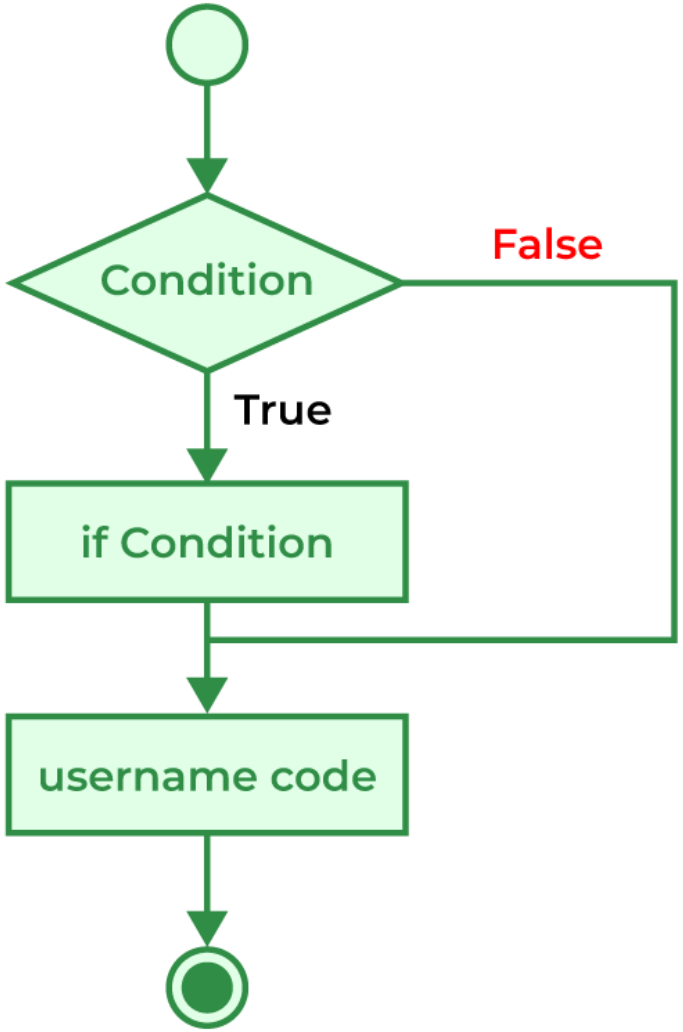
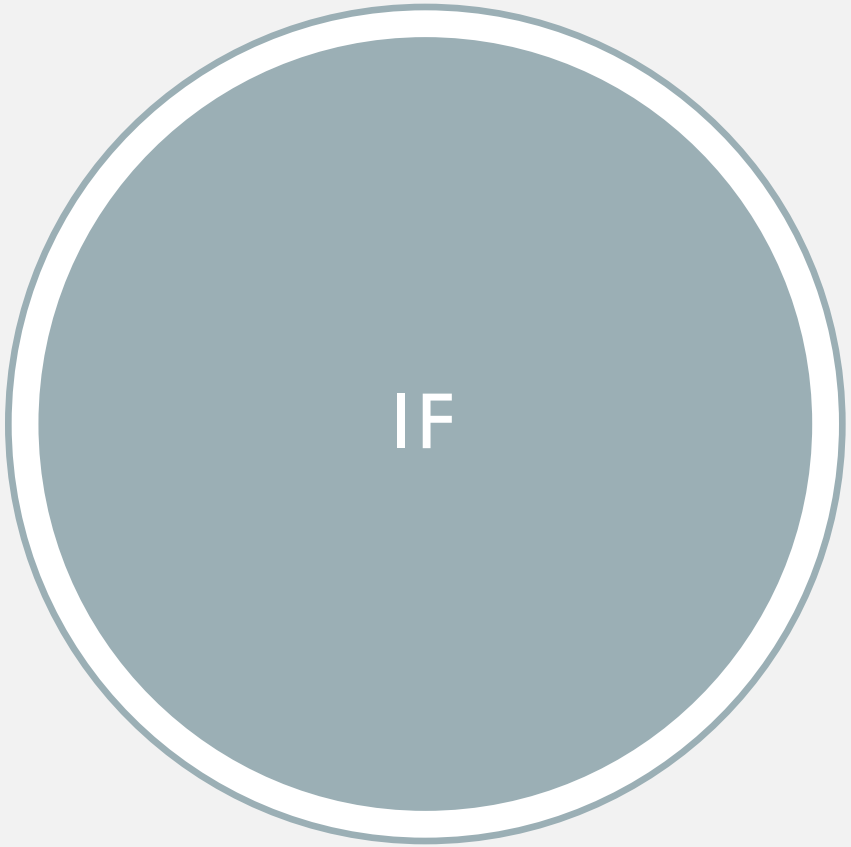

