Eating, wiring

(javascript)

Bricks, paing

(CSS)

House Plan

(HTML)

house

Use of Firefox inspect tools

To clear the console clear() and enter

**Javascript code to change background at every click**

const htmlBody = document.querySelector('body');

const randomClickFunction = function(){

const colors = ["green", "orange", "grey", "red", "blue" , "green"];

const randomIndex = Math.floor(Math.random() \* colors.length);

const randomColor = colors[randomIndex];

htmlBody.style.backgroundColor = randomColor;

console.log('The user clicked and the color is set to ' + randomColor);

}

htmlBody.onclick = randomClickFunction

Simple button modifications and application of simple javascript

<!DOCTYPE html>

<html lang="en">

<head>

 <meta charset="UTF-8">

 <style>

  button{

   background-color: transparent;

   border: 1px solid navy;

   padding: 20px;

   font-size: 1.4rem;

   border-radius: 10px;

  }

  button:hover{

   background-color: navy;

   border: none;

   color: white;

  }

 </style>

 <meta http-equiv="X-UA-Compatible" content="IE=edge">

 <meta name="viewport" content="width=device-width, initial-scale=1.0">

 <title>Example</title>

</head>

<body>

 <button>CLICK ME</button>

 <div class="container"></div>

 <script>

  function onClickEvent(){

   const el = document.createElement('p');

   el.innerText = 'Clicked the button';

   document.querySelector('.container').appendChild(el);

  }

  document.querySelector('button').onclick = onClickEvent;

 </script>

</body>

</html>

**JavaScript Variables and Data Types**

const myArray = ["10", "20", "30", "40", "50"];

myArray

Array(5) [ "10", "20", "30", "40", "50" ]

myArray.push("70")

6

myArray

Array(6) [ "10", "20", "30", "40", "50", "70" ]

myArray.reverse()

Array(6) [ "70", "50", "40", "30", "20", "10" ]

Use console.log() to print something in console

One line comment //

Multiline comment /\* \*/

**Variables and datatypes**

Variable declaration and assignment

const firstVar = 'Amitha Shehan';

const secondVar = 50;

const thirdVar = {firstProperty: 'hello world'};

console.log(firstVar)

console.log(secondVar)

console.log(thirdVar)

declaring variable first and then assigning value to it

let newVar;

newVar = 50;

console.log(newVar)

**Accepted declaration and assigning**

Const var1 = 50; //cannot be reassigned/redeclared

Let var2 = 90; //used for reassignment

Var var3 = 50; // this is a legacy method. Do not recommend using this. Var can be redeclared

Use const when we have values not to change as the program runs

Use let when we assign new values to variables in the long run

Use capslock when defining const variables

const TAX\_RATE = 0.08;

**Use of const and function**

const variable1 = 10;

//const variable2 = variable1;

const variable3 = "Amitha";

const variable4 = {variableType:"object",variableValue:"somevalue"};

const variable5 = (function(){

return "Hello , my name is ";})();

const variable6 = variable5 + variable3;

**Boolean variables**

Const variable7 = false

**To get to know the datatype of a declared variable**

typeof variable1;

javascript is a dynamically typed language which means we do not need to define the variable type (number, string, Boolean) when we declare a variable.

Converting a string number and add it to a number

Both single quotes and double quotes can be used in string declaration. It is recommended to use single quotes

When assigning values to a number do not use commas

**Arrays**

Varieties of data can be included into arrays’

const array2 = [10, 'string 1', {prop:'sdkfjd'}, [10,20,330]];

console.log(array2[0]);

console.log(array2[1]);

console.log(array2[2]);

console.log(array2[3]);

console.log(array2[3][2]);

**Objects**

const objectVariable = {prop1: 20, prop2:50};

to access a certain property

objectVariable.prop2;

or

objectVariable[‘prop1’]

