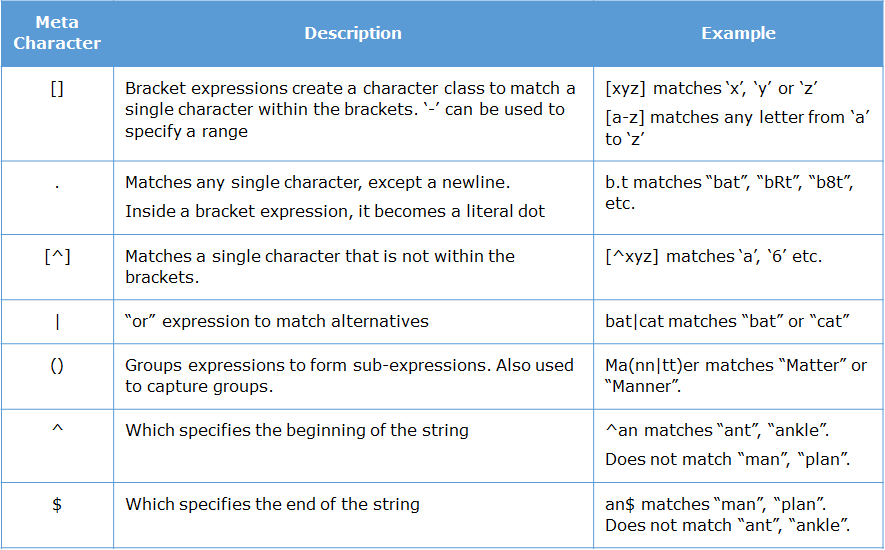
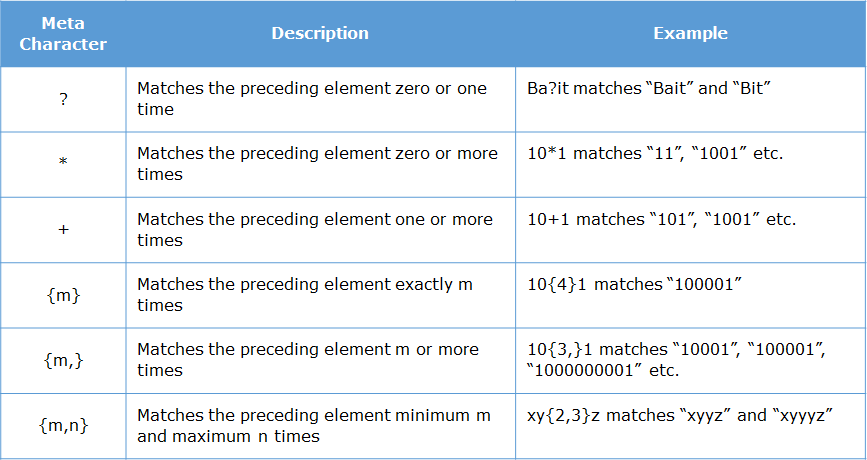
# RegEX





# URI and URL difference

[example.com/Project/posts](http://www.example.com/Project/posts) – URI (uniform resource identifier) can identify but not locate

http://www. [example.com/Project/posts](http://www.example.com/Project/posts) – URL ( UR Locator) can be used to locate and identify a resource

# WEB API

In fact console.log() is actually not part of the language. It is provided by the browser as part of its API. It is through that API we are able to interact with the browser and print something on the browser's console.

There are many Web API. Some of which we will cover in the course are:

* XMLHttpRequest
* DOM
* Notification
* Storage

**Note:** Since these API's are provided by the browser, they will not work in Node

XMLHttpRequest and DOM are Asynchronus Javascript And XML (AJAX) in nature

setTimeOut(fn(),timeInMilliSec) is DOM api following AJAX technique

Same, XMLHttpRequest and its methods like http.open(‘GET’,url) follow Asynchronus i.e AJAX techniques

To make sure the function executes once the response is received we use callbacks, if there are too many callbacks nested up like callback has another callback (callback hell) to avoid this problem we use promise.

function getTrip(){

return new Promise(function(resolve){

setTimeout(function() {

resolve("Lets go to Trip");

}, 2000);

});

};

This code promises a states that if everything goes well then it will resolve the “Lets go..” data

function bookFlight() {

return new Promise(function (resolve) {

setTimeout(resolve(5600), 2000);

})

}

function bookHotel(flightPrice) {

return new Promise(function (resolve) {

setTimeout(resolve(7000 + flightPrice), 1000);