# REPASO JAVASCRIPT BÁSICO

Para ifs, funciones, Arrays, métodos de Arrays y objetos.

# ¿QUÉ ES UN IF?

Explicación teórica del If y las condiciones definidas con truthys, falsys y comparadores






- Un if es una instrucción en programación que sirve para tomar decisiones.
- Funciona como una pregunta que le hacemos a la computadora: "¿Esto es cierto o no?".
- Si la respuesta es sí, la computadora ejecuta cierto código.
- Si la respuesta es no, ignora ese código o puede ejecutar una alternativa (else).
- El if se usa para que el programa haga algo sólo si se cumple cierta **condición**

```
if (condicion) {

    // Código si se cumple la condición

}   else    {

    // Código si no se cumple la condición

}
```

# SINTAXIS DE UN IF

# ¿CÓMO DEFINO UNA **CONDICIÓN**?

- Cuando escribimos una condición dentro de un if, la computadora evalúa si la condición es verdadera o falsa.
- Para eso, hay dos conceptos importantes en JavaScript:
  - Los valores **Truthy** *La condición se cumple*
  - Los valores **Falsy** *La condición no se cumple*
- La condición es quien va entre paréntesis en el if

# ¿QUÉ SON LOS TRUTHYS Y FALSYS?

**Valores Truthy.**

- Cualquier valor que **no** sea :
  - ***false***
  - El número 0
  - La cadena vacía “ ”
  - ***Null***
  - ***Undefined***
  - ***NaN***
- Ejemplos de valores "truthy" incluyen números distintos de cero, cadenas con texto, y objetos.

