***JAVA***

**DÍA 1**. *Variables*.

Para nombrar una variable se usa:

* **Let**: para definir una variable
* **Const**: para definir una constante
* **Var**: menos utilizado actualmente

Java te permite redefinir las variables sin muchos problemas. Se reescribe la anterior. Las constantes no se pueden reasignar.

Para hacer una suma de variables sería, por ejemplo, let sum = variable1 + variable2

console. log(“ “) : para escribir algo

Tipos de datos:

* **Number**: num o floatnum (decimales)
* **Boleanas**: dos posibilidades TRUE y FALSE
* **String**: se puede usar ‘ ‘ o “ “.

Se puede definir como NULL la variable también. Sería nula.

Console.log (typeof(variable))

dice el tipo de variable.

Console.log (Number(variable) + Number (variable2))

Cambia string por number. No todo se puede cambiar. Las boleanas tienen 2 valores si se pasan a nº (0 o 1)

Cuando

console.log(str == trueValue)

daría true si es 1. Si ponemos un nº como 0 daría falso.

Cuando

console. log(emptyString == true)

daría false porque un valor indefinido no puede ser definido como cierto

Si usamos solo un = estamos asignando un valor. Si usamos dos ( == ) estamos haciendo una comparación. Si usamos tres ( ===) es que es exactamente igual. No hace una conversión al tipo de dato del otro como con dos.

Se puede usar una boleana como comparación. Por ejemplo, si la boleana es 0 (false) se puede hacer un bool >true. Lo que pasaría el false a un nº. Sería poner console.log( bool > true). Si lo cambiamos por 1 seguiría dando false porque 1 no es mayor que 1.

Otra opción es poner <= lo que haría dos comparaciones y daría dos resultados. Es cierto que es igual a 1, pero es falso que sea menor.

Si comparamos nºs y strings se convertiría uno de los dos al otro.

Console.log(str == 1)

**RECUERDA:**

- Definir una variable con la primera letra de la primera palabra en minúscula y la segunda, tercera… en MAYUS. “Camel Case”  
- Que las líneas de texto van entre “ “.

-Que se puede dejar sin valor. Algo especial de JS. No existe como tal o no tiene todavía valor. Se sigue pudiendo usar como variable.

- No se puede empezar las variables con nºs y caracteres especiales.

- Escribiendo node “nombre del .js” lo ejecutamos en la terminal

- Que el símbolo $[] significa que se está escribiendo un parámetro dentro de un string.

- Los boleanos se pueden convertir a nº o a string con

console.log (String(trueValue));

- Undefined puede convertirse a cualquier tipo de dato.

Console.log (Boolean(undefined));

- ctrl + barra se pone lo seleccionado en verde.

-Con shift + alt se puede escribir varias veces lo mismo

- La diferencia entre switch y else if es que en switch se sigue ejecutando cada caso si no pones break. Mientras que en else if se queda en el primer caso en el que sea acertado.

- si pones !variable se usa el valor contrario a la variable.

**DÍA 2.** *OPERADORES LÓGICOS*

*CONDICIONALES*

-   Let output = num == null || num == undefined ? 0 : 5 + num

Si el nº es nulo o indefinido el valor es 0. Si no se cumple eso se suma 5 al nº original.

Se puede usar como checkpoint para conseguir un nº ya que un valor indefinido puede hacer que deje de funcionar el resto del código.

Let str = conditionA && conditionB ? “in bounds” : “out of bounds”;

Si la condición A y B son correctas está dentro de los parámetros, si no lo son están fuera.

-if (variableA = x) -o == variableB- {

    PASAN COSAS

    }

    else if ( “  “  “) {

    PASAN OTRAS COSAS

    }

    else{

    SI NO SE CUMPLE EL RESTO

    }

    -   SWITCH () {

    Case 1:

        Break;

    Case 2:

        Break;

    .

    .

    .

    Default:

*LOOPS*

-for (let i =0; i < 10; i = i +2).

