```
//########################## JAVASCRIPT NOTES ##################################

// AUTHOR : Devansh Singh Rathore (CSE, IIT Palakkad)
// SOURCE : Javascript Fundamentals for Absolute Beginners 2018
//        (CodeOnce, Youtube), by Bob Tabor

//#############################################################################

//TOP JAVASCRIPT FRAMEWORKS ::
//1. AngularJS (2.0 to 4)
//2. ReactJS
//3. Meteor.js
//4. Node.JS
//5. EmberJS

//IMPORTANT LINKS & WEBSITES ::
//0. https://nodejs.org/en
//1. https://caniuse.com :
//    Lets us know about Browser support tables for different modern
//    web techniques.
//2. https://babeljs.io :
//    Its to let you know how syntax for a command code in another
//    version would look like.
//3. https://deal.codetrick.net
//    for general info. on node.js, react.js, etc. and courses

//SETTING UP THE DEVEOPMENT ENVIRONMENT ::
//Steps for Ubuntu:
//1. Open Terminal
//2. Type node
//3. If it doesn't recognizes type : "sudo apt install nodejs"

//Trial Code :
console.log('Javascript is case sensetive!');
//This particular code is to print a message on console.

//COMMAND TO RUN A JAVASCRIPT FILE ::
//eg. node abc.js

//BASIC JAVASCRIPT SYNTAX ::
//We expect a Javascript code to interact with HTML elements on a webpage,
//or to write video games, animations.

//There's a difference between the language itself and the encironment it
//runs on.

//Javascipt is a case sensetive language.

let x=3;
let y=7;
let z=x+y;
console.log('Answer after addition : ' + z); //output> Answer after addition : 10

//'let' keyword helps us to initialize a variable(identifier).

//The plus'+' used in the log code is actually concatenating the string
//with the value of z.

//VARIABLES ::
var x=3; //output> error //on my terminal                    ***
x=5;

//'var' is another keyword that works exactly same as 'let'(newer one).
//But it certainly can be problamatic in some special cases, that is why
//we generally avoid to use it.
```

```
const m=7;
let myName = 'Devansh';

//'const' keyword is to set a constant which can't be changed in a program.
//we can use 'let' for a particular variable only once in its scope.

let n;
console.log(n); //output> undefined
console.log(typeof n) //output> undefined
n=8;

//we can even declare a variable without initialization, but printing it
//would result in 'undefined'.

//All variables(or identifiers) can begin with letter, '_', or '$'.
//Variables can have letters, numbers, '$', '_', but nothing else, no space.
//Moreover we can't use any keyword as a variable.
//Try to use camelCasing in variable names.

X=8;

//Excitingly, using a variable without 'let' declaration/initialization,
//doesn't throws error unless it is a part of some print message, calculation.

//COMMENTS ::
//For single line comments: '//'
//For multiple line comments: '/*'--text--'*/'

//DATATYPES ::
//What makes Javascipt different from other programming languages is that,
//variables doesn't have any datatype but the datatypes are assigned to the
//values that the variables obtain.

let w=12;
let e=true;
console.log(typeof w); //output> number
console.log(typeof e); //output> boolean
console.log(typeof myName); //output> string

//For string it needs to be in single or double quotes.
//Type of uninitialized(can be declared) variables is also undefined.

let a=1;
let b='3';
let c=2;
let d=4;
let D=a+b+c;
console.log(typeof a, typeof b, typeof c) //output> number string number
let E=a+c+b;
console.log(typeof a, typeof b, typeof c) //output> number string number
console.log(typeof D, typeof E) //output> string string
console.log(D) //output> 132
console.log(E) //output> 33
let F=a+b+c+d;
console.log(F) //output> 1324

//For printing a valid result in above case Javascipt takes preceeding
//numbers as string to concatenate with following string.

b=parseInt(b, 10) //10 is for decimal
let r=parseInt('coDeSpRinter', 10)
console.log(r) //output> NaN
let r0=isNaN(r)
console.log(r0) //output> true

//Here the command 'parseInt' is for the typecasting operation to int, which
```

```
//takes a variable and the radix of the number it is to be typecasted to.

//'NaN' will appear if try to typecast a non-number value using 'parseInt'.

//We can use 'isNaN' command to verify if a variable is NaN or not.

//There is nothing like 'parseBoolean' to typecast into boolean.


//*** -> command line might produce error.
//################################################################################

//KEY POINTS::
//1. Javascript doesn't requires indentation for different scopes.
//2. Writing two different console.log() prints out ouput in two different lines.
//3. Check out, what does 'delete' keyword is used for in Javascript.

//EXPRESSIONS AND OPERATORS::
let b=2;
let c=3;

//Three expressions here......can you find them?
let a=b+c;

//Answer->
//1. Let a ... variable declaration.
//2. Perform evaluation b+c.
//3. Result assigned to a.

//Operators:
//1. Assignment -> '='
//2. Arithematic -> '+', '-', '*', '/', '%'
//3. Increment/Decrement -> '++', '--'

let d=4;
let e=d%3;
console.log(d++); //output> 4
console.log(e); //output> 1
console.log(++d); //output> 6

//4. String operators -> '+', ' '' ', ' "" '
//5. Precedence -> () //for setting up the arithematic operations according to
BODMAS

let f = 1+2*3;
let g = (1+2)*3;
console.log(f,g); //output> 7 9

//6. Function Invocation operators -> ()
//7. Logical -> and: &&, or: ||
//8. Member Accessor operators -> '.'