Nested object declaration and access to them using dot notation

const nestedVariable = {

layer1: {

layer2:{

layer3:{

targetValue: 20}}}}

nestedVariable.layer1.layer2.layer3.targetValue

**Functions**

Defining a function

const functionContainerVariable = function(){

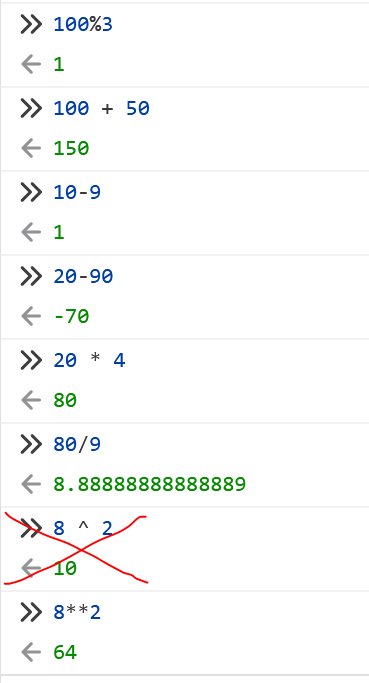
return 20;

}

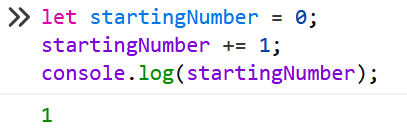
Calling a function

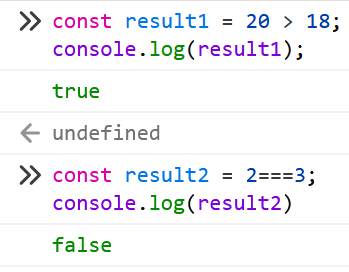
functionContainerVariable()

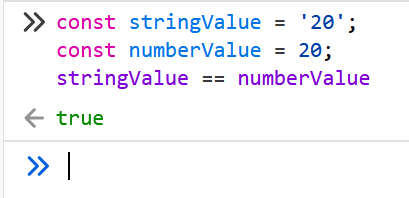
**Javascript operators**

Arithmetic operators

Increment and decrement operator does not work on const variables, but works on let.

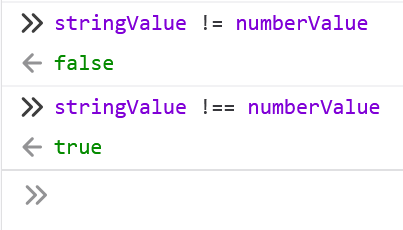


**Comparison operators**



JS converts both variables in to one type and then compares them, we get false if we try

stringValue === numberValue;

double equal does not compare data type, but triple equal compares both data type and value

If we compare two arrays, objects with different names but content is same, when we == or === them answer is false

We cannot use equal operators == or === on arrays and objects.

To do so we can use library called lodash.

**Ternary operator**

const result = 20 === 20 ? 'values match': 'values does not match';

Ternary operator is same as if condition

let resultVariable;

if(20 == 20){

resultVariable = 'values match';

}

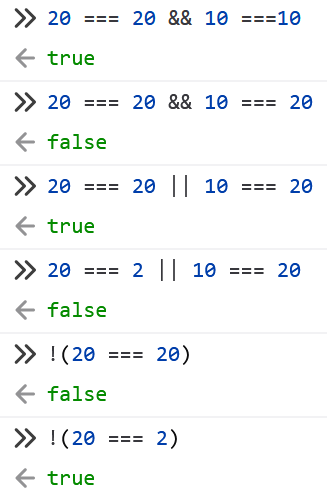
else{

resultVariable = 'values do not match';

}

console.log(resultVariable);

**Logical Operators**



**Conditions**

**If condition if(){} else{}**

if('some string' === 'another string'){

console.log('Strings are equal');

}else{

console.log('strings are not equal');

}

**Switch-case**

const colors = ["red", "green", "blue", "orange"];

const randomIndex = Math.floor(Math.random() \* colors.length);

const randomColor = colors[randomIndex];

console.log(randomColor);

switch (randomColor) {

case "orange":

console.log("color is orange");

break;

case "blue":

console.log("color is blue");

break;

case "green":

console.log("color is green");

break;

case "red":

console.log("color is red");

break;

default:

console.log('no color found');

