



# Coding Bootcamp

## Sprint 1



# Repaso de JavaScript



# Repaso de temas

- Tipos de datos.
- Operadores.
- If/Else.
- For/While Loops.
- Funciones.
  - Parámetros por referencia y por valor.
  - Scope.
- Arrays (Arreglos).
- Noción de objetos.



¿Qué es programar?



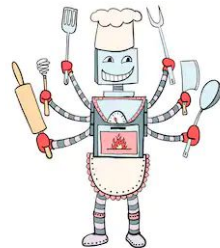
# ¿Qué es programar? (1/2)

Programar es “*crear un conjunto de instrucciones que serán ejecutadas por una computadora*”.

Por el momento, las computadoras son máquinas bastante “tontas”. Hay que decirles, con lujo de detalles (y **sin ambigüedades**), lo que queremos que hagan.

---

**Analogía:** Escribir un programa es como escribir una **receta de cocina** para una persona que nunca cocinó algo en su vida. Imagínense que en lugar de una persona es un robot. No alcanza con especificar los ingredientes y las cantidades. Es necesario explicar con precisión cada uno de los pasos que hay que realizar. Ejemplo: “*Girar la cuchara 90 veces en sentido horario a una velocidad de 300 rpm. Si quedan grumos, girar en sentido anti-horario a 120 rpm hasta que desaparezcan, etc*”. A la persona (o al robot) no le deben quedar dudas sobre cómo proceder.



# ¿Qué es programar? (2/2)

## Tipos de lenguajes de programación:

- **Código de máquina / Código binario**

- Es el menor nivel de abstracción. Consiste en en 1's y 0's.

```
0110001100
1011010110
1111011110
```

- **Lenguaje Assembly**

- Representación simbólica de código de máquina.

- **Lenguajes compilados**

- Lenguajes de alto nivel que necesitan ser compilados a código de máquina.
- Ej: C, C++, Go, Fortran, Pascal, Java.



- **Lenguajes interpretados**

- Lenguajes de alto nivel que no necesitan ser compilados. Programas residen en memoria.
- Ej: JavaScript, Python, PHP, Ruby.



Más nivel de abstracción





# Variables

# Variables (1/5)



Las variables son **contenedores que permiten guardar información** (valores) y poder usarla más tarde.

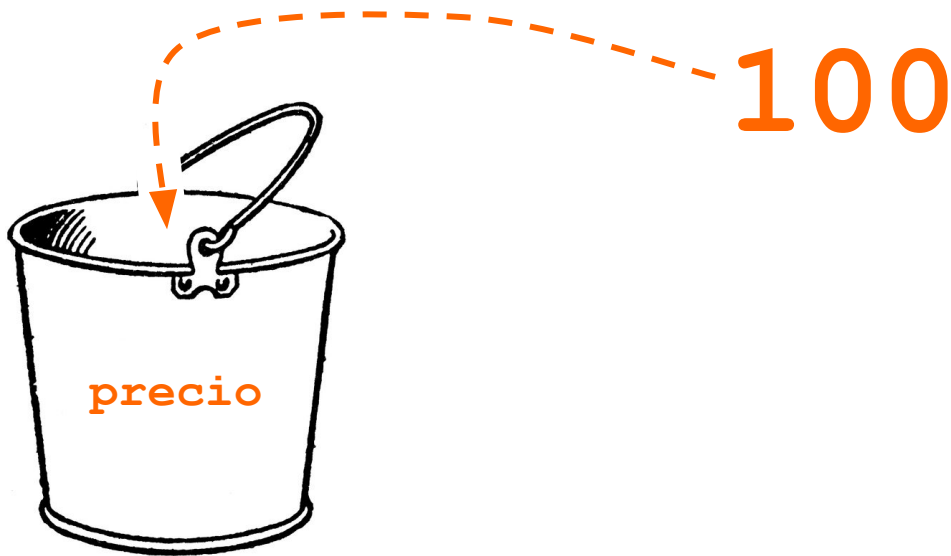






## Variables (2/5)

Para definir una variable en JavaScript (versión [ES5](#)) y asignarle un valor, se escribe el siguiente código: `var precio = 100;`



Ver [documentación](#).



