DWC

(Desarrollo Web en entorno cliente)

Índice

1.- Objetos 1
2.- Creación de objetos 1
3.- Literales y propiedades 1
4.- La sintaxis de corchetes 3
5.- Propiedades calculadas 5
6.- Limitaciones de nombres de propiedad 6
7.- Comprobar la existencia de propiedades 7
8.- El bucle "for... in" 8
9.- Ordenación de objetos 9
10.- Métodos de objeto 11
11.- This 12
    11.1.- This en JavaScript 13
12.- JSON (JavaScript Object Notation) 14
    12.1.- La sintaxis de JSON 14
    12.2.- Métodos JSON 19
    12.2.1.- JSON.stringify 19
    12.2.1.- JSON.parse 21
13.- Resumen 22

1.- Objetos

Como sabemos, hay ocho tipos de datos en JavaScript. Siete de ellos se denominan "primitivos", porque sus valores contienen una sola cosa (ya sea una cadena o un número o lo que sea).

Los objetos en si son arrays asociativos, es decir son arrays en los que la información no se guarda en posiciones accesibles por índices númericos, si no que se almacena en posiciones accesibles por nombres.

2.- Creación de objetos

Por el contrario, los objetos se utilizan para almacenar colecciones con clave de varios datos y entidades más complejas. En JavaScript, los objetos tocan casi todos los aspectos del lenguaje. Por lo tanto, debemos comprenderlos primero antes de profundizar en cualquier otro lugar.

Se puede crear un objeto con corchetes {…} con una lista opcional de propiedades. Una propiedad es un par "clave: valor", donde `clave` es una cadena (también llamada "nombre de propiedad"), y `value` puede ser cualquier cosa.

Se puede crear un objeto vacío utilizando una de estas dos sintaxis:

```
let user = new Object(); // "object constructor" syntax
let user = {};  // "object literal" syntax
```

Por lo general, {...} se utilizan los corchetes. Esa declaración se llama un objeto literal.

3.- Literales y propiedades

Podemos poner inmediatamente algunas propiedades en pares de {...} "clave: valor":

```
let user = {     // an object
  name: "John",  // by key "name" store value "John"
  age: 30        // by key "age" store value 30
};
```

Una propiedad tiene una clave (también conocida como "nombre" o "identificador") antes de los dos puntos ":" y un valor a la derecha.

En el objeto user, hay dos propiedades:

- La primera propiedad tiene el nombre "name"y el valor "John".
- La segunda tiene el nombre "age"y el valor 30.

Se puede acceder a los valores de las propiedades utilizando la notación de puntos:

```
// get property values of the object:
alert( user.name ); // John
alert( user.age ); // 30
```

El valor puede ser de cualquier tipo. Agreguemos uno booleano:

```
user.isAdmin = true;
```

Para eliminar una propiedad, podemos usar el operador **delete**:

```
delete user.age;
```

También podemos usar nombres de propiedades de varias palabras, pero luego se deben entrecomillar:

```
let user = {
  name: "John",
  age: 30,
  "likes birds": true  // multiword property name must be quoted
};
```

La última propiedad en la lista puede terminar con una coma:

```
let user = {
  name: "John",
  age: 30,
}
```

Facilita agregar / eliminar / mover propiedades, porque todas las líneas se vuelven iguales.

Un Objeto definido como constante puede ser cambiado

Tenga en cuenta: un objeto declarado como const se puede modificar.

Por ejemplo:

```
const user = {
  name: "John"
};
user.name = "Pete"; // (*)

alert(user.name); // Pete
```

Puede parecer que la línea (*) causaría un error, pero no. **El const corrige el valor de user, pero no su contenido** (cambia valores, pero estructura)

4.- La sintaxis de corchetes

Podemos acceder a las propiedades de los objetos como si fuera un array (de hecho un objeto es un array en JavaScript).

Esto soluciona algunos problemas como los siguientes.

```
// this would give a syntax error
user.likes birds = true
```

JavaScript no entiende eso. El punto requiere que la clave sea un identificador de variable válido. Eso implica: no contiene espacios, no comienza con un dígito y no incluye caracteres especiales ($ y _están permitidos).

