

UT01: Fundamentos de JavaScript

let, var, const

Las variables declaradas con **let** no pueden ser redeclaradas.
Las variables declaradas con **let** deben ser declaradas antes de su uso.
Las variables declaradas con **let** tienen alcance de bloque.

Las variables declaradas con **const** no pueden ser redeclaradas.
Las variables declaradas con **const** no pueden ser reasignadas.
Las variables declaradas con **const** tienen alcance de bloque.

Las variables declaradas con **var** funcionan como las declaradas con **let**, pero no tienen alcance de bloque, sino que actúan como globales.

Tipos de datos

Los tipos de datos no se declaran explícitamente, sino que se determinan en función del valor que le asignamos a las variables.
El tipo de dato de una variable puede ser dinámico, esto quiere decir que el tipo de datos de una variable puede cambiar al asignarle un nuevo valor.
Con el operador **typeof** podemos determinar el tipo de datos de una variable.

prompt, alert, confirm e innerHTML

prompt muestra un diálogo modal para indicarle al usuario que introduzca información.
alert nos permite mostrar un mensaje al usuario.
confirm muestra un diálogo en el que le preguntamos algo al usuario, dándole la oportunidad de que conteste sí o no.
Para mostrar información de un modo "más avanzado", utilizaremos **innerHTML** para escribir código HTML.

String() y Array()

h.concat(m);

'HolaMundo'

Existen mejores forma de concatenar strings

h.startsWith("H");

true

Comprobar si un string comienza con otro string

h.endsWith("a");

true

Comprobar si un string finaliza con otro string

h.includes("o");

true

Comprobar si un string incluye otro string

h.toLowerCase();

'hola'

Convertir a minúsculas

h.toUpperCase();

'HOLA'

Convertir a mayúsculas

h.repeat(2);

'holahola'

Repetir el string

h.replace("ola","ello");

'Hello'

Sustituir una parte de string por otro

h.slice(1,3);

'ol'

String entre la posición 1 y 3

h.substring(1,3); ???

'ol'

h.substr(1,3);

'ola'

String desde la posición 1 y tamaño 3

dias.split(" ");

['', 'lun', 'mar', 'mié', '']

"trocea" el string y lo convierte en un array

dias.trim();

'lun mar mié'

Quitar espacios iniciales y finales

dias.trim().split(" ");

['lun', 'mar', 'mié']

Combinar las dos operaciones anteriores en una sola línea de código

dias.trim().split(" ").join(", ");

'lun, mar, mié'

Volver a unir con el método join de Array, utilizando un nuevo separador

NaN (Not a Number)

typeof(NaN) - 'number'

NaN

NaN

NaN representa un valor numérico indeterminado o imposible. Es un valor global.

Number.NaN

NaN

También es una propiedad de Number.

typeof(NaN)

number

Aunque NaN es el acrónimo de "Not a Number", su tipo de datos es number.

NaN.constructor.name

number

Otra manera de averiguar el tipo de datos de NaN.

Number.isNaN(NaN)

true

Efectivamente, NaN es NaN

NaN == NaN

false

Todos los NaN son diferentes, son valores numéricos indeterminados.

Number.isNaN(3)

false

3 es un número concreto y por tanto no es indeterminado.

Number.isNaN("tres")

false

"tres" es un string y no un número indeterminado.

Number.isNaN(NaN)

true

Cierto, NaN es un número indeterminado.

```
let num = parseInt("tres")
```

Number.isNaN(num)

true

Cierto, debemos utilizar el método isNaN para determinar si una variable es un número indeterminado o NaN.

```
let num = parseInt("tres")
```

num == NaN

false

num es NaN, pero todos los NaN son diferentes porque son indeterminados.

Number.isNaN(3-"dos")

true

Cierto, restarle el texto "dos" al número 3 da como resultado un número imposible.

Number.isNaN(3-NaN)

true

Cierto, una operación con NaN da como resultado un número indeterminado o NaN.

String(NaN)

"NaN"

Estamos intentando convertir NaN String mediante un **typecast** (conversión explícita). La conversión es posible y el resultado es el texto "NaN".

Number(NaN)

NaN

Ya era un número indeterminado.