# Variables (3/5)

Prueben en la consola escribir lo siguiente:

```
> var precioA = 100;  
> var precioB = 50.5;  
> precioA = precioA + precioB;  
> var precioC;
```

Línea 1: Sentencia que declara una variable llamada `precioA` y le asigna el valor 100.

Línea 2: Sentencia que declara una variable llamada `precioB` y le asigna el valor 50.5.

Línea 3: Sentencia que asigna la suma de `precioA` + `precioB` a la variable `precioA`.

Línea 4: Sentencia que declara una variable llamada `precioC`.



## Variables (4/5)

Las variables pueden guardar el resultado de **expresiones**.

```
> var unNumero = 5 + 7;  
> var unString = "Hola" + " " + "Mundo";
```

Importante: no se guardan las expresiones en sí mismas, lo que se guarda es el resultado.



# Variables (5/5) – ¿Cómo se nombran?

- Comenzar los nombres con letras.  
(Después verán casos en lo que las nombres pueden empezar con \$ o \_).
- Sólo usar letras, números, \$ y \_.
- NO usar espacios.
- Recordar que los nombres son case sensitive (*precio* y *preCio* son distintos).
- Evitar palabras reservadas (palabras que ya son usadas por JS).  
Ej: una variable no puede ser llamada `var` ni `typeof`.
- Usar nombres mnemotécnicos (Ej: es mejor usar `precioTotal` que `x1`).
- Es común usar camelCase (Ej: `precioVariableTotal` en lugar de `preciovariabletotal`).
- Es común (y recomendable) usar nombres en inglés.



# Tipos de datos

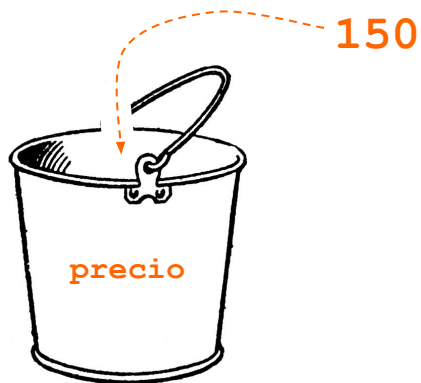


# Tipos de datos (1/2)

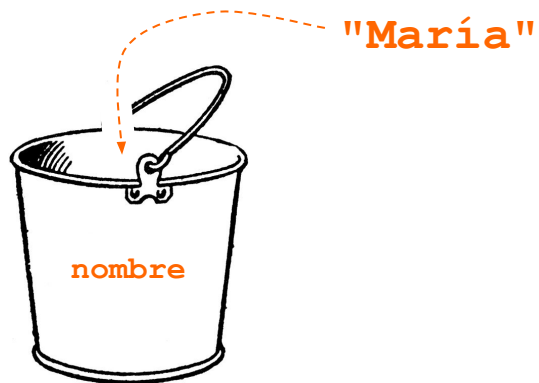
Hay varios tipos de datos en JavaScript. Ejemplos:

- `String`  
Conjunto de caracteres encerrados por comillas (simples o dobles).
- `Number`  
Pueden ser números enteros o decimales.
- `Boolean`  
Puede tomar 2 valores: `true` o `false`.

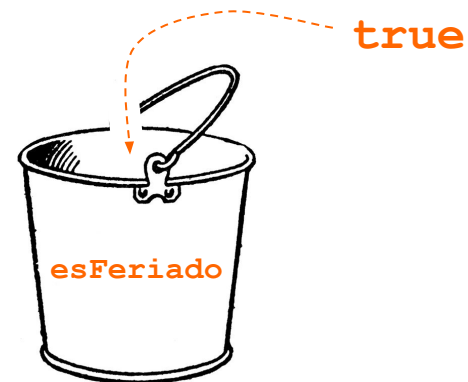
# Tipos de datos (2/2)



Variable de tipo Number.



Variable de tipo String.



Variable de tipo Boolean.

