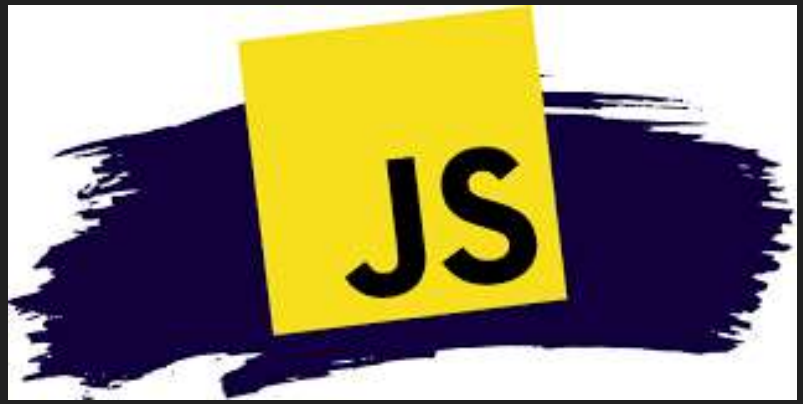

MAKE IT REAL
CAMP

Javascript



JavaScript es el lenguaje de programación que debes usar para añadir características interactivas a tu sitio web, (por ejemplo, juegos, eventos que ocurren cuando los botones son presionados o los datos son introducidos en los formularios, efectos de estilo dinámicos, animación, y mucho más). También puede utilizarse en backend con Node.js en un entorno de servidor.

Variables, Operadores y Condicionales

Cadenas de texto:

```
"Texto entre comillas dobles"
'Texto entre comillas simples'
```

Números:

```
console.log(1 + 2)
console.log(3 * 4 + 5)
console.log(8 / 2)
```

Valores y expresiones booleanas:

```
$ node
> 5 > 3
true
> 5 >= 3
true
> 4 < 4
false
> 4 <= 4
true
> 2 === 2
true
> 2 !== 2
false
> "ruby" === "javascript"
false
> "ruby" !== "javascript"
true
```

```
$ node
> 1 == "1"
true
> 1 === "1"
false
```

Tipos de datos

Javascript tiene un **tipado débil** (no-tipados)

- **Tipos simples del lenguaje:**
  -> `Number (float), String, Boolean, Date, symbol, BigInt`
- **Tipos especiales:**
  -> `NaN, null, undefined`
- **Estructuras de datos:**
  -> `Array, Object`

OPERADORES LÓGICOS Y RELACIONALES	DESCRIPCIÓN	EJEMPLO
==	Es igual	a == b
===	Es estrictamente igual	a === b
!=	Es distinto	a != b
!==	Es estrictamente distinto	a !== b
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual	a <=b
&&	Operador and (y)	a && b
||	Operador or (o)	a || b
!	Operador not (no)	!a

DIFERENCIAS ENTRE == VS === EN JAVASCRIPT

Operador ==

- Realiza una comparación de igualdad suave.
- Realiza una conversión automática de tipo antes de comparar los valores.
- Puede producir resultados incorrectos debido a la conversión automática de tipo.
- Puede comparar valores de diferentes tipos.
- Considera null y undefined como equivalentes.

VS

Operador ===

- Realiza una comparación de igualdad estricta.
- Mientras que el operador === no lo hace.
- Mientras que el operador === Garantiza una comparación precisa y segura.
- Sólo puede comparar valores del mismo tipo.
- Mientras que el operador === los diferencia.

Las variables son uno de los conceptos básicos de la programación y nos permiten almacenar información temporal que podemos usar más adelante en nuestros programas.

```
var name = "Germán"; // cámbialo por tu nombre
console.log("Hola " + name);
```

Las condicionales en programas son grupos de sentencias o sentencias individuales que te permiten condicionar la decisión entre la elección de una opción y otra.