La premisa inicial es que i es igual a 0. El ciclo termina cuando i llega a 10. Se va sumando cada vez 2 más a i. Si pones i++ solo se suma 1.

-While (e <10){

Console.log(“Iteracion de while $[e]”);

e++;

}

**RECUERDA:**

- && se usa como “and” y || se usa como “or”, XOR se usa como “exclusive or”

- si pones =! Significa que es igual o distinto.

- en vez de usar if/else, una versión acortada sería ? ”si se cumple la condición” : “si no se cumple”;

- no se pone el break si lo que quieres es que se ejecute desde un punto hasta el final.

- el default no es necesario

- Se usa mucho más el for.

- el símbolo de $ en $[e] es para coger el valor actual de e y no el inicial del loop.

- Ctrl + C para parar de ejecutar la máquina

- Si defines i dentro del loop se mantendrá solo durante ese loop. Si quieres que siga siendo funcional fuera de este tienes que especificarlo.

**DIA 3.** *FUNCIONES*

En console. log () “log” es una función (o método). Lo que va dentro de () se consideran argumentos. Se pueden tener tantos argumentos como se quiera (arg1, arg2, arg3…). Hay 2 tipos de funciones en JS. Las que dan un resultado y las que realizan operaciones sin darlo. Un ejemplo del 1er caso:

Function nameofmyfunction (arg1, arg2, arg3…) {

    Let sum = arg1 + arg2 + arg3

    return sum;

    }

    Let output = nameofmyfunction (2,3)

    Console.log(output);

Dentro de function por {} es donde se hacen los cálculos.. Return devuelve un valor que será posible usar fuera de la función. Declaramos la variable fuera de la función. Ahí podemos asignarles valores.

El segundo caso donde no se devuelve nada:

Function sayHello () {

    Console.log(“Hello”);

    }

    Let hello = sayHello();

    Console.log(hello);

    Daría Undefined

    Let func = sayHello;

    Func();

Las funciones siempre empiezan con function como palabra clave. Puede tener un cálculo dentro de los parámetros o algún código fuera. Por ejemplo:

Function printName () {

    Console.log(“Hola “ + nombre);

}

Una constante permite que se pueda escribir las variables fuera de la función.

Function estaaprobado (estudiante){

    If(estudiante.aprobado) {

    Console.log(“Puede empezar el curso de FS”);

    } else{

    Console.log(“Repite”);

    Otra opción sería:

    Let func = function (arg1){

    Console.log ( “added” + arg1);

    }

    Func = (string);

}

}

También se puede usar Switch

Switch

Case true:

Case false:

Para asignar valores a una variable, no hay que ponerlo seguido. Sería:

Const listaDeEstudiantes = [

    {

    fullName: “Manuel Benítez”,

    estaAprobado: true,

    },

    {

    fullName: “Manuel Benítez”,

    estaAprobado: true,

    },

    {

    fullName: “Manuel Benítez”,

    estaAprobado: true,

    },

    ];

    Const estaAprobado = (arrayDeEstudiantes) => {

    arrayDeEstudiantes. forEach((estudiante) => (

        if (estudiante.estaAprobado) {

    console.log (estudiante.fullName);

    }

    ));

*CALLBACK*

Dentro de una función podemos añadir otra función. Para eso está callback. Para que, cuando una función termine, se inicie otra. Callback no lleva como argumento una variable o una constante, como el resto de cosas que hemos visto, sino una función.

Un ejemplo sería:

Function first (){

    Console.log (1);

    }

    Function second (){

    Console.log(2);

    }

    First();

    Second();

En este caso, cabría la posibilidad de que se ejecute Second primero que First porque hay un proceso que no termina de ejecutarse en First. En ese caso añadiríamos un callback. Quedaría tal que así:

Function first (callback) {

    Console.log (1);

    Callback();

    }

    Function second (){

    Console.log(2);

    }

    First(second);

}

Second sería la función callback y First sería la principal.

