

# JavaScript:



STRINGS:	1
MATH:	3
DATE:	5
ARRAYS:	6
OBJETOS:	9
DOM:	11
EVENTOS:	15
TESTEOS DE FUNCIONES:	17
EXPRESIONES REGULARES:	19
JASMINE:	24

## STRINGS:

**charAt()** : carácter en una determinada posición del texto.

```
let nombre = "Miguel Angel";
for (let i = 0; i < nombre.length; i++) {
    console.log(nombre.charAt(i));
}
```

**concat()** : concatenar cadenas de texto

**endsWith()** : comprobar si un texto termina con una determinada subcadena

```
console.log(frase.endsWith('acordarme'));  
console.log(frase.endsWith('hidalgo'));
```

**includes()** : comprobar si un texto contiene una subcadena

```
console.log(frase.includes('cuyo'));
```

**indexOf()** : buscar la posición de una subcadena dentro de un texto. Se puede indicar el índice para empezar a buscar

```
let frase = "En un lugar de la Mancha de cuyo nombre no quiero  
acordarme";  
console.log(frase.indexOf("Mancha"));
```

**lastIndexOf()** : buscar la última posición de una subcadena dentro de un texto

```
console.log(frase.lastIndexOf("de"));
```

**padEnd()** : rellenar con una subcadena al final del texto

```
variable = variable.padEnd(15, '*');  
console.log("Después de padEnd:", variable);  
  
variable = variable.padStart(20, '#');
```

**padStart()** : rellenar con una subcadena al principio

**repeat()** : repetir una subcadena un número de veces

**replace()** : reemplazar una subcadena dentro de un texto(una sola vez)

```
frase = frase.replace('Mancha', 'Andalucia');
```

**replaceAll()** : reemplazar una subcadena dentro de un texto (varias veces)

**search()** : buscar una subcadena dentro de un texto. Se pueden usar expr. regulares

```
console.log(frase.search("lugar", 12));
```

**slice()** : extraer una subcadena de un texto. Acepta índices negativos

**split()** : dividir un texto usando algún carácter (p.e. una coma)

```
let frase = "Cebollas;Patatas;Pimientos;Tomates";  
let verduras = frase.split(";");
```

**startsWith()** : comprobar si un texto comienza por una subcadena

```
let frase = "En un lugar de la Mancha de cuyo nombre no quiero  
acordarme";  
console.log(frase.startsWith("En"));  
console.log(frase.startsWith("lugar"));
```

**substr()** : extraer una subcadena de un texto

```
let frase = "En un lugar de la Mancha de cuyo nombre no quiero acordarme";

let palabraConSubstr = frase.substr(6, 5);
console.log("Usando substr: " + palabraConSubstr);

let palabraConSubstring = frase.substring(6, 11);
console.log("Usando substring: " + palabraConSubstring);
```

**substring()** : extraer una subcadena de un texto

```
let frase = "En un lugar de la Mancha de cuyo nombre no quiero acordarme";
let frase2 = frase.substring(5, 11);
```

**toLowerCase()** : convertir texto a minúsculas

**toUpperCase()** : convertir texto a mayúsculas

```
console.log(nombre.toUpperCase());
```

**trim()** : quitar de un texto los espacios del principio y final

```
let nombre = "  Javier Soldado";
console.log(nombre.trim());
```

**trimEnd()** : quitar de un texto los espacios del final

**trimStart()** : quitar de un texto los espacios del principio

## MATH:

**abs()** : valor absoluto

```
document.write(Math.abs(3));
```

**ceil()** : función techo (redondeo hacia arriba el entero más cercano)

```
console.log(Math.ceil(2.7));
```

**cos()** : función trigonométrica coseno

```
let cos30 = Math.cos(radianes);
```

**E** : constante de Euler

**exp()** : función exponencial

**floor()** : función suelo (redondeo hacia abajo el entero más cercano)

```
console.log(Math.floor(2.7));
```

**log()** : logaritmo natural (también llamado neperiano)

```
console.log(Math.log10(12.5));
```

**log10()** : logaritmo en base 10

```
console.log(Math.log(15.7));
```

**max()** : función máximo

```
console.log(Math.max(-8, 7))
```

**min()** : función mínimo

```
console.log(Math.min(-8, 7));
```

**PI** : constante PI

```
console.log(Math.PI);
```

**pow()** : función potencia

```
console.log(Math.pow(Math.E, 3));
```

**random()** : número pseudoaleatorio entre 0 y menor estricto que 1

```
console.log(Math.random() * 6 + 1);
```

**round()** : redondear al entero

```
console.log(Math.round(2.7));
```

**sign()** : signo de un número

**sin()** : función trigonométrica seno

```
let sin45 = Math.sin(radianes);
```

**sqrt()** : raíz cuadrada

```
console.log(Math.sqrt(128.12));
```

**tan()** : función trigonométrica tangente

**trunc()** : truncado (no confundir con el redondeo)

Ejemplos:

### **Función para redondear un número con un determinado número de decimales**

```
function redondearDecimales(numero,decimales){  
    return Math.round(numero * Math.pow(10,decimales)) / Math.pow(10,decimales);  
}
```

Donde el argumento *numero* indica el número a redondear y *decimales* el número de decimales que deseamos.