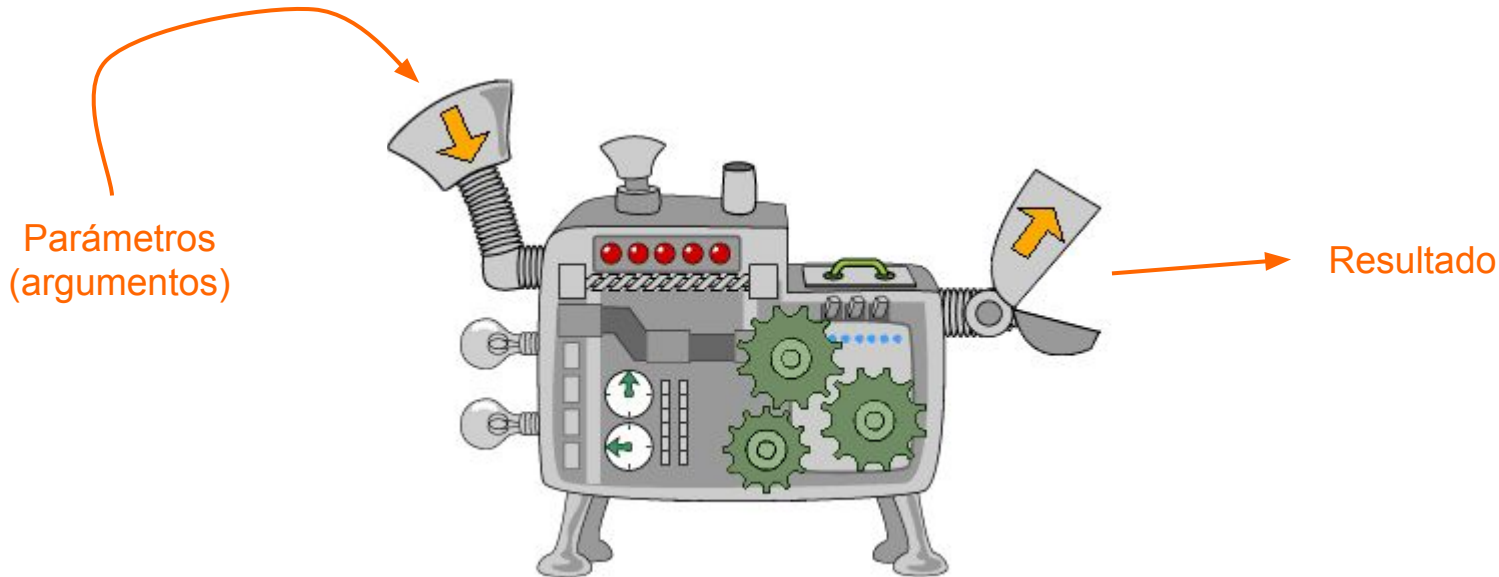


# Funciones



# ¿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**).





# ¿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*.

# ¿Cómo se **declara** una función?

👉 Aprovechar **VSC** al máximo. Usar el auto-complete al crear una `function`.



```
function nombreDeLaFuncion(arg1, arg2, etc) {  
  /**  
   * Aquí se coloca el "cuerpo" de la función.  
   * Eventualmente, la función puede retornar un  
   * resultado. En ese caso, se utiliza la palabra  
   * `return` para definirlo.  
   */  
}
```

Esta forma de definir una función se llama "Function [Declaration](#)".



# ¿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](#)".



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

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

```
calcularSueldoLiquido(65000); // Retorna 52000.
```

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



# Llamar a una función antes de ser definida

```
alert("El sueldo líquido es: " + calcularSueldoLiquido(30000));  
  
// ...más código...  
  
function calcularSueldoLiquido(sueldoBruto) {  
    var descuento = 0.80;  
    return sueldoBruto * descuento;  
}
```

Esto se puede hacer siempre y cuando la función haya sido definida usando "Function [Declaration](#)".



## Otra forma de declarar una función

```
var calcularSueldoLiquido = function(sueldoBruto) {  
    var descuento = 0.80;  
    return sueldoBruto * descuento;  
}  
  
/**  
 * Al hacerlo de esta forma, siempre se debe llamar a la  
 * función LUEGO de haber sido declarada.  
 */  
alert(calcularSueldoLiquido(30000));
```

Esta forma de definir una función se llama “Function [Expression](#)”.

⚠ Por el momento, esta forma de definir funciones **no** será de mucha utilidad. La usaremos en próximas clases.

# Funciones anónimas

```
$("#hack-academy").on("click", function () {  
    /**  
     * Código que se ejecuta si se realizó un  
     * click en el elemento con id="hack-academy"  
     */  
});
```

La función anónima es un tipo de declaración "*Function [Expression](#)*".