Hay una "notación de corchetes" alternativa que funciona con cualquier cadena:

```
let user = {};

user["likes birds"] = true; // set

// get
alert(user["likes birds"]); // true

// delete
delete user["likes birds"];
```

Ahora todo está bien. Tenga en cuenta que la cadena dentro de los corchetes se cita correctamente (cualquier tipo de comillas servirá).

Los corchetes también proporcionan una forma de obtener el nombre de la propiedad como resultado de cualquier expresión, a diferencia de una cadena literal, como una variable de la siguiente manera:

```
let key = "likes birds";

// same as user["likes birds"] = true;
user[key] = true;
```

Aquí, la variable key puede calcularse en tiempo de ejecución o depender de la interacción con el usuario. Y luego lo usamos para acceder a la propiedad. Eso nos da mucha flexibilidad.

Por ejemplo:

```
let user = {
  name: "John",
  age: 30
};

let key = prompt("What do you want to know about the user?", "name");

// access by variable
alert( user[key] ); // John (if enter "name")
```

La notación de puntos no se puede usar de manera similar:

```
let user = {
  name: "John",
  age: 30
};

let key = "name";
alert( user.key ) // undefined
```

5.- Propiedades calculadas

Podemos usar corchetes en un objeto literal, al crear un objeto. Eso se llama propiedades calculadas.

Por ejemplo:

El significado de una propiedad calculada es simple: [fruit] significa que se debe tomar el nombre de la propiedad fruit.

```
let fruit = prompt("Which fruit to buy?", "apple");

let bag = {
  [fruit]: 5, // the name of the property is taken from the variable fruit
};

alert( bag.apple ); // 5 if fruit="apple"
```

Entonces, si un usuario escribe "apple", bag se convertirá {apple: 5}.

Esencialmente, eso funciona igual que:

```
let fruit = prompt("Which fruit to buy?", "apple");
let bag = {};

// take property name from the fruit variable
bag[fruit] = 5;
```

Podemos usar expresiones más complejas dentro de corchetes:

```
let fruit = 'apple';
let bag = {
  [fruit + 'Computers']: 5 // bag.appleComputers = 5
}
```

Los corchetes son mucho más potentes que la notación de puntos. Permiten cualquier nombre de propiedad y variables. Pero también son más engorrosos de escribir.

Por lo tanto, la mayoría de las veces, cuando los nombres de propiedad son conocidos y simples, se utiliza el punto. Y si necesitamos algo más complejo, entonces cambiamos a corchetes.

En el código real, a menudo usamos variables existentes como valores para los nombres de propiedades.

Por ejemplo:

```
function makeUser(name, age) {
  return {
    name: name,
    age: age,
    // ...other properties
  };
}

let user = makeUser("John", 30);
alert(user.name); // John
```

En el ejemplo anterior, las propiedades tienen los mismos nombres que las variables. El caso de uso de hacer una propiedad a partir de una variable es tan común, que hay un de valor de propiedad especial para acortarla.

En lugar de name: name simplemente podemos escribir name, así:

```
function makeUser(name, age) {
  return {
    name, // same as name: name
    age,  // same as age: age
    // ...
  };
}
```

Podemos usar propiedades normales y “especiales” en el mismo objeto:

```
let user = {
  name,  // same as name:name
  age: 30
};
```

6.- Limitaciones de nombres de propiedad

Como ya sabemos, una variable no puede tener un nombre igual a una de las palabras reservadas de idioma como "for", "let", "return", etc.

Pero para una propiedad de objeto, no existe tal restricción:

```
// these properties are all right
let obj = {
  for: 1,
  let: 2,
  return: 3
};

alert( obj.for + obj.let + obj.return );  // 6
```

En resumen, no hay limitaciones en los nombres de propiedades. Pueden ser cadenas o símbolos (un tipo especial para identificadores, que se cubrirán más adelante).