Por ejemplo, si queremos redondear el número 166.386 a dos decimales usaríamos **redondearDecimales(166.386, 2)**

### **Función para convertir grados a radianes**

```
function grados2radianes(grados){  
    return grados/180*Math.PI;  
}
```

---

## DATE:

El objeto **Date** nos permite manipular fechas: obtener el mes de una fecha, el día de la semana, el año, intervalo de tiempo entre dos fechas, etc.

Conceptos relacionados con el objeto Date:

**UTC:** es la hora universal, también llamada GMT (hora en el meridiano 0, que pasa por la ciudad de Greenwich)

**Epoch (o Unix Time):** es el tiempo transcurrido desde el 1/1/1970

En el siguiente enlace de w3schools tienes documentados los distintos métodos y propiedades que proporciona la clase Date. En los exámenes debes tener dichos materiales offline, ya que podrías necesitar usar cualquiera de ellos (debes saber interpretar la sintaxis que aparece en la documentación).

**new Date()** : valor absoluto b

**getDate()** : obtener el día del mes de una fecha (1 a 31)

**getDay()** : obtener el día de la semana de una fecha(0 a 6)

**getFullYear()** : obtener el año de una fecha

**getHours()** : obtener las horas de una fecha (0 a 23)

**getMinutes()** : obtener los minutos de una fecha (0 a 59)

**getMonth()** : obtener el mes de una fecha (0 a 11)

**getTime()** : devuelve el nº msg transcurridos desde el 1/1/1970 hasta la fecha indicada

**getTimezoneOffset()** : devuelve la diferencia horaria (en minutos) con la hora UTC

```
fechaActual.getTimezoneOffset()
```

**now()** : devuelve nº msg transcurridos desde el 1/1/1970 hasta el momento actual

**setDate()** : establecer en una fecha el día del mes (1 a 31)

**setDay()** : establecer en una fecha el día de la semana (0 a 6)

**setFullYear()** : establecer en una fecha el año

**toLocaleDateString()** : muestra en formato texto la parte del día,mes, año de una fecha (sin la hora)

**toLocaleTimeString()** : muestra en formato texto una hora

**toLocaleString()** : muestra en formato texto la parte del día, mes, año y hora de una fecha

### Cómo calcular el día siguiente a una fecha determinada

```
let fecha=new Date(2020,1,28); // 28 de febrero de 2020 (año bisiesto)
```

```
fecha.setDate(fecha.getDate()+1); //29 de febrero de 2020
```

Duplicar una fecha a partir de una ya creada:

```
fecha1 = new Date(2024,10,6);
```

```
fecha2 = new Date(fecha1.getFullYear(), fecha1.getMonth(), fecha1.getDate());
```

Crear un objeto date con una fecha personalizada:

```
let fecha = new Date(2005, 3, 20);
```

Calcular días de una fecha a otra:

```
function calcularDias() {  
    let fecha1 = new  
Date(document.getElementById('fecha1').value);  
    let fecha2 = new  
Date(document.getElementById('fecha2').value);  
  
    let diferenciaMilisegundos = Math.abs(fecha2 - fecha1);  
  
    let diasTranscurridos = Math.ceil(diferenciaMilisegundos /  
(1000 * 60 * 60 * 24));  
  
    console.log(diasTranscurridos);  
}
```

---

## ARRAYS:

**delete:** borrar un elemento del array segun su posicion.

```
delete ciudades[2];
```

**concat** : concatenar arrays

```
let mezcla = frutas.concat(verduras);
```

**copyWithin()** : copiar eltos dentro de un array

**fill()** : rellenar array con un valor

**forEach()** : para recorrer arrays

```
frutas.forEach(function (fruta) {  
    console.log(fruta);  
});
```

**includes()** : comprueba si un array contiene un determinado elto

```
let frutas = ['Plátanos', 'Naranjas', 'Pomelos', 'Fresas'];
console.log(frutas.includes('Naranjas'));
```

**indexOf()** : devuelve el primer índice de un elto buscado

```
let frutas = ['Plátanos', 'Naranjas', 'Pomelos', 'Fresas'];
console.log(frutas.indexOf('Pomelos'));
```