Además, notar que la función anónima se está pasando como parámetro a la función `on`. Es algo típico en JS.





# Elección de nombres para los parámetros

Al elegir nombres para los parámetros de una función:

- Usar nombres que tengan algún significado.
- No usar: `arg1`, `arg2`, `arg3`, `x`, `y`, `z`...

Mala práctica:

```
function toCelsius(arg) {  
    return (5/9) * (arg-32);  
}
```

Buena práctica:

```
function toCelsius(fahrenheit) {  
    return (5/9) * (fahrenheit-32);  
}
```

# Evitar que una función afecte variables externas



```
var temperaturaEnCelsius;
```

La función modifica una variable externa. Es una mala práctica y hay que tratar de evitarlo.

```
function toCelsius(fahrenheit) {
```

```
    temperaturaEnCelsius = (5/9) * (fahrenheit-32);
```

```
}
```



# Comparadores

(Operadores de comparación)

# Comparadores



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

También se los llama operadores de comparación.



# Condicionales

## (if / else)



# Sentencia `if`

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

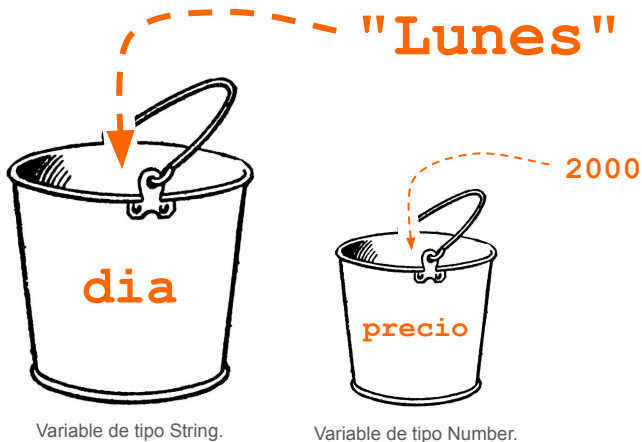
```
if (CONDICIÓN) {  
  
    // Bloque de código a ejecutar  
  
    // si la condición se cumple...  
  
}
```

```
if (LLUEVE) {  
  
    // Agarrar el  
  
    // paraguas...  
  
}
```

👉 La condición es algo que puede ser `true` o `false`. En realidad, *truthy* o *falsy*.

# Sentencia `if`

Ejemplo:



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

```
if (dia === "Martes") {  
    alert("Hay descuentos en SiSi! - 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.

# Sentencia `else`

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



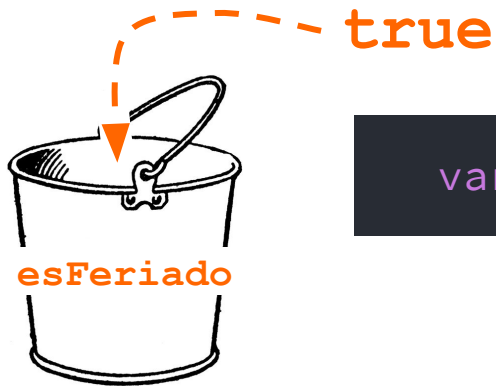
```
if (CONDICIÓN) {  
  
    // Bloque de código a ejecutar si la condición se cumple...  
  
} else {  
  
    // Bloque de código a ejecutar si la condición NO se cumple...  
  
}
```

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



# Sentencia else

Ejemplo:



```
var esFeriado = true;
```

```
if (esFeriado) {  
    alert("¡Hoy no se trabaja!");  
} else {  
    alert("Hay que trabajar...");  
}
```

¿Cuál de los `alert` se muestra? ¿Podría no mostrarse ninguno? ¿Podrían mostrarse los dos?



# 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:

```
if (hora > 18) {  
    alert('Está cerrado, es tarde.');
```

```
} else if (hora < 9) {  
    alert('Está cerrado, es temprano.');
```

```
} else {  
    alert('Está abierto');
```

```
}
```



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 equipo = "Wanderers";
```

```
if (equipo === "Nacional" || equipo === "Peñarol") {  
    alert("¡Sos hincha de un cuadro grande!");  
} else {  
    alert("Sos hincha de un cuadro chico 😞");  
}
```

¿Qué alert se ejecuta?

