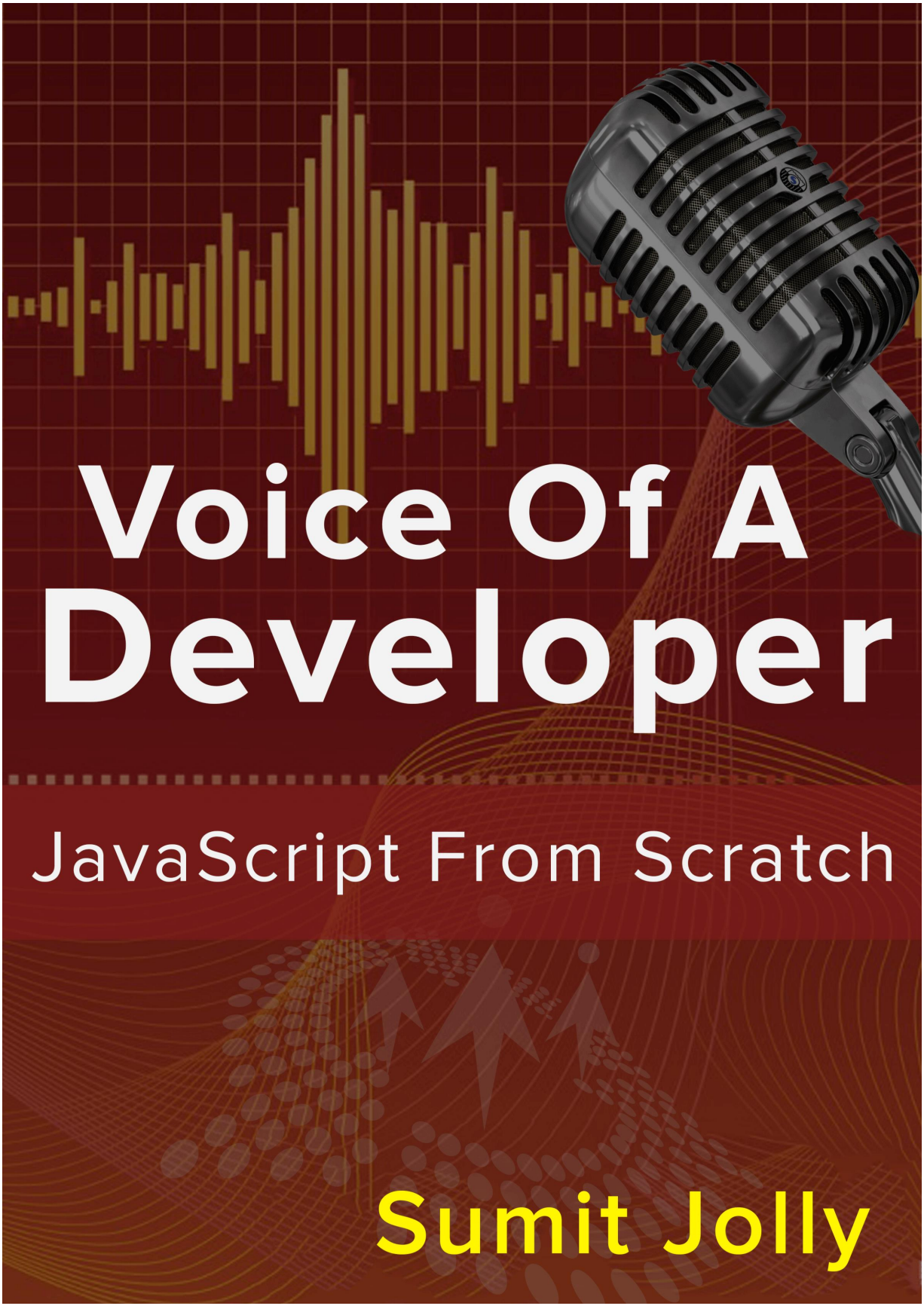

Voice Of A
Developer
JavaScript From Scratch
Sumit Jolly

# Voice of a Developer: JavaScript From Scratch



**Sumit Jolly**
(Microsoft & C# Corner MVP)

# Table of Index

## About The Author

*Sumit Jolly is a Microsoft MVP and two times C# Corner MVP, working as "Technology Architect" and love programming. He possess over a decade of experience working on Microsoft technologies, open source softwares, cloud, databases etc. He uses technology to solve business problems and make someone life "easy". He believes in The Pareto principle (also known as the 80–20 rule). In 20% time he contributes back to community, i.e., to help other developers, share knowledge via different medium articles, speaking etc.*

**Sumit Jolly**

(Microsoft & C# Corner MVP)

# JavaScript Objects

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript. What mistakes a developer generally makes and what differences they should be aware of.

**Q: Can you use save as a  reference in JavaScript like we have in other programming languages?**

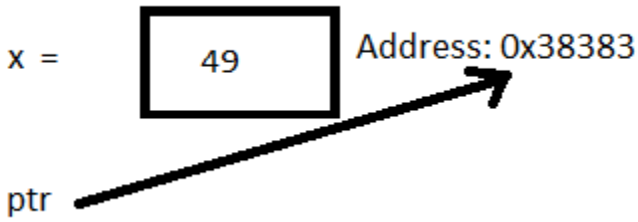
**Ans:**

Yes, let's go back in C / C++ days where we have a concept of pointers,

**ex:**

```
int x=49;
int *ptr;
ptr = &x;
```

This means that address of x is stored in ptr.




Therefore, ptr is pointing to integer. The advantage of this is you could change value in x via pointer

**Ex:**

```
*ptr=50;
printf("%d", x); // 50
```

Similarly, in JavaScript the primitive data is saved-as-value. On the contrary, save-as-reference mechanism is there for objects,

**Ex:**

```
> var person = {name: "Sumit"};
> var anotherPerson = person;
> person.name = "Ravi";
> console.log(anotherPerson.name); // Ravi
> console.log(person.name); // Ravi
```

The variable is a reference, not a value

**Object Own and Inherited Properties**

The own properties are properties that were defined on the object, while the inherited properties were inherited from the object's Prototype object.

We learnt about own properties in part 1, ex- person.name // own property

**Prototypal inheritance – Inherited property**

In most OOPS based languages, classes inherit from classes. In JavaScript, it's prototype-based. Objects inherit from Objects. And, we know an object is a prototype of classes. Therefore, it's prototypical based inheritance

**Q: How do we implement prototypal inheritance?**

**Ans:** Use property called __proto__

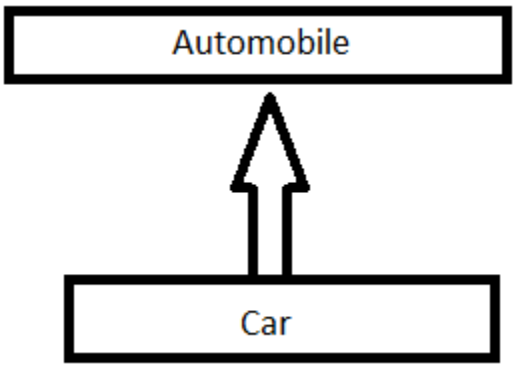
**Ex:**

```
> var automobile = {engine: true};
> var car = {tyres: 4};
```

Now, we know car is a type of automobile, so we could inherit from automobile.

> car.__proto__ = automobile; //inheritance
> car.engine; // true

Hence, there is CAR => Automobile (car is a automobile)



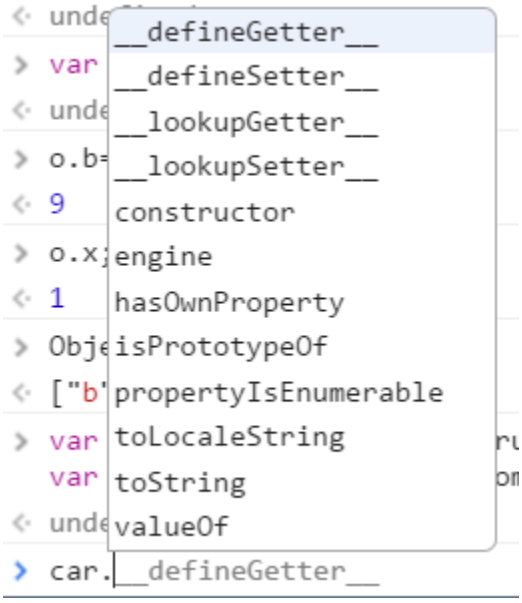

| Enumerable | Non-Enumerable |
|---|---|
| Own properties are enumerable. This means you can iterate over these | Inherited properties are non-emunerable. These are hidden unless asked specifically |
| Ex- car.tyres | Ex car.engine |
| Object.keys (car) //tyres; | |
| car.propertyIsEnumerable("tyres"); //true | car.hasOwnProperty("engine"); //false |
| car.hasOwnProperty("tyres"); // true | car.propertyIsEnumerable("engine"); //false |

**Q: Is __proto__ only way to prototypal inheritance?**

**Ans:** No, you could you Object to create it,

**ex-**
var automobile = {engine: true};

var car = Object.create(automobile);




Now, you could see car.engine is available

**Q: Is there any other way to check property in object?**

**Ans:** Yes, you could use below pattern to validate,

**ex-**
> "engine" in car; //true
> "valueOf" in {}; //true

# JavaScript Data Types

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript and what mistakes generally developers make and what differences they should be aware of.

**Data types**

A very common word and very basic in programming language, JavaScript provides different data types to hold different types of values.

**Q: How many data types are there in JS?**

**Ans:** There are two types of data types in JavaScript.

**Primitive data type**

- String
- Number
- Boolean
- Undefined
- Null

**Non-primitive (reference) data type**

- Object
- Array
- RegExp

**Q: How do you specify data type in JS?**

**Ans:** It's simple and could be defined using var keyword only

**Ex-**

```
var x=10; // holds number
var x=”javascript”; // holds string
```

As a developer, I am always curious to know the size of datatypes. But there is no sizeof function available which provides me size in JavaScript. But it’s good to know memory consumption when we work on bigger applications.

**Download sizeof.js**

Download one of the files below and upload it to your web server.

**Example-**

```
// define an object
var object =
    {
      'boolean' : true,
      'number'  : 1,
      'string'  : 'a',
      'array'   : [1, 2, 3]
    };

// determine the size of the object
var size = sizeof(object);
//output: size is 92 bytes
```

**Q: What is the difference in Undefined and Null?**

**Ans:** In JavaScript, undefined means a variable has been declared but has not yet been assigned a value, such a,

```
var TestVar;
alert(TestVar); //shows undefined
alert(typeof TestVar); //shows undefined
null is an assignment value. It can be assigned to a variable as no value:
```

```
var TestVar = null;
alert(TestVar); //shows null
alert(typeof TestVar); //shows object
```

**Objects**

Objects are variables too. However, objects can contain many values. Ex-

```
var person = {age:59, address:"Delhi”};
```

**Q: Do you know simple way to create objects?**

**Ans:**

```
var obj={};
//undefined
typeof(obj);
"object"
Object Properties
You can access object properties in two ways:
objectName.propertyName
or
objectName["propertyName"]
```

**Ex-**

```
person.age
Or
Person[“age”]
```

**Object Methods**

You access an object method with the following syntax:

objectName.methodName()

**Example**

```
var person = {age:59, address:"Delhi", fullName:function(){console.log('John Doe');return 'John Doe';}};
```

If you access the property without parenthesis (), it will return the function definition:

**person.fullName**

```
// function () {console.log('John Doe'); return 'John Doe';}
Otherwise it'll execute the method
person.fullName();
// John Doe
```

**Loop over all properties in Object**

We often use FOR statement to loop, but there is another feature, for in, which is pretty useful to iterate over all properties of Object

**Ex-**

```
for(a in person){console.log(a)}
age
address
fullName
```

**Q: How do you add/delete properties to existing object?**

**Ans:** It is simple in JavaScript, by giving a value

**Ex-**

```
person.country = "India";
```

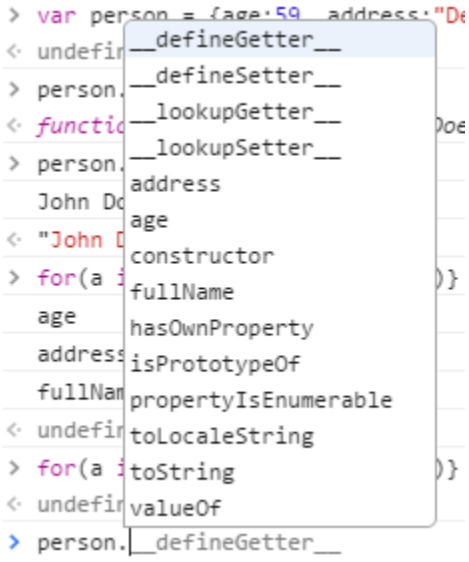
To delete you could use delete keyword

**Ex-**

delete person.age;

**Access operator**

Go to browser console window and try to access all properties /methods via access operators, ex-




**Q: Can you access a property, which do not exist in Object?**

**Ans:** Yes, you can access a property on an object that does not exist but will result in undefined.

**Ex-**

```
person.lastName;
//undefined
```

**Q: Can you access a method, which do not exist in Object?**

**Ans:** No, you cannot access a method on an object that does not exist because it'll yield an error.

**Ex-**

```
1. person.lastName();
```

```
> person.lastName()
✖ ▼Uncaught TypeError: person.lastName is not a function(…)
      (anonymous function)            @ VM651:2
      InjectedScript._evaluateOn      @ VM43:878
      InjectedScript._evaluateAndWrap @ VM43:811
      InjectedScript.evaluate         @ VM43:667
```

## JavaScript Engines

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript, and what mistakes developers generally make and what differences they should be aware of.

**Let's start with Q&A:**

**Q: What is the difference between interpreted v/s compiled languages?**

**Ans:**
Compiled: In compiled language, a compiler will translate the program directly into code that is specific to the target machine, which we call machine code. Then the computer will run the machine code on its own.

Interpreted: In an interpreted language the code is not directly run by the machine; there is another program which reads and then executes the code. This other program is known as the interpreter.

**Q: Is JavaScript an interpreted language or compiled language?**

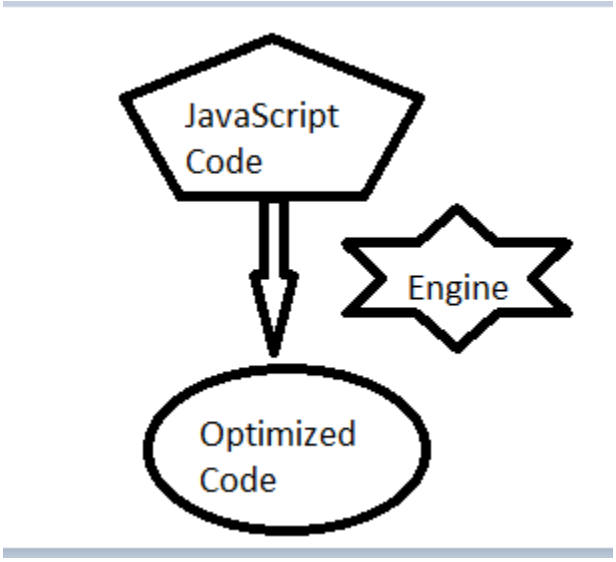
**Ans:**
Historically it was designed as interpreted language.
JavaScript differs from traditional C, C++ languages. JavaScript is JIT (Just-in-time) compiled. The reason behind this approach is for huge speed gains which modern JavaScript engines provide.

