

# Anotaciones Javascript

## Javascript Types

Son los diferentes tipos de contenidos o valores que javascript puede recibir y leer, son la unidad más básica que existe en el motor del código, también se le llaman *primitive types*:

- **Number**(Los números que normalmente se usan)
- **String**(Lo que conocemos como texto, javascript usa caracteres especiales para texto)
- **Boolean**(Lógica comparativa a lo que decimos true(verdadero) o false(falso), también el código lo toma como 1 ó 0, como algo con valor o algo sin valor o vacío).
- **Undefined**(es la manera de javascript de avisar que un archivo o variable no tiene un valor definido o está vacío cuando no debería)
- **Null**(Es lo mismo que undefined pero es usado de manera voluntaria por parte del programador para asignar como vacío o nulo deliberadamente un objeto, así le hace saber al motor de javascript y al programador que analice después el código que ese valor(Null) fué designado por el programador por alguna razón).
- **Object**(significa objeto, es una colección o conjunto de propiedades o *properties*, en la forma de nombres y valores, así que es un conjunto que guarda una lista de nombres con sus propiedades.)

## Data Structures / Estructura de datos

- Arrays
- Objects

## Arrays

Los arrays son listas en javascript que pueden guardar una serie de elementos de diferente tipo, aunque es recomendado guardar solo elementos del mismo tipo en una sola lista, generalmente mezclar elementos de diferente tipo en una sola lista puede generar conflictos en la lógica del programa. Los arrays no entran en la categoría de javascript types por la razón de que para el motor de javascript cada item sigue siendo un número, de ahí que se pueda indexar o acceder a un item de un array usando números ejm:

```
array = ['item0', 'item1', 'item2', 'item3']
```

para acceder a uno de ellos solo se escribe array[1]; y se accede al 'item1'.

### Javascript Array Methods / Métodos

Algunos métodos a tener en cuenta para agregar, remover, concatenar, cambiar, etc, los arrays:

- *splice()*, se escribe primero el nombre de la variable que contiene el array, más un punto seguido, e inmediatamente se escribe '*splice*' seguido de unos paréntesis en donde se escribe el código para remover o agregar uno o varios elementos de la lista o array, dentro de los paréntesis se escribe en orden de izquierda a derecha:  
`array.splice( 4, 0, 'taste');`  
el primer carácter tiene que ser un número indica en que lugar se quiere agregar o remover contando desde 0 en adelante en la posición de los items de la lista, el siguiente carácter es otro número que indicará cuántos items de la lista se borrarán, si no se desea borrar ningún item de la lista, simplemente escribir 0, después del segundo número se puede escribir entre paréntesis el item que se quiere agregar en la lista, el código quedaría así:

```
let array = ['Banana', 'Apples', 'Oranges', 'Blueberries'];
```

```
array.splice( 4, 0, 'taste');
```

```
console.log(array);
```

- *sort()*; Se usa para ordenar una lista o array para que sea de manera alfabética o numérica de forma ascendente de menos a más (ejm. [1,2,3,4,etc][a,b,c,d,f,etc]).
- *array.reverse()*; se usa para ordenar al revés una lista o array de más a menos.
- *array.sort((a, b) => a > b)*; se usa para reordenar de la forma deseada una lista o array, se escribe el nombre de la variable que contiene el array luego un punto, luego el nombre sort más el resto entre paréntesis, las letras a y b dentro del primer paréntesis son para indicar el orden primero en que se desea ordenar los items de la lista, y las letras a > b, se usan para indicar el orden final o el que se verá reflejado en el resultado, el signo mayor que(>) es para indicar hacia donde va el orden de la lista, a > b indicaría que la lista va de manera ascendente de primero hacia abajo desde la primera letra del alfabeto hasta la última, indicando un orden progresivo dependiendo del tipo de items en la lista esto

significa un orden ascendente como de 1 a 100, si se cambia de  $b < a$  significa que queremos que la lista esté ordenada desde la última letra del alfabeto o ítem de la lista hasta la primera.

