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 archuvo 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 u 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 caracter tiene que ser un número indica en que lugar se quiere agregar o remover contando desde 0 en adelante en la posición de los itms de la lista, el siguiente caracter es otro número que indicará cauántos 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',  'Bluberries'];

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 hacis 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 item de la lista hasta la primera.

Ejercicios usando **array properties**

Usar la siguiente variable:

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

- Remover el item 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 alfabetico, 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 items, la diferencia es que en objects se pueden guardar una propiedad y un valor por cada item a diferencia de los arrays que cada item de la lista se guarda con el valor de un número y no se le puede asignar a cada item un nombre en específico. ejm:

arrays:

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

objects:

```
const user = {
```

```
    name: 'Jhon', // -> guarda propiedades(name) y valores para cada prop
    age: 34,
    hobby: 'soccer',
};

user.favouriteFood = 'spinach'; // -> agrega un nuevo item llamado favour

console.log(user); // -> da como resultado la lista anterior más el nuevo
```

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',
```

- *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 excercise

- 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 usuarioas 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 usename");
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 item 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 items en la lista **todo**, el cual en ese momento se detendra el proceso.

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

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

El último bloque de código tiene tres declaraciones entre paréntesis, primero se define la variable i dandole 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 items 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 items de la variable todos entonces se ejecuta el proceso interno que está entre corchetes({console.log...imprime el item 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 items de la variable todos y de esa manera el console.log(todos[i]) va a imprimir los items 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 conosla 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 valot 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 c
    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 parentésis 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 va
todosimportant.forEach(logTodos); // -> da como resultado todos los items
```

Como se puede ver en el ejercicio anterior uno de los beneficios de separar la function de la

declaraciónde 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 conosla, escribir en caso de que sea una función: [].forEach y darle Enter, saldrá resultados como este: *ƒ forEach() { [native code] }* dando a entender que ese browser sí soporta esa función.