//9. Code Block operators -> {}
//10. Array element access operators -> []

//ARRAYS::
let h=[1,3,5,7,9];
let i=['coDeSpRinter', 'iamrakesh28', 'mr.convict', 're_cursed', 'tymefighter',
'wildcat'];

//Indexes in Javascript are 0 based.

console.log(h); //output> [1, 3, 5, 7, 9]
console.log(i[0]); //output> coDeSpRinter
console.log(typeof h); //output> object
h[3]=8;
```

```
//To change the values in array(no worry of copy(deep/shallow)).
//Arrays are of 'object' datatype.
//Arrays can include elements of different datatypes.

let j=[1,'DSR08', true];
console.log(k[5]); //output> undefined                          ***
console.log(j.length); //output> 3

//We have '.length' property to give out length of array.

j[8]=10;
console.log(j); //output> [1, 'DSR08', true, , , , , , 10]
console.log(j.length); //output> 9
console.log(j[6]) //output> undefined

//Such an array with some empty elements is called sparse array.
//Printing array element which is unspecified, outputs undefined.

i.pop();
console.log(i); //output> ['coDeSpRinter', 'iamrakesh28',..., 'tymefighter']
i.push('vjac');
i.push('wildcat');
console.log(i.length); //output> 7

//Pop operation removes the last element of the array.
//While the push operation adds new elements at the back of the array.

//FUNCTIONS IN JAVASCRIPT::
//Function declaration:

function Intro()
{
      console.log(i);
//When the function is called:
//output>
[ 'coDeSpRinter','iamrakesh28','mr.convict','re_cursed','tymefighter','vjac','wildcat'
]
}

Intro(); //function call
let l=Intro;
l();

//The above mentioned way is another way of function call.

function quote(name)
{
      console.log("Play"+name);
}
quote(' cool !'); //output> Play cool !
quote(' it hard'); //output> Play it hard

//We don't have to specify any datatype in argument of function in Javascript.

function theBestCoder(m)
{
      return m[1];
}
let n=theBestCoder(i);
console.log(n); //output> iamrakesh28

//We can pass arrays in functions.
//Use 'return' keyword for returning any value.

//FUNCTION EXPRESSIONS::
setTimeout(function (){console.log('Wait for 3 seconds');},3000);
```

```
//output> Wait for 3 seconds(after 3 seconds)

//'setTimeout' is used to execute a function(1st argument) after the
//provided t milliseconds(2nd argument).

let count=0;
function virtualTime()
{
      setTimeout(function(){                          //output> Hi 0
            if(count<3)                             //     Hi 1
            {                                       //     Hi 2
                  console.log("Hi " + count++);
                  virtualTime();
            }
      },2000);
}
virtualTime();

(function(){
      console.log("IIFE!!"); //output> IIFE!!
})();

//Immediately Invoked Function Expressions(IIFE) is a method to automatically
//execute a function in Javascript without any function call.
//Notice the function is surrounded by '(',')' and then the function operator '()'
//is seperately used.

//DECISION STATEMENTS::
//if, else, ternary operators, switch statements

let o=5;
if(o%2==0) console.log("5 is Even");
else if(1==true) console.log("Surprize.."); //output> Surprize..
else console.log("5 is Odd");

//switch use 1:
let hero='xman'
switch(hero)
{
      case 'superman':
      console.log("x-ray vision");   //output> hand blades
      case 'xman':                   //        Its a super-hero
      console.log("hand blades");
      default:
      console.log("Its a super-hero");
}

//without breaking the cases, we execute every case after the dezired case.
//'default' is for the default case if no other case is executed.

//switch use 2:
let he='Xman'
switch(he.toLowerCase())
{
      case 'superman':
      console.log("x-ray vision");   //output> hand blades
      break;
      case 'xman':
      console.log("hand blades");
      break;
      default:
      console.log("Its a super-hero");
      break;
}

//'break' statements are used to exit the smallest code block{}.
```

```
//The '.toLowerCase()' function is to convert the argument into lowercase.

let p=1, q='1';
let res = (p==q) ? 'equal' : 'not equal';
console.log(res) //output> equal
let re = (p===q) ? 'equal' : 'not equal';
console.log(re) //output> not equal
let r = (p!==q) ? 'not equal' : 'equal';
console.log(r) //output> not equal

//These are called ternary operations.
//The operator '==' allows Javascript to compare two values, and if they are not of
//same datatype, then javascript can even typecast them into same datatype and
compare.
//But '===' is strict comparison and doesn't allows typecasting.
//Similar is the use of strict inequality '!==' operator.


//*** -> command line might produce error.
//#############################################################################

//KEY POINTS::
//1. Javascript code runs fine without even ending some code lines with ';'.
//2. 'Namespacing' is a technique employed to avoid collisions with other objects
or variables
//in the global namespace. They're also extremely useful for helping organize
blocks of
//functionality in your application into easily manageable groups that can be
uniquely identified.
//3. JSON: JavaScript Object Notation, and is used to send information between two
different
//user systems.

//ITERATION STATEMENTS::
//for, while, do_while loops

let a=[1,3,5,7,9];
for(i=0;i<a.length;i++)
{
      console.log(a[i]);
}

//There is no need to use 'let' keyword in initialization of loop index.
//Rest of the for/While loop syntax is same as that in C/C++.

let b=1;
while(b<10)
{
      console.log(b++);
      if(b==8)break;
}

//Its not compulsory to use code block {}, for single line commands, according
//to dependencies and its scope.
//Similar to this is the use of do_while loops in Javascript.