Bucles clásicos

pruebaFor(): muestra los 6 niveles de encabezado utilizando un bucle for

```
function pruebaFor() {
  document.write('<h1 style="color:red;">Bucle for</h1>');
  for (let i = 1; i <= 6; i++) {
    document.write('<h${i}>Encabezado nivel ${i}</h${i}>');
  }
}
```

pruebaWhile(): muestra los 6 niveles de encabezado utilizando un bucle while

```
function pruebaWhile() {
  document.write('<h1 style="color:red;">Bucle while</h1>');
  let i = 1;
  while (i <= 6) {
    document.write('<h${i}>Encabezado nivel ${i}</h${i}>');
    i++;
  }
}
```

pruebaDoWhile(): muestra los 6 niveles de encabezado utilizando un bucle do while

```
function pruebaDoWhile() {
  document.write('<h1 style="color:red;">Bucle do while</h1>');
  let i = 1;
  do {
    document.write('<h${i}>Encabezado nivel ${i}</h${i}>');
    i++;
  } while (i <=6)
}
```

pruebaContinue(): sin modificar el bucle for anterior, utiliza continue para no mostrar el nivel 4.

```
function pruebaContinue() {
  document.write('<h1 style="color:red;">Continue</h1>');
  for (let i = 1; i <= 6; i++) {
    if (i == 4) continue;
    document.write('<h${i}>Encabezado nivel ${i}</h${i}>');
  }
}
```

```
}
```

pruebaBreak(): sin modificar el bucle for anterior, utiliza break para mostrar hasta el nivel 4.

```
function pruebaBreak() {
  document.write('<h1 style="color:red;">Break</h1>');
  for (let i = 1; i <= 6; i++) {
    document.write('<h${i}>Encabezado nivel ${i}</h${i}>');
    if (i == 4) break;
  }
}
```

Funciones

Una **expresión de función** consiste en "guardar" una función en un variable, con lo que el typeof de la variable será function. En la expresión podemos prescindir del nombre de la función, por lo que tendríamos una función anónima.

```
// Declaración de Función
function sum(a, b) {
  return a + b;
}
```

```
// Expresión de Función
let sum = function(a, b) {
  return a + b;
};
```

Funciones callback

Son funciones que son pasadas por parámetros a otras funciones para ser llamadas posteriormente.

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
```

```
function showOk() {
  alert( "Estás de acuerdo." );
}
```

```
function showCancel() {
  alert( "Cancelaste la ejecución." );
}
```

```
// uso: las funciones showOk, showCancel son pasadas como argumentos de ask
ask("Estás de acuerdo?", showOk, showCancel);
```

Funciones flecha

Las funciones de flecha se pueden usar de la misma manera que las expresiones de función. Por ejemplo, para crear dinámicamente una función.

```
let func = (arg1, arg2, ..., argN) => expression;
```

```
let age = prompt("What is your age?", 18);
let welcome = (age < 18) ?
  () => alert('¡Hola!') :
  () => alert("¡Saludos!");

welcome();
```

Expresiones de función ejecutadas inmediatamente (IIFE)

Son funciones que se ejecutan tan pronto como se definen.

```
(function () {
  statements;
})();
```

Objeto arguments

Hay funciones que tienen un número de parámetros variables. Con el objeto arguments accesible desde dentro de las funciones declaradas con function (no funciones en funciones flecha) podemos manipular el conjunto de argumentos.

```
//Ejemplo de uso de objeto arguments
function f1(){
  return `Número de parámetros: ${arguments.length} Parámetro 1:
${arguments[0]}`
}
```

Parámetros rest

Otra opción para manipular los parámetros cuando son variables en cuanto a número, son los parámetros rest.

```
//Ejemplo de uso de parámetros rest
function f2(a,b,c,...d){
  return `Número de parámetros variables: ${d.length} Parámetro 1 de
la colección: ${d[0]}`
}
function pruebaBucles(...pruebas) {
  for (let i = 0; i < pruebas.length; i++) {
    pruebas[i]();
    document.write('<br>');
  }
}
```

Opciones de gestión de errores (try, catch y throw)

```
function getNombreMes(mes) {
  mes = mes - 1; // Ajustar el número de mes al índice del array (1 =
Ene, 12 = Dic)
  var meses = new Array("Ene", "Feb", "Mar", "Abr", "May", "Jun",
"Jul", "Ago", "Sep", "Oct", "Nov", "Dic",
);
  if (meses[mes] != null)
    return meses[mes];
  throw("Número de mes no válido");
}
```

```

}

// La función se usa así
try{
  getNombreMes(25);
  // Código para utilizar el mes obtenido
}
catch (e){
  alert(e); // Interfaz de usuario
}

```

Método forEach()

