//COMMENTS

//These are Single Line Comments

/\*

These are Multi-Line Comments.

\*/

// DATA TYPES AND VARIABLES

/\* There are 7 different types of Data in JS, they are:

undefined, null, boolean, string, symbol, number and object.

\*/

/\* To store Data we require variables, there are 3 ways to define a variable in JS

\*/

var myName = "Ayush";

/\* "var" is used as a GLOBAL variable, once declared, we can use it anywhere in the program.

And in JS, it's like Python no need tell which type of data are we storing in the variable,

it'll figure out itself but unline Python, it's optional to use semicolon to end every executable

line ie., we can use ";" or not to end the line it doesn't matter.

\*/

let number = 5;

/\*This acts as a local variable, it can only be used within the scope of where it was declared.

\*/

const pi = 3.14;

/\* This type of variable acts as a constant, once a value is assigned to it, it can't be changed throughout the program.

Notice that here, after the line we haven't used semicolon(;) and that's fine in JS.

\*/

//STORING VALUES WITH ASSIGNMENT OPERATOR

var a;

console.log(a); // Allows us to see things in console.

a=7;

console.log(a);

//INITIALIZING VARIABLS WITH ASSIGNMENT OPERATOR

var a=9;

var c= "Apples are";

console.log(c + " Red"); //Concatenation of two strings

//CASE SENSITIVITY IN VARIABLES

/\* JS is a Case Sensitive language thus varibles in uppercase and lowercase won't be same\*/

var a=5;

var A=7;

console.log(a)

console.log(A)

a++;

A--;

console.log(a)

console.log(A)

a+= 12

A-=5

console.log(a)

console.log(A)

a\*=3

A/=2

console.log(a)

console.log(A)

//ESCAPE CHARACTERS

var myStr = "I am a \"double quoted\" string inside \"double quotes\""

/\* Where ever you see " (quote sign) put a backward slash (\) infront of it. It's just like

with newline character "\n"

\*/

/\* Another way to use "(double quote sign) inside a string is by declaring the whole string

inside a Single Quote Sign(' ').

\*/

/\* -----var myStr = "<a href="http://www.google.com" target="\_blank">Google</a>";----

This statement is not valid\*/

var myStr = '<a href="http://www.google.com" target="\_blank">Google</a>';

/\* Another way of using both single quote sign(' ') and double quote sign(" ") inside a string

is by using back-ticks( it's on the key just below Esc)

\*/

var myStr = `<a href="http://www.google.com" target="\_blank">Google</a>`;

console.log(myStr)

var myStr = "First Line\n \t \\Second Line\n\t Third Line"

console.log(myStr)

var myStr = "Ayush Kumar Tripathi";

var l = myStr.length; //Notice that the .length counts space as a character as well

console.log(l)

console.log(myStr[2]) //Indexing starts from 0

/\* Individual characters of strings are immutable, but string as a whole can be changed

\*/

var myStr = "Jello World";

//myStr[0] = "H" // This is WRONG

myStr = "Hello World" // This is allowed

// ARRAYS

var myArray = ["Ayush",1,"Kumar",2,"Tripathi","3"]

// This is valid, we can store an element of any datatype in our array.

var myNewArray = [["Ayush","Kumar","Tripathi"],[1,2,3]] // 2-D Array

console.log(myNewArray[0][0])

console.log(myArray[3])

//Unlike Strings, we can change the value of an element in an array using indexing

myArray[1] = 7

myArray[3] = 8

myArray[5] = 9

console.log(myArray[1])

console.log(myArray[3])

console.log(myArray[5])

// PUSH AND POP IN ARRAYS

var myArray = [["Ayush",1],["Kumar",2]]

myArray.push(["Tripathi",3])

console.log(myArray)

myArray.pop()

console.log(myArray) // pop() always removes the last element from the array.

