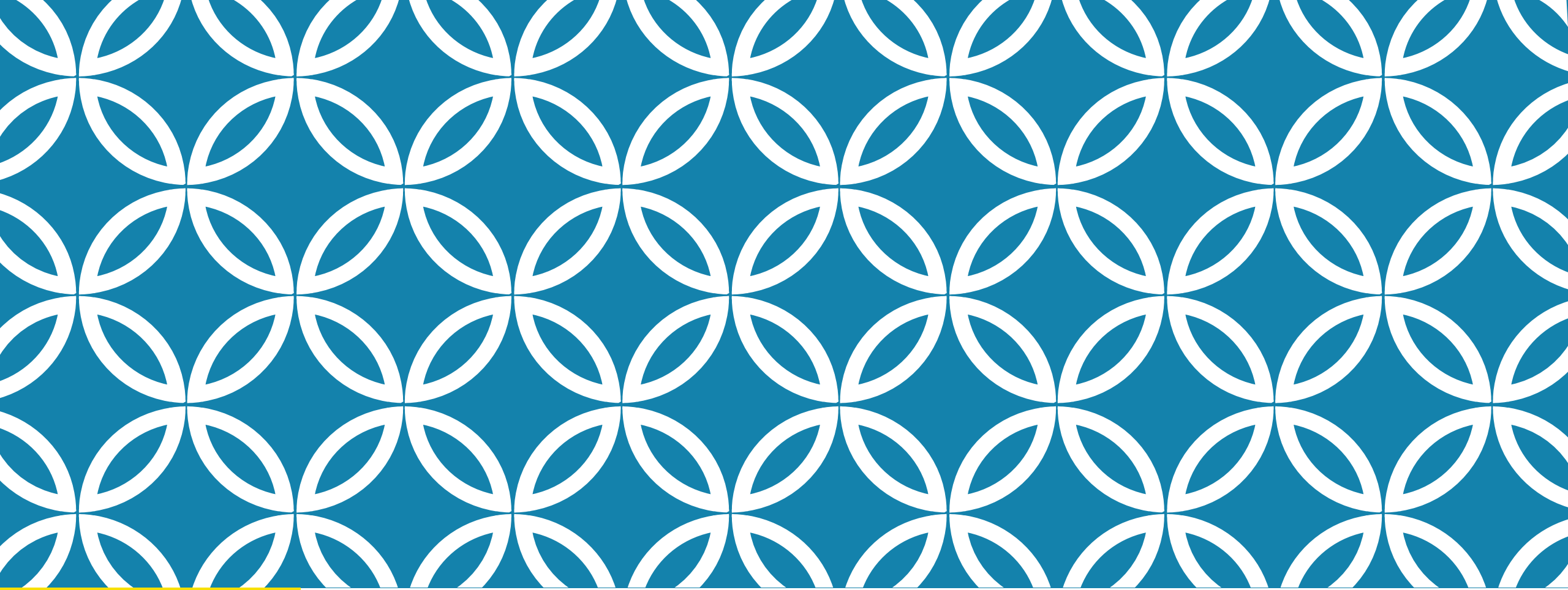
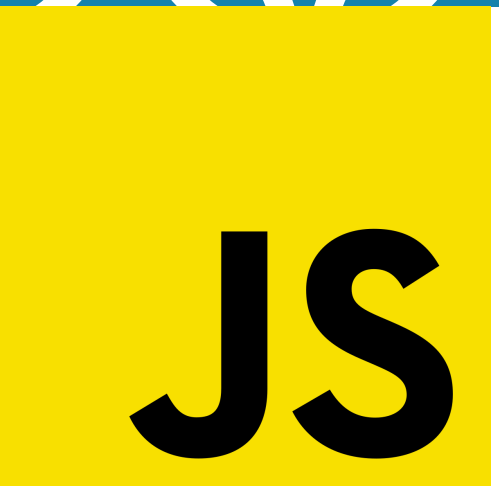




