OBJETOS JSON

Un objeto es un tipo de datos de JavaScript, al igual que un número o una cadena. Como tipo de datos,un objeto puede estar contenido en una variable
Hay cuatro formas de construir un objeto en JavaScript:

- El objeto literal, que usa llaves: {}
- El constructor de objetos, que usa la palabra clave new
- El método create de Object
- Función constructora con new

1. El objeto literal, que usa llaves: {}

Un objeto está limitado por llaves {}. Dentro del objeto tendremos una lista de pares: una llave y un valor separados por dos puntos :; cada par está separado por una coma.

```
var jugador = {
  nombre : "Pedro",
  salud : 20,
  hablar: function() { return "Hola"; }
};
```

2. El constructor de objetos

También podemos usar el constructor Object para crear objetos:


```
var jugador = new Object();
jugador.nombre = "Pedro";
jugador.salud = 20;
jugador.hablar =  function() { return "Hola"; }
```

Si se omite la palabra new se crea también un objeto vacío. El acceso es igual que en los objetos literales.

3. El método create de Object

Método create
Object cuenta un método, create, que sirve para crear objetos a partir de un determinado prototype:

Método create

```
// Por ahora utilizaremos el prototipo estándar
var jugador = Object.create(Object.prototype);
jugador.nombre = "Pedro";
jugador.salud = 20;
jugador.hablar = function() { return "Hola"; }
```

4. Función constructora

Se utiliza una función constructora para crear clases en JavaScript

Función constructora

```
// El nombre de la función constructora suele empezar por mayúscula
function Jugador(nombre, salud) {
  this.nombre = nombre;
  this.salud = salud;
  this.hablar = function() { return "Hola"; }
}

var jugador = new Jugador("Pedro", 20);
```

Propiedades y métodos

Una **propiedad** es la asociación entre un nombre (clave) y un valor dentro de un objeto, y puede contener cualquier tipo de datos. Una propiedad generalmente se refiere a la característica de un objeto.

Un **método** es una función que es el valor de una propiedad de objeto y, por lo tanto, una tarea que un objeto puede realizar.

`this`

Dentro del cuerpo de función o método está disponible un valor especial de solo lectura llamado `this`.

Normalmente, esta palabra clave se relaciona con funciones que son propiedades de objetos. Cuando se invocan los métodos, esta palabra clave toma el valor del objeto específico en el que se invocó:

```
const objeto =  {
  nombre: 'Juan',
  hablar: function() {
    return 'Mi nombre es ' + this.nombre;
  }
}

objecto.hablar(); // Mi nombre es 'Juan'
```

Es importante comprender que this está vinculado de acuerdo a *cómo se llama a la función no a dónde se declara la función*. Es decir, this está ligado a objeto no porque hablar es una propiedad de objeto, sino porque lo llamamos directamente en objeto (objeto.hablar()).

Propiedades de datos

Acceso
Podemos acceder a un objeto (a sus propiedades y métodos) de 2 formas: .
Con la notación punto (obj.propiedad) . Con la notación corchetes (obj["propiedad"])

Acceso a las propiedades

```
jugador.nombre;    // "Pedro"
jugador.hablar();  // "Hola"
```

Agregando y modificando las propiedades de un objeto

Para agregar una nueva propiedad a un objeto o modificar una existente, debe asignarse un nuevo valor a una propiedad con el operador de asignación =.

Por ejemplo, podemos agregar un tipo de datos numéricos al objeto jugador como la propiedad edad.

```
jugador.edad = 29;    // Se agrega una nueva propiedad
jugador.salud = 30;   // Se modifica una propiedad existente
```

Agregando y modificando las propiedades de un objeto

Lo mismo se puede hacer con los métodos:

```
jugador.luchar = function() { return "¡Lucha!"; }
```

Eliminando propiedades de un objeto
Para eliminar una propiedad de un objeto se utiliza la palabra clave delete. delete es un operador que elimina una propiedad de un objeto.

```
// Elimina la propiedad edad
delete jugador.edad;
```

Recorriendo las propiedades de un objeto

JavaScript tiene un tipo incorporado de bucle for que está específicamente diseñado para iterar sobre las propiedades de un objeto. Esto se conoce como bucle for ... in.

```
for (let llave in jugador) {
  console.log(llave + ": " + jugador[llave]);
}
```

El bucle for ... in no debe confundirse con el bucle for ... of, que se usa exclusivamente en el tipo de objeto Array.

Propiedades de acceso (getters y setters)

Desde ES6 (ES2015), tenemos la posibilidad de usar getters y setters para definir propiedades en nuestros objetos.
Una función que obtiene un valor de una propiedad se llama getter y una que establece el valor de una propiedad se llama setter.