//For example

var myArray = [1,2,3]

myArray.pop()

console.log(myArray) // Last element of array is removed

var myArray = [1,2,3]

myArray.shift()

console.log(myArray) // Removes an element from the beginning of array

var myArray = [1,2,3]

myArray.unshift(0)

console.log(myArray) // Adds an element at the beginning of an array

//Note that nested array is nothing but Multidimensional Arrays.

// Also note that, you cannot include a Multi-Line Comment withing another Multi-Line Comment

/\*---------------------------------------- FUNCTIONS ------------------------------------------\*/

function display(){ // Unlike Python, here we require Brackets for Function

console.log("Hello World");

}

display();

function add(a,b){ // Note that Datatype of arguments is not defined, JS learns it on it's own

console.log(a+b);

}

add(10,5);

/\*----------------------------------- Global Scope and Functions --------------------------------\*/

var myGlobal = 10 // It's like declaring int myGlobal = 10 outside the scope of every function

function func1(){

var local = 10 /\* Here,"local" variable is local to this function only this is because

We've used "var" keyword to declare it inside this function, thus this

variable will be local to this function only.

\*/

/\* oopsGlobal = 5; \*/ /\* Here, "oopsGlobal" variable is a Global variable since this variable can be accessed outside this function as well. This is because we've declared this variable without using "var" keyword.

And YES it is possible to declare a variable without using "var","let" or "const". Although Scrimba doesn't allow this type of declaration of variable but this will run in our Browser.

\*/

}

function func2(){

var output = ""

if(typeof myGlobal != "undefined"){

output+="myGlobal: " + myGlobal // "myGlobal" was already declared as a global function thus will be accessed withing this function as well . Thus, output = myGlobal: 10

}

if(typeof oopsGlobal != "undefined"){

output += "oopsGlobal: " + oopsGlobal /\*This statement will also run because oopsGlobal is a Global variable thus it will be defined within this function as well

output = oopsGlobal: 5\*/

}

console.log(output)

}

func1();

func2(); // Scrimba won't run this ie., it'll give as "assignment to undeclared variable oopsGlobal (/index.js:13)"

/\*------------------------------- Local Scope of Variables in Functions -----------------------\*/

function func(){

var myVar = 5;

console.log(myVar)

}

func();

/\* console.log(myVar) \*/ // Error Will Occur as "myVar is not defined (/index.js:216)" since scope of myVar was within the function "func"

// Global vs Local

/\* A Global and a Local variable can have same name but the Local variable will be given preference over Global Variable\*/

var outerWear = "T-Shirt";

function myOutfit(){

var outerWear = "Sweater"

return outerWear;

}

console.log(myOutfit());

/\*------------------------------------ Function with no Return value ------------------------\*/

var sum = 0

function addFive(num){

sum = num +5;

}

sum = addFive();

console.log(addFive); /\* Only the declaration statement of our function is

printed out ie., addFive(num)

\*/

console.log(sum) /\*null value is store in sum, as no value is returned by our funciton\*/

/\*-------------------------------------JSON.stringify() -------------------------------------- \*/

var testArr = [1,2,3,4,5]

console.log("Array = " + testArr) // testArr give each element

console.log("Array = " + JSON.stringify(testArr)) /\* JSON.stringify() converts "testArr" into a string and then added to "Array = ", thus giving us the whole array with square brackets

\*/

/\* ------------------------ DIFFERENCE BETWEEN "==" and "===" ----------------------------------\*/

/\* "==" is used to check for equality but it doesn't check for datatype equality ie., it does datatype conversion at the time of checking.

But, "===" is called Strict Equality Operator because in it's case not only the value of variable should be equal but Datatype of variable should also match ie., it doen't do Datatype Conversion at the time of checking\*/

function check(val){

if(val ==5){

console.log("Datatype conversion is done")

}

if (val ===5){

console.log("Datatype conversion is not done")

}

}

check("5"); /\* Only First if statement will be executed because "==" operator does datatype conversion thus, it'll convert string to integer to check for value of string and integer, since both are 5 thus if condition is satisfied. But Second if statement won't get executed because the DataType is not same for both values, one is an Integer and another one is a String.

