 (cont)

¿No sería mucho mejor poder hacer algo así?

REPETIR 8 VECES, y que `i` varíe entre 0 y 7:

```
console.log(marcas[i]);
```

🙌 Esto se logra usando *loops*.

`for` *Loops*

`for` *Loops* (1/4)

El objetivo de un `for` *loop* es iterar un bloque de código mientras se cumpla cierta condición. Se suele usar para ejecutar un bloque de código una determinada cantidad de veces (pre-conocida).

```
for (var i = 1; i <= 100; i++) {

    // Bloque de código que se ejecutará repetidamente:

    console.log("Iteración número: " + i);

}
```

El `console.log` se ejecuta 100 veces.

`i++` es equivalente a hacer:
`i = i + 1`

`for` *Loops* (2/4)

Resultado impreso en la consola:

```
for (var i = 1; i <= 100; i++) {
  console.log("Iteración número: " + i);
}
```

i=1

```
Iteración número: 1
```

```
for (var i = 1; i <= 100; i++) {
  console.log("Iteración número: " + i);
}
```

i=2

```
Iteración número: 1
Iteración número: 2
```

```
for (var i = 1; i <= 100; i++) {
  console.log("Iteración número: " + i);
}
```

i=3

```
Iteración número: 1
Iteración número: 2
Iteración número: 3
```

`for` *Loops* (3/4)

👉 Aprovechar **VSC** al máximo. Usar el auto-complete al crear un `for`.

Forma general:

```
for (SENTENCIA_INICIAL;  CONDICIÓN;  SENTENCIA_FINAL) {

    // Bloque de código que se ejecutará repetidamente...

}
```

- **Sentencia Inicial**: se ejecuta una sola vez, antes de la primer iteración.
- **Condición**: es una expresión que se evalúa antes de cada iteración del *loop*. El *loop* se ejecuta mientras la condición sea `true`.
- **Sentencia Final**: se ejecuta inmediatamente después de cada iteración.

`for` *Loops* (4/4)

Ejemplo: Supongamos que es necesario mostrar un `select` con todos los años desde 2022 hasta 1900. Escribir todo el código a mano sería una locura. Mejor sería insertar los `option` usando un *loop*.


```
<select id="year" name="year">
    <option value="" disabled selected>Seleccionar...</option>
    <option value="2022">2022</option>
    <option value="2021">2021</option>
    <option value="2020">2020</option>
    ...
    <option value="1900">1900</option>
</select>
```

Ejercicios (cont)

Ejercicio 6 – `for` *Loop*

Usando JavaScript, programar un `for` *loop* que imprima en la consola todos los números múltiplos de 4 del 0 al 100.

Consola:

```
0
4
8
12
...
100
```

Tal vez puede resultar útil saber que:

- `x` es múltiplo de `4` si el resto de la división entera (el módulo) entre `x` y `4` es igual a cero.
- Si fuese necesario, es posible usar un `if` dentro del `for`.

Recorrer un *Array*

(usando un `for`, un `forEach` o un `for...of`)

Recorrer un *Array* – Usando un `for`

```
var marcas = ["BMW", "Peugeot", "Chevrolet", "Subaru", "Nissan"];
```

```
for (var i = 0; i < marcas.length; i++) {

    console.log(marcas[i]);

}
```

Esta es la forma más “tradicional” o “antigua” de recorrer un *array*.

Si bien la sintaxis del `for` es menos amigable con respecto a otros métodos más modernos utilizados para recorrer *arrays* (ej: `for..of`), hay algunos casos en los que puede ser útil debido a su flexibilidad.

Nota: El nombre de la variable `i` es arbitraria; por ejemplo, se le podría haber llamado `indice`.

Recorrer un *Array* – Usando un `for...of`

```
var marcas = ["BMW", "Peugeot", "Chevrolet", "Subaru", "Nissan"];
```

```
for (var item of marcas) {

    console.log(item);

}
```

El `for...of` (ver docs) está disponible desde la versión ES6 de JavaScript (año 2015) y no funciona en ninguna versión de Internet Explorer. Posiblemente es la sintaxis más sencilla que existe para recorrer *arrays*.

Nota: El nombre de la variable `item` es arbitraria; por ejemplo, se le podría haber llamado `marca`.

Recorrer un *Array* – Usando un `forEach`

```
var marcas = ["BMW", "Peugeot", "Chevrolet", "Subaru", "Nissan"];
```

```
marcas.forEach(function(item) {

    console.log(item);

});
```

⚠️ Actualmente esta alternativa para recorrer *arrays* está siendo poco utilizada. El `for..of` es una mejor solución (y más moderna).