## Ejercicios usando **array properties**

Usar la siguiente variable:

```
const array = ['Banana', 'Apples', 'Oranges', 'Bluberries'];
```

- Remover el ítem Banana del array.
- Ordenar el array usando sort.
- Poner 'Kiwi' al final del array.
- Remover 'Apples' del array.
- Ordenar al revés el array.(No en orden alfabético, solo al revés ejm: ['a', 'c', 'b'] al revés sería ['b', 'c', 'a'])  
Debería dar como resultado al final:  
['Kiwi', 'Oranges', 'Blueberries']
- Usando el siguiente array acceder a 'Oranges':

```
const array2 = ["Banana", ["Apples", ["Oranges"], "Blueberries"]];
```

Resultado:

```
const array = ['Banana', 'Apples', 'Oranges', 'Bluberries'];
const array2 = ["Banana", ["Apples", ["Oranges"], "Blueberries"]];

array.splice(0, 1);
array.sort((a, b) => a > b);
array.splice(3, 0, 'Kiwi');
array.splice(0, 1);
array.reverse();

console.log(array); // -> Array(3) [ "Kiwi", "Oranges", "Bluberries" ]
console.log(array2[1][1]); // -> Array [ "Oranges" ]
```

## Objects

Los objects(objetos), son parecidos a los arrays, pues almacenan una lista de varios ítems, la diferencia es que en objects se pueden guardar una propiedad y un valor por cada ítem a diferencia de los arrays que cada ítem de la lista se guarda con el valor de un número y no se le puede asignar a cada ítem un nombre en específico. ejm:

arrays:

```
const list = ['apple', 'banana', 'orange']; // -> solo guarda valores por ítem.
```

objects:

```
const user = {
  name: 'Jhon', // -> guarda propiedades(name) y valores para cada propiedad('Jhon')
  age: 34,
  hobby: 'soccer',
};
```

```
user.favouriteFood = 'spinach'; // -> agrega un nuevo ítem llamado favouriteFood y su valor es de 'spinach' en e
```

```
console.log(user); // -> da como resultado la lista anterior más el nuevo ítem agregado = { name: "Jhon", age:
```

Los objects se pueden guardar dentro de arrays y los arrays se pueden guardar dentro de objects también. ejm:

- *Arrays en objects:*

```
const user = {
  name: 'Jhon',
```

```
    certificates: ['computer science', 'engineer', 'programming software']
  }
```

Si se requiere llamar un contenido de un objeto dentro de una lista o array se debe hacer de esta forma, teniendo en cuenta el array ya hecho antes de este párrafo:

```
user.certificates[1]
```

```
console.log(user.certificates[1]) // -> ['computer science', 'engineer', 'programming software']
```

- *Objects en arrays*

```
const certificateUsers = [ {
  userName = 'Jhon',
  certificateArea = 'computerscience',
}
{
  userName = 'andrew',
  certificateArea = 'computerscience',
}
{
  userName = 'Jack',
  certificateArea = 'computerscience',
}
]
```

Si se requiere llamar un contenido de un objeto dentro de una lista o array se debe hacer de esta forma, teniendo en cuenta el array ya hecho antes de este párrafo:

```
certificateUsers[0].certificateArea;
```

```
console.log(certificateUsers[0].certificateArea;) // -> computerscience
```

También se pueden agregar functions dentro de objects, a esto se le llama **Methods**, los métodos son funciones dentro de un object, ya hemos visto varios métodos como console.log(), list.sort(),etc, para que funcione los métodos correctamente a la hora de llamarlos se debe poner un paréntesis al final del nombre de la función, en el caso de console, la palabra console se refiere al objeto y la palabra log se refiere a una función que está dentro de console.ejm:

```
const user = {
  name: 'Jhon',
  shout: function() {
    console.log('gritar');
  }
}
```

```
user.shout() // -> 'gritar'
```

Tanto los arrays como los objects pueden ser vacíos. ejm:

```
const objectVacio = {};
const items1 = [];
```

```
console.log(objectVacio); // -> object {}
console.log(items1); // -> array []
```

## Facebook exercise

- Hacer un facebook mediante console.log y prompts, tendrá que pedir nombre de usuario y contraseña y validar que sean las que están guardadas en las variables escritas y permitir ver lo que otros usuarios han escrito en facebook y si no escribe bien la contraseña o usuario mandar un mensaje de error que diga que metió datos incorrectos:

```
const database = [
  {
    username: 'andrew',
    password: 'supersecret'
  }
]
```

```
const newsfeed = [
  {
    username: 'Bobby',
    timeline: 'so tired from all that learning'
  },
  {
    username: 'Sally',
    timeline: 'Javascript is so cool!'
  }
]
```

```

];

const userNamePrompt = prompt("What's your username");
const passwordPrompt = prompt("What's your password");

function signIn(user, passw) {
    if ( user === database[0].username &&
        passw === database[0].password) {
        console.log(newsfeed);
    }else {
        alert('Sorry, wrong username and password');
    }
}

signIn(userNamePrompt, passwordPrompt);

```

En el anterior ejercicio se utilizó **&&**, y las funciones condicionales *if* y *else* estos métodos hacen parte de los **Javascript Logical Operators** y **Javascript Conditionals**.