**isArray()** : comprueba si un objeto es de tipo Array

**join()** : devuelve el array como texto usando el separador indicado

**lastIndexOf()** : devuelve el último índice de un elto buscado

```
console.log(frutas.lastIndexOf('Plátanos'));
```

**length** : devuelve la longitud del array

**pop()** : extraer un elto del final del arra

**push()** : añadir un elto al final del array

```
ciudades.push('Jaen', 'Granada');
```

**reverse()** : invierte el orden de un array (modifica el array original)

**shift()** : extrae el primer elto de un array

**slice()** : devuelve un subarray (sin modificar el original)

**sort()** : ordenar array (modifica el array original). Solo funciona con texto, para otro tipo de datos (number, objetos, etc) hay que suministrar una función comparadora

```
ciudades.sort();
console.log("Ordenado: " + ciudades);
ciudades.sort().reverse();
console.log("Orden Inverso: " + ciudades);
```

**splice()** : añade/elimina eltos de un array (el array original se ve modificado)

```
<!-- Crea el array ['Plátanos', 'Naranjas', 'Pomelos', 'Fresas'].
      Usando el método splice debes eliminar el segundo elemento del
      array
      (comprueba que realmente se ha eliminado 'Naranjas').
      Posteriormente
      y usando de nuevo splice debes insertar 'Limones' entre el elemento
      Plátanos' y 'Pomelos' (comprueba que realmente se ha insertado
      correctamente). -->
```

```
<script>
  let frutas = ['Plátanos', 'Naranjas', 'Pomelos', 'Fresas'];
  console.log(frutas);

  frutas.splice(1, 1);
  console.log(frutas);

  frutas.splice(1, null, 'Limones');
```

```
console.log(frutas);  
</script>
```

**unshift()** : añadir un elto al principio de un array

```
ciudades.unshift('Cadiz');
```

Métodos que realizan iteraciones sobre el array:

**every(), filter(), find(), findIndex(), findLast(), findLastIndex(), map(), some()**

**every:** El método **every** en JavaScript se utiliza para verificar si **todos los elementos de un array cumplen con una condición** especificada en una función de callback.

```
function esPar(numero) {  
    return numero % 2 === 0;  
}  
let todosPares1 = numeros1.every(esPar);
```

**map:** El método **map** en JavaScript se utiliza para **crear un nuevo array transformando cada elemento del array original** según una función de callback que se ejecuta para cada elemento

```
let cuadrados = numeros.map(cuadrado);  
console.log(cuadrados);  
  
function cuadrado(numero) {  
    return numero * numero;  
}
```

**filter:**El método **filter** en JavaScript se utiliza para **crear un nuevo array que contiene solo los elementos del array original que cumplan una condición** especificada en una función de callback.

```
let impares = numeros.filter(function (numero) {  
    return numero % 2 !== 0;  
});
```

**Find:**el método **find** debes devolver el primer elemento del array que sea un número impar

```
let primerImpar = numeros.find(function (numero) {  
    return numero % 2 !== 0;  
});
```

**// Para ordenar números hay que usar una función comparadora**

```
let numeros=[2, 5, 0, 3.5, -2, 12];
```

```
numeros.sort(function(num1,num2){ return num1-num2; })
```



---

## OBJETOS:

Ejemplo de clase Persona con los siguientes miembros:

- Propiedad `_nombre`
- Propiedad `_edad`
- Método de instancia `cumplirAños()`
- Método de instancia `toString()`
- Getter `nombre`
- Setter `edad`

```
class Persona {  
  
    constructor(nombre,edad){  
        this._nombre = nombre;  
        this._edad= edad;  
    }  
  
    cumplirAños(){  
        this._edad++;  
    }  
  
    get edad(){  
        return this._edad;  
    }  
  
    set edad(años) {  
        if(años>=0) this._edad = años;  
    }  
  
    toString(){  
        return `${this._nombre} , ${this._edad} años`;  
    }  
  
}
```

Los tres puntos (...) en `...notas` se llaman **operador rest** en JavaScript. Se utilizan para recoger un número indefinido de argumentos y agruparlos en un array.