El método `forEach` (ver docs) está disponible desde la versión ES5 de JavaScript (año 2009) y funciona a partir de Internet Explorer 9. Sólo se puede usar para recorrer *arrays* (a diferencia del `for` que tiene otros usos).

Nota: El nombre del parámetro `item` es arbitrario; por ejemplo, se le podría haber llamado `marca`.

Ejercicios (cont)

⚠️ Los siguientes ejercicios son muy importantes porque mezclan conceptos clave como Variables, If/Else, *Arrays*, *Fors* y Funciones.

Ejercicios (1/5)

7. Definir una función llamada `mostrarArray` que recibe como argumento un *array* cualquiera y lo muestra (imprime) en la consola. Cada elemento debe ir en una nueva línea.

8. Definir una función llamada `estaElemento` que recibe como argumentos un array cualquiera y un elemento cualquiera, y retorna `true` si dicho elemento se encuentra en el *array*. En caso contrario retorna `false`.

9. Definir una función llamada `maximo` que recibe como argumento un *array* cualquiera y retorna el elemento más grande de la lista. Suponer que el *array* recibido es una lista de números.

EXTRA & AVANZADO

Loops Anidados

“A veces es necesario colocar loops dentro de otros loops.”

EXTRA & AVANZADO

Ejemplo de loops anidados

El típico ejemplo de utilización de *loops* anidados, se da cuando se debe recorrer una lista de elementos, y que a su vez cada uno de estos elementos contiene otra lista adentro.

Por ejemplo, si en un *blog* se tiene una lista de artículos, cada artículo tiene una lista con los usuarios que le hicieron *Like*, y se quiere desplegar dicho contenido en la pantalla, primero será necesario recorrer la lista de los artículos (con un *primer loop* o *loop externo*) y luego para cada artículo será necesario recorrer su lista de usuarios (con un *segundo loop* o *loop interno*). 👉 Ver siguiente slide.

EXTRA & AVANZADO

Ejemplo de loops anidados en un *blog*:

Loop externo

Artículo 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Loop interno →

Likes: María Pérez, José González, Martín Gómez.

Artículo 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Loop interno →

Likes: María Pérez, José González, Martín Gómez, Victoria Rodríguez.