7.- Comprobar la existencia de propiedades

Una característica notable de los objetos en JavaScript, en comparación con muchos otros lenguajes, es que es posible acceder a cualquier propiedad. ¡No habrá error si la propiedad no existe!

La lectura de una propiedad no existente solo devuelve undefined. Entonces podemos probar fácilmente si la propiedad existe:

```
let user = {};

alert( user.noSuchProperty === undefined ); // true means "no such
property"
```

También hay un operador especial "in"para eso.
La sintaxis es:

```
"key" in object
```

Por ejemplo:

```
let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age exists
alert( "blabla" in user ); // false, user.blabla doesn't exist
```

Tenga en cuenta que en el lado izquierdo in debe haber un nombre de propiedad Eso suele ser una cadena entrecomillada.

Si omitimos las comillas, eso significa una variable, debe contener el nombre real que se comprobará.

Por ejemplo:

```
let user = { age: 30 };

let key = "age";
alert( key in user ); // true, property "age" exists
```

¿Por qué existe el operador in? ¿No es suficiente para comparar undefined? Bueno, la mayoría de las veces la comparación undefined funciona bien. Pero hay un caso especial cuando falla, pero "in" funciona correctamente.

Es cuando existe una propiedad de objeto, pero almacena undefined:

```
let obj = {
  test: undefined
};

alert( obj.test ); // it's undefined, so - no such property?

alert( "test" in obj ); // true, the property does exist!
```

En el código anterior, la propiedad obj.test existe técnicamente. Entonces el operador in funciona bien.

Situaciones como esta ocurren muy raramente, porque undefined no debería asignarse explícitamente. Usamos principalmente null para valores "desconocidos" o "vacíos".

8.- El bucle "for... in"

Para recorrer todas las propiedades de un objeto, existe una forma especial del bucle: for..in. Esto es algo completamente diferente del for(;;) que vimos en los bucles.

La sintaxis:

```
for (key in object) {
  // executes the body for each key among object properties
}
```

Por ejemplo, vamos a recorrer todas las propiedades de user:

```
let user = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let key in user) {
  // keys
  alert( key );  // name, age, isAdmin
  // values for the keys
  alert( user[key] ); // John, 30, true
}
```

Todas las construcciones "for" nos permiten declarar la variable de bucle dentro del bucle, como let key aquí.

Además, podríamos usar otro nombre de variable aquí en lugar de key. Por ejemplo, "for (let prop in obj)" también se usa ampliamente.

9.- Ordenación de objetos

¿Se ordenan los objetos? En otras palabras, si hacemos un bucle sobre un objeto, ¿obtenemos todas las propiedades en el mismo orden en que se agregaron? ¿Podemos confiar en esto?

La respuesta corta es: "**ordenado de una manera especial**": las propiedades enteras se ordenan, otras aparecen en orden de creación.
Como ejemplo, consideremos un objeto con los códigos telefónicos:

```
let codes = {
  "49": "Germany",
  "41": "Switzerland",
  "44": "Great Britain",
  // ..,
  "1": "USA"
};

for (let code in codes) {
  alert(code); // 1, 41, 44, 49
}
```

EE.UU. (1) va primero
luego Suiza (41) y así sucesivamente.
Los códigos telefónicos van en orden ascendente, porque son enteros. Así lo vemos 1, 41, 44, 49.

¿Propiedades enteras? ¿Que es eso?
El término "propiedad de enteros" aquí significa una cadena que se puede convertir a un entero sin un cambio.

Entonces, "49" es un nombre de propiedad entero, porque cuando se transforma en un número entero y viceversa, sigue siendo el mismo. Pero "+49" y "1.2" no son:
Por otro lado, si las claves no son enteras, se enumeran en el orden de creación. Por ejemplo:

```
let user = {
  name: "John",
  surname: "Smith"
};
user.age = 25; // add one more

// non-integer properties are listed in the creation order
for (let prop in user) {
  alert( prop ); // name, surname, age
}
```

