

Curso de JavaScript Básico

Prerequisites

- ✓ Una cuenta en Github (<https://github.com/>)

Es completamente gratis! Github es una especie de red social para desarrolladores y un repositorio de código.

- ✓ Buena actitud de aprendizaje!

¿Por qué JavaScript?

- ✓ Es el lenguaje de programación más popular de la actualidad.
- ✓ Es el único lenguaje que entienden los navegadores.
- ✓ Gracias a una plataforma llamada `Node.js` es posible ejecutar JavaScript por fuera del navegador.
- ✓ Se pueden crear todo tipo de aplicaciones con JavaScript: aplicaciones de escritorio, aplicaciones móviles, aplicaciones para la línea de comandos, backends de aplicaciones Web, entre otros.

Primeros Pasos

- ✓ Ejecutar código en repl.it .
- ✓ Ver errores de sintaxis.
- ✓ Comentarios

Tipos y operadores básicos

✓ Números

Operadores aritméticos (suma, resta, multiplicación, división, módulo, exponentes), precedencia.

✓ Cadenas de texto (strings)

Definir, imprimir, concatenar

✓ Booleanos

Expresiones booleanas, && (y), || (ó) y ! (negación).

Variables

Las variables nos permiten almacenar información temporal que podemos usar más adelante en nuestros programas.

```
var nombre = "Pedro Perez"  
var edad = 20  
var casado = false  
  
console.log(nombre)  
console.log(edad)  
console.log(casado)
```

Nombramiento de variables

El nombre de una variable debe comenzar con \$, _ o una letra. Después puede contener letras, dígitos, _ y \$.

```
var primerNombre;  
var $edad = 20;  
var _c4s4do = false;
```



```
var 6nombre;  
var ed&ad = 20;
```



Obtener información del usuario

En el navegador podemos usar el comando `prompt` para pedirle información al usuario.

```
var nombre = prompt("¿Cuál es tu nombre?");  
console.log("Hola" + nombre);
```


Convirtiendo un string a un número

`prompt` siempre nos devuelve un string. Si queremos convertir el string a un número debemos usar `parseInt`.

```
var edad = prompt("¿Cuál es tu edad?");  
edad = parseInt(edad)
```

Condicionales

Las condicionales nos permiten tomar diferentes caminos en la ejecución del código de acuerdo a los criterios que definamos.

La sintaxis más simple de un condicional es la siguiente:

```
if (<condición>) {  
    // código que se ejecuta si se cumple la condición  
}
```

Condicionales (else)

Existen dos atajos que te van a permitir escribir código más corto: `else` y `else if`

```
if (<condición>) {  
    // código que se ejecuta si se cumple la condición  
} else {  
    // código que se ejecuta si NO se cumple la condición  
}
```

Condicionales anidados

```
var num = 8;

if (num < 10) {
  console.log("El número es menor a 10");
} else {
  if (num > 10) {
    console.log("El número es mayor a 10");
  } else {
    console.log("El número es igual a 10");
  }
}
```

Condicionales (else if)

Existen dos atajos que te van a permitir escribir código más corto: `else` y `else if`

```
if (<condición 1>) {  
    // código que se ejecuta si la condición 1 se cumple  
} else if (<condición 2>) {  
    // código que se ejecuta si condición 1 NO se cumple  
    // pero condición 2 se cumple  
} else if (<condición 3>) {  
    // código que se ejecuta si condición 1 y condición 2  
    // NO se cumplen pero condición 3 se cumple  
} else {  
    // código si ninguna de las condiciones se cumple  
}
```

Operadores lógicos

Podemos utilizar los operadores lógicos y (&&) y ó (||) para crear condiciones más complejas.

```
var num = 15;

if (num >= 10 && num <= 20) {
  console.log("El número está entre 10 y 20");
}
```

```
var num = 8;

if (num < 10 || num > 20) {
  console.log("El número está por fuera del rango");
}
```

Operador ternario

Nos permite escribir condicionales en una sola línea. Se recomienda utilizarlo únicamente en condicionales cortos.

```
<condición> ? <si verdadero> : <si falso>
```

```
var num = 10
```

```
console.log(num > 10 ? "Mayor" : "Menor o igual")
```

Generar un número aleatorio

Para generar un número aleatorio entre 0 (incluyendo) y 1 (excluyendo) utiliza `Math.random`:

```
var num = Math.random()
```

Para generar un número aleatorio entre 1 y 10 utiliza `Math.random` combinado con `Math.floor`:

```
var num = Math.floor(Math.random() * 10) + 1
```


Ciclos (while)