# 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 filosofía de 5º de liceo.





# Arrays



# ¿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 de manipular los valores por separado, se manipula el conjunto.
- El largo de los arrays es variable y pueden contener elementos repetidos.
- 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*. Ver [más información](#).



# ¿Cómo se crea un Array?

## Método 1:

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

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

## Método 2:

Se utilizan las palabras `new` y `Array`, y se pasan los valores separados por comas `", "`.

```
var marcas = new Array("BMW", "Peugeot", "Chevrolet", "Subaru", "Nissan");
```

Ambos métodos son equivalentes, pero el más usado y simple es el primero.

# 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.

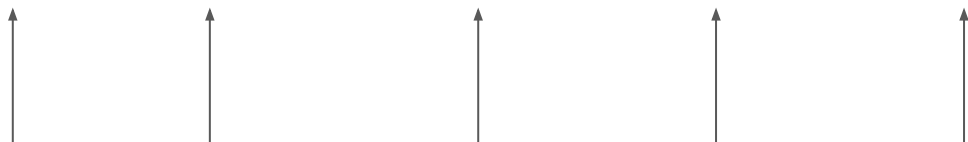
Esto varía según cada lenguaje. Los que comienzan en 0 suelen conocerse como *zero-index Array*.



# Acceder a un elemento de un Array

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

Índice:      0            1            2            3            4



```
var marcaPreferida = marcas[2];
```

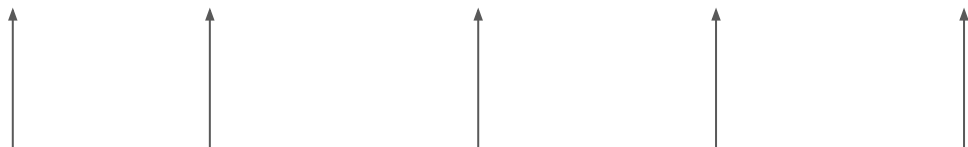
¿Cuál es el valor de `marcaPreferida`?



# Agregar un elemento a un Array

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

Índice:      0            1            2            3            4



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

¿Qué hubiese pasado si se hacía: `marcas[1] = "Volvo";`?



# Largo de un Array

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

Índice:      0            1            2            3            4



```
console.log(marcas.length);
```

¿Cuál es el valor de la propiedad `length`?



# Métodos de un Array

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

Todo array en JavaScript viene con una serie de métodos muy útiles. Ejemplos:

```
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.  
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](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Methods_2)





# Recorrer un Array

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

```
for (var i = 0; i < marcas.length; i++) {  
    console.log(marcas[i]);  
}
```

Otra forma:

```
marcas.forEach(function(item) {  
    console.log(item);  
});
```

Notar que estas dos formas no son totalmente equivalentes. El `forEach` se usa mucho por su facilidad de uso, pero es menos "poderoso". El `forEach` sólo permite recorrer el array de una forma. En cambio el `for` es más personalizable.



# for Loops



# for Loops (1/3)

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 (2/3)



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 (3/3)

👉 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.



# while Loops



# while Loop (1/3)

El `while` loop se parece a una sentencia `if`.

En ambos casos se ejecuta un bloque de código asociado dependiendo de una condición. La diferencia está en que el `while` ejecuta el código **repetidamente hasta que la condición deje de cumplirse**.

```
if (CONDICIÓN) {  
    // Bloque de código que se ejecuta  
    // una vez si la condición se  
    // cumple.  
}
```

```
while (CONDICIÓN) {  
    // Bloque de código que se ejecuta  
    // repetidamente mientras  
    // la condición se cumpla.  
}
```

El `while` se ejecuta mientras la condición se cumpla, o dicho de otro modo, hasta que la condición sea `false` (falso).



## while Loop (2/3)

Ejemplo:

```
while (tempSalon > 23) {  
    console.log("Aire acondicionado prendido.");  
    // Largar aire frío.  
    // Medir temperatura nuevamente.  
}
```





## while Loop (3/3)

Otro ejemplo:

```
var contador = 0;
while (contador < 50) {
    console.log(contador);
    contador = contador + 2;
}
```

Si bien este ejemplo es correcto, hubiese sido una mejor práctica resolverlo con un `for`.

Nota: Es usual que los `while` vayan acompañados de un contador. ¿Qué pasaría si se omitiese la línea `contador = contador + 2; ?`



# Objetos



# ¿Qué es un objeto? (JavaScript)

Un objeto es una **entidad** que contiene una **colección de propiedades**.

Una propiedad es una asociación de clave-valor.

Las propiedades son similares a las variables comunes de JavaScript, la diferencia es que las propiedades son variables que están unidas a un determinado objeto.

El valor de una propiedad puede ser una función y en ese caso a la propiedad se le llama **método**.