```

function mostrarLista(listaFrutas){
  document.write('<ul>')
  listaFrutas.forEach((fruta) =>
{document.write('<li>${fruta}</li>`'))
  document.write('</ul>')
}

mostrarLista(["pera", "platano", "manzana"]);

```

Objetos

```

let personal={}
personal.nombre="Juan"
personal.apellido1="García"
personal.apellido2="González"

let persona2= new Object
persona2.nombre="Antonio" // Notación de puntos
persona2['apellido1']="Pérez" // Notación de corchetes
persona2.apellido2="Pérez"

let persona3 = {nombre: "Ana", apellido1: "Díaz", apellido2: "Díaz"}
function mostrarNombre(){console.log(`${this.nombre} ${this.apellido1}
${this.apellido2}`)}

personal.mostrar=mostrarNombre
persona2.mostrar=mostrarNombre
persona3.mostrar=mostrarNombre
personal.mostrar();

listaPersonas = []
listaPersonas.push(personal)
listaPersonas.push(persona2)
listaPersonas.push(persona3)
listaPersonas.forEach(persona => {persona.mostrar()})

```

JSON y persistencia mediante localStorage

```
let obtenerJSON = (objeto) => JSON.stringify(objeto);
```

```
let almacenarObjeto = (key, objeto) => localStorage.setItem(key, obtenerJSON(objeto));
```

Definición de clases (enfoque clásico)

```
// función constructora sin encapsulamiento
let agenda = [];
function Persona(pNombre, pApellidos){
    this.nombre = pNombre; //Propiedad pública
    this.apellidos = pApellidos; //Propiedad pública
    this.mostrar=function(){console.log(`${this.apellidos}, ${this.nombre}`)}
}
let p1=new Persona("Juan", "Pérez")
let p2=new Persona("Pedro", "Pérez")
agenda.push(p1,p2);
agenda.forEach(persona=>{persona.mostrar()})

// función constructora con encapsulamiento
let agenda = [];
function Persona(pNombre, pApellidos){
    let nombre = pNombre; //propiedad privada
    let apellidos = pApellidos; //propiedad privada
    this.setNombre = function(pNombre){nombre=pNombre} //Método público
    this.setApellidos = function(pApellidos){apellidos=pApellidos}
    //Método público
    this.mostrar=function(){console.log(`${apellidos}, ${nombre}`)}
    //Método público
}
let p1=new Persona("Juan", "Pérez")
let p2=new Persona("Pedro", "Pérez")
agenda.push(p1,p2);
agenda.forEach(persona=>{persona.mostrar()})
```

Date()

let momentoCero = new Date(0);

Thu Jan 01 1970 00:00:00 GMT+0000

¿Cómo se guardan internamente las fechas? ... un número ... milisegundos desde 01 de enero de 1970. A ese número se le denomina timestamp, en este ejemplo es 0. Se retorna un objeto fecha.

let antesDeCero = new Date(-1);

Wed Dec 31 1969 23:59:59

¿Qué hay antes del momento cero? Con números negativos podemos representar fechas anteriores al 01 de enero de 1970. En este ejemplo se retorna también un objeto fecha.

Date.parse("1970-01-01 00:00:0000")

0

Otra forma de obtener el momento 0 es mediante un método estático. NO se retorna un objeto fecha sino un número o timestamp.

```
let fch = new Date("2021-10-30 00:00");
```

Sat Oct 30 2021 00:00:00 GMT+0100

Podemos obtener un objeto fecha con new Date pasando por parámetro un string con formato año-mes-día.

```
let fch = new Date(2021, 09, 30,0,0);
```

Sat Oct 30 2021 00:00:00 GMT+0100

Otra forma más que da el mismo resultado que el ejemplo anterior.

```
let timeStamp=fch.getTime();
```

1635548400000

Con getTime podemos obtener el timestamp de un objeto fecha.

```
let fch=new Date("2020/30/30")
```

```
fch.getTime()
```

NaN

fch es "Invalid Date"

```
let unDia = 24*60*60*1000;
```

86400000

86400000 es el número de milisegundos que tiene un día.

```
let unaHora = 60*60*1000;
```

3600000

3600000 es el número de milisegundos que tiene una hora.