Artículo 3

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

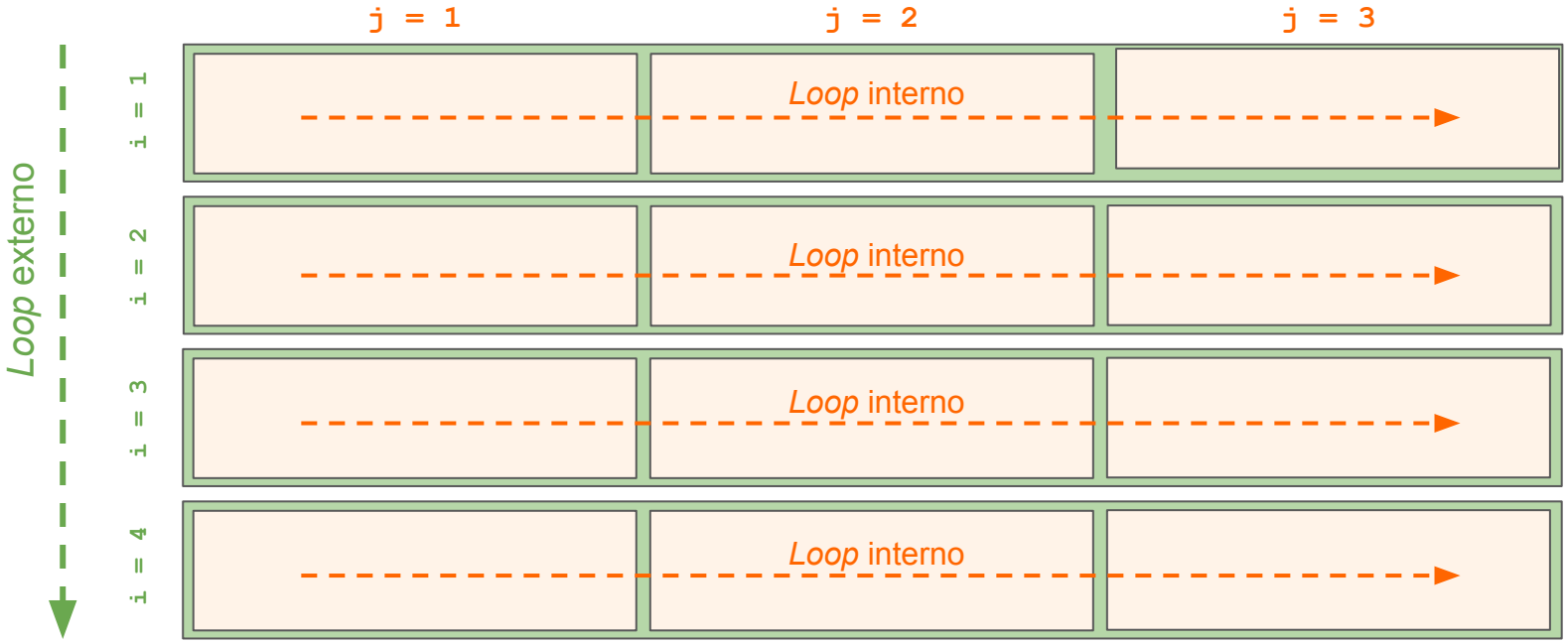
Loop interno →

Likes: Martín Gómez, Ana López.

EXTRA & AVANZADO

Ejemplo de loops anidados en una matriz:

Es común usar *loops* anidados cuando se quiere recorrer una estructura con forma de tabla/matriz (con filas y columnas). De hecho, este caso, es una generalización del ejemplo anterior (*blog*).



EXTRA & AVANZADO

Ejemplo de loops anidados en una matriz (cont):

El ejemplo anterior se puede implementar con el siguiente código:

```
for (var i = 1; i <= 4; i++) {
    // Se recorre una fila.
    for (var j = 1; j <= 3; j++) {
        // Se recorre una columna.
        console.log(i + "." + j);
    }
}
```

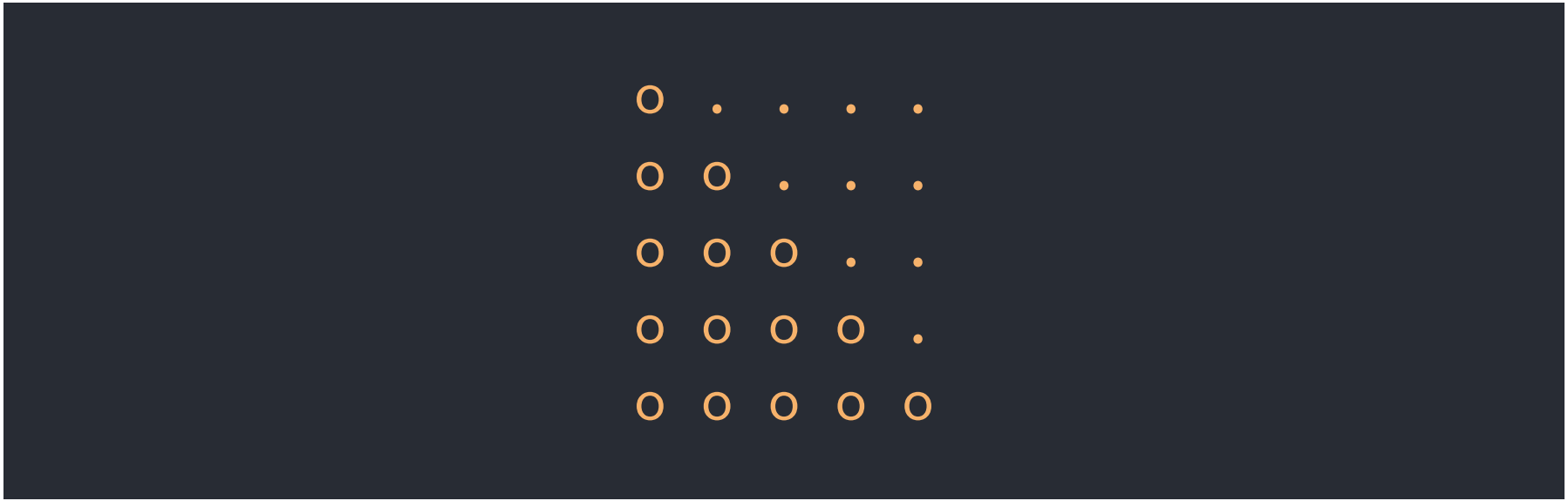
EXTRA & AVANZADO

Ejercicio 10

EXTRA & AVANZADO

Ejercicio 10 – `for` *loops* anidados

Se busca aplicar el concepto de `for` anidado para imprimir en consola el siguiente patrón:



EXTRA & MUY AVANZADO

Ejercicio 11

EXTRA & MUY AVANZADO

Ejercicio 11 – Loops y DOM

Insertar al final de una página, usando JavaScript, 64 cuadrados blancos y negros que formen un tablero de Ajedrez.

Ayuda:

- Crear una clase CSS para el cuadrado blanco y otra clase CSS para el cuadrado negro.
- Para insertar los cuadrados en la página, se puede utilizar la función `insertAdjacentHTML` o `append`.