Learn JavaScript Objects:

<https://www.w3schools.com/js/js_object_definition.asp>

Official Documentation: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model>

**Java Script object with a method**

var **person** = {  
    firstName: "John",  
    lastName : "Doe",  
    id       : 5566,  
    fullName : function() {  
        return **this**.firstName + " " + **this**.lastName;  
};

var myFather = new Person("John", "Doe", 50, "blue");  
var myMother = new Person("Sally", "Rally", 48, "green");

**Object Prototypes:**

**Prototype Inheritance**

All JavaScript objects inherit properties and methods from a prototype.

* Date objects inherit from Date.prototype.
* Array objects inherit from Array.prototype.
* Person objects inherit from Person.prototype.

The Object.prototype is on the top of the prototype inheritance chain:

Date objects, Array objects, and Person objects inherit from Object.prototype.

The JavaScript prototype property allows you to add new properties to object constructors:

function Person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}  
Person.prototype.nationality = "English";

The JavaScript prototype property also allows you to add new methods objects constructors:

function Person(first, last, age, eyecolor) {  
    this.firstName = first;  
    this.lastName = last;  
    this.age = age;  
    this.eyeColor = eyecolor;  
}  
Person.prototype.name = function() {  
    return this.firstName + " " + this.lastName;  
};

**Node Good Video Tutorial:**

<https://www.youtube.com/watch?v=mbzCe1of_Hs>

npm init

**JQuery CallBack Mechanism:**

## jQuery Callback Functions

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical syntax: **$(*selector*).hide(*speed,callback*);**

<!DOCTYPE html>

<html>

<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>

<script>

$(document).ready(function(){

$("button").click(function(){

$("p").hide(1000);

alert("The paragraph is now hidden");

});

});

</script>

</head>

<body>

<button>Hide</button>

<p>This is a paragraph with little content.</p>

</body>

</html>

$(document).ready(function(){

$("button").click(function(){

$("p").hide(1000);

alert("The paragraph is now hidden");

});

});

Node JS:

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](https://developers.google.com/v8/). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [npm](https://www.npmjs.com/), is the largest ecosystem of open source libraries in the world.

DOCS: <https://nodejs.org>

1. What is Module?

Ans:

To execute a node program:

Node has to be installed🡪 node -version🡪node checker.js

Codes:

## Detail Description of the Projects:

Initial Project: Node-app

1] 01\_helloNodeWorld.js

console.log("Hello world");

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

console.log("Iterator:"+index);

}

var a={

'name':'kingshuk',

'age': '26'

}

console.log(a);

2] 02\_dataTypes.js

//Order of eval and how stirng affects the eval

var x = 16 + "Volvo";

console.log(x);

var x = "16" + "Volvo";

console.log(x);

var x = 16 + 4 + "Volvo";

console.log(x);

var x = "Volvo" + 16 + 4;

console.log(x);

//Double v/s Single Quotes

var answer = "It's alright"; // Single quote inside double quotes

console.log(answer);

var answer = "He is called 'Johnny'"; // Single quotes inside double quotes

console.log(answer);

var answer = 'He is called "Johnny"'; // Double quotes inside single quotes

console.log(answer);

//Numbers

var x1 = 34.25; // Written with decimals

console.log(x1);

var x2 = 34; // Written without decimals

console.log(x2);

var y = 123e5; // 12300000

console.log(y);

var z = 123e-5; // 0.00123

console.log(z);

//Array

var cars = [ "Saab", "Volvo", "BMW" ];// Now x is a array

console.log(cars);

console.log("Type:"+typeof cars);

//Typeof examples on various datatypes

var yy; // Now y is undefined

console.log(typeof yy);

var x = 5; // Now x is a Number

console.log(typeof x);

var x = 3.14; // Now x is a Number

console.log(typeof x);

var x = "John"; // Now x is a String

console.log(typeof x);

var x = false; // Now x is a boolean

console.log(typeof x);

var x = true; // Now x is a boolean

console.log(typeof x);

var x = {name:'John', age:34}; // Now x is a object

console.log(typeof x);

//Null and undefined

var person1 = null; // Value is null, but type is still an object

console.log(typeof person1);

var person2 = undefined; // Value is undefined, type is undefined

console.log(typeof person2);

console.log();

03] 02\_objects1.js

var employee={

id : 2343,

name: "Kingshuk",

employeecard : function(){

return this.id + " "+this.name;

}

}

console.log(employee.employeecard());

var myHonda={

color:"red",

wheels:4,

engine: {cylinders:4,size:2.22},

getInfo : function(){

var result="A beautiful "+this.color + " car with "+this.engine.cylinders+" cylinders with "+this.engine.size+" size";

console.log(result);

}

}

console.log(myHonda);

myHonda.getInfo();

var myMaruti={

color : "green",

wheels:4,

engine: {cylinders:4 ,size:2.2},

}

console.log(myMaruti);

console.log(myMaruti.color);

console.log(myMaruti.wheels);

console.log(myMaruti.engine.size);

04] 02\_objects2.js[JavaScript Constructor Use & Prototype Explanation]

function Car(pColor,pWheels,pEngine){

this.color=pColor;

this.wheels=pWheels;

this.engine=pEngine;

this.drive=function(mode){

console.log("Driving in : "+mode+" Mode");

}

}

Car.prototype.getInfo = function(){

var result="A Beautiful " + this.color+" car with "+this.engine.cylinders+" cylinders " + this.engine.size;

console.log(result);

}

Car.pat=20;

var honda=new Car("red",4,{cylinders:4,size:2.2});

console.log(honda);

honda.drive("fast");

honda.getInfo();

console.log(honda.pat);//here this will be undefined since you have not defined Car.prototype.pat

03\_moduleExport[Export a js file and use it some other file]

**Cals.js**

var Calculator = {

add : function (a,b) {

return a+b;

},

sub : function (a,b) {

return a-b;

},

mul : function (a,b) {

return a\*b;

}

}

var tempFunction=function () { //this is basically a function which is assigned to

//a variable without name

return "This is Temp function";

};

var tempVar="This is not visible";

module.exports=Calculator;

module.exports.newFunction=tempFunction; //We have to give the function a name inorder to access

module.exports.tempVar=tempVar;

**For accessing the module:**

var calc=require("./calc.js");

console.log(calc);

console.log("2+3 =" + calc.add(2,3));

console.log("2-3 =" + calc.sub(2,3));

console.log("2\*3 =" + calc.mul(2,3));

console.log(calc.newFunction());

console.log(calc.tempVar);

**Node JS Features:**

* Non-Blocking IO.
* Event Looping

1. What is the difference between Blocking(Synchronous Call ) and Non-Blocking (Asynchronous)?

Ans:

Here there is a synchronous call, till the DB returns the thread is blocked.

// Example 1 - Synchronous (blocks)

var result = database.query("SELECT \* FROM hugetable");

console.log("Query finished");

console.log("Next line");

Output:

Query finished  
Next line

Here Asynchronous Call[Non-Blocking]- So multi-threading possible

// Example 2 - Asynchronous (doesn't block)

database.query("SELECT \* FROM hugetable", function(result) {

console.log("Query finished");

});

console.log("Next line");

Next line

Query finished

04\_readFile:

var fs=require('fs');

var args=process.argv.length; //No of arguments passed in the command line

console.log(process.argv);//Pring the argumets array

if(args < 3){ //Checking the no of i/p arguments from commandline is 3

console.log("Usage: node 04\_readFile.js <file>");

process.exit(1);

}

var total= 0;

for(i=2;i<args;i++){

console.log(i);

total ++;

}

console.log("Total # of args provided to this program were: "+ total);

var fileToRead=process.argv[2]; //file to read

var contents=fs.readFileSync(fileToRead);

var lines=contents.toString().split('\n').length;

console.log(fileToRead + " had "+ lines + " lines in it");

1. What are callbacks?

Ans: A callback function is another function that can be passed as an argument to another function, and it is usually evoked after some kind of event.

Very Good Explanation: <https://www.youtube.com/watch?v=haz4SBcEYAw>

**Callbacks:**

A callback is a function passed to some asynchronous process that should be called when the process has completed.

Callbacks are just a specific case of functions being passed as a parameter in JavaScript.

**Anatomy of Callback:**

* **By convention, we pass the callback as the last parameter to any asynchronous function.**

**(param1, param2, … callback)**

* **The callback should always expect parameters in the form of (error, … otherReturnedValues)**



**05\_fileFilterASync:**

var fs=require("fs");

var path=require("path");

var noOfFiles1=0;

var filterListInSync=function anyNameorWithouName(dir,filterstr,callback) {

fs.readdir(dir,function(err,list) {

if(err){

return callback(err);

}

list=list.filter(function(file) {

noOfFiles1++;

console.log("::"+noOfFiles1);

return path.extname(file) === "."+filterstr;

});

console.log("call solution provided");

callback(null,list);

});

}

//callbackFunction(error, ...otherReturnedValues)

function fileFilterCallback(err,list) {

console.log("=======");

if (err) {

return console.log("Error occured : ",err);

}

var noOfFiles=0;

list.forEach(function(file){

noOfFiles ++;

});

console.log("Total no of files" + noOfFiles);

}

var dir=process.argv[2];

var filterstr=process.argv[3];

filterListInSync(dir,filterstr,fileFilterCallback);

//mainFunction(parameter1,parameter2,callbackFunction)

//run command: node 05\_fileFilterASync.js "D:\Node Workspace\node-app" js

//it will count the no of files with js extention

**qrCodeDemo.js :**

First you need to install the Module qr-image

Command : npm install qr-image -G

**Then Run the code:**

var qr=require('qr-image');

var fs=require('fs');

var code=qr.image('https://developer.mozilla.org/en-US/',

{type : 'svg'});

var output=fs.createWriteStream('fmQrcode.svg');

code.pipe(output);

The QR- Image will be generated for the given Website.

Scan it with a scanner.

**Launching Hello world Web server:**

**06\_helloWorldWS.js**

const http = require("http");

const host="localhost";

const port=3000;

var server=http.createServer(function (request, response) {

response.writeHead(200, {'Content-Type': 'text/plain'});

response.end('Hello World............NodeJs server...........');

}).listen(port,host,function(){

console.log('Server running at http://localhost:3000/');

});

Creating A custom Node Project With all specification:

* Create a new Folder, 02\_node-app-custom-project-qrImage
* Then Command: npm init [Prior npm node package manager has to be there]

Add all the project specifications.

* Install the required package by the Command, npm install[Details of the command: <https://docs.npmjs.com/cli/install> ] npm install qr-image -D[Installed as development dependencies]
* npm install 🡪By default will install all modules listed as dependencies in [**package.json**](https://docs.npmjs.com/files/package.json)
* Then execute the code,

Index.js

var qr=require('qr-image');

var fs=require('fs');

var code=qr.image('http://115.112.141.37:8080/',{type : 'png'});

var output=fs.createWriteStream('generatedQrcodeofJenkins.png');

code.pipe(output);

QR code of the URL will be generated.

## Protractor:

## Suite Block

Jasmine is a testing framework for JavaScript. **Suite** is the basic building block of Jasmine framework. The collection of similar type test cases written for a specific file or function is known as one suite. It contains two other blocks, one is **“Describe()”** and another one is **“It()”**.

One Suite block can have only two parameters, one **“name of that suite”**and another **“Function declaration”** that actually makes a call to our unit functionality that is to be tested.

## Jasmine Framework:

## What is Jasmine?

## Ans: Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests.

## Official Link: <https://jasmine.github.io/>

## Very Good official Docs :<https://jasmine.github.io/2.1/introduction>

## Video Tutorials: <https://www.youtube.com/watch?v=Ac2sBrkSK-c&list=PL_noPv5wmuO9op-OQ22SbHcqFGGHA6iIZ>

## Project Jasmine:

## Create a directory 04\_Jasmine\_Basics and switch to it

## npm install jasmine -g

## Then jasmine init 🡪 a spec folder gets created[And the Jasmine basically looks for the spec folder]

## Write all the tests inside the spec folder.

describe() Block:

Suite is the basic building block of Jasmine framework. The collection of similar type test cases written for a specific file or function is known as one suite.

A test suite begins with a call to the global Jasmine function describe with two parameters: a string and a function. The string is a name or title for a spec suite - usually what is being tested. The function is a block of code that implements the suite.

it() Block :

Specs are defined by calling the global Jasmine function it, which, like describe takes a string and a function. The string is the title of the spec and the function is the spec, or test. A spec contains one or more expectations that test the state of the code. An expectation in Jasmine is an assertion that is either true or false. A spec with all true expectations is a passing spec. A spec with one or more false expectations is a failing spec.

Expectations:

Jasmine has built in checks call Expectations.

Expectations are built with the function expect which takes a value, called the actual. It is chained with a Matcher function, which takes the expected value.

Integrating reports in protractor: <https://www.npmjs.com/package/protractor-jasmine2-html-reporter>

Data provider: <https://moduscreate.com/blog/protractor-and-jasmine-data-provider-write-once-test-many/>

# **Various TimeOut issues in protractor:**

[https://github.com/angular/protractor/blob/master/docs/timeouts.md#angular]

# Timeouts

Because WebDriver tests are asynchronous and involve many components, there are several reasons why a timeout could occur in a Protractor test.

**Timeouts from Protractor**

### **Waiting for Page to Load**

When navigating to a new page using browser.get, Protractor waits for the page to be loaded and the new URL to appear before continuing.

* Looks like: an error in your test results - Error: Timed out waiting for page to load after 10000ms
* Default timeout: 10 seconds
* How to change: To change globally, add getPageTimeout: timeout\_in\_millis to your Protractor configuration file. For an individual call to get, pass an additional parameter: browser.get(address, timeout\_in\_millis)

### **Waiting for Angular**

Before performing any action, Protractor waits until there are no pending asynchronous tasks in your Angular application. This means that all timeouts and http requests are finished.

