Contents

[Console 1](#_Toc110621745)

[Variables 1](#_Toc110621746)

[Primitive data Types 1](#_Toc110621747)

[Reference Type 2](#_Toc110621748)

[Type Conversion 2](#_Toc110621749)

[Operations 2](#_Toc110621750)

[Number Operations 2](#_Toc110621751)

[String Operations 2](#_Toc110621752)

[Template literals / string 3](#_Toc110621753)

[Arrays 3](#_Toc110621754)

[Object literals 5](#_Toc110621755)

[Date Literal 5](#_Toc110621756)

[Conditional statement 6](#_Toc110621757)

[Logical operators 6](#_Toc110621758)

[Switch 7](#_Toc110621759)

[Functions 7](#_Toc110621760)

[IIFEs (Immediately Invokable Function Expression) 7](#_Toc110621761)

[Methods 8](#_Toc110621762)

[Loops 8](#_Toc110621763)

[For loop 8](#_Toc110621764)

[While loop 9](#_Toc110621765)

[Do While Loop 9](#_Toc110621766)

[ForEach Loop 9](#_Toc110621767)

[For in 9](#_Toc110621768)

# Console

Console can be used to display info in multiple ways.

Console.log, Console.table, Console.warn, Console.error, Console.clear, console,.time(‘id’), console.timeend(‘id’)

# Variables

Dynamically typed, can store and change any type.

Var : older versions on javascript. Can be assigned new value after initializing.

Let : introduced in ES6 (2015). Can be assigned new value after initializing.

Const : introduced in ES6 (2015). Cant reassign to new value after initializing.

Uninitialized variables have value ‘undefined’

Variables can have only letters, number, \_ and $ in name and they cannot start with number.

For const type we can not reassign variable to something else, but we can change the value if it is object.

const person = {name= ‘Ashpak’, age=34} person.age=35 works fine,

## Primitive data Types

// Premitive types

//string

const name =  'Ashpak';

//number

const age = 34;

//boolean

const isMarried = true;

//null

const superRich = null;

//undefined. This cant be if type const.

let wealth;

//symbol

const sym = Symbol();

## Reference Type

// REFERENCE TYPE - objects

//array

const skills = ['python','terraform','javascript','aws']

//object

const address = {house : 'M23', city: 'pune'}

//date

const today = new Date()

# Type Conversion

Type conversion can be done like String(value), Number(value) to string or numbers.

Same is supported using value.toString() or parseInt(value), ParseFloat(value)

Javascript does type coercion as well by automatically converting types depending on operation

const val1 = String(5);

const val2 = 6;

const sum = val1 + val2; // output 56

# Operations

## Number Operations

Apart from basic mathematical operations +,-,/,\*,% we have Math operations for different things like

Math.round(2.4) which outputs 2 or Math.ceil(2.4) which oututs 3. Math.sqrt(64), Math,pow(8,2), Math.min(4,5,30,20) etc……

## String Operations

Concatenation through ‘+’ ex. Str1+str2 concatenates two strings.

Str.length to get length

Str.toUppercase() and lowercase for changing string case.

String is like array so we can access str[2] to get 3rd char in string

Str.indexOf(‘chr’) return the zero based index of given chat in string from left

Str.lastIndexOf(‘chr’) return zero based index of given string from right

Str.charAt(‘2’) gives character at index 2

Str.subtring(0,3) returns first 3 chars in string

str.slice(0,3) same as substring return 3 chars from index zero.

Str.slice(-3) returns last 3 characters form string

Str.split(‘-’) returns zero index array after splitting string with char ‘-’

Str.replace(‘abc’,’xyz’) replace ‘abc’ with ‘xyz’ in string

Str.includes(‘abc’) returns true/false depending on if ‘abc’ is part of str

## Template literals / string

Instead of concatenating strings using + we can use template literal syntax ${var} to make it easy when it comes to very big strings. Specially in javascript when we create dynamic html elements this comes handy. We can call variables, functions, conditional ternary operators etc. in syntax. It starts with ` which is sing next to 1 ket and comes with ~ key.

let myname ='ashpak';

let myage = 36;

let html = `<div>

                <h1> ${myname} </h1>

                <p> ${sayhello()}</p>

                <p> ${age>30 ? 'over 30' : 'below 30'} </p>

            </div>`

# Arrays

Arrays can be created with [] or with new constructor new Array()

const numbers = [43,56,33,23,44,36,5];

const numbers2 = new Array(22,45,33,76,54);

Array can have any type of items

const mixed = [22, 'Hello', true, undefined, null, {a:1, b:1}, new Date()];

array support very large number of methods. Note const works on num array even if we mutate it, because we are mutating items inside object store in ‘num’ and not replacing object.

const num = [43,56,33,23,44,36,5];

// Get array length -> returns 7 total number of items

val = num.length;

// Check if num is array

val = Array.isArray(num);

// Get single value using index

val = num[3]; //--> return 23

val = num[0]; // -> return 43

// Insert into array

num[2] = 100;

// Find index of particular value

val = num.indexOf(36); // -> returns 5

const num = [43,56,33,23,44,36,5];

//Add item on to end of array

num.push(250); //--> makes array [43,56,33,23,44,36,5,250]

//Add on to front if array

num.unshift(120); //--> makes array [120,43,56,33,23,44,36,5,250]