**Valores Falsy**

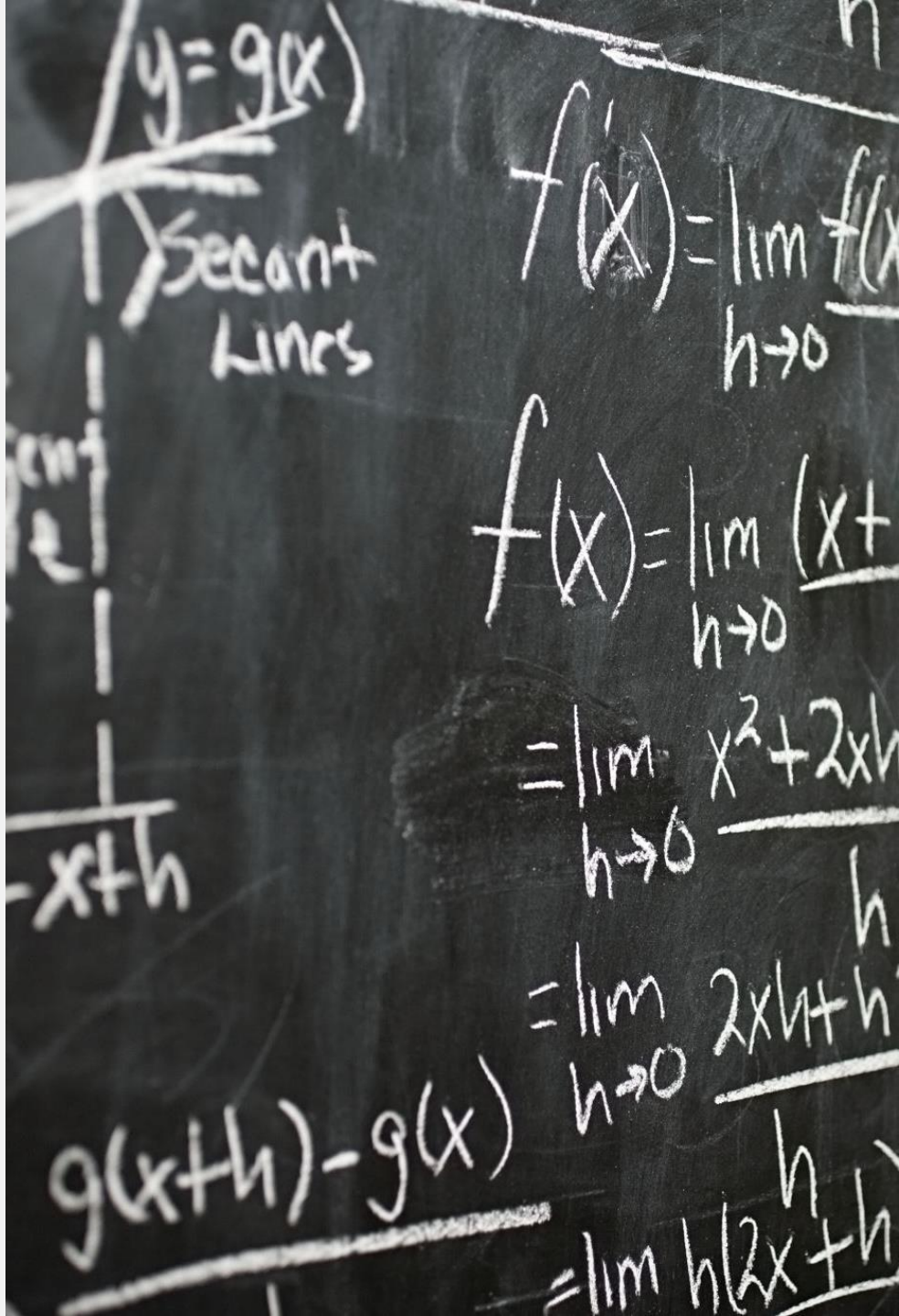
- Es todo lo que no sea un truthy
- Si la condición en un if tiene un valor "falsy", entonces el código en el bloque del if no se ejecuta.
- En su lugar, se puede ejecutar el bloque **else** si está presente.

# OPERADORES DE COMPARACIÓN

Operadores lógicos que comparan 2 variables u valores para crear un truthy o falsy

# ¿QUÉ ES UN OPERADOR COMPARADOR?

- Los operadores de comparación permiten comparar dos valores y devuelven un resultado "truthy" o "falsy".
- Por ejemplo, 5 > 3 devuelve true porque 5 es mayor que 3.
- Todos los comparadores sólo comparan dos valores o variables
- Se puede usar un paréntesis para poner precedencias



| Operador | Descripción | Ejemplo | Resultado |
|---|---|---|---|
| `==` | Igualdad débil | `5 == "5"` | `true` |
| `===` | Igualdad estricta | `5 === "5"` | `false` |
| `!=` | Desigualdad débil | `5 != "5"` | `false` |
| `!==` | Desigualdad estricta | `5 !== "5"` | `true` |
| `>` | Mayor que | `7 > 5` | `true` |
| `<` | Menor que | `3 < 5` | `true` |
| `>=` | Mayor o igual | `5 >= 5` | `true` |
| `<=` | Menor o igual | `4 <= 5` | `true` |