Entonces, para solucionar el problema con los códigos telefónicos, podemos "hacer trampa" haciendo que los códigos no sean enteros. Agregar un "+"signo más antes de cada código es suficiente.
Ahora funciona según lo previsto.

```
let codes = {
  "+49": "Germany",
  "+41": "Switzerland",
  "+44": "Great Britain",
  // ..,
  "+1": "USA"
};
for (let code in codes) {
  alert( +code ); // 49, 41, 44, 1
}
```

10.- Métodos de objeto

Los objetos generalmente se crean para representar entidades del mundo real, como usuarios, pedidos, etc.

```
let user = {
  name: "John",
  age: 30
};
```

En el mundo real, un usuario puede actuar o realizar acciones, por ejemplo: seleccionar algo del carrito de compras, iniciar sesión, cerrar sesión, etc.

Las acciones están representadas en JavaScript por funciones en propiedades.
Ejemplos de métodos

```
let user = {
  name: "John",
  age: 30
};
user.sayHi = function() {
  alert("Hello!");
};
user.sayHi(); // Hello!
```

Aquí acabamos de utilizar una **expresión de función** para crear la función y asignarla a la propiedad user.sayHi del objeto. Podemos llamarlo igual que vimos en las expresiones de función
Una función que es propiedad de un objeto se llama **método**.
Por supuesto, podríamos usar una función previamente declarada como método

```
let user = {
  // ...
};
// first, declare
function sayHi() {
  alert("Hello!");
};
// then add as a method
user.sayHi = sayHi;

user.sayHi(); // Hello!
```

Muy importante a la función se la llama por el nombre pero sin las comillas

11.- This

Es común que un método de objeto necesite acceder a la información almacenada en el objeto para hacer su trabajo.

Por ejemplo, el código dentro user.sayHi() puede necesitar el nombre de user. Para acceder al objeto, un método puede usar la palabra resrevada **this**.

El valor de this es el objeto "antes del punto", el que se usa para llamar al método. Aquí durante la ejecución de user.sayHi(), el valor de this será user.

```
let user = {
  name: "John",
  age: 30,

  sayHi() {
    // "this" is the "current object"
    alert(this.name);
  }

};

user.sayHi(); // John
```

Técnicamente, también es posible acceder al objeto sin hacer referencia a this y hacerlo a través de la variable externa:

```
let user = {
  name: "John",
  age: 30,

  sayHi() {
    alert(user.name); // "user" instead of "this"
  }

};
```

Pero ese código no es confiable. Si decidimos copiar user a otra variable, por ejemplo, admin = user y sobrescribir user con otra cosa, accederá al objeto incorrecto.

Además en POO existen los métodos setters que son los que deberían modificar el estado de una propiedad desde dentro del objeto y no hacerlo desde fuera de él, para ello los setters utilizan this

11.1.- This en JavaScript

En JavaScript, la palabra clave se this se comporta diferente de la mayoría de los otros lenguajes de programación. Se puede usar en cualquier función.
No hay error de sintaxis en el siguiente ejemplo:

```
function sayHi() {
  alert( this.name );
}
```

El valor de this se evalúa durante el tiempo de ejecución, según el contexto.

Por ejemplo, en JavaScript podemos utilizar una función para gestionar el evento click de una serie de botones de nuestra web, es decir no vamos a crear una función para cada botón, sino que todos tendrán la misma función. This dentro de esa función permitirá saber cuál es el botón que la ha llamado y en función de eso realizar las acciones necesarias.

En otros lenguajes de programación this se utiliza dentro del objeto para hacer referencia a las propiedades y métodos de ese objetol.

En JavaScript this se utiliza para más cosas. En funciones su valor se evalúa en el momento de la llamada con el elemento que realiza la llamada a la función. Es decir dentro de la función this es como si fuera el elemento que llamo a la función

El concepto de tiempo de ejecución evaluado this tiene ventajas y desventajas. Por un lado, una función puede reutilizarse para diferentes objetos. Por otro lado, la mayor flexibilidad crea más posibilidades de errores.

Aquí nuestra posición no es juzgar si esta decisión de diseño del lenguaje es buena o mala. Entenderemos cómo trabajar con él, cómo obtener beneficios y evitar problemas.