## Javascript Logical Operators / Javascript conditionals

los **Javascript Logicals Operators**(Operadores lógicos), son usados para comparar un dato o variable o valor de otro, y analizar si son iguales o diferentes, generalmente se usan junto a los **Conditionals** para permitir o no permitir una acción y en caso de que no pase la comparativa, dar como una opción otra acción.

**&&** Significa *y*, sólo da true como resultado si los dos valores o variables son iguales, de lo contrario así uno sea cierto y el otro no, no va a ejecutar la función.

**||** Significa *o* y da la opción de que si al menos uno de los dos valores o variables es cierto puede permitir la función a realizar.

**!** Significa lo opuesto o *false*, en algunas ocasiones se necesita revertir el resultado de una variable a su valor lógico opuesto, si por ejemplo una variable da como resultado true, si le agregamos **!** antes de la variable dará como resultado false , pasa lo mismo al revés.

Los **Conditionals** se usan generalmente junto a los Logical Operators, significan literalmente **if**(si tal cosa es cierta), y **else**(caso contrario...)

## Overview

- **Declaración de functions / Function declarations**

```

function newFunction() {

}

```

- **Function expression / Expresión de functions**

```

const newFunction = function() {

}

```

- **expressions / expresiones**

```

1+3;
const a = 2;
return true;

```

- **calling or ejecuting a function / LLamar o ejecutar una Function**

```

alert();
newFunction(param1, param2);

```

- **assign a variable / asignar una variable**

```

const a = 1;

```

- **function vs method**

```

estoesunaFunction();
obj.estoesunMethod();

```

```

function estoesunaFunction() {

}

```

```
const obj = {
  estoEsUnMethod: function() {

  }
}
```

## Loops en Javascript

- *for*
- *while*
- *do*
- *forEach*

Los loops son una forma de repetir un proceso en el código que se requiera repetir, se puede controlar las veces y que partes del código se repiten:

el siguiente ejercicio es un **todo list / lista de tareas** donde se repetira la variable **i** las veces que sean necesarias hasta terminar de nombrar cada ítem de la lista 'todos', la variable **i** tendrá un valor de cero, y mediante el código **i++** se le sumara 1 cada vez que se repita el código hasta que la variable **i** tenga el valor de 5 el cuál es el mismo valor que la cantidad de ítems en la lista **todo**, el cual en ese momento se detendra el proceso.

```
const todos = [
  'clean room',
  'brush teeths',
  'exercise',
  'study javascript',
  'eat healthy'
]

for (let i=0; i < todos.length; i++) {
  console.log(todos[i]);
} /*
clean room
brush teeths
exercise
study javascript
eat healthy
*/
```

El último bloque de código tiene tres declaraciones entre paréntesis, primero se define la variable **i** dándole el valor de cero, seguido se está haciendo una comparativa entre la variable **i** con valor definido cero y el **length** (largo o cantidad) de la variable **todos** (el cual tiene 5 ítems por lo tanto su **length** es de valor 5), luego seguido esta la variable **i** con dos signos de suma, indicando una suma de valor de 1 a la variable **i** aumentando su valor.

El proceso entonces sería:

- definir la variable **i** con el valor de cero
- si **i** es menor que la cantidad de ítems de la variable **todos** entonces se ejecuta el proceso interno que está entre corchetes (`{console.log...imprime el ítem de la variable todos que tenga el mismo número o valor que tiene en ese momento la variable i}`), luego se ejecuta el último proceso o código de **i++** que suma el valor de 1 a el valor que tiene la variable **i**, se repite el proceso completo hasta que **i** llegue a sumar el valor de 5 el cual es el mismo valor que la cantidad de ítems de la variable **todos** y de esa manera el `console.log(todos[i])` va a imprimir los ítems de la variable **todos** uno por uno.