//BASICS OF SCOPE::
let c=c0='Ahmed';
function scopeTest()
{
      console.log(c0); //output> Ahmed
      let c='Himanshu';
      console.log(c); //output> Himanshu
}
scopeTest();
console.log(c); //output> Ahmed
```

```
//The scope of c defined inside the function is within {}, and outside that, it
//doesn't holds valid.
//While for a global variable, the scope includes complete program.
//Variable defined in outer scope can be accessed/modified in inner scope.

//RETURNING FUNCTIONS FROM FUNCTIONS::
//In web developement, we avoid declaring global variables and functions, as that
is
//considered to be a bad practice. Instead we try to return functions from
functions!!.

//DON'T DO THIS EVER:
let d='I am a global variable, avoid Me!';
console.log(d);
let e=function(){console.log('Me too!');}

function one(){return 'one';}
let f=one;
let g=one();
console.log(typeof f); //output> function
console.log(typeof g); //output> string

//This happened because f has just become another name for the function which can
be
//used in function call, but g is the value returned by the function.

function three()
{
      return function(){
            console.log("returning function!");
      }
}
let h=three();
h(); //output> returning function!

//In Javascript, we can even return a function itself.

function two()
{
      return function(){
            return "Its 2 bro..";
      }
}
console.log(two()); //output> [Function]
console.log(two()()); //output> Its 2 bro..

//OBJECT LITERALS::
let person ={
      college: 'IIT Palakkad',      //specifically defined object literal syntax
      name: 'BGP',
      subject: 'Maths',
      favouriteStudent: 'AhmedZeDdy',
      classAverage: function(){
            let AhmedMarks=20;
            return AhmedMarks-5;
      },
      intro: function(){
            console.log(this.name + ', ' + this.subject);
      }
}
console.log(person.college); //output> IIT Palakkad
console.log(person.classAverage()); //output> 15
person.intro(); //output> BGP, Maths
console.log(person['favouriteStudent']); //output> AhmedZeDdy
```

```
//Object literals is same as class in C/C++.
//To define a class variables/functions, we need to use ':' operator, and seperate
out all
//the variables by ','.
//We use '.this' to refer to the class object.
//Note that we can also call any variable/method of object by: object['variable/
method_name'].
//We always define a object within {}, which is its identity.

let friends={};
friends.names = ['AZD', 'DSR', 'HJ'];
console.log(friends.names[2]); //output> HJ

//Its another method to define object variables/methods after declaration.
//Note that, here we use '=', instead of ':' to assign a value to object variable.

let college = { student: { branch: 'CSE'}}
console.log(college.student.branch); //output> CSE

//This procedure is called chained or nested classes, which is somewhat used in
something
//called 'namespace' in Javascript.
//Functions defined inside object literals are generally termed as 'methods'.

let q={
      myQuote: [
            {q0:'Play ', user:'DSR08'},
            {q1:'Cool!', user:'re_cursed'}
      ]
}
console.log(q.myQuote); //output> [ { q0: 'Play ' }, { q1: 'Cool!' } ]

//Javascript allows nested objects inside arrays and other such hierarchies.


//*** -> command line might produce error.
//#############################################################################

//KEY POINTS::
//1. Disadvantages of using Global variables:
  //a. Each variable that you define in global scope is not going to be removed
from
  //computer's memory until the web browser tab navigates to new webpage. The more
we
  //add into the global scope, the more memory its going to consume, while that
tab is
  //open for a particular webpage.
  //b. If we merge codes, or by mistake use a declared global variable as a new
variable
  //in a function, then collision might occur.
//2. We prefer to use 'let' keyword over 'var' keyword for variable declaration.
//3. The "use strict" directive was new in ECMAScript version 5. It is not a
statement,
//but a literal expression, ignored by earlier versions of JavaScript. The purpose
of
//"use strict" is to indicate that the code should be executed in "strict mode".
With strict
//mode, you can not, for example, use undeclared variables.
  //eg. "use strict";
  //  x=3.14; //this will cause an error, because x is not defined.

//MODULE PATTERN AND REVEALING MODULE PATTERN::
let counter=(function(){
      //private stuff
      let count=0;
      function print(message){
```

```
            console.log(message+'==>'+count);
      }
      //return an object
      return {
            value: count,
            get: function(){return count;},
            set: function(v){count = v;},
            increment: function(){
                  count+=1;
                  print('After increment: ');
            },
            reset: function(){
                  print('Before reset: ');
                  count=0;
                  print('After reset: ');
            }
      }
}
)();
console.log(counter.count); //output> undefined
console.log(counter.value); //output> 0
counter.increment(); //output> After increment: ==>1
counter.increment(); //output> After increment: ==>2
console.log(counter.value); //output> 0
counter.reset(); //output> Before reset: ==>2
                 //        After reset: ==>0

//This model returns an object from IIFE, and this technique is called 'module
pattern'.
//Note the use of 'print' defined function and not 'console.log()'.
//We cannot access private variables like 'count' outside by using
'counter.count', it
//will be undefined as we return only the object containing 'value',
'increment','reset'
//to the counter.
//Note that value of 'counter.value' even after increment is 0 only, because we
//accidently created something called 'Closure' in javascript.

counter.set(8);
console.log(counter.get()); //output> 7
counter.reset(); //output> Before reset: ==>7
                 //        After reset: ==>0

//Other than this, there is something called 'revealing module'.

let reveal=(function(){
      //private stuff
      let cnt=0;
      function print(msg){
            console.log(`${msg}==>${cnt}`); //note the way of using string.