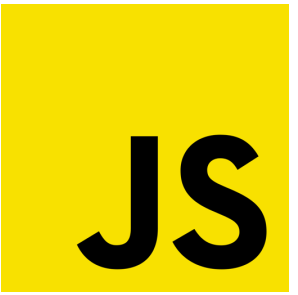
UT6. FUNDAMENTOS DE JS

Desarrollo de interfaces
2ºDAM

TABLA DE CONTENIDOS

1. El lenguaje JS
2. Sintaxis general
3. Tipos de datos
4. Sentencias de control
5. Objetos nativos de JS
6. Funciones predefinidas del lenguaje
7. Funciones de usuario
8. Arrays
9. Objetos definidos por el usuario

1. EL LENGUAJE JAVASCRIPT



JS es un lenguaje interpretado

Se define como

- Orientado a objetos
- Imperativo
- Basado en prototipos
- Débilmente tipado
- Dinámicamente tipado

Se usa principalmente en el lado del cliente, implementado como parte del navegador web para dar mejoras a la interfaz web y dar comportamiento dinámico.

VERSIONES DE JS

JS es un dialecto de ECMAScript y esta definido por dicho estándar.

- Creado por Brendan Eich para Netscape en 1995. En principio llamado LiveScript → JavaScript
- En 1997 ECMA crea el primer starndar ECMAScript
- **ECMAScript 5, ES5,** lanzado en 2009 fue una versión que introdujo muchas mejoras y fue un estándar durante muchos años.
- **ECMAScript 6,** También denominada **ES6,** de 2015. introdujo grandes mejores como el uso de clases y módulos. Es la versión más extendida soportada por todos los navegadores actuales excepto IE.
- A partir de dicha versión se ha ido liberando una versión por año. Por ejemplo la versión de 2020 es ECMAScript 11.
- Podemos comprobar en https://html5test.com/ qué versiones de ECMAScript soporta nuestro navegador.

2. SINTÁXIS GENERAL

ECMAScript tiene una sintaxis similar a C++ o Java.

Comentarios de una línea `// comment`

Comentarios multilínea `/* comment */`

El punto y coma al final de línea es opcional pero recomendable

No usar palabras reservadas para nombrar variables

El tipado de datos es débil. No hace falta declararlo.

Gramática léxica de JS:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Lexical_grammar

LA CONSOLA DE DEPURACIÓN

Podemos ver la consola en el navegador en la opción “inspeccionar”

Podemos interactuar con el objeto del sistema console invocando sus métodos

```
console.log('Hola mundo')              // mensaje general
console.info('Esto es un mensaje'); // mensage de información
console.warn('Cuidadito')             // mensaje de advertencia
console.error('Error fatal');      //mensaje de error
```

Con la orden debugger podemos parar la ejecución del script para depurar

2. DECLARACIÓN DE VARIABLES

Podemos declarar de tres formas nuestras variables

- var → forma tradicional. Se puede acceder desde cualquier punto del script.
- Let. → variable que solo es accesible desde el “scope” donde se declara (función o bloque de código)
- cons → constante que solo es accesible desde el “scope” donde se declara.

```
var num = 5; //variable tradicional. Accesible en todo el script
let cadena = "hola"; //variable accessible solo en su "scope"
const booleano = true;  //constante accesible en su "scope"
```

Se recomienda usar let o const dependiendo de si queremos que el valor varíe o no

3. TIPOS DE DATOS

Tipos de datos primitivos

- Number: Números
- String: Cadena de texto
- Boolean: true / false
- Undefined: no definido
- (*)Null: valor nulo

3.1 NUMBERS

Cualquier positivo o negativo

- Ej. 4, 5, 34, 0, -4, -30

Cualquier número con decimales o sin ellos

- Ej. 34.4 , 0,5 , -9,45 , 150000000

Operaciones aritméticas

- + - * /

Operaciones de comparación (Explicación igualdades)

- < > >= <= == === !=

3.1 OPERACIONES CON NUMBERS

typeof

NaN

isNan()

toString()

.toFixed()

3.2 STRINGS

Una cadena de texto entre comillas

- “Hello world”
- ‘Hello world’

Operación de concatenar

- cadena = ‘Hola’ + ‘mundo’

