<codoacodo/>

Agencia de Aprendizaje a lo largo de la vida

FULL STACK PYTHON
Clase 16

Javascript 4

BA

Objetos

Les damos la bienvenida

Vamos a comenzar a grabar la clase

<codoacodo/>

Clase 15

Programación modular con funciones

- Funciones. ¿Qué son? Scope global y local.
- Programación modular vs. Funciones.
- Función anónima y función flecha.
- Callbacks y clausuras.

Clase 16

Objetos

- Objetos. ¿Qué son y cómo se usan?
- Propiedades y métodos.
- Función constructora.
- El objeto String y sus métodos.
- El objeto Math, sus propiedades y métodos.

Clase 17

Arrays, Storage y JSON

- Arrays.
- Funciones para operar arrays.
- Trabajar con array de objetos.
- Web Storage.
- JSON. Formato y ejemplos de uso.

Objetos

Prácticamente todo lo que utilizamos en Javascript son objetos. Los **objetos** en **JavaScript**, como en tantos otros lenguajes de programación, se pueden comparar con objetos de la vida real. Las variables, por ejemplo, son objetos de diferentes tipos.
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: le solicitamos al objeto información sobre un elemento mediante un botón y el objeto la devuelve.

Objetos | ¿Qué son?

Un **objeto de JavaScript** tiene **propiedades** asociadas a él. Una propiedad de un objeto se puede explicar como una variable asociada al objeto. Las propiedades de un objeto básicamente son lo mismo que las variables comunes de JavaScript, excepto por el nexo con el objeto. Las propiedades de un objeto definen las características del mismo. Se accede a las propiedades de un objeto con la **notación punto**:

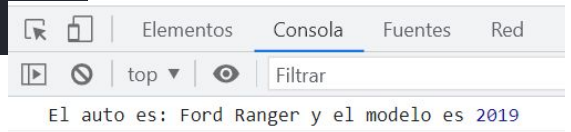
```
//nombreDelObjeto.propiedadDelObjeto
texto1 = objectName.propertyName
```

Tanto el nombre del objeto como el nombre de la propiedad son sensibles a mayúsculas y minúsculas.

Objetos | ¿Cómo se crean?

Un objeto se puede crear usando el operador de asignación. Y puedes definir propiedades simplemente asignándoles un valor. Por ejemplo, vamos a crear un **objeto** llamado **miAuto** y le vamos a asignar **propiedades** denominadas **marca**, **tipo**, y **modelo** de la siguiente manera:

```
// Creamos el objeto
var miAuto = new Object()
// Creamos las propiedades
miAuto.marca = 'Ford'
miAuto.tipo = 'Ranger'
miAuto.modelo = 2019
console.log("El auto es:", miAuto.marca, miAuto.tipo,"y el modelo
es",miAuto.modelo)
```



Objetos | ¿Cómo se crean?

Se pueden crear también mediante un **iniciador de objeto** (o **literal**), que es una lista delimitada por comas de cero o más pares de nombres de propiedad y valores asociados al objeto, encerrados entre llaves (**{}**):

```
// Creamos el objeto
var miAuto = {
     marca: 'Ford',
     tipo: 'Ranger',
     modelo: 2019
}
```

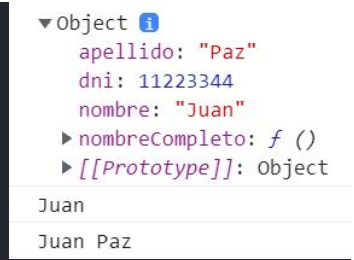
Las propiedades de un objeto que no han sido asociadas a un valor en el momento de la creación del mismo **undefined**.

Objetos

Ejemplo: Creamos y usamos un objeto, con propiedades y métodos.

```
var persona = {
    nombre: "Juan", //variable del objeto. Par variable: valor,
    apellido: "Paz",
    dni: 11223344,
    //Método: es una propiedad más
    nombreCompleto: function () {
        return this.nombre + " " + this.apellido
        //El string que devuelve tiene información del propio objeto, por eso usamos “this”
    }
}

console.log(persona) // Imprimo el objeto
console.log(persona.nombre) // Imprimo una propiedad del objeto: Juan
console.log(persona.nombreCompleto()) // Imprimo el resultado del método: Juan Paz
```



Objetos | Notación de corchetes

Se puede acceder o establecer las propiedades de los objetos mediante la notación de corchetes **[ ]** . En los objetos cada propiedad está asociada con un valor tipo String que se puede utilizar para acceder a ella. Por lo tanto puedes acceder a las propiedades del objeto **miAuto** de la siguiente manera:

```
miAuto['marca'] = 'Ford'
miAuto['tipo'] = 'Ranger'
miAuto['modelo'] = 2019
```

El nombre de una propiedad puede ser cualquier cadena válida de JS. Pero si no es un identificador válido de JS (por ejemplo, comienza con un número) solo se puede acceder utilizando la notación de corchetes.

Objetos | Métodos

Los **métodos** son el equivalente de las funciones, pero dentro de un objeto. Proporcionan al objeto la capacidad de interactuar con otros objetos o con el resto del programa.

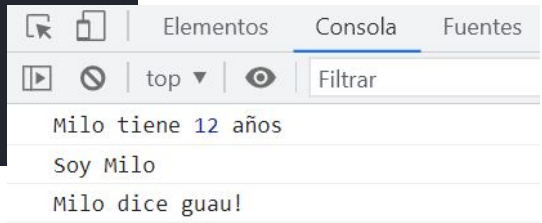
Para escribirlos, colocamos su nombre seguido de paréntesis **( )**. El bloque de código que compone el método se escribe entre llaves **{ }** y pueden devolver resultados mediante **return**, igual que las funciones.

Se invocan desde el resto del programa usando la notación punto, pero usando paréntesis luego de su nombre para diferenciarlas de las propiedades.

Objetos | Métodos

Por ejemplo, veamos cómo definir un nuevo objeto **Perro**, con los métodos **quienSoy()** y **ladrar()**:

```
var perro = {
    nombre: "Milo",
    edad: 12,
    vivo: true,
    quienSoy() {return "Soy " + this.nombre},
    ladrar() {return this.nombre + " dice guau!"}
}
console.log(perro.nombre,"tiene",perro.edad,"años")
console.log(perro.quienSoy())
if (perro.vivo) {
    console.log(perro.ladrar())
}
```



Objetos | Clases

Las **clases** son una suerte de "*molde*" que podemos usar para crear varios objetos del mismo tipo. Usamos un **constructor** y *this* para asignar valores a las **propiedades** de los **objetos instanciados**:

```
// Clase Perro, con su constructor:
class Perro {
    constructor( nombre, edad, vivo){
         this.nombre = nombre
         this.edad   = edad
         this.vivo   = vivo
         }
     }
```

```
// Instanciamos dos objetos clase Perro:
var perro1 = new Perro ("Lola", 4, true)
var perro2 = new Perro ("Lassie", 10, false)

// Modificamos alguna de sus PROPIEDADES:
perro1.nombre = "Toby"
perro2.edad = 6
```

Usamos **this** para asignar valores a las propiedades del objeto que estamos creando con **new Perro**.

Objetos | Función constructora

Para definir un tipo (**clase**) de objeto, creamos una función que especifique su nombre, propiedades y métodos. Supongamos que deseas una clase llamada “Auto” para crear objetos “auto”, y deseas que tenga las siguientes propiedades: marca, tipo y modelo. Podrías escribir la siguiente función:

```
//Función constructora (clase)
function Auto(marca, tipo, modelo) {
   this.marca = marca
   this.tipo = tipo
   this.modelo = modelo
   }
```

```
//Creamos el objeto miAuto
var miAuto = new Auto('Ford','Focus', 2019)
//Creamos el objeto miFurgon
var miFurgon = new Auto('Renault','Traffic', 2010)
```

Observa el uso de **this** para asignar valores a las propiedades del objeto en función de los valores pasados a la función.

Objeto String

String

Cuando hablamos de una variable que posee información de **texto**, decimos que su tipo de dato es **String**. Hay dos formas de crear una variable de texto:

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

Los String son tipos de datos primitivos, y como tal, es más sencillo crearlos de forma literal que usar el constructor **new**. Para delimitar un texto, se pueden utilizar comillas simples, comillas dobles o backticks (o comilla invertida o francesa).

String

Un string puede tener cero, uno o varios caracteres.

```
// Declaración literal
const texto1 = "¡Hola a todos!"
var texto2 = "Otro mensaje de texto"
var vacio = ""
var texto3 = "123" // Cuidado, NO es un Number!

// Declaración con el constructor new String
const texto1 = new String("¡Hola a todos!")
var texto2 = new String("Otro mensaje de texto")
```

String | Propiedades y métodos

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

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.

String | .length y .concat(str1, str2...)

La propiedad **.length** devuelve el número de caracteres de una cadena.

```
// Creamos una variable de texto
var texto1 = "¡Hola a todos!"
largo = texto1.length
console.log(largo)   //14
// Se puede usar directamente:
console.log("Hola".length) //4
```

.concat(str1, str2...) concatena cadenas. Su función es similar al operador “+”:

```
// Creamos una variable de texto
var cad = "¡Hola "
// Concatenamos con otra
var saludo = cad.concat("Codo a Codo!")
// Y en la consola vemos "¡Hola Codo a Codo!"
console.log(saludo)
```

String | charAt(pos)

Charat devuelve el carácter ubicado una posición determinada dentro del string. Podemos guardarlo en una variable, mostrarlo en el documento o en la consola. Cada caracter está almacenado en una posición:

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

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

```
var cad= "hola como estas"
document.write("CHARAT <br>")
document.write(cad.charAt(0)) // devuelve "h"
var pos1= cad[1] //Almaceno en pos1 el caracter 1(“o”)
var pos2= cad[20] //indefinido (no hay elemento nro 20)
document.write(pos1) //devuelve o
document.write(pos2) //undefinded
```

CHARAT
h
o
undefined

String | . indexOf() y .lastIndexOf()

. indexOf(str) devuelve la posición de la primera aparición de **str** dentro de la cadena, **.indexOf(str, from)** hace lo propio, pero a partir de la posición indicada por **from**. Y **.lastIndexOf(str, from)** devuelve el carácter ubicado a partir de la posición indicada por **from** pero partiendo desde el final:

```
var cad = "hola como estas"
document.write(cad.indexOf("a"))         //Muestra “3” en el documento HTML
document.write(cad.indexOf("a",4))       //Muestra “13” en el documento HTML
document.write(cad.lastIndexOf("o"))     //Muestra “8” en el documento HTML
document.write(cad.lastIndexOf("o", 7)) //Muestra “6” en el documento HTML
```

En este ejemplo, **IndexOf** cuenta desde el principio de la cadena, si agregamos un valor más al método empieza a contar desde ese valor. En cambio, **lastIndexOf** cuenta desde el final de la cadena.

String | Más métodos

El objeto String posee una gran cantidad de métodos. Veamos algunos más:

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**.

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.

```
// esto es una Template String
var saludo = `¡Hola Mundo!`
```

Plantilla de cadena de caracteres (template string)

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:

```
// 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:

```
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:

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

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

```
var usuario = {nombre: 'Juan Perez'}
console.log(`Estás conectado como ${usuario.nombre.toUpperCase()}.`)
// Estás conectado como JUAN PEREZ.
var divisa = 'Pesos'
console.log(`Los precios se indican en ${divisa}. Convierte ${divisa} en tu moneda local.`)
//Los precios se indican en Pesos. Convierte Pesos 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:

```
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(a + " + " + b + " es " + suma(a,b)) // 12 + 21 es 33
console.log(`${a} + ${b} es ${suma(a,b)}`) // 12 + 21 es 33
```

Podremos escribir una cadena en varias líneas, sin necesidad de concatenar:

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

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

Objeto Math

Objeto Math

Math es un objeto que tiene propiedades y métodos para constantes y funciones matemáticas. Todas las propiedades y métodos de Math son estáticos (no es necesario llamar al constructor). Estas son las **constantes** disponibles:

Constante	Descripción	Valor
`Math.E`	Número de Euler W	`2.718281828459045`
`Math.LN2`	Logaritmo natural en base 2 W	`0.6931471805599453`
`Math.LN10`	Logaritmo decimal W	`2.302585092994046`
`Math.LOG2E`	Logaritmo base 2 de E	`1.4426950408889634`
`Math.LOG10E`	Logaritmo base 10 de E	`0.4342944819032518`
`Math.PI`	Número PI W o Π	`3.141592653589793`
`Math.SQRT1_2`	Raíz cuadrada de 1/2	`0.7071067811865476`
`Math.SQRT2`	Raíz cuadrada de 2	`1.4142135623730951`

Objeto Math | Métodos matemáticos

Los siguientes métodos matemáticos están disponibles en JS a través del objeto Math.

Algunos de ellos sólo están disponibles en ECMAScript 6:

Método	Descripción	Ejemplo
NUMBER `Math.abs(x)`	Devuelve el valor absoluto W de `x`.	$\|x\|$
NUMBER `Math.sign(x)` ES2015	Devuelve el signo del número: `1` positivo, `-1` negativo	
NUMBER `Math.exp(x)`	Exponenciación W . Devuelve el número `e` elevado a `x`.	e^x
NUMBER `Math.expm1(x)` ES2015	Equivalente a `Math.exp(x) - 1`.	e^x-1
NUMBER `Math.max(a, b, c...)`	Devuelve el número más grande de los indicados por parámetro.	
NUMBER `Math.min(a, b, c...)`	Devuelve el número más pequeño de los indicados por parámetro.	
NUMBER `Math.pow(base, exp)`	Potenciación W . Devuelve el número `base` elevado a `exp`.	$base^{exp}$
NUMBER `Math.sqrt(x)`	Devuelve la raíz cuadrada W de `x`.	$\sqrt{x}$
NUMBER `Math.cbrt(x)` ES2015	Devuelve la raíz cúbica W de `x`.	$\sqrt[3]{x}$
NUMBER `Math.imul(a, b)` ES2015	Equivalente a `a * b`, pero a nivel de bits.	
NUMBER `Math.clz32(x)` ES2015	Devuelve el número de ceros a la izquierda de `x` en binario (32 bits).	

Objeto Math | Métodos matemáticos

Veamos algunos ejemplos de las funciones mencionadas anteriormente:

```
Math.abs(-5); // 5
Math.sign(-5); // -1
Math.exp(1); // e, o sea, 2.718281828459045
Math.expm1(1); // 1.718281828459045
Math.max(1, 40, 5, 15); // 40
Math.min(5, 10, -2, 0); // -2
Math.pow(2, 10); // 1024
Math.sqrt(2); // 1.4142135623730951
Math.cbrt(2); // 1.2599210498948732
Math.imul(0xffffffff, 7); // -7

// Ejemplo de clz32 (count leading zeros)
const x = 1;
"0".repeat(Math.clz32(x)) + x.toString(2);
// Devuelve "00000000000000000000000000000001"
```

```
Abs:  5
Sign:  -1
Exp:  2.718281828459045
Expm1:  1.718281828459045
Max:  40
Min:  -2
Pow:  1024
Sqrt:  1.4142135623730951
Cbrt:  1.2599210498948732
Imul:  -7
clz32:  00000000000000000000000000000001
```

Objeto Math | Método random()

Math.random() retorna un número al azar entre los valores 0 y 1, con 16 decimales. Si queremos obtener un número entero al azar entre los límites a y b, se puede hacer lo siguiente:

```
// Obtenemos un número al azar entre [0, 1) con 16 decimales
let x = Math.random();
// Multiplicamos x por el valor máximo que buscamos (5)
x = x * 5;
// Redondeamos hacia abajo, obtenemos un entero
x = Math.floor(x);
```

```
0.3289342187867974
1.644671093933987
1
```

Este ejemplo almacena en x un valor al azar entre 0 y 5 (5 no incluido). Si presionamos F5 veremos el cambio en la consola.

Objeto Math | Métodos de redondeo

Es muy común necesitar métodos para redondear números y reducir el número de decimales o aproximarse a una cifra concreta. Para ello, de forma nativa, Javascript proporciona los siguientes métodos de redondeo:

Método	Descripción
NUMBER `Math.round(x)`	Devuelve el redondeo de **X** (*el entero más cercano*)
NUMBER `Math.ceil(x)`	Devuelve el redondeo superior de **X**. (*el entero más alto*)
NUMBER `Math.floor(x)`	Devuelve el redondeo inferior de **X**. (*el entero más bajo*)
NUMBER `Math.fround(x)` ES2015	Devuelve el redondeo de **X** (*flotante con precisión simple*)
NUMBER `Math.trunc(x)` ES2015	Trunca el número **X** (*devuelve sólo la parte entera*)

Objeto Math | Métodos de redondeo

Ejemplos de los diferentes métodos de redondeo:

```
// Redondeo natural, el más cercano
Math.round(3.75); // 4
Math.round(3.25); // 3
// Redondeo superior (el más alto)
Math.ceil(3.75); // 4
Math.ceil(3.25); // 4
// Redondeo inferior (el más bajo)
Math.floor(3.75); // 3
Math.floor(3.25); // 3
// Redondeo con precisión
Math.round(3.123456789); // 3
Math.fround(3.123456789); // 3.1234567165374756
// Truncado (sólo parte entera)
Math.trunc(3.75); // 3
Math.round(-3.75); // -4
Math.trunc(-3.75); // -3
```

```
Round (natural):  4
Round (natural):  3
Ceil (superior):  4
Ceil (inferior):  4
Floor (inferior):  3
Floor (inferior):  3
Round (natural):  3
Fround (con precisión):  3.1234567165374756
Trunc (truncado):  3
Round (natural):  -4
Trunc (truncado negativo):  -3
```

Material extra

Artículos de interés

Material de lectura:

- ¿Qué son los objetos?
- Trabajando con objetos
- Uso de For In y For Of
- For In en W3Schools
- For Of en W3Schools

Videos:

- ¿Qué son y cómo crear objetos?
- For, For In y For Of, buenas prácticas

Actividades prácticas:

- Del archivo “**Actividad Práctica - JavaScript Unidad 2**” están en condiciones de hacer los ejercicios: 26 a 29.
- Agregar JavaScript a un sitio, y con **template string** modificar el header y footer del HTML por Javascript.

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- Realizar los Ejercicios de repaso.

Todo en el Aula Virtual.

Muchas gracias por tu atención.

Nos vemos pronto