*ARROW FUNCTIONS*

Solo funcionan como funciones anónimas o como variables. Se utiliza => de ahí el nombre. Tiene el return implícito por lo que no necesita ponerlo.

Let arrowfunction = (arg1) => arg1 + 3;

ó

Let arrowfunction = (arg1, arg2) => {

Let intermediate = arg2 + 3

Return arg1 + intermediate;

}

*VARIABLES SCOPE*

Hay diferentes órdenes de variables dentro de JS. El primero es global scope es todo lo que no está dentro de if, for, function… El block Scope, es lo que está dentro de If. Otro es el function scope que es, obviamente, lo que está dentro de una función. Sobrescribe lo que está en global scope dentro de la función. Algo del estilo pasa con Block Scope. Lo que BS funciona con ES6 por lo que es ligeramente diferente. Cuando se usa Var. Con var se sobreescribe en Global Scope.

**RECUERDA:**

*Funciones*

- Si lo declaras dentro de la función solo va a servir para esa función.

- hay funciones que se declaran con

const printname () => {

    Console.log(nombre);

    };

    Const callback = () => {

        Console.log(“Este es mi callback);

    };

    printName(“Manuel” + “Benitez”, callback);

    const nombre = returnName (“Manuel”);

    console.log (nombre, “Este es el nombre devuelto”)

- Los parámetros de una función se separan por coma.

- Cuando la función te devuelte los datos, tienes que añadirlo a una variable. Esta variable puede ser una base de datos.

- Una vez declarada una función se puede usar antes o después en el código.

- const y function pueden ser intercambiables

*Callback*

- setTimeout se escribe (función, tiempo de delay) ; También se puede añadir dentro de la función. Por ejemplo:

setTimeout (function (){ console.log()}, tiempo)

*Arrow Function*

-Se pueden escribir sin usar corchetes, pero cuando se usan hay que añadir return;

*Variables Scope*

- this se usa dentro de una función o callback. Recopila información de toda la función para poder usarla en otro momento. Si, por ejemplo llamo a una variable this.something en global. Si la uso en global, es válida para todo global. Pero no tendría valor dentro de una función.

*¡INFO ÚTIL!*

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions>

<https://en.wikipedia.org/wiki/Don%27t_repeat_yourself>

**Día 4***.POO (Programación Orientada a Objetos)*

Es parecido a las funciones. Pero, en este caso, puedes meter dentro diferentes variables y usarlas fuera. También puedes añadir variables desde fuera del Objeto. Se declara diferente a lo que hemos visto. Sería, por ejemplo:

Let obj = {

    “key” = “value”,

    “number” = 2

    }

    Obj [“number”] = 7;

Cambia la variable “number a partir de ese momento

Obj[“newKey”] = 10;

Añade una nueva variable al Objeto

Console.log (obj[“newKey”])

Console.log(Object.keys(obj))

Mostrará en el terminal las variables que hay en el Objeto (obj).

El Uso de IN en Objetos: en el caso de que queramos saber si una variable se encuentra dentro de un Objeto usamos IF e IN. Por ejemplo, si queremos saber si la variable number se encuentra dentro se escribiría algo tal que así:

If (“number” in obj){

    Console.log (obj.number + 10)

    } else{

    Console.log (“number doesn’t exist)

    }

    De esta forma, el console. log no dará error.

    Además, también podemos usar el comando For. Sería algo como:

    For (const variable in obj) {

    Console. Log (typeof(variable);

    Console.log (typeof(obj[variable]));

    }

En el primer caso saldría todo strings. En el segundo, daría el tipo de cada key.

*Usar el valor vs la referencia*

Cuando usamos el valor dentro de la variable no se modifica el contenido de la variable inicial. Cuando usamos una referencia los cálculos se hacen sobre la variable inicial. Por ejemplo, cuando usamos una variable:

Function changeValue(obj, key, val) {

    Obj[key] = val;

    }

    Console.log(obj.number)

    changeValue (obj, “number”, 15);

    Console.log(obj.number)

Si ejecutamos ambos console el valor diferirá en cada caso. Siendo el primero 10 (valor definido con anterioridad -al principio de POO-) y el segundo, 15(el nuevo valor).