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

```
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");
  }
}
```

Funciones

Son rutinas que se construyen para ser ejecutadas en el programa, deben tener la palabra reservada function, tener o no una lista de parámetros entre paréntesis y separadas con comas y estar entre llaves {}.

```
function <name>([arg1], [arg2], ...) {
  // cuerpo de la función
  return <valor de retorno>;
}
```

Existen dos formas alternas de crear una función, dada la evolución de JS desde ES6.

```
var saludo = function (nombre) {
  return 'Hola ' + nombre;
};
console.log( saludo('Jonathan') );
```

```
let saludo = nombre => `Hola ${nombre}`;
console.log( saludo('Jonathan') ); //Imprime Hola Jonathan
```

Nombres de Funciones

Nombres convencionales para las variables

Utilice alfabetos, números, $ y _

No se permiten otros caracteres especiales

```
function sayHello$() { }  // valid
function say-Hi() { }     // invalid
```

Propiedad de la Función ".name"

Devuelve el nombre de la función

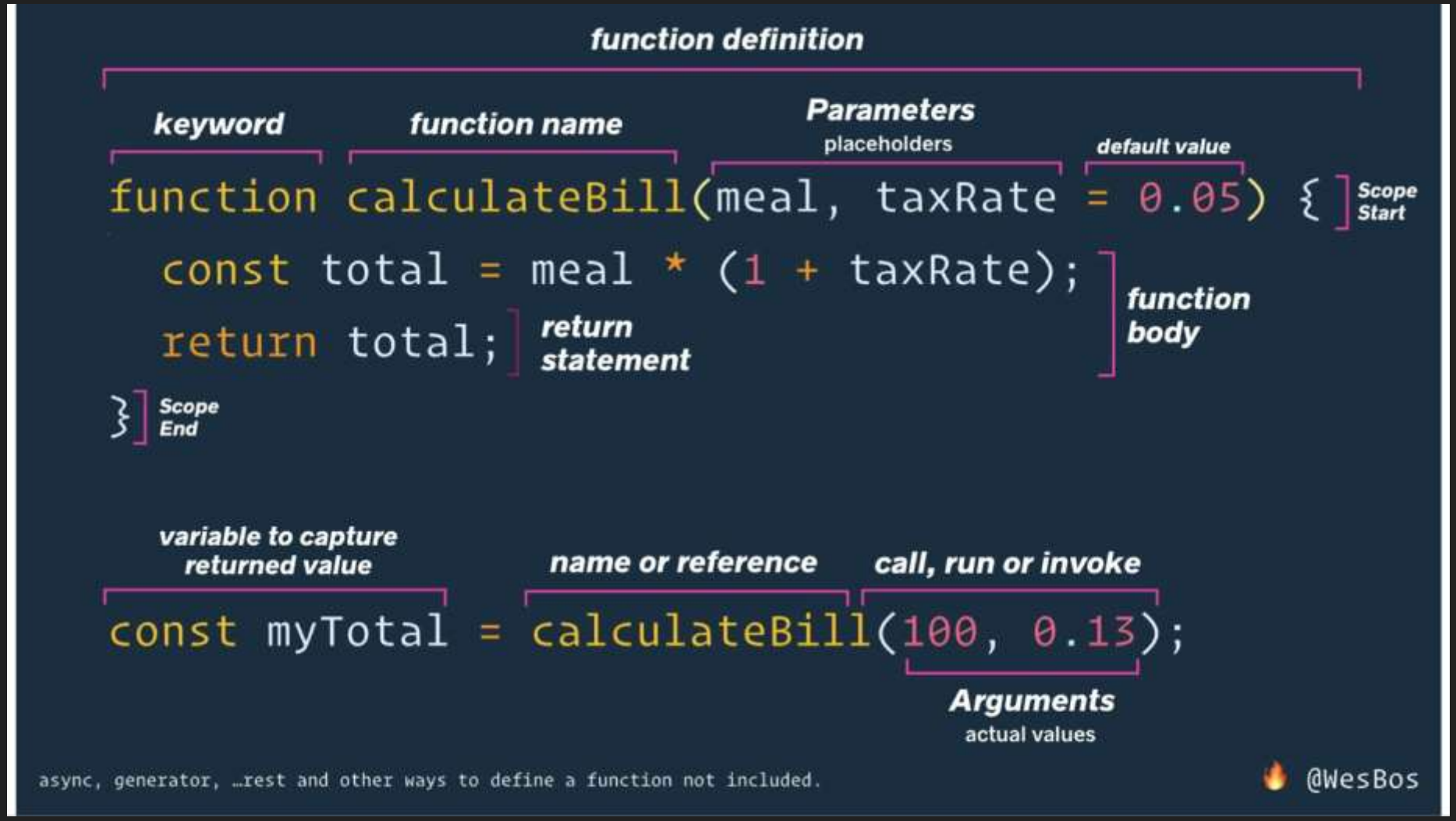
Devuelve "anónimo" para funciones anónimas

```
function sayHello() { }
console.log(sayHello.name);
```