      }
      function getCnt(){return cnt;}
      function setCnt(v){cnt=v;}
      function incrementCnt(){
            cnt+=1;
            print('After increment: ');
      }
      function resetCnt(){
            print('Before reset: ');
            cnt=0;
            print('After reset: ');
      }

      //return an object
      //'reveals' the public functions
      return {
```

```
            get: getCnt,
            set: setCnt,
            increment: incrementCnt,
            reset: resetCnt
      }
})();
console.log(reveal.get()); //output> 0

//This model is 'revealing model' as here we reveal public functions through
certain
//objects returned to it.

//CLOSURES::
//A closure is a feature in JavaScript where an inner function has access to the
outer
//(enclosing) function’s variables — a scope chain.

//The closure has three scope chains:
//1. It has access to its own scope — variables defined between its curly brackets.
//2. It has access to the outer function’s variables.
//3. It has access to the global variables.

function sayHi(name)
{
      return function(){
            console.log('Long live '+name);
      }
}
let vaibhav=sayHi('vjac')
vaibhav(); //output> Long live vjac

//'this' KEYWORD::
function first(){
      return this;
}
console.log(first()===global) //output> true

//'this' is node's global object, because that's where the first method was called.

function second(){
      'use strict';
      return this;
}
function third(){
      'CSE';
      return this;
}
console.log(second()===global); //output> false
console.log(second()===undefined); //output> true
console.log(third()===global); //output> true
console.log(third()===undefined); //output false

//Here 'use strict' is undefined value and is bound to nothing, hence its
undefined.
//The rule around binding of 'this' keyword changes according to the context, here
//the contexts were 'use strict', 'CSE'.

let myObject={value: 'My object '}
global.value = 'Global object ';
function fourth(name){
//Return something different depending on how we call this method.
      return this.value + name;
}
console.log(fourth()); //output> Global object undefined
console.log(fourth('DSR')); //output> Global object DSR
console.log(fourth.call(myObject, 'DSR')); //output> My object DSR
```

```
console.log(fourth.apply(myObject, ['DSR'])); //output> My object DSR

//Here '.call()', '.apply()' are a built-in function, and will bind the
'this.value'
//from that of 'myObject' function.
//Difference between them is that, 'call()' method takes arguments seperately while
//the 'apply()' method takes arguments as an array.

//Use of 'call()' function explained:
function fifth(){
      console.log(this.firstName + ' ' + this.lastName);
}
let person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
let person1 = {
  firstName:"Deepak",
  lastName: "Sir",
  print: fifth
}
let person2 = {
  firstName:"Chandru",
  lastName: "Sir",
  print: fifth
}
console.log(person.fullName.call(person2)); //output> Chandru Sir.
person1.print(); //output> Deepak Sir
person2.print(); //output> Chandru Sir

//Thus 'this' keyword depends on how a function is called, and the site from which
it
//is called.

//Take a look at: this_keyword.html                                          **

//DESTRUCTURING::
//Method for unpacking the elements of arrays or other such structures.

let marvels=['satvik', 'amit', 'kaushal'];
let s,a,k,b,others;
[s,a,k]=marvels;
console.log(s,a,k); //output> satvik amit kaushal
[b, ...others]=marvels;
console.log(b); //output> satvik
console.log(others); //output> [ 'amit', 'kaushal' ]

//To take all the rest of the elements use '...' with a variable name which will
contain
//array of rest of the elements.

let year, model;
({year, model}={
      make: 'bmw',
      model: '745li',
      year: 2010,
      cost: 5000
});
console.log(year); //output> 2010
console.log(model); //output> 745li

//Just note the use of IIFE in the previous example for destructuring.


//** -> there's a file linked for it.
```

```
//#############################################################################

//KEY POINTS::
//1. Note the use of (`) instead of (') for javascript for string literals.
//2. Javascript majorly deals with objects similar to classes in python/C/C++, but
//classes/objects in javascript are dynamic, and static in python/C/C++.

//STRING TEMPLATE LITERALS::
let name='Devansh';
console.log(`Hi ${name}`); //ouput> Hi Devansh

//${variable_name} inside the string would equivalent to variable value.

let para =
`     The art of reading and studying
      consists in remembering the essentials
      and forgetting what is not essential.
                        – Adolf Hitler`;
console.log(para) //output>
//    The art of reading and studying
//    consists in remembering the essentials
//    and forgetting what is not essential.
//                        – Adolf Hitler

//Indentaion gets preserved in the ouput.

function getReasonCount() {return 1;}
console.log(`Give me ${(getReasonCount()==1 ? 'one good reason':'a few reasons')}
to try this !`);
// output> Give me one good reason to try this !

//REGULAR EXPRESSIONS (REGEX)::
let pattern = /xyz/;
console.log(pattern); //output> /xyz/
console.log(typeof pattern); //output> object

//Text inside '/../' is called a regular expression which we search in other
strings.
//They are of object type.

let sent1='This is xyz a test';
let sent2='Hello viewers';
console.log(pattern.test(sent1)); //output> true
console.log(sent1.replace(pattern, 'just')); //output> This is just a test
console.log(sent2.replace(pattern, 'just')); //output> Hello viewers
console.log(sent1); //output> This is xyz a test
console.log(sent1.match(pattern)); //output> [ 'xyz', index: 8, input: 'This is
xyz a test' ]
console.log(sent2.match(pattern)); //output> null

//The '.test()', is to check if the regex is present in sentence or not.
//The '.replace()' replaces regex with provided string if its present in the
sentence
//, otherwise keeps the sentence unchanged.