// Remove item from end of the array. Opposite to push.

num.pop();  //--> makes array [120,43,56,33,23,44,36,5]

// Remove item from front of array

num.shift();  //--> makes array [43,56,33,23,44,36,5]

// Splice/remove values mentioned in range (start index, number of items)

num.splice(1,3); //--> makes array [43,44,36,5] by removing 3 items starting index 1

//Reverse items in array. Last item will be f.irst and first will be last

 num.reverse(); //--> makes array [5,36,44,43]

// sort the array. If it is strings then sorted alphabetically

num.sort();

// if array is number then sort work differently

let newarr = [100,55,44,43,36,23,5]

newarr.sort()  //-> returns [100,23,36,43,44,5,55] which is not correct

// for number array return we need to give direction of sort using

// 'compare function

// returns --> [5,23,36,43,44,55,100]

val = num.sort(function(x, y){

   return x - y;

});

// Reverse sort returns --> [100,55,44,43,36,23,5]

val = num.sort(function(x, y){

   return y - x;

});

# Object literals

Objects in javascript is json like structure which can story any data types in its values section in key value formation (including functions)

JSON : Java Script Object Notification…so JSON itself comes from Javascript.

const person = {

    firstName: 'Ashpak',

    lastName: 'Mulani',

    age: 34,

    email: 'ashpaklmulani@gmail.com',

    hobbies: ['music', 'sports'],

    address: {

      city: 'Pune',

      state: 'MH'

    },

    getBirthYear: function(){

      return 2022 - this.age;

    }

  }

// Get specific value

val = person.firstName;

val = person.hobbies[1];

val = person.address.state;

val = person.address['city'];

val = person.getBirthYear();

# Date Literal

const todey = new Date();

let birthday = new Date('9-10-1981 11:25:00');

// there can be multiple ways initial value ford ate object can be passed

val = todey.getMonth();

val = todey.getDate();

val = todey.getDay();

val = todey.getFullYear();

val = todey.getHours(); // same for mimuts, secs, miliseconds

val = today.getTime(); // return time part of date

birthday.setMonth(2); // same for date, year, hours, minuts etc...

# Conditional statement

if(true){

    do something

}

else{

    do something else

}

// condition check

if(a==b)

// checks if a and b are equal, it doesn’t check type though. if a=5 and b='5' we will still get true

if(a===b)

//checks if value of a is equal to b and it will also check if type are equal. if one is string

// another is integer then it will return false

if(a!=b)

// checks not equality in value but not type. !== checks value as well as type.

// we can also check >, < greater than or less than values

# Logical operators

if(name==='abc' && age < 10){

    console.log("abc is a kid")

}else if(name='abc' && age<20)

{

    console.log("abc is teenager")

}else{

    console.log("abc is adult")

}

// AND is used as &&

//OR operator is used as ||  (two lines)

Instead of writing bigger if else for smaller checks we can use **ternary operations**

let val =5;

val==5 ? 'if executed' : 'else executed'

// condition ? true part : false part

# Switch

let color = 'red';

switch(color){

    case 'red':

        console.log('color is red');

        break;

    case 'blue':

        console.log('color is blue');

        break;

    default:

        console.log('color is default');

        break;

}

# Functions

// default value can be mentioned in function declaration.

function fullname(firstName='abc', lastName='xyz'){

    return `${firstName} ${lastName}`;

}

console.log(fullname('ashpak','mulani'))

//-> outputs 'ashpak mulani'

console.log(fullname('ashpak'))

//-> outputs 'ashpak xyz'

Function can also be defined a variable

const square = function(x){

    return x\*x;

}

square(5);

## IIFEs (Immediately Invokable Function Expression)

// Immidiately invokable function expression - IIFEs

// This pattern helps invoking function directley where we define it.

//check ('ashpak') at the end whihc is involing the fumction by providing params

(function hello(name){

    console.log(`Hello ${name} !!!`)

})('ashpak');

// result 'Hello ashpak'

## Methods

When function is added in object its called object/property method

const todo = {

    add : function(){

        console.log('inside add function')

    },

    edit : function(id){

        console.log(`inside edit function ${id}`)

    }

}

todo.delete = function(){

    console.log('inside delete')

}

todo.add();

todo.edit(10);

todo.delete();

# Loops

## For loop

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

    console.log(`current value is ${i}`)

}

For loop also supports ‘continue’ and ‘break’ keywords

## While loop

let i=0;

 while(i<10){

    console.log(`current value ${i}`);

    i++;

 }

## Do While Loop

let i=0;

 do{

    console.log(`current value ${i}`);

    i++;

 }

 while(i<10);

## ForEach Loop

Helps to go through each element on the list.

 let cars= ['Jaguar', 'Farrari', 'Toyota', 'Suzuki']

 cars.forEach(function(car){

    console.log(car);

 })

## For in

For each only works on list, but not on objects. If we have to go through items in object then for in can be used

let cars = {car1 : "suzuki", car2 : "honda"}

for(car in cars){

    console.log(car) //-> outputs car1, car2

    console.log(cars[car]) //-> outputs suzuki, honda

}

// notice use of cars[car] and we have not used cars.car to access items in

// cars object because it will fail if we do so, since JS will try to access

// .car key from cars object which doesn’t exists. We want to access key from cars object with value inside car variable so we have used cars[car]