También se pueden usar **while** y **do while** para hacer loops:

- **while:**

```
const counterOne = 10;
while(counterOne > 0) {
  console.log(counterOne);
  counterOne--;
}
```

La palabra **while** significa **mientras**, mientras **counterOne** sea mayor que 0, entonces ejecutar el siguiente bloque de código (el que está entre corchetes `{}`), el bloque de código que está entre corchetes tiene `console.log counterOne` esto significa que va a aparecer en la consola del sitio web el valor de la variable **counterOne**, luego hace la operación de **counterOne--** que significa que al valor actual de **counterOne** se le resta una unidad o 1, como **counterOne** tiene un valor de 10 al restarle uno queda con valor de 9, y 9 sigue siendo mayor que 0, por lo tanto todo el proceso de **while** se repite hasta que el valor sea cero.

- **do while:**

```
const counterTwo = 10
do {
  console.log(counterTwo);
  counterTwo--;
} while (counterTwo > 0);
```

El orden en **do while** es algo diferente, comienza con **do** significa **hacer**, hacer lo que esta entre corchetes del primer bloque de código en el cuál se imprime en consola el valor actual de counterTwo y luego se pasa a restar un valor de 2 a el valor actual de counterTwo, y luego de ejecutarlo, pasa a ejecutar el segundo bloque de código entre corchetes en el cual está la condicional **mientras**, mientras counterTwo sea mayor que 0 se repite todo el proceso.

- **For Each**

Este método Fué agregado en el ECMAScript5 (versión de javascript actualizado):

```
const todos = [
  'clean room',
  'brush teeths',
  'excercise',
  'study javascript',
  'eat healthy'
]

todos.forEach(function(a, b) {
  console.log(a, b);
})

/* Da como resultado:
clean room 0      app.js:105:13
brush teeths 1    app.js:105:13
excercise 2       app.js:105:13
study javascript 3 app.js:105:13
eat healthy 4     app.js:105:13*/

/* dato importante, cuando se
imprime algo por consola, en la parte derecha se muestra
el número de la línea de código en la que se escribió el código
que indica u ordena imprimir o ejecutar el código completo, en este caso en
la consola aparecerá en la parte derecha app.js:105:13, indicando
la línea en donde esta el código que ejecuta, esto es de ayuda
en casos cuando salen errores y queremos saber en dónde podemos
analizarlos. */
```

**forEach** toma cada *item* de una variable, array, etc y escribimos que hacemos con esos *items/objetos*, pero primero guardamos cada item que se recibe, como *argumentos* de una *función/function* (la cual en el código del ejemplo está sin definir, pues no se le ha puesto un nombre a dicha función), del array en una variable en este caso la variable '*a*' (o cualquiera que decida colocar), luego se indica que se imprima en la consola una variable '*a*', de ese momento en adelante la variable '*a*', tiene guardado cada item del array, por lo tanto en realidad se esta pidiendo que imprima el valor de la variable '*a*' que en este caso es un item del array, **forEach** tiene la tarea de tomar cada item del array seleccionado por lo tanto el proceso se repetirá las veces que sean necesarias hasta completar todo el bloque de código indicado en todos los items que contiene el array.

si queremos acceder, o seleccionar el index o número de cada item del array todos, en **forEach** simplemente agregamos una nueva variable o función, en este caso es 'b', el código toma la variable o función que esté de segundo dentro de parentesis en function() y la asigna como una función para indexar los items del array.

También se puede separar la definición de la función y aparte llamar la función con **forEach**:

```
const todos = [
  'clean room',
  'brush teeths',
  'excercise',
  'study javascript',
  'eat healthy'
]
const todosImportant = [
  'clean room!',
  'brush teeths!',
  'excercise!',
  'study javascript!',
  'eat healthy!'
]

function logTodos(todoitem, i) {
  console.log(todoitem, i);
}

todos.forEach(logTodos); // -> da como resultado todos los items de la variable todos
todosImportant.forEach(logTodos); // -> da como resultado todos los items de la variable todosImportant
```

Como se puede ver en el ejercicio anterior uno de los beneficios de separar la function de la declaración de **forEach**, es que podemos usar dicha function por aparte para ejecutar otro bloque de código en base a otros objetos, o arrays, de esa manera estaríamos ahorrando escribir código innecesario o repetido y aumentando la facilidad de lectura del código y optimizando la velocidad con que escribimos el código.