//However the sentence(main string) doen't gets finally changed after replace
operations.
//'.match()' gives the details of pattern in sentence.eg. index: 8, means that the
pattern
//was found at index 8 of the sentence.

let id1=sent1.match(pattern);
let id2=sent2.match(pattern);
console.log(id1.index); //output> 8

//console.log(id2.index); //output> error, because its null.
```

```
//BUILT-IN NATIVES::
//The primitive built-in natives : String(), Number(), Boolean(), Object(),
Function(),
//Symbol().
//Built-in natives not having primitive versions : Array(), RegExp().
//Additional ones: Date(), Error().

let myStr = new String('Great Game');
console.log(myStr); //output> [String: 'Great Game']
console.log(myStr.toString()); //output> Great Game
console.log(typeof myStr); //output> object

//The 'new' keyword creates construtor call for function.
//Note that 'myStr' is still an object and not string.

let myPrimitive= 'DsR!';
console.log(myPrimitive, typeof myPrimitive); //ouput> DsR! string
myPrimitive=myPrimitive.toLowerCase();
console.log(myPrimitive, typeof myPrimitive); //output> dsr! string

let num = new Number(7);
console.log(num, typeof num); //output> [Number: 7] 'object'
let pri = num.valueOf();
console.log(pri, typeof pri); //output> 7 'number'

//'.valuOf', '.toLowerCase', '.toString' are some of in-built methods.

//CONSTRUCTOR FUNCTION CALLS WITH THE 'new' KEYWORD::
function car(make, model, year){
      this.make=make;
      this.model=model;
      this.year=year;
}
let myCar = new car('bmw','745li','2010');
console.log(myCar); //output> car { make: 'bmw', model: '745li', year: '2010' }

//This is a general way of creating object(class in other programming languages)
//variables of an object.

function MyFunc(){
      console.log('HIroSHIMA');
}
let myfunc = new MyFunc();
console.log(typeof myfunc); //output> object

//Once 'myfunc' was created using 'new' keyword procedure, it no more remains a
//function and is a object, and hence:
//    myfunc(); <== will throw error.

//Note that we generally keep first alphabet of primitive function(or class)
always capital
//as a convention, so that we can identify that we can call them using 'new'
keyword.

//OBJECTS AND THE PROTOTYPE CHAIN::
let carModel={
      make: 'bmw',
      model: '745li',
      year: 2010
};
let car1 = Object.create(carModel);
console.log(car1.make); //output> bmw
console.log(Object.getPrototypeOf(car1)); //output> { make: 'bmw', model: '745li',
year: 2010 }
carModel.doors=4;
console.log(car1.doors); //output>4
```

```
//Note the use of 'Object.create()'.
//'.getPrototypeOf()' is used to return the original prototype of a particular new
//object created.
//There's always a link between newly created object and its prototype, i.e. when
we
//update a prototype its objects gets additional properties.

console.log(carModel.hasOwnProperty('doors')); //output> true
console.log(car1.hasOwnProperty('doors')); //output> false
car1.make = 'audi'
console.log(car1.make); //output> audi

//That means, the property 'doors' is owned by prototype and just carried on to
its objects.
//We can alter object properties without affecting their prototype.


//#############################################################################

//JAVASCRIPT CLASSES::
//Its main part of OOP concepts.

class Car{
      constructor(make,model,year){
            this.make=make;
            this.model=model;
            this.year=year;
      }

      print(){
            console.log(`${this.make} ${this.model} (${this.year})`);
      }
}
let myCar = new Car('bmw','745li',2010);
myCar.print(); //output> bmw 745li (2010)

//To define constructor, the term 'constructor' is used.
//As usual constructor is automatically called.
//'new' keyword is used during class object declaration.

class sportsCar extends Car{
      revEngine(){
            console.log(`Vrrroom goes the ${this.model}`);
      }
}
let mySportsCar = new sportsCar('dauge', 'viper', 2011);
mySportsCar.print(); //ouput> dauge viper (2011)
mySportsCar.revEngine(); //output> Vrrroom goes the viper

//Keyword 'extends' is used for inheriting classes.
//The inherited class retains all the primary class objects/methods.
//Notice that we even haven't created the constructor function of 'sportsCar'
class.

//ARROW FUNCTIONS::
let a = () => {console.log('Hello readers!');}
let b = (name) => {console.log(`Hi ${name}`);}
a(); //output> Hello readers!
b('DSR'); //output> Hi DSR

//This is another variation of syntax to declare/define a function.
//It uses specifically '=>' operator.

let add = (m,n) => {return a+b;}
console.log(add(7,3)); //output>10
```

```
let Canony=['AZD','DSR','HJ'];
let c=0;
Canony.map((name)=> {c++;console.log(`cool${name} ${c}`);})
//output> coolAZD 1
//        coolDSR 2
//        coolHJ 3

//The '.map()' function is used to map ech element of the object to a function i.e.
//to pass through a function.
//The mapping is dine using '=>' function here.

let d = Canony.map((name)=> {return `cool${name}`;})
console.log(d) //output> [ 'coolAZD', 'coolDSR', 'coolHJ' ]

//TRUTHY AND FALSY VALUES::
//The conditions below are falsy, and hence the else command executes.

if(false){}else{console.log('false is falsy');}
if(null){}else{console.log('null is falsy');}
if(undefined){}else{console.log('undefined is falsy');}
if(0){}else{console.log('0 is falsy');}
if(''){}else{console.log('an empty string with single-quotes is falsy');}
if(NaN){}else{console.log('NaN is falsy');}
if(""){}else{console.log('an empty string in double-quotes is falsy');}

//Eveything else is Truthy.