**Q: Which category of language JavaScript belongs?**

**Ans:** JavaScript is considered as a high-level language. It's readable by humans easily.

**Q: What is a JavaScript engine?**

**Ans:** It's a program or library, which executes JavaScript code.




**Evolution of JavaScript engines**

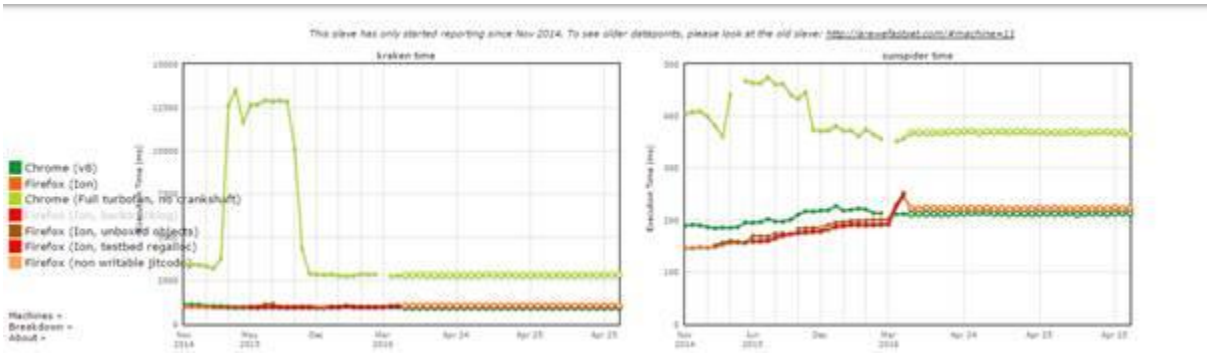
The 1st JavaScript engine was created by Brendan Eich in late 1990s for Netscape. It was named SpiderMonkey, and was written in C++.

Since after then there were many engines launched. Here is the list of popular ones:

| Browser | JavaScript Engine |
|---|---|
| Mozilla | SpiderMonkey |
| Chrome | V8 |
| IE & Edge | Chakra |
| Node.js | V8 |
| PhantomJS (headless browser) | JavaScript Core |

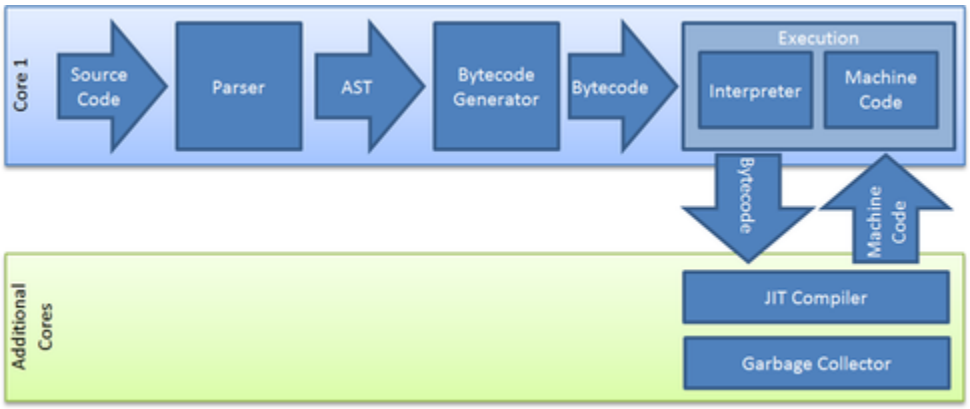
**Q: Which JavaScript engine scores high?**

**Ans:** You can compare how various engines perform in benchmarking produced on http://arewefastyet.com/




**Q: How JavaScript engine works?**

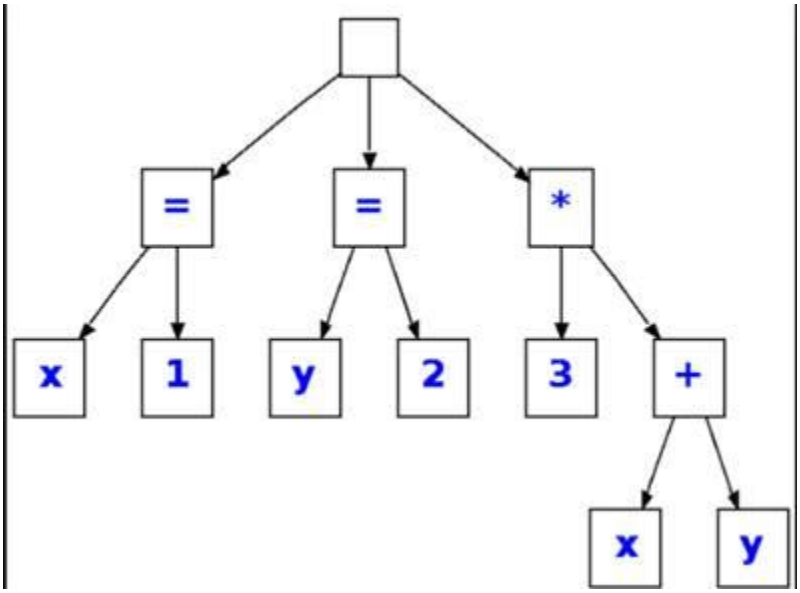
**Ans:** Here is the process,




The parser takes JavaScript and generates a Abstract Syntax Tree (AST).

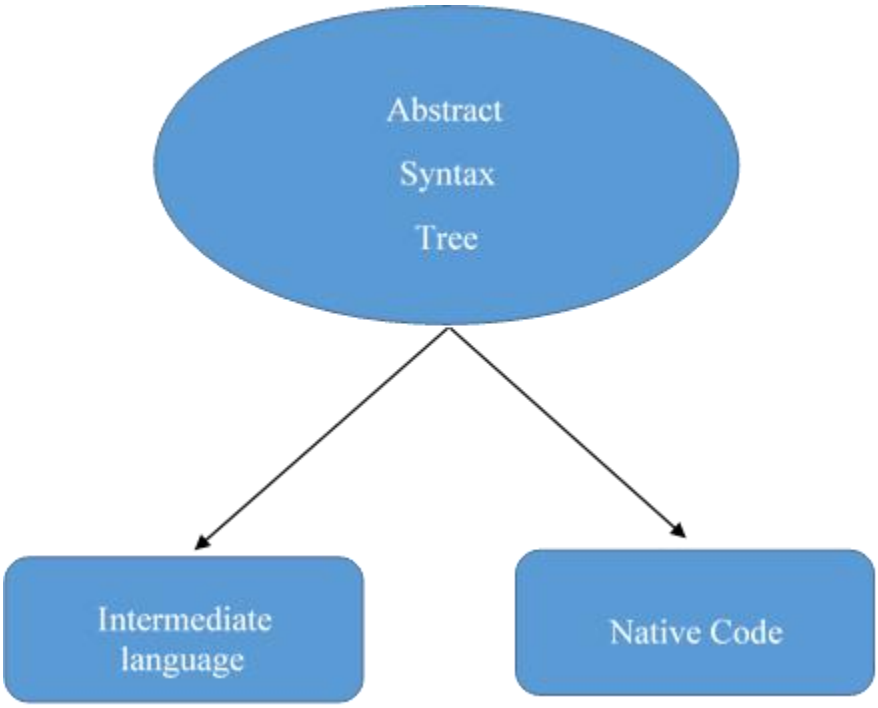
**AST:** It's just a syntax tree, and a tree representation of source code,

**ex:**




After AST is generated, there are two choices depending upon JavaScript engine to generate bytecode

For ex- Google V8 & Chakra generates Native code and both are really fast engines to run JavaScript.



## Common Mistakes

JavaScript is language of the Web. This part will talk about my observations, learned during my decade of software development experience with JavaScript. What mistakes developers generally make and what differences they should be aware of.

As a developer, sometimes I face issues because of my silly programming mistakes and I have learned from these. So, I found pattern which programmers do, and I'll try to save your debugging time, so you don't waste time like I did in past.

**Mistake 1: Equality checks**
We check variables for certain values but there are two types of operators we could use. I suggest using strict operators over type converting.

Type converting operator like == converts operands if they are not of same type. To demonstrate or test you could try the following statements and see output.

```
1. 1 == 1 // true
2. "1" == 1 // true
3. 1 == '1' // true
4. 0 == false // true
```

So, the mistake of using == operator results in TRUE value for 1=="1", which could be wrong in a real scenario. Therefore, it is always advisable to use type strict operators.

Strict operators like === don't convert operands and returns true if the operands are strictly equal.

```
1. 1 === 1 // true
2. "1" === 1 // false
3. 0 === false // true
```

**Mistake 2: Concatenation**

JavaScript uses + operator for concatenation & addition. Now, another common mistake is to use mix number & string operands. For ex-

```
1. var y = '10';
2. var x= 1 + y; // will give 110
```

Use function parseInt in such scenario to rescue you, otherwise you'll spend time in debugging.

```
1. var x= 1 + parseInt(y); // will give 11
```

**Mistake 3: Float point numbers**

Programming languages are not always correct to match floating point numbers. And, let's see right way to match floating point numbers. For ex-

```
1. var f = 0.1;
2. var g = 0.2;
3. var h= f + g;
4. (h === 0.3) // false because h is 0.30000000000000004
```

Floating numbers are saved in 64 bits. Hence, the right way to do is:

```
1. var h = (f * 10 + g * 10) / 10; // h is 0/3
2. (h === 0.3) // true
```

**Mistake 4: Not using var to declare variables**

It's a common mistake to not use var keyword and declare variables. JavaScript will not give an error if you skip var keyword but the scope will be different of variables.

**Scenario 1**

```
1. var foo = ‘test’;
2. console.log (foo); // will print test
```

**Scenario 2**

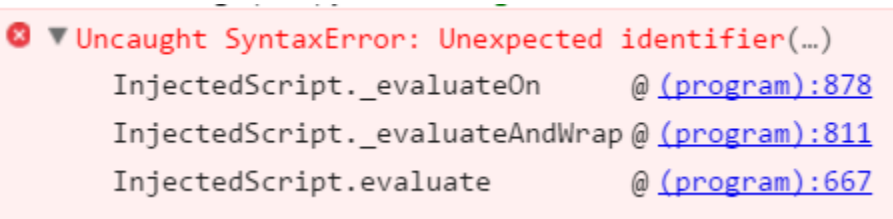
```
1. foo = ‘test’;
2. console.log (foo); // will print test
```

**Q: What is the difference in two approaches?**

Ans: It depends upon the scope where you declare the variable. Let me revise the code as below:

**Scenario 1**

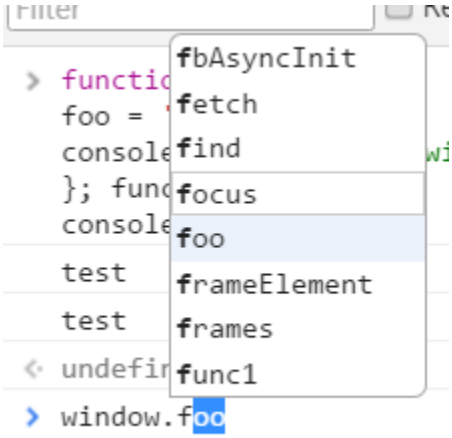
```
1. function func1(){
2. var foo = 'test';
3. console.log (foo); // will print test
4. }; func1();
5. console.log (foo);
```




**Scenario 2**

```
1. function func1(){
2. foo = ‘test’;
3. console.log (foo); // will print test
4. }();
5. console.log (foo); // will print test
```

The reason why Scenario 2 worked because foo gets created in global scope, i.e., in WINDOW object of JavaScript,




In bigger projects, if you add more variables in global scope then it will consume allocated memory and not free up, which could potentially result in memory leaks issue.

**Mistake 5: Usage of undefined as a variable**

Often we use undefined as a variable. Let's take a scenario below:

```
> myVar === undefined
▶ Uncaught ReferenceError: myVar is not defined(…)
```

The correct way is to use typeof when checking for undefined:

```
> typeof myVar === "undefined"
< true
```

I hope you'll find it useful and you can give your suggestions in comment box, if you know some other mistake pattern.

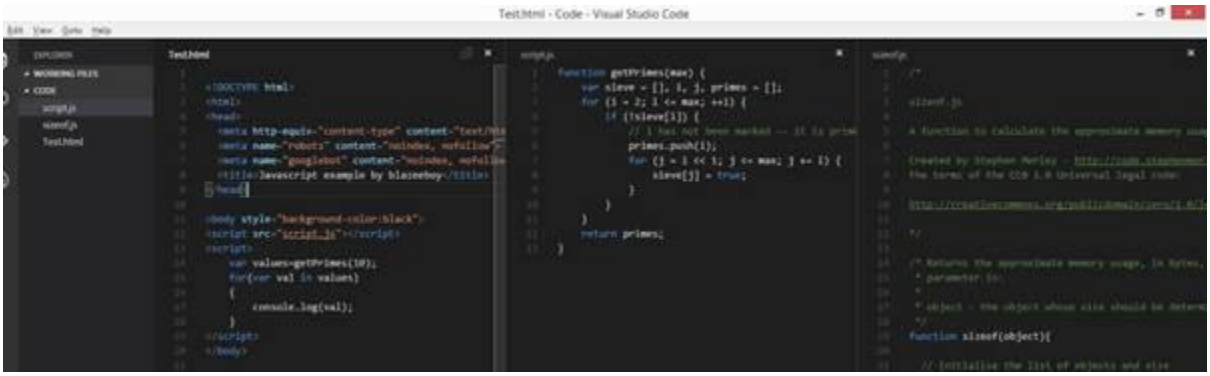
## JavaScript Editors

JavaScript is language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

Developers need editors to write code. It is recommended to use a good editor, which provides many features to increase agility. I explore different editors both in backend and frontend technologies. On Windows platform, for JavaScript I tried three editors, VS Code, Sublime & Webstorm. I will talk about these briefly in this part and in my opinion, I find VS Code as the best editor to work with.

### VS Code

Side-by-Side Editing is what I like the most. You can open up to 3 files parallel for editing in VS Code. To do this, click file by pressing Ctrl key,



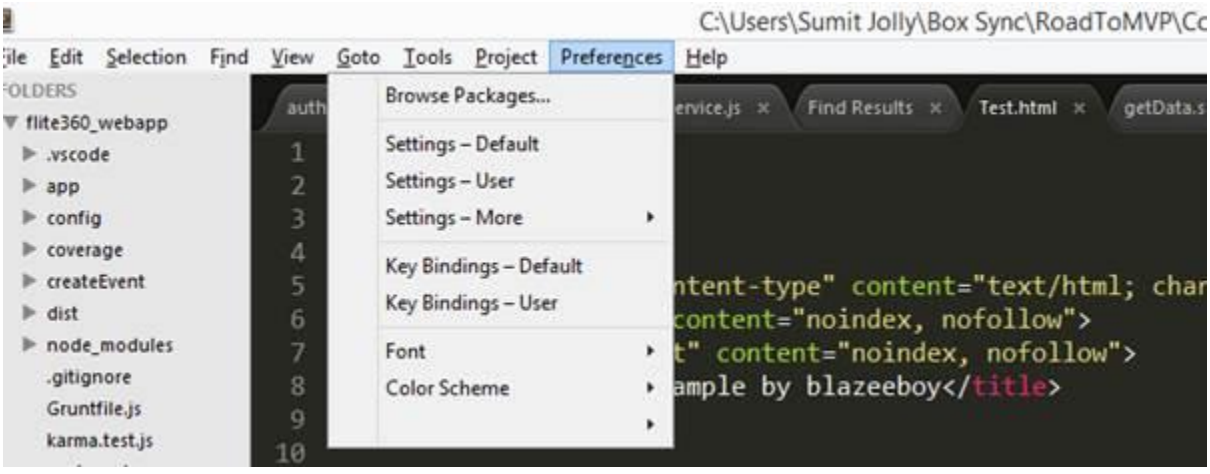
**IntelliSense:** VS Code inherits Visual Studio intelliSense, which is liked by developers. Therefore, they can quickly write code.