```
let fch=new Date(2021,10,30);
```

```
fch.setFullYear(2022);
```

```
console.log(fch.getFullYear());
```

2022

setFullYear cambia el año a un objeto fecha.

Date()	
METHODS	
<code>n.UTC(y, m, d, h, i, s, ms)</code>	timestamp
<code>n.now()</code>	timestamp of current time
<code>n.parse(str)</code>	convert str to timestamp
<code>n.setTime(ts)</code>	set UNIX timestamp
<code>n.getTime()</code>	return UNIX timestamp
UNIT GETTERS / SETTERS (ALSO <code>getUTC()</code> / <code>setUTC()</code>)	
<code>n.get / .setFullYear(y, m, d)</code>	(yyyy)
<code>n.get / .setMonth(m, d)</code>	(0-11)
<code>n.get / .setDate(d)</code>	(1-31)
<code>n.get / .setHours(h, m, s, ms)</code>	(0-23)
<code>n.get / .setMinutes(m, s, ms)</code>	(0-59)
<code>n.get / .setSeconds(s, ms)</code>	(0-59)
<code>n.get / .setMilliseconds(ms)</code>	(0-999)
<code>n.getDay()</code>	return day of week (0-6)
LOCALE & TIMEZONE METHODS	
<code>n.getTimezoneOffset()</code>	offset in mins
<code>s.toLocaleDateString(locale, options)</code>	
<code>s.toLocaleTimeString(locale, options)</code>	
<code>s.toLocaleString(locale, options)</code>	
<code>s.toUTCString()</code>	return UTC date
<code>s.toString()</code>	return American date
<code>s.toTimeString()</code>	return American time
<code>s.toISOString()</code>	return ISO8601 date
<code>s.toJSON()</code>	return date ready for JSON

Internacionalización (I18n)

Código para escribir en consola	Resultado al ejecutar	Comentarios
<pre>new Intl.DateTimeFormat("es", { weekday: "short", day: "2-digit", month: "long", year: "numeric", hour: "2-digit", minute: "2-digit" }).format(new Date(2021,09,10,8,0));</pre>	dom, 10 de octubre de 2021, 08:00	
<pre>new Intl.DateTimeFormat("ja", { weekday: "short", day: "2-digit", month: "long", year: "numeric", hour: "2-digit", minute: "2-digit" }).format(new Date(2021,09,10,8,0));</pre>	2021 年 10 月 10 日 曜日 08:00	¡En japonés!
<pre>let importe=125.86; Intl.NumberFormat('en-EN', { style: 'currency', currency: 'EUR' }).format(importe)</pre>	€125.86	Formato en inglés
<pre>let importe=125.86; Intl.NumberFormat('es-ES', { style: 'currency', currency: 'EUR' }).format(importe)</pre>	125,86 €	Formato en español

Atributo	Significado	Posibles valores
weekday	Día de la semana	long, short o narrow.
day	Día del mes	numeric o 2-digit.
month	Mes	numeric, 2-digit, long, short o narrow.
year	Año	numeric o 2-digit.
era	Era actual	long, short o narrow.
hour	Hora	numeric o 2-digit.
hour12	formato 12/24 horas	Lo activa o lo desactiva
minute	Minutos	numeric o 2-digit.
second	Segundos	numeric o 2-digit.
fractionalSecondDigits		Dígitos de las fracciones de segundos: 1, 2 o 3.
timeZoneName		Nombre de la zona horaria: long o short.

UT02: Definición y gestión de colecciones y objetos

map(), filter() y reduce()

```
let a=[1,2,3,4]
```

```
a.map((e)=>{return e*2})
```

```
[2, 4, 6, 8]
```

map retorna un array con el mismo número de elementos que el original, pero "modificados". En este caso la modificación consiste en multiplicarlos por 2.

```
a.filter((e)=>{return !(e%2)})
```

```
[2, 4]
```

filter retorna un array de los elementos que cumplen una condición determinada que hemos programado. En este ejemplo la condición es que sea divisible por 2.

```
a.reduce((ac,e)=>{return ac+e})
```

```
10
```

reduce "reduce" todos los elementos de un array a uno solo. Durante las iteraciones el acumulador (ac) va creciendo. Cuando finalizan las iteraciones se retorna el acumulador que se ha ido construyendo iteración a iteración.

```
a.reduce((ac,e)=>{return ac+e},5)
```

```
15
```

Podemos definir un valor inicial para el acumulador.

```
a.filter((e)=>{return !(e %2)}).reduce((ac,e)=>{return ac+e})
```

```
6
```

Podemos encadenar filter() y reduce() para obtener la suma de los números pares.

```
a.map((e)=>{return `${e}`})
```

```
['<li>1</li>','<li>2</li>','<li>3</li>','<li>4</li>']
```

Map() es ideal también para transformar elementos de un array mediante concatenación.

```
a.map((e)=>{return `${e}`}).join("")
```

```
'<li>1</li><li>2</li><li>3</li><li>4</li>'
```

Ahora combinamos join() con map().

