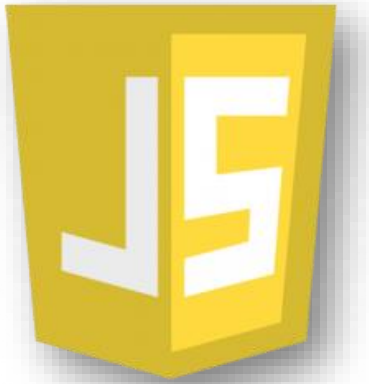
Codo a Codo 4.0

FULL STACK PYTHON

html - css3 - bootstrap - javascript
vue.js - sql - python - django

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

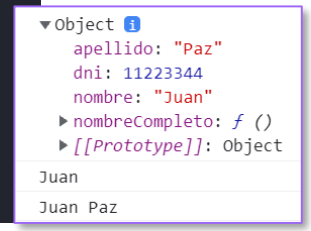
```
object = {
    propiedad1: valor1,
    propiedad2: valor2,
    propiedadn: valorn,
    otraPropiedad: function () {
        return ...;
    }
}
```

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.

String

En programación, cuando hablamos de una variable que posee información de texto, decimos que su tipo de dato es String. En Javascript, es muy sencillo crear una variable de texto, hay dos formas de hacerlo:

Constructor	Descripción
new String(s)	Crea un objeto de texto a partir del texto ***s*** pasado por parámetro.
's'	Simplemente, el texto entre comillas. Notación preferida.

Los String son tipos de datos primitivos, y como tal, es más sencillo utilizar los literales que la notación con new. Para englobar los textos, se pueden utilizar comillas simples ', comillas dobles " o backticks ` (o comilla invertida o francesa). **Podemos pensar el String como un "*array de caracteres*".**

JS

```
// Literales
const texto1 = "¡Hola a todos!";
const texto2 = "Otro mensaje de texto";

// Objeto
const texto1 = new String("¡Hola a todos!");
const texto2 = new String("Otro mensaje de texto");
```

JavaScript llama String a una cadena de caracteres o a un solo caracter

String | Propiedades y métodos

Propiedades

Propiedad	Descripción
.length	Devuelve el número de caracteres de la variable de tipo string en cuestión

Métodos

Método	Descripción
.charAt(pos)	Devuelve el carácter en la posición pos de la variable.
.concat(str1, str2...)	Devuelve el texto de la variable unido a str1, a str2...
.indexOf(str)	Devuelve la primera posición del texto str.
.indexOf(str, from)	Idem al anterior, partiendo desde la posición from.
.lastIndexOf(str, from)	Idem al anterior, pero devuelve la última posición.

Para estos métodos ver ejemplo strings (.html y .js)

String | Métodos | charAt y concat

```
var cad= "hola como estas";
```
JS

h	o	l	a		c	o	m	o		e	s	t	a	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Cada carácter está representado por una posición

.charAt(pos)

```
document.write("CHARAT <br>");
document.write(cad.charAt(0)); // devuelve "h"
var pos1= cad[1]; //devuelve o
var pos2= cad[20]; //indefinido
document.write(pos1); //devuelve o
document.write(pos2); //indefinido
```
JS

CHARAT
h
o
undefined

Charat *permitirá mostrar un carácter determinado de acuerdo a una posición. Podemos guardarlo en una variable y luego mostrarlo en el documento HTML, en la consola o luego utilizarlo en otra tarea.*

.concat(str1, str2...)

```
document.write(cad.concat(". ¿todo bien?", " más texto"));
```
JS

Concat *agregará a la cadena anterior más texto, que pueden estar separados por comas.*

CONCAT
hola como estas. ¿todo bien? más texto

String | Métodos | indexOf y lastIndexOf

```
var cad= "hola como estas";
```
JS

h	o	l	a		c	o	m	o		e	s	t	a	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Cada carácter está representado por una posición

. indexOf(str), .indexOf(str, from) y .lastIndexOf(str, from)

```
document.write(cad.indexOf("a"));
document.write(cad.indexOf("a",4));
document.write(cad.lastIndexOf("o"));
document.write(cad.lastIndexOf("o", 7));
```
JS

INDEXOF
3
13

LASTINDEXOF
8
6

***IndexOf** devuelve 3 porque la letra "a" se encuentra en esa posición, pero al considerar desde la posición 4, representada por un espacio, la posición de "a" será la 13.*

*Por su parte, **lastIndexOf** devolverá la última posición de la letra o (la última posición de "o" en **como**), pero al considerar desde la posición 7 la última aparición será la 6, ya que considera de derecha a izquierda.*

String | Más métodos

Método	Descripción
.repeat(n)	Devuelve el texto de la variable repetido n veces.
.toLowerCase()	Devuelve el texto de la variable en minúsculas.
.toUpperCase()	Devuelve el texto de la variable en mayúsculas.
.trim()	Devuelve el texto sin espacios a la izquierda y derecha.
.replace(str, newstr)	Reemplaza la primera aparición del texto str por newstr.
.substr(ini, len)	Devuelve el subtexto desde la posición **ini** hasta **ini+len**.
.substring(ini, end)	Devuelve el subtexto desde la posición ini hasta end.

Para estos métodos ver ejemplo strings-metodos (.html y .js)

String | Más métodos | repeat, toLowerCase y toUpperCase

```JS
var cad= "Aprendiendo JavaScript ";
```

.repeat(n)

```JS
document.write(cad.repeat(4));
```

Repeat *repetirá una* ***n*** *cantidad de veces la cadena de texto*

REPEAT
Aprendiendo JavaScript Aprendiendo JavaScript Aprendiendo JavaScript Aprendiendo JavaScript

.toLowerCase() y .toUpperCase()

```JS
document.write(cad.toLowerCase());
document.write(cad.toUpperCase());
```

TOLOWERCASE
aprendiendo javascript

TOUPPERCASE
APRENDIENDO JAVASCRIPT

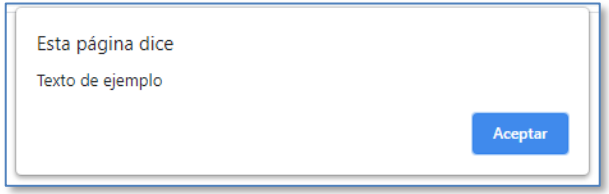
*Estos métodos convierten a mayúsculas (**toUpperCase**) y minúsculas (**toLowerCase**) una cadena de texto*

String | Más métodos | trim y replace

```
var cad2= "        Texto de ejemplo"
```
JS

.trim()

```
alert(cad2.trim());
```
JS

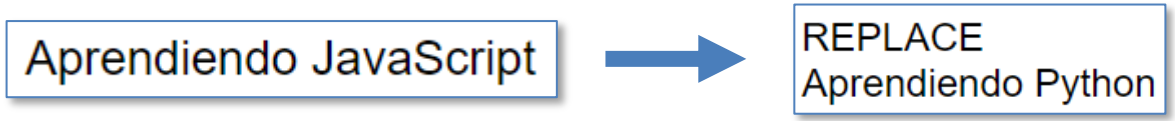


En este caso se eliminan los espacios sobrantes de la cadena de texto y se muestran en un alert.

.replace(str, newstr)

```
document.write(cad.replace("JavaScript", "Python"));
```
JS

*Dentro de una cadena de caracteres, **replace** sustituye una subcadena por otra*



String | Más métodos | substr y substring

```JS
var cad= "Aprendiendo JavaScript ";
```

.substr(ini, len)

```JS
document.write(cad.substr(12, 4));
```

SUBSTR
Java

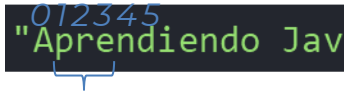
*En este caso, extraemos desde la posición 12 los 4 caracteres (largo) que conforman la palabra **Java** de la palabra JavaScript.*

.substring(ini, end)

```JS
document.write(cad.substring(1, 4));
```

SUBSTRING
pre

Aquí extraeremos desde el caracter 1 hasta el 4 (no inclusive)



Plantilla de cadena de caracteres (template string)

Las ***Template Strings*** utilizan las comillas invertidas o backticks para delimitar sus contenidos, en vez de las tradicionales comillas simples o dobles de las cadenas de texto normales.