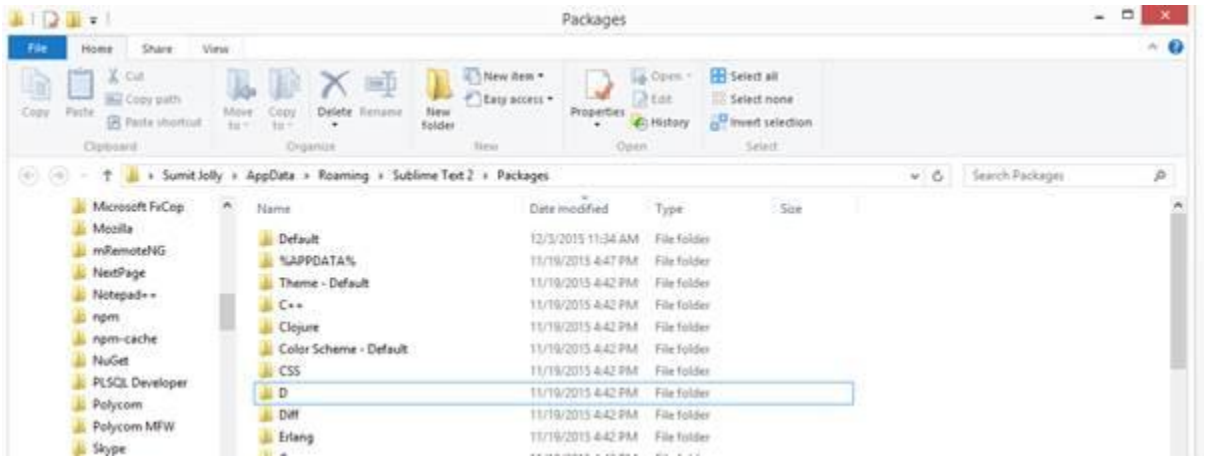
VS Code comes with Git integration so it is easy for Developers to integrate with online code repository

### Sublime Text

It is a very nice editor to write JavaScript. There are many features, which you shall try to explore.

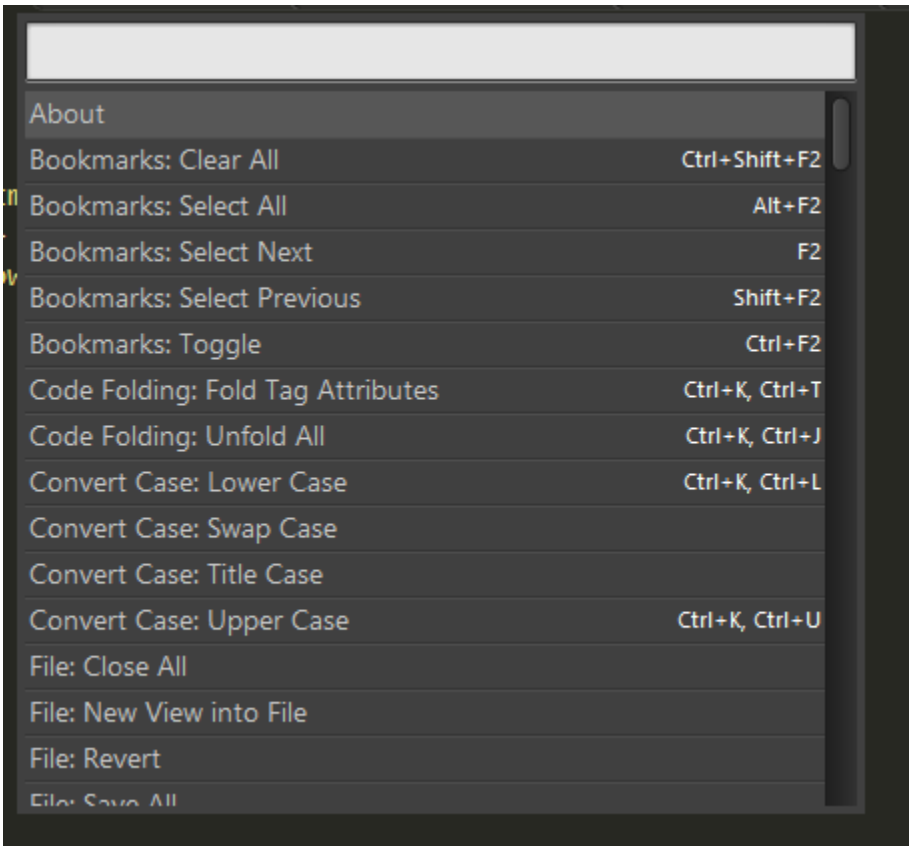

Package Control is a package manager for Sublime that allows you to easily install any of the hundreds of awesome plugins made by other developers. You can go to 'Browse Packages' and look at existing packages available.






## Command Palette

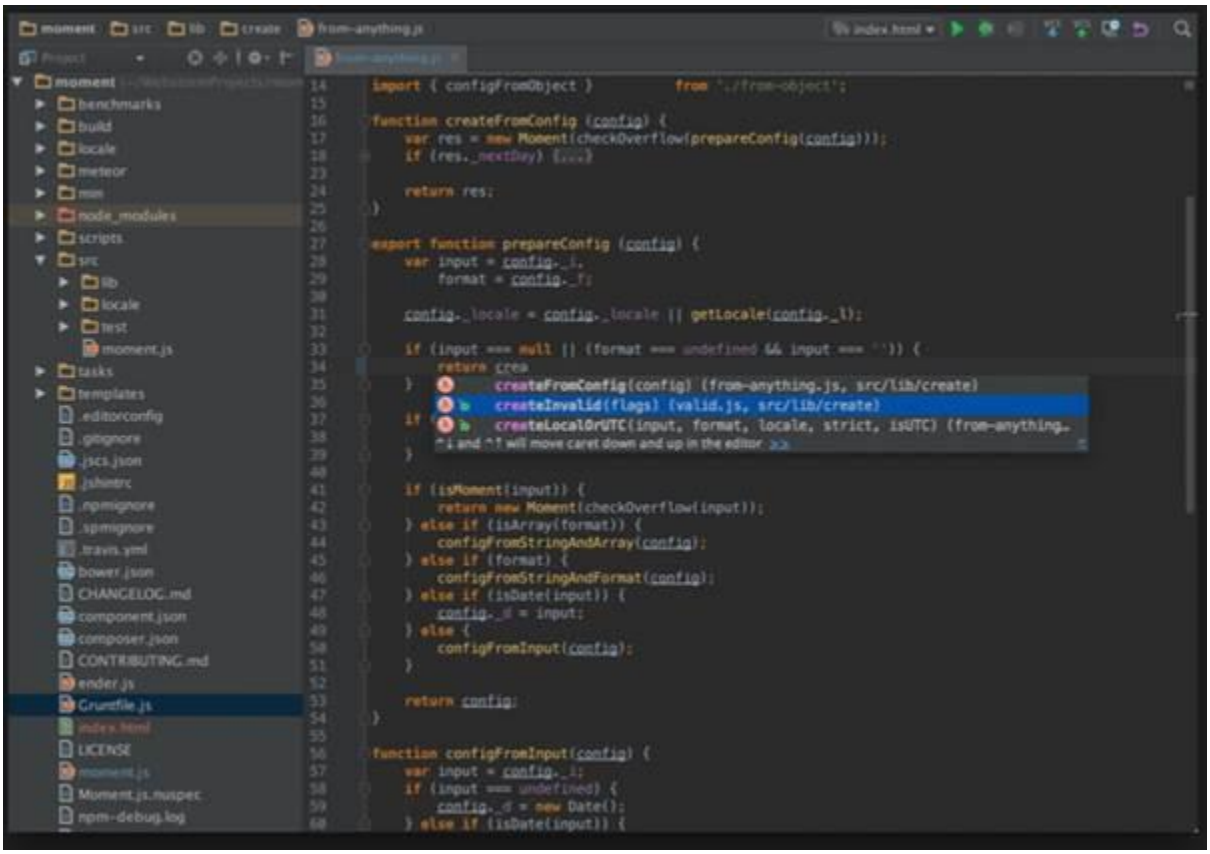
This is one of the most used commands. In Windows, you can use shortcut ctrl + shift + p to invoke it,



You can explore more features like Plugin, Build etc. that are cool in Sublime.

**Webstorm**

It is developed by JetBrains, but only free to try for 30 days. The main advantages of using Webstorm is it incorporates Google JSTestDriver. If you are a fan of TDD and write front end unit tests then it is a good editor for you to work with.



I like a few plugins ex- Live Edit: it immediately allows you to see changes in browser.

# JavaScript VSCode

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

I believe a good editor makes life easy for a developer. My favorite editor is VSCode.

**About VSCode**

As stated,

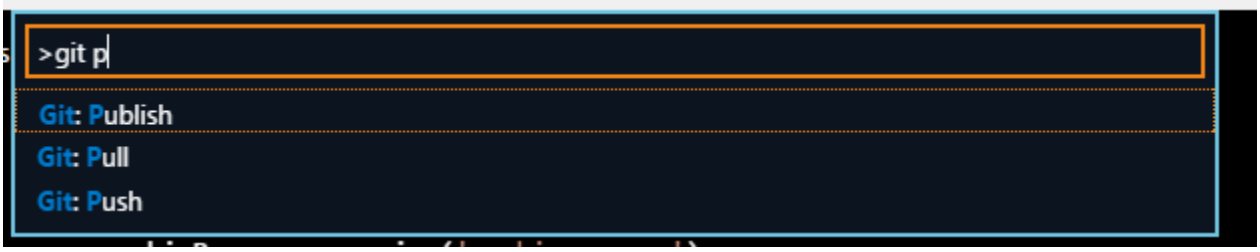
"Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, OS X and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (C++, C#, Python, PHP) and runtimes.".

In the last series I talked about a few editors, but what makes VSCode awesome? In my point of view VSCode is breath of fresh air. Let's deep dive:

**Command Palette**

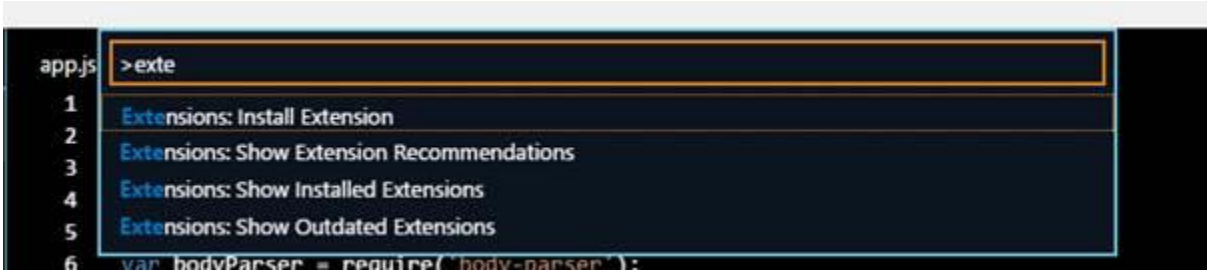
A very handy tool so developers can fire commands from editor only. Press Ctrl+Shift+P or F1 and a pop up will open,


Now you can search your respective command, ex,

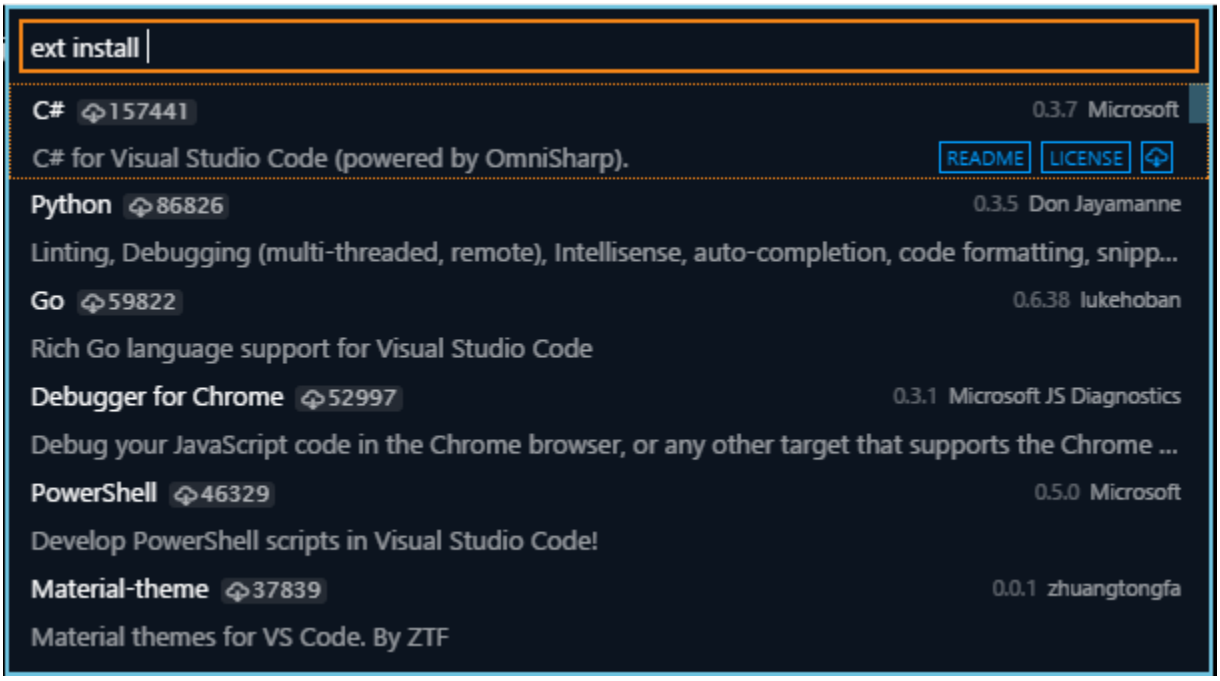



**Extensions**

It supports 30+ programming languages out-of-the-box. You can install extensions by opening Command Palette.




Select “Install Extension” & you will see a huge list there,




You can find extensions available online at marketplace to install:

VS Code also supports APIs that allow you to develop extensions. You can use JavaScript or TypeScript to build your own extensions and debug and test them in VS Code

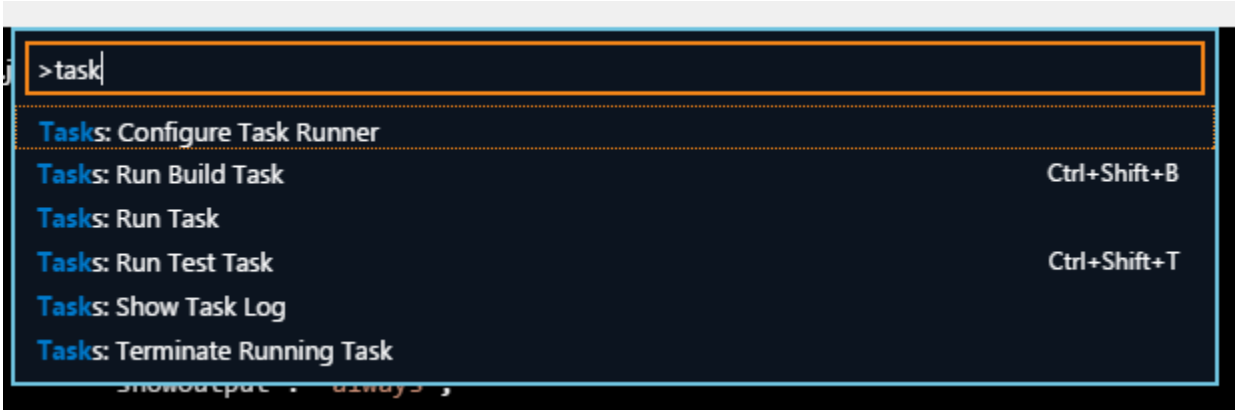
**Tasks**

I am in love with this feature of VSCode. You can write your own tasks and VSCode is intelligent to determine and show in task runner list. I have chosen

Grunt task runner. Before we move forward, please install NODE & NPM to install node packages. To install grunt use below command

npm grunt install

Open Command Palette and choose “Tasks: Configure Task Runner”