Los ciclos nos permiten repetir la ejecución de un trozo de código varias veces.

```
while (<condición>) {  
    // código que se ejecuta mientras se cumple  
    // la condición  
}
```

Ciclos (do ... while)

Si queremos ejecutar al menos una vez el bloque de código:

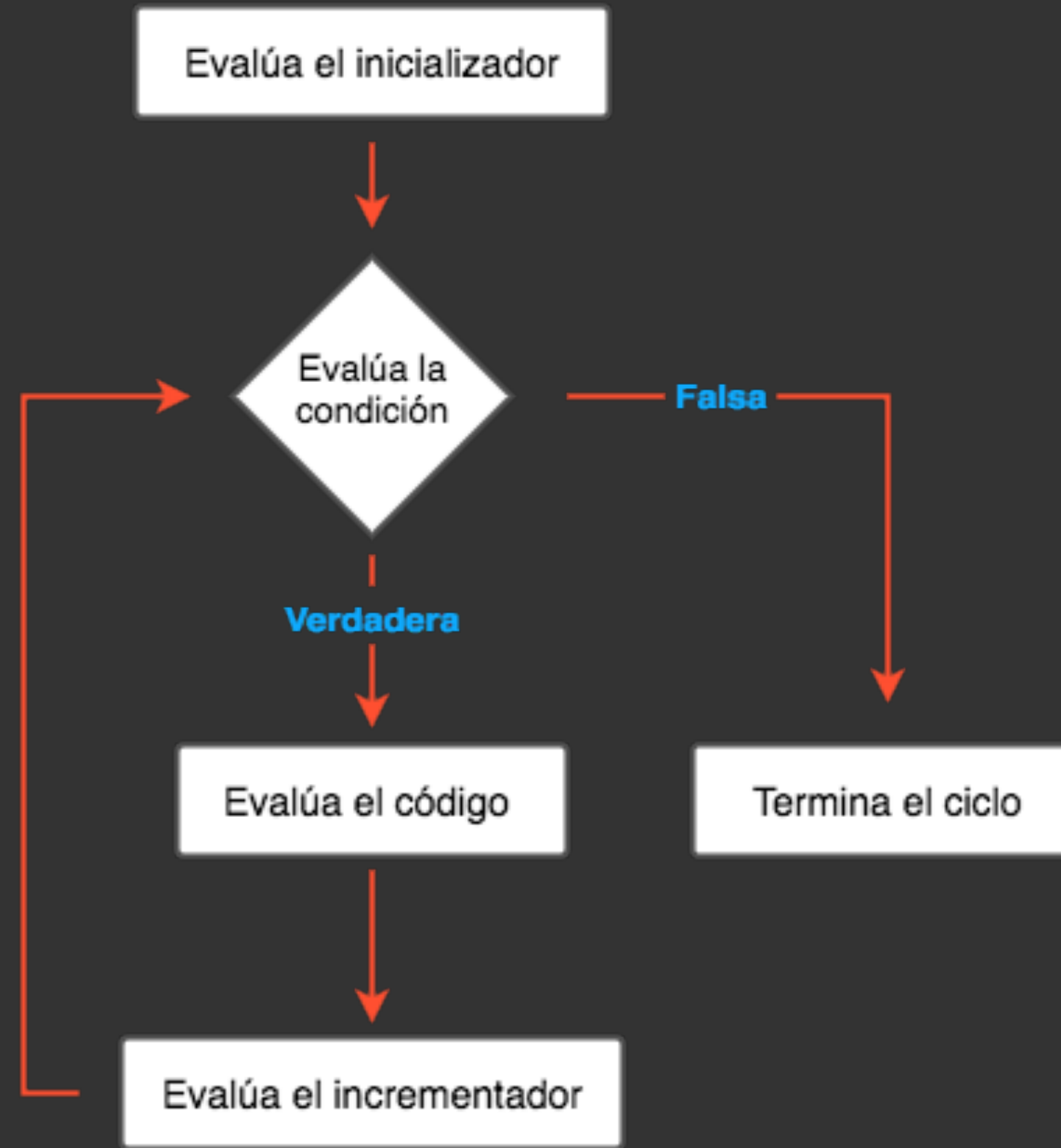
```
do {  
    // código que se ejecuta una vez y mientras se cumple  
    // la condición  
} while (<condición>)
```

Ciclos (for)

`for` es un atajo para el patrón de ciclos que requiere una combinación de inicialización, condición e incremento:

```
for (<inicializador>; <condición>; <incremento>) {  
    // código que se ejecuta mientras se cumple  
    // la condición  
}
```

Ciclos (for)



Arreglos

Un arreglo es una lista ordenada de elementos. Los elementos pueden ser de cualquier tipo: números, booleanos, strings, otros arreglos u objetos

```
const array = [1, "Pedro", true, [2, "Juan"]]
```

Arreglos (operaciones básicas)

Las operaciones básicas son:

- Obtener la longitud del arreglo.
- Obtener el valor de una posición.
- Recorrer el arreglo.
- Reemplazar un elemento.
- Insertar nuevos elementos.
- Eliminar elementos.
- Unir los elementos en un string.
- Separar un string en un arreglo (p.e. palabras o caracteres).