if(true){console.log('true is truthy')}
if({}){console.log('an empty object is truthy')}
if([]){console.log('an emty array is truthy')}
if('Canony'){console.log('a non-emty string is truthy')}
if(10){console.log('+ve integers are truthy')}
if(-10){console.log('-ve integers are truthy')}
if(-1.23){console.log('non-zero floats are truthy')}
if(Infinity){console.log('positive infinity is truthy')}
if(-Infinity){console.log('negative infinity is truthy')}

//Note how we define NaN, Infinity, -Infinity in Javascript.

//'null' TYPE::
let regex = /xyz/;
let value = 'This is a sample string.';
let result = value.match(regex);
console.log(result); //output> null
console.log(typeof result); //output> object
console.log(typeof null); //output> object

//It results 'null' if there is no such regular expression in the provided string.
//The type of 'null' element is an object.
//'null' is not '0', nor 'undefined', neither an empty string. It just means that
//object reference was expected but was not set to any value.

//DATE OBJECTS::
let today = new Date();
let tom = new Date('February 29, 2000, 07:01:23');
let yes = new Date('2000-02-29T07:01:23');
let dby = new Date('2000,02,29');
let day = new Date('2000-02-30T07:01:23');
let one = new Date('2000,02,30');
console.log(today); //output> 2019-03-23T11:44:07.261Z
console.log(tom); //output> 2000-02-29T01:31:23.000Z
console.log(yes); //output> 2000-02-29T07:01:23.000Z
console.log(dby); //output> 2000-02-28T18:30:00.000Z
console.log(day); //output> 2000-03-01T07:01:23.000Z
console.log(one); //output> 2000-02-29T18:30:00.000Z
```

```
//In the representation for 'yes', 'T' used is for time.
//Above mentioned are the possible ways to print date and time.
//The 'new Date()' return today's date and time.
//If we pass non-existing date as in 'day', it will return the date corresponding
//to the extended no. of days.

let elapsedTime = today-tom;
console.log(elapsedTime); //output> 601554298744
console.log(yes.getDay()); //output> 2
console.log(yes.getTime()); //output> 951807683000
console.log(yes.getDate()); //output> 29
console.log(yes.getMonth()); //output> 1
console.log(yes.getYear()); //output> 100

let two = new Date('2001-01-28T07:01:23');
console.log(two.getDay()); //output> 0
console.log(two.getTime()); //output> 980665283000
console.log(two.getDate()); //output> 28
console.log(two.getMonth()); //output> 0
console.log(two.getYear()); //output> 101

//'elapsedTime' gives out the exact time difference in milliseconds between the
two times.
//'.getDay' and similar functions are to return the exact day, date.. of that
particular
//Date object.
//'.getDay': 0->Sunday, 1->Monday, ..., 6->Saturday.
//similraly we have '.getHours()', '.getMinutes()', '.getSeconds()',
'.getMilliseconds()'.

//There are also in-built methods for conversion between UTC-time and local date/
time.

//STRING METHODS::
console.log('Its a name dood!'.toUpperCase()); //output> ITS A NAME DOOD!
let primes = '2,3,5,7,11,13';
let mySplit = primes.split(',');
console.log(mySplit); //output> [ '2', '3', '5', '7', '11', '13' ]

//We can directly apply string methods to strings.
//The '.split()' method allows to split the string into various parts on each
argument
//character provided(here ','), and stores it into an array.

let fact = 'BGP makes maths electives boring & complicated.';
let mySlice = fact.slice(10,25);
console.log(mySlice); //output> maths electives
let mySubString = fact.substr(10,15);
console.log(mySubString); //output> maths electives

//Note that here 10 is the index from where we have to start slicing and 25 is the
//index next to the last element of the sliced part.
//Similar works the the 'substr' to extract out a substring, but takes first input
//as the initial index and second input as the substring length.

let e=fact.endsWith('ted.');
let f=fact.startsWith('AZeDdy');
let g=fact.includes('bor');
console.log(e,f,g); //output> true false true

//These three methods returns boolean for checking if the string ends with, starts
//with or includes the given substrings as arguments to these methods.

let myRepeat = 'La! '.repeat(3);
console.log(myRepeat); //output> La! La! La!
```

```
//The repeat method is to return a string with repeated string many times.

let myTrim = '    Captain Marvel    ';
console.log(myTrim.length); //output> 22
console.log(myTrim.trim(), myTrim.trim().length); //output> Captain Marvel 14

//Basically '.trim()' methods trims out additional spaces from starting and the end
//of the string.

//ARRAY METHODS::
//There are methods like push and pop for the array in Javascript.

let names = ['AZD', 'DSR', 'HJ', 'PS', 'RK', 'SU', 'VJ'];
let userNames =
['tymefigther','DSR08','re_cursed','mr.convict','iamrakesh28','wildcat','vjac'];
let rollno = [02, 11, 13, 22, 24, 27, 29];
let fibonacci = [0,1,1,2,3,5,8,13];

let conCat = rollno.concat(fibonacci);
let fibstr = fibonacci.join(':');
console.log(conCat); //output> [ 2, 11, 13, 22, 24, 27, 29, 0, 1, 1, 2, 3, 5, 8,
13 ]
console.log(fibstr); //output> 0:1:1:2:3:5:8:13

//'.concat()' just concatenates the arrays and don't removes the repeatative
elements.
//The '.join()' method joins the elements of array keeping passed char/string in
between each
//element and returns it in form of a string.