# TABLA DE OPERADORES COMPARADORES

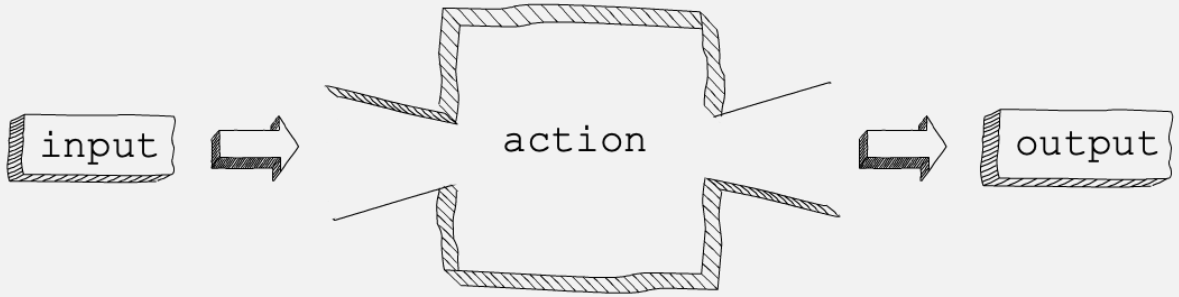
Nota: No hay que confundir el operador igualdad (==) con el operador asignación (=) que se usa para asignar un valor a una variable.

# ¿CÓMO PUEDO USAR UN IF Y UN ELSE?

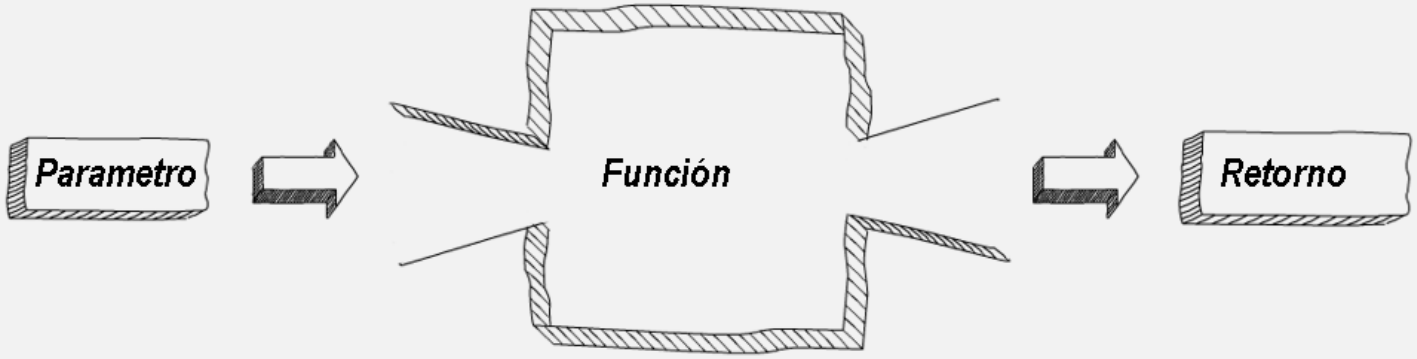
```
let edad = 16;

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

# ¿QUÉ ES UNA FUNCIÓN?

- Al igual que las funciones matemáticas, las funciones en Javascript son un conjunto de acciones que (generalmente) transforma información, con algunas particularidades que explicaré a continuación. Pueden pensarla como una función de una curva f(x) = y.



# PARTES DE LAS FUNCIONES