Para saber si una función está disponible o soportada en un explorador de internet o browser en específico (opera, safari, mozilla firefox, brave, google chrome, etc), se puede abrir una página cualquiera del navegador y abrir el modo consola, escribir en caso de que sea una función: [].forEach y darle

Enter, saldrá resultados como este: `f.forEach() { [native code] }` dando a entender que ese browser sí soporta esa función.

## Facebook exercise 2

En el siguiente ejemplo/ejercicio se usará uno de los **Loops** llamado **for**, para poder comprobar si el usuario que escribe sus datos de inicio de sesión está registrado en la base de datos de los usuarios registrados previamente:

```
const database = [
  {
    username: 'andrew',
    password: 'supersecret'
  },
  {
    username: 'sally',
    password: 'sally123'
  },
  {
    username: 'ingrid',
    password: 'ingrid123'
  }
]

const newsfeed = [
  {
    username: 'Bobby',
    timeline: 'so tired from all that learning'
  },
  {
    username: 'Sally',
    timeline: 'Javascript is so cool!'
  },
  {
    username: 'Mitch',
    timeline: 'Javascript is preetty cool!'
  }
];

function isValid(username, password) {
  for (let i=0; i < database.length; i++) {
    if(database[i].username === username &&
      database[i].password === password) {
      return true;
    }
  }
  return false;
}

function signIn(username, password) {
  if(isValid(username, password)) {
    console.log(newsfeed);
  } else {
    alert('Sorry, wrong username and password');
  }
}

const userNamePrompt = prompt("What's your username");
const passwordPrompt = prompt("What's your password");

signIn(userNamePrompt, passwordPrompt);
```

### Algunos puntos a tener en cuenta:

Para poder ejecutar todo el código se debe hacer un llamado, se escribe el nombre de la función principal que activa todo el código por decirlo así (en este caso `signIn(userNamePrompt, passwordPrompt)`); y como se puede notar, para poder ejecutar una función siempre debe escribirse al final dos paréntesis, `signIn` tiene entre paréntesis dos variables **userNamePrompt** y **passwordPrompt**, dichas variables guardan como valor una función la cual es `prompt("what's your username")`; así que al llamar la función `signIn(userNamePrompt, passwordPrompt)`; se está en primer lugar ejecutando la función `signIn`, la cual tiene dos argumentos dentro de su función (`username, password`) esos dos argumentos son 'definidos' de manera específica en el momento de llamarse la función `signIn` pues entre paréntesis se modifica `username` y `password` por las variables `userNamePrompt` y `passwordPrompt`, que contienen una función `prompt` cada una, por lo tanto el proceso en orden sería así:

- Se ejecuta la función `signIn` con argumentos entre paréntesis `userNamePrompt` y `passwordPrompt`.
- El código busca donde se encuentran los argumentos `usernamePrompt` y `passwordPrompt`, y los encuentra como variables que contienen una función interna cada uno.
- El código las ejecuta una por una en orden.

- La página solicita que el usuario escriba su username y su password.
- mientras solicita username de primero lo que el usuario escriba el código lo guarda como username y lo mete dentro de username en la function `signIn`.
- El código pasa a verificar o ejecutar de primero username en el bloque de código de la function `signIn`
- La function `signIn` pide hacer una verificación *if/else*, la cual verifica si la operación de la function `isUserValid` es `true`.
- El código pasa a ejecutar la function ***isUserValid*** la cual toma lo que el usuario escribió como **username** y lo analiza dentro de un ***Loop*** tipo ***for***, si resulta cierto, regresa el valor de *true*, si no lo es, regresa el valor de *false*.
- El código vuelve a function ***signIn*** y valida el `if(isUserValid(username))`, si es *true* entonces imprime el contenido de la variable **newsfeed**, si es *false* ejecuta una **alert** con el mensaje 'Sorry, wrong username and password'.
- Una vez que hace el proceso con **username**, vuelve y lo repite con **password**, le pide al usuario que escriba su password, guarda el contenido que el usuario escribió en password y valida si ***isUserValid*** regresa como *true* o *false*.
- Si regresa como *true*, imprime el contenido de la variable **newsfeed** en la consola de la página, si es *false* ejecuta la **alert**.