console.log(fibonacci.shift()); //output> 0
console.log(fibonacci); //output> [ 1, 1, 2, 3, 5, 8, 13 ]

//The '.shift()' method shifts the array elements one step left and returns the
0th element.

fibonacci.unshift(0);
console.log(fibonacci); //output> [ 0, 1, 1, 2, 3, 5, 8, 13 ]

//We can even use '.unshift()' to add elements to the array by passing ','
seperated elments
// to the method which will be added in the front of array.

console.log(conCat.reverse()); //output> [ 13, 8, 5, 3, 2, 1, 1, 0, 29, 27, 24,
22, 13, 11, 2 ]
console.log(conCat.sort()); //output> [ 0, 1, 1, 11, 13, 13, 2, 2, 22, 24, 27, 29,
3, 5, 8 ]
let ab= names.sort();
console.log(ab===names); //output> true

//'.reverse()' reverses the elements of the array and '.sort()' sorts the array
elements.

console.log(userNames.indexOf('wildcat')); //output> 5
console.log(userNames.indexOf('abc')); //output> -1
console.log(conCat.lastIndexOf(2)); //output> 7

//The '.indexOf()' method returns the index of passed element, in the array.
//The '.lastIndexOf()' method returns the index of last occurence of the passed
element, in
//the array.
//These methods return '-1'if they sre not present in the array.

let canony = names.filter((x)=> {if(x<'Nobody!')return x;})
console.log(canony); //output> [ 'AZD', 'DSR', 'HJ' ]
```

```
let bb=0;
names.forEach((x) => {bb+=1;})
console.log(bb); //output> 7
console.log(rollno.every((roll) => roll<25)); //output> false
console.log(rollno.some((roll) => roll<25)); //output> true

//To filter the elements out of array under certain criterias we use '.filter()'
method.
//'forEach()' method is to carry out a procedure for each single element in array.
//'every()' returns a boolean value according to the condition passed to it,
checking that
//consition for all the elements of array.
//Similarly '.some()' mehod checks for any single element that satisfies given
condition &
// returns boolean according to it.


//#############################################################################

//IMPORTANT LINKS AND WEBSITES::
//0. Google Fonts

//KEY POINTS::
//1. In order to print (') single quote as a part of string we need to add a '\'
before the
//single quote so as to make Javascript recognize (') not as a end of string
command.
//2. For adding a Javascript code in HTML file, we need to write the code within
the
//'<script>' and '</script>' tags.

//ERROR HANDLING WITH TRY CATCH::
//Javascript tries to correct the code as much as possible, and produce reasonable
outputs.

let a = 7.23 * (undefined)/'ai';
console.log(a) //ouput> NaN

//That was certainly meaningless, but still instead of raising error it gave some
output.

function beforeTryCatch(){
      let obj=undefined;
      console.log(obj.b);
      console.log('If previous linethrows an exception you\'ll can never see
this');
}
//beforeTryCatch(); //<-- Commented as above function throws error.
function afterTryCatch(){
      try{
            let obj=undefined;
            console.log(obj.b);
            console.log('If previous line throws an exception you\'ll can never
see this');
      }
      catch(error){
            console.log('Caught a bug! : ' + error.message);
      }
      finally{
            console.log('This will happen regardless of error occurs or not.')
      }
      console.log('When I am sad, I hack!!!');
}
afterTryCatch();
//output> Caught a bug! : Cannot read property 'b' of undefined
//        This will happen regardless of error occurs or not.
```

```
//        When I am sad, I hack!!!

//This represents general syntax of using 'try'/'catch'.
//Note that once an error is found, no other following commands in 'try' is
executed.
//However content of 'finally' is always executed.
//We can also print error message in Javascript.

function performCalc(obj){
      if(!obj.hasOwnProperty('b')){
            throw new Error('object missing property');
      }
      return 8;
}
function perform(){
      let obj={};
      let value=0;
      try{
            value=performCalc(obj);
      }
      catch(error){
            console.log(error.message);
      }
      if(value!=0)
      {
            value+=1;
            console.log(value);
      }
}
perform(); //output> object missing property

//Note that, since the throw action takes place the 'return 8' doesn't takes place
and
//the 'value' remains '0'.

//UNDERSTANDING THE DOCUMENT OBJECT MODEL(DOM)::
//Here we will be dealing with most of the web developement techniques. How the
documents
//are created, how do we specify which functions are to be called on provided user
input.

//The Document Object Model (DOM) is a programming API for HTML and XML documents.
It
//defines the logical structure of documents and the way a document is accessed
and manipulated.

//The Document Object Model is a cross-platform and language-independent
application programming
//interface that treats an HTML, XHTML, or XML document as a tree structure
wherein each node is
//an object representing a part of the document. The DOM represents a document
with a logical tree.

//Note: 'node' here should not be confused by 'nodejs', they are different.

//Each element node can have attribute nodes associated with it.We can find a node
or a collection
//of nodes that satisfy a particular criteria, and then APIs provide us control to
modify those
//collection of nodes i.e. text, attributes of the node, changing the class that
was associated
//with the given node, we can add nodes, delete nodes.

//APIs also allows to associate events and arrays by the web browser.

//Take a look at: dom-intro.html                                              **
```

```
//This explains us how to use javascript function and codes in html and link with
it.

//Take a look at: dom-intro2.html and dom-intro2.js                              **
//This explains us how to link with a document code of javascript with html and
check for
//the console messages and outputs.

//WORKING WITH DOM NODES::
//Take a look at dom-nodes.html and dom-nodes.js.                                **
//Just check out what this html file results to.