En el caso de tu función:

```
añadirNota(...notas) {  
  
    notas.forEach(element => {  
  
        this._notas.push(element);  
  
    });  
  
}
```

Crear un atributo que sea un array de objetos:

```
class Parking {  
  
    constructor(plazasTotales, costeMinuto) {  
  
        this._plazasTotales = plazasTotales;  
  
        this._costeMinuto = costeMinuto;  
  
        this._plazas = [];  
  
    }  
  
    entradaVehiculo(matricula, nPlaza, fechaHora) {  
  
        this.plazas.push({ matricula: matricula, nPlaza:  
nPlaza, fechaHora: fechaHora });  
  
    }  
  
}
```

**Posteriormente tenemos que crear instancias a partir de la clase Persona:**

```
const persona1 = new Persona ("Javier", 51);
const persona2 = new Persona ("Lidia", 28);
persona1.cumplirAños(); // persona1 ahora tendrá 52 años
console.log(persona1.edad); // Muestra por pantalla los años de persona1
console.log(persona2.toString()); // Muestra por consola el nombre y edad de persona2
persona2.edad = 30; // Modificamos la edad de la persona
```

Poder crear un objeto sin tener que declarar clase:

ejemplo:

```
this.vehiculos[nPlaza-1] = (-matricula: matricula, _fechaHora:
fechaHora);
```

Otra forma seria:

```
let nuevoVehiculo = { matricula, nPlaza, fechaHora };
this.vehiculos.push(nuevoVehiculo);
```

---

## DOM:

Los selectores son métodos que se ejecutan sobre el objeto document que nos permiten seleccionar uno o varios elementos HTML de la página web.

Los selectores más habituales en JS son:

- **getElementById:** elto cuyo id sea igual al que le pasamos como argumento

(ej: document.getElementById('nombre1') selecciona el elto HTML con id='nombre1')

- **getElementsByName:** eltos cuya etiqueta sea igual a la que le pasamos como argumento

(ej: document.getElementsByTagName('p') ; devuelve todos los párrafos de la web)

- **getElementsByName**: eltos cuyo atributo name sea igual a la que le pasamos como argumento

(ej: document.getElementsByName('color') ; devuelve todos elementos con atributo name igual a 'color')

- **getElementsByClassName**: eltos cuyo atributo class sea igual al que le pasamos como argumento

(ej: document.getElementsByClassName('principal'); devuelve todos los eltos de la página con atributo class='principal')

El primero de los selectores (getElementById) devuelve un solo elemento HTML (o null si no existe). El resto (getElementsByTagName, getElementsByClassName, getElementsByName) devuelve una colección de elementos HTML (un HTMLCollection) incluso aunque haya un solo elemento o ninguno, que habrá que recorrer con una estructura de tipo reiterativa como un for.

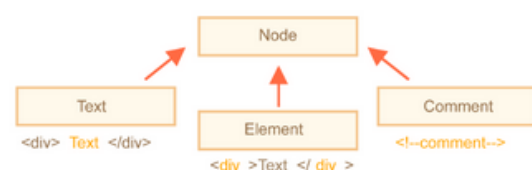
### Acceso al DOM mediante Nodes o Elements:



A tener en cuenta:

- Los Nodes puede ser de tres tipos , de tipo Node.ELEMENT\_NODE ,Node.COMMENT\_NODE y Node.TEXT\_NODE . Esta información se almacena en la propiedad **nodeType** del Node.
- Para acceder a los elementos HTML de una página es más conveniente y sencillo hacerlo mediante **Elements**

Nodes	Elements	Descripción
childNodes	children	Hijos
nextSibling	nextElementSibling	Siguiente hermano
previousSibling	previousElementSibling	Hermano previo
firstChild	firstElementChild	Primer hijo
lastChild	lastElementChild	Último hijo
parentNode	parentElement	Padre de un nodo/elemento



El DOM se puede manipular mediante **código HTML** y mediante **objetos** de tipo Element:

Usando código HTML (dato tipo string)	Usando objetos de tipo Element (dato tipo object)
innerHTML	document.createElement(etiqueta)
outerHTML	document.createTextNode(texto)
insertAdjacentHTML	appendChild(nodo)
	insertBefore(nodo1,nodo2)
	removeChild(nodo)
	replaceChild(nuevo,viejo)
	cloneNode(incluirDescendientes)

Agregar parrafo a un div:

```
<div id="divPrincipal"></div>
  <button onclick="agregarParrafo()">Agregar Parrafo</button>
<script>
  let contador = 0;

  function agregarParrafo() {
    let nParrafo = document.createElement("p");

    nParrafo.innerHTML = 'Parrafo numero ' + contador;

document.getElementById("divPrincipal").appendChild(nParrafo);
    contador++;
  }
</script>
```

Para modificar todos los parrafos debemos recorrer todos los parrafos que haya

```
let tfuente = document.getElementById("size").value;

let parrafos = document.getElementsByTagName("p");

for (let i = 0; i < parrafos.length; i++) {
  switch (tfuente) {
```

```

        case "small":
            parrafos[i].style.fontSize = "14px";
            break;
        case "medium":
            parrafos[i].style.fontSize = "19px";
            break;
        case "large":
            parrafos[i].style.fontSize = "25px";
            break;
    }
}