```
document.body.innerHTML=`<ul>${a.map((e)=>{return`<li>${e}</li>`}).join("")}</ul>`
```

- 1
- 2
- 3
- 4

Y ya el resultado final en el document.

some(), every(), find(), findIndex() y reduceRight()

[1,2,3,4].some(e => e<3)

true

¿Hay algún elemento menor que 3? Sí

[1,2,3,4].every(e => e<3)

false

¿Todos los elementos son menores que 3? No

["lun", "mar", "mie", "jue", "vie","sab"].findIndex(dia => dia=="mie")

2

Devuelve el índice del primer elemento que cumple la condición.

["lun", "mar", "mie", "jue", "vie","sab"].findIndex(dia => dia=="dom")

-1

No lo ha encontrado.

{diaCorto: "lun", diaLargo: "lunes"}, {diaCorto: "mar", diaLargo: "martes"}].find(dia => dia.diaCorto=="mar").diaLargo

'martes'

Devuelve un objeto que cumple con la condición.

{diaCorto: "lun", diaLargo: "lunes"}, {diaCorto: "mar", diaLargo: "martes"}].find(dia => dia.diaCorto=="dom").diaLargo

Uncaught TypeError: Cannot read properties of undefined

¡OJO! Excepción porque find devuelve undefined y estamos intentando obtener la propiedad diaLargo de undefined

["lun", "mar", "mie", "jue", "vie","sab"].reduceRight((diasConcatenados , dia) => diasConcatenados + "-" + dia)

'sab-vie-jue-mie-mar-lun'

Concatenación con reduceRight reduce de derecha a izquierda al contrario que reduce.

splice(): borrar, insertar y reemplazar

let dias=["lun", "mar", "mie", "dom", "vie","mar"]

dias.splice(3,0,"jue")

['lun', 'mar', 'mie', 'jue', 'dom', 'vie', 'mar']

Insertar un sólo elemento. Insertar un elemento en la tercera posición.

dias.splice(4,1)

['lun', 'mar', 'mie', 'jue', 'vie', 'mar']

Borrar un sólo elemento. Borrar el elemento que está en la cuarta posición.

dias.splice(5,1,"sab")

['lun', 'mar', 'mie', 'jue', 'vie', 'sab']

Reemplazar un sólo elemento. Reemplazar el quinto elemento.

```
let dias=["lun", "mar", "mie", "dom", "vie","mar"]
dias.splice(3,3,"jue","vie","sab")
['lun', 'mar', 'mie', 'jue', 'vie', 'sab']
Reemplazar varios elementos.
```

```
let dias=[{diaCorto: "lun", diaLargo: "lunes"}]
dias.splice(1,0, {diaCorto: "mar", diaLargo: "martes"})
[{...}, {...}]
Insertar un objeto completo.
```

```
let dias=["lun", "mar", "mie", "dom", "vie","mar"]
dias.splice(-1,1)
['lun', 'mar', 'mie', 'dom', 'vie']
Borrar el último elemento. Similar al método pop.
```

spread() y Set()

```
let lenguajesAna=programadores[1].lenguajes
let lenguajesPedro=programadores[2].lenguajes
let lenguajesAnaPedro = [...lenguajesAna , ...lenguajesPedro ]
['C', 'JS', 'Java', 'Python', 'JS', 'Java', 'C++']
Obtenemos un array de 7 elementos resultantes de la unión de los lenguajes de Ana con los de Pedro. Podemos observar que hay lenguajes repetidos. Esto sería equivalente a concatenarlos con el método concat, de la siguiente forma lenguajesAna.concat(lenguajesPedro)
```

```
let a=[2,3,4]
let b=[7,8,9]
let c=[1,...a,'cinco', {num: 'seis'},...b]
[1, 2, 3, 4, 'cinco', {...}, 7, 8, 9]
En este ejemplo combinamos en el array c elementos de diferentes tipos de datos junto con el operador spread.
```

```
let lenguajesAnaPedro = new Set([...lenguajesAna,...lenguajesPedro])
Set(5) {'C', 'JS', 'Java', 'Python', 'C++'}
Podemos utilizar el operador spread para obtener el conjunto de los distintos lenguajes de programación de Ana y Pedro. Obtenemos un conjunto de 5 elementos, sin elementos repetidos.
```

```
let lenguajesAnaPedro = Array.from(new Set([...lenguajesAna,...lenguajesPedro]))
['C', 'JS', 'Java', 'Python', 'C++']
Si queremos convertir el Set en un array, podemos utilizar el método from de la clase Array.
```

```
programadores.map(programador => {return { ...programador, menorDe25: (programador.edad <= 25 ? true : false)}})
```

Obtenemos 3 objetos programador con 6 atributos

En este ejemplo queremos añadir un nuevo atributo a los objetos programador. El uso del operador spread nos evita tener que escribir los nombres de los 5 atributos de programador y además el código es más corto y legible porque al ver "...programador" sabemos que se refiere a todos los atributos del objeto.

sort()

```
let numeros=[3,2,6,1,9,5,30,20]
```

```
numeros.sort((primerElemento,segundoElemento)=> { if (primerElemento < segundoElemento) return -1; else return 1 })
```

```
[1, 2, 3, 5, 6, 9, 20, 30]
```

Si retornamos un valor > 0, entonces se ordena primerElemento después de segundoElemento.

Si retornamos un valor < 0, entonces se ordena primerElemento antes que segundo elemento.

Si retornamos un valor ===0, entonces se mantiene el orden original de los elementos

```
numeros.sort((a,b)=> (a<b ? -1 : 1))
```

```
[1, 2, 3, 5, 6, 9, 20, 30]
```

De forma más resumida podemos utilizar el operador ternario de comparación, y utiliza otros nombres para los parámetros. Se suele utilizar a y b.