12.- JSON (JavaScript Object Notation)

JavaScript Object Notation (JSON) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o vice versa).

Se puede considerar también como un estándar para intercambiar información entre plataforma.

Inicialmente se utilizaba XML, pero con la evolución de la Web el XML se ha convertido en un lenguaje muy pesado para un simple intercambio de datos.

Prácticamente todos los lenguajes de programación incluyen librerías para poder trabajar con estos lenguajes de marcas.

JSON es un formato de datos basado en texto que sigue la sintaxis de objeto de JavaScript, popularizado por Douglas Crockford. Aunque es muy parecido a la sintaxis de objeto literal de JavaScript, puede ser utilizado independientemente de JavaScript, y muchos entornos de programación poseen la capacidad de leer (convertir; parsear) y generar JSON.

Un objeto JSON puede ser almacenado en su propio archivo, que es básicamente sólo un archivo de texto con una extension .json, y una MIME type de application/json.

JSON es útil cuando se quiere transmitir datos a través de una red. Debe ser convertido a un objeto nativo de JavaScript cuando se requiera acceder a sus datos. Ésto no es un problema, dado que JavaScript posee un objeto global JSON que tiene los métodos disponibles para convertir entre ellos.

12.1.- La sintaxis de JSON

JSON define seis tipos de valores: objects, arrays, strings, numbers, booleans y el valor especial null. Los espacios (espacios en blanco, tabuladores, retornos de carro y nueva línea) pueden introducirse antes o después de cualquier valor, sin afectar a los valores representados. Esto hace que un texto JSON sea mucho más fácil de leer por humanos.

Un objeto JSON es un contenedor, no ordenado de parejas clave/valor. Una clave puede ser un string, y un valor puede ser un valor JSON (tanto un array como un objeto). Los objetos JSON se pueden anidar hasta cualquier profundidad. Un array JSON es una secuencia ordenada de valores, donde un valor puede ser un valor JSON (tanto un array como un objeto).

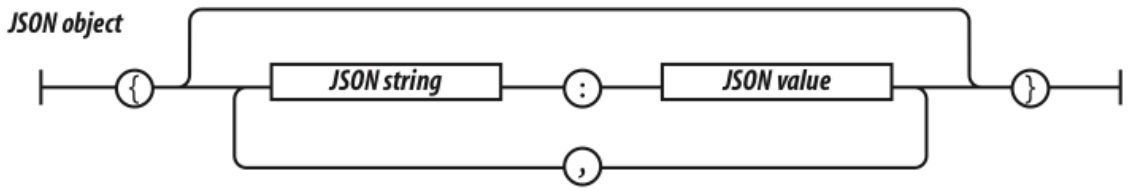
La gran mayoría de lenguajes incluyen características para trabajar de manera cómoda con valores JSON en ambos sentidos: partiendo de un objeto u array para convertirlo a una cadena de caracteres, o a partir de una cadena de caracteres, obtener los valores JSON.

En esencia los caracteres que definen un objeto JSON son:

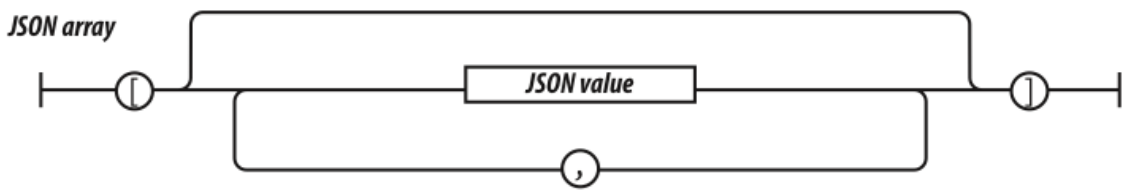
- {} representan un objeto
- [] representan un array
- “” se utilizan para el nombre de las propiedades
- : se utilizan para separar el par propiedad y valor
- , se utilizan para separar los pares propiedad y valor dentro de un objeto, y para separar las propiedades del objeto

La sintaxis de los valores JSON es la siguiente:

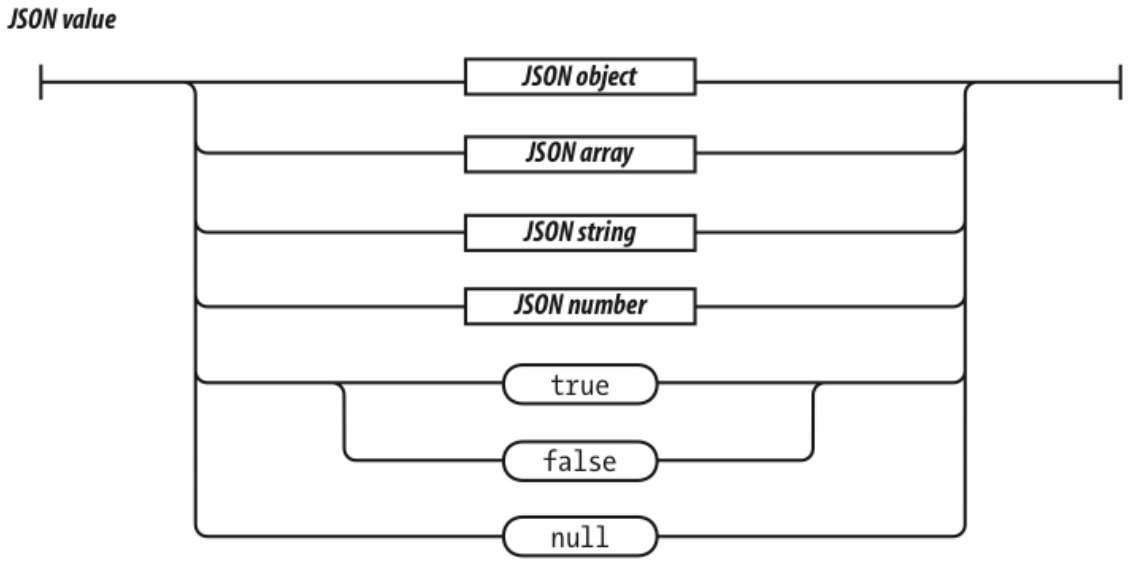
Para definir un objeto



Para definir un array



Tipos de valores admitidos en JSON



Un ejemplo de objeto sencillo en JSON

```
employee={ "firstName":"John" , "lastName":"Doe" }
```

Aqui tenemos un objeto JSON sencillo con dos campos de tipo texto. Es la representación que hemos visto de los objetos.

Un ejemplo de array en JSON

```
department= { "name": "Web development",
              "employees": [
                            { "firstName":"John" , "lastName":"Doe" },
                            { "firstName":"Anna" , "lastName":"Smith" },
                            { "firstName":"Peter" , "lastName":"Jones" }
                           ]
            }
```

Aqui tenemos un objeto JSON formado por dos campos, el primero name de tipo texto y el Segundo campo llamado employees de tipo array con tres elementos de tipo objeto formados por dos campos firstname y lastname

Un ejemplo con anidación en varios niveles

```
    superHeroes= {
                "squadName": "Super hero squad",
                "homeTown": "Metro City",
                "formed": 2016,
                "secretBase": "Super tower",
                "active": true,
                "members": [
                            {
                             "name": "Molecule Man",
                             "age": 29,
                             "secretIdentity": "Dan Jukes",
                             "powers": [
                                       "Radiation resistance",
                                       "Turning tiny",
                                       "Radiation blast"
                                       ]
                             },
                             {
                             "name": "Madame Uppercut",
                             "age": 39,
                             "secretIdentity": "Jane Wilson",
                             "powers": [
                                       "Million tonne punch",
                                       "Damage resistance",
                                       "Superhuman reflexes"
                                       ]
                             },
                             {
                              "name": "Eternal Flame",
                              "age": 1000000,
                              "secretIdentity": "Unknown",
                              "powers": [
                                        "Immortality",
                                        "Heat Immunity",
                                        "Inferno",
                                        "Teleportation",
                                        ]
                              }
                            ]
                }
```

En este ejemplo hemos creado un objeto con varias propiedades con valores de los diferentes tipos.