```

Crear un párrafo con el número del párrafo y cambiarle el color:

```

cont++;
let color = document.getElementById("txtColor").value;
let p = document.createElement('p'); // Crear un nuevo elemento <p>
let texto = document.createTextNode("Párrafo " + cont); // Crear el
texto del párrafo
p.appendChild(texto); // Añadir el texto al párrafo
p.style.color = color; // Aplicar el color al párrafo
document.getElementById('1').appendChild(p); // Añadir el párrafo al
contenedor con id="1"
cont++;
let color = document.getElementById("txtColor").value;
let p = document.createElement('p'); // Crear un nuevo elemento <p>
let texto = document.createTextNode("Párrafo " + cont); // Crear el
texto del párrafo
p.appendChild(texto); // Añadir el texto al párrafo
p.style.color = color; // Aplicar el color al párrafo
document.getElementById('1').appendChild(p); // Añadir el párrafo al
contenedor con id="1"

```

**insertAdjacentHTML:** permite insertar contenido HTML a partir de un elemento. Tiene dos parámetros: donde quiere insertar el elemento y que elemento quiere crear:

```

lista.insertAdjacentHTML("beforeend", `<li>${nombre}</li>`);
document.getElementById('nombre').value = "";

```

**createElement:** crea el objeto y escribe texto dentro usando **textContent** para poderle un valor de texto y luego lo agregara al padre usando **appendChild**

```

let li = document.createElement("li");
li.textContent = nombre;
lista.appendChild(li);
document.getElementById('nombre').value = "";

```

El **innerHTML** es una propiedad de los elementos del DOM (Document Object Model) que permite obtener o establecer el contenido HTML de un elemento. A través de esta propiedad, puedes manipular el contenido de un elemento de manera sencilla, ya sea para leer el HTML dentro de un contenedor o para cambiarlo por completo.

Puedes usar **innerHTML** para obtener el contenido HTML de un elemento, es decir, todo el texto y las etiquetas HTML que se encuentran dentro de él.

```
document.getElementById('miElemento').innerHTML = "<p>Nuevo contenido con <strong>HTML</strong></p>";
```

Establecer el contenido HTML:

- También puedes usar **innerHTML** para establecer un nuevo contenido HTML dentro de un elemento. Esto reemplaza completamente el contenido dentro de ese elemento, incluyendo el texto y las etiquetas HTML.

```
document.getElementById('miElemento').innerHTML = "<p>Nuevo contenido con <strong>HTML</strong></p>";
```

**innerHTML** es útil cuando necesitas obtener o modificar el contenido HTML completo dentro de un elemento.

**setAttribute():** añadir atributo a un elemento:

```
document.getElementById('tabla1').setAttribute("border", "5");
```

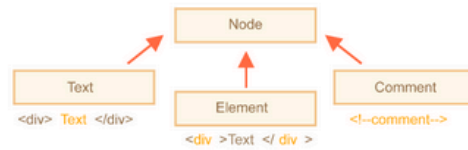
**removeAttribute("border"):** elimina un atributo de un elemento:

```
document.getElementById('tabla1').removeAttribute("border");
```

## Acceso al DOM mediante Nodes o Elements:



Nodes	Elements	Descripción
childNodes	children	Hijos
nextSibling	nextElementSibling	Siguiente hermano
previousSibling	previousElementSibling	Hermano previo
firstChild	firstElementChild	Primer hijo
lastChild	lastElementChild	Último hijo
parentNode	parentElement	Padre de un nodo/elemento



A tener en cuenta:

- Los Nodes puede ser de tres tipos , de tipo `Node.ELEMENT_NODE` ,`Node.COMMENT_NODE` y `Node.TEXT_NODE` . Esta información se almacena en la propiedad **nodeType** del Node.
- Para acceder a los elementos HTML de una página es más conveniente y sencillo hacerlo mediante **Elements**

## EVENTOS:

manejador de eventos:

```
<h1>Manejador eventos compartido (Ver consola depuración)</h1>
<button id="boton1">Boton 1</button>
<button id="boton2">Boton 2</button>
Div 1
<div id="div1"></div>
Div 2
<div id="div2"></div>

<script>
function manejadorEventos(event) {
    console.log("Objeto que generó el evento:", event.target);
    console.log("Tipo de evento:", event.type);
}

document.getElementById('boton1').addEventListener('click',
manejadorEventos);
document.getElementById('boton2').addEventListener('click',
manejadorEventos);
document.getElementById('div1').addEventListener('mouseenter',
manejadorEventos);
document.getElementById('div2').addEventListener('mouseenter',
manejadorEventos);
```