* Looks like: an error in your test results - Timed out waiting for asynchronous Angular tasks to finish after 11 seconds.
* Default timeout: 11 seconds
* How to change: Add allScriptsTimeout: timeout\_in\_millis to your Protractor configuration file.

You may also need to fix this problem with a change to your application.

#### **AngularJS**

If your AngularJS application continuously polls $timeout or $http, Protractor will wait indefinitely and time out. You should use the [$interval](https://github.com/angular/angular.js/blob/master/src/ng/interval.js) for anything that polls continuously (introduced in Angular 1.2rc3).

#### **Angular**

For Angular apps, Protractor will wait until the [Angular Zone](https://medium.com/@MertzAlertz/what-the-hell-is-zone-js-and-why-is-it-in-my-angular-2-6ff28bcf943e) stabilizes. This means long running async operations will block your test from continuing. To work around this, run these tasks outside the Angular zone. For example:

this.ngZone.runOutsideAngular(() => {

setTimeout(() => {

// Changes here will not propagate into your view.

this.ngZone.run(() => {

// Run inside the ngZone to trigger change detection.

});

}, REALLY\_LONG\_DELAY);

});

As an alternative to either of these options, you could disable waiting for Angular, [see below](https://github.com/angular/protractor/blob/master/docs/timeouts.md#how-to-disable-waiting-for-angular).

### **Waiting for Angular on Page Load**

Protractor waits for the angular variable to be present when loading a new page.

* Looks like: an error in your test results - Angular could not be found on the page: retries looking for angular exceeded
* Default timeout: 10 seconds
* How to change: To change globally, add getPageTimeout: timeout\_in\_millis to your Protractor configuration file. For an individual call to get, pass an additional parameter: browser.get(address, timeout\_in\_millis)

### ***How to disable waiting for Angular***

If you need to navigate to a page which does not use Angular, you can turn off waiting for Angular by settingbrowser.waitForAngularEnabled(false). For example:

browser.waitForAngularEnabled(false);

browser.get('/non-angular-login-page.html');

element(by.id('username')).sendKeys('Jane');

element(by.id('password')).sendKeys('1234');

element(by.id('clickme')).click();

browser.waitForAngularEnabled(true);

browser.get('/page-containing-angular.html');

**Timeouts from WebDriver**

### **Asynchronous Script Timeout**

Sets the amount of time to wait for an asynchronous script to finish execution before throwing an error.

* Looks like: an error in your test results - ScriptTimeoutError: asynchronous script timeout: result was not received in 11 seconds
* Default timeout: 11 seconds
* How to change: Add allScriptsTimeout: timeout\_in\_millis to your Protractor configuration file.

**Timeouts from Jasmine**

### **Spec Timeout**

If a spec (an 'it' block) takes longer than the Jasmine timeout for any reason, it will fail.

* Looks like: a failure in your test results - timeout: timed out after 30000 msec waiting for spec to complete
* Default timeout: 30 seconds
* How to change: To change for all specs, add jasmineNodeOpts: {defaultTimeoutInterval: timeout\_in\_millis} to your Protractor configuration file. To change for one individual spec, pass a third parameter to it: it(description, testFn, timeout\_in\_millis).

**Timeouts from Sauce Labs**

If you are using Sauce Labs, there are a couple additional ways your test can time out. See [Sauce Labs Timeouts Documentation](https://docs.saucelabs.com/reference/test-configuration/#timeouts) for more information.

### **Maximum Test Duration**

Sauce Labs limits the maximum total duration for a test.

* Looks like: Test exceeded maximum duration after 1800 seconds
* Default timeout: 30 minutes
* How to change: Edit the "max-duration" key in the capabilities object.

### **Command Timeout**

As a safety measure to prevent Selenium crashes from making your tests run indefinitely, Sauce limits how long Selenium can take to run a command in browsers. This is set to 300 seconds by default.

* Looks like: Selenium took too long to run your command
* Default timeout: 300 seconds
* How to change: Edit the "command-timeout" key in the capabilities object.

### **Idle Timeout**

As a safety measure to prevent tests from running too long after something has gone wrong, Sauce limits how long a browser can wait for a test to send a new command. This is set to 90 seconds by default. You can adjust this limit on a per-job basis.

* Looks like: Test did not see a new command for 90 seconds. Timing out.
* Default timeout: 90 seconds
* How to change: Edit the "idle-duration" key in the capabilities object.

Working with Excel:

<https://www.npmjs.com/package/exceljs>

>npm install exceljs --dev-save