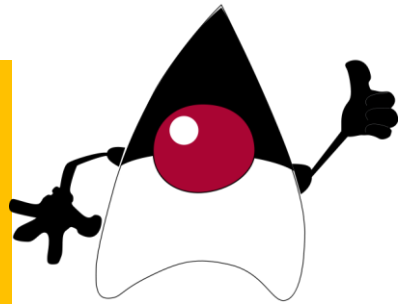
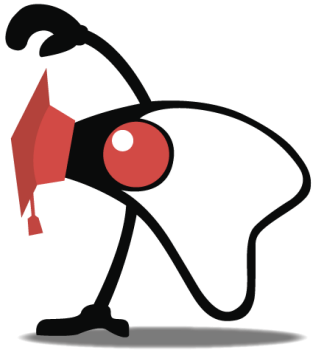
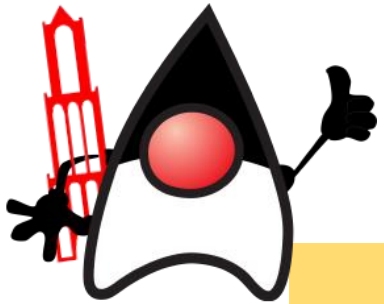
Curso FullStack Python

Codo a Codo 4.0







JavaScript

Parte 6

Objetos

Uno de los aspectos más importantes del lenguaje Javascript es el concepto de **objeto**, puesto que prácticamente todo lo que utilizamos en Javascript, son objetos. Sin embargo, tiene ligeras diferencias con los objetos de otros lenguajes de programación.

¿Por qué hablamos de objetos?

Los objetos son muy potentes en programación, en JavaScript los arrays son objetos y los strings también. En JavaScript existe el tipo de datos llamado objeto como en otros lenguajes, al tener más variables en su interior podemos pensar a los atributos del objeto como propiedades del objeto.

Esto es muy útil, muy práctico, porque ya voy a tener una variable que va a tener asociada toda la información que necesito guardar de una determinada entidad, de un determinado objeto.

El paradigma orientado a objetos habla de objetos porque nosotros estamos más familiarizados en la vida real a interactuar con cosas y las cosas no son más que objetos. Una persona puede ser considerada como objeto en términos de programación porque va a tener propiedades y comportamiento asociado. Al comportamiento nosotros lo vemos a través de los métodos: yo le digo al objeto "*devolveme la información del texto que estás mostrando a través de un botón y me lo devuelve*", a través de un métodos de tipo getter, que me devuelven información del objeto, no lo modifican.

Más información: https://lenguajejs.com/javascript/fundamentos/objetos-basicos/

Objetos

¿Qué son los objetos?

En Javascript, existe un tipo de dato llamado **objeto**. No es más que una ***variable especial*** que puede contener más variables en su interior, es decir más información que una variable común. De esta forma, tenemos la posibilidad de organizar ***múltiples variables*** de la misma temática dentro de un objeto, pensándolas como *atributos o propiedades* del objeto. En Javascript, como en otros lenguajes se pueden crear de esta manera:

```
const objeto = new Object(); // Esto es un objeto «genérico» vacío
```

Los literales de los objetos en Javascript son las llaves {}:

```
const objeto = {};   // Esto es un objeto vacío, es equivalente al anterior, pero es
 más corto, rápido y cómodo, por lo que se aconseja declararlos así.
```

Sintaxis: Los objetos también pueden tener métodos:

JS

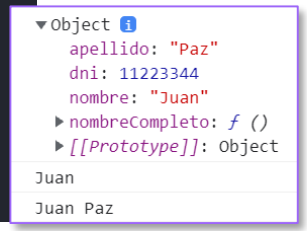
```
object = {
    propiedad1: valor1,
    propiedad2: valor2,
    propiedadn: valorn,
    otraPropiedad: function () {
        return ...;
    }
}
```

Ver ejemplo objetos (.html y .js)

Objetos

Ejemplo:

```JS
var persona = {
    nombre: "Juan", //variable del objeto. Par variable: valor,
    apellido: "Paz",
    dni: 11223344,
    //Método: es una propiedad más
    nombreCompleto: function () { //Esta variable guarda
                                  //el resultado de una función
        return this.nombre + " " + this.apellido; //Función anónima
        //El string que devuelve tiene información del propio objeto
    }
};
console.log(persona) // Imprimo el objeto
console.log(persona.nombre) // Imprimo una propiedad del objeto: Juan
console.log(persona.nombreCompleto()) // Imprimo el resultado de una
                                      // función del objeto: Juan Paz
```



¿Qué es this? La palabra clave ***this*** en JavaScript se refiere al objeto al que pertenece. Utilizamos ***this*** en vez del nombre del objeto porque puede llegar a cambiar y evitamos que se generen conflictos.

