DESARROLLO WEB EN ENTORNO CLIENTE

UT4.- PROGRAMACIÓN CON ARRAYS, FUNCIONES Y OBJETOS DEFINIDOS POR EL USUARIO

CURSO 18/19

Profesora: Teresa Jiménez Nieto

Objetivos

- Conocer en detalle las principales funciones del lenguaje JavaScript.
- Crear funciones personalizadas para realizar tareas específicas.
- Comprender el objeto Array y familiarizarse con sus propiedades y métodos.
- Crear objetos personalizados.
- Definir propiedades y métodos de los objetos personalizados.

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

- ***eval(cadena[,objeto]):*** Convierte una cadena que pasamos como argumento en código JavaScript ejecutable.
  - Cadena: repsesenta una expresión, sentencia o secuencia de sentencias en JavaScript. La expresión puede incluir variables y propiedades de objetos existentes.
  - Objeto: si se especifica, la evaluación se restringe al contexto del objeto especificado.

Ejemplos:

```
1. eval(new String("2 + 2")); // devuelve un objeto String que contiene "2 + 2"
2. eval("2 + 2"); // devuelve 4
3. var expresion = new String("2 + 2");
   eval(expresion.toString());
```

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

- ***parseInt(cadena, base):*** convierte el string pasado como argumento en un valor numérico de tipo entero.
  - Si se encuentra con un carácter que no se corresponde con la base especificada devuelve NaN.
  - Si no se especifica la base o se especifica como 0, JavaScript asume lo siguiente:
    - Si el parámetro cadena comienza por "0x", la base es 16 (hexadecimal).
    - Si el parámetro cadena comienza por "0", la base es 8 (octal) → desaconsejado.
    - Si el parámetro cadena comienza por cualquier otro valor, la base es 10 (decimal).

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

- Ejemplos parseInt:

parseInt(10)	10
parseInt(10.33)	10
parseInt("34 45 66")	34
parseInt(" 60 ")	60
parseInt("40 años")	40
parseInt("tenía 40 años")	NaN
parseInt("12", 8)	10
parseInt("010")	8
parseInt("0x10")	16
parseInt("F", 16)	15

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

- ***parseFloat(cadena):*** convierte la cadena que pasamos como argumento en un valor numérico de tipo flotante.
  - Si el primer carácter no se puede convertir a número, devuelve NaN.

Ejemplos:

```
1. parseFloat("3.14"); //Devuelve 3,14
2. parseFloat("314e-2"); //Devuelve 3,14
3. parseFloat("0.0314E+2"); //Devuelve 3,14
4. v var cadena = "3.14"; parseFloat(cadena); //Devuelve 3,14
5. parseFloat("3.14más caracteres no dígitos"); //Devuelve 3,14
6. parseFloat("FF2"); //Devuelve NaN
```

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

- ***isNaN(valor):*** Is Not a Number. Intenta convertir el argumento a número; si no puede, devuelve true; en caso contrario devuelve false.

Ejemplos:

```
1. isNaN(NaN) //devuelve true
2. isNaN("string") //devuelve true
3. isNaN("12") //devuelve false
4. isNaN(12) //devuelve false
```

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

- ***isFinite(numero):*** si el argumento es NaN, infinito positivo o negativo devuelve false; en caso contrario devuelve true.
- Ejemplos:
  - isFinite(Infinity); //falso
  - isFinite(NaN); //falso
  - isFinite(-Infinity); //falso
  - isFinite(0); //verdadero
  - isFinite(2e64); //verdadero
  - isFinite("0"); //verdadero, hubiera sido falso en el caso de usar Number.isFinite("0") que es más robusta.

1. FUNCIONES PREDEFINIDAS DEL LENGUAJE

- ***Number(objeto)*** / ***String(objeto)*:** convierten el objeto pasado como parámetro a Number o String.