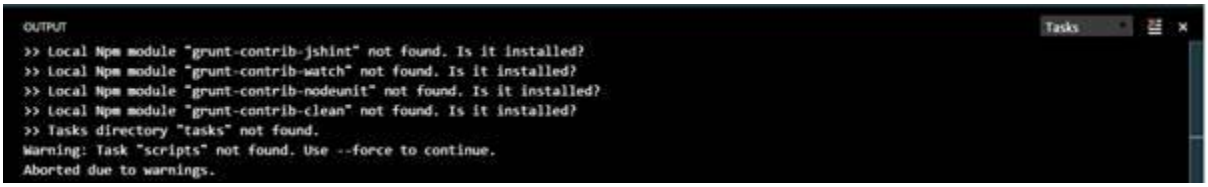

VSCode will create tasks.json file. Delete everything and add following code for Grunt task runner

```
{
    // See http://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "0.1.0",
    "command": "grunt",
    "isShellCommand": true,
    "args": ["--no-color"],
    "showOutput": "always",
    "tasks": [
        {
            "taskName": "scripts",
            "isBuildCommand": true,
            "showOutput": "always"
        },
```

```
    {
      "taskName": "sass",
      "isBuildCommand": true,
      "showOutput": "always"
    }
  ]
}
```

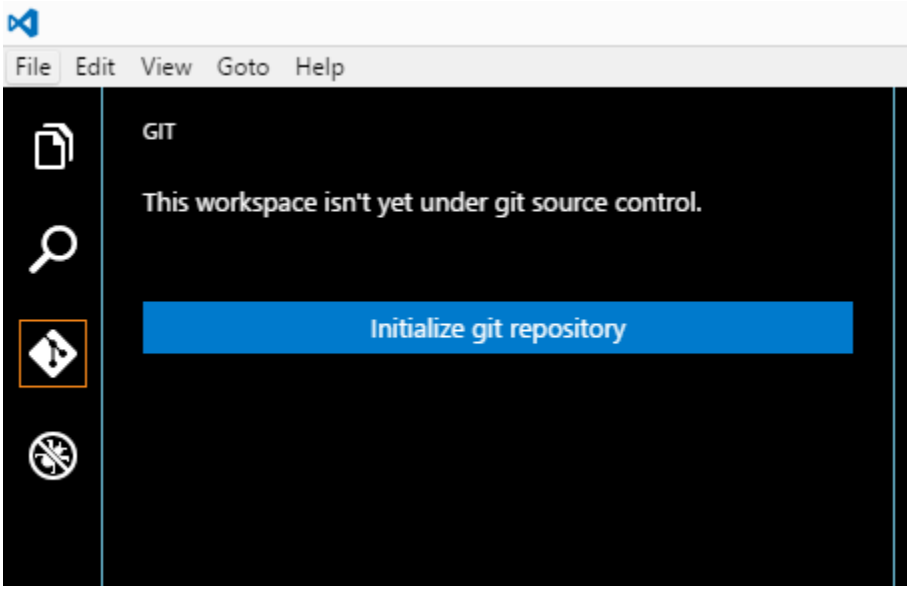
With above code we're creating 2 tasks, i.e., scripts & sass. Now press F1 and type "tasks" and select "Tasks: Run task". You'll see sass & scripts added in it,


For now, we haven't installed all modules of Grunt therefore after running scripts you'll face below error in OUTPUT window. This is beyond scope of this part but Output screen is similar to Visual Studio Editions.




## Git Integration

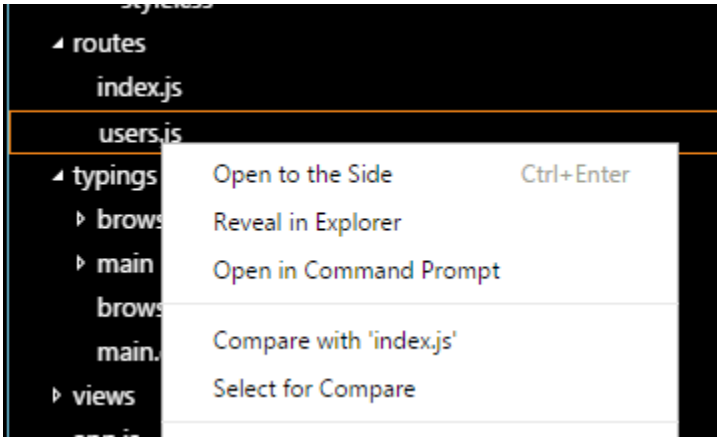
VSCode side navigation contains reference to Git repo (Ctrl+Shift+G). You can initialize Git repo and start using Git commands like pull, push, commit etc.




**Misc features**

Go to line: F1 and type ":" and you can got line number you want in file,

Compare files: Right click on a file and "select compare file" then right click and select another file to compare.




Below view will showcase match/mismatch lines. I don't use any other file comparison tool because now VSCode provides it.


## Debugging

The real power of VSCode is debugging capabilities. You will not miss the wonderful experience of Visual Studio Editions here. My next part is dedicated to Debugging capabilities of VSCode, please read once published.

## Debugging Capabilities of VSCode

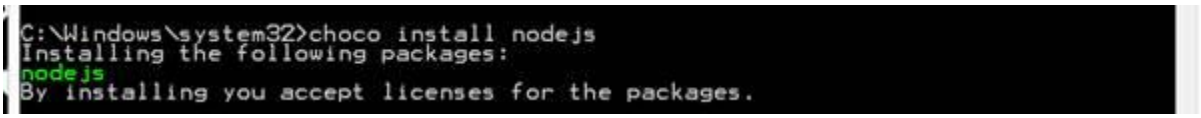
JavaScript is language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In last part we learnt about VSCode – an awesome editor. In this part we'll learn debugging capabilities of VSCode. Before we move forward, let us setup NodeJS application.

**Setup Node.js application**

I am sure you must have heard about LAMP. Similarly, there is a new stack for web application development is MEAN (MongoDB, Express.js, AngularJS, and Node.js) — used to develop web applications.

To install Node on windows you can download online or use Chocolate Windows download manager to install it




**Express.Js:**

It is a framework for building and running Node applications. Below command will install it globally
npm install -g express-generator

Now, scaffold a new application,

**express webApplication1**

```
$ express webApplication1

   create : webApplication1
   create : webApplication1/package.json
   create : webApplication1/app.js
   create : webApplication1/routes
   create : webApplication1/routes/index.js
   create : webApplication1/routes/users.js
   create : webApplication1/views
   create : webApplication1/views/index.jade
   create : webApplication1/views/layout.jade
   create : webApplication1/views/error.jade
   create : webApplication1/bin
   create : webApplication1/bin/www
   create : webApplication1/public
   create : webApplication1/public/images
   create : webApplication1/public/stylesheets
   create : webApplication1/public/stylesheets/style.css

   install dependencies:
     $ cd webApplication1 && npm install

   run the app:
     $ DEBUG=webApplication1:* npm start

   create : webApplication1/public/javascripts
```

Hence it'll create a basic setup of files to run Node.js application. Now, we can install node modules by using npm install command,

```
$ cd webApplication1/
$ npm install
npm WARN deprecated jade@1.11.0: Jade has been renamed to pug, please install
e latest version of pug instead of jade
npm WARN engine is-buffer@1.1.3: wanted: {"node":">=0.12"} (current: {"node":"
10.35","npm":"1.4.28"})
npm WARN engine is-buffer@1.1.3: wanted: {"node":">=0.12"} (current: {"node":"
10.35","npm":"1.4.28"})
debug@2.2.0 node_modules\debug
└── ms@0.7.1

cookie-parser@1.3.5 node_modules\cookie-parser
├── cookie@0.1.3
└── cookie-signature@1.0.6

serve-favicon@2.3.0 node_modules\serve-favicon
├── fresh@0.3.0
├── parseurl@1.3.1
├── ms@0.7.1
└── etag@1.7.0

morgan@1.6.1 node_modules\morgan
├── on-headers@1.0.1
├── basic-auth@1.0.3
├── depd@1.0.1
```

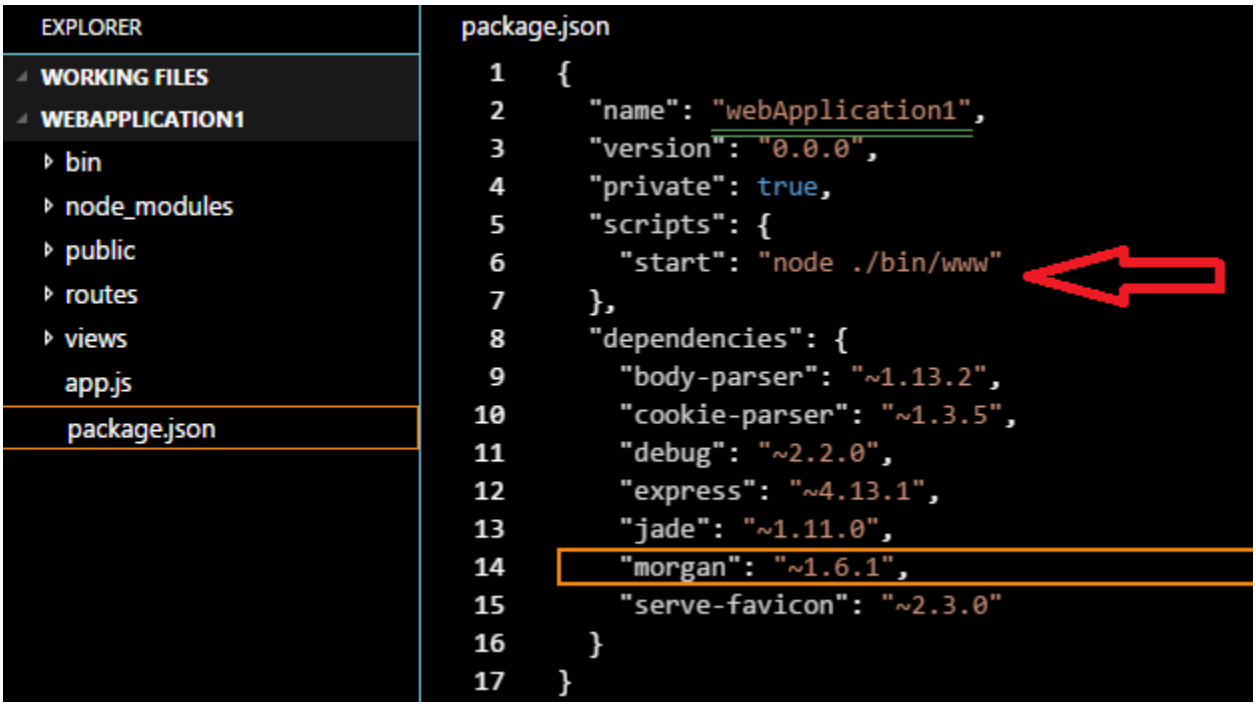
The tree structure would look like below,

```
$ ls -al
total 17
drwxr-xr-x 1 Sumit Jolly 1049089    0 Apr 29 12:58 ./
drwxr-xr-x 1 Sumit Jolly 1049089    0 Apr 29 12:56 ../
-rw-r--r-- 1 Sumit Jolly 1049089 1442 Apr 29 12:56 app.js
drwxr-xr-x 1 Sumit Jolly 1049089    0 Apr 29 12:56 bin/
drwxr-xr-x 1 Sumit Jolly 1049089    0 Apr 29 12:59 node_modules/
-rw-r--r-- 1 Sumit Jolly 1049089  334 Apr 29 12:56 package.json
drwxr-xr-x 1 Sumit Jolly 1049089    0 Apr 29 12:56 public/
drwxr-xr-x 1 Sumit Jolly 1049089    0 Apr 29 12:56 routes/
drwxr-xr-x 1 Sumit Jolly 1049089    0 Apr 29 12:56 views/
```

Package.json file contains dependencies modules and pointer where to start. We'll run the application using below command.

**npm start**

And it will open the application at 3000 port and you can browse to http://localhost:3000 to run the application,



Let us read detail in ./bin/www file. Use F1 to use open command palette to get there.

```
var app = require('../app');
var debug = require('debug')('webApplication1:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
```

It creates an http server at port 3000 (default). If you would like run your application at another port, you could change the one that you can modify in ./bin/www file

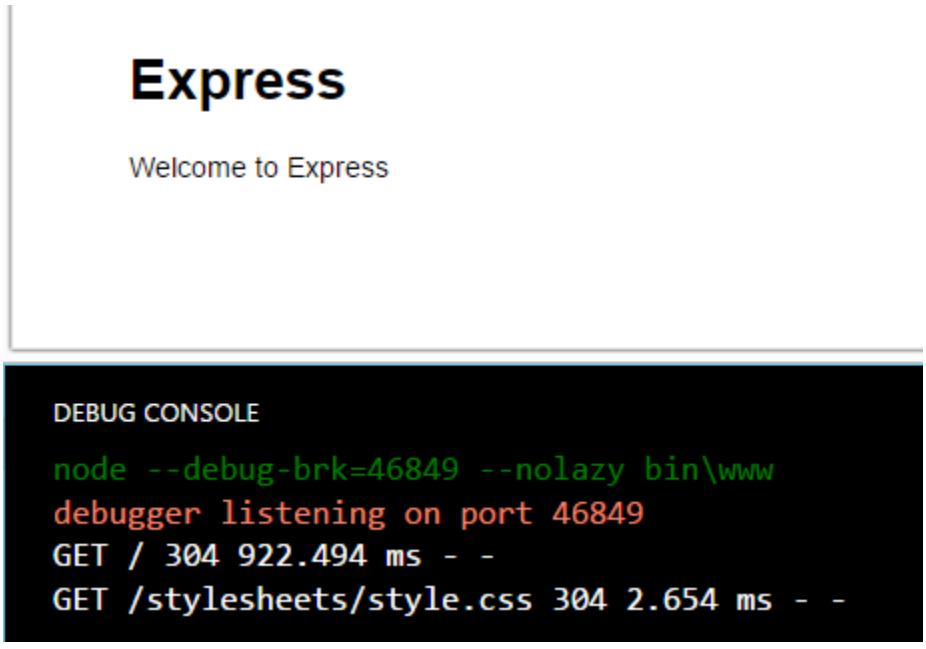
**Setup application in debug mode**

Before we enable debugging Press F5 to launch application in debug mode and choose Node.js. It will create launch.json configuration file like tasks.json earlier


**Run application**

You can run application by F5 and see below output on Debug Console,

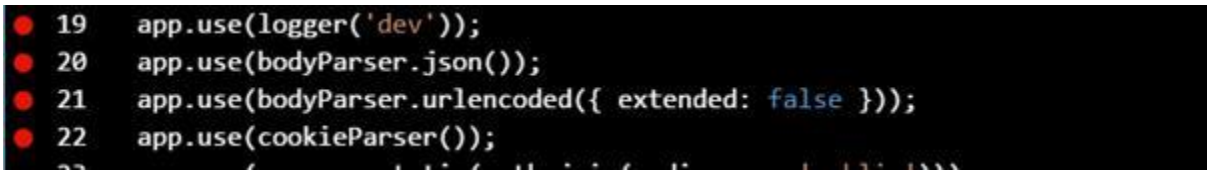

Run localhost:3000 and your node application is up and running (I have also attached Debug message),

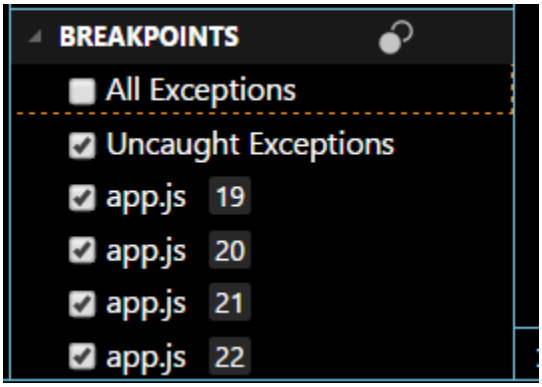



**Debugging**

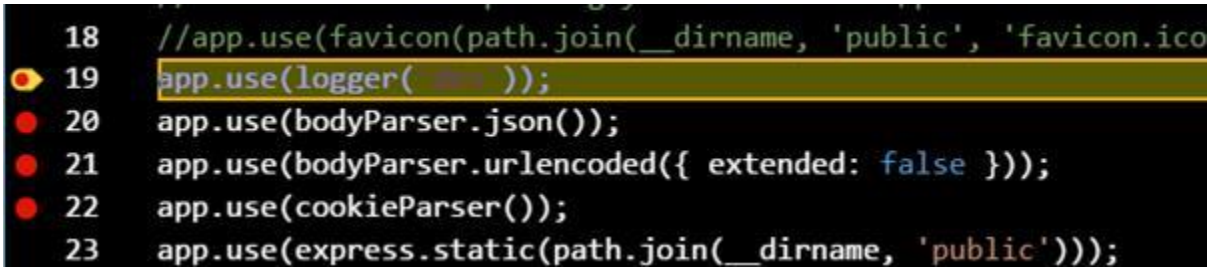
-  VSCode has built-in debugging support for Node.js (JavaScript, TypeScript, and any other language that gets transpiled to JavaScript). You can open debug pane by clicking its icon and see various windows.