Operaciones de comparación de cadenas

- == , === , !=

3.2 OPERACIONES CON STRINGS

Concatenar texto

Template literals

typeof

.length

.includes()

.slice(start,end)

.replace('este texto','por este otro')

.trim()

.split(',')

3.2 EJERCICIO CON STRINGS

Te han pedido que incluyas en el texto de la próxima
newsletter unas variables dinámicas que vienen del usuario.
Nombre: Amparo
Interés: Breath of the wild
Descuento: 20% en tu próxima compra
Texto: Hola USUARIO,
Como regalo de bienvenida al Club de Gamers
queremos ofrecerte un descuendo del DESCUENTO en el próximo lanzamiento de tu interés JUEGO.
Esperamos que lo disfrutes,
UN saludo,

3.3 BOOLEAN

Solo admite dos valores: true o false

- Es útil para comprobar estados de la aplicación

Se aconseja que su nombre defina el estado positivo

- Ej: userIsLogged

3.3 BOOLEAN

Boolean(valor); nos devuelve el valor booleano de una condición o variable

True (Verdadero)	False (Falso)
1	0, -0
"Whatever"	""
3.14	NaN
100 > 5	null
1 < 100	undefined
'1' == 1	'1' === 1

3.4 UNDEFINED

Es un tipo de datos que indica **ausencia de valor**

Ej:

```
var a
console.log(a)
console.log(typeof a) // → undefined
```

3.5 NULL

Sí que tiene valor pero es null (un objeto que apunta a vacío)

Ej:

Var a = null

console.log (a)

Console.log (typeof a) // → objetc

4. SENTENCIAS DE CONTROL

Sentencias condicionales:

If… [else[if…]]

switch

Operador ternario condicional

4. SENTENCIAS DE CONTROL

Bucles o repeticiones:

for (tradicional bucle que itera con un contador)

for..in (recorre propiedades de un objeto)

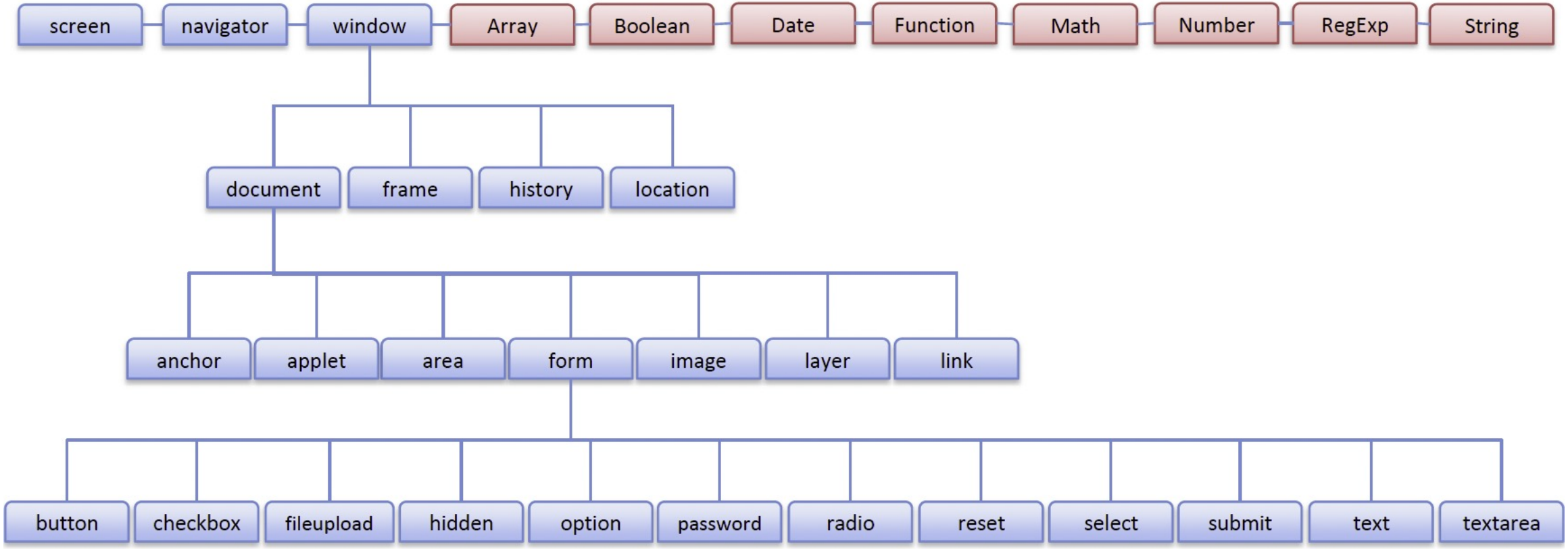
for..of (EC6) (recorre una colección de objetos)