Funciones Anónimas

No hablaremos en este módulo

```
sayHello
```


function definition
keyword
function name
Parameters
placeholders
default value
function calculateBill(meal, taxRate = 0.05) {
Scope Start
const total = meal * (1 + taxRate);
return total;
return statement
function body
}
Scope End
variable to capture returned value
name or reference
call, run or invoke
const myTotal = calculateBill(100, 0.13);
Arguments
actual values
async, generator, …rest and other ways to define a function not included.
@WesBos

Default parameters

MAKE it REAL .camp

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

call y apply

MAKE it REAL .camp

```
var pedro = {
  name: "Pedro"
};

var greet = function() {
  return "Hola " + this.name;
};

greet.call(pedro); // "Hola Pedro"
```

```
func.call(valueForThis, arg1, arg2, …);

func.apply(valueForThis, arrayOfArgs);
```

MAKE IT REAL CAMP

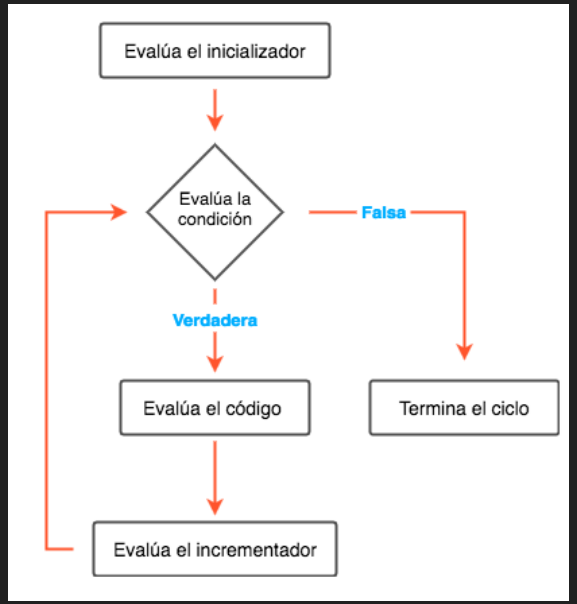
Estamos Trabajando en la misión 6

Ciclos

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

```
var i = 0;
while (i < 850) {
  console.log("Hola mundo");
  i = i + 1;
}
```

```
while (true) {
  console.log("Hola Mundo");
}
```




```
for (var i = 0; i < 850; i ++) {
  console.log("Hola mundo");
}
```

```
for(;;) {
  // el cuerpo del ciclo también es opcional
}
```

Arreglos

Un arreglo es una lista ordenada de elementos de cualquier tipo.

```
var array = [1, "Pedro", true, false, "Juan"]
```

```
> array[1]
"Pedro"
> array[2]
true
> array[3]
false
> array[4]
"Juan"
```

```
var array = [1, "Pedro", true, false, "Juan"];

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

```
var array = ["Pedro"];
array.push("Germán"); // ["Pedro", "Germán"]
array.push("Diana"); // ["Pedro", "Germán", "Diana"]
```

```
var array = [1, 2, 3, 2]
array.indexOf(2) // 1
array.indexOf(8) // -1
```

Arreglos - Vectores - Array

Lista o colección de valores

Los arreglos pueden contener muchos valores diferentes de diferentes tipos de datos

Cada valor tiene un índice

Un *índice* es un valor numérico único que representa los datos dentro de la matriz.

Longitud del arreglo

Después de crear un arreglo, puede verificar su longitud en cualquier momento con **`arrayName.length`**

JavaScript Array Methods

pop()	shift()	find()
push()	unshift()	forEach()
toString()	reverse()	map()
join()	concat()	reduce()
splice()	slice()	every()
sort()	filter()	some()

Métodos de Arrays en JavaScript

@ccadeller