```
numeros.sort((a,b) => a-b)
```

```
[1, 2, 3, 5, 6, 9, 20, 30]
```

La forma más sencilla simplemente es:

a-b si queremos ordenar de forma **ascendente**

b-a si queremos ordenar de forma **descendente**.

```
numeros.sort()
```

```
[1, 2, 20, 3, 30, 5, 6, 9]
```

Si no indicamos ningún criterio de ordenación se realiza una ordenación convirtiendo números a texto.

```
programadores.sort((a,b) => a.nombre.localeCompare(b.nombre))
```

Obtenemos 3 objetos con el orden Ana, Antonio, Pedro

Para ordenar **string**, debemos utilizar el método localeCompare. En este ejemplo se ordenan los programadores por nombre en orden **ascendente**.

```
programadores.sort((a,b)=>b.nombre.localeCompare (a.nombre))
```

Obtenemos 3 objetos con el orden Pedro, Antonio, Ana

Se ordenan los programadores por nombre en orden **descendente**.

Ejemplos map(), filter() y reduce()

```
let pilots1 = pilots.map(pilot => pilot = `${pilot.name}, experiencia: ${pilot.years}`);
console.log(pilots1);
```

```
let pilots2 = pilots.map(pilot => pilot = { name: pilot.name, years: pilot.years });
let pilots2v2 = pilots.map(pilot => ({ name: pilot.name, years: pilot.years }));
console.log(pilots2);
```

```
let pilots3 = pilots.filter(pilot => pilot.years > 16);
console.log(pilots3);
```

```
let pilots4 = pilots.reduce((exp, pilot) => exp + pilot.years, 0);
console.log(pilots4);
```

```

let pilots5 = pilots.filter(pilot => pilot.years > 15).map(pilot => ({
name: pilot.name, id: pilot.id }));
console.log(pilots5);

let pilots6 = pilots.map(pilot => ({ id: pilot.id, iniciales:
pilot.name.split(" ").map(i => i[0]).join(".") }));
console.log(pilots6);

let pilots7 = pilots.map(pilot => ({ id: pilot.id, iniciales:
pilot.name.split(" ").reduce((inic,palabra) => (inic+palabra[0]+'.'),
'') }));
console.log(pilots7);

let ayuntamientos1 = municipios.filter(municipio => municipio.poblacion
< 2000);
console.log(ayuntamientos1);

let ayuntamientos2 = municipios.filter(municipio => municipio.poblacion
< 2000).map(municipio => ({ municipio: municipio.municipio, poblacion:
municipio.poblacion }));
console.log(ayuntamientos2);

let ayuntamientos3 = municipios.filter(municipio => municipio.isla ==
"La Palma").reduce((poblacion, municipio) =>
poblacion+municipio.poblacion, 0);
console.log(ayuntamientos3);

let ayuntamientos4 = municipios.map(municipio =>
(`${municipio.municipio}, población: ${municipio.poblacion}`));
console.log(ayuntamientos4);

let ayuntamientos5 = municipios.filter(municipio => municipio.isla ==
"La Gomera").map(municipio => ({ municipio: municipio.municipio,
poblacion: municipio.poblacion }));
console.log(ayuntamientos5);

```

Ejemplos sort()

```

function sort1(municipios) {
  return municipios.filter(municipio => municipio.poblacion <
2000).map(municipio => ({ municipio: municipio.municipio, poblacion:
municipio.poblacion })).sort((a, b) => a.poblacion-b.poblacion)
}

function sort2(municipios) {
  return municipios.filter(municipio => municipio.poblacion < 2000)
.map(municipio => ({ municipio: municipio.municipio, poblacion:
municipio.poblacion })).sort((a, b) => b.poblacion-a.poblacion);
}

function sort3(municipios) {
  return municipios.filter(municipio => municipio.poblacion <
2000).map(municipio => ({ municipio: municipio.municipio, poblacion:
municipio.poblacion })).sort((a, b) =>
a.municipio.localeCompare(b.municipio))
}

```