```
var nombreDelObjeto = {  
  clave1: valor1,  
  clave2: valor2,  
  clave3: valor3  
};
```

Propiedades. (Cada propiedad se separa con una coma ",").

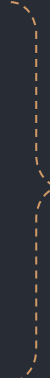
Las claves y valores se separan con dos puntos ":".



# El concepto de *objeto* es similar al de la vida real...

Ejemplo: un **auto**. Básicamente es un objeto que tiene un montón de propiedades como: color, peso, marca, modelo, velocidad máxima, kilometraje, etc.

```
var auto = {  
  marca      : "VW",  
  color      : "gris",  
  peso       : 1200,  
  velmax     : 220,  
  kilometraje : 45000  
};
```



Propiedades



# Acceder a una propiedad de un objeto

Dado el objeto llamado `persona1`:

```
var persona1 = {  
  nombre      : "María",  
  apellido    : "Rodríguez",  
  edad        : 36,  
  email       : "maria.rod@gmail.com",  
  nacionalidad : ["Uruguay", "España"]  
};
```

En este caso, el valor de la propiedad es un array. De hecho, las propiedades pueden tener cualquier tipo de valor, incluso otros objetos.

Se accede sus propiedades de esta forma:

```
persona1.apellido; // Devuelve "Rodríguez".  
persona1.edad;     // Devuelve 36.
```



# Métodos

Cuando una propiedad recibe como valor una función, se le llama método.

```
var persona1 = {  
  nombre      : "María",  
  apellido    : "Rodríguez",  
  edad        : 36,  
  email       : "maria.rod@gmail.com",  
  nacionalidad : ["Uruguay", "España"],  
  nombreCompleto : function () {  
    return this.nombre + " " + this.apellido;  
  }  
};
```

La palabra `this` es una palabra reservada del lenguaje. En JavaScript tiene un comportamiento muy especial, y por lo tanto se recomienda leer estos [docs de MDN](#).

```
persona1.nombreCompleto(); // Retorna "María Rodríguez"
```



# Buenos hábitos



# Buenos hábitos (1/3)

- Cuidar la **indentación** de sus archivos. Apóyense en su editor de código de preferencia (ejemplo: [Visual Studio Code](#)) y/o plugins como [Prettier](#).
- Usar espacios y saltos de línea de forma adecuada.
- Recordar usar los “;” al final de cada sentencia.
- Usar nombres descriptivos para variables, funciones y parámetros, preferentemente en inglés.

## MALA PRÁCTICA

```
var x=10,y=5
console.log(x+y)
var ciudad="Montevideo"
window.alert("Hola "+ciudad+".")
```

## BUENA PRÁCTICA

```
var ageChild1 = 10;
var ageChild2 = 5;
console.log(ageChild1 + ageChild2);
var city = "Montevideo";
window.alert("Hola " + city + ".");
```





## Buenos hábitos (2/3)

MALA PRÁCTICA

```
var x=10; var y=5;  
if(x==y){console.log("¡Misma edad!")}else{console.log("Edades diferentes")}
```

BUENA PRÁCTICA

```
var ageChild1 = 10;  
var ageChild2 = 5;  
  
if (ageChild1 === ageChild2) {  
    console.log("¡Misma edad!");  
} else {  
    console.log("Edades diferentes");  
}
```

# Buenos hábitos (3/3)

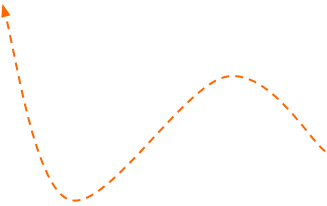


```
function calc_s() {  
    // Código de la función...  
}
```

MALA PRÁCTICA

```
function calcularSueldo() {  
    // Código de la función...  
}
```

BUENA PRÁCTICA



Sólo con mirar el nombre, no queda claro qué hace esta función.