Las principales funcionalidades que aportan las Template Strings son:

- Interpolación de cadenas.
- Posibilidad de incluir (y evaluar) expresiones dentro de cadenas.
- Definición de cadenas de texto en varias líneas sin tener que usar hacks.
- Formatear cadenas de manera avanzada.
- Cadenas etiquetadas.

```JS
// esto es una Template String
var saludo = `¡Hola Mundo!`;
// esto es una cadena normal con comillas simples
var saludo = '¡Hola Mundo!';
// esto es una cadena normal con comillas dobles
var saludo = "¡Hola Mundo!";
```

*Para colocar **backticks** (comillas hacia atrás) utilizamos **ALT + 96***

Plantilla de cadena de caracteres (template string)

Interpolación de cadenas

Una de las mejores características de las ***Template Strings*** es la interpolación de cadenas. La interpolación permite utilizar cualquier expresión válida de JavaScript (como por ejemplo la suma de dos variables) dentro de una cadena y obtener como resultado la cadena completa con la expresión evaluada.

Las partes variables de una *Template String* se denominan *placeholders* y utilizan la sintaxis **${ }** para diferenciarse del resto de la cadena. Ejemplo:

JS

```
// Sustitución simple de cadenas
var nombre = "Juan";
console.log(`¡Hola ${nombre}!`);
// resultado => "¡Hola Juan!"
```

Plantilla de cadena de caracteres (template string)

Como dentro de las partes variables de la cadena se puede incluir cualquier expresión válida de JavaScript, en la práctica sirven para mucho más que mostrar el contenido de una variable. En los siguientes ejemplos se muestran cómo interpolar algunas operaciones matemáticas sencillas:

JS

```
var a = 10;
var b = 10;
console.log(`¡JavaScript se publicó hace ${a+b} años!`);
// resultado => ¡JavaScript se publicó hace 20 años!

console.log(`Existen ${2 * (a + b)} frameworks JavaScript y no ${10 * (a + b)}.`);
// resultado => Existen 40 frameworks JavaScript y no 200.
```

Plantilla de cadena de caracteres (template string)

Dentro de un valor interpolado también se puede utilizar cualquier función:

```JS
function fn() { return "Este es el resultado de la función"; }
console.log(`Hola "${fn()}" Mundo`);
// resultado => Hola "Este es el resultado de la función" Mundo
```

La sintaxis ${} también funciona con expresiones que invocan métodos y acceden a propiedades:

```JS
var usuario = { nombre: 'Juan Perez' };
console.log(`Estás conectado como ${usuario.nombre.toUpperCase()}.`);
// resultado => "Estás conectado como JUAN PEREZ.";

var divisa = 'Euro';
console.log(`Los precios se indican en ${divisa}. Utiliza nuestro conversor
 para convertir ${divisa} en tu moneda local.`);
// resultado => Los precios se indican en Euro. Utiliza nuestro conversor p
ara convertir Euro en tu moneda local.
```

Plantilla de cadena de caracteres (template string)

La ventaja de usar *template strings* es el uso de expresiones incrustadas y la posibilidad de interpolación de cadenas de texto con ellas, facilitando la concatenación de valores. Ejemplo:

```js
function suma(a,b){
    return a+b;
   }

var a=Number(prompt("ingrese un numero a:"));
var b=Number(prompt("ingrese un numero b:"));
console.log("la suma entre " + a + " y " + b + " es: " + suma(a,b));  //  la suma entre 12 y 21 es: 33
console.log(`la suma entre ${a} y ${b} es: ${suma(a,b)}`);          //  la suma entre 12 y 21 es: 33
```

JS

También podremos escribir una cadena en varias líneas, sin la necesidad de usar \ o + para concatenar:

```js
var cadena = `Línea número 1 de la cadena
Línea número 2 de la cadena`;
console.log(cadena);
```

JS

```
Línea número 1 de la cadena
Línea número 2 de la cadena
```

Fuente (para ampliar): Clic aquí

Ver archivos template-string y template-string2 (.html y .js)

Ejemplo de uso de template string:

- Agregar JavaScript a un sitio, y con ***template string*** modificar el header y footer del HTML por Javascript.

Ver archivos de la carpeta ej-template-string

Ejercicios

- Del archivo **“Actividad Práctica - JavaScript Unidad 2”** están en condiciones de hacer los ejercicios: 19 al 29. Los ejercicios **NO** son obligatorios.