Ejemplo:

```
var hoy = new Date();
document.write(String(hoy));
```

2. FUNCIONES DEFINIDAS POR EL USUARIO

- Bloques de código usados para ejecutar una tarea en particular.
- Conviene definirlas en el head y luego llamarlas desde cualquier punto de la web con un script.
- **Hay que invocarla** para que se ejecute:
  - Desde el código
  - Cuando ocurre un evento
  - Automáticamente

```
function nombre (<argumentos>) {
  <instrucciones>
  <return dato>
}
```

2. FUNCIONES DEFINIDAS POR EL USUARIO

- Los **parámetros primitivos** (como puede ser un número) son **pasados** a las funciones **por valor**.
- El valor es pasado a la función, pero si la función cambia el valor del parámetro, *este cambio no es reflejado globalmente* o en otra llamada a la función.
- Si pasa un **objeto** (p. ej. un valor no primitivo, como un Array o un objeto definido por el usuario) como parámetro, éste es **pasado por referencia**
- Si la función cambia las propiedades del objeto, este cambio es visible desde fuera de la función

2. FUNCIONES DEFINIDAS POR EL USUARIO
2.1. Funciones anónimas

- **No tienen nombres.**
- La asigno a una variable y la invoco con el nombre de la variable y los parámetros entre paréntesis.

```
var producto = function (a, b) { return a * b;};
var resultado = producto(3,6);
alert (typeof producto); //devueve function
alert (resultado);
```

- Muy usadas al trabajar con eventos.

2. FUNCIONES DEFINIDAS POR EL USUARIO
2.1. Funciones anónimas

- Otra manera de crear una función anónima es mediante el **constructor Function**.

```
var miFuncion = new Function ("a", "b", "return a*b;");
var resultado2 = miFuncion(5,7);
alert (resultado2);
```

- Podemos hacer que se **autoinvoque** una función:

```
//FUNCIONES ANÓNIMAS AUTOINVOCADAS
(function () { alert ("¡Hola!");}());
```

2. FUNCIONES DEFINIDAS POR EL USUARIO
2.2. Parámetros y argumentos

- **Parámetros:** son los nombres que aparecen en la definición de una función. Una función puede tener 0 o más parámetros.
- **Argumentos:** son los valores que pasamos a (y que recibe) una función

REGLAS DE LOS PARÁMETROS:

- No se especifica el tipo de los parámetros.
- No se verifican los tipos de los argumentos.
- No se comprueba el número de los argumentos recibidos.

2. FUNCIONES DEFINIDAS POR EL USUARIO
2.2. Parámetros y argumentos

PARÁMETROS POR DEFECTO

- Cuando llamamos a una función con menos argumentos de los declarados. Los valores que faltan no están definidos.

```
function suma (a, b){
    if (b === undefined)
        b = 0;
    return a + b;
}
var resultado = suma (4);
alert (resultado);
```

2. FUNCIONES DEFINIDAS POR EL USUARIO
2.2. Parámetros y argumentos

PARÁMETROS POR EXCESO

- Si llamamos a una función con más argumentos de los que ha sido declarada. Los valores que nos llegan pueden capturarse a través de un objeto (incluido en la función) llamado arguments.

```
function valores (){
    alert ("El número de argumentos es
    "+arguments.length);
    for (var i=0; i < arguments.length; i++){
        alert ("Argumento "+i+"="+arguments[i]);
    }
}
valores (4, 6, 8, 2, 7, 5);
```

2. FUNCIONES DEFINIDAS POR EL USUARIO
2.3. Objeto arguments

- Los argumentos de una función son mantenidos en un objeto similar a un array.
- Dentro de una función, los argumentos pasados a la misma pueden ser direccionados de la siguiente forma:

arguments[i]

  - donde i es el número ordinal del argumento, comenzando desde cero.
  - el primer argumento pasado a una función sería ***arguments[0]***.
  - el número total de argumentos es mostrado por ***arguments.length***.