```
function sort4(municipios) {
  return municipios.filter(municipio => municipio.poblacion <
2000).map(municipio => ({ municipio: municipio.municipio, poblacion:
municipio.poblacion })).sort((a, b) =>
b.municipio.localeCompare(a.municipio))
}
```

Desestructuración

```
let carrera = {nombre: "maratón", distancia:42, totalParticipantes:300}
let a = [1, 2, 3, 4]
```

Desestructuración con todos los elementos de un array

```
let [uno,dos,tres,cuatro] = a;
```

Desestructuración del primer y último elemento de un array

```
let [uno,,,cuatro] = a;
```

Desestructuración de array omitiendo los 2 primeros elementos

```
let [,...resto] = a;
console.log(resto) → [3,4]
```

Desestructurar array utilizando operador rest

```
let [uno, dos, ...rest] → a;
console.log(uno) → 1;
console.log(dos) → 2;
console.log(rest) → [3,4]
```

Desestructurar array utilizando una función

```
function obtenerDistancias() {
  return [42,21,8,4];
}
let [maraton,mediamaraton,...otrasDistancias] = obtenerDistancias();
```

Intercambio de valores

```
let km4 = 8;
let km8 = 4;
[km4,km8] = [km8,km4]
console.log(km4) → 4
```

Desestructurar objeto

En lugar de

```
let nombre = carrera.nombre;
let distancia = carrera.distancia;
let totalParticipantes = carrera.totalParticipantes;
```

Podemos hacer

```
let {nombre,distancia,totalParticipantes} = carrera;
```

Desestructurar objeto cambiando nombre de atributo

```
let {nombre,distancia,totalParticipantes:inscritos} = carrera;
```


Desestructurar objeto con operador rest

```
let {nombre,...otrosDatos} = carrera;  
console.log(nombre) → maratón  
console.log(otrosDatos) → {distancia: 42, totalParticipantes: 300}
```

Renombrar directamente