La propiedad members es un array de objetos con varias propiedades en la que powers es otra propiedad de tipo array de valores string

Podríamos acceder a los elementos del objeto JSON como hemos visto para los objetos con la notación del punto o a través de los corchetes.
Lo más habitual es acceder a través de la notación del punto utilizada en todos los lenguajes para el acceso a los valores almacenados en las propiedades de los objetos.

```
superHeroes.homeTown
superHeroes['active']
```

Acceso a las propiedad homeTown a través de la notación del punto
Acceso a las propiedad active a través de corchetes

Un acceso más complicado debido a la estructura en varios niveles de nuestro objeto superHeroes podría ser

```
superHeroes['members'][1]['powers'][2]
```

Aquí estamos accediendo al 3 poder del segundo miembro

1. Primero el nombre de la variable superHeroes.
2. Dentro de esta variable para acceder a la propiedad members utilizamos ["members"].
3. Members contiene un array formado por objetos. Para acceder al segundo objeto dentro de este array se utiliza [1].
4. Dentro de este objeto, para acceder a la propiedad powers utilizamos ["powers"].
5. Dentro de la propiedad powers existe un array que contiene los superpoderes del héroe seleccionado. Para acceder al tercer superpoder se utiliza [2].

También podríamos haber utilizado

```
superHeroes.members[1].powers[2]
```

12.2.- Métodos JSON

JSON (JavaScript Object Notation) es un formato general para representar valores y objetos. Se describe como en el estándar RFC 4627 Inicialmente se creó para JavaScript, pero muchos otros lenguajes también tienen librerías para manejarlo. Por lo tanto, es fácil usar JSON para el intercambio de datos entre el cliente y el servidor

Como hemos visto JSON es muy útil y hoy en día es el estándar de comunicación para el desarrollo web. Las ventajas de JSON respecto otros formatos son varías, principalmente su ligereza y que simplemente son caracteres en una sintaxis muy sencilla.

A la hora de trabajar nos interesa utilizar toda la potencia de los objetos para poder acceder a sus propiedades y métodos, pero a su vez a la hora de mandar la información al servidor o de almacenarla internamente en una **cookie** o usando **localSorage** nos interesa que ese objeto sea convertido a una cadena de texto. A su vez también cuando recibamos información desde el servidor o leamos una cookie estaremos leyendo cadenas de texto, pero después de leerlas las querremos utilizar en nuestros scripts como objetos.

Una vez visto esto se ve claramente que vamos a tener que transformar nuestros objetos JSON en cadenas de texto y viceversa. Para ello JavaScript dispone de métodos adecuados.

12.2.1.- JSON.stringify

Digamos que tenemos un objeto complejo, y nos gustaría convertirlo en una cadena, enviarlo a través de una red o simplemente enviarlo con fines de registro.

JSON.stringify convierte un objeto en un JSON (cadena de texto con la sintaxis JSON)

```
let student = {
  name: 'John',
  age: 30,
  isAdmin: false,
  courses: ['html', 'css', 'js'],
  wife: null
};
```

```
let json = JSON.stringify(student);

alert(typeof json); // we've got a string!

alert(json);
/* JSON-encoded object:
{
  "name": "John",
  "age": 30,
  "isAdmin": false,
  "courses": ["html", "css", "js"],
  "wife": null
}
*/
```

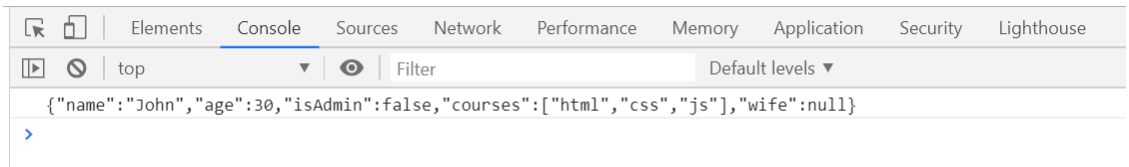
En el ejemplo anterior tendriamos el objeto student convertido a una variable de texto llamada json

También se puede comprobar el resultado por la consola para ver los diferentes tipos de datos.

```
console.log(student)
```

Devuelve un objeto

```
console.log (JSON.stringify(student));
```



Devuelve un string

JSON.stringify elimina todas las funciones que aparecen en el objeto.

