TYPES

7 tipos primitivos

1. String
2. Number
3. Bigint (introducido en ECMASCcript 2020)
4. Boolean
5. Null
6. Undefined
7. Symbol
8. Object:
   - Array
   - Function

```
"Any Text"
123.45
BigInt("123412341234432112342112 34");
true or false
null
undefined
Symbol('something')
{ key: 'value' }
[1, "text", false]
function name() { }
```

Object

Un objeto es un tipo de dato en JavaScript que se utiliza para almacenar una combinación de datos en un par **key-value**.

```
var user = {
  name: "John Doe",
  yeatOfBirth: 1998,
  calculateAge: function () {
    // code
  }
}
```

Key
Estas son las claves (key) en el objeto `user`.

Value
Estos son los valores de las respectivas claves en el objeto `user`

Method
Si una clave tiene una función como valor, se le suele llamar método.

FUNCTION

Una función (***function***) es simplemente un conjunto de código incluido en una sección. Este conjunto de código **SÓLO** se ejecuta cuando se llama a la función. Las funciones permiten organizar el código en secciones y reutilizar el código.

Unar una función tiene solo 2 partes:

- Declarar/definir una función
- Ejecutar/Invocar una función

```
// Declarar la función
function someName(param1, param2) {

  let a = param1 + "love" + param2;
  return a;
}
// Invocar (ejecutar, llamar) función
someName("me", "you");
```

Parámetros (Opcional)

Opcionalmente una función puede tomar parámetros. Luego, la función puede usar esta información dentro del código.

Nombre de la función

Es solo el nombre que le das a tu función.
Tip: Haga que los nombres de sus funciones describa lo que hace la función

Bloque de código

Cualquier código entre llaves { ... } se denomina "bloque de código" o simplemente "bloque". Este concepto no se limita sólo a las funciones. Las "sentencias if", los "bucles for" entre otras.

Retorno (Opcional)

Opcionalmente una función puede retornar un valor cuando se invoca. Una vez que se retorna en una función, no se ejecutan más lineas de código dentro de la función

Invocar la función

Invocar, llamar o ejecutar una función significa lo mismo. Cuando escribimos el nombre de la función, en este caso `someName`, seguido del símbolo entre paréntesis (), se ejecuta el código dentro de la función.

Pasando argumentos

En el momento de invocar la función, los argumentos son los valores reales que se pasan a la función.

'THIS' KEYWORD

La palabra reservada `this` se utiliza dentro de una función. Esta palabra reservada es simplemente una referencia a otro objeto.

A qué se refiere la palabra reservada `this` depende del escenario o de la forma en que se implementa la función:

Escenario #1: `this` dentro de una función
La palabra `this` apunta a un objeto global.

Escenario #2: `this` dentro de un método
La palabra `this` apunta a un objeto en el que se encuentra el método.

Escenario #3: `this` cuando se ejecuta una función con `call`, `bind`, `apply`
Cuando se llama a una función utilizando el método `.call(param)`, `.bind(param)` o `.apply(param)`, el primer parámetro se convierte en el objeto al que se refiere la palabra reservada `this`.

Nota Importante:
En el navegador, `global` es el objeto `window`.
En Node.js `global` es el objeto `global`.

```
var name = "John";

function fun() {
  console.log(this.name);
}


const user = {
  name: "Mario",
  yearOfBirth: 1990,
  calcAge: function() {
    const currentYear = (new Date()).getFullYear();
    return currentYear - this.yearOfBirth;
  }
}




fun();           // 'this' es global
user.calcAge(); // 'this' es el objeto user
fun.call(user); // 'this' es el objeto user
```

PROMISE

¿Qué es una Promesa?

Promise es un objeto que proporciona una construcción útil cuando se trabaja con tareas asincrónicas.

Trabajar con promesas comprende dos partes:

- Crear una promesa.
- Usar una promesa.

¿Qué es una tarea asíncrona?

Una tarea asincróna es aquella en la que un proceso de terceros realiza una tarea.

Ejemplos:

- Solicitar/enviar datos a la base de datos.
- Solicitar/enviar datos vía protocolo HTTP.

```
// Crear una promesa
const p = new Promise((resolve, reject)) => {
  setTimeOut(() => {
    if(condition) {
      resolve('Successfull login')
    } else {
      reject('Login failed')
    }
  })
}
```

```
// Usando una promesa
p.then((res) => {
  console.log(res);
})
.catch((err) => {
  console.log(err);
})
```

Nota: El 90% de las veces trabajarás con promesas preexistentes. El paso de "crear una promesa" lo realizará la librería, framework o el entorno que esté utilizando. Ejemplo de promesas:

fetch, **axios**

```
fetch('/data.json')
  .then(res => res.json())
  .then(data => {
    console.log(data);
  })
  .catch(err => ...)
```

```
axios.get('https://api.sample.com/data')
  .then(res => {
    console.log(res.data);
  })
  .catch(err => ...)
```