})

}

function getTotal(){

bookFlight()

.then(function (flightData) { return bookHotel(flightData) })

.then(function (cumulativeData) { console.log(" Total is " + cumulativeData) })

}

getTotal()

we can use await inside async function to make code wait till promise gets resolved, above code can be written in

async function getTotal(){

var flightData=await bookFlight();

var cumulativeData=await bookHotel(flightData);

console.log(" Total is " + cumulativeData)

}

Suppose there is a JSON file in ../json url then we can access through XMLHttpRequest

Function getData(url){

var xhr = new XMLHttpRequest();

xhr.open(‘GET’,url);

xhr.onload = function () { // Invokes after getting response

var json = JSON.parse(xhr.responseText)

document.getElementById(“response”).InnerText = JSON.stringify(json);

}

}

# UNIT TESTING:

. JS doesn’t have any inbuild compiler to detect errors, we can only find them directly in the browser. So unit testing is very important during coding in the Js. Keeping console.log() statements in the code in every function is lengthy process so automated testing JASMINE is used. Test suits created are executed with the help of karma.

Eg for test suite:

describe('TotalTravelFare calculation Suite:',function(){

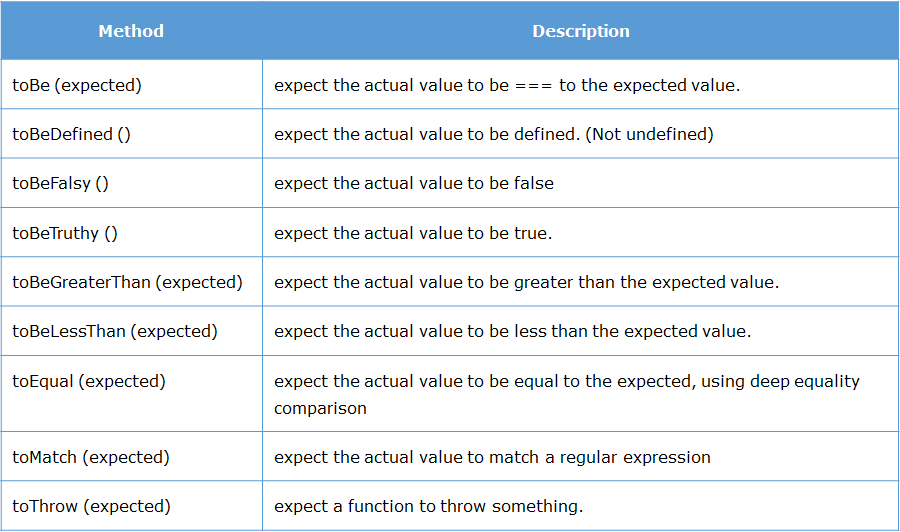
it('Test Case 1: Inputs are correct',function(){

expect(totalTravelFare(1000,20)).toEqual(1200);

});

})

describe,it,expect is global jasmine function, toEqual are Jasmine matchers. Others are



Can have not like – it(“sd”,function(){expect(12).not.toBeLessThan(10)})

To maintain resources and not to invoke or access common functions or data we use beforeEach and afterEach

beforeEach(function(done) {

setTimeout(function() {

string = 'Stormhold';

}, 1500);

});

describe('Value of n', function() {

var n = 0;

afterEach(function(done) {

setTimeout(function() {

n++;

done();

}, 1500);

});

it('is 0', function() {

expect(n).toEqual(0);

});

it('is 1', function() {

expect(n).toEqual(1);

});

});

After the test cases are written in .js file to run those we need KARMA (Test runner tool)

npm install -g karma jasmine-core

Karma is:

* A tool that spawns a web server which executes the source code against the test code for each browsers connected.
* When executed, it automatically captures the browser specified by the developer during Karma configuration.
* It then displays the results on the command line.
* It watches all the files specified within configuration file and if there are any changes, it will trigger the corresponding spec again on the browser.

Conclusion: We create test cases using jasmine global functions and run them on KARMA tool by command ( karma init ) to get karma.config file with all dependencies followed by (karma start).

# Document Object Model

DOM API – Using the DOM API we can interact with the HTML elements and the browser window using JavaScript. DOM stands for Document Object Model. It models the HTML document into an object. That means, everything we see in the HTML page is represented as objects in JavaScript.