- **Breakpoint**: Like Visual Studio Edition, you can press F9 and set breakpoint. All breakpoints will appear in breakpoints window. Open app.js and set breakpoints,

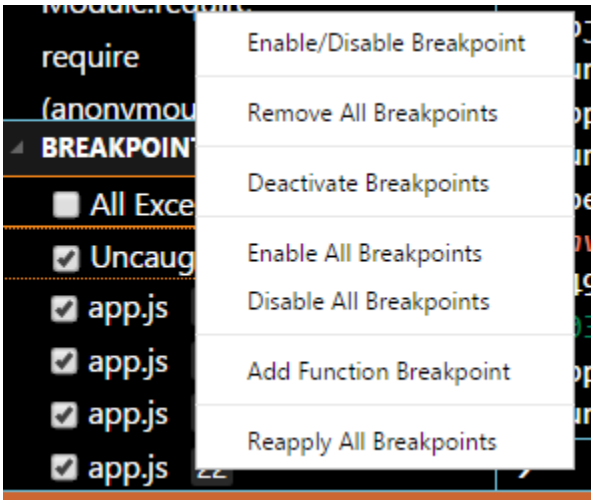



Now you can run by pressing F5 and execution will stop at the breakpoint. Press F10, F11 depending upon you want to Step Into or Step Out.



It also allows you to create function breakpoints you can also do that in Breakpoint window,




You can set breakpoints on functions, variables etc. which you create in your application. There are two other windows helpful for debugging, i.e., Watch & Call Stack.


You can also evaluate expressions in Debug Console.

## JavaScript OOP

JavaScript is language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

We use OOPS in programming languages like C++, C#, Java. It is fundamental of our programming patterns.

**Q: Is JavaScript an object oriented language?**

**Ans:** Yes it is, but many developers differ because there is no “class” keyword in it. I believe JavaScript has all the ingredients to be an OOP language. We can read in detail OOP in JavaScript

**Encapsulation**

Encapsulate means “enclose (something) in or as if in a capsule”. It refers enclosing all the functionalities within that object, so it is hidden from rest of the application. There are many ways to create Objects (refer http://www.c-sharpcorner.com/article/voice-of-a-developer-javascript-objects/ Voice of a Developer – part 2 for details)

Below is the ubiquitous object literal pattern, my favorite way to declare objects:

```
1. var myObj = {name: "Sumit", profession: "Developer", getDetails: function(
   n, p){
2.         return n+' ' +p;
3.     }
4. };
```

**Inheritance**

JavaScript supports prototypal inheritance than classical inheritance. We studied this in part2; let us dig it in detail further. There are two concepts:

**Prototype property:**

Every JavaScript function has a prototype property and you attach properties and methods on this prototype property when you want to implement inheritance. Firefox and most versions of Safari and Chrome have a __proto__ “pseudo” which we have learnt in part2. There is another property called prototype property for inheritance.

**Example:**

```
1. function baseFunction(){};
2. var func1 = new baseFunction();
3. baseFunction.myOwnProperty = 'my own emunerable property';
4. baseFunction.prototype.testInheritanceProperty = 'other instances can inhe
   rit it';
5. var func2 = new baseFunction();
6. var func3 = new baseFunction();
7. console.log(func1.myOwnProperty);
8. console.log(func2.testInheritanceProperty); // other instances can inherit it

9. console.log(func3.testInheritanceProperty); // other instances can inherit it
```

**Prototype attribute:**

JS runtime first look property on the object. If not found, then looks for the property on the object’s prototype—the object it inherited its properties from. This is called as prototype chaining. JavaScript uses this prototype chain to look for properties and methods of an object.

```
1. Object.prototype.newProperty=100;
2. var obj = {} // creates a new Object
3. obj.newProperty; // 100
```

As I mentioned above there is hidden property __proto__ as a connector.


This describes connection of prototype chaining from obj->Object->property you're trying to access.

**Q: Why we get undefined while accessing a property?**

**Ans:** Now I believe you know the answer, i.e., if a property does not exist in object or parent or onwards then JS runtime returns undefined

**Note:** Properties Inherited by all Objects

If you notice when you create a new Object it'll inherit properties & method from Object.prototype ex- toString(), toLocaleString()

**Abstraction**

It means to hide certain details and show required objects. This improves

readability and maintainability of code. For a good code quality and to reduce risk of an object getting modified outside accidentally, it is good to adopt abstraction.

**Q: Is there a support in JavaScript on Abstraction?**

**Ans:** There is no support or feature developed in JavaScript, but we can look for alternatives.

**Let us see an example of abstraction:**

```
1. var Logger = {
2. log : function(log){}
3. } //Logger object which has a constructor function
4. var ConsoleLogger = function() {}; //It is a blank function
5. ConsoleLogger.prototype = Object.create(Logger);
6. ConsoleLogger.prototype.log = function(log) {
7. console.dir (log);
8. }
```

**Output**


**JavaScript Useful Reserved Keywords**

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

JavaScript has many useful reserved keywords. I will walk you through usage and benefits of these keywords

**Delete keyword**

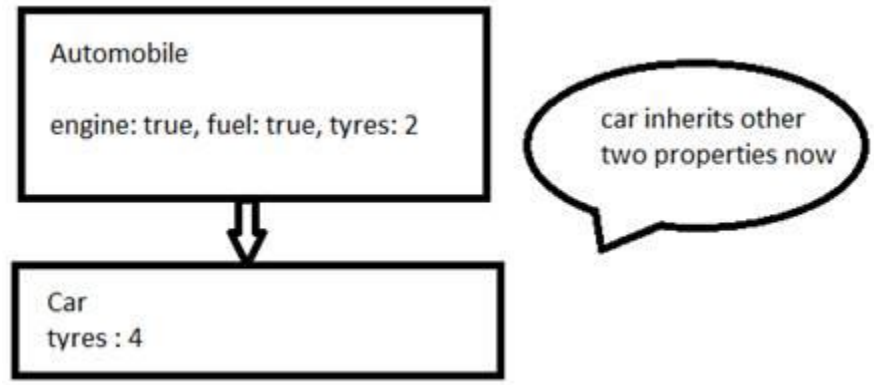
When we work on large applications then we work with many objects. We are familiar with problems like memory leak, performance of application is not good, and similar issues arise at a later stage. I always recommend that we know how much memory we are consuming in our application.

Ex- Chrome provides a quick way to look into memory usage,

```
> window.performance.memory
< MemoryInfo {totalJSHeapSize: 21700000, usedJSHeapSize: 14300000, jsHeapSizeLimit: 793000000}
```

The main advantage is to delete unwanted property from object,

```
1. var automobile = {engine: true, fuel: true, tyres:2 };
2. var car = Object.create (automobile); //car is instanceof automobile
3. console.log (car.tyres); // 2 it inherits from parent class
4. //as we know car has own tyres then we can create own property of car obj
   ect
5. car.tyres = 4; // will make own tyre property of car object
```




```
1. //now we want to delete property tyres from car
2. delete car.tyres ; //will return true
```

**Q: We have deleted tyres from car object, what will happen if we print car.tyres?**

**Ans:** The output of below statement will be 2 because it refer tyres property in automobile object,

```
console.log (car.tyres); // print 2

To delete tyres completely remove from automobile object

delete automobile.tyres; //true
console.log (automobile.tyres); // print undefined
```

**This keyword**

"this" is not a variable. It is a keyword. You cannot change the value of this. It refers to the global object in all global code.

We know to create multiple objects we can use Object Constructor function. We'll use this keyword to create property which belong to function,

```
function Automobile (engine, fuel, tyres)
{
this.engine = engine;
this.fuel = fuel;
this.tyres = tyres;
}
var car =new Automobile (true, true , 4);
var bike =new Automobile (true, true , 2);
```

Second example, this refers to the parent object in the body of the function,

```
var counter = {
  val: 0,
  increment: function () {
    this.val += 1;
  }
};
counter.increment();
```

```
console.log(counter.val); // 1
counter['increment']();
console.log(counter.val); // 2
```

**New operator**

The new operator creates a new object. Any JavaScript function can be used as a constructor function with new.

**Example**

```
function F1 (val) {
  this.val = val;
}
var f = new F1("Foo");
console.log(f.val); // Foo
console.log(this.val); // ReferenceError: val is not defined. Because val is associated with function object

If we call f without new then this will refer to the global object (which is ex Windows )
function F1 (val) {
  this.val = val;
}
var f = F1("Foo");
console.log(this.val); // Foo
```

## JavaScript Functions

JavaScript is language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In JavaScript, functions are the 1st class citizens of language. These are the bread and butter of JavaScript. Go in deep now:

### Functions in-depth

Function keyword, followed by a name, followed by parentheses (). You can get function definition or invoke the function, **ex:**

```
1. function myFunction(a, b) {
2. return a * b; // Function returns the multiplication of a and b
3. }
```

```
> myFunction
< function myFunction(a, b) {
      return a * b;
  }
```

The () invokes the function,

```
> myFunction(2,3);
< 6
```

### Properties of function

Let us dive into some properties of function, ex-


- **Call v/s Apply:**

  If you want to assign value of a function we use straight forward way, i.e.,

```
1. var val = myFunction (2,3); // val = 6
```

  There are two other way to achieve it using call or apply property,

```
1. var val = 0;
2. myFunction.apply(val,[2,3]); // 6 as output
3. myFunction.call(val,2,3); // 6 as output
```

| Call requires the parameters be listed explicitly | Apply lets you invoke the function with arguments as an array. |
|---|---|

- **The "constructor" property**

  Every object has a built-in property named constructor. It refers to the function which created it. Let us understand in detail:

```
1. function myFunction(a, b) {
2.     return a * b;           // Function returns the multiplication of a an
   d b
3. }
```

```
var m = new myFunction();
console.log( m.constructor == myFunction); // true
console.log(m.__proto__.constructor == myFunction); // true, the ot
her way to check is
```

- **Defining Your Own Function Properties**

  Many times we need a unique number say GUID or identifies which increments with each call made to function. In such scenario, it is recommended to attach a property with a function. It is acting like a static property to keep track of last value returned.

```
function myFunction(a, b) {
    ++myFunction.p1;
    return a * b;           // Function returns the multiplication of a an
d b
}
myFunction.p1 = 1;
myFunction();
myFunction.p1 ; // will print 2
```

- **Length property**

  In a function, we pass arguments to process input and sometime we don't pass correct number of arguments. Therefore, it is good practice to check required number of arguments needed for a particular function.

```
function myFunction(a, b) {
    return a * b;
}
myFunction(2); // will give NaN because b is undefined
Its time to fix above function by using property length
function myFunction(a, b) {
    var argsRequired = arguments.length;
    var argsActual = arguments.callee.length;
```

```
9.    if (argsRequired!=argsActual) {
10.      throw new Error("Wrong # of args passed");
11.   }
12.   return a * b;
13.}
```

```
> myFunction(2,3)
<· 6
> myFunction(2)
⊗ ▶ Uncaught Error: Wrong # of args passed(…)
```

## JavaScript Functions Invocations

JavaScript is a language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In the last part I covered functions in depth. In this part I will cover various ways to invoke functions.

Functions are first class citizens in JavaScript. There are many ways to declare functions in JavaScript. Let's understand each.

### Function Declaration

The traditional way of defining a function is:

```
1. function logMe() //function name logMe
2. {
3.     console.log ('Log me in');
4. }
5. logMe();
```

If you skip the function name then it'll become an anonymous function, we'll talk about them in detail later. But here is the code:

```
1. (function ()
2. {
3.     console.log ('Log me in');
4. })(); // we're calling this anonymous function;
```

### Function hoisting

It is the process in which JavaScript runtime hoists all the functions declared using function declaration syntax at the top of JavaScript file, look at the example below:

```
function logMe()
{
    console.log ('Log me in');
}
logMe();
function logMe()
{
    console.log ('Log me again');
}
logMe();
```

What do you think the output? Will it be:

Log me in
Log me again

No, it'll be:

Log me again
Log me again

This is because of function hoisting because JavaScript will replace the first logMe with the second one and place it at the top.

**FUNCTION EXPRESSION**

It's the second form of declaring a function.

```
var log = function (){
    console.log('Log me in');
}
```

Now writing logMe after the function is futile because it can now be called using the assigned variable log.

Consider the following example:

```
1. var log = function logMe(){
2.     console.log ('Log me in');
3. }
4. log();
5. var log = function logMe(){
6.     console.log ('Log me again');
7. }
8. log();
```

So what shall be the output of it? Is it:

Log me again
Log me again

No, it's:

Log me in
Log me again

The reason is because it's function expression is part of the variable and not directly a function. So JavaScript will not place it at the top.

**Anonymous function**

It's a function without a name. You can assign a function to a variable. Another way of declaring an anonymous function is within an object literal. For example:

```
1. var programmer = {
2.     lang:  'JavaScript',
3.     browser: 'Chrome',
4.     getFullSpecs: function(){
```

```
        console.log(this.lang+ ' running inside ' + this.browser);
    }
}
```

Here an anonymous function is assigned to the getFullSpecs property. Incase of anonymous function JavaScript doesn't apply function hoisting.  We will learn more about these functions in next part.

**Function() Constructor**

The function() constructor expects any number of string arguments. The last argument is the body of the function; it can contain arbitrary JavaScript statements, separated from each other by semicolons.

```
var multiply = new Function("x","y","var z=x*y; return z;");
multiple(10,20);
```

**Output:** 200

# JavaScript Anonymous Functions

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In the last part I covered functions in depth. Now, I will cover Anonymous Function in detail covering its advantages, disadvantages & differences.

**Anonymous Function**

We know what a function is and we need to understand what anonymous function is. Anonymous means,

A function without a name is an anonymous function. We store these inside a variable name. So, the invocation happens using the variable name.

**Below is an example:**

```
1. var sum = function(a,b){return a+b;};
2. sum();
```

**Advantages of anonymous function**

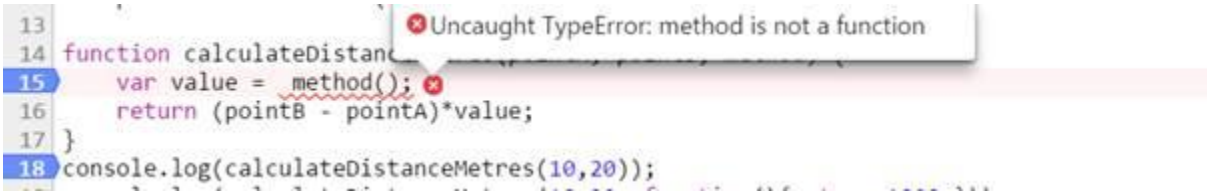
First advantage is that JavaScript allows you to pass anonymous function as an object to other functions. You don't need to explicitly define the function before passing it as a parameter to another function. Example: below is distance calculator between two points. In method*calculateDistanceMetres* we're passing an anonymous function.

```
1. function calculateDistanceMetres(pointA, pointB, method) {
2.     var value =  method();
3.     return (pointB - pointA)*value;
4. }  console.log (calculateDistanceMetres(10,20, function(){return 1000;}));
```

Now, what if we don't pass anonymous function then what will happen?

```
1. console.log (calculateDistanceMetres(10,20));
```

Then it will throw below error because method() cannot execute any code,




In order to solve above case, let us adjust our code which will cover both scenarios

```
1. function calculateDistanceMetres(pointA, pointB, method) {
2.     if (method) {
3.         var value = method();
4.         return (pointB - pointA) * value;
5.     }
6.     else{
7.         return (pointB - pointA) ;
8.     }
9. }
10.console.log(calculateDistanceMetres(10, 20)); //output 10
11.console.log(calculateDistanceMetres(10, 20, function () { return 1000; })); //
   output 10000
```

Other popular implementation of anonymous function is in popular frameworks & libraries. In Node.js http module createServer method accepts a function which is executed at each HTTP request.

```
1. var http = require("http");
2. var server = http.createServer(function(request, response) {
3. response.write("Hello World");
4. response.end();
5. });
6. server.listen(1234);
```

Second advantage is that it's hidden from the outside world and only accessible to only programmer objects.