Las propiedades de acceso se construyen con métodos de obtención “getter” y asignación “setter”.
En un objeto literal se denotan con get y set:

```
let obj = {
  get propName() {
    // getter, el código ejecutado para obtener obj.propName
  },

  set propName(value) {
    // setter, el código ejecutado para asignar obj.propName = value
  }
};
```

Propiedades de acceso (getters y setters)

```
const obj = {
  get contador() {return this._contador;},
  set contador(valor) {this._contador = valor;}
 };

obj.contador = 10;
obj.contador;   // 10
```

```
1  <!DOCTYPE html>
2  <script>
3  "use strict";
4
5  let user = {
6    name: "John",
7
8
9    get fullName() {
10     return this.name;
11   },
12
13   set fullName(value) {
14     this.name = value
15   }
16 };
17
18 // set fullName se ejecuta con el valor dado.
19 user.fullName = "Alice ";
20
21 alert(user.name); // Alice
22
23 </script>
```

EJEMPLO

```
const person = {
  firstName: "John",
  lastName: "Doe",
  language: "",
  set lang(lang) {
    this.language = lang;
  }
};

// Set an object property using a setter:
person.lang = "en";

// Display data from the object:
document.getElementById("demo").innerHTML = person.language;
```

EJEMPLO:HTML

¿Por qué usar Getters y Setters?

- Da una sintaxis más simple.
- Permite la misma sintaxis para propiedades y métodos.
- Puede asegurar una mejor calidad de datos

Ejemplos:Constructores de objetos

- function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
  }

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>

<script>
// Constructor function for Person objects
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}

// Create a Person object
const myFather = new Person("John", "Doe", 50, "blue");

// Display age
document.getElementById("demo").innerHTML = "My father is " + myFather.age + ".";
</script>

</body>
</html>
```

Agregar propiedad/método

```
myFather.nationality = "English";


  myFather.name = function () {
    return this.firstName + " " + this.lastName;
  };
```

ARRAY DE OBJETOS

```
var unArray = [{pepe:1},{juan:2},{antonio:3}]


alert (unArray[1].juan);
unArray.push({luis:4});

alert(unArray[3].luis);
```

```
for (x in unArray){console.log(unArray[x])}
```

Visualización

Visualización de las propiedades del objeto

```
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

document.getElementById("demo").innerHTML = person.name + "," + person.age + "," + person.city;
```

Visualización

Visualización de las propiedades del objeto

```
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

let txt = "";
for (let x in person) {
txt += person[x] + " ";
};

document.getElementById("demo").innerHTML = txt;
```

Visualización

Con Object podemos convertir un objeto en array

```
const person = {
  name: "John",
  age: 30,
  city: "New York"
};

document.getElementById("demo").innerHTML = Object.values(person);
```

AGREGAR GETTERS/SETTERS

En JavaScript, también puede usar el método para agregar getters y setters. Por ejemplo Object.defineProperty()

La sintaxis para usar es:Object.defineProperty()

El método toma tres argumentos.Object.defineProperty()

- El primer argumento es el objectName.
- El segundo argumento es el nombre de la propiedad.
- El tercer argumento es un objeto que describe la propiedad.

```
Object.defineProperty(obj, prop, descriptor)
```

```
const student = {
    firstName: 'Monica'
}

// getting property
Object.defineProperty(student, "getName", {
    get : function () {
        return this.firstName;
    }
});

// setting property
Object.defineProperty(student, "changeName",
    {
    set : function (value) {
        this.firstName = value;
    }
});

console.log(student.firstName); // Monica

// changing the property value
student.changeName = 'Sarah';

console.log(student.firstName); // Sarah
```

En JavaScript, se puede usar un prototipo para agregar propiedades y métodos a una función constructora. Y los objetos heredan propiedades y métodos de un prototipo. Por ejemplo,

```
// constructor function
function Person () {
    this.name = 'John',
    this.age = 23
}

// creating objects
const person1 = new Person();
const person2 = new Person();

// adding property to constructor function
Person.prototype.gender = 'male';

// prototype value of Person
console.log(Person.prototype);

// inheriting the property from prototype
console.log(person1.gender);
console.log(person2.gender);
```

```
Person { gender: 'male' }
male
male
```

Prototype

Insertar funciones nuevas

```
1  // constructor function
2  function Person () {
3      this.name = 'John',
4      this.age = 23
5  }
6
7  // creating objects
8  const person1 = new Person();
9  const person2 = new Person();
10
11 // adding a method to the constructor function
12 Person.prototype.greet = function() {
13     console.log('hello' + ' ' +  this.name);
14 }
15
16 person1.greet(); // hello John
17 person2.greet(); // hello John
```

Prototype

Si se cambia un valor de prototipo, todos los objetos nuevos tendrán el valor de propiedad modificado. Todos los objetos creados anteriormente tendrán el valor anterior. Por ejemplo,

```
1  // constructor function
2  function Person() {
3      this.name = 'John'
4  }
5
6  // add a property
7  Person.prototype.age = 20;
8
9  // creating an object
10 const person1 = new Person();
11
12 console.log(person1.age); // 20
13
14 // changing the property value of prototype
15 Person.prototype = { age: 50 }
16
17 // creating new object
18 const person3 = new Person();
19
20 console.log(person3.age); // 50
21 console.log(person1.age); // 20
```

JSON