### Obtener valor de input mientras se escribe:

```
<h1>Eventos</h1>

Nombre <input type="text" id="nombre">
<p id="pNombre"></p>

<script>
  // Escucha el evento 'input' en el cuadro de texto
  document.getElementById('nombre').addEventListener("input",
function () {

    let nombre = document.getElementById('nombre').value;
    document.getElementById('pNombre').textContent = "Hola " +
nombre;

  });
</script>
```

### mousedown/mouseup: Cuando se presiona y se suelta un elemento:

```
<!-- Elaborar un programa que muestre un div de color rojo.
  Cuando se presione cambiar el color a amarillo y luego de soltar el
botón
  del mouse volver a pintar de rojo. -->

<div id="div1"></div>
<p>Lorem ipsum dolor sit amet consectetur adipisicing elit.
Excepturi, doloribus illo officia animi, natus cumque aut provident
veritatis et hic, quibusdam nobis! Illo tenetur, consequatur harum
debitis laborum rerum
architecto.</p>

<script>
  document.getElementById('div1').addEventListener('mousedown',
amarillo);

document.getElementById('recuadro2').addEventListener('mouseup', rojo);

  function amarillo() {
    document.getElementById('div1').style.backgroundColor =
'yellow';
  }

  function rojo() {
```

```
document.getElementById('div1').style.backgroundColor =
'red';
}
</script>
```

**mouseover/mouseout:** Cuando se pasa o no el raton por arriba.

---

## TESTEOS DE FUNCIONES:

not: Invierta el comparador siguiendo esta expectativa

```
expect(thing).not.toEqual(realThing);
```

toEqual:

```
describe("Testeo de que todos los elementos sean numeros:", () => {
  it("Todos los elementos deben ser numeros", () => {
    let resultado = bonoloto();
    resultado.forEach(function (num) {
      expect(typeof num).toEqual("number");
    });
  });
});
```

toBeInstanceOf:

```
describe("Testeo de que devuelve un array", () => {
  it("Comprobar que la funcion bonoloto() devuelve un array:", () => {
    {
      expect(bonoloto()).toBeInstanceOf(Array);
    }
  });
});
```

toThrowError:

```
it("Debería lanzar un error cuando algún elemento no es un número", ()
=> {
  expect(() => promedio([7, "hola", 3])).toThrowError(
    "El elemento debe ser un Number"
  );
});
```

toBeNaN: deberia devolver un nan

```
expect(resultado).not.toBeNaN(); // No debería devolver NaN
```

toBeCloseTo: se indica un valor esperado y una precision:

```
expect(resultado).toBeCloseTo(5.33, 2); // Resultado esperado: (4 + 7 + 5) / 3 = 5.33
```

toHaveLength: tener un numero de elementos o tamaño:

```
describe("Testeo de que el array tiene 6 elementos", () => {  
  it("Deberia tener 6 elementos:", () => {  
    expect(bonoloto()).toHaveLength(6);  
  });  
});
```

toBeGreaterThanOrEqual(expected):

```
expect(array[i]).toBeGreaterThanOrEqual(1); // Verificar que es >= 1  
expect(array[i]).toBeLessThanOrEqual(49); // Verificar que es <= 49
```

```
describe("Testeo de ejercicio 1", () => {  
  describe("Testeo de la funcion areaPiramide()", () => {  
    const datos = [  
      { lado: 6.8, altura: 9, areaEsperada: 177.083 },  
      { lado: 7.1, altura: 9.4, areaEsperada: 193.092 },  
      { lado: 7.4, altura: 9.8, areaEsperada: 209.793 },  
    ];  
  
    for (let i = 0; i < datos.length; i++) {  
      it(`Comprobar si la piramide con lado ${datos[i].lado} y altura: ${datos[i].altura} tiene un area de ${datos[i].areaEsperada}`, () => {  
        expect(areaPiramide(datos[i].lado, datos[i].altura)).toEqual(  
          datos[i].areaEsperada  
        );  
      });  
    }  
  });  
  
  describe("Comprobar si la funcion areaPiramide() devuelve un number",  
    () => {  
    it("Debe devolver un número", () => {  
      const resultado = areaPiramide(5, 7);  
      expect(resultado).toBeInstanceOf(Number);  
    });  
  });  
  
  describe("Comprobar si la funcion areaPiramide() lanza un error  
cuando el lado es negativo", () => {
```

```

    it("Debería lanzar un error cuando algún elemento es negativo", ()
=> {
        expect(() => areaPiramide(-3, 6)).toThrowError(
            "Los parámetros de entrada deben tener valores positivos"
        );
    });
});
describe("Comprobar si la funcion areaPiramide() lanza un error
cuando la altura es negativa", () => {
    it("Debería lanzar un error cuando algún elemento es negativo", ()
=> {
        expect(() => areaPiramide(3, -6)).toThrowError(
            "Los parámetros de entrada deben tener valores positivos"
        );
    });
});
});