JSON es una especificación independiente del lenguaje de **solo datos**, por lo que se omiten algunas propiedades de objeto específicas de JavaScript.

Se eliminan en la conversión a string:

- Propiedades de la función (métodos).
- Propiedades simbólicas.
- Propiedades que almacenan undefined.

```
let user = {
  sayHi() { // ignored
    alert("Hello");
  },
  [Symbol("id")]: 123, // ignored
  something: undefined // ignored
};

alert( JSON.stringify(user) ); // {} (empty object)
```

12.2.1.- JSON.parse

Para decodificar una cadena JSON, necesitamos otro método llamado JSON.parse. Este método lo que realiza es la conversión de una cadena de texto en formato JSON a un objeto.

Después de aplicar este método podremos acceder al objeto como si fuera un objeto creado con propiedades y su acceso será como el visto para objetos utilizando la notación del punto o los corchetes.

```
let userData = '{ "name": "John", "age": 35, "isAdmin": false, "friends": [0,1,2,3] }';

let user = JSON.parse(userData);

alert( user.friends[1] ); // 1
```

Destacar que **userData** esta definido **entre comillas** porque realmente es un **string**

Esta conversión puede generar algún resultado no esperado en función de que la cadena de texto contenga algún error en la sintaxis JSON. Es decir si la cadena de texto no está bien definida según la sintaxis JSON evidentemente la conversión no se realizara correctamente.

Errores habituales

```
let json = `{
  name: "John",                     // mistake: property name without quotes
  "surname": 'Smith',               // mistake: single quotes in value (must be double)
  'isAdmin': false                  // mistake: single quotes in key (must be double)
  "birthday": new Date(2000, 2, 3), // mistake: no "new" is allowed, only bare values
  "friends": [0,1,2,3]              // here all fine
}`;
```

Además, JSON no admite comentarios. Agregar un comentario a JSON lo invalida.

13.- Resumen

- Los objetos son **arrays asociativos** con varias características especiales. Almacenan propiedades (pares **clave-valor**), donde:
  - Las claves de propiedad deben ser cadenas o símbolos (generalmente cadenas).
  - Los valores pueden ser de cualquier tipo.

- Para acceder a una propiedad, podemos usar:
  - La notación de puntos: obj.property.
  - Notación de corchetes obj["property"]. Los corchetes permiten tomar la clave de una variable, como obj[varWithKey].

- Operadores adicionales:
  - Para eliminar una propiedad: delete obj.prop.

- Para comprobar si existe una propiedad con la clave dada: "key" in obj.
- Para iterar sobre un objeto: for (let key in obj)bucle.

- Lo que hemos visto en este capítulo se llama un "objeto simple", o simplemente Object. Hay muchos otros tipos de objetos en JavaScript:
  - Array para almacenar colecciones de datos ordenadas,
  - Date para almacenar la información sobre la fecha y la hora,
  - Error para almacenar la información sobre un error.

  A veces decimos algo como "Tipo array" o "Tipo de fecha", pero realmente no son tipos propios, sino que pertenecen a un solo tipo de datos de "objeto". Y lo extienden de varias maneras.

- Las funciones que se almacenan en las propiedades del objeto se denominan "métodos".
  - Los métodos permiten que los objetos "actúen" como object.doSomething().
  - Los métodos pueden hacer referencia al objeto como this.
  - El valor de this se define en tiempo de ejecución.

- Cuando se declara una función, puede usarse this, pero ese this no tiene valor hasta que se llama a la función.

- JSON es un formato de datos que tiene su propio estándar independiente y bibliotecas para la mayoría de los lenguajes de programación.
  - JSON admite objetos simples, matrices, cadenas, números, booleanos y null.
  - JavaScript proporciona métodos JSON.stringify para serializar en JSON y JSON.parse para leer desde JSON.