**Example:**

```
var programmer = {
    lang:  'JavaScript',
    browser: 'Chrome',
    getFullSpecs: function(){
        console.log(this.lang+ ' running inside ' + this.browser);
    }
}  programmer.getFullSpecs(); //only programmer object could access it
```

Third advantage it is widely used where you can pass like a callback. For example:

```
$('#div1').click= function(){
    alert('clicked')
};
Or:

function CallMe(callback)
{
    console.log('callback is called after');
    callback && callback();//check whether there is data for callback or not
}
CallMe(function(){alert('call me')});
```

**Disadvantages of anonymous function**

- If you have large program then debugging is tough because you don't know name of the function in call stack. The call stack will show it as anonymous function.

  Debug above program in Chrome & see Call Stack. You can notice anonymous function when the execution is at code function () { return 1000; }


- These functions cannot be reused.
- These functions cannot be unit tested easily. Therefore, you may need to refactor the code.

**Differences**

| Anonymous function | Named function |
|---|---|
| This function gets created only after its declaration. JavaScript doesn't do hoisting like in the case of normal function. | This function is loaded before any code executes, so the JavaScript engine does what is called hoisting [we will cover this concept in later part]. |
| You will get error if you try calling it before than its declaration. | You can call the function before or after declaration anywhere in the code. |

**JavaScript Pure And Impure Function**

JavaScript is a language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In the last part I covered anonymous function. In this part I will cover pure and impure function in detail. Before I cover this, we should understand some aspects of functional programming so we know what is mutable and immutable objects.

## Functional programming

It is a different style of programming language that uses declarative approach rather than imperative. In declarative programming, you are using a more descriptive way to define what you want to do and not how you want to do some action. Example, LINQ in C#

*bottles.Where(price<20).OrderBy(brand).Take(10);*

It will find bottles where price is less than 20, order by brand and take top 10.

Another aspect of declarative programming is that it avoids changing state and mutable data. In simple terms, once a variable is bound to a value, it cannot be changed.

**Two more concepts**

**Mutable**

Mutable means it can be changed. Different programming languages have different approaches towards implementation of mutability or immutability.

In JavaScript, Objects are mutable as they are addressed by reference and not by value.

**Example**

```
1. var y={a:1, b:2};
2. var x =y;
```

The above will not duplicate y in x, but it'll point x to y. Please refer objects (part 2) for detail.

**Immutable**

Immutable means it cannot be changed. In JavaScript strings are immutable. Example,

```
> var s = String("this is string");
< undefined
> s[0]="T";
< "T"
> s
< "this is string"
```

You cannot change string value. If you want to modify a string it'll always return a new string. Remember using trim, slice, toUpper functions

```
> var newString = s.toUpperCase();
< undefined
> newString
< "THIS IS STRING"
```

**Pure function**

These are the functions which always return the same value when given the same

arguments. They take some parameters, return a value based on these, but don't change parameter values. Example product is a function which will always give same output depending upon the input.

```
1. function product(a, b) {
2. return a * b;
3. }
4. console.log(product(2, 3)); // always give 6 whenever you pass 2, 3
```

Another nice property of pure function is that it doesn't modify the states of variables out of its scope.

**Example**

```
1. var x = 10;
2. function pureFunction ( a ) {
3. return a + 2;
4. }
```

```
> pureFunction(x);
< 12
> x
< 10
```

pureFunction doesn't modify the variable outside of its scope , i.e., "x". It a nutshell pure function doesn't have side effects.

**Impure function**

Impure function may have side effects and may modify arguments which are passed to them. The return value will depend upon the arguments. There may be a case where for same arguments you'll get different values:

**Example**

```
var count = 0;

function Hits() {
count += 1;
}
Hits ();// will make count 1
Hits ();// will make count 2
Hits ();// will make count 3
```

Here it's using an external variable count and also modifying it. If a function has side effects whether it updates any file or database it'll also fall under the category of impure function.

```
function impureFunction ( a ) {
file('update.txt');
}
```

**Summary of differences**

| Pure function | Impure function |
|---|---|
| No side effects like update or db calls | May have side effects |
| Don't modify arguments which are passed to them | May modify arguments passed to them, |
| Always return the same value | Even if you call with same arguments, you may get different values. |

# JavaScript Closures

JavaScript is a language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In an aspiration to write better code, JavaScript provides closure. We'll understand it now, but before that I'll shift gears to a few other concepts like scope & callback

**Scoping**

In programming, scoping is the part of a computer program where you bind a variable and its visibility. There are two types of scoping, i.e. lexical and dynamic. JavaScript is based on lexical or static scoping. The functions are lexically rather than dynamically scoped. JavaScript blocks scopes using {}, and a new scope is created when you create a new function.

A quick glance at another concept called callback:

**Callback**

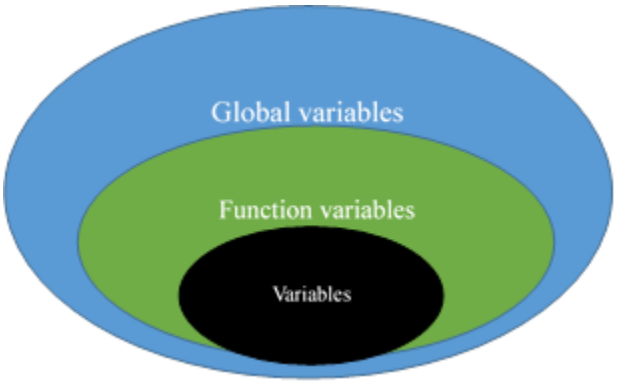
A callback is basically a function that accepts another function as a parameter, I hope it recalls your memory of higher order functions. At some point, the higher order function can call the function passed as parameter, this is a callback.

The common use of closure we are familiar with is setTimeout function.

```
function fnc1() { alert("Hello World"); }
window.setTimeout(fnc1, 1000);
```


**Closures**

A closure is an inner function that has access to the outer function's variables. It can access complete chain of variables. Illustration:

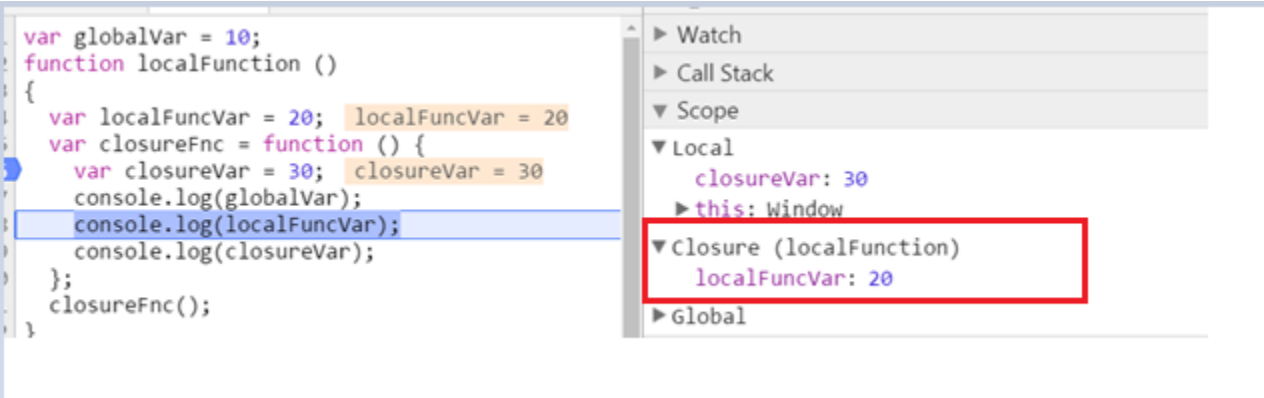



Closure function can access own variables, function variables in which it's created and global variables. See from code perspective now,

```
1.  var globalVar = 10;
2.
3.  function localFunction()
4.  {
5.      var localFuncVar = 20;
6.      var closureFnc = function()
7.      {
8.          var closureVar = 30;
9.          console.log(globalVar);
10.         console.log(localFuncVar);
11.         console.log(closureVar);
12.     };
13.     closureFnc();
14. }
```

```
> localFunction();
  10
  20
  30
```

I added debugger and checked Stack. We can notice Closure (localFunction) created.



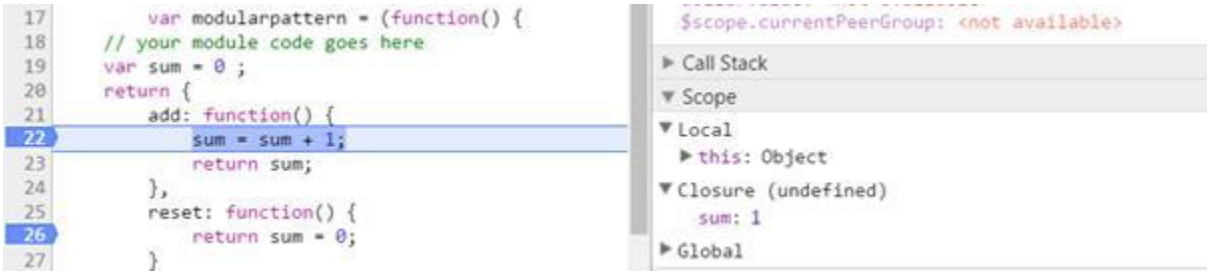

**Use of closure**

A very popular use case is Module pattern, so it can implement OOPS public, private methods concept. Closure help to emulate this, example:

```
1.  var modularpattern = (function()
2.  {
3.     // your module code goes here
4.     var sum = 0;
5.     return
6.     {
7.        add: function()
8.        {
9.           sum = sum + 1;
10.          return sum;
11.       },
12.       reset: function()
13.       {
14.          return sum = 0;
15.       }
16.    }
17.}());
```

```
18.console.log(modularpattern.add()); // 1
19.console.log(modularpattern.add()); // 2
20.console.log(modularpattern.reset()); // 0
```

Stack: **If you observe value of sum will increment or reset.**




**The objective is to hide variable accessibility from the outside world.**

Disadvantage of closure

**Memory leaks:** Closure occupies memory and GC (Garbage collector) cannot collect memory within active closures. Hence, the chance of memory leaks is high if closure is not used well.

Debug: I found it comparatively difficult to debug closure if there is big nesting, i.e., function within function and so on....

## JavaScript Currying

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

Every programmer wants to write good & reusable code. Curry can help to achieve that.
Context
We've explored functional programming aspects of JavaScript like closure, similarly another useful concept is currying. Here's some important information before I move ahead with currying.

Arity
Arity refers to the number of arguments a function can accept. And, you can have functions to take n number of arguments, which is called as variadic functions. And you can leverage arguments to slice into unary, binary arguments depending upon your requirement.

**Example,**

```
1. function showArgs(a, b, c)
2. {
3.     var args = [].slice(arguments); // [] represent Array literal
4.     // you can also invoke as [].slice.call(arguments), both are same represen
   tation
5.     console.log(arguments.length);
6. }
```

```
> showArgs(1,2,3)
  3
```

**Currying**

Let's cook tasty functions!

We know that extra / excess params passed to a function is ignored and not processed. Hence, it goes wasted, also if you've too many params passed to function it. Look at this binary function.

```
1. function sum(x, y) {
2. return x + y;
3. }
```

It could be written as,

```
1. var add = sum(10);
2. add (20); // 30
3. add (55); // 65
```

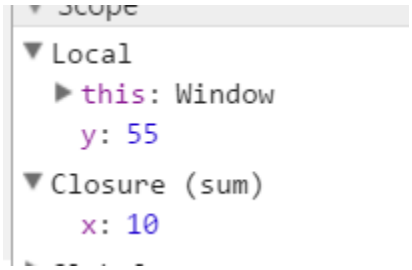
Below is the debug mode of above code where values of x, y are mentioned,


**Steps:**

- sum function returns a function – this is closure implementation.
- add – stores declaration as,

```
function (y) {
return x + y;
}
```

- Now, whenever we invoke add (55) it calls above code and here is the value in watch.




Now if your arguments grow then you could increase and it could turn like,

```
function sumFour(w)
{
   return function(x)
   {
      return function(y)
      {
         return function(z)
         {
            return w + x + y + z;
         }
      }
   }
}
sumFour(1)(2)(3)(4);
```

**Advantages of currying**

- If you're not aware of the parameters in advance, then you can store the partial result.
- It doesn't allow more than specified number of arguments, ex-

```
> sumFour(1)(2)(3)(4)(5)
✖ ▶ Uncaught TypeError: sumFour(...)(...)(...)(...) is not a function(…)
```

In above function if we pass (5) as extra argument then the function will throw an error.

**Disadvantages of currying**

- It is not a good idea to use currying when the function has a lot of arguments.
- Some people feel it's just another representation of way we write functions.
- It is associated with partial application. We'll cover partial application in some part as it's beyond the scope of this eBook.

## JavaScript Chaining

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

**Chaining**

This technique got / retrieved popularity from jQuery. We write series of statements one after the other like a chain,

**ex,**

```
1. $("#h1").text("Change text").css("color", "red");
2. Another example of chaining while working with strings:
3. var s="hello";
4. s.substring(0,4).replace('h','H'); //Hell
```

The Chaining Method is also known as Cascading, because it repeatedly calls one method on an object forming a chain/continuous line of code.

**Chaining methods**

**Ques: What is returned if there is no return statement in method?**

**Ans:** undefined

**Ques: How is chaining implemented?**

**Ans:** When a method returns this; the entire object is returned & it is passed to next method and it is called chaining. Example,

```
1. var games = function()
2. {
3.     this.name = '';
4. }
5. games.prototype.getName = function()
6. {
7.     this.name = 'Age of Empire';
8.     return this;
9. }
10.games.prototype.setName = function(n)
11.    {
12.        this.name = n;
13.    }
14.    //now execute it
15.var g = new games();
16.g.getName().setName('Angry Birds');
17.console.log(g.name);
```

We've created getName method which returns this and implemented chaining.

**Advantages**

- Code is more maintainable, simple, lean
- It is easy to read chaining code

**Simplify code after chaining**

Use case: Let us examine traditional way to program things. We have a problem to sort string below:

**Approach 1**

```
1. var vehicles = "Bike|Car|Jeep|Bicycle";
2. vehicles = vehicles.split( "|" );
```

```
vehicles = vehicles.sort();
vehicles = vehicles.join( "," );
console.log(vehicles);
```

**Output:** Bicycle, Bike, Car, Jeep

**Approach 2:**

```
var vehicles = "Bike|Car|Jeep|Bicycle";
vehicles = vehicles.split('|').sort().join(',');
console.log(vehicles);
```

Output: **Bicycle, Bike, Car, Jeep**

JavaScript Array Methods

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

New Array methods were introduced, and widely used now. I find these extremely useful and many modern JS libraries are heavily dependent on these, e.g., UnderscoreJS

We'll look at common used Array methods

**Map**

The map() method creates a new array with the results of calling a function for every array element.

**Parameters**

**Callback -** produces an element of the new Array & runs over each element

- **currentValue -** current element being processed

- **index -** index of current Array element
- **array -** array map was called upon

```
var numbers = [1,2,3,4];
var ary1=numbers.map (function (item,index) {
return item *2;
});
console.log(ary1); // [2, 4, 6, 8]
```

Therefore, the above code leverages Map method & iterates over each item and produces a  new Array containing product by 2. Index is optional to pass & you could use inside callback function (item,index)

**Reduce**

The reduce() method reduces the array to a single value.