Operaciones con strings

Las operaciones más importantes con los strings son:

- Obtener la longitud.
- Recorrer un string.
- Dividir un string.
- Unir un arreglo en un string.
- Pasar a mayúsculas o minúsculas.
- Extraer una parte del string.
- Reemplazar parte del string.
- Validar si un string contiene una subcadena específica.

Funciones

Una función es un bloque de código reutilizable que vas a poder **invocar** desde otros lugares de la aplicación para realizar alguna tarea específica.

Una función se compone de un nombre, unos argumentos, un cuerpo y, opcionalmente, un valor de retorno.

```
function hola() {  
  console.log("Hola Mundo");  
}  
  
hola()
```


Funciones (argumentos)

Las funciones pueden recibir cero o más argumentos (también conocidos como parámetros).

```
function hola(name) {  
  console.log("Hola " + name);  
}  
  
hola("Pedro")  
hola("María")
```

Funciones (retorno)

Las funciones pueden, opcionalmente, retornar un valor utilizando la palabra clave `return`:

```
function hola(name) {  
    return "Hola " + name;  
}  
  
var g1 = hola("Pedro");  
console.log(hola("María"));
```

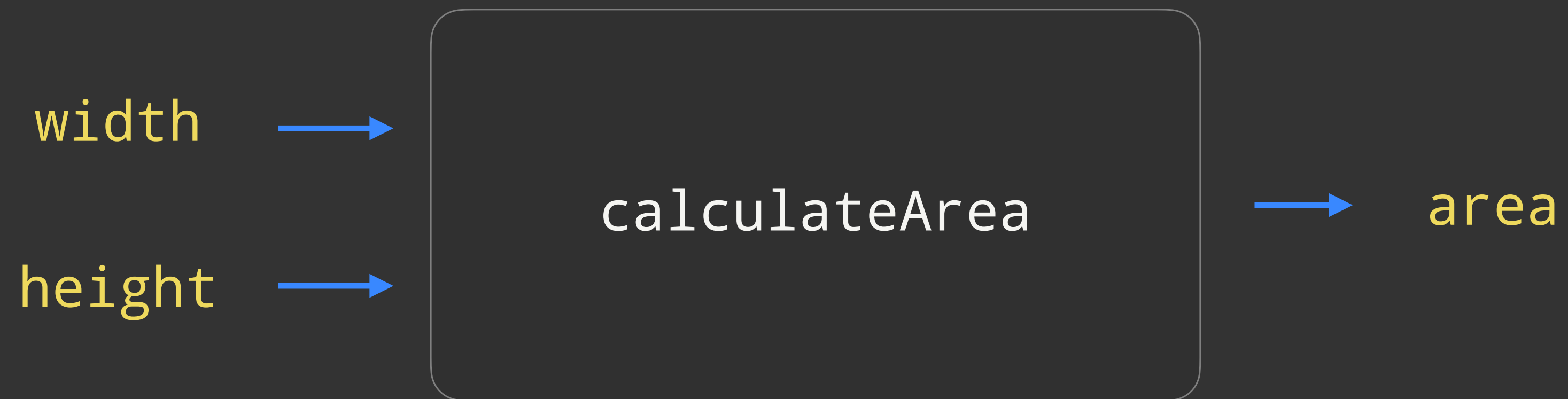
Anatomía de una función

The diagram illustrates the anatomy of a JavaScript function using the example: `function calculateArea(width, height) { var area = width * height; return area; }`. Brackets and labels identify the following parts:

- Palabra clave** (Keyword): `function`
- Nombre** (Name): `calculateArea`
- Argumentos** (Arguments): `width, height`
- Cuerpo** (Body): The code block inside the curly braces, including `var area = width * height;` and `return area;`
- valor de retorno** (Return value): The value being returned, `area`, indicated by a bracket under the `return` statement.

```
function calculateArea(width, height) {  
    var area = width * height;  
    return area;  
}
```

Funciones como cajas negras



Objetos literales

Los objetos nos ayudan a agrupar información para representar el mundo real.