While (tradicional bucle mientras);

Nota: Diferencias entre for..in y for..of

5. OBJETOS NATIVOS DE JAVASCRIPT

Los objetos nativos de JS se ordenan de forma jerárquica



5.1 EL OBJETO DATE

Permite trabajar con fechas y horas

Almacena el número de milisegundos desde las 00:00:00 UTC del 1 de enero de 1970

El constructor Date:

```
new Date()
new Date(milisegundos)
new Date(cadenaFecha)
new Date(año_num,mes_num,dia_num
        [,hor_num,min_num,seg_num,mils_num])
```

Si no proporciona argumentos, el constructor crea un objeto Date con la hora y fecha de hoy según la hora local.

Fuente: https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Date

5.1 EL OBJETO DATE.

Funciones del objeto global Date:

Métodos				
getDate()	getTime()	getUTCMonth()	setMonth()	setUTCMonth()
getDay()	getTimezoneOffset()	getUTCSeconds()	setSeconds()	setUTCSeconds()
getFullYear()	getUTCDate()	parse()	setTime()	toDateString()
getHours()	getUTCDay()	setDate()	setUTCDate()	toLocaleDateString()
getMilliseconds()	getUTCFullYear()	setFullYear()	setUTCFullYear()	toLocaleTimeString()
getMinutes()	getUTCHours()	setHours()	setUTCHours()	toLocaleString()
getMonth()	getUTCMilliseconds()	setMilliseconds()	setUTCMilliseconds()	toTimeString()
getSeconds()	getUTCMinutes()	setMinutes()	setUTCMinutes()	toUTCString()

5.2 EL OBJETO MATH

Permite realizar operaciones matemáticas complejas en JavaScript.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math

Métodos		
abs()	exp()	random()
acos()	floor()	round()
asin()	log()	sin()
atan()	max()	sqrt()
ceil()	min()	tan()
cos()	pow()	

Propiedades
E
LN2
LN10
LOG2E
LOG10E
PI
SQRT1_2
SQRT2

5.3 EL OBJETO NUMBER

Permite realizar operaciones con tipo de datos numérico.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Number

Métodos
toExponential()
toFixed()
toPrecision()

Propiedades
MAX_VALUE
MIN_VALUE
NaN
NEGATIVE_INFINITY
POSITIVE_INFINITY

5.4 EL OBJETO STRING

Permite manejar cadenas de texto

Métodos			
anchor()	fixed()	link()	strike()
big()	fontcolor()	match()	sub()
blink()	fontsize()	replace()	substr()
bold()	fromCharCode()	search()	substring()
charAt()	indexOf()	slice()	sup()
charCodeAt()	italics()	small()	toLowerCase()
concat()	lastIndexOf()	split()	toUpperCase()

Propiedades
Lenght

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String

6. FUNCIONES PREDEFINIDAS DEL LENGUAJE

JavaScript cuenta con una serie de funciones integradas en el lenguaje.

Dichas funciones se pueden utilizar sin conocer todas las instrucciones que ejecuta.

Simplemente se debe conocer el nombre de la función y el resultado que se obtiene al utilizarla.

6. FUNCIONES PREDEFINIDAS DEL LENGUAJE

encodeURI

encodeURIComponent

decodeURI

decodeURIComponent

Cómo utilizarlos: https://www.freecodecamp.org/espanol/news/url-codificacion-como-utilizar-encodeuricomponent-javascript/

6. FUNCIONES PREDEFINIDAS DEL LENGUAJE

eval() → convierte y evalúa una cadena pasada como si fuese código JS

isFinite() → Verifica si el numero que le pasamos es un numero finito.