\*/

check(5); /\* Both If statements are executed as in first if there is no need of datatype conversion as both values being compared are integer values and the second if statement will also get executed because both Datatype and Value of variable is same as the value being compare to it

\*/

/\* ----------------------------INEQUALITY AND STRICT INEQUALITY OPERATOR------------------------\*/

function check2(num,val){

if(val !=5){

console.log(num + " Datatype conversion is done")

}

if (val !==5){

console.log(num + " Datatype conversion is not done")

}

}

check2(1,"5") /\* In this case, "!=" Inequality operator does DataType conversion and since both the values are same after converting string to integer thus first if statement won't be executed. But, for second if statement since the ANY ONE of the Datatype or Value must be different for second if statement to get executed and here the datatype is different thus it gets executed.

\*/

check2(2,7) /\* For first if statement to get executed, only value must\*/

/\*------------------------------------------- SWITCH ------------------------------------------\*/

function sequentialSizes(val){

var answer = ""

/\* If val lies in [1,3] then answer will be Low

else if vale lies in [4,6] then answer will be Mid

else if val lies in [7,9] then answer will be High

\*/

/\* Also note that, checking of values is done STRICTLY in switch ie., whether value of variable "val" is 1,2,3 etc. is checked strictly ie., both value and DataTypes are checked.

\*/

switch(val){

case 1:

case 2:

case 3:

answer = "Low";

break;

case 4:

case 5:

case 6:

answer= "Mid";

break;

case 7:

case 8:

case 9:

answer = "High";

break;

default:

answer = "Default";

break;

}

return answer

}

console.log(sequentialSizes(8))

/\*---------------------------------------------- LITTLE MATHS --------------------------------\*/

var val = 5.6

console.log(Math.round(val))

val = Math.round(val)

console.log(Math.pow(val,2))

val = 16

console.log(Math.sqrt(val))

/\*------------------------------------------OBJECTS-------------------------------------------\*/

/\* Objects are like dictionary in Python, Objects in JS are Key Value Pairs, here in JS it's known as Properties and Values Pair.\*/

var myDog = {

"name" : "Tipu",

"legs" : 4,

"tails" : 1,

"Number of friends" : 10,

10 : 5

}; // ";" can be omitted, it's optional, it's there to denote that statement has ended

/\* There are two ways to access values of Properties of Objects:

(1) Using Dot Operator

- Can't be used when there is a space in the name of the Property

(2) Using Square Brackets

- Can be used everytime.

\*/

var obj = myDog.name;

console.log(obj)

myDog.name = "Salakha" // Changing the Value of Property using Dot Operator

console.log(myDog.name)

var frnds = myDog["Number of friends"]; /\* Notice that here that the name of the property has space in it thus we can't use the Dot Operator to Access the Property, this is where we use Square Brackets to access the value of the given property.

\*/

console.log(frnds)

myDog["Number of friends"] = 15;

console.log(myDog["Number of friends"]) // Changing the Value of Property using Square Brackets

/\* console.log(myDog.10)\*/ /\* Wrong way to access it, it'll give an error "Error: SyntaxError:unknown:Unexpected token, expected , (363:17) (/index.js:1)"

\*/

console.log(myDog[10]) /\* But here, we can use Square Brackets to access that Property as well \*/

var val = 10

console.log(myDog.val) // Gives NULL as output

console.log(myDog[val]) // Gives 5 as output

/\*--------------------------- Adding and Deleting Properties from an Object---------------------\*/

myDog["Bark"] = "Bow - Bow"

myDog.teeth = 32

console.log(myDog.Bark)

delete myDog.Bark; // Deleting a Property from myDog Object

console.log(myDog.Bark)

console.log(myDog.teeth)

delete myDog["teeth"]; // Deleting a Property from myDog Object

console.log(myDog.teeth)

// Objects can used as a Dictionary / Map as Well

var lookup = {

"first" : 1,

"second" : 2,

"third" : 3

};

var result = lookup["second"]; // We can access the value of the key "second" using this notation.