}

**Looops**

for (let i = 0; i < 10; i++){

console.log(i);

}

To print only numbers in a array

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

if (typeof array[i] == 'number') {

console.log(array[i]);

}

}

Arrays, Objects and loops in one code

const myArray1 = [

{

name: "amitha",

age: 29,

},

{

name: "kasun",

age: 19,

},

{

name: "John",

age: 24,

},

{

name: "kane",

age: 89,

},

];

for (let i = 0; i < myArray1.length; i++){

const theName = myArray1[i].name;

const theAge = myArray1[i].age;

console.log(theName + ' is ' + theAge + ' years old')

}

**Functions**

Function myFunction(){content}

After defining a function we need to execute, call or invoke the function

function myFunction() {

console.log('hello world this is my first javascript function');

}

myFunction();

The way to call the function just after defining the function (immediately invoked function) (declaring and execution at the same time)

**(function myFunction(){content})()**

(function myOtherFunction() {

console.log('declared and executed in same line ');

})();

**Parameters and Arguments**

function myFunction1(param1, param2) {

console.log(param1);

console.log(param2);

}

myFunction1('argument 1', 'argument 2');

**Another way to declare function in JS**

const anotherFunction = function () {

console.log('another function');

}

anotherFunction();

This is an anonymous function. The function does not have a name, but it is assigned to a variable.

**Arrow Function**

const arrowFunction = () => {

console.log('I am arrow function');

}

arrowFunction();

**Return statement in functions**

const myFunction = () => {

const a = 15;

return a;

}

console.log(myFunction());

we can return Boolean values and functions as well.

We should not put any line of code after return statement

It’s not going to be read after return statement

const myNumber = 50; // defining a global variable

let myFunction = () => {

if (myNumber < 25) { //accessing global variable

return 'the number is less than 25';

}

return 'the number is greater than 25';

}

console.log(myFunction());

**Assigning a variable to a function**

function myFunction() {

return 50;

}

console.log(myFunction());

const aliasVariable = myFunction;

console.log(aliasVariable() + 20);

**Assigning a variable to an object**

function myFunction() {

return 50;

const myObj = {

prop1: 20,

prop2: myFunction

}

console.log(myObj.prop2());

When a function is bound to an object it is called a method.

const myName = 'Amiths';

console.log(myName.replace('s', 'a'));

console.log(myName);

const myCorrectName = myName.replace('s', 'a');

console.log(myCorrectName);

We cannot change myName variable, but instead we can use replace method to assign a new variable.

We can call methods one upon another

const str = 'Hello World';

console.log(str.toUpperCase().split(" ").indexOf('WORLD'));

**Built in JavaScript utility methods**

**Callback functions**

function myCallback(someNumber){

return someNumber \* 2;

}

function mainFunction(randomNumber, shouldCall, callback){

let result = randomNumber;

if(shouldCall){

result = callback(randomNumber)

}

return result;

}

mainFunction(20,true,myCallback);

or without declaring the function in a separate block we can include it in the mainFunction calling block

mainFunction(20, true, function(num){

return num\*2;

})

We can use arrow function as well.

mainFunction(20, true, (num) => {

return num \* 2;

})

Another way

mainFunction(20, true, num => num \* 2)

Reason for using call back functions

\* Reusablility

\* Asynchronous programming

The map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

---------------------------------------------------------------------------------------

const myArray = [2, 4, 6, 8];

function myCustomMapOperationCallback(eachElementFromArray){

return eachElementFromArray \* 2; //trasformed value

}//map function creates a new array it cannot modify existing array

//.map is the built in function for array

const myNewArray = myArray.map(myCustomMapOperationCallback);

console.log(myNewArray);

const strArr = ['amitha', 'shehan', 'perera'];

const newStrArr = strArr.map(eachElementFromArray => eachElementFromArray[0])

console.log(newStrArr)

Array(3) [ "a", "s", "p" ]

---------------------------------------------------------------------------------------

basic idea is we can use .map function over and over again

Primitives

everything in JS is an object.

--------------------------------------------------------------------------------------

const string1 = new String('hello, world!'); // this is an object, not a string, new is an operator

const string2 = 'hello, world!'; //this is a primitive

console.log(string1 === string2); //false

console.log(string1 == string2); //true

--------------------------------------------------------------------------------------

false

true

--------------------------------------------------------------------------------------

const myDate = new Date(); //creates a new instance of Date object (taking a copy)

javascript Date (refer zack gollwitzer guide - a complete guide to javascript dates)

--------------------------------------------------------------------------------------

const myDate = new Date();

myDate

myDate.toString()

myDate.toISOString()

myDate.getMonth()

myDate.getFullYear()

--------------------------------------------------------------------------------------

Javascript Expressions

knowing a wrong email address

used for validation

pattern matching

--------------------------------------------------------------------------------------

const emailValidatorRegEx = new RegExp('^.+@.+\..+$');

const userInput = 'invalidemail@g';

const isValid = emailValidatorRegEx.test(userInput);

console.log(isValid);

--------------------------------------------------------------------------------------

^ - start at the beginning of text

. - all characters

+ - i want to match any character one or more time

@ - looking to @ symbol

+ - i want to match any character one or more time

\ - escape

. - look for a period

. - any characters

+ - i want to match any character one or more time

$ - end of the line

Regular Expressions is a combination of identifiers and quantifiers

identifiers - can be special characters, or phraces that we are trying to match

--------------------------------------------------------------------------------------

const regex1 = RegExp('g');

const string1 = 'my favorite food is steak';

const string2 = 'my favorite thing to do is code';

regex1.test(string1)

--------------------------------------------------------------------------------------

false

we can compare not only characters but also phraces like 'favorite'.

this is case sensitive

common way of writing regular Expressions

/favorite/ test(string1)

comparing character groups

/[a-z]/.test(string1) //compares simple letters from a to zack

[A-Z] compares caplital letters from A - Z.

[0-9] compares digits 0-9

[A-Z0-9a-z ] compares capital letters + simple letters + digits + spaces

[\d] same as [0-9]

[\s] white space characters

[\w] all letters and numbers

/^f/ first letter of the text f(if is f true otherwise false)

/f$/ first letter of the text f(if is f true otherwise false)

/(code | steak)/ match either code or steak

Quantifiers

/[a-z]/.exce(string1) returns Array ["m"] because first letters matches

/[a-z ]{5}/exec.(string1) retur first five characters including space

Flags

const str = 'hello world, 2021 @ more of a string'; //to match the following string

/^[a-z ]+,[0-9 ]+@[a-z ]+$/.exec(str)

or

/.+/.exec(str) // this checks any character one or more times

String methods

we will be covering replaceAll, toUpperCase, substring, trim, match

const myString = 'My Dog jumped on the bed, my dog is a bad Dog';

const newString = myString.replaceAll('Dog', 'cat');

console.log(newString);//this is case sensitive

//so lets use regular expressions

const correctStr = myString.replaceAll(/[Dd]{1}og/g,'cat') // g is the global flag

console.log(correctStr);

str.toUpperCase();

str.substring(indexStart, [indexEnd]) // indexEnd is optional.

to capitalize first letter in a sentence

const myString = 'hello world'

const result = myString[0].toUpperCase() + myString.substring(1, myString.length);

Trim method removes white spaces from both ends not in the middle, use full when removing white spaces in start and end from data from external sources

Array methods

pop removes the last element in the array

const arr = [1,2,3,4,5]

arr.pop()//removed 5 from array

push adds element(s) to the end of the array

arr.push(5,6,7)

//arr = [1,2,3,4,5,6,7]

unshift adds elements to the beginning of the array

arr.unshift(0)// arr = [0,1,2,3,4,5,6,7]

slice creates a basic copy(not nested elements) of an array slide(start,[end])

const copyArr = arr.slice();//copyArr = [0,1,2,3,4,5,6,7]

const copyArr2 = arr.slice(3) //copyArr2 = [3,4,5,6,7]

The splice() method changes the contents of an array by removing or replacing existing elements and/or adding new elements in place

To create a new array with a segment removed and/or replaced without mutating the original array, use toSpliced(). To access part of an array without modifying it,

splice(start)

splice(start, deleteCount)

splice(start, deleteCount, item1)

splice(start, deleteCount, item1, item2, itemN)

const months = ['Jan', 'March', 'April', 'June'];

months.splice(1, 0, 'Feb');

// Inserts at index 1

console.log(months);

// Expected output: Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');

// Replaces 1 element at index 4

console.log(months);

// Expected output: Array ["Jan", "Feb", "March", "April", "May"]

findIndex() method

The findIndex() method returns the index of the first element in an array that satisfies the provided testing function. If no elements satisfy the testing function, -1 is returned.

we can use call back functions in findIndex() method.

the find() method, which returns the first element that satisfies the testing function (rather than its index).

const array1 = [5, 12, 8, 130, 44];

const isLargeNumber = (element) => element > 13;

console.log(array1.findIndex(isLargeNumber));

// Expected output: 3

const moreComplexArray = [

{

firstName: 'Bob',

lastName: 'Smith'

},

{

firstName: 'Alice',

lastName: 'Smith'

},

{

firstName: 'Jon',

lastName: 'Smith'

},

{

firstName: 'Jon',

lastName: 'Smith'

}

];

//we cannot use indexOf method in array + objects

const moreComplexArray = [

{

firstName: "Bob",

lastName: "Smith",

},

{

firstName: "Alice",

lastName: "Smith",

},

{

firstName: "Jon",

lastName: "Smith",

},

{

firstName: "Jon",

lastName: "Smith",

},

{

firstName: "Jon",

lastName: "Doe",

}

];

console.log(

moreComplexArray.findIndex((arrItem) => {

return arrItem.lastName !== "Smith";

})

);

map method

The map() method creates a new array populated with the results of calling a provided function on every element in the calling array.

const array1 = [1, 4, 9, 16];

// Pass a function to map

const map1 = array1.map(x => x \* 2);

console.log(map1);

// Expected output: Array [2, 8, 18, 32]

to change the category of all blogposts to webdev

const blogPostsFromDatabase = [

{

title: 'How to use the map() function',

category: 'uncategorized'

},

{

title: 'What is JavaScript',

category: 'uncategorized'

},

{

title: 'Why are you crazy enough to learn to code?',

category: 'uncategorized'

},

];

function changeCategory(arrayItem) {

arrayItem.category = 'Web Development';

return arrayItem;

};

const newDBArray = blogPostsFromDatabase.map(changeCategory);

console.log(newDBArray);

//or we could use

blogPostsFromDatabase.map(arrItem => {

arrItem.category = 'Web Development';

return arrItem;

})

forEach() method

the forEach() method executes a provided function once for each array element. just like for loop

const array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

// Expected output: "a"

// Expected output: "b"

// Expected output: "c"

const array1 = [1, 2, 3, 4, 5];

array1.forEach(element => console.log(element));

includes() method

The includes() method determines whether an array includes a certain value among its entries, returning true or false as appropriate.

const array1 = [1, 2, 3];

console.log(array1.includes(2));

// Expected output: true

const pets = ['cat', 'dog', 'bat'];

console.log(pets.includes('cat'));

// Expected output: true

console.log(pets.includes('at'));

// Expected output: false

filter() method

The filter() method creates a shallow copy of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);

