Práctica 4

Sesiones 9, 10 y 11

Objetivos

- Conocer los fundamentos del lenguaje de programación JavaScript.
- Entender la programación web en el cliente.
- Comprender cómo JavaScript sirve para crear webs dinámicas.
- Ser capaz de desarrollar guiones en JavaScript utilizando todas las prestaciones que ofrece el lenguaje.

Introducción a JavaScript

Conforme la Web gana popularidad, se requiere cada vez más un lenguaje de programación que se ejecute en el cliente. Brendan Eich creó un lenguaje denominado LiveScript a mediados de los 90 del siglo pasado, que llegó a denominarse JavaScript más tarde. En 1997 fue estandarizada la versión 1.1 de este lenguaje, aunque realmente se diseñó un estándar denominado ECMAScript el cual es la base para las implementaciones de JavaScript (realmente es el lenguaje). Actualmente está en vigor la versión 9 de ECMAScript (2018). Éste incluye dos partes más: Document Object model (DOM) y Browser Object Model (BOM). Es un lenguaje interpretado que se ejecuta en el navegador.

Fundamentos de JavaScript

He aquí algunas de las consideraciones básicas del lenguaje de programación:

- El código JavaScript debe ir en un elemento script de HTML:

```
<script type="text/javascript">
        alert("Hola, Mundo");
</script>
```

Ya no es necesario indicar el tipo del guión, pues JavaScript es el lenguaje por defecto.

- También el código JavaScript puede ir en un fichero externo. Para ejecutarlo se emplea el atributo *src* de *script*:

```
<script  src="miGuion.js"></script>
```

- Los guiones de JavaScript pueden ir indistintamente en los elementos *head* o *body*. Lo habitual es colocar las funciones en el primero, o al final de la página. Colocándolo en el *head*, se está indicando que todo el código JavaScript debe ser descargado, analizado e interpretado antes de

que la página comience a mostrarse. Esto puede suponer un tiempo adicional de descarga largo, razón por lo cual se pueden meter en el *body*. Se ejecutan cuando se termine de analizar la página si tienen el atributo *defer* (además, con este atributo nos aseguramos de que no cambiarán la página). Si no, justo cuando se encuentre.

- Los comentarios se hacen mediante /* … */ ó //.
- Operadores aritméticos: +,-,*, / y %. Precedencia igual que en otros lenguajes. Se emplean los paréntesis para establecer un orden de evaluación requerido.
- Cadenas de caracteres entre comillas dobles o simples (*"Hola. Esto es una cadena de caracteres." y 'esto, también'*).
- El operador de concatenación de cadenas de caracteres es + (*"hola" + "adiós"*).
- Las variables booleanas almacenan *true* y *false*.
- El carácter '\' se emplea como indicador de caracteres con significado.
- '*n*' y '*t*' para introducir una nueva línea y un tabulador, respectivamente.
- El operador *typeof* devuelve una cadena con el tipo de dato del operando (*typeof 4.5* devolvería "*number*". Se puede utilizar también como una función let *resultado=typeof("hola"); (*devuelve *"string"*).
- Los valores numéricos se pueden comparar mediante los operadores relacionales '>=', '<=', '<', '>', '!=' y '=='. Estos operadores son de aplicación a las cadenas de caracteres y devuelven los valores *true* o *false*.
- Los operadores lógicos son: '&&', '||' y '!', y devuelven los valores *true* o *false*.
- false será equivalente a una cadena vacía (" "), 0 ó *NaN*, *null* y *undefined*. n*ull* y *undefined* se también se emplean para indicar que las variables están vacías. La primera, es un valor especial que indica nada, vacío o valor desconocido; la segunda, que el valor no ha sido asignado.
- Todas las sentencias de JavaScript finalizan con un ';' o no… cuando hay una nueva línea para finalizar la sentencia se puede dejar sin ;.
- Para declarar variables se emplea la palabra reservada *var* (var altura=5.5;). Las variables se puden nombrar incluyendo los caracteres '$' y '_'. No se puede emplear ninguna variable que no haya sido declara previamente, aunque sea en el momento de asignarle un valor.
- También se puede usar *let* para hacer la asignación (let altura=5.5;). La diferencia entre let y var es la siguiente: la primera limita el alcance de la variable declarada al bloque de ejecución, expresión o declaración en la que se encuentre; la segunda, al ámbito de la función o al global (si no se encuentra en ningún bloque). Actualmente se emplea *let* en lugar de var, que se considera anticuado.

```
{
        let variable_let = 'valor variable_let';
        var variable_var = 'valor variable_var';
        console.log('var en el bloque {}: ' + variable_var);
        console.log('let en el bloque {}: ' + variable_let);
}
console.log('var fuera de bloque: ' + variable_var);
console.log('let fuera de bloque: ' + variable_let);
```

```
let var1 = 5;
if (var1 > 1) {
        let var2 = 10;
        var var3 = 15;
        console.log(var1 + var2 + var3); // Se escribe 30
        }
console.log(var1 + var2 + var3);  // Daría error pues var2 no existe aquí.
console.log(var1 + var3); // Se escribe 20.
```