// We can use this property of object acting as a Map to replace

// Switch statements

console.log(result)

/\*--------------------------- Checking whether an Object has a Given Property or Not--------------\*/

function checkObj(checkProp){

if (myDog.hasOwnProperty(checkProp)){

return myDog[checkProp]

}

else{

return "Not Found";

}

}

console.log(checkObj("name")) // Salakha

console.log(checkObj("Age")) // Not Found

/\*--------------------------------------- Array Of Objects------------------------------------\*/

var myArr = [

{

"name" : "Ayush",

"Age" : 21,

"Roll No." : 43914802716,

"Percentage" : [10.00,94.6,77.33] // Array can also be value of a Property

},

{

"name" : "Shreya",

"Age" : 17,

"Roll No." : 21,

"Percentage" : [10.00,97.8,99.0]

}

];

console.log(myArr[1].name) // Output - Shreya

console.log(myArr[0]["Roll No."]) /\* Here '.' notation can't be used because the property

name "Roll No." has space in it \*/

/\*------------------------------------------ Nested Objects ----------------------------------\*/

var nestedObject = {

"car" : {

"inside" : {

"seats" : 7,

"Sunroof" : "Yes",

"Airbags" : 7,

"Charging Ports" : 2

},

"outside features": {

"wheels" : 4,

"Clearance from Ground in inches" : 18

}

}

};

var airbags = nestedObject.car.inside.Airbags;

console.log(airbags);

var wheels = nestedObject.car["outside features"].wheels;

console.log(wheels);

var clearance = nestedObject.car["outside features"]["Clearance from Ground in inches"];

console.log(clearance);

/\*---------------------------- Accessing Nested Arrays withing the Object------------------\*/

var syllabus = [

{

Sem : 1,

subjects : [

"Maths",

"Physics",

"Chemistry"

]

},

{

Sem : 2,

subjects : [

"Introduction to Programming",

"Maths-2",

"Physics-2",

"Chemistry-2"

]

}

];

var programmingSubject = syllabus[1].subjects[0]; /\* Using '.' and '[]' notation we can

navigate and select items from nested arrays

within Objects

\*/

console.log(programmingSubject);

/\*---------------------------------------- Record Collection -----------------------------------\*/

var collection = {

"2548" : {

"album" : "Slippery When Wet",

"artist" : "Bon Jovi",

"tracks" : [

"Let It Rock" ,

"You Give Love a Bad Name"

]

},

"2468" : {

"album" : "1999",

"artist" : "Prince",

"tracks" : [

"1999",

"Little Red Corvette"

]

},

"1245" : {

"artist" : "Robert Palmer",

"tracks" : [ ]

},

"5439" : {

"album" : "ABBA Gold"

}

};

// To make a copy of an object we use JSON.parse() and JSON.stringify()

// For example, here to make a copy of "collection" object we use the following code:

var collectionCopy = JSON.parse(JSON.stringify(collection));

// Now, a copy of the object "collection" is stored in "collectionCopy"

/\* Now, task is to make a function which takes id, property, value. If the property doesn't contain

an array then change it's value to the value that has been passed to the function.

If the property contains an array then simply add the value to that array, like in "tracks".

If the property contains an empty string then delete the property.

\*/

function updateRecords(id, prop, value){

if (value === ""){

delete collection[id][prop]; // Prefer using '[]' notation when it's not known whether

// the property name will contain space or not.

}

else

if(prop === "tracks"){

collection[id][prop] = collection[id][prop] || [];

/\* This line means that if prop === tracks then we check it's value using

"collection[id][prop]". Now there can be two conditions, whether the track array already exist or it's an empty array:

(1) If the track array already exists then we set "collection[id][prop]" to itself only.

OR

(2) If the track array doesn't already exist then we set "collection[id][prop]" to an empty array as shown above using "[]".