// Expected output: Array ["exuberant", "destruction", "present"]

const allOrders = [

{

productName: "Tea Pot",

isDigital: false,

isCancelled: false,

isOpen: false,

},

{

productName: "Mens hoodie",

isDigital: false,

isCancelled: true,

isOpen: false,

},

{

productName: "Coding book",

isDigital: true,

isCancelled: true,

isOpen: false,

},

{

productName: "Atomic habits book",

isDigital: true,

isCancelled: false,

isOpen: false,

},

];

const digitalOrders = allOrders.filter((arrItem) => {

return arrItem.isDigital;

});

console.log(digitalOrders);

const cancelledDigitalOrders = allOrders.filter(arrItem => {

return arrItem.isDigital && !arrItem.isCancelled

})

console.log(cancelledDigitalOrders);

reduce() method

The reduce() method executes a user-supplied "reducer" callback function on each element of the array, in order, passing in the return value from the calculation on the preceding element. The final result of running the reducer across all elements of the array is a single value.

The first time that the callback is run there is no "return value of the previous calculation". If supplied, an initial value may be used in its place. Otherwise the array element at index 0 is used as the initial value and iteration starts from the next element (index 1 instead of index 0).

Perhaps the easiest-to-understand case for reduce() is to return the sum of all the elements in an array:

simple example is to take the sum of an array

const array1 = [1, 2, 3, 4];

// 0 + 1 + 2 + 3 + 4

const initialValue = 0;

const sumWithInitial = array1.reduce(

(accumulator, currentValue) => accumulator + currentValue,

initialValue

);

console.log(sumWithInitial);

// Expected output: 10

//another way

const array = [1, 2, 3, 4];

function reducerCallBack(sum, arrItem) {

return sum + arrItem;

}

console.log(array.reduce(reducerCallBack, 0));

//another way to write callback function

array.reduce((sum, arrItem) => sum+=arr,0)

we cannot use == or === to check similarity between two objects

const obj1 = { prop1: 'some Value' };

const obj2 = { prop1: 'some Value' };

console.log(obj1 === obj2);

console.log(obj1 == obj2);

//answers we get are

false

false

we can use isEqual() method from lodash library to check similarity between objects

Math object

no need create(cannot) instances

Math.PI //return pi value

Math.abs(-60) //return 60

Math.ceil(Math.PI) //return 4

Math.max(2,5,8) //return 8

Math.random() // return a random value between 0 & 1

//generating a random value between 0-100

/\*

const randomNumber = Math.random();

const randomNumberTo100 = randomNumber \* 100;

const randomNumberFloor = Math.floor(randomNumberTo100);

console.log(randomNumberFloor);

\*/

//all in one line

const randNum = Math.floor(Math.random() \* 100);

console.log(randNum)

const myArray = [2, 5, 6, 8, 1.2, 23];

const randomIndex = Math.floor(Math.random() \* myArray.length);

console.log(myArray[randomIndex]);

javascript errors

3 main types of errors

1.reference error

2.syntax error

3.type error

reference error

The ReferenceError object represents an error when a variable that doesn't exist (or hasn't yet been initialized) in the current scope is referenced.

syntax error

The SyntaxError object represents an error when trying to interpret syntactically invalid code

type error

The TypeError object represents an error when an operation could not be performed, typically (but not exclusively) when a value is not of the expected type.

A TypeError may be thrown when:

an operand or argument passed to a function is incompatible with the type expected by that operator or function; or

when attempting to modify a value that cannot be changed; or

when attempting to use a value in an inappropriate way.

ex- when we add 2 objects

Error Handling

try {

var num = 45;

num.toUpperCase();

}

catch (error) {

console.log(error instanceof TypeError);

console.log(error.message);

}

//answer is

true

num.toUpperCase is not a function

//NaN, null, undefined

we get NaN when we are going to convert a string into a number or doing math operations on it

const myString = 'some string';

Number(myString) //NaN

null

const num = null; //null has a false value

if(num){

console.log(this will not be reached)

} else{

console.log(this will be reached)

}

undefined

just declaring a variable and not assigning a value

works same as in if condition

to extract digits in a number

const num = 299;

+num.toString()[0];