

ETAPA 1 Uso GitHub

<https://forgoodfirstissue.github.com/>

CLONE QUE OCUPA MENOS: `--depth = 1`

Clona repositorio desde el último commit (profundidad 1).

FORK: copia total del repositorio y cambiarle el dueño. Se copian commits también.

CLONE: copiarlo con los mismos permisos y obtener el histórico.

si una página es MIT se puede distribuir y todo.

FORMAS DE CLONAR:

HTTPS (no recomendado): pide usuario y contraseña.

SSH (recomendado): configurar la llave de SSH

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account> → Conexión con SSH → Generación de una nueva llave SSH

CLI: <https://cli.github.com/>

GitHub Flow: PR pequeñas pero frecuentes.

ETAPA 2 Desarrollo Backend sin Frameworks (Node.js y TypeScript)

JavaScript (necesario para aprender TypeScript)

Primitivos: números, cadenas de texto y booleanos.

Crear variable → `let number = 4`

`let texto = "Hola"`

`let isUserLogged = true`

`let number = 10 → let otroNumero = number → otroNumero = 10`

Operador typeof → `const MAGIC_NUMBER = 7`

`typeof MAGIC_NUMBER // "number"`

`typeof undefined // "undefined"`

`typeof true // "boolean"`

`typeof 42 // "number"`

`typeof "Hola mundo" // "string"`

Mostrar en consola (`console.log()`) → `console.log("Hola, JavaScript")`

`// -> 'Hola, JavaScript'`

`const nombre = 'JavaScript'`

`console.log(nombre)`

`// -> 'JavaScript'`

OBJETOS

Ejemplo 1

```
const name = 'PS5'  
const price = 550  
const company = 'Sony'
```

describo el objeto con las propiedades que tiene (llave y valor)

```
const gameSystem = {  
  name = 'PS5',  
  price = 550,  
  company = 'Sony',  
  games: ['God of War', 'Last of Us']  
  specs: {  
    cpu = 'AMD',  
    gpu = 'AMD',  
    ram = 16,  
    disk = 800  
  },  
  
  runGame (game){  
    console.log(game)  
  }  
}
```

```
console.log(gameSystem.name)  
PS5
```

```
console.log(gameSystem.price * 2)  
1100
```

```
gameSystem.games[1]  
'Last of Us'
```

```
gameSystem.specs.ram  
16
```

```
gameSystem.runGame('Elden Ring')  
Elden Ring
```

```
const propertyName = 'company'  
gameSystem[propertyName] → gameSystem['company'] → gameSystem.company
```

Ejemplo 2

```
const persona = {  
  'nombre completo': 'Miguel Ángel'  
}
```

```
persona['nombre completo']  
'Miguel Ángel'
```

```
const key = 'nombre completo'  
persona[key]  
'Miguel Ángel'
```

ARRAYS

```
const frutas = ["manzana", "pera", "plátano", "fresa"]  
console.log(frutas.length) // 4
```

```
const frutas = ["manzana", "pera", "plátano", "fresa"]  
frutas.length = 2 // corta el array a 2
```

```
console.log(frutas) // ["manzana", "pera"]  
console.log(frutas.length) // 2
```

.push() nos permite **añadir** un elemento al **final** de un array:

```
const frutas = ["plátano", "fresa"]  
frutas.push("naranja")  añade un elemento a la lista  
console.log(frutas) // ["plátano", "fresa", "naranja"]
```

.pop() **elimina** y devuelve el **último** elemento de un array:

```
const frutas = ["plátano", "fresa", "naranja"]  
const ultimaFruta = frutas.pop()  elimina y devuelve el último elemento de un array:
```

```
console.log(frutas) // ["plátano", "fresa"]  
console.log(ultimaFruta) // "naranja"
```

.shift() **elimina** y devuelve el **primer** elemento de un array. Lo mismo que **.pop()**, pero con el primer elemento en lugar del último:

```
const frutas = ["plátano", "fresa", "naranja"]  
const primeraFruta = frutas.shift()
```

```
console.log(frutas) // ["fresa", "naranja"]  
console.log(primeraFruta) // "plátano"
```

.unshift() **añade** un elemento al **principio** de un array. Lo mismo que **.push()**, pero con el primer elemento en lugar del último:

```
const frutas = ["plátano", "fresa", "naranja"]  
frutas.unshift("manzana")
```

```
console.log(frutas) // ["manzana", "plátano", "fresa", "naranja"]
```

.concat() concatena dos arrays:

```
const numbers = [1, 2, 3]
```

```
const numbers2 = [4, 5]
```

```
const allNumbers = numbers.concat(numbers2)
```

```
console.log(allNumbers) // [1, 2, 3, 4, 5]
```

ITERACIÓN

```
let frutas = ['🍎', '🍌', '🍓']
```

```
let i = 0 // lo usaremos como índice
```

```
while (i < frutas.length) {
```

```
  console.log(frutas[i])  imprime el elemento en la posición i
```

```
  i++  incrementamos el índice en 1 para la siguiente iteración
```

```
}
```

```
let frutas = ['🍎', '🍌', '🍓']
```

```
for (let i = frutas.length - 1; i >= 0; i--) {
```

```
  console.log(frutas[i])  imprime el elemento en la posición i
```

```
}
```

FUNCIONES

Ejemplo 1

```
function suma() {
```

```
  return 1+1
```

```
}
```

```
const resultado = suma()
```

```
console.log(resultado)
```

Math.random(): devuelve un número aleatorio entre 0 y 1, con decimales.