```
function nombreDeFuncion(parametro) {

    //Se transforma la información ingresada por parámetro
    const retorno = parametro + 5

    return retorno
}
```

# ¿TODAS LAS FUNCIONES DEBEN TENER UN PARÁMETRO Y RETORNO?

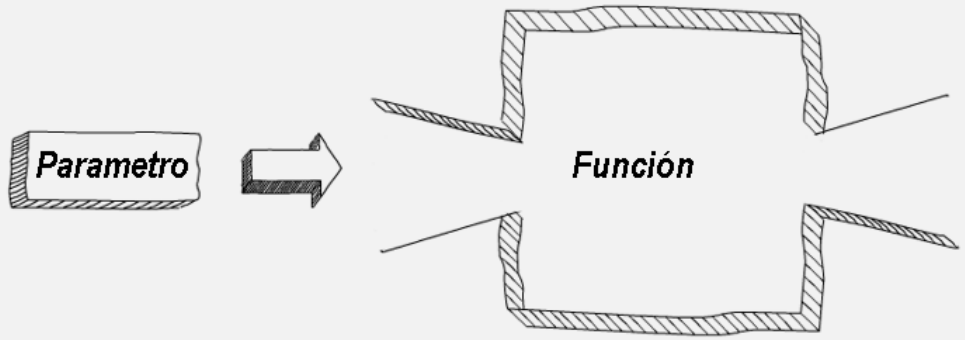
A diferencia de una función matemática, las funciones en programación no necesariamente debe ser un flujo de datos en donde se ingresa, transforma y devuelve algo.

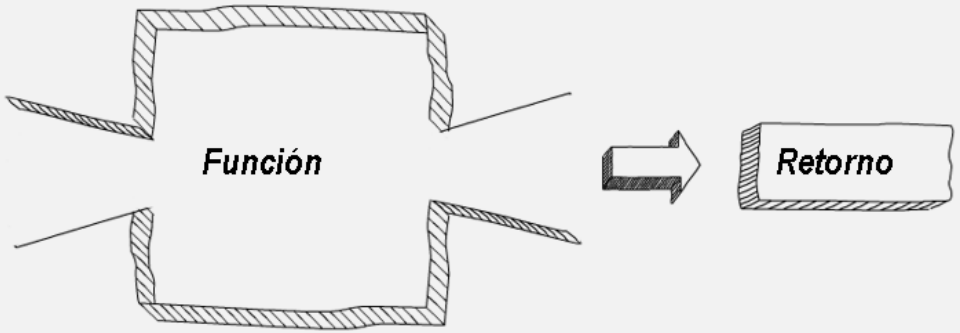
Tener o no parámetros o retorno no es obligatorio.

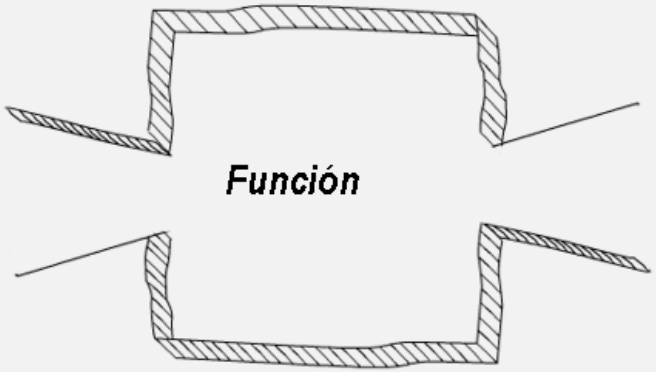
```
function mostrarEnPantalla(dato) {
    alert(dato)
}
```

```
function conseguirUsuariosBd() {
    const retorno = fetchBd("usuarios")
    return retorno
}
```

```
function refrescarPágina() {
    location.reload();
}
```



# ¿CUÁLES SON LOS TIPOS DE FUNCIONES?

Javascript tiene muchísimos tipos de funciones distintas, pero aquí veremos las 3 más importantes que hay que saber utilizar.

# FUNCIONES DEFINIDAS

- Son los tipos más comunes de funciones en todos los lenguajes de programación.
- En Javascript, ahorra el paso de declarar una variable y asignarle el valor de la función, ya que ese paso está implícito.
- Estas declaraciones son equivalentes, pero escritas de distintas formas.
  (En la primera no me entraba si escribía los 3 parámetros)