Number.isNaN() → Comprueba si el valor pasado es un NaN. Es más robusto que isNaN()

String() → Devuelve un objeto pasado como string.

Number() → Devuelve el objeto pasado como number.

parseInt() → Convierte la cadena que le pasamos a un numérico entero.

parseFloat() → Convierte la cadena que le pasamos como un numérico en punto flotante.

7. FUNCIONES DEL USUARIO

Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

Es posible crear funciones personalizadas diferentes a las funciones predefinidas por el lenguaje.

Hay dos tipos de declaraciones de funciones

- La clásica mediante la palabra reservada function
- Mediante las arrow functions (desde ES6)

7.1 DECLARACIÓN DE FUNCIONES

Creación de funciones mediante declaración.

```
function nombreFuncion ( [parámetros )
{
     instrucciones
    [return valor]
}
```

8.1 DECLARACIÓN DE FUNCIONES

De esta manera la función existirá a todo lo largo del código. Podemos ejecutar “saludar” antes de declararla incluso.

```
function saludar() {
  return "Hola";
}

saludar(); // 'Hola'
typeof saludar; // 'function'
```

Fuente https://lenguajejs.com/javascript/fundamentos/funciones

7.1 DECLARACIÓN DE FUNCIONES

Declaración de funciones por expresión. Asignamos la función a una variable.

Aquí no se pude invocar la función antes de hacer la asignación

```
// El segundo "saludar" (nombre de la función) se suele omiti
const saludo = function saludar() {
  return "Hola";
};

saludo(); // 'Hola'
```

7.1 DECLARACIÓN DE FUNCIONES

Podemos incluso declarar una función a partir del objeto global Function.

```
const saludar = new Function("return 'Hola';");

saludar(); // 'Hola'
```

7.1 DECLARACIÓN DE FUNCIONES

Podemos declarar una **función anónima o "lambda"** para asignársela a una variable. El nombre de la función no haría falta es redundante.

```
// Función anónima "saludo"
const saludo = function () {
  return "Hola";
};

saludo; // ƒ () { return 'Hola'; }
saludo(); // 'Hola'
```

7.2 FUNCIONES CALLBACK

A grandes rasgos, un **callback** (*llamada hacia atrás*) es pasar una **función B por parámetro** a una **función A**, de modo que la función A puede ejecutar esa función B de forma genérica desde su código, y nosotros podemos definirlas desde fuera de dicha función

```
// fB = Función B
const fB = function () {
  console.log("Función B ejecutada.");
};

// fA = Función A
const fA = function (callback) {
  callback();
};

fA(fB);
```

Fuente: https://lenguajejs.com/javascript/fundamentos/funciones/#callbacks

7.3 ARROW FUNCTIONS

Son una forma corta de escribir funciones que aparece en Javascript a partir de **ECMAScript 6**.

Diferencias y limitaciones:

- No tiene sus propios enlaces a this o super y no se debe usar como métodos.
- No tiene argumentos o palabras clave new.target.
- No apta para los métodos call, apply y bind, que generalmente se basan en establecer un ámbito o alcance
- No se puede utilizar como constructor.
- No se puede utilizar yield dentro de su cuerpo.

Fuente:
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Functions/Arrow_functions

7.3 ARROW FUNCTIONS

```
const func = function () {
  return "Función tradicional.";
};

const func = () => {
  return "Función flecha.";
};
```

7.3 ARROW FUNCTIONS

```
// Función tradicional
const f1 = function (a){
  return a + 100;
}

// Desglose de la función flecha

// 1. Elimina la palabra "function" y coloca la flecha entre el argumento y el corchete de apertura.
const f2 = (a) => {
  return a + 100;
}

// 2. Quita los corchetes del cuerpo y la palabra "return" — el return está implícito.
const f3 = (a) => a + 100;

// 3. Suprime los paréntesis de los argumentos
const f4 = a => a + 100;
```

8. ARRAYS

Un Array es un conjunto ordenado de valores relacionados.