```
    <script>

const persona = {
  nombre: 'Pepe',
  apellido: 'Perez',
  edad: 45,
  direccion: {
    ciudad: 'Madrid',
    pais: 'España'
  }
}

// Pasar un objeto de JavaScript a JSON
const personaJSON = JSON.stringify(persona)
console.log(personaJSON)

// Pasar un JSON a un objeto de JavaScript
const personaObjeto = JSON.parse(personaJSON)
console.log(personaObjeto)
console.log(persona.direccion.ciudad);
console.log(persona.apellido)



    </script>
```

JSON

```
const persona = {
  nombre: 'Pepe',
  apellido: 'Perez',
  edad: 45,
  direccion: {
    ciudad: 'Madrid',
    pais: 'España'
  }
}
alert(persona);
alert(persona.direccion.ciudad);

const persona1 = {
  nombre: 'Pepe',
  apellido: 'Perez',
  edad: 45,
  direccion: {
    ciudad: 'Madrid',
    pais: 'España'
  }
}

// Object.keys - devuelve las claves de un objeto
console.log(Object.keys(persona1))
// Object.values - devuelve los valores de un objeto
console.log(Object.values(persona1))
// Object.entries - devuelve las claves y valores de un objeto
console.log(Object.entries(persona1))
// Object.assign - asigna propiedades de un objeto a otro
const persona2 = Object.assign({}, persona1)
persona2.nombre = 'Ana'
console.log(persona2)
console.log(persona)
```

Object.keys():El método devuelve una matriz de nombres de propiedad propios de un objeto dado , Se itera en el mismo orden que lo haría un bucle normal.

Object.values():El método devuelve una matriz de los valores de un objeto dado

Object.entries(): El método devuelve una matriz de pares de propiedades con clave y valor de un objeto dado . [key, value]

El método Object.assign()copia todas las propiedades de uno o más objetos fuente a un objeto destino. Devuelve el objeto destino. En este caso se produce clonado de objeto (ver diapositiva siguiente)

JSON

```
const target = { a: 1, b: 2 };
const source = { b: 4, c: 5 };

const returnedTarget = Object.assign(target, source);

console.log(target);
// expected output: Object { a: 1, b: 4, c: 5 }
```

Clonando un objeto

```
var obj = { a: 1 };
var copy = Object.assign({}, obj);
console.log(copy); // { a: 1 }
```

Clases

ECMAScript 2015, también conocido como ES6, introdujo las clases de JavaScript.

```
class ClassName {
  constructor() { ... }
}
```

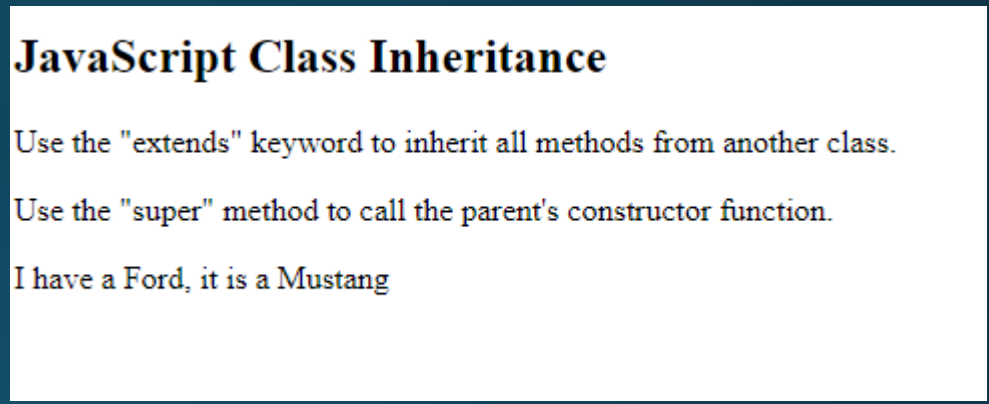
```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
```

Clases

```
class Alumno {
    constructor(nombre, apellidos, edad) {
        this.nombre = nombre
        this.apellidos = apellidos
        this.edad = edad
    }
    getInfo() {
        return 'El alumno ' + this.nombre + ' ' + this.apellidos + ' tiene ' + this.edad + ' años'
    }
}

let cpo = new Alumno('Carlos', 'Pérez Ortiz', 19)
console.log(cpo.getInfo())     // imprime 'El alumno Carlos Pérez Ortíz tiene 19 años'
```

HERENCIA

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Class Inheritance</h2>

<p>Use the "extends" keyword to inherit all methods from another class.</p>
<p>Use the "super" method to call the parent's constructor function.</p>

<p id="demo"></p>

<script>
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
</script>

</body>
```



Con getters and setters con clases

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Class Gettter/Setter</h2>

<p>A demonstration of how to add getters and setters in a class, and how to use the getter to get the property value.</p>

<p id="demo"></p>

<script>
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  get cnam() {
    return this.carname;
  }
  set cnam(x) {
    this.carname = x;
  }
}

let myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.cnam;
</script>

</body>
</html>
```

JavaScript Class Gettter/Setter

A demonstration of how to add getters and setters in a class, and how to use the getter to get the property value.

Ford