## Document Object Model

Javascript se usa principalmente para cambiar el contenido de un sitio web de maneras que HTML5, y CSS3 no podrían. Para poder modificar el contenido y estructura de un sitio web javascript se vale de un método llamado Document Object Model. Javascript puede:

- Cambiar todos los elementos de HTML en un sitio web.
- Cambiar todos los atributos de HTML en una página
- Cambiar todos los estilos de CSS en la página
- Remover todos los elementos existentes y atributos.
- Agregar nuevos elementos HTML y atributos
- Javascript puede reaccionar a todos los HTML eventos existentes en la página
- Puede crear nuevos eventos en la página.

La diferencia entre HTML, CSS y Javascript, es que mientras HTML y CSS pueden modificar elementos de la página, no puede hacerlo una vez la página esté cargada en el sitio web, su modificación se limita a realizarse antes de ser cargada y ejecutada en el navegador web, en cambio Javascript tiene la capacidad de cambiar o modificar todos los elementos de la página web ***mientras*** la página web esta en uso, funcionando y el usuario está visitándola e interactuando con ella.

En el momento que un sitio web carga, se crea el DOM(Document Object Model), el cuál permite modificar, agregar o eliminar elementos HTML y CSS dentro de la página web, ejm:

Si se carga cualquier página web, abrir la consola de desarrollo en la misma página web, y escribir ***document***, darle enter para ejecuta el comando y aparecerá un objeto llamado documento ese objeto es el DOM, si escribimos en la consola `__document.write('hello')_` y le damos ejecutar, la página web inmediatamente reescribirá solo lo que escribimos en la consola, mediante document reescribimos todo el contenido de la página web, sin embargo como es un sitio web ya construido por otros programadores, el cambio o modificación que hagamos en esa página será temporal, al volver a cargar la página se reestablecerá el contenido inicial de la página.

Para hacer esto posible, cada navegador web tiene en sus servidores(donde se guarda la información que se dispone a los usuarios), un motor que compila y ejecuta el código javascript que nosotros escribamos o enviemos mediante los archivos que hagamos y subamos al sitio web, por ejemplo, google tiene V8 engine, firefox tiene SpiderMonkey y Safari tiene Nitro.

`document.write('Hello');` con este código estamos diciendo a el motor del navegador que escojamos para cargar la página web que el objeto document(toda la página agrupada en un objeto 'document'), realice un método o función el cual se llama **write**, el cual ordena al motor del navegador web que escriba o sobrescriba en el caso de que la página web en que estemos ya tenga contenido, lo que nosotros escribimos entre paréntesis justo al lado de la palabra write. El motor del navegador web toma esa instrucción como código javascript, lo lee, parte por parte y luego ejecuta el código, modificando los elementos que hayamos escrito de la página.

Document es un objeto el cual es por decirlo así lo que vemos e la pantalla al desplegar una página web, ese objeto es un hijo de otro objeto mayor llamado window, el cual no sólo guarda el objeto document dentro de su function, sino que también guarda al lado de document cientos de otras propiedades.

Como el object window es el que precede a todos los demás objetos de la página web, se puede usar la palabra `windows.alert('Hello');` y se ejecutará alert normalmente, porque alert es uno de los objetos pertenecientes a window, tanto si se escribe `alert('Hello');` como si se escribe `window.alert('hello');`; de cualquier manera la página web reconoce el código, sin embargo no se puede escribir `windows.write('Hello');`; porque la function o método 'write', es perteneciente a document a pesar de que document sea perteneciente a window, write está dentro de document por lo tanto no puede ser accesado o llamado por window, solo mediante document.

## Document Selectors

Los selectors seleccionan diferentes elementos de una página web para modificarlos.

- ***document.getElementsByTagName('h1')***: Selecciona los elementos tag de HTML(h1, h2, p, div, etc).
- ***document.getElementsByClassName('Second-phrase')***: Selecciona el elemento Class con el nombre dentro del paréntesis.
- ***document.getElementById('Firstsubtitle')***: Selecciona un elemento Id que se llame Firstsubtitle.
- ***document.querySelector()***; / ***document.querySelector.All()*** : `querySelector` selecciona el primer elemento de el elemento que se especifique dentro del paréntesis, por ejemplo si es una lista solo seleccionará el primer li de la lista, `querySelectorAll` selecciona todos los elementos contenidos dentro del elemento especificado dentro del paréntesis. Se pueden especificar seleccionar uno o más elementos dentro del paréntesis. Generalmente se recomienda usar estos dos selectors en vez de los anteriores *getElement*, son más precisos y actualizados.