*FUNCIONES DE OBJETOS*

Así es como he resuelto el problema:

let user = {

    name: "Cosimo",

    age: 30,

    };

   let newUser = {};

    for (let key in user) {

        newUser[key]=user[key];

    }

Otra manera en la que lo he resuelto (*clone*):

let clone = Object.assign({}, user);

clone.name= "Ricardo";

console.log(user.name);

console.log(clone.name);

Siendo user.name el objeto copiado.

*Comando*: **Freeze**

definir como constante un objeto no es realmente útil ya que se puede seguir modificando. Para que un valor dentro de un objeto se mantenga constante tenemos que usar el comando Freeze:

console.log("Ejercicio 41");

const smartphone = {

    brand: "Apple",

    name: "Iphone 13",

    price: 1300,

    };

    Object.freeze(smartphone);

    smartphone.price = 100;

    console.log(smartphone);

Al ejecutar esto el valor que se mantiene es el de 1300.

Object. Freeze (obj)

Obj. Number = 2;

Console.log(obj)

Es muy útil cuando se tiene un Objeto complejo con muchas operaciones matemáticas y no quieres que se modifiquen los valores por algún motivo.

*CONSTRUCTOR FUNCTIONS*

Es una herramienta que nos da el Objeto en un determinado formato con unas determinadas keys. Se suele nombrar igual que otros Objetos, pero la primera en Mayus. Se suele usar el comando this. Le asignamos nuevas keys con this. Ejemplo:

function Person (name, age, nationality) {

    this.name = name;

    this.age = age;

    this.nationality = nationality;

    }

    Let greg =new Person (“Greg”, 28, “USA”);

    Let susan =new Person (“Susan”, 34, “Canada”);

Ejercicio Resuelto:

function Smartphone(brand, name, price) {

    this.brand=brand;

    this.name=name;

    this.price=price;

}

let samsung =  new Smartphone("samsung", "a14", 600);

let xaomi = new Smartphone("xaomi", "velvet", 350);

console.log(samsung);

console.log(xaomi);

Si los argumentos se definen dentro de console. log dan undefined. Hay que crearlos aparte Lo bueno de esto es que no tienes que tener solo una variable. Puedes crear más. Si la función de la que parte no retorna nada siempre va a dar undefined.

*NESTED OBJECTS*

Básicamente objetos dentro de otros objetos.

Let nested = {

    “subObject”: {

        “value”: 1,

        “value2”: 5,

        “nested”: {

            “value”: 3,

        }

}

}

Console.log(nested.subObject.value). \*

Se escribiría tal que así.\*

Ejercicio Resuelto:

const student = {

    personalData: {

        name: "Rafael",

        surname: "Coin",

        age: 27,

    },

    id: 1,

    school: "Liceo",

    year: 3,

    };

    console.log(student.personalData);

En el caso de que queramos evitar un error siempre podemos añadir un if (nested… === undefined) y una línea de texto que informe del error. También podemos añadir el terciary en vez del if. El terciary era el ? “si pasa” : “si no pasa”;

Otras opciones serían algo así:

Let value = nested. subObject.doesNotExist?.value === undefined

Console.log (value + 10);

Si añadimos un valor ya no está indefinido. Así nos aseguramos de que no salga un error.

**RECUERDA:**

*POO*

- Object hace referencia a la librería Objetos de JS. De la misma forma, keys forma parte de dicha librería.

- Dentro del Objeto se puede definir una función.

- this no funciona de la misma manera dentro de POO que en funciones.

- Un Objeto puede tener otros objetos en su interior.

- Cuando usamos una función estamos dando una referencia, lo que nos permite cambiar el valor. Esto no pasa cuando usamos un primitivo como un nº, una string o un boleano.

- Si usamos el comando copy y lo modificamos. No estamos cambiando el valor original. Por ejemplo, copy = a

- se puede usar el parámetro is para comparar dos argumentos y ver si son iguales.

Por ejemplo:

console. log (Object. Is(arg1,arg2))

Es igual que cuando escribimos ===.

*Funciones de Objetos*

-Si ponemos let copy =obj; no estamos copiando el objeto como tal. Si lo queremos copiar realmente habría que poner:

Function changeValue(obj, key, val) {

    Let copy = Object.assign ({}, obj); \*

    copy[key] = val;

    return copy;

    }

    Let output = changeValue (obj, “number”, 2)

    Console.log (obj.number)

    Console.log(output.number)

\*- Dejamos el primer argumento vacío y el segundo es lo del objeto que queremos copiar

*Nested Functions*

- Los : es la manera en la que se crea los objetos secundarios.

- Un error muy común es que no esté definido un nivel. Como en este caso:

Console.log(nested.subObject.doesNotExist.value)

Entonces habría que poner un ? que significaría que, en el caso de que estuviera definido, se llevaría a cabo el comando. Quedaría así.

Console.log(nested.subObject.doesNotExist?.value);

*¡INFO ÚTIL!*

*POO*

<https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos>

<https://javascript.info/object-copy>

*Funciones de Objetos*

MDN WEB Docs

<https://universidadeuropea.com/blog/programacion-orientada-objetos/#:~:text=La%20programaci%C3%B3n%20orientada%20a%20objetos%20es%20un%20modelo%20de%20programaci%C3%B3n,l%C3%B3gica%20necesaria%20para%20esa%20manipulaci%C3%B3n>

**Día 5**. *ARRAYS*

Permite separar la información en diversas listas para que sea más fácilmente accesible. Los objetos no permiten ordenar las keys. Por lo que si quisiéramos un orden específico tendríamos que usar:

Let keys = Object.keys(obj);

Las array se escriben con [] y un orden.

Let array = [arg1, arg2, arg3…];

Podemos usar diferentes tipos de valores (nº, bolean, letras…), pero no usa keys. Pero, como el array siempre sigue el mismo orden cuando se usa, se diferencia usando nºs. Por ejemplo, si escribimos:

Console.log(array[0]);

nos estamos refiriendo al arg1, pues se empieza a contar en 0. (zero indexing)

Diferencias entre array y objetos:

Los objetos son más “clunky”. Tienes que estar escribiendo todo el rato keys, mientras que con los arrays, al mantenerse el orden, podemos escribir todos los valores que queramos. Como no siempre queremos añadir keys a los valores, el array es más práctico.

Otras fórmulas que se utilizan:

Console.log (array.length);

For (i = 0; i < array.length < i++) {

Console.log(array[i]);

}

Ejercicio Resuelto:

console.log("Ejercicio 47");

const names = ["Luca", "Marco", "Vittorio", "Giovanni"];

console.log(names.pop());

names.push("Pippo");

names.unshift("Giovanni");

console.log(names);

Array de objetos:

se pueden crear arrays de objetos array= [

    id :1,

    name:"Rafa",

    apellido:"Coin",

    age=27,

]

array = [

    {

        id : 1,

        nombre : "Rafa",

        aproved : true,

    },

    {

        id : 2,

        nombre : "Luis",

        aproved: true,

    }

]

Hacer un loop con arrays:

numbers.forEach((numero) => {

    console.log(numero);

})      funciona como un loop y recorre todos los nºs del array

        Se puede hacer con los estudiantes con ((estudiante, indice))

        console.log(indice, estudiante.name) si solo queremos el nombre.

Un ejemplo de modificar un array más complicado:

const students3 = [

    { id: 1, name: "Luca", surname: "Rossi", age: 20 },

    { id: 2, name: "Mario", surname: "Bianchi", age: 24 },

    { id: 3, name: "Giovanni", surname: "Brambilla", age: 30 },

    ];

    console.log(students3.pop());

    students3.push ({

        id: 3,

        name: "Francesco",

        surname: "Verdi",

        age: 41

    }

        );

        for(i = 0; i > students3.length; i++){

            console.log(students3[i]);

        }

El for es necesario para poder leer en la terminal todos los valores del array.