```
pilots.map(pilot=>`${pilot.name}`)  
es igual a  
pilots.map(({name}) => `${name}`)  
renombrar  
pilots.map(({name:nombre}) => `${nombre}`)
```

UT03: Definición y gestión de colecciones y objetos

Importación y exportación

El sistema de módulos actual de JS, denominado ECMAScript Modules o EMS facilita la organización del código en aplicaciones grandes ya que JS provee un sistema mediante el que los módulos o ficheros .js pueden exportar elementos (funciones, constantes, etc) que son importadas por otros módulos o ficheros .js.

<https://lenguajejs.com/javascript/modulos/que-es-esm/>

Los ficheros .js pueden tener un módulo de **exportación** que permiten poner elementos a disposición de otros ficheros .js.

<https://lenguajejs.com/javascript/modulos/export/>

La **importación** de elementos o import es la operación inversa a la exportación o export.

<https://lenguajejs.com/javascript/modulos/import/>

Seleccionar elementos: hijos, hermanos y padre

getElementsByTagName

document.**getElementsByTagName**("li")

Retorna un colección con todos los elementos que tienen la etiqueta que hemos especificado

document.**getElementsByTagName**("li")[1]

Permite seleccionar un elemento determinado indicando su posición en la colección.

previousElementSibling

document.**getElementsByTagName**("li")[1].**previousElementSibling**

Permite seleccionar el elemento anterior de uno dado

nextElementSibling

document.**getElementsByTagName**("li")[1].**nextElementSibling**

Selecciona el elemento posterior a un elemento dado

children

document.**getElementsByTagName**("ul")[0].**children**[1]

Retorna una colección de tipo HTMLCollection con todos los elemento hijos

firstElementChild

document.**getElementsByTagName**("ul")[0].**firstElementChild**

Retorna el primer elemento hijo de un elemento dado

lastElementChild

document.**getElementsByTagName**("ul")[0].**lastElementChild**

Retorna el último elemento hijo de un elemento dado

parentElement

document.**getElementsByTagName**("ul")[0].**parentElement**

Retorna el elemento padre de un elemento dado

hasChildNodes

document.getElementsByTagName("ul")[0].hasChildNodes()

Retorna true si un elemento tiene nodos hijo

childElementCount

document.getElementsByTagName("ul")[0].childElementCount

Retorna el número de elementos hijo que tiene un elemento

Seleccionar nodos: hijos y hermanos

previousSibling

document.getElementsByTagName("li")[2].previousSibling

Permite seleccionar el nodo anterior de uno dado

nextSibling

document.getElementsByTagName("li")[1].nextSibling

Selecciona el nodo posterior a un elemento dado

childNodes

document.getElementsByTagName("ul")[0].childNodes

Retorna una colección de tipo NodeList con todos los nodos hijos

children

document.getElementsByTagName("ul")[0].children

Retorna, a diferencia de childNode, una colección de tipo HTMLCollection con todos los elemento hijos

firstChild

document.getElementsByTagName("ul")[0].firstChild

Retorna el primer nodo hijo de un elemento dado

lastChild

document.getElementsByTagName("ul")[0].lastChild

Retorna el último nodo hijo de un elemento dado

Seleccionar y modificar elementos

getElementById

document.getElementById("pizza")

Retorna el elemento cuyo id hemos indicado como parámetro

getElementsByClassName

document.getElementsByClassName("roja")

Retorna una colección con los elementos que tienen la clase CSS que hemos especificado por parámetro

querySelectorAll

frutasRojas=document.querySelectorAll(".roja")

Retorna una colección con TODOS los elementos seleccionados por el selector CSS que hemos indicado como parámetro

elementosReceta=document.querySelectorAll("body>ul>li")

Permite realizar selecciones de elementos utilizando toda la potencia de los selectores CSS

ingredientesMacedonia=elementosReceta[0].nextElementSibling.querySelectorAll("li")

Lo podemos utilizar también en un "subárbol" del documento

querySelector

primeraFrutaRoja = document.querySelector(".roja")

Retorna sólo el PRIMER elemento de todos los seleccionados por el selector CSS que hemos indicado como parámetro

listaCardinales=document.querySelector("ul")

Permite seleccionar un elemento para luego añadirle elementos hijo.

style

primeraFrutaRoja.style.backgroundColor="red"

Permite modificar el estilo de un elemento previamente seleccionado

textContent

primeraFrutaRoja.textContent="Fresón - 50gr"

Modifica el texto de un elemento

ingredientesMacedonia[0].textContent

Permite obtener el texto de un elemento

forEach

frutasRojas.forEach(function(elemento){elemento.style.backgroundColor="red"})

Permite recorrer la colección de elementos previamente seleccionado para modificar, por ejemplo, el estilo de los elementos

createElement, appendChild, insertAdjacentElement

createElement

miLi=document.createElement("li")

Es un método de **document** que crea un elemento del tipo que le pasemos por parámetro. Inicialmente dicho elemento estará vacío y "huérfano"

appendChild

listaCardinales.appendChild(miLi)

Permite añadir a un elemento un hijo que le pasamos por parámetro

insertAdjacentElement

document.body.insertAdjacentElement("afterbegin",miL1)

Permite añadir un hijo a un elemento. Podemos especificar 4 opciones para el primer parámetro: beforebegin, afterbegin, beforeend y afterend

removeChild

removeChild

document.body.removeChild(miL1)

Permite eliminar un nodo hijo de un elemento dado

listaIngredientes=document.getElementById("pizza").nextElementSibling

El método getElementById permite seleccionar una receta por id

platano=listaIngredientes.getElementsByTagName("li")[2]

El método getElementsByTagName lo podemos utilizar para seleccionar el ingrediente que queremos eliminar

document.getElementById("pizza").nextElementSibling.removeChild(platano)

El método removeChild permite eliminar el ingrediente que ya hemos seleccionado

Attribute, classList

className

document.getElementById("tarta").className="roja"

Es una propiedad que tienen todos los elementos. Nos permite añadir o modificar el atributo "class" de cualquier elemento

removeAttribute

document.getElementById("tarta").removeAttribute("class")

Permite eliminar un atributo de un elemento.

setAttribute

document.getElementById("tarta").setAttribute("class","roja")

Permite añadir un atributo a un elemento. Es una alternativa a className

classList.add

document.querySelectorAll(".roja").forEach(function(elemento){elemento.classList.add("resaltar-rojas-1")})

Permite modificar el aspecto de los elementos a través de un cambio de clase en lugar de cambiar su estilo de forma directa

classList.remove

document.getElementById("tarta").classList.remove("resaltar-rojas-1")



Permite eliminar una clase de la lista de clases a las que pertenece un elemento

classList.toggle

document.querySelectorAll(".roja").forEach(function(elemento){elemento.classList.toggle("resaltar-rojas-1")})

Permite añadir una clase (si no la tiene) o quitar una clase (si la tiene) de la lista de clases de un elemento. Es como un interruptor que enciende lo que está apagado y apaga lo que está encendido. Devuelve true o false.

Aplica

t	DOMTokenList()	= List of classes
PROPERTIES		
n	.length	number of items
METHODS		
b	.contains(item)	check if item exists
	.add(item)	add item to list
s	.item(n)	return item number n
	.remove(item)	del item from list
b	.toggle(item)	del item if exist, add else