```
function declaracionDeFuncion(/* Parametros */) { return "Retorno" }

function declaracionDeFuncion(
    parametro1,
    parametro2,
    parametroN
) {
    return "Retorno"
}

function declaracionDeFuncion(parametro1, parametro2, parametroN) {
    return "Retorno"
}
```

# FUNCIÓN FLECHA

- Segundo tipo más común de funciones en todo Javascript.

- Muy útiles cuando se quiere guardar una función en un objeto (método) o pasar una función como parámetro a otra función.

- Las funciones, por lo general, se verán de la siguiente forma:

  () => {}

```
const funcion1 = (parametros) => { return "Retorno" }

let funcion2
funcion2 = (parametros) => { return "Retorno" }

/* Variable es igual a una función */
```

# FUNCIÓN FLECHA CON RETORNO IMPLÍCITO

- Un subtipo de la función flecha
- No es necesario escribir return
- No es necesario usar llaves
- Muy utilizado para los métodos de los Arrays o vectores que veremos a continuación (map, filter, forEach, etc)
- No hace falta poner paréntesis si hay un solo parámetro para la función

```
const funcionRetornoImplicito = () => "Retorno"

let funcionRetornoImplicito2
funcionRetornoImplicito2 = (parametros) =>
    "Retorno"

/* Variable es igual a una función */
```

```
const funcion = parametroÚnico =>
    "Mi único parametro es " + parametroÚnico

const sumar = (x, y) =>
    x + y
```

# EN RESUMEN

- Acuérdense del nombre y cómo definir estas 3 funciones y con eso les va a alcanzar para entender la sintaxis de lo que viene a continuación

```
const flecha = (parametro) => { return "Retorno" }

function declaracion(parametro) { return "Retorno" }

const retornoImplicito = (parametro, parametro2) => "Retorno"

const retornoImplicito2 = parametro => "Retorno"
```

# ¿QUÉ ES UNA FUNCIÓN INTERNAMENTE?

- Al igual que los números, arrays y strings, las funciones son **un tipo de dato como cualquier otro** que se guarda en una variable.
- Como consecuencia, nosotros podemos diferenciar 2 formas de llamar a una función:

  1) Por definición (No ponemos paréntesis)
  2) Por invocación (Ponemos paréntesis)

- En la siguiente diapositiva, mostramos ambas con esta función de aquí:

```
function funcionEjemplo() {
    return "Hola Mundo!"
}
```

# DIFERENCIAS INVOCACIÓN Y DEFINICIÓN

## DEFINICIÓN

- La utilizamos para enviar funciones por parámetro *(callbacks)* lo cual veremos más adelante en clases posteriores

```
console.log(funcionEjemplo)
ƒ funcionEjemplo() {
    return "Hola Mundo!"
}
```

## INVOCACIÓN

- La utilizamos para ejecutar las líneas de código dentro de la función. La invocación de una función es igual a su retorno.

```
console.log(funcionEjemplo())
Hola Mundo!
```

# OBJETOS

Un objeto en JavaScript es una colección de datos y **métodos** (funciones). Se usa para agrupar información relacionada.

Los datos pueden ser de ***CUALQUIER*** tipo.

Un objeto en sí, también es un tipo de dato.

# ¿CÓMO DEFINO A UN OBJETO?

- Un objeto en Javascript es un tipo de dato como cualquier otro que puede ser asignado a una variable, pasado como parámetro, guardado en un array o retornado desde una función.
- Se caracterizan por estar definidos entre llaves y tener duplas de **clave: valor** separadas por comas.
- No confundir con cualquier tipo de función en algún retorno o asignación.

{ } Es un **Objeto**
() => {} Es una **Función**

```
const pedro = {
    nombre: "Pedro",
    apellido: "Molina",                 // String
    edad: 21,                           // Número
    comidasFavoritas: ["Arroz"],        // Array (Vector)

    mascota: {                          // Objeto definido dentro de otro objeto
        nombre: "Pancho",
        especie: "Perro",
        edad: 4,
        saludar: () => { alert ("Guau!") }
    },

    saludar: () => { alert("Hola!") } // Función Flecha
}
```

# ALTERNATIVAMENTE, SE PUEDE DEFINIR A PEDRO Y PANCHO ASÍ