For in

For in recorre las propiedades de un objeto, por ejemplo, un vector o un string.

Sintaxis:

```JS
for (var in object) {
    //bloque de codigo a ser ejecutado
}
```

Parámetro	Descripción
var	Necesario. Una variable que itera sobre las propiedades de un objeto.
Object	Necesario. El objeto especificado que se iterará + -

Ejemplo For in con array: Con el for in podemos iterar sobre el objeto array. Requiere que haya un objeto que se pueda recorrer, que se pueda iterar.

```JS
var vec=["a","b","c","d"]
for(var i in vec){
  console.log(i)
}
```

Muestra las posiciones, los índices

0
1
2
3

```JS
for(var i in vec){
    console.log(vec[i])
}
```

Muestra el contenido

a
b
c
d

For in

Ejemplo For in con objetos:

```js
var persona = {
    nombre: "Ana",
    apellido: "Paz",
    edad: 25
};
var x;
for (x in persona) {
    console.log(x);
}
```

```
nombre
apellido
edad
```

Muestra el nombre de la propiedad

```js
var x;
for (x in persona) {
    console.log(x + ": " + persona[x]);
}
```

```
nombre: Ana
apellido: Paz
edad: 25
```

Muestra el nombre de la propiedad y el valor asociado

En el for-in no tenemos que decirle como en el for “desde donde” ni “hasta donde” ni “el paso” porque directamente lo recorre de principio a fin y en ***i*** va guardando temporalmente cada una de las posiciones.

Ver ejemplo for-in-for-of (.html y .js)

For of

Es parecido al for in pero me permite recorrer en forma muy simple a las cadenas de caracteres, de esta manera accedo más fácilmente a cada una de las letras.

Sintaxis:

```JS
for (variable of iterable) {
    //statement
}
```

En cada iteración el elemento (propiedad enumerable) correspondiente es asignado a variable.

Ejemplo:

Defino un string, que es un objeto, y con el ***for of*** por cada elemento del string hago una copia en la variable **letra**. Queda una copia y voy mostrando letra por letra. Es muy útil para recorrer cadenas de caracteres. Tengo en letra cada carácter y lo imprimo uno por uno.

```JS
//Ejemplo 1
let cad = "hola";
for (let letra of cad){
     console.log(letra);
}
```

```
h
o
l
a
```

For of

También puedo recorrer un array de cadenas de texto y un array de números:

```JS
//Ejemplo 2
let nombres=["Juan","Ana","Luis"]
for (let item of nombres){
     console.log(item);
}
```

```
Juan
Ana
Luis
```

```JS
// Ejemplo 3
let numeros = [2, -4, 99]
for (let elem of numeros) {
    console.log(elem);
}
```

```
2
-4
99
```

De esta manera podríamos recorrer una cadena de caracteres de un mail, para saber si es un mail válido. Desde JavaScript podemos validar si tiene un @ y al menos un punto después del @.

Ver ejemplo for-in-for-of (.html y .js)

LocalStorage

Vimos que veníamos trabajando con variables en JavaScript, pero si las modificábamos perdíamos la información. Hay una forma de almacenar esa información en formato texto, en algún lugar local. Cuando se refiere a “local” se refiere al navegador del cliente, es decir quien esté navegando el sitio Web.

Tenemos dos formas de grabar:

- **A nivel local**: Cuando cierro el navegador esa información queda almacenada.
- **A nivel de sesión**: Va a durar lo que dure la sesión, hasta que cierre el navegador.

Estas dos formas nos van a permitir para almacenar información importante del usuario.

Esto me puede servir en este caso: yo sé que en el navegador el usuario siempre accede a esta parte del contenido del sitio, claramente a mi usuario le interesa eso, pero voy con otro usuario y a ese otro usuario le interesará otra parte del sitio. Mi usuario tiene que cargar datos y siempre son los mismos, eso lo puedo almacenar en algún lugar, reconocer que esa persona ya me guardó información con respecto a ella en algún lugar, accedo, levanto y no se lo vuelvo a pedir porque ya la tengo. Esto es una posibilidad de almacenamiento local que nos da JavaScript.