//** -> there's a file linked for it.


//###################### LINKED FILES ##########################################

//------------- this_keyword.html -------------------------------------------

<html lang='en'>
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <meta http-equiv="X-UA-Compatible" content="ie=edge">
      <title>Document</title>
</head>
<body>
      <button onclick="clickHandler(this)">Click me</button>

      <script>
            function clickHandler(arg){
            console.log(arg); //ouput> <button onclick="clickHandler(this)">Click
me</button>
            console.log(arg.innerText); //output> Click me
            console.log(this); //gives me all the child objects of window.
            }
      </script>
</body>
</html>

//-----------------------------------------------------------------------------


//------------ dom-intro.html ------------------------------------------------

<html lang='en'>
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <meta http-equiv="X-UA-Compatible" content="ie=edge">
      <title>Document</title>
</head>
<body>
      <h1>Header</h1>
      <ul>
            <li>One</li>
            <li>Two</li>
            <li>Three</li>
      </ul>
      <button onclick="alert('Hi DSR..')">Click Me</button>
      <button onclick="console.log('Its a console print.');">Don't Click Me</
button>
      <br>
      <button onclick="clickHandler('from the button click event')">Click Here</
button>
```

```
      <script>
            function clickHandler(message){
                  console.log('Hi...'+message);
            }
      </script>
</body>
</html>

//-------------------------------------------------------------------------------


//------------ dom-intro2.html ------------------------------------------------

<html lang='en'>
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <meta http-equiv="X-UA-Compatible" content="ie=edge">
      <title>Document</title>
</head>
<body>
      <h1>Header</h1>
      <ul>
            <li>One</li>
            <li>Two</li>
            <li>Three</li>
      </ul>

      <button id="myButton">Click Here</button>
      <script type="text/javascript" src="dom-intro2.js"></script>

</body>
</html>

//-------------------------------------------------------------------------------


//------------ dom-nodes.html -------------------------------------------------

<html>
      <head>
            <link rel="stylesheet" href="<link href="https://fonts.googleapis.com/
css?family=Lato|Poppins" rel="stylesheet">
            <link rel="stylesheet" href="dom-nodes.css">
      </head>
      <body>
            <h1>Javascript Notes by DSR</h1>
            <h2>Final Project</h2>

            <div id="resultContainer">
                  <button id="myButton">Click Me</button>
                  <div id="colorDiv"></div>
                  <div id="resultDiv"></div>
            </div>

            <script type="text/javascript" src="dom-nodes.js"></script>
      </body>
</html>

//-------------------------------------------------------------------------------


//--------------- dom-intro.js ----------------------------------------------

(function(){
      function clickHandler (message){
```

```
            console.log('Hi...'+message);
      }

      //Get reference to myButton
      let myButton = document.getElementById('myButton');
      myButton.addEventListener('click',clickHandler('Hi from iife'));
})();

//-------------------------------------------------------------------------


//-------------- dom-intro2.js ------------------------------------------------

(function(){
      function clickHandler (message){
            console.log('Hi...'+message);
      }

      //Get reference to myButton
      let myButton = document.getElementById('myButton');
      myButton.addEventListener('click',function(){clickHandler('Hi from iife')});
})();

//-------------------------------------------------------------------------


//-------------- dom-nodes.js ------------------------------------------------

(function(){
      function incrementCounter(){
            counter+=1;
      }
      function updateUI(){
            const colors=[
                  {name: 'AZD', value:'#e74c3c'},
                  {name: 'DSR', value:'#2980b9'},
                  {name: 'HJ', value:'#2ecc71'},
                  {name: 'PS', value:'#2c3e50'},
                  {name: 'RK', value:'#f1c40f'},
                  {name: 'VJ', value:'#d35400'},
            ];
            let result=document.getElementById('resultDiv');

            //Relative to the font size of the element.
            //(2em means 2 times the size of current font)
            result.innerText = counter;
            if(counter>0)
            {
                  result.style.fontSize = counter +'em';
            }

            //Divide first into second, return the remainder.
            //Access the element of the colors array to grab
            //the color object:
            let remainder = counter % color.length
            result.style.color = colors[remainder].value;

            //Clear out all existing child color divs
            let colorDiv = document.getElementById('colorDiv');
            colorDiv.innerHTML = '';

            //Re-add the child color divs
            for(i=0;i<colors.length;i++)
            {
                  //Creating node dynamically with the intent of
                  //adding it to the colorDiv.
```

```
                  let node=document.createElememt('div');
                  let textnode=document.createTextNode(colors[i].name);
                  node.appendChild(textnode);
                  node.style.backgroundColor=colors[i].value;

                  //Alternatively we could have amde this into a CSS
                  //style and added that to the node.ClassList ...
                  node.style.width='150px';
                  node.style.height='50px';
                  node.style.cssFloat='left';
                  node.style.paddingLeft='10px';
                  node.style.paddingTop='10px';

                  if(i==remainder)
                  {
                        node.style.height='45px';

                        //Example of adding a class to the node 's
                        //classList.
                        node.classList.add('selected');
                  }
                  colorDiv.appendChild(node);

            }
      }

      function handleClick(){
            incrementCounter();
            updateUI();
      }

      let counter=0;
      let myButton = document.getElementById('myButton');
      myButton.addEventListener('click',function(){
            incrementCounter();
            updateUI();
      });

      updateUI();
})();

//-------------------------------------------------------------------------

//###################### THE ##################################################
//######################### END ###############################################
```