- **getAttribute / setAttribute:** se deben usar colocando primero `document.querySelector('li').getAttribute('lista1');` , se debe primero obtener el elemento al que se quiere especificar que atributo obtener (elementos y atributos son diferentes), y luego se escribe el atributo al que se quiere acceder, se obtendrá el contenido del atributo, si se escribe en la consola del navegador se podrá ver el contenido del atributo seleccionado, ejm:

```
<!DOCTYPE html>
<html>
<body>
<h1 numberattribute='37'> Javascript </h1>

</body>
</html>
```

```
// Javascript
document.querySelector('h1').getAttribute('numberattribute', '1000'); // -> <h1> numberattribute='1000'> Javasc
```

los paréntesis después de escribir `getAttribute` son para indicar (de izquierda a derecha), primero, el nombre del atributo que queremos modificar, y segundo el contenido que queremos sobrescribir, borrando el contenido anterior, en este caso cambiamos el 37 por el 1000.

- **style:** así queremos cambiar las propiedades de las clases en HTML (estilos), se puede usar `document.querySelector().style.backgroundColor = 'blue';` en el ejemplo anterior se cambia el color del fondo de el elemento `h1` a color azul.

Sin embargo, no se recomienda usar `style` para cambiar propiedades de `css` en `html`, pues lo que realmente hace es escribir el estilo `css` en el elemento `h1` de el documento `html`, y esto puede generar conflictos de prioridad con los estilos ya establecidos en `css`.

```
<!DOCTYPE html>
<html>
<body>
<h1 backgroundColor='blue'> Javascript </h1> <!-- Como se puede notar se escribió el estilo css dentro del eleme

</body>
</html>
```

- **className / classList:** Se usa para agregar o modificar un estilo `css` en el archivo `css`, no se agrega o escribe el estilo dentro del elemento `html` sino dentro del archivo `css`, para eso debe tenerse en cuenta que se está especificando o pidiendo usar una clase ya declarada en el archivo `css`. `className` se usa generalmente para cambiar el estilo de una sola clase y elemento, mientras que `classList` se usa generalmente para cambiar las clases de varios elementos agrupados por ejemplo elementos `'li'`, aunque la diferencia está en que `classList` cuenta con tres opciones: `add`, `remove`, y `toggle`, las cuales (en orden): agregan una clase nueva sin borrar las anteriores, `remove` remueve todas las clases que previamente tenga y pone la nueva, y `toggle` intercambia entre agregar o quitar una clase nueva dependiendo de la acción, ejm:

- **Con className:**

```
<!-- HTML -->
<h1> title </h1>

/* CSS */
.bluetitle {
    background-color: blue;
}

/* Javascript */
document.querySelector('h1').className = '.bluetitle';
```

- **Con classList:**

```
<!-- HTML -->
<ul>
    <li>Lista</li>
    <li>Lista</li>
    <li>Lista</li>
    <li>Lista</li>
</ul>

/* CSS */
.bluelista {
    background-color: blue;
}

/* Javascript */
document.querySelector('li').className = '.bluelista';
```

- **Con classList:**

```
<!-- HTML -->
<ul>
    <li>Lista</li>
    <li>Lista</li>
    <li>Lista</li>
    <li>Lista</li>
</ul>
```

```
/* CSS */
.bluelista {
    background-color: blue;
}
```

```
/* Javascript */
document.querySelector('ul').classList.add('redletra'); //En este caso se agrega la clase redletra, más no reemp
document.querySelector('ul').classList.remove('redletra');// en este caso se remueve o sobrescribe la clase que
document.querySelector('ul').classList.toggle('redletra');//en este caso es parecido a add en el sentido de que
```

- ***innerHTML:***

Se usa para sobrescribir nuevos elementos o contenido dentro de un elemento de un archivo HTML, o agregar un nuevo elemento HTML con su propio contenido, si se escribe el mismo elemento que ya existe en HTML, lo que hará será sobrescribir el anterior con el contenido nuevo, y si el elemento es nuevo solo se agregará a la página o HTML pero no removerá nada:

- *innerHTML* *Sobrescribe un elemento HTML con uno nuevo:*

```
<h1>Título</h1>
```

```
document.querySelector('h1').innerHTML = '<h1>Nuevo título</h1>';
```

```
<h1>Nuevo título</h1>
```