LocalStorage

Definición y uso

Las propiedades **localStorage** y **sessionStorage** permiten guardar pares clave / valor en un navegador web.

El objeto ***localStorage*** almacena datos sin fecha de vencimiento. Los datos no se eliminarán cuando se cierre el navegador y estarán disponibles el próximo día, semana o año.

Los datos almacenados en ***sessionStorage*** son eliminados cuando finaliza la sesion de navegación - lo cual ocurre cuando se cierra la página.

Los datos guardados son siempre formato texto.

Video tutorial "Uso del local storage en JavaScript": https://youtu.be/hb8O0qRqiSk

Más información:

https://www.w3schools.com/jsref/prop_win_localstorage.asp

https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_win_localstorage

LocalStorage

Ejemplo:

JS

```
if (typeof(Storage) !== "undefined") {
    // Store
    localStorage.setItem("apellido", "Perez"); // No existe, lo agrega.
    localStorage.setItem("apellido", "Pérez"); // Existe, lo reemplaza
    localStorage.setItem("nombre", "Juan");
    // Retrieve
    document.getElementById("resultado").innerHTML = localStorage.getItem
("apellido");
} else {
    document.getElementById("resultado").innerHTML = "Su navegador no sop
orta Web Storage.";
}
```

HTML

```
<body>
    <div id="resultado">
    </div>
    <script src="localstorage.js"></script>
</body>
```

localStorage es un objeto que tiene un método asociado setItem.

*Los métodos **getters** y **setters**, me permiten acceder a información del objeto, leerla y grabarla.*

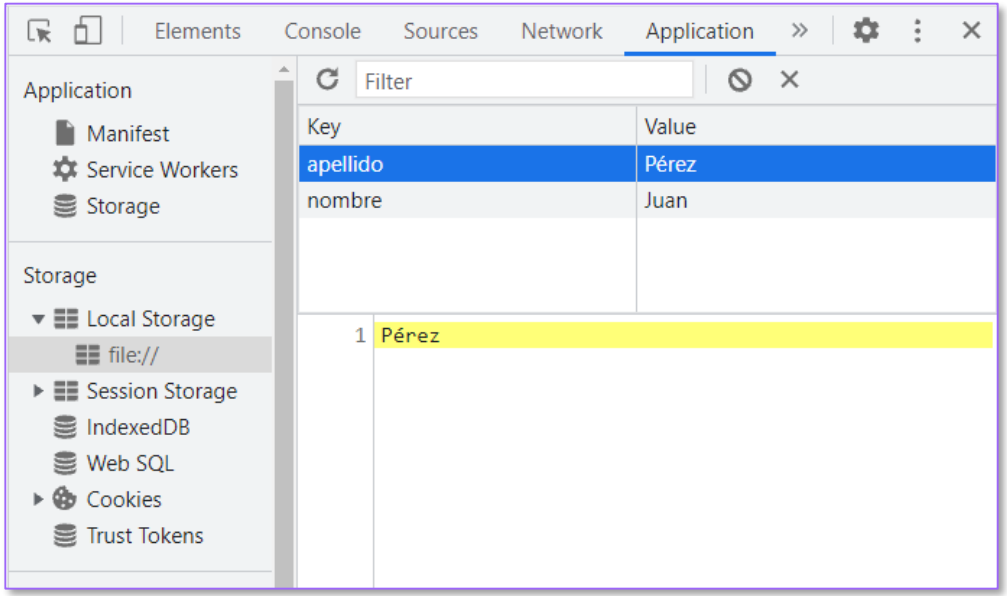
Esta información siempre se guarda en formato de texto, con el par clave / valor.

*En este caso "apellido" no existe, la estoy creando con setItem y le estoy dando el valor **Perez** en primera instancia y luego lo reemplazo por **Pérez**.*

LocalStorage

Ejemplo (continuación):

Para poder ver estos datos accedemos al navegador. Vamos a ver que en el ***body*** se muestra el apellido, gracias a que hemos utilizado `document.getElementById("resultado").innerHTML = localStorage.getItem("apellido");`, pero también vamos a poder verlo al inspeccionar la página yendo a la pestaña Application:



Ver ejemplo localstorage (.html y .js)