**RECUERDA:**

- Objetos no tiene la capacidad de hacer length como las arrays.

- Al usar el zero indexing si queremos sacar el último valor del array escribimos el nºfinal -1.

- Si se escribe:

Console.log(array.length – 1);

No estamos sacando el último valor del array. Lo que estamos es eliminándolo. Por lo que puede destruir el código.

-Si quieres cambiar un valor del array se escribe de forma parecida a como se hace con objetos:

Array[0] = 10;

- Si ponemos un valor que no exista en el array, lo estaremos creando. Si después de eso miramos la longitud del array veremos que hay valores vacíos. Es decir entre el valor añadido y el valor final del array se crean huecos. Quedarían como undefined. Mientras más huecos creemos más se consumirá de memoria y menos eficiente será el código.

- Si quieres añadir un valor se usar push.

Array.push[10];

Se pasa de 18 valores, por ejemplo, a 19.

- Hay una opción contraria. Si lo que queremos es quitar un valor escribimos.

Console.log(array.pop());

No solo quita el último valor. También nos lo devuelve para que podamos usarlo.

- Otro código que se utiliza es:

Array. Unshift(10);

Al igual que push añade un valor al array. En este caso lo hace al principio del array

- Otro código no tan útil que se utiliza es:

Console.log(array.join(“ ¡ “));

Nos saca todos los valores del array en un string. Con los paréntesis podemos elegir la separación.

const names2 = ["Luca", "Marco", "Vittorio", "Giovanni"];

console.log(names2.join("-"));

*¡INFO ÚTIL!*

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>

EJERCICIOS:

<https://edabit.com/challenge/3hdXjfJozQySRC3gE>

<https://edabit.com/challenges/javascript>

<https://edabit.com/challenge/3ZtykTsx3GSoPHyBb>

De 9 a 11 para hacer el examen. Como muy tarde a las 10 15 para conectarse porque son 45 min. Hay que tener la cámara abierta.

Ej. Resuelto:

 /\*

    Write a function that takes an array of objects as a parameter, each of which represents a book with the following properties:

    title, author, year, and genre. The function should return an array of strings that contains the title and author of each book

    in the format "title by author". If the year of the book is after 2000, append " - Modern" to the end of the string.

    If the genre of the book is "Science Fiction", append " - SciFi" to the end of the string. Print the new array.

    \*/

    console.log("Ejercicio Extra 5");

    function titleAuthor () {

         const books = [

        {

            title : "La Rueda del Tiempo",

            author: "Robert Jordan",

            year: 2005,

            genre: "fantasy",

        },

       {

            title: "Fundación",

            author: "Asimov",

            year: 1925,

            genre: "science fiction",

        },

        {

            title : "El problema de los tres cuerpos",

            author: "Liu",

            year: 2019,

            genre: "science fiction",

        },

       {

            title : "El Hobbit",

            author: "Tolkien",

            year: 1950,

            genre: "fantasy",

        }

     ]

    const title = books.map (item => item.title);

    const author = books.map (item =>item.author);

    const year = books.map (item => item.year);

    const genre = books.map (item => item.genre);

for (i =0 ; i< books.length ; i++) {

    var newBooks= (String(title[i] + " by " + author[i]));

    if(year[i] > 2000) {

       var tiempo = (String(year[i] + " - Modern"));

    } else {

       var tiempo= (String (year[i] + " - classic"));

    }

     if(genre[i] === "science fiction"){

       var sciFi= ("SciFi");

    } else {

        var sciFi = ("Not SciFi");

    }

    const booksByAuthor =new Array (newBooks, tiempo, sciFi);

    console.log(booksByAuthor);

}

    }

    titleAuthor();