```
const pancho = {
    nombre: "Pancho",
    especie: "Perro",
    edad: 4,
    saludar: () => { alert ("Guau!") }
}

const pedro = {
    nombre: "Pedro",
    apellido: "Molina",                 // String
    edad: 21,                           // Número
    comidasFavoritas: ["Arroz"],        // Array (Vector)
    mascota: pancho,                    // Objeto Asignado a variable
    saludar: () => { alert("Hola!") },  // Función Flecha

    presentarse: function() {           // Función Tradicional
        alert("Me llamo " + this.nombre)   // Las funciones flechas no pueden usar "this"
    }
}
```

# ¿CÓMO ACCEDO A LOS VALORES DE UN OBJETO?

- Para acceder a la información guardada dentro de un objeto, se debe concatenar la variable del objeto, un punto y la clave del valor correspondiente.
- Si se quiere acceder a un método, se hace de la misma forma, excepto que además de tener la clave se debe agregar un paréntesis con los parámetros del método.
- Accediendo a las variables de esta forma, se pueden modificar los valores del objeto o agregar nuevos

```
> pedro.presentarse()
  Me llamo Pedro
< undefined
> pedro.edad
< 21
> pedro.mascota.especie
< 'Perro'
> pancho.especie
< 'Perro'
> pancho.nombre
< 'Pancho'
```

# MODIFICANDO Y AGREGANDO VALORES DEL OBJETO

```
> pedro.altura
< undefined
> pedro.altura = 1.84
< 1.84
> pedro.altura
< 1.84
```

```
> pedro.saludar = () => {
      console.log("Me cansé de saludar!")
  }
> pedro.saludar()
  Me cansé de saludar!
```

```
> pedro.nombre = "Ramiro"
< 'Ramiro'
> pedro.nombre
< 'Ramiro'
```

```
> pedro.mascota.saludar = () => {
      console.log("Me cansé de saludar yo también!")
  }
> pancho.saludar()
  Me cansé de saludar yo también!
```

# ARRAYS Y SUS MÉTODOS

También llamados vectores

# ¿QUÉ ES UN ARRAY?

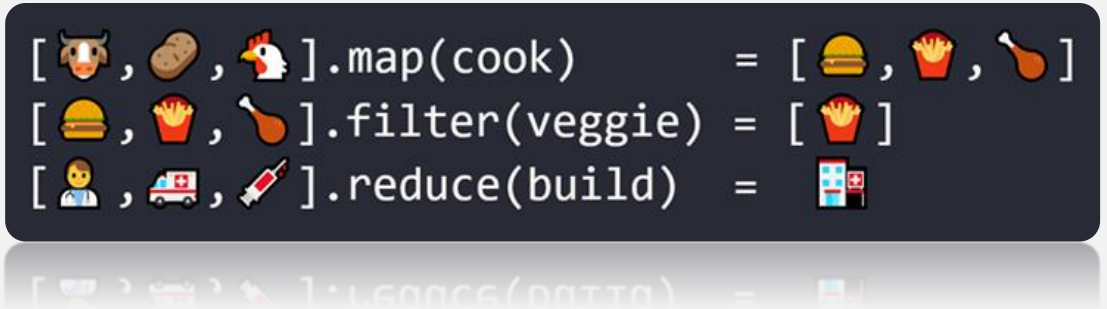
- Un array en JavaScript es una estructura de datos que permite almacenar múltiples valores en una sola variable.
- Los arrays se pueden crear usando corchetes [ ] y los elementos se separan por comas.
- Cada elemento en el array tiene un índice, que comienza en 0.