- Se puede declarar una variable pero no asignarle ningún valor inicialmente. En este caso, el tipo de la misma es *undefined: let mensaje; resultado=typeof(mensaje); / / resultado toma undefined.*
- Para declarar una constante se usa la palabra reservada *const.*
- Cuando se empieza a ejecutar un programa JavaScript, se crea un entorno de variables, junto con sus valores correspondientes. Este entorno está inicializado con un conjunto de variables estándares que el navegador cargará y asignará valores. Las variables del entorno existirán hasta que el navegador cargue otra página.
- Las funciones en JavaScript se invocan igual que en otros lenguajes. Pueden o no devolver un valor.
- Ventanas de mensajes o diálogos incluidos en el entorno estándar:
  - *alert("Texto");* → muestra una ventana con el texto en su interior. También puede mostrar números.
  - *alert(confirm("Iremos?"));* → Muestra el texto así como dos botones (OK/Cancel) y devuelve *true* o *false* según se haya hecho click en OK o en Cancel, respectivamente.
  - *alert(prompt("Dime tu película favorita:","..."));* → Muestra el texto pasado como primer argumento y un campo de texto para meter un texto de entrada. El segundo argumento es el texto que aparecerá por defecto en el campo de texto.
- Para escribir texto en la propia página web podemos usar la fución *document.write('texto'),* o si simplemente queremos tener un control sobre las salidas de ciertas funciones podemos usar la consola de javascript mediante la función *console.log('texto')*. Para poder ver dichos valores tendremos que acceder a dicha consola (en el navegador Chrome: menú->herramientas->Consola de JavaScript).
- Las estructuras repetitivas de JavaScript son los bucles *while* y *for*:

```
let contador = 0;
while (contador <= 12)
        contador = contador + 2;
for (let contador = 0; contador <= 12; contador = contador + 2)
        document.write(contador);
```

Para los cuerpos de bucles de más de una línea, se emplean las llaves para delimitar su ámbito. Se emplea la sentencia *break* para romper la ejecución de bucle, continuando después de ella.

- Existen los operadores ++, - -, +=,...
- Se emplean las palabras *break* y *continue* con el mismo significado que en otros lenguajes de programación.

- Las estructuras condicionales de JavaScript son los *if* y *switch*:

```
for (let cont = 0; cont < 20; cont++) {
        if (cont > 15)
            document.write(cont + "**");
        else if (cont > 10)
            document.write(cont + "*");
        else  document.write(cont);
}
let dia=new Date().getDay();

switch (dia)
{
case 0: x="Domingo";
        break;
case 1: x="Lunes";
        break;
case 2: x="Martes";
        break;
case 3: x="Miércoles";
        break;
case 4:x="Jueves";
        break;
case 5:x="Viernes";
        break;
case 6:x="Sábado";
        break;
}
```

Funciones

- Las funciones se declaran mediante la palabra reservada *function*.
- Pueden devolver o no valores. En caso de hacerlo, se emplea la palabra return, y si no, realmente devuelven un valor "undefined".

```
function add(a, b) {
        return a + b;
}
 document.write(add(2, 2));
function mostrarMensaje(mensaje) {
  alert("¡¡"+ mensaje + "!!");
}
```

```
mostrarMensaje("OOOOleeeeee");
```

- Los nombres de funciones pueden asignarse a variables y luego emplearlas:

```
function mostrarMensaje(mensaje) {
        let funcion=alert;
         funcion("¡¡"+ mensaje + "!!");
}
mostrarMensaje("OOOOleeeeee");
```

- Las variables declaradas dentro de una función son locales a la función, pero las que se declaran fuera pueden ser empleadas dentro de las funciones.
- Los parámetros se pasan por copia.
- Se puden declarar parámetros con valores por defecto:

```
function showMessage(from, text = “defecto”) {...}
```