```
[🍉, 🍋, 🍎, 🍊].at(1)                          // 🍋
[🍉, 🍋, 🍎, 🍊].push(🍌)                      // [🍉, 🍋, 🍎, 🍊, 🍌]
[🍉, 🍋, 🍎, 🍊].pop()                         // [🍉, 🍋, 🍎]
[🍉, 🍋, 🍎, 🍊].fill(🍌)                      // [🍌, 🍌, 🍌, 🍌]
[🍉, 🍋, 🍎, 🍊].join(" ")                     // "🍉 🍋 🍎 🍊"
[🍉, 🍋, 🍎, 🍊].shift()                       // [🍋, 🍎, 🍊]
[🍉, 🍋, 🍎, 🍊].reverse()                     // [🍊, 🍎, 🍋, 🍉]
[🍉, 🍋, 🍎, 🍊].unshift(🍌)                   // [🍌, 🍉, 🍋, 🍎, 🍊]
[🍉, 🍋, 🍎, 🍊].includes(🍋)                  // true
[🍉, 🍋, 🍎, 🍊].map(fruit => fruit + 🍌)     // [🍉🍌, 🍋🍌, 🍎🍌, 🍊🍌]
[🍉, 🍋, 🍎, 🍊].filter(fruit => fruit === 🍉) // [🍉]
[🍉, 🍋, 🍎, 🍊].every(fruit => fruit === 🍊)  // false
[🍉, 🍋, 🍎, 🍊].findIndex(fruit => fruit === 🍋) // 1
```

Manipulación de arreglos

Push e Pop – Afecta el final de un Array

- **`array.push(valores)`** - añade uno o más elementos al final de un array y devuelve la nueva longitud del array.
- **`array.pop()`** - elimina el último elemento de un array y lo devuelve.

Shift e Unshift – Afecta el frente de un Array

- **`array.shift()`** - elimina el primer elemento del array y lo retorna.
- **`array.unshift(valores)`** - agrega uno o más elementos al inicio del array, y devuelve la nueva longitud del array.

Concat

Une dos arrays para crear un nuevo array (nueva cadena)

Push y Pop

```
const arr1 = ["A", true, 2];
console.log(arr1.push("new value"));
console.log(arr1);
console.log(arr1.pop()); //Elimina el último elemento
console.log(arr1);
```

```
4
["A", true, 2, "new value"]
"new value"
["A", true, 2]
```

Shift y Unshift

```
const arr1 = ["A", true, 2];
console.log(arr1.unshift("new value"));
console.log(arr1);
console.log(arr1.shift()); //Elimina el ultimo elemento
console.log(arr1);
```

```
4
["new value", "A", true, 2]
"new value"
["A", true, 2]
```

Concat

```
const arr1 = ["A", true, 2];
const arr2 = ["B", false, 3];
const newArr = arr1.concat(arr2)
const newArr2 = arr2.concat([1,2,3])
console.log(newArr);
console.log(newArr2);
```

```
["A", true, 2, "B", false, 3]
["B", false, 3, 1, 2, 3,]
```

MAKE IT REAL CAMP

Estamos Trabajando en la misión 7

Objetos Literales

Los objetos en JavaScript nos ayudan a agrupar información. Un objeto no es más que un conjunto de propiedades en donde cada propiedad está compuesta de una llave y un valor.

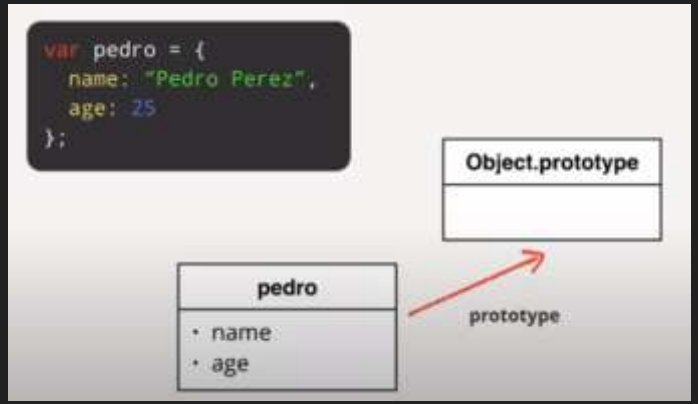
```
var persona = {
  nombre: "Germán",
  apellido: "Escobar",
  edad: 35,
  estatura: 1.8
}
```