```
let numeros = [1, 2, 3, 4, 5]

let letras = ["a", "b", "c", "d", "e"]

let objetos = [pedro, pancho]

let mezcla = [1, "c", pancho, pancho, pancho, 5]
```

```
> letras[0]
< 'a'
> letras[3]
< 'd'
```

# ¿MÉTODOS DE ARRAYS?

- ¿No habíamos definido a los métodos como funciones dentro de objetos? ¿Cómo pueden tener métodos los Arrays?
- En javascript, la mayoría de los tipos de dato como las Funciones, Arrays u clases son realmente objetos disfrazados, los cuales guardan muchísimas duplas clave-valor, en las cuales se encuentran métodos como array.map(), array.forEach() u array.filter()
- Estos métodos se invocan de la misma manera que se invoca cualquier valor de un objeto según una clave



# ARRAY.FOREACH()

- Es un método que recibe como parámetro una **función flecha**
- Esta función flecha **recibe como parámetro una variable que puede tener cualquier nombre y representa al elemento**
- Lo que hace el forEach es **aplicar la función flecha a todos los elementos del array**
- La función flecha no necesita devolver nada necesariamente
- El método forEach no devuelve nada

```
let numeros = [1, 2, 3, 4, 5]

let suma = 0

numeros.forEach(
    elemento => {
        suma = suma + elemento
    }
);

console.log(suma)
```

```
15
```

# ARRAY.FILTER()

- La función que recibe por parámetro debe devolver verdadero o falso
- Para esto se pueden usar comparaciones de igualdad o mayor-igual (==, <, >, >=, <=)
- Devuelve un array
- Los elementos que devuelvan false no estarán en este nuevo array

```
let numeros = [1, 2, 3, 4, 5]

const filtro = numeros.filter(
    elemento => elemento >=4
)

console.log(filtro)
```

# ARRAY.MAP()

- Generalmente, este método recibe como parámetro una **función flecha de retorno implícito.**
- La función de retorno implícito debe recibir un solo parámetro de cualquier nombre (generalmente "elemento"). Este parámetro simbolizará un elemento del array.
- Lo que retorne la función se convertirá en el nuevo elemento del array.
- Este método devuelve un nuevo array. Este nuevo array es el resultado de ejecutar la función flecha con cada uno de sus elementos, y guardarlos en el lugar correspondiente.

```
let numeros = [1, 2, 3, 4, 5]

let numerosMultiplicados

numerosMultiplicados = numeros.map(
    elemento => elemento * 10
)
```

```
numerosMultiplicados
▶ (5) [10, 20, 30, 40, 50]
```

# EJEMPLO ÚTIL DE ARRAY.MAP() CON OBJETOS

```
const objetos = [ pedro, pancho ]

let nombreYEdad = objetos.map(

    //Si se quiere redefinir un objeto de esta forma los paréntesis son importantes
    elemento => ({ nombre: elemento.nombre, edad: elemento.edad })

)
```

```
nombreYEdad
▼ (2) [{…}, {…}] i
  ▶ 0: {nombre: 'Pedro', edad: 21}
  ▶ 1: {nombre: 'Pancho', edad: 4}
    length: 2
```

# ¿DÓNDE PUEDO ENCONTRAR MÁS MÉTODOS?

- Como la mayoría de las cosas, la programación se basa en la investigación propia. Uno nunca puede terminar de aprender todo, especialmente en un campo que está en constante cambio como este. Lo más recomendable es nunca quedarse con sólo las cosas que te enseñan (o quizá las cosas que ***no*** te enseñan) en la secundaria.

  Los mejores programadores investigaron por su propia cuenta y entrenaron sus proyectos a partir de su propia creatividad.

  Experimenten, prueben y aprendan. Nunca está de más saber muchas cosas.

# LINKS ÚTILES PARA MÉTODOS DE ARRAYS

https://www.w3schools.com/js/js_array_methods.asp

https://www.w3schools.com/js/js_array_iteration.asp