- Se pueden declarar funciones dentro de otras, en cuyo caso, el entorno local de las primeras contendrá el de las segundas:

```
let variable = "fuera";
function funcionPadre() {
        let variable = "local";
        function funcionHija() {
                document.write(variable);
        }
        funcionHija();
}
funcionPadre();
```

- OJO: sólo se crean entornos locales a partir del cuerpo de las funciones. Se pueden crear bloques de código en cualquier momento pero no se crean entornos de variables.
- Se pueden devolver funciones internas y luego invocarlas:

```
let variable = "fuera";
function funcionPadre() {
        let variable = "local";
        function funcionHija() {
           document.write(variable);
        }
        return funcionHija;
       }

let hija = funcionPadre();
```

```
hija();


function funcionPara Sumar(cantidad) {
        function sumar(numero) {
                return numero + cantidad;
        }
 return sumar;
}
let aniadirDos = funcionParaSumar(2);
let aniadirCinco = funcionParaSumar(5);
 document.write(aniadirDos(1) + aniadirCinco(1));
```

- No sólo se devuelve el nombre de la función, sino que también implícitamente se devuelve el entorno asociado.
- Se permite recursividad.
- Existe el array *arguments* que almacena todos los argumentos pasados a una función. Sólo se puede emplear dentro de ella (el primer argumento está en la posición 0). Con *arguments.length* podremos saber cuántos se han pasado a la función.
- No se pueden sobrecargar funciones.
- Se pueden crear valores función al estilo a como se crean e inicializan variables:

```
let suma = function(a, b) {
        return a + b;
};  /* OJO con este ; pues es como si hiciéramos suma=3; */
 document.write(suma(5, 5));
function funcionParaSumar(cantidad) {
        return function (numero) {
                return cantidad + numero;
        };
}
let suma= funcionParaSumar(5);
 document.write(suma(3));
```

- Funciones tipo *arrow*:

```
 let suma = (a, b) => a + b;
/* O lo que es lo mismo: */
let sum = function(a, b) {
 return a + b;
};
```

Objetos

- En JavaScript todo son objetos (*Object*, de forma genérica; una cadena de caracteres, *String*; un número, *Number*; un vector, *Array*; una función, *Function*).
- Los objetos son tipos de datos especiales que tienen propiedades y métodos.
- En JavaScript no existe el concepto de clase. No se crean clases explícitamente.
- Las propiedades se acceden mediante un punto, *cadena.length*, o alternativamente mediante la sintaxis de arrays, *cadena["length"];*
- Los métodos, con la sintaxis clásica: *cadena.toUpperCase();*
- La creación de objetos se puede hacer de dos formas diferentes:

```
libro= new Object();
libro.autor="Cervantes";
libro.titulo="Don Quijote de la Mancha";
libro.anio=1615;
libro.isbn="343.4335.234";
/* ó */
libro={autor:"Cervantes", titulo:"Don Quijote de la Mancha", anio:1615, isbn:"343.4335.234"};
```

- Los constructores son funciones que se denominan igual que el objeto:

```
function Libro(autor,titulo,anio,isbn)
{
        this.autor=autor;
        this.titulo=titulo;
        this.anio=anio;
        this.isbn=isbn;
}
let quijote= new Libro("Cervantes", "Don Quijote de la Mañcha", 1615, " 343.4335.234");
```

- Se pueden añadir propiedades en cualquier momento. Simplemente asignándole un valor:

```
quijote.editorial="Salvat";
```

- Se puede emplear *this* para acceder a un dato miembro de un objeto.
- Los métodos se asignan a un objeto añadiéndolos dentro del constructor correspondiente:

```
function Libro(autor,titulo,anio,isbn)
{
        this.autor=autor;
        this.titulo=titulo;
        this.anio=anio;
        this.isbn=isbn;
        function cambiarTitulo(titulo)
```

```
        { this.titulo=titulo; }
}
```

- Se pueden crear propiedades que no tengan un nombre como tal, sino, por ejemplo, un objeto cadena de caracteres:

```
let mensaje={"De": "Juanma", "Para": "Jose", "Asunto":"Asignatura", "Cuerpo":"Hola, Jose...."};
```

- Para acceder a los valores que almacena, el objeto se indexa como un vector por dicha propiedad:

```
mensaje["De"]="Pepe";
```

- Existe un operador para comprobar si una propiedad existe en un objeto. Este es *in*, el cual devuelve *true* si está y *false* en caso contrario:

```
let resultado= "De" in mensaje;
 document.write(resultado);
```

- Para borrar una propiedad se emplea *delete*: *delete mensaje["De"];*
- Se puede comprobar si existe una propiedad comparándola con *undefined*:

```
let resultado= {}
let existe = (resultado.propiedad === undefined); // Devuelve falso.
```

- Para obtener los valores de los atributos miembro de un objeto se emplea el siguiente bucle:

```
let profesor={nombre:"Juanma",apellidos:"Fernández"};
for (x in profesor)
        { datos_profesor=datos_profesor + profesor[x]; }
```

- Cuando disponemos de una referencia a un objeto, se puede indicar que no apunte a ningún objeto asignándole *null*.
- Para conocer si un objeto es instancia de un constructor se emplea: let *res= quijote instanceof libro;* Otro ejemplo: let *res2= mivector instanceof Array;*
- Los métodos son propiedades que almacenan una función:

```
Libro.prototype.mostrarAutor = function() {
  alert(this.autor);
}
```

- Los datos miembro pueden ser *configurable*, si puede borrarse mediante *delete*; *enumerable*, si puede ser devuelta por un bucle for-in; *writable*, si puede ser cambiada, y *value*, que contiene el valor real de la propiedad. Por defecto, al añadir una propiedad a un objeto, ésta es configurable, enumerable y writable. Para cambiar y asignar una de estas propiedades, se emplea el método *defineProperty*:

```
let persona = {};
Object.defineProperty(persona, "nombre", {
        writable: false,
        configurable: false,
        value: "Juanma"
});
```

- Una vez que una propiedad se ha asignado como no configurable, no se puede cambiar. Cualquier intento, da lugar a una excepción.
- Existen también "Accessor properties" (*configurable, enumerable, get* y *set*). Las dos últimas son funciones para obtener y devolver el valor de una propiedad.

```
let libro = {
	_anio: 2004,
	edicion: 1
};
Object.defineProperty(libro, "anio", {
	get: function(){
		return this._anio;
	},
	set: function(valor){
		if (valor > 2004) {
			this._anio = valor;
			this.edicion += valor - 2004;
		}
	}
});
libro.anio = 2005; // OJO, no le estamos asignado directamente un valor a _anio.
alert(libro.edicion); //2
```

- Cuando se precede el nombre de una propiedad mediante "_" se está dando a entender que no será accedida directamente, sino implícitamente mediante *get* y *set*.
- Para definir varias propiedades a la vez:

```
let libro = {};
Object.defineProperties(libro, {
		_anio: {
			value: 2004
			},
		edicion: {
			value: 1
			},
		anio: {
			get: function(){
			return this._anio;
			},
			set: function(valor){
```

```
                                        if (valor > 2004) {
                                        this._anio = valor;
                                        this.edicion += valor - 2004;
                                        }
                                }
                        }
});
```

- Prototipos y herencia: En JavaScript los objetos tienen una propiedad oculta e interna denominada property. Cuando queremos acceder a una propiedad de un objeto que no existe, la toma de la propiedad property.

```
let animal = {
            come: true
            anda()
            {alert("Animal anda");}
            };
let conejo = {
            salta: true;
            };
conejo.__proto__ = animal;

alert(conejo.come);
alert(conejo.salta);
// ----------------------
  let animal= {
            come: true
            anda()
            {alert("Animal anda");}
            };
let conejo = {
            salta: true;
            __proto__: animal;
            };
conejo.anda();
// --------------------------
  let animal= {
            come: true
            anda() {};
```

```
        };
    let conejo = {
        salta: true;
        __proto__: animal;
        };
    conejo.anda()= function() {
    alert("El conejito camina para adelante, el conejito camina para atrás");
```

Clases (A partir de ES6)

A partir de la versión 6 de ECMAScrpit, se incluye el concepto de clase en JavaScript, facilitando mucho más todo la programación orientada a objetos de JavaScript.

- La creación de clases se hace de la siguiente manera:

```
class Usuario {
    /* Constructor */
     constructor(nombre) {
            this.nombre = nombre;
    }
    /* Métodos */
    mostrarNombre() {
        alert(this.nombre);
        }
}
```

- Y su uso:

```
let usuario = new Usuario("Juanma");
usuario.mostrarUsuario();
```

- En JavaScript una clase es una función que toma el código del constructor. Los métodos se almacenan en el prototipo.
- Se pueden crear getter y setter en la clase:

```
class Usuario {
    id = "1234"; // No se comparte por los objetos creados a partir de la clase.
     constructor(nombre) {
        this.nombre = nombre;
    }
    get nombre() {
        return this.nombre;
      }
```

```
        set nombre(valor) {
                if (valor.length < 4) {
                        alert("Nombre demasiado corto);
                 return;
                 }
           this._name = value;
         }
        }

        let usuario = new User("Juanma");
        alert(usuario.nombre);
        usuario = new User("");
```

En el enlace https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Classes podéis encontrar un tutorial sobre clases, objetos y herencia en ES8.