```

Todas las opciones disponibles:

<b>1. not</b>	<b>2</b>
<b>2. toBeCloseTo(expected, precision)</b>	<b>2</b>
<b>3. toBeDefined()</b>	<b>3</b>
<b>4. toBeFalse()</b>	<b>3</b>
<b>5. toBeTrue()</b>	<b>3</b>
<b>6. toBeGreaterThan(expected)</b>	<b>3</b>
<b>7. toBeGreaterThanOrEqual(expected)</b>	<b>4</b>
<b>8. toBeInstanceOf(expected)</b>	<b>4</b>
<b>9. toBeLessThan(expected)</b>	<b>5</b>
<b>10. toBeLessThanOrEqual(expected)</b>	<b>5</b>
<b>11. toBeNaN()</b>	<b>6</b>
<b>12. toBeNull()</b>	<b>6</b>
<b>13. toBeUndefined()</b>	<b>6</b>
<b>14. toContain(expected)</b>	<b>6</b>
<b>15. toEqual(expected)</b>	<b>7</b>
<b>16. toHaveSize(expected)</b>	<b>7</b>
<b>17. toThrowError(message)</b>	

## EXPRESIONES REGULARES:

**^:** comienzo de expresión

**\$:** fin de expresion

[ ]: expresión de rango

{ }: número de veces que se debe hacer/repetir

/s: detecta un espacio

+: el elemento puede aparecer una o varias veces

?: puede estar o no estar

\d [0-9]

\D [^0-9]

\w [a-zA-Z0-9\_]

\W [^a-zA-Z0-9\_]

\s espacio en blanco

| alternancia (OR)

i: especifica que la búsqueda se realiza sin diferenciar entre mayúsculas y minúsculas.

g: indica que los caracteres a buscar pueden ser encontrados varias veces en la cadena de caracteres.

m: el modificador m realiza coincidencia de patrones en el modo multilínea. En este modo, si la cadena a buscar contiene varias líneas, el metacaracter ^ y \$ deben coincidir con el inicio y final de una línea, además de que coincida con el comienzo y el final de una cadena.

search: busca la primer ocurrencia del string según la expresión que le hemos pasado como parámetro

replace: realiza una búsqueda y reemplazo

split: tiene por objetivo subdividir un string y retornar un vector con dichas partes

match: Tiene como parámetro una expresión regular y retorna un vector con todos los string que cumplen el patrón especificado

Número positivo de 3 dígitos:

```
let valor = prompt('Ingrese un número entero positivo de 3 dígitos');
let patron = new RegExp('^[0-9]{3}$');
if (patron.test(valor))
    alert('Se ingresó un valor entero positivo de 3 dígitos');
else
    alert('No se ingresó un valor entero positivo de 3 dígitos');
```

Cualquier carácter alfabético ya sea mayuscula o minuscula:

```
let patron = /^[a-zA-Z]/;
```

Comprobar que finaliza en cualquier vocal:

```
let patron2 = /[aeiouAEIOUáéíóú]$/;
```

Comprobar que una cadena tiene como minimo 1 vocal:

```
let palabra4 = prompt('Ingrese una palabra');
```

```

let patron4 = /[aeiouAEIOUáéíóú]/;
if (patron4.test(palabra4))
    alert('La palabra tiene al menos una vocal');
else
    alert('La palabra no tiene vocales');

```

Comprobar que una cadena tiene un caracter extraño:

```

let oracion = prompt('Ingrese una oracion');
let patron5 = /[^\$. * + ? = ! : | \ / ( ) [ ] { } ]/;
if (patron5.test(oracion))
    alert('La oración tiene al menos un carácter ^ $. * + ? = ! : | \ / ( ) [ ] { }');
else
    alert('La oración no tiene ningún carácter ^ $. * + ? = ! : | \ / ( ) [ ] { }');

```

Matrícula que lleva dos letras tres números y dos letras:

```

let patente = prompt('Ingrese una patente');
let patron2 = /^[a-z]{2}[0-9]{3}[a-z]{2}$/;
if (patron2.test(patente))
    alert('La patente ingresada es correcta');
else
    alert('La patente ingresada no es correcta');

```

Palabra con al menos una vocal:

```

let palabra = prompt('Ingrese una palabra con al menos una vocal:');
let patron = /[aeiou]+/;

```

Comprobar si lleva un + o un - delante:

```

let valor = prompt('Ingrese un valor numérico entero, puede anteceder el +/-:');
let patron2 = /^[+\-]?[0-9]+$/;