3. ARRAYS

- El objeto Array es un objeto nativo de JavaScript.
- Un array es una colección de valores homogénea o no y ordenada.
- Características:
  - Almacena en una misma variable múltiples valores: valores primitivos, objetos, etc.
  - Se referencian con un índice numérico.
  - Puede almacenar diferentes tipos de datos.

3. ARRAYS
3.1. Operaciones básicas con un array

- **Crear un array:**
  - Manera recomendada:
    - var nombreArray = [<valores separados por comas>];

```
var frutas = ["Manzana", "Platano"];
console.log(frutas.length); // 2
```

  - Usando el constructor del objeto Array (puede darnos errores en alguno navegadores:
    - var nombreArray = new Array(<valores separados por comas);

```
miArray = new Array("Hola",30, 3.14); //Array de elemenetos heterogéneos
miArray2 = new Array("Viento", "Lluvia", "Fuego", "Tierra"); //Array de elementos homogéneos
miArray3 = new Array(5); //Array vacío con 5 elementos
```

3. ARRAYS
3.1. Operaciones básicas con un array

- **Acceso a elementos de un array:**
  - nombreArray[<indice>]: el primer elemento es 0.

```
var primero = frutas[0]; // Manzana
var ultimo = frutas[frutas.length - 1]; // Platano
```

- Mostrar todo el array con alert, document.write, etc
  - alert(nombreArray) (recomendado con nombreArray.toString)
- Con un bucle for y la propiedad lenght podemos mostrar los valores de un array uno a uno

3. ARRAYS
3.2. Propiedades

Propiedad	Descripción
length	Devuelve la longitud del array
prototype	Permite agregar al objeto Array las propiedades que queramos.

Ejemplo de prototype:

```
Array.protoype.descriptor = null;
dias = new Array ('lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes');
dias.descriptor = "Dias laborables de la semana";
```

3. ARRAYS
3.3. Métodos

- ***Array.isArray(<nombreArray>):*** devuelve true si es un objeto de tipo Array. Si ponemos typeof(nombreArray) devuelve object.
- ***<nombreArray> instanceof Array***: devuelve true si es un Array.

3. ARRAYS
3.3. Métodos

- **Métodos para mostrar un array:**
  - ***toString():*** convierte el array a cadena.
  - ***join(<separador>):*** convierte el array a cadena separado por un separador indicado por parámetro. Ej. documet.write(frutas.join(" * "));

Unir los elementos de un array en una cadena:

```
var frutas=["Manzana","Platano","Kiwi"];
var cadena = frutas.join(":");
        // cadena = "Manzana:Platano:Kiwi";
```

3. ARRAYS
3.3. Métodos

- **Métodos para añadir, extraer o borrar elementos**:
  - ***pop()*:** extrae (y elimina) el último elemento.
  - ***shift()*:** extrae (y elimina) el primer elemento.
  - ***delete*** *nombreArray[<indice>]* : elimina el elemento y lo transforma a undefined.
  - ***push(<elemento>)*:** añade un elemento al final del array.

Añadir al final del array:

```
var nuevoTamano = frutas.push("Naranja");
        // ["Manzana", "Platano", "Naranja"] y devuelve el tamaño del array
```

Eliminar del final del array:

```
var ultimo = frutas.pop(); // borra Naranja (del final) y lo devuelve
        // ["Manzana", "Platano"];
```

3. ARRAYS
3.3. Métodos

- ***unshift((<elemento>):*** añade un elemento al principio del array.