Un objeto es un conjunto de propiedades. Cada propiedad está compuesta por llave y valor.

```
var persona = {  
  nombre: "Pedro",  
  apellido: "Perez",  
  edad: 20,  
  estatura: 1.8  
}
```

Diagram illustrating the structure of a JavaScript object literal:

- The object is defined as `var persona = {`.
- Properties are listed inside the curly braces: `nombre: "Pedro",`, `apellido: "Perez",`, `edad: 20,`, and `estatura: 1.8`.
- The closing brace is `}`.
- Annotations below the code identify the components of a property:
 - `nombre` is labeled as **Llave** (Key).
 - `"Pedro"` is labeled as **valor** (Value).
 - The entire pair `nombre: "Pedro"` is labeled as **Propiedad** (Property).

Objetos (operaciones básicas)

Las operaciones básicas son:

- Obtener el valor de una llave.
- Modificar el valor de una llave.
- Eliminar una propiedad de un objeto.
- Recorrer las propiedades de un objeto.

Objetos (métodos)

El valor de una propiedad de un objeto puede ser una función:

```
var persona = {  
  nombre: "Pedro",  
  saludar: function() {  
    console.log("Hola")  
  }  
}
```

Objetos (métodos)

Los métodos pueden utilizar otras propiedades del objeto utilizando la palabra clave `this`:

```
var persona = {  
  nombre: "Pedro",  
  saludar: function() {  
    console.log("Hola, soy " + this.nombre)  
  }  
}  
  
persona.saludar() // "Hola, soy Pedro"
```


Funciones constructoras

Las funciones constructoras nos permiten crear objetos a partir de una misma “plantilla”. Para diferenciar a las funciones constructoras de una función normal se capitaliza la primera letra.

Para instanciar (crear un objeto) a partir de una función constructora se utiliza la palabra clave `new`.

```
function Persona(name) {  
  this.nombre = name  
}  
  
const p1 = new Persona("Pedro")  
const p2 = new Persona("María")
```

Funciones constructoras (métodos)

Para agregar métodos a las funciones constructoras utilizamos el prototype.

```
function Persona(nombre) {  
  this.nombre = nombre  
}  
  
Persona.prototype.saludar = function() {  
  console.log("Hola, me llamo " + this.nombre)  
}  
  
const p1 = new Persona("Pedro")  
p1.saludar() // Hola, me llamo Pedro
```

Métodos estáticos

Los métodos estáticos son métodos que se pueden invocar directamente sobre la función constructora, no es necesario tener una instancia.

```
function Persona(nombre) {  
  this.nombre = nombre  
}  
  
Persona.saludar = function() {  
  console.log("Hola!")  
}  
  
Persona.saludar() // Hola!
```

Funciones constructoras de JavaScript

JavaScript incluye funciones constructoras como String, Number, Boolean, Array, Date, etc. Los tipos básicos y los arreglos también son creados a partir de estas funciones constructoras.

```
const a1 = [1, 2, 3, 4]
const a2 = new Array(1, 2, 3, 4)

const s1 = "Hola Mundo"
const s2 = new String("Hola Mundo")
```

Extendiendo la funcionalidad de JavaScript

Utilizando el `prototype` puedes agregarle métodos a los objetos nativos de JavaScript como `Array`, `String`, etc.

```
Array.prototype.sum = function() {  
    var total = 0;  
  
    for (var i=0; i < this.length; i++) {  
        total += this[i];  
    }  
  
    return total;  
}
```

Programación funcional

Un lenguaje se considera funcional cuando las funciones se pueden:

- Asignar a variables.
- Pasar como argumento de otras funciones.
- Retornar desde otras funciones.

Métodos funcionales en arreglos

- `forEach`
- `filter`
- `map`
- `reduce`

Parámetros por defecto

```
function makeRequest(url, timeout=2000, callback=function() {}){  
  // the rest of the function  
}
```