```

Numero de 5 digitos:

```

let valor = prompt('Ingrese un valor numérico de 5 dígitos');
let patron = /^\d{5}$/;

```

Comprobar si una palabra tiene al menos 4 caracteres:

```

let oracion = prompt('Ingrese una oracion');
let patron2 = /\s?\w{4}\s?/;

```

Comprobar que un tipo y una factura tengan a,b o c y que tiene 1 o mas digitos:

```

let factura = prompt('Ingrese tipo y nro de factura:');

```

```
let patron = /^[a|b|c]\d+$/;
```

Comprobar si existe una cadena:

```
let oracion = 'saliendo de la casa que esta en la montaña';
    let patron1 = /la/;
    let vector1 = patron1.exec(oracion);
    document.write(vector1.index);
    document.write('<br>');
    document.write(vector1.input);
    document.write('<br>');
    let patron2 = /lax/;
    let vector2 = patron2.exec(oracion);
    document.write(vector2);
```

Comprobar que contiene una cadena sin tener en cuenta mayúsculas y minúsculas:

```
let palabra = 'Administracion';
    let patron = /adm/i;
    if (patron.test(palabra))
        document.write('La palabra ' + palabra + ' contiene la
cadena adm sin tener en cuenta mayúsculas/minúsculas');
```

Metodo search:

```
let oracion = 'La noche está muy oscura y estrellada.';
    if (oracion.search(/estrella/) != -1)
        alert('el string estrella está contenido en la variable
oracion');
```


Metodo replace:

```
let oracion2 = 'uno dos tres uno dos tres uno dos tres';
    let cadena = oracion2.replace(/uno/g, '1');
    alert(cadena); //1 dos tres 1 dos tres 1 dos tres
```

Metodo match:

```
let cadena2 = '(2,3)-(4,4)-(9,3)';
    let vec = cadena2.split(/-/);
    for (let x = 0; x < vec.length; x++) {
        document.write(vec[x] + '<br>');
    }
```

javascript

 Copiar código

```
/^[+-]?[0-9]+$/
```

## Explicación:

1. `^` y `$`:

- Aseguran que toda la cadena cumpla con el formato definido (sin caracteres adicionales antes o después).

2. `[+-]?`:

- `[+-]` permite que el número sea precedido por un `+` o un `-`.
- `?` indica que esta parte es opcional (puede estar o no).

3. `[0-9]+`:

- Obliga a que la cadena contenga uno o más dígitos.


4. Resultado:

- Valida números positivos, negativos y no firmados.



## 1. Nuevo patrón de expresión regular:

javascript

 Copiar código



```
/^(?:[01]\d|2[0-3]):[0-5]\d:[0-5]\d$/
```

- `[01]\d|2[0-3]` :
  - Valida horas de `00` a `23`.
  - `[01]\d` cubre horas de `00` a `19`.
  - `2[0-3]` cubre horas de `20` a `23`.
- `[0-5]\d :`
  - Valida minutos y segundos de `00` a `59`.
- `::`
  - Asegura que los dos puntos sean literales entre horas, minutos y segundos.
- `$` y `^` :
  - Se aseguran de que toda la cadena sea un formato válido, sin caracteres adicionales.

EXPRESION grados minutos y segundos:

## Desglose de la expresión regular

regex

 Copiar  Editar

```
^((\d{0-8}\d)\s(\d{0-5}\d)'\s(\d{0-5}\d)''\s[NS]|90\s0{1,2}'\s0{1,2}''\s[NS])$
```

### Explicación por partes

- ^** → Indica el inicio de la cadena.
- ((\d{0-8}\d)\s(\d{0-5}\d)'\s(\d{0-5}\d)''\s[NS]|90\s0{1,2}'\s0{1,2}''\s[NS])**
  - (\d{0-8}\d)** → Un número entre 0 y 89 (primer dígito puede ser 0-8 y el segundo 0-9). Representa **los grados** (°).
  - \s** → Literalmente " " (grados) seguido de un espacio.
  - (\d{0-5}\d)'** → Un número entre 0 y 59 (minutos).
  - \s** → Espacio.
  - (\d{0-5}\d)''** → Un número entre 0 y 59 (segundos).
  - \s[NS]** → Espacio seguido de "N" o "S" (indica si la latitud es Norte o Sur).
- Alternativa: |90\s0{1,2}'\s0{1,2}''\s[NS]**
  - Permite la latitud máxima de **90° 00' 00" N** o **90° 00' 00" S**.
  - 90\s0{1,2}'\s0{1,2}''\s[NS]** → Se asegura de que, si los grados son 90, los minutos y segundos sean **00** o **0**.
- \$** → Indica el final de la cadena.