ThePower business school

🇪🇸 Objetos JS🎁🤖

En JavaScript, un objeto es una colección de pares clave-valor. Las claves son cadenas de texto o símbolos, mientras que los valores pueden ser cualquier tipo de dato, incluyendo otro objeto. Los objetos se usan para almacenar y acceder a datos de manera estructurada.

Aquí hay algunos ejemplos de objetos en JavaScript:

```
// Un objeto vacío
let obj = {};

// Un objeto con algunas propiedades
let obj = {
  name: 'John',
  age: 30,
  job: 'developer'
};

// Acceder a las propiedades de un objeto
console.log(obj.name); // 'John'
console.log(obj['age']); // 30

// Modificar las propiedades de un objeto
obj.name = 'Jane';
obj['age'] = 35;

// Añadir nuevas propiedades a un objeto
obj.location = 'New York';

// Eliminar propiedades de un objeto
delete obj.job;
```

También puedes usar la notación de corchetes para acceder a las propiedades de un objeto, lo que te permite usar variables para seleccionar las propiedades. Por ejemplo:

```
let prop = 'name';
console.log(obj[prop]); // 'Jane'
```

Además de las propiedades, los objetos también pueden tener métodos, que son funciones asociadas al objeto. Por ejemplo:

```
let obj = {
  name: 'John',
  age: 30,
  greet: function() {
    console.log(`Hello, my name is ${this.name}`);
  }
};

obj.greet(); // 'Hello, my name is John'
```

Objetos y propiedades

Un objeto de JavaScript tiene propiedades asociadas a él. Una propiedad de un objeto se puede explicar como una variable adjunta al objeto. Las propiedades de un objeto básicamente son lo mismo que las variables comunes de JavaScript, excepto por el nexo con el objeto.

```
var myAvenger = new Object();
myAvenger.name = 'Captain America';
myAvenger.power = 80;
myAvenger.creator = 'Stan Lee';
```

El ejemplo anterior también se podría escribir usando un iniciador de objeto, que es una lista delimitada por comas de cero o más pares de nombres de propiedad y valores asociados de un objeto, encerrados entre llaves ({}).

```
let myAvenger = {
    name: 'Captain America',
    power: 80,
    creator: 'Stan Lee'
};
```

Las propiedades no asignadas de un objeto son **undefined** .

```
myAvenger.alias // undefined
```

También puedes acceder o establecer las propiedades de los objetos en JavaScript mediante la notación de corchetes **[ ].**

```
myAvenger['name']  = 'Captain America';
```

Podemos acceder a estos también usando el valor de una variable, la definimos con el nombre y luego la usamos, ahora no os parecerá útil pero en un futuro volveremos a este punto.

```
let name = 'name';
myAvenger[name] = 'Captain America';

let power = 'power';
myAvenger[power] = 80;
```

Enumerar propiedades objecto

A partir de ECMAScript 5, hay dos formas nativas para enumerar/recorrer las propiedades de objetos.

- **bucles for...in:** Este método recorre todas las propiedades enumerables de un objeto y su cadena de prototipos.

```
let batman= {
    nombre: "Bruce",
    apellidos: "Wayne",
```

```
    localizacion: "Gotham",
    profesion: "Multimillonario"
}

for (let clave in batman) {
    console.log("Batman tiene la clave " + clave+ " con valor: " + batman[clave]);
}

//Retorna
//Batman tiene la clave nombre con valor: Bruce
//Batman tiene la clave apellidos con valor: Wayne
//Batman tiene la clave localizacion con valor: Gotham
//Batman tiene la clave profesion con valor: Multimillonario
```

- **Object.keys(o):** Este método devuelve un arreglo con todos los nombres de propiedades enumerables ("**claves**") propias (no en la cadena de prototipos) de un objeto **o**.

```
let batman= {
    nombre: "Bruce",
    apellidos: "Wayne",
    localizacion: "Gotham",
    profesion: "Multimillonario"
}

let keys = Object.keys(batman);
console.log(keys);
//Retorna ["nombre", "apellidos", "localizacion", "profesion"]
```

Funciones constructoras

Como alternativa, puedes crear un objeto con estos dos pasos:

- Definir el tipo de objeto escribiendo una función constructora. Existe una fuerte convención, con buena razón, para utilizar en mayúscula la letra inicial.
- Crear una instancia del objeto con el operador **new**.

Para definir un tipo de objeto, crea una función para el objeto que especifique su nombre, propiedades y métodos. Por ejemplo, supongamos que deseas crear un tipo de objeto para coches. Quieres llamar **Car** a este tipo de objeto, y deseas que tenga las

siguientes propiedades: **make**, **model** y **year**. Para ello, podrías escribir la siguiente función.

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}
```

Observa el uso de **this** para asignar valores a las propiedades del objeto en función de los valores pasados a la función. Ahora puedes crear un objeto llamado **myCar** de la siguiente manera.

```
let mycar = new Car('Eagle', 'Talon TSi', 1993);
```

Definición de métodos

Un *método* es una función asociada a un objeto, o, simplemente, un método es una propiedad de un objeto que es una función. Los métodos se definen normalmente como una función, con excepción de que tienen que ser asignados como la propiedad de un objeto.

```
objectName.methodname = functionName;

var myObj = {
  myMethod: function(params) {
    // ...hacer algo
  }

  // O ESTO TAMBIÉN FUNCIONA

  myOtherMethod(params) {
    // ...hacer algo más
  }
};
```

Uso de this para referenciar objetos

JavaScript tiene una palabra clave especial, **this**, que puedes usar dentro de un método para referirte al objeto actual. Por ejemplo, supongamos que tienes 2 objetos,**Manager** e **Intern**. Cada objeto tiene su propio **name**, **age** y **job**. En la función **sayHi()**, observa que hay **this.name**. Cuando se agregan a los 2 objetos, se pueden llamar y devuelve el **'Hola, mi nombre es'** y luego agrega el valor **name** de ese objeto específico. Como se muestra abajo.

```
const Manager = {
  name: "John",
  age: 27,
  job: "Software Engineer"
}

const Intern= {
  name: "Ben",
  age: 21,
  job: "Software Engineer Intern"
}

function sayHi() {
    console.log('Hola, mi nombre es ', this.name)
}

// agrega la función sayHi a ambos objetos
Manager.sayHi = sayHi;
Intern.sayHi = sayHi;

Manager.sayHi() // Hola, mi nombre es John'
Intern.sayHi() // Hola, mi nombre es Ben'
```

this se refiere al objeto en el que se encuentra. Puedes crear una nueva función llamada **howOldAmI()** que registra una oración que dice cuántos años tiene la persona.

```
function howOldAmI() {
  console.log('Tengo ' + this.age + ' años.')
}
Manager.howOldAmI = howOldAmI;
Manager.howOldAmI() // Tengo 27 años.
```

Definición captadores → getters & setters

Un captador (getter) es un método que obtiene el valor de una propiedad específica. Un establecedor (setter) es un método que establece el valor de una propiedad específica. Puedes definir captadores y establecedores en cualquier objeto principal predefinido o en un objeto definido por el usuario que admita la adición de nuevas propiedades.

```
var o = {
  a: 7,
  get b() {
    return this.a + 1;
  },
  set c(x) {
    this.a = x / 2;
  }
};

console.log (o.a); // 7
console.log (o.b); // 8 <-- En este punto se inicia el método get b().
o.c = 50;         // <-- En este punto se inicia el método set c(x)
console.log(o.a); // 25
```

Ten en cuenta que los nombres de función de los captadores y establecedores definidos en un objeto literal usando "[gs]et *propiedad*()" (en contraposición a __**define [GS]etter**__) no son los nombres de los captadores en sí, aunque la sintaxis **[gs]et *propertyName*() {}** te puede inducir a pensar lo contrario.

Los captadores y establecedores también se pueden agregar a un objeto en cualquier momento después de la creación usando el método **Object.defineProperties**. El primer parámetro de este método es el objeto sobre el que se quiere definir el captador o establecedor. El segundo parámetro es un objeto cuyo nombre de propiedad son los nombres **getter** o **setter**, y cuyos valores de propiedad son objetos para la definición de las funciones **getter** o **setter**. Aquí hay un ejemplo que define el mismo **getter** y **setter** utilizado en el ejemplo anterior

```
var o = { a: 0 };
```

```
Object.defineProperties(o, {
    'b': { get: function() { return this.a + 1; } },
    'c': { set: function(x) { this.a = x / 2; } }
});

o.c = 10; // Ejecuta el establecedor, que asigna 10/2 (5) a la propiedad 'a'
console.log(o.b); // Ejecuta el captador, que produce un + 1 o 6
```

Eliminar propiedades de un objecto

Puedes eliminar una propiedad no heredada (más información en POO) mediante el operador **delete**. El siguiente código muestra cómo eliminar una propiedad.

```
//Crea un nuevo objeto, myobj, con dos propiedades, a y b.
var myobj = new Object;
myobj.a = 5;
myobj.b = 12;

// Elimina la propiedad a, dejando a myobj solo con la propiedad b.
delete myobj.a;
console.log ('a' in myobj); // muestra: "false"
```

Comparar objetos

Como sabemos los objetos son de tipo referencia en JavaScript. Dos distintos objetos nunca son iguales, incluso aunque tengan las mismas propiedades. Solo comparar la misma referencia de objeto consigo misma arroja verdadero.

```
// Dos variables, dos distintos objetos con las mismas propiedades
var fruit = { name: 'apple' };
var fruitbear = { name: 'apple' };

fruit == fruitbear; // devuelve false
fruit === fruitbear; // devuelve false

// Dos variables, un solo objeto
var fruit = { name: 'apple' };
var fruitbear = fruit; // Asigna la referencia del objeto fruit a fruitbear

// Aquí fruit y fruitbear apuntan al mismo objeto
fruit == fruitbear; // devuelve true
```

```
fruit === fruitbear; // devuelve true

fruit.name = 'grape';
console.log(fruitbear); // Produce: { name: "grape" }, en lugar de { name: "apple" }
```

Uso de **entries** para devolver un array

El método **Object.entries()** nos permite generar un array cuyos elementos son arrays correspondientes a las claves y valores que tengan cada una de las propiedades de un objeto. El orden de estos arrays será el mismo que el que tendríamos recorriendo el objeto manualmente.

```
const character = {
    firstName: 'James',
    lastName: 'Bond',
    age: 50,
    code: 007
};

const agent = Object.entries(character);

console.log(agent);

/*Retorna ->
[
  ["firstName", "James"],
  ["lastName", "Bond"],
  ["age", 50],
  ["code", 007]
]

*/
```