Arrays

(http://www.w3schools.com/js/js_obj_array.asp)

- Los arrays se inicializan mediante corchetes: let *animales= ["perro", "gato", "león"].*
- Se asignan valores mediante un índice: *animales[4]="tigre"*. El acceso es el típico: *animales[1];*
- El primer índice del vector es 0.
- Se pueden crear arrays vacíos: let *animales[];*
- Realmente un vector es un objeto de la clase *Array*, por lo que se pueden aplicar métodos a dichos objetos:
    - Construcción: let *coches= new Array(); ó let coches= new Array("Seat", "Audi", "Volvo");*
    - Para incluir nuevos elementos se emplea el método *push*: *animales.push("elefante");* Se añade al final de array.
    - El número de elementos del array se obtiene mediante la propiedad *length* (*animales.length*).
    - El índice de un valor se obtiene por medio de *indexOf*: let *pos= coches.indexOf("Audi");*
    - Para concatenar Arrays se emplea el método *concat*.
    - Para obtener el último elemento se usa el método *pop*.
    - Otros métodos: *join, reverse, shift, slice, sort, splice, toString, unshift,...*
- Se pueden almacenar diferentes tipos de datos en un array, ya que todos los componentes de un Array se consideran objetos.
- Se pueden añadir elementos al vector en cualquier posición, aunque no esté lleno hasta ella.

Cadenas de caracteres

(http://www.w3schools.com/jsref/jsref_obj_string.asp.)

- Son objetos.
- Creación: let *nombre="Juan Manuel"; ó let nombre='Juan';* No hay diferencia entre "X" y 'X'.
- Se puede acceder a una posición concreta (los índices comienzan en 0): let *car=nombre[3];*
- Se emplea el carácter '\' para escapar otros caracteres.
- Longitud de la cadena: *nombre.length;*
- Encontrar la posición donde comienza una subcadena: *let pos=nombre.indexOf("Manuel");* Devuelve -1 si no está la subcadena.
- Sustitución de subcadenas: let *cad="Juan Manuel"; pos= cad.replace("Juan", "José");*
- Conversión a mayúsculas y minúsculas: *cad2=cad.toUpperCase(); y cad3=cad.toLowerCase();*
- Conversión de una cadena a un vector: *let str="a,b,c,d,e,f"; let n=str.split(",");* (Separador).
- Más métodos en http://www.w3schools.com/js/js_obj_string.asp.

Otros objetos

- Date (http://www.w3schools.com/js/js_obj_date.asp).
- Math (http://www.w3schools.com/js/js_obj_math.asp).
- RegExp (http://www.w3schools.com/js/js_obj_regexp.asp).

Ejercicios (utiliza *document.write('texto'+valor)* para escribir en la página web:

1) Escribe una función mayorQue, que reciba como argumento un número y devuelva una función que represente una prueba lógica. Cuando se llame a esta función devuelta con un único argumento, devolverá verdadero si el número dado es mayor que el número empleado para crear la función de la prueba lógica y falso, en caso contrario.

2) Crea un vector de 10 alturas de persona e informa de cuántos tienen una altura mayor o igual que 1.80m y cuántos menor.

3) Escribe un guión que, dados dos arrays de números enteros, diga cuál tiene el mayor valor acumulado. Escribe una función que haga dicha suma de valores.

4) Crea una función media y completa el ejercicio anterior con la media de cada array.

5) Define la clase Vehiculo (marca, modelo, anio_de_matriculacion, anio_de_venta, potencia, cilindrada, combustible), para representar vehiculos de segunda mano, crea propiedades de acceso, crea un array de vehiculos y escribe funciones para obtener el vehículo más antiguo, determinar cuántos están en un intervalo de años pasados como argumentos y enumerarlos.

Bibliografía

- http://www.w3schools.com/js/default.asp
- http://javascript.info
- https://medium.freecodecamp.org/the-complete-javascript-handbook-f26b2c71719c
- https://frontendmasters.com/books/javascript-enlightenment/