Math.floor(): redondea un número hacia abajo.

```
function getRandomNumber() {
```

```
  // recuperamos un número aleatorio entre 0 y 1
```

```
  const random = Math.random() // por ejemplo: 0.6803487380457318
```

```
  // lo multiplicamos por 10 para que esté entre 0 y 10
```

```
  const multiplied = random * 10 // -> 6.803487380457318
```

```
  // redondeamos hacia abajo para que esté entre 0 y 9
```

```
  const rounded = Math.floor(multiplied) // -> 6
```

```
// le sumamos uno para que esté entre 1 y 10
const result = rounded + 1 // -> 7

// devolvemos el resultado
return result
}
```

PARÁMETROS

```
function saludar(nombre) {
  console.log("Hola, " + nombre)
}
saludar("Pablo")
Hola, Pablo
```

Ejemplo 2

```
function sumar (a, b){  lo que espera la función son parámetros (a, b)
  return a + b
}
```

`sum(2, 3)` lo que se le pasa cuando llamamos a la función se llaman **argumentos** (2, 3)
5

FUNCTION EXPRESSION

```
// esto es una function expression
const sum = function (a, b) {
  return a + b
}
sum(1, 2) // 3
```

```
// esto es una declaración de función
function sum(a, b) {
  return a + b
}
```

TypeScript

TypeScript online:

<https://www.typescriptlang.org/play/?#code/MYewdgziA2CmB00QHMAUAiAEjAhgAgBUBPA B1gGVgAnASxIBd0BKAKDzyA>

Sintaxis: <https://www.typescriptlang.org/docs/handbook/2/basic-types.html> (muy parecido a JS)

VARIABLES

```
1 var myString = "Esto es una cadena de texto"
2 console.log(myString)
3
4 let myString2 = "Esto es una cadena de texto"
5 myString2 = "Aquí cambio el valor de la cadena de texto"
6 console.log(myString2)
```

```
[LOG]: "Esto es una cadena de
texto"
```

```
1 var myString = "Esto es una cadena de texto"
2 var myString = 5
3 /* ERROR, está inicializada como una cadena de texto */
4 console.log(myString)
5
```

TIPADO FUERTE

```
6 let myString2 : string = "Esto es una cadena de texto"
7 myString2 = "Aquí cambio el valor de la cadena de texto"
8 myString2 = "6"
9 console.log(myString2) // imprime un 6 ya que entre "" se reconoce como string
10
11 let myNumber : number = 7
12 myNumber = myNumber + 4
13 console.log(myNumber) // imprime un número
14 console.log(typeof myNumber) // imprime number (el tipo de la variable)
15
```

con console.log se pueden concatenar sin importar el tipo.

FUNCIONES

```
16 function myFunction(): string {
17     return "Mi función"
18 }
19 console.log(myFunction())
```

```
[LOG]: "Mi función"
```

```
3 function sumTwoNumbers(firstNumber: number, secondNumber: number): number {
4     return firstNumber + secondNumber
5 }
6 console.log(sumTwoNumbers(5, 10))
```

LISTAS, SET Y MAP

```
2 let myList: Array<string> = ["Raul", "Troya", "DAW"]
3 console.log(myList)
```

```
[LOG]: ["Raul", "Troya", "DAW"]
```

```
let mySet: Set<string> = new Set(["Raul", "Troya", "DAW"])
console.log(mySet)
```

```
[LOG]: Set (3) {"Raul",
"Troya", "DAW"}
```

El SET no admite repetidos, imprimirá la lista sin dichos valores.

```
2 let myMap: Map<string, number> = new Map([["Raul", 18]])
3 myMap.set = ("Raul", 18)
4 console.log(myMap)
```

```
[LOG]: Map (1) {"Raul" => 18}
```

BUCLES

```
for (const value of myList) {
  console.log(value)
}

let myCounter = 0

while(myCounter < myList.length){
  console.log(myList[myCounter])
  myCounter++
  // si no voy incrementando el contador, el bucle siempre es 0, por lo que es infinito
}
```

CLASES

```
1  class MyClass {  
2      name : string  
3      age : number  
4  
5      constructor (name: string, age: number){ //parametrizado  
6          this.name = name  
7          this.age = age  
8      }  
9  }  
10  
11  let myClass = new MyClass("Raul", 18)  
12  console.log(myClass.name)
```

En este caso imprime solo el nombre, que es lo que está seleccionado en el console.