Añadir al principio del array:

```
var nuevoTamano = frutas.unshift("Fresa") // añade al principio y devuelve el
tamaño
        // ["Fresa", “Manzana”, "Platano"];
```

Eliminar del principio del array:

```
var primero = frutas.shift(); // borra Fresa(del principio) y lo devuelve
        // [“Manzana”, "Platano"];
```

3. ARRAYS
3.3. Métodos

- Podemos añadir elementos con:
  - nombreArray[<indice>] = <elemento>. Cuidado con sobreescribir o dejar huecos (los rellena con undefined).
- ***concat(<lista de arrays separados por comas>):*** une el array inicial con un segundo array…

Concatenar dos arrays:

```
var frutas1 = ["Manzana", "Platano"];
var frutas2 = ["Ciruela"];
frutas1.concat["Kiwi","Higo","Pera"];
        //frutas1=["Manzana","Platano","Kiwi","Higo","Pera"];
frutas1.concat(frutas2);
        //frutas1=["Manzana","Platano","Kiwi","Higo","Pera","Ciruela"];
```

3. ARRAYS
3.3. Métodos

- ***slice(<posición inicio>[, <fin>]):*** devuelve un subarray desde la posición indicada hasta (sin incluir) la final.
- ***splice(<posición inicio>, nº elementos a borrar[, <elementos a añadir separados por comas>]):*** elimina el número de elementos de un array empezando por la posición inicio. Si tiene el tercer parámetro, además los sustituye por él.

Tomar elementos según su posición (inicial y final):

```
var frutas=["Manzana","Platano","Kiwi", "Pera","Higo"];
var seleccion = frutas.slice(0, 1); // seleccion = ["Manzana","Platano"]
var ultimos = frutas.slice(–2); // seleccion = ["Pera","Higo"]
```

Eliminar elementos según su posición (inicio, número de elementos):

```
var frutas=["Manzana","Platano","Kiwi", "Pera","Higo"];
var eliminado= frutas.splice(2, 2); // frutas = ["Manzana","Higo"]
var ultimos = frutas.splice(-1);// frutas = ["Manzana"]
```

- **Métodos para buscar:**
  - ***indexOf(<elemento>[,<pos>]):*** devuelve el primer índice que se encuentre del elemento. Podemos pasarle a partir de qué posición va a buscar.
  - ***lastIndexOf (<elemento>[,<pos>]):*** devuelve el último índice que se encuentre del elemento.

3. ARRAYS
3.3. Métodos

- **Métodos para ordenar e invertir el orden del array**:
  - ***reverse()*:** invierte el orden de un array.
  - ***sort()*:** ordena el array.
    - Si queremos ordenar array por números debe contener cifras, si son almacenados como cadenas se comparan carácter a carácter. Ej “35” y “340”, coge primero 340.

3. ARRAYS
3.3. Métodos

Ordena según el orden lexicográfico:

```
var frutas=["Manzana","Platano","Kiwi","Higo","Pera","Ciruela"];
var frutasOrdenado = frutas.sort ();
        //frutas=["Ciruela", "Higo", "Kiwi", "Manzana", "Pera", "Platano"]
        //Devuelve el array ordenado
```

Invierte el orden de los elementos del array:

```
var frutas=["Manzana","Platano","Kiwi","Higo","Pera","Ciruela"];
var frutasReves = frutas.reverse();
        //frutas=["Ciruela", "Pera", "Higo", "Kiwi", "Platano", "Manzana"
        //Devuelve el array ordenado del revés
```

- Consultar todos los métodos: https://www.w3schools.com/jsref/jsref_obj_array.asp

3. ARRAYS
3.4. Arrays multidimensionales

- Son arrays que almacenan en cada posición otros arrays a su vez. También podemos crearlos tridimensionales, etc.

Crear un array bidimensional mediante otros arrays:

```
var datos = new Array();
datos [0] = new Array ("Programacion", "1DAW", 10);
datos[1] = new Array ("DWEC", "2DAW", 9);
datos[2] = new Array ("DIW", "2DAW", 5);
```

Crear un array bidimensional usando corchetes:

```
var datos = [
            ["Programacion", "1DAW", 10],
            ["DWEC", "2DAW", 9],
            ["DIW", "2DAW", 5]
];
```

3. ARRAYS
3.4. Arrays multidimensionales

Acceder a una posición de un array multidimensional:

Tenemos que indicar tantas posiciones como dimensiones (índices) tenga el array:

```
var elemento = datos [0][2];
```