**DOM**is not a programming language, it is an interface through which JavaScript access the elements of HTML pages.

One of the most common method associated with the document object is:

**getElementById(id):** It is used to access element by its id.

**getAttr("attributeName"):**It is used to access the tag attribute of the DOM object

**innerHTML:** innerHTML is used to set or get the HTML content of the element.

**innerText:** innerText is used to set or get the Text content to element.

**value:** value is used to get or set value of the element.

**alert("message")** - this will display an alert box with the given message

**confirm("message")** - this will display an confirm box with the given message

**write("message")**- this will overwrite the existing HTML content of the page with the message given

console.log(document.getElementById("username").getAttribute("placeholder"));

console.log(document.getElementById("password").value)

//All elements are also stored as arrays. We can also access based on the index position.

console.log(document.forms[0].elements[1].getAttribute("maxlength"))

directly like document.formName.fieldName.value

<form><input type=”text” name = “iname” id=”ID”></form>

Document.form.iname.value

Also we can get by: const field = document.querySelector(‘.className or id’) (it runs a query and gives the first matching css class or id with string passed) to return all use querySelectorAll

Field.addeventListener(‘click’, () => { }} (will invoke if field is clicked)

Field.classList.add(‘new class’)v(adds new class to the field)

We can get details of the element on which the event took place using **event.target**

<input type="radio" name="gender" value="male" onclick="display(event)">Male

<script>

function display(e){

console.log(e.target.name);

console.log(e.target.value); }

</script>

Suppose there is Div1 which has onclick event and has child Div2 inside it which also have onclick event then if we click on div2 then both div2 click event followed by div1 click event gets invoked this is called **Event Bubbling**. If you wat to prevent this we can use **event.preventDefault()** .

Tip: Just to make page reactive use container/fluid class at begging and then create div with class row and inside it use classes col-md-4 etc then create forms etc in it

# Session Storage

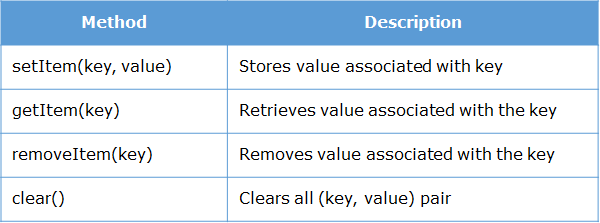
Session storage is a Web Storage API introduced in HTML5

We can store the data using web storage in two ways:

1. For particular session i.e. **session**storage: For session storage, *sessionStorage*object is used
2. Across sessions i.e. **local**storage: For local storage, *localStorage*object is used

Both these objects are created by JS runtime engine of browser.

Web storage API provides following four methods for managing data:



# Notification

We can create desktop notifications in JavaScript using the Notification API

To gain access from user: Notification.requestPermission()

To display notification:

1. function notify(){
2. Notification.requestPermission();
3. if(Notification.permission === "default"){
4. alert("Please grant permission");
5. }
6. else {
7. var notify = new Notification("New Mail",{body:"You have 1 unread email"});
8. }
9. }
10. Possible values for permission are denied,granted,default, states current choice

6. body – Defines notification body

# Cross-site scripting (XSS)

Cross-site scripting (XSS) is a security bug that can affect websites through injecting code

Eg: <img src = 1 onerror=”s=document.createElement(‘script’);s.src=’../evil.js;document.body.appendChild(s)’”>

<img src = x onerror = “alert(document.cookie)”> //gives session storage id

This can be prevented by running regEx on every input fields we provide before processing that data(validating input), using a template with context-aware by auto escaping or manually doing the escaping on every input data (<script>alert(‘hello’)</script> is converted into &lt;script&gt;alert(‘hello’) &lt;/script&gt; this is called escaping which is encoding actually)

Objects:

Objects are mutable, you cannot create a replica of it

Eg: var emp = {name : “Pavan”}

Var x = emp; // this is calling emp by x, not creating a new x

x.name = “XYZ” // this will change both the emp.name and x.name value

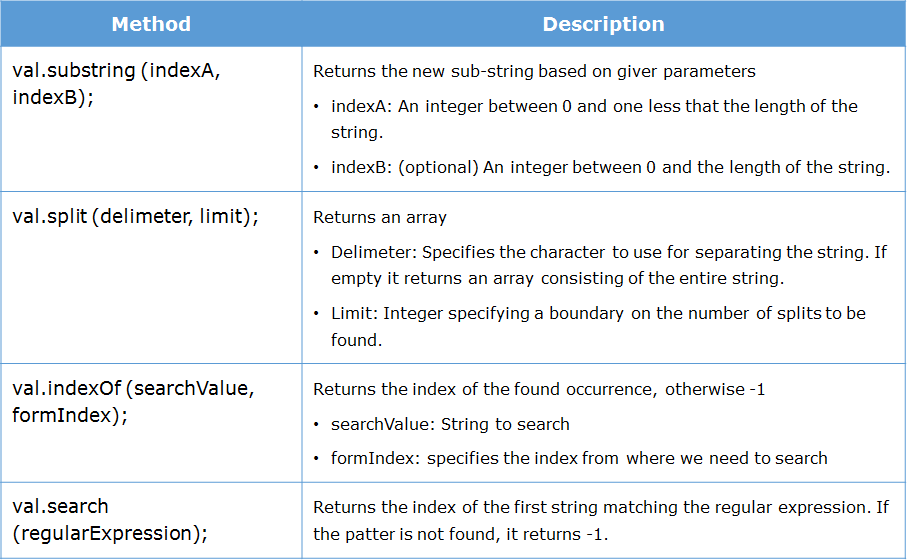
IF we want to change the key name just assign old key value to new key value and delete the old one

Eg: emp.fullname = emp.name

Delete emp.name

To know whether object emp has ‘example’ property then emp.hasOwnProperty(‘example’) returns true or false.

# String operations



# WEB Worker

When we need to do I/O operations or run scripts in the background simultaneously then we use web workers.

Eg: this is worker.js file which sends message through **postMessage** method of var i continuously with 500millisec gap

var i = 0;

function countNumbers(){

if(i < 100000){

i = i + 1;

postMessage(i);

}

setTimeout("countNumbers()", 500); // Wait for sometime before running this script again

}

countNumbers();

HTML page to use this

<html>

<script type="text/javascript">

if(window.Worker){

var worker;

worker = new Worker("worker.js");

worker.onmessage = function(counter){

document.getElementById("countervalue").innerHTML = counter.data;

};

} else{

alert("Web worker is not supported in your browser");

}

</script>

</head>

<body>

<div id="countervalue">

<!-- Messages received from web worker will be displayed here -->

</div>

</body>

</html>

# Closures

If a function takes another function as parameter then it is called Higher order functions. If functions have another functions inside it then it is called closures.

A closure is an inner function that has access to the variables present in the **lexical scope**(parent scope) of the outer/containing function. Therefore it can outlive the lifetime of the outer function and hence can continue to access the variables of the outer function.

function outer() {

var count = 0;

function inner() {

count=count + 1;

return count;

}

return inner;

}

private\_function=outer();

console.log(private\_function()); // 1

console.log(private\_function()); // 2

console.log(private\_function()); // 3

Here we can access inner function without invoking outer thus it can outlive outer fn.

# **Immediately Invoked Function Expression (IIFE)**

If function(mostly anonumous) is invoked immediately after it is created then it is call IIFE

Eg: (function (num1, num2){return num1 + num2;})(100, 200) // 300

So if we invoke () immediately after the anonumous fn then it will run and gives result.

Another example:

var outer = (function () {

var counter = 0;

return function () { return counter += 1; }

})(); // var outer = (...an anonymous function...)();

console.log(outer()); //1

console.log(outer()); //2

console.log(outer()); //3

here also anonymous fn gets invoked by () in the end and it returned a function which gets called in the console.log by outer().

Note: IIFE means ()() it should not be ();()

# Object with Function

An object can be created by using **constructor**function. The syntax is:

function Name(param1,param2,param3) {

this.param1 = param1;

this.param2 = param2;

this.param3 = param3;

this.swipeIn=function(){console.log("Swipe in by "+this.param1);}

};

var obj = new Name(p1,p2,p3);