\*/

collection[id][prop].push(value);

}

else{

collection[id][prop] = value;

}

return collection;

}

console.log(updateRecords(2468,"tracks","test"));

/\* --------------------------------------WHILE LOOP -----------------------------------------\*/

var myArray = [];

var i=0;

while(i<5){

myArray.push(i);

i++;

}

console.log(myArray);

/\* ----------------------------------------FOR LOOP-----------------------------------------\*/

var myArray = [];

var ourArray = [];

for(var i=0,j=0 ; i<5,j<5 ; ++i,j++){

myArray.push(i);

ourArray.push(i+j);

}

console.log(myArray);

console.log(ourArray);

/\* ---------------------------Iterating through an array using for loop -----------------------\*/

var ourArray = [10,11,12,13,15,16,17,18];

var counter = 0;

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

counter += ourArray[i];

}

console.log(counter);

/\*------------------------------------------ Nested For Loop--------------------------------------\*/

function multiplyAll(arr){

var product = 1;

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

for(var j=0;j<arr[i].length;++j){

product \*= arr[i][j];

}

}

return product;

}

var product = multiplyAll([[1,2],[3,4],[5,6,7]]);

console.log(product);

/\*----------------------------------------- DO-WHILE LOOP ----------------------------------------\*/

var myArray = [];

var i = 10;

while(i<5){

myArray.push(i);

i++;

}

var ourArray = [];

var j=10;

do {

ourArray.push(j);

j++;

}while(j<5)

// Note that you can output TWO OR MORE VALUES using "console.log" as shown below:

console.log(i,myArray); // Output - 10,[]

console.log(j,ourArray); // Output - 11, [10]

// Thus, Do-While Loop atleast executes once even if the condition is not satisfied at the beginning

// But, For and While Loops won't execute even once when the condition is not satisfied.

/\*-------------------------------------------PROFILE LOOKUP --------------------------------------\*/

// TO BE DONE FROM 2:24:12 IN VIDEO

/\*------------------------------ GENERATE RANDOM FRACTIONS/DECIMAL NUMBERS------------------------\*/

var randomDecimal = Math.random(); // Math.random() returns a random Decimal number in the // range [0,1).

console.log(randomDecimal);

/\*----------------------------------- GENERATE RANDOM WHOLE NUMBERS-------------------------------\*/

// A random Whole Number can be accomplished using "Math.floor()" property

var randomWhole = Math.floor(Math.random() \* 20); // Note that the Math.random() has a range of [0,1) , thus, Math.random() \* 20 has a range of [0,20) and if we do Math.floor() of that range then we get a random whole number in the range [0,19].

console.log(randomWhole);

/\* --------------------------GENERATE RANDOM WHOLE NUMBERS WITHIN A RANGE ------------------------\*/

function randomRange(myMin, myMax){

var randomWhole = Math.floor(Math.random() \* (myMax - myMin +1)) + myMin;

return randomWhole;

}

var randomWhole = randomRange(5,15); // Note that, here, our function "randomRange" returns a random whole number in the range [myMin,myMax] , notice that both the end points are included.

console.log(randomWhole);

/\* ----------------------------------------"parseINT" FUNCTION -----------------------------------\*/

function convertToInteger(str){

return parseInt(str); // "parseInt()" takes a string and converts it into its integer value. If "parseInt()" is not able to convert the string into integer then it returns a value "NaN" (Not a Number).

}

var intValue = convertToInteger("98");

console.log(intValue);

/\*------------------------------------ parseInt() with a Radix------------------------------------\*/

// Radix is the base of the number that we want to convert.

function convertToInteger(str){

return parseInt(str,2); // '2' here represents the base of the number that needs to be converted. By default the value of radix is 10.

}

var intValue = convertToInteger("1011");

console.log(intValue);

/\* ----------------------------------------TERNARY OPERATOR---------------------------------------\*/

// Ternary Operator works as follows:

// condition ? statement-if-true : statement-if-false

function checkEqual(a,b){

return a===b? true : false;

}

/\* ---------------------------------MULTIPLE CONDITION TERNARY OPERATOR---------------------------\*/

function checkSign(num){

return num > 0 ? "positive" : num < 0 ? "negative" : "zero";

}

console.log(checkSign(-1))