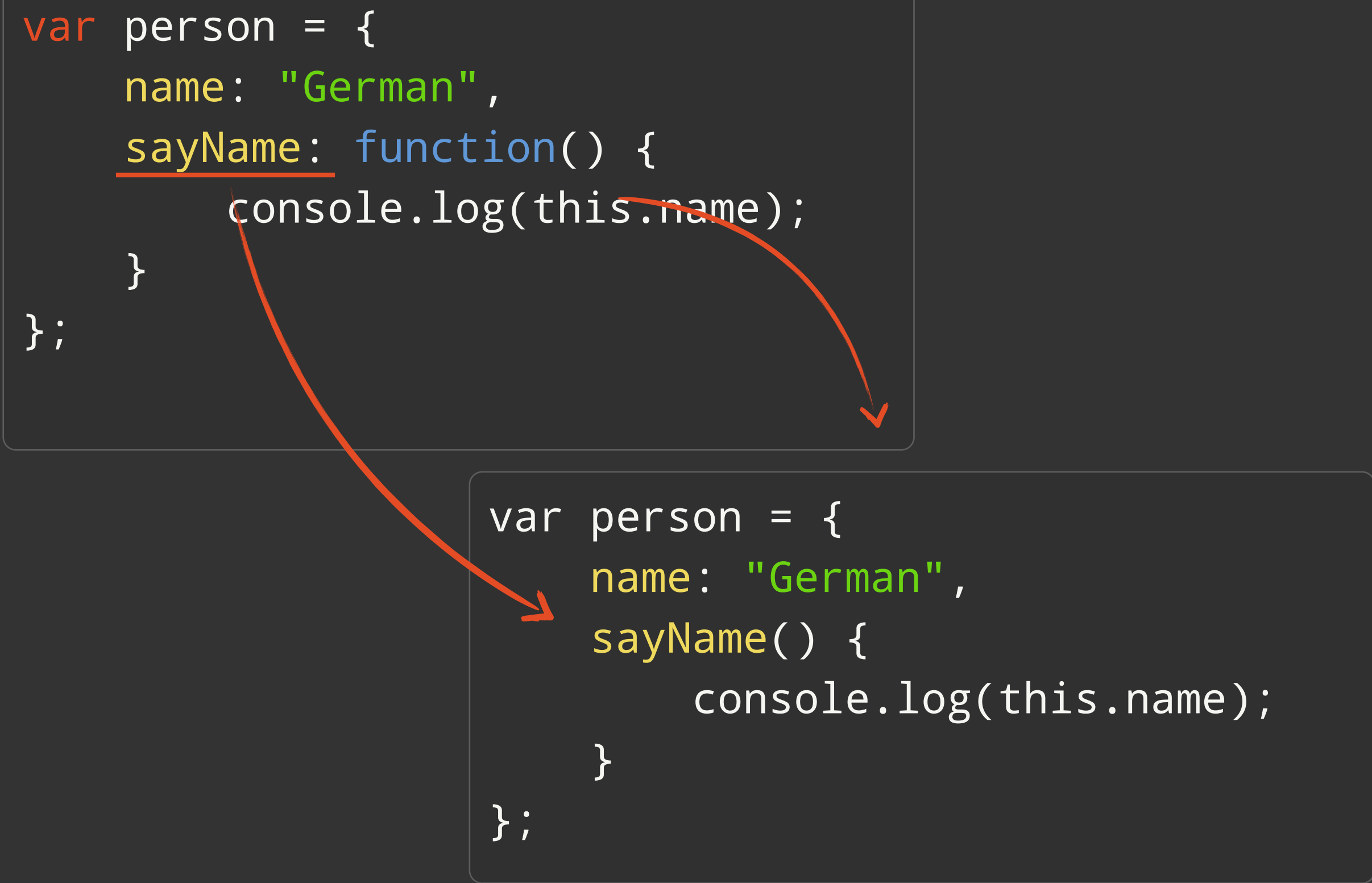
```
makeRequest("/foo");
```

```
makeRequest("/foo", 500);
```

```
makeRequest("/foo", 500, function(response) {  
  // do something  
});
```


Métodos concisos

```
var person = {  
  name: "German",  
  sayName: function() {  
    console.log(this.name);  
  }  
};
```



```
var person = {  
  name: "German",  
  sayName() {  
    console.log(this.name);  
  }  
};
```

Classes

```
class Person {  
  
    constructor(name) {  
        this.name = name;  
    }  
  
    sayName() {  
        console.log(this.name);  
    }  
}
```

```
let person = new Person("German");  
person.sayName(); // imprime "German"
```

Clases

Las clases son sólo azúcar sintáctica sobre las funciones constructoras.

```
let Person = function(name) {  
  this.name = name;  
};  
  
Person.prototype.sayName = function() {  
  console.log(this.name);  
}
```

Desestructuración

```
let [a, b] = [1, 2, 3, 4]  
console.log(a) // 1  
console.log(b) // 2
```

```
let { a, b } = { a: 1, b: 2, c: 3 }  
console.log(a) // 1  
console.log(b) // 2
```

Hacer parejas para los ejercicios

<https://go.makeitreal.camp>

Curso de JavaScript Básico

¡Bienvenid@s! Empezamos en un momento ...