```
var person = {
  name: "Pedro",
  sayHi: function() {
    console.log("Hola!");
  }
}
```

```
pedro['name']; // "Pedro Perez"

pedro['name'] = "Juan";
pedro['height'] = 1.8;

delete pedro['height'];
```



¿Qué son los objetos en JavaScript?

Representaciones reales de objetos del mundo - ¡usando código!

Ej: book, song, library, playlist

Los objetos en JavaScript tienen propiedades asociadas

Ex: book has *title, author, isAvailable*

Los objetos JavaScript tienen métodos asociados

Ex: You can *checkIn, checkOut* a book

```
const book = {
        title: "1984",
        author: "George Orwell",
        isAvailable: false,

        checkIn: function(){
            this.isAvailable=true;
        },
        checkOut: function(){
            this.isAvailable=false;
        }
};
console.log(typeof book);
```

```
object
```

Sintaxis y Creación de Objetos (literal)

¡Colección de pares nombre = valor desordenados!

Piense en hashMaps (el nombre es clave, se asigna al valor)

Ex: *title*="1984", *checkin*=function(){ ... }

Este es un ejemplo de un objeto literal

Es la forma más sencilla de **crear** un objeto JavaScript en la definición

Simplemente encierre los pares de nombre y valor entre { }, separados por comas

```
const book = {

        title: "1984",
        author: "George Orwell",
        isAvailable: false,

        checkIn: function(){
           this.isAvailable=true;
        },
        checkOut: function(){
           this.isAvailable=false;
        }

};
```

Propiedades del objeto (atributos)

Las propiedades tienen nombres inmutables y valores mutables

Los nombres utilizan las mismas convenciones que las variables

Método de acceso 1: notación con punto

Use: <object>.<name>

Ej: `book.title` da "1984"

Método de acceso 2: notación entre corchetes

Use: <object>["name"] (like with hashmaps)

Ej: `book["title"]` da "1984"

```
const book = {

        title: "1984",
        author: "George Orwell",
        isAvailable: false,

        checkIn: function(){
            this.isAvailable=true;
        },
        checkOut: function(){
            this.isAvailable=false;
        }

};
```

Métodos de objeto (acciones)

Los métodos son propiedades especiales cuyos valores son definiciones de funciones.

Se utilizan para definir tareas que operan sobre datos de objetos.

Métodos de acceso como propiedades

Ej: `book.checkIn` da [Function: checkIn]

Ej: `book["checkIn"]` da [Function: checkIn]

Invocar métodos como funciones

Ej: `book.checkIn()` ejecuta ese método

Ej: `book["checkIn"]()` hace lo mismo

```
const book = {

        title: "1984",
        author: "George Orwell",
        isAvailable: false,

        checkIn: function(){
            this.isAvailable=true;
        },
        checkOut: function(){
            this.isAvailable=false;
        }

};
```

La palabra clave "this" (contexto)

Las funciones necesitan acceso al contexto de tiempo de ejecución.

Los métodos (de objeto) pueden necesitar acceso a valores de propiedad de pares

Las funciones (independientes) pueden usar variables globales en la ejecución

"this" se asigna al contexto relevante en tiempo de ejecución

Se asigna al objeto adjunto - para métodos de objeto

Se asigna a un objeto global: para funciones independientes

Puede estar indefinido (modo estricto) o cambiado (por ejemplo, se aplica el uso)

Sobre el objeto global

por defecto es el objeto de la ventana en el tiempo de ejecución del navegador

predeterminado al objeto global en el tiempo de ejecución de Node.js

utilice "globalThis" para hacer referencia al objeto global de forma coherente

```
const book = {
        isAvailable: false,
        checkIn: function(){
           this.isAvailable=true;
           return this;
        }
};
console.log( book.checkIn() );
// prints book object
```

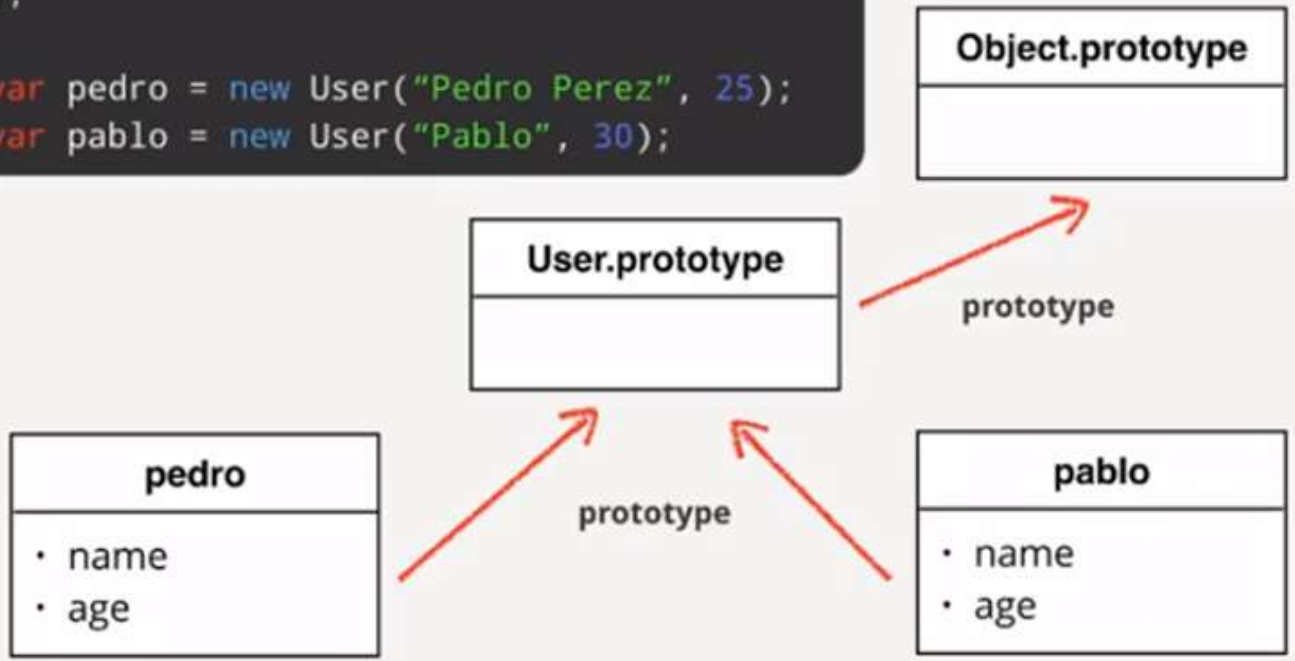
```
function checkIn() {
  return this;
}
console.log( checkIn() );
// prints global object
```

Prototipos

MAKE it REAL .camp

```
function User(name, age) {
  this.name = name;
  this.age = age;
};

var pedro = new User("Pedro Perez", 25);
var pablo = new User("Pablo", 30);
```



Property	Description
constructor	Returns a function that created instance.
__proto__	This is invisible property of an object. It returns prototype object of a function to which it links to.

Method	Description
hasOwnProperty()	Returns a boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain.
isPrototypeOf()	Returns a boolean indication whether the specified object is in the prototype chain of the object this method is called upon.
propertyIsEnumerable()	Returns a boolean that indicates whether the specified property is enumerable or not.
toLocaleString()	Returns string in local format.
toString()	Returns string.
valueOf	Returns the primitive value of the specified object.

MAKE IT REAL CAMP

Estamos Trabajando en la misión 8

Dadas las sesiones anteriores, es momento de aplicar las habilidades adquiridas en JS hacia el navegador, hay que darle vida a los sitios web para que el usuario se sienta cómodo al utilizarlas.

¡Es momento de pasar al siguiente nivel!