# ParseInt and ParseFloat

Used to convert string to intiger

parseInt("5"); // 5 parseFloat("1.23"); // 1.23

parseInt("5.5"); // 5 parseFloat(".123"); // 0.123

parseInt("5p0"); // 5 parseFloat("1.23abc"); // 1.23

parseInt("p50"); // NaN (Not a Number) parseFloat("a123") // NaN

tip: typeof(NaN) = number //NaN = 0

typeof(3.14) = number //there is no type called float

# NODE and Express

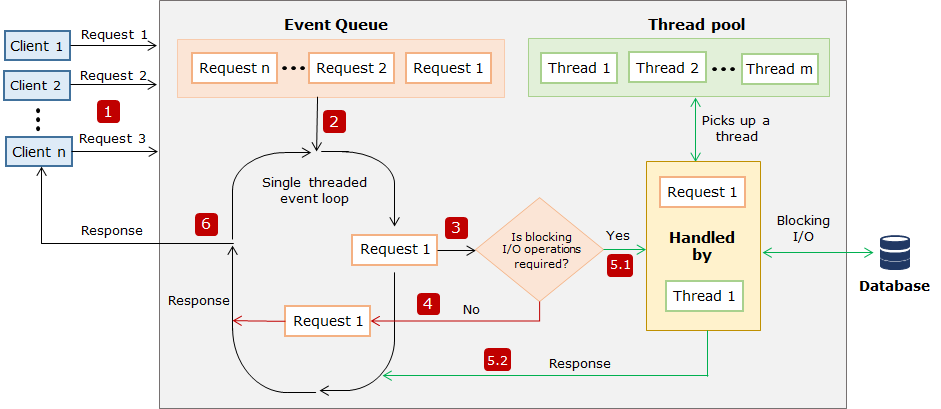
**Node**.**js**® is a open-source JavaScript runtime built on Chrome's V8 JavaScript engine.

Express is the most popular framework for creating web applications in Node.js. It is lightweight and provides easy connectivity with database like MySQL and MongoDB. Using Express we can handle requests, create views and manage routes.

**Reduced resources**, **Library support (NPM), Asynchronous and Event-driven**, **Wide client side (Angular, react) and database connectivity (Mongo, MYSQL)**

* V8 is Google's open source high-performance JavaScript engine, written in C++ and used in Google Chrome, designed to improve the performance of the JavaScript execution in browsers.
* At the time of execution, it converts the JavaScript code into machine code using **Just-In-Time (JIT) Compiler**, to achieve high speed. Therefore byte code or intermediate code will not be created

Node.js environment is created based on **Single Threaded with Event Loop Model** which is built using JavaScript's callback mechanism.



# Modularization

Modularization is a software design technique in which the functionality of a program is separated into independent modules, such that each module contains a desired functionality.

Syntax to export: exports.<module\_name> = <module\_implementation>

export.add = (x,y){return x+y} //just like in js export class abc or export function xyz() or export default fn e()

const myCalculator = require('./calculator'); //just like in js import e,{abc,xyz} from ‘filepath’

myCalculator.add(1,2) //3

if ther is one export then we can use module.exports = add so if we import by require in another file it will directly refer to add i.e myCalculator(1,2) // 3

# Modules

These modules are used first by assigning them to local variable by required() method

Eg: const express = require(‘express’);app = express(); const fs = require(‘fs’)

Nodemon – it is a package which will automatically refresh the server if any changes occur (just like how ng serve and autosave works in angular), we install it just like express and other modules(npm i nodemon) and run the files nodemon app.js in cli instead of node app.js

fs – this module is used to read,write,append a log file (error , request logger)

fs.writeFile(file\_path, data, callback) , appendFile() (write will overwite the content and creates new if not exist) (callback function is invoked eventhough the write is success or not)

eg: const fs = require('fs');

let fileWrite = () => {

let str = "Hey, there.. ";

fs.writeFile('./log.txt', str + " At: " + new Date().getFullYear(), (err) => {

if (!err)

console.log('Data Written!');

else

throw err;

})

}

fileWrite();

eg: const fs = require('fs');

fs.readFile('demo.txt', 'utf8', (err, content) => { // charset is optional

if(!err)

console.log(content);

else

throw err; });

## http: (webserver module)

permits us to send data over the HTTP protocol, is used to build a HTTP server

let server = http.createServer((req, res) => {

res.write('Hello World! I have created my first server!');

res.end();

})

server.listen(3000);

console.log("Server started... Running on localhost:3000")

this is creating server in node.js, later this entire thing is replaced by express with simple app.listen(3000) (app.listen takes a callback just like fs.readfile above which is optional)

if we send res.send(“<i>italic</i>”) then it will take it as plain text not as html so to solve this we use response.writeHead(200,{"Content-Type":"text/html"}); now it will print as *italic*

Tip: http and fs are built in modules(provided internally by node.js) directly used by required() but a NPM(provided externally by node.js) module like express has to install and use it

npm install <package\_name>[@<version>] //version optional takes latest

This will create a folder **node\_modules** in the current directory and put all the packages related files inside it.

# Express

Express is a **layer** built on Node.js which helps us to **manage our web server**and **routes**.