**Parameters**

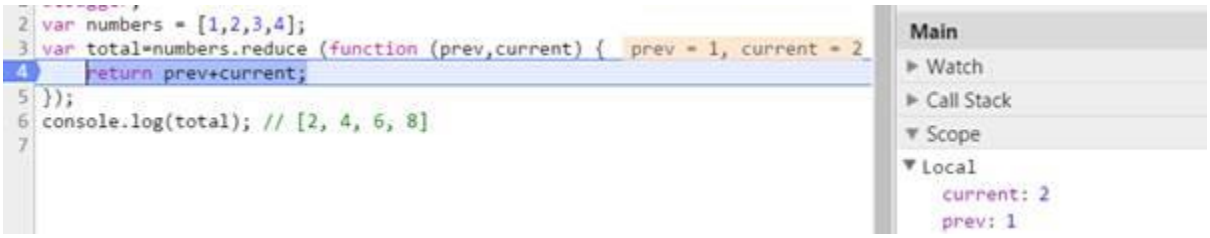
A callback function runs on each value and takes four arguments:

- **previousValue -** value returned previously
- **currentValue -** current element being processed
- **index -** index of current Array element
- **array -** array map was called upon

**Example**

```
var numbers = [1,2,3,4];
var total=numbers.reduce (function (prev,current) {
return prev+current;
});
console.log(total); // 10
```

Look at stack trace where prev & current values could be debugged. This method processes two values at a time.




**Filter**

It creates a new array with all elements which pass the test.

**Parameters**

**Callback -** to test each element of array. It return true to keep the element, false otherwise.

```
1. var numbers = [1,2,3,4];
2. var ary1 = numbers.filter (function (value) {
3. return value > 2;
4. });
5. console.log (ary1); // [3,4]
```

**Find**

It returns a value in array if it's found, otherwise undefined is returned.

**Parameters**

**Callback -** to test each element of array.

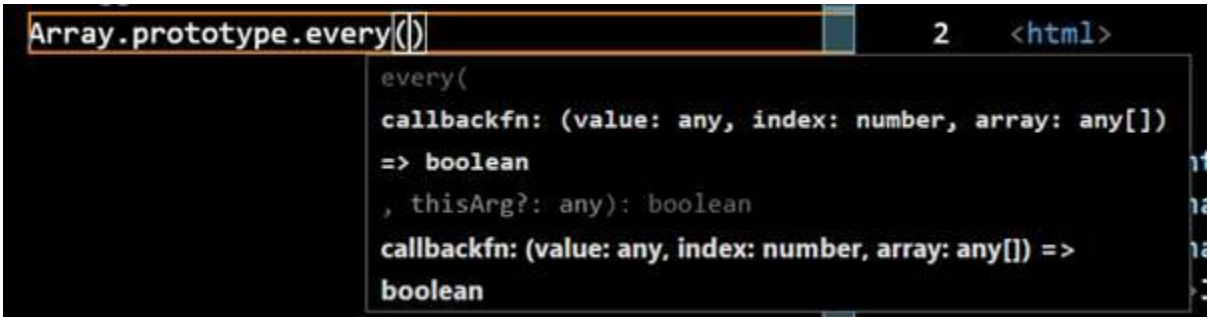
- **element -** current processed value.
- **index -** index of current Array element.
- **array -** array map was called upon.

```
1. var numbers = [1,2,3,4];
2. var value = numbers.find (function (element) {
```

```
3. return element == 2;
4. });
5. console.log(value); // 2
```

**Every**

The purpose of this method is to iterate over each element of an Array and validate for all elements. Here is the syntax from VSCode.




**Example**

```
1. function isPrime(number)
2. {
3.     var start = 2;
4.     while (start <= Math.sqrt(number))
5.     {
6.         if (number % start++ < 1) return false;
7.     }
8.     return number > 1;
9. }
10.var numbers = [3, 5, 7];
11.var flag = numbers.every(isPrime);
12.console.log(flag); // returns true
```

# JavaScript ECMAScript 6

JavaScript is a language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

### ECMAScript History

ECMAScript is the basis of JavaScript. ECMA International standardizes it. The other implementations for it is Jscript, ActionScript besides JavaScript. The original name was Mocha, later LiveScript and finally JavaScript. JavaScript was released in late 90s as part of Netscape Navigator. Brendan Eich at Netscape invented JavaScript.

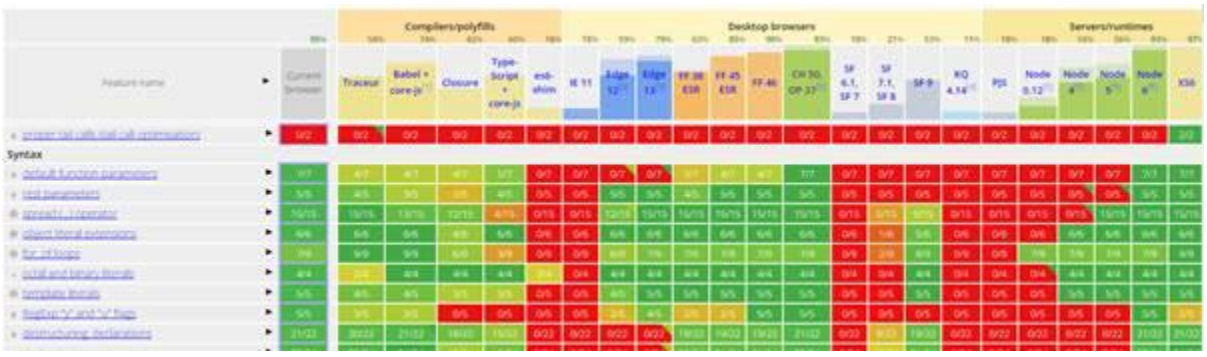
### Current Edition

The 5th edition of ECMAScript was standardized in 2009 and it is implemented in all modern browsers. ECMAScript 6th edition was standardized in 2015. This was partially implemented in most modern browsers.

### ECMAScript 6 (ES6) compatibility

You can check features of ECMAScript 5/6/7 (ES.next or ES2016) compatibility tables with modern browsers.

**URL:** http://kangax.github.io/compat-table/es6/



If you observe, IE6 to IE9 are not mentioned in above matrix.

**Q: Does it mean that we cannot use ES6 for old browsers?**

Ans: Not really, there are definitely ways to tackle it.
**Implement ES6 older browsers**

In older implementation of browsers (ex- IE8, FireFox 3.5 and before), JavaScript was purely an interpreted language. This means it executes without compilation. In modern browsers, JavaScript could be interpreted or compiled using a JIT (just-in-time) compiler. This makes it faster and performance improvement. Having said that a new concept got introduced which is transpiler.

**Transpiler**
It takes source code written in one language and converts into another language that has a similar level of abstraction. These are transplied languages e.g., TypeScript, CoffeeScript, you could use to take advantage of features not provided out-of-box by JavaScript. As output, code is converted into JavaScript.

Similarly, we can convert ES5 to ES5 to support in old browsers. There are two popular transpilers are:

**Traceur:** It supports ES6 as well as ES.next features. For more information checkout.

**Babel:** It was known as 6to5 previously. Now it is renamed to Babel.

**Q: What is the quickest way to test JavaScript?**

**Ans:** Console in browser

**Browser Console**

In browser, it is embedded and works on the concept of REPL read–eval–print loop. REPL is an interactive language shell to test language quickly. Press F12 to

invoke console interactive shell. Advantage is you do not have to open editor and type code. You can do in browser directly.

FYI: there are many REPL available for different languages, you can try online.

**Explore ES6**

As developers, we always want to get our hands dirty and try things. There are many options to explore/test ES6.

**Developer tools in Browsers**

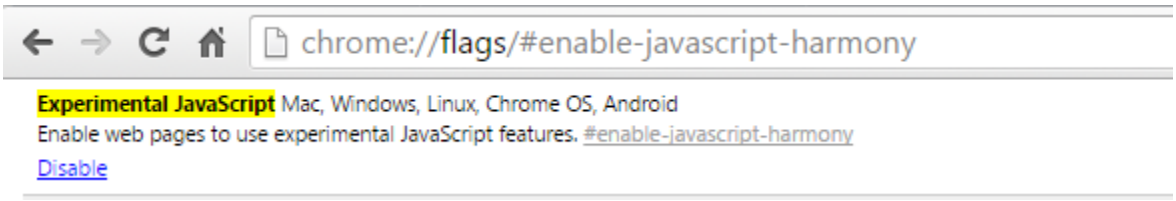
- **Edge**

  It provides good support to ES6 script which you can validate at kangax. If you are using IE11 then ensure that you have Edge emulation selected in Developer tools F12


  Currently Microsoft Edge browser scores around 80% on ES6 compatibility table. Edge is based on Chakra engine and supports many features like classes etc. If you are interested in more details to understand roadmap of Microsoft, please visit.

- **Chrome**

  Many ES6 features are hidden behind a flag called "Experimental JavaScript features". In browser enter URL: chrome://flags/#enable-javascript-harmony, enable this flag, restart Chrome.



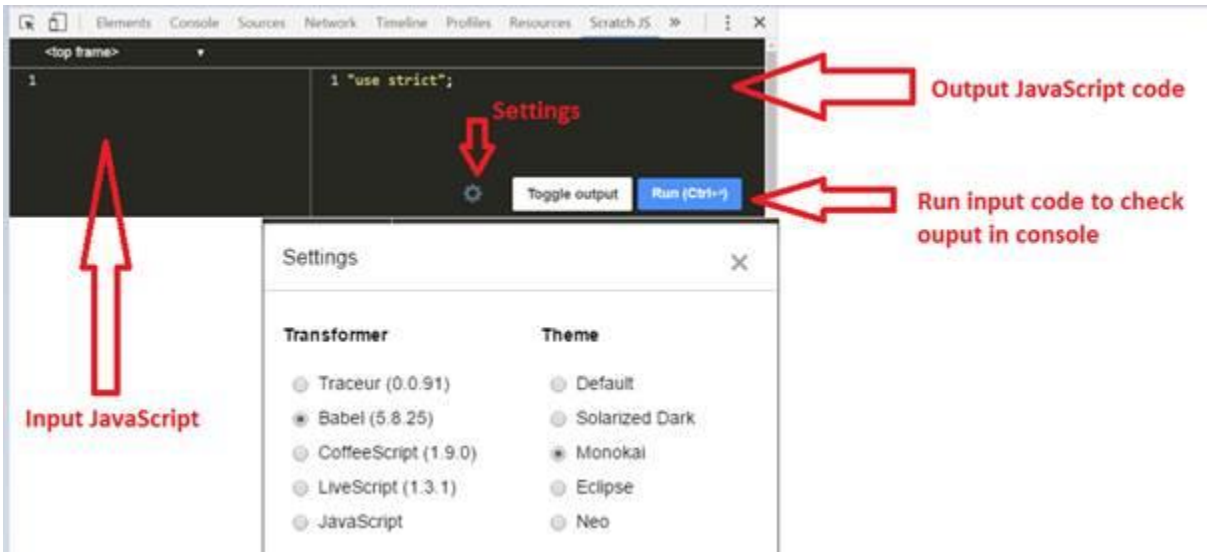

Chrome V8 release 4.9 JavaScript engine support 91% ES6
You can try Firefox, Safari, Opera etc. also

- **REPL**

**ScratchJS**: Install chrome extensions where you could try different transpilers to test out the new ES6 features.

URL

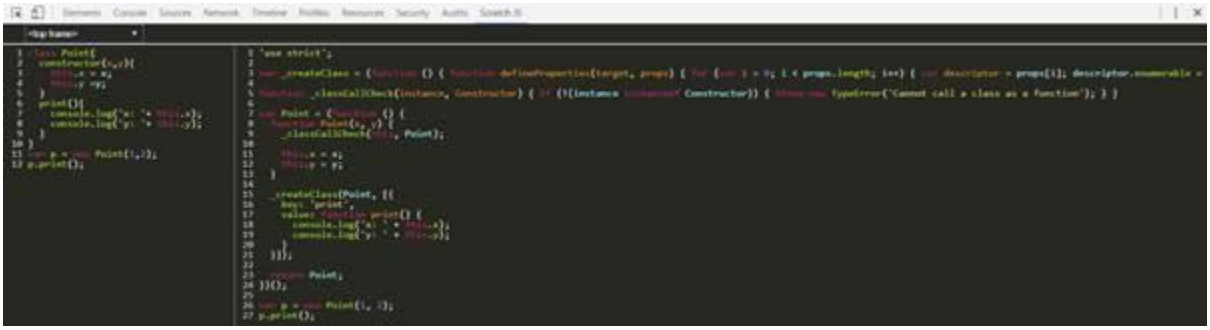

Open DevTools and “Scratch JS” is added into it.

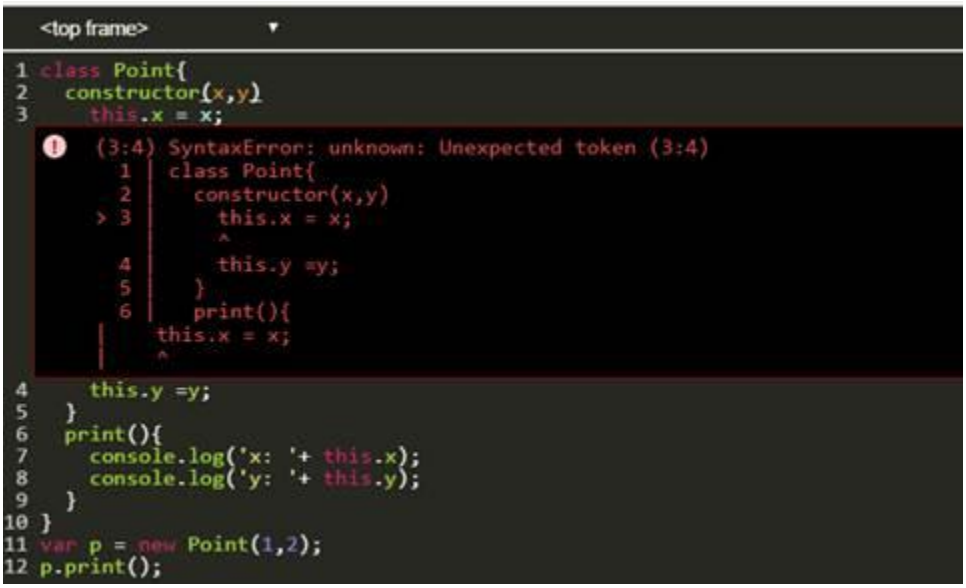



We have a transformer Babel selected where we can transform ES6 to ES5. Here is a sample ES6 code which you could paste in left pane,

```
class Point{
constructor(x,y){
this.x = x;
this.y =y;
}
print(){
console.log('x: '+ this.x);
console.log('y: '+ this.y);
}
}
var p = new Point(1,2);
p.print();
```

After you enter above sample code in left-pane it'll generate output in right hand pane in parallel. Also provides intellisense when you're writing your code. Refer images below:






## Fiddle

Fiddle is an online solution where you can type your code and convert ES6 to ES5. I find it useful in restricted environments where you cannot add extensions or update browser

**Try various fiddles online**

- http://www.es6fiddle.com/
- http://www.typescriptlang.org/play/

## Adoption

I think you feel as pumped up to use ES6 after reading this part as I feel while writing it!

Hold your breath for a sec and consider where you want to leverage ES6. There could be multiple use cases:

- Do you want to use in existing or new (greenfield) project?
- Do you want to use at client or server side like NodeJS?
- Do you want to use in mobile framework like Cordova?

I always believe for projects half a job is done if we use right tools. So do requirement analysis and match right tools and then use in your project.