Imprimir datos de un array multidimensional:

Utilizamos tantos bucles for anidados como dimensiones (índices) tenga el array.

```
for (i=0; i<datos.length; i++){
   for (j=0; j<datos[i].length; j++){
      document.write("Elemento "+i+","+j+": "+datos[i][j]);
   }
}
```

4. OBJETOS

- En JavaScript casi todo son objetos
- Un objeto es una **colección de propiedades**.
- Una **propiedad** es una asociación entre un nombre y un **valor**.
- Un valor de propiedad puede ser una función, la cual es conocida entonces como un **método** del objeto.
- Además de los objetos que están predefinidos en el navegador, podemos definir nuestros propios objetos.

4. OBJETOS

4.1. Definición de objetos

- Podemos **definir y crear** nuestros propios objetos:
  1. **Utilizando un literal** (valor fijo formado por parejas de tipo nombre:valor) podemos crear un objeto simple.

     var persona1 = {nombre:"Pepito", apellido:"Pérez"};

  2. Utilizando la **palabra new** podemos crear un objeto simple:

     var persona2 = new Object();

     persona2.nombre ="Juanito";

  3. Definir un **constructor de un objeto y crear objetos del tipo construido.**

4. OBJETOS
4.1. Definición de objetos

- **Constructor**: función especial para crear un objeto. Empieza en ***mayúscula***:

```
function Coche()
{//Propiedades y métodos}
```

Creación de un objeto:

```
var unCoche = new Coche();
```

Coche()

```
var cocheazul = new Coche();
var cocherojo = new Coche();
var cocheverde = new Coche();
```

4. OBJETOS
4.1. Definición de objetos

- **Definir las propiedades del objeto**: se crean en el ***constructor*** precedidas por la ***palabra reservada this.***

"Constructor" sin parámetros:

```
function Coche()
{
  this.marca ="";  //Vacío
  this.modelo  = "";  //Vacío
  this.combustible  ="Diesel";
                  //Inicializado
  this.cantidad=0;  //Inicializado
}
```

Crear un objeto vacío (sin propiedades):

```
var cocheVacio  = new Coche();
```

Cambiar valores en las propiedades:

```
cocheVacio.marca  = "Seat";
cocheVacio.modelo  = "Ibiza";
cocheVacio.combustible  = "Diesel";
cocheVacio.cantidad  = 40;
```

"Constructor" con parámetros:

```
function Coche(marca, modelo, combustible, cantidad)
{
  this.marca =marca;
  this.modelo  = marca;
  this.combustible  = combustible;
  this.cantidad=cantidad;
  //Cada propiedad toma los valores recibidos por parámetro
}
```

Crear un objeto inicializado (con propiedades):

```
var miCoche = new Coche("Seat", "Ibiza", "Diesel", 40);
```

Acceder a las propiedades:

```
document.write("Mi  coche es un "+miCoche.marca+"
"+miCoche.modelo);
```

4. OBJETOS
4.2. Propiedades

- **Acceso a las propiedades de un objeto**:
  - nombreObjeto.nombrePropiedad;
  - nombreObjeto[“nombrePropiedad”];
  - nombreObjeto[expresión];

```
var persona1 = {nombre:"Pepito", apellido:"Pérez", edad:"30"};
alert (persona1.nombre);
alert (persona1["apellido"]);
var expresion = "edad";
alert (persona1[expresion]);
```

4. OBJETOS
4.2. Propiedades

- **Recorrer las propiedades de un objeto:**
  - for (<nombre_variable> in <nombre_objeto>){
    <nombre_objeto>[<nombre_variable>];
    }

```
for (x in persona1){
    alert(persona1[x]);
}
```

4. OBJETOS
4.2. Propiedades

- **Añadir una nueva propiedad a un objeto**:
  - nombreObjeto.nuevaPropiedad = valor_propiedad;

```
persona1.pais = "Portugal";
alert(persona1.pais);
```

- **Borrar una propiedad de un objeto**:
  - delete nombreObjeto.nombrePropiedad;

```
delete persona1.edad;
```

4. OBJETOS
4.3. Métodos

- En JavaScript ***son propiedades*** que tienen dentro una función.
- Permiten modificar las propiedades de los objetos. Empiezan en minúscula.
- **Creación de métodos de un objeto**:

```
nombreMétodo : function() {
        //instrucciones
}
```

4. OBJETOS
4.3. Métodos

- **Acceso a los métodos de un objeto:**

  nombreObjeto.nombreMétodo();

```
var persona1 = {nombre:"Pepito",
                apellido:"Pérez",
                nombreCompleto: function(){
                    return this.nombre + " " + this.apellido;
                    }
                };
alert (persona1.nombre);
alert (persona1.apellido);
alert (persona1.nombreCompleto());
```

4. OBJETOS
4.4. Prototipos

- Todos los objetos tienen un prototipo que a su vez es un objeto.
- Un objeto tiene una **propiedad prototype** que contiene un objeto.

```
/* Sintaxis de creación de un prototipo de un objeto usando la función
constructor: */
function Persona (nombre, apellido, edad){
    this.nombre = nombre;
    this.apellido = apellido;
    this.ano = edad;
    this.nombreCompleto = function (){
        return this.nombre + " " + this.apellido;
    }
}
var pepito = new Persona ("Pepito", "Perez", "30");
var juanito = new Persona ("Juanito", "López", "50");
```

4. OBJETOS
4.4. Prototipos


```
/* Añadir una propiedad a un objeto */
pepito.pais = "Portugal"; //Solo se añade a pepito, no a juanito
```

```
/* Añadir un método a un objeto */
pepito.nacimiento = function(){
    return "Edad "+this.edad;
} //Solo se añade a pepito, no a juanito
```

- **Añadir una propiedad al prototipo**:
  - Añadirlo en el constructor (definición del prototipo).
  - Mediante la sintaxis: NombreObjeto.prototype.propiedad

```
Persona.prototype.telefono = "132456";
```

4. OBJETOS
4.4. Prototipos

- **Añadir un método al prototipo**:
  - Añadirlo en el constructor (definición del prototipo).
  - Mediante la sintaxis: NombreObjeto.prototype.metodo = …

```
Persona.prototype.telefono = function(){
    return "Teléfono de la persona: "+this.telefono;
}
```

5. HERENCIA

- Sintaxis: `ClaseHija.prototype = new ClasePadre();`

```
function Persona (nombre, apellido, edad){
    this.nombre = nombre;
    this.apellido = apellido;
    this.ano = edad;
    this.nombreCompleto = function (){
        return this.nombre + " " + this.apellido;
    }
}
function Alumno (matricula, curso){
    this.matricula = matricula;
    this.curso = curso;
}
Alumno.prototype = new Persona();
alumno1 = new Alumno();
alumno1.nombre = "Sofía";
alumno1.apellido = "Suárez"
alumno1.matricula = 12;
alert (alumno1.nombreCompleto());
```

5. HERENCIA
5.1. Herencia en ES6

- ECMAScript 2015 (ES6) introduce un nuevo set de palabras reservadas implementando clases.
- Aunque estos constructores se parecen más a Java, no son clases. **JavaScript permanece basados en prototipos.**
- Se introdujeron las nuevas palabras reservadas class, constructor, static, extends, y super.

5. HERENCIA
5.1. Herencia en ES6


```
class Poligono{
  constructor(altura,

        anchura){

    this.altura = altura;
    this.anchura = anchura;
  }
}
```

```
class Cuadrado extends Poligono{
  constructor(lado) {
    super(lado, lado);
  }
  get area() {
    return this.altura *
           this.anchura;
  }
  set fijarLado(nuevoLado) {
    this.altura = nuevoLado;
    this.anchura = nuevoLado;
  }
}

var cuadrado = new Cuadrado(2);
```