Cada uno de estos valores se denomina elemento y cada elemento tiene un índice que indica su posición numérica en el Array.

Es necesario declarar un Array antes de poder usarlo.

8.1 DECLARACIÓN DE ARRAYS

```
const a1 = new Array(); // nuevo array vacío.

const a2 = new Array(4); //Nuevo array de 4 elementos

//nuevo array con tres elementos definidos
const apellidos = new Array('Perez', 'Martínez', 'González');

//así podemos inicializar un array con valores:
for (let i = 0; i < 10; i++) {
    a1[i] = Math.random();
}
```

8.2 RECORRER ARRAYS

Podemos recorrerlo de tres modos:

- Con un bucle for
- Con un bucle for of (desde ES6)
- Con el método forEach

```
//Podemos recorrerlo con el  tradicional bucle for

for (let i = 0; i < a1.length; i++) {
    console.log(a1[i]);
}

//ahora vamos a recorrerlo pero con un bucle for_of (ES6)

for (num of a1) {
    console.log(num);
}

//También podemos recorrerlo con un forEach

a1.forEach(function(elemento, indice, array) {
    console.log(elemento, indice);
})
```

8.3 PROPIEDADES Y MÉTODOS DE LOS ARRAYS

El objeto Array tiene dos propiedades:

1.lenght:

```
for (let i=0; i< codigos_productos.length; i++){
    document.write(codigos_productos[i] + "<br>");
}
```

2.prototype:

```
Array.prototype.nueva_propiedad= valor;
Array.prototype.nuevo_metodo= nombre_de_la_funcion;
```

8.3 PROPIEDADES Y MÉTODOS DE LOS ARRAYS

Métodos	
`push()`	`shift()`
`concat()`	`pop()`
`join()`	`slice()`
`reverse()`	`sort()`
`unshift()`	`splice()`

Documentación:
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array

9. OBJETOS

Un objeto es una colección de propiedades, y una propiedad es una asociación entre un nombre (o *clave*) y un valor.

El valor de una propiedad puede ser una función, en cuyo caso la propiedad es conocida como un método.

Además de los objetos que están predefinidos en el navegador, puedes definir tus propios objetos

Fuente:
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Working_with_Objects

9.1 DECLARACIÓN DE OBJETOS

Podemos declararlos de dos modos:

```
//podemos declarar un objeto vacío e ir creando sus propiedades
const myCar = new Object();
myCar.make = 'Ford';
myCar.model = 'Mustang';
myCar.year = 1969;

//podemos crear directamente un objeto dando sus propiedades y métodos

const myCar2 = {
    make: 'Seat',
    model: '127',
    year: '1975'

}
```

9.1 DECLARACIÓN DE OBJETOS

Así podemos declarar métodos: Es una propiedad que a su vez es una función.

La palabra reservada this nos permite acceder a las propiedades del propio objeto:

```
//declaración de un objeto con propiedades y métodos
const persona = {
    nombre: 'juan',
    apellido: 'sin miedo',
    edad: 30,
    saludar: function () {
        //this es la palabra reservada para acceder a un valor del objeto
        console.log(`Me llamo ${this.nombre}`);
        console.log('Hola!');
    }
};
```

9.2 FUNCIÓN CONSTRUCTORA

Podemos crear una función constructora y luego crear instancias de un objeto.

```
function Persona(nombre, apellido, fechaNac) {
    this.nombre = nombre;
    this.apellido = apellido;
    this.anioNac = fechaNac;
    this.getFechaNac = function () {
        return this.anioNac;
    }
}

//creamos una instancia del objeto Persona
const p1 = new Persona('Pepe', 'Pérez', 1975);
```

9.3 DECLARACIÓN DE CLASES

Desde ES6 podemos declarar clases. La clase tendrá un método “constructor”.

Podemos declara métodos setters y getters.

```
class Persona {
    //método constructor
    constructor(nombre, apellidos, anioNac) {
        this._nombre = nombre;
        this._apellidos = apellidos;
        this._anioNac = anioNac
    }
    //podemos declarar métodos getters, setters...
    get anioNac() {
        return this._anioNac;
    }

}
```