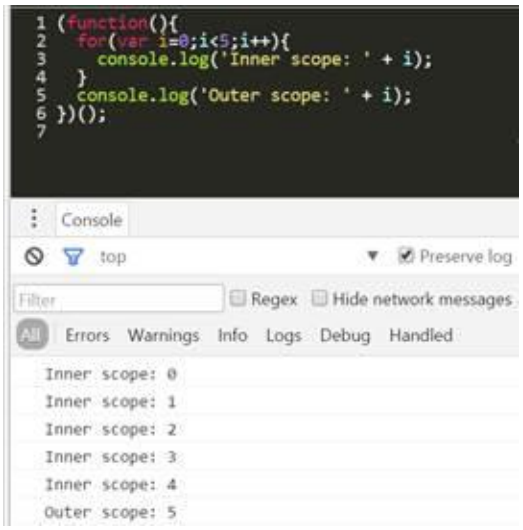
## JavaScript ECMAScript 6 Features v1

JavaScript is a language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In the last part we learned about ECMAScript 6 and now we will understand some features which developers can use to write good code.

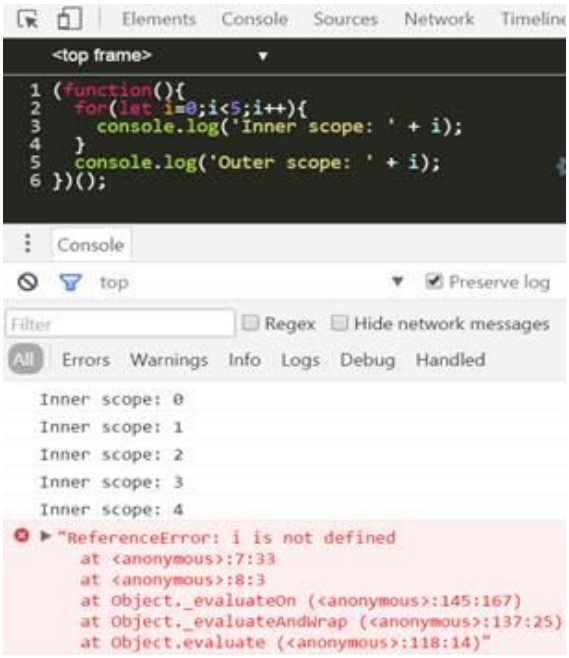
1. **Let keyword**

   We know that var is used to declare variables in JavaScript. JavaScript supports hosting which creates a function level scope than block level scope like in most programming languages. Example- self invoking function print value of “i” outside “for” loop also.



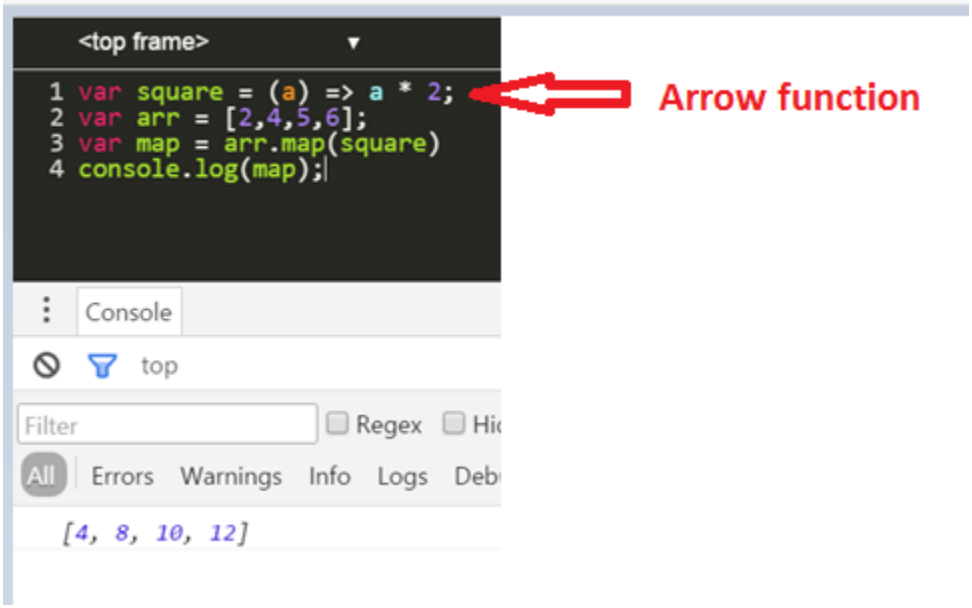

Therefore, scope of 'i' is not limited to {} within 'for' but it is bind to the function.

LET keyword allows developers to declare local variables within the scope of a block. We'll tweak the above example and notice the difference.



2. **Arrow functions**

I think arrow functions are syntactic sugar for developers. These are less verbose, and has a shorter syntax as compared to function expressions. Example




Square is an arrow function, accepts a single argument and return value implicitly. To me it also resembles functional programming aspect where style of programming is imperative than declarative.

If I transform the above code in ES5, it will transform into below code but with same output.

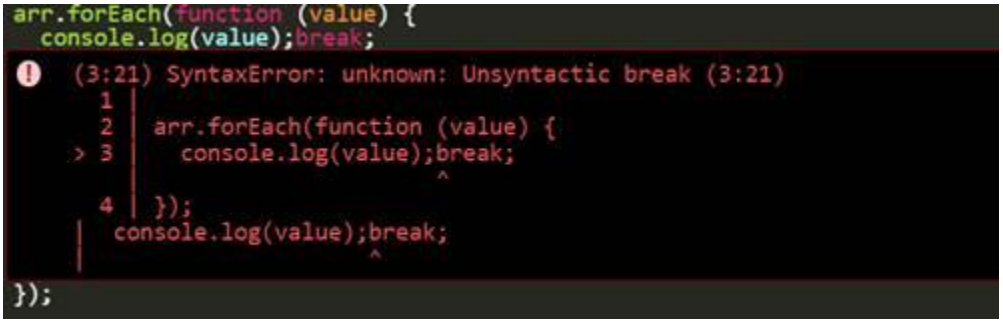
```
var square = function square(a) {
  return a * 2;
};
var arr = [2, 4, 5, 6];
var map = arr.map(square);
console.log(map);
```

3. **For..of statement**

It iterates over arrays or other iterable objects. The code gets executed for each element inside the object. Look at comparison of code below:

| ES5 | ES6 |
|---|---|
| let arr = [1, 2, 3, 4, 5];<br>let sum = 0;<br>for (var i= 0;i< arr.length; i++) {<br>sum += arr[i];<br>}<br>console.log(sum); //sum will be 15 | let arr = [1, 2, 3, 4, 5];<br>let sum = 0;<br>for (let i of arr){<br>sum += i;<br>}<br>console.log(sum); //sum will be 15 |

4. We could notice difference in 'for..of' implementation in contrast with the loop we're using for past many years. In ES5 we have another loop available which is forEach. The disadvantage of forEach is you cannot use break statement.




5. **Templates**

It is an alternative for string concatenation. It creates templates to embed values. In C#, C programming language there is such functionality available:

**Example in C#:**

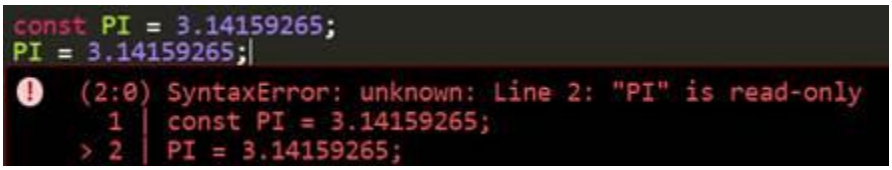
*Console.WriteLine ("{0}, {1}", "Hello", "World"); //Hello World*
*Console.WriteLine (string.Format("{0}, {1}", "Hello", "World"));*

Similarly, in ES6 we have templates available for substitution of value

*let person = { title: 'Ms', fname: 'John', lname: 'Doe'};*
*let template = `${person.title} ${person.fname} ${person.lname}!`;*
*console.log(template); // Ms John Doe!*

6. **Const Keyword**

We can create immutable variables, by using const keyword. This is common in other programming languages but ES6 adopted now. Variable created by const is readonly so after declaration and assignment it is read-only.



# JavaScript ECMAScript 6 Features v2

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In last part we learnt about ECMAScript 6 features and now we will learn some more cool features.

6. **Default values**

   We can add default values to function expressions. In ES5, default value is undefined if value is not passed via argument. Example ES5.

```
function coordinates(x, y)
{
console.log(x);
console.log(y);
}
coordinates(10); // 10 undefined
```

   In ES6 we will change it to,

```
function coordinates(x=10, y=20)
{
console.log(x);
console.log(y);
}
coordinates(10); //10 20
```

7. **Modules**

This is the feature I like the most! We need to define each module in a file then export keyword to export. On receiving side we use import keyword to import. This modular approach is really good solution for code reusability and abstraction.

Modules are designed around the export & import keywords.


8. **Classes**

This is a favorite keyword of OOPS developers but JavaScript is prototypal based and function is a first class citizen in JavaScript. Introducing classes in ES6 is debatable and many people have notion that it is not in favor of JavaScript prototypal nature.

Leaving aside, we can understand what lies in this feature.

Like OOPS programming languages, classes in ES6 are revolving around class & constructor. Example,

```
class Automobile
{
    constructor(type)
    {
        this.type = 'bike';
        this.engine = '150 CC';
    }
    getEngine()
    {
        return this.engine;
```

```
    }
}
var vehicle = new Automobile('Honda');
console.log(vehicle.getEngine()); // 150 CC
```

**Inheritance**

Use extends keyword for inheritance to inherit from base class. Example

```
class LogView extends View
{
   render()
   {
      var compiled = super.render();
      console.log(compiled);
   }
}
class LogView extends View {}
```

9. **Multi-line Strings**

It's a sugar syntax to cut long,

```
var s = 'hello '
+ 'world';
console.log(s);
```

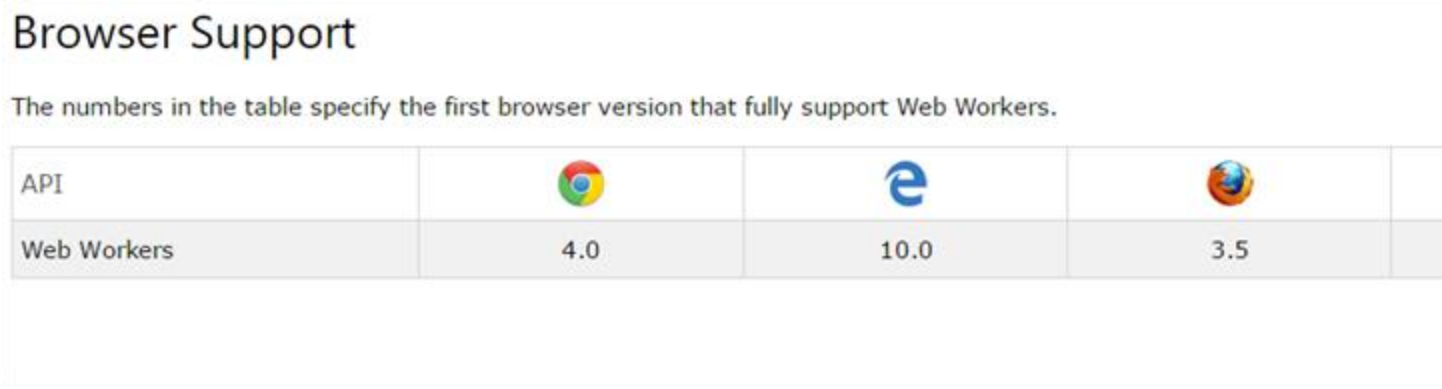
## JavaScript Web Workers

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In the last part we learned about ECMAScript 6 and now we will understand some features which developers can use to process complex scripts in background via Web Workers.

### Web Worker

A web worker is a JavaScript running in the background, without affecting the performance of the page. They are generally external script files that run in the background. It is supported by majority of browsers:




### Advantages

We know web browsers increased a lot over the past few years and it is primarily because of lot of work done on its engines, ex- V8 (Google), Chakra (Microsoft). The JavaScript so far runs in a single thread. The problem with single threaded architecture is that it blocks the code and UI becomes unresponsive incase of running complex script. There are various ways to solve this problem:

- Offload work to server, but to make apps faster fat client is preferred

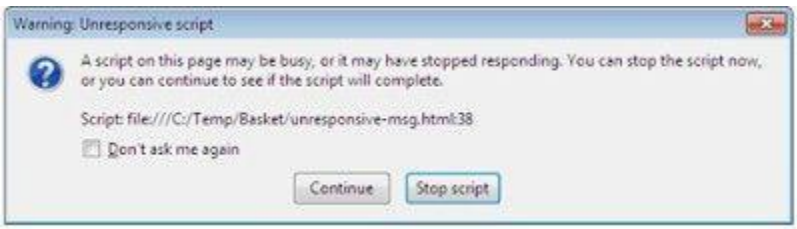
- Use asynchronous calls, but many complex ecosystem of async calls & promises could lead into callback hell
- Leverage multi threading. Interesting!

Web Workers solve this issue by providing capability of multi threading in JavaScript.

Let's define a problem first and solve for that.

**Problem statement**

If you run a heavy JavaScript script then the UI becomes unresponsive and throws the following error. This is a very common problem “Unresponsive script” that we all have seen some day or the other.




I've written sample code with a problem and the solution also.

**Index.html**

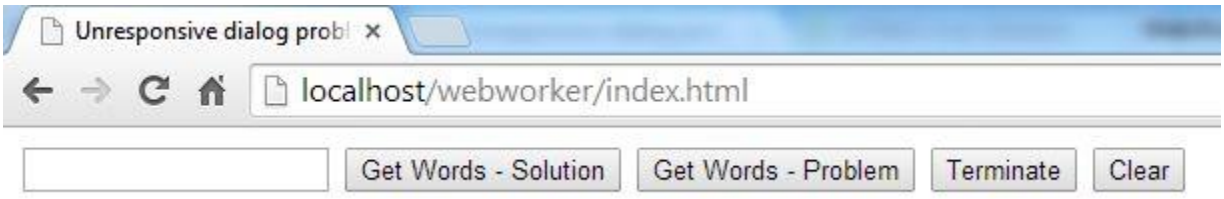
```
<!DOCTYPE html>
<html>
<head>
    <title>Unresponsive dialog problem</title>
</head>
<body>
<input id="num">
<button id="btnWordsSol">Get Words - Solution</button>
<button id="btnWordsPrb">Get Words - Problem</button>
```

```
<button id="btnTerminate">Terminate</button>
<button id="btnClear">Clear</button><br><br>
<div id="msg"></div>
<script type="text/javascript" src="js/vendor/jquery-
1.10.2.min.js"></script>
<script type="text/javascript">
   var worker=new Worker('js/solutions.js');
   $('#btnClear').click(function() {
      $('#msg').text('');
   });

   $('#btnTerminate').click(function() {
      $('#msg').text('');
      if(worker)
      {
         worker.terminate();
         worker=null;
         $('#msg').text('worker destroyed');
         $('#prg').val(0)       ;
      }
   });
   $('#btnWordsSol').click(function() {
      $('#msg').text('');

      var num=$('#num').val();
      if(worker == null)
         worker=new Worker('solution.js');
      worker.postMessage(num);
      worker.onmessage=function(event) {
         if(event.data.percent!=null)
            $('#prg').val(event.data.percent)
         $('#msg').text(event.data.showText);
      }

      worker.onerror=function(event) {
```

```
44.          $('#msg').text(event.lineno+ ' : '+ event.message);
45.      }
46.  });
47.
48.  $('#btnWordsPrb').click(function() {
49.      $('#msg').text('');
50.
51.      var text1="Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
     do eiusmod tempor incididunt ut labore t dolore magna aliqua. Ut enim ad
     minim veniam,quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
     commodoconsequat. Duis aute irure dolor in reprehenderit in voluptate vel
     it esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupida
     tat nonproident, sunt in culpa qui officia deserunt mollit anim id est laboru
     m.";
52.      var arrOfWords=text1.split (" ");
53.      // console.log('length of our para: '+arrOfWords.length)    ;
54.      var tempStr="";
55.      var index;
56.      var num=$('#num').val();
57.      for (var i = 1; i <= num; i++) {
58.          index= Math.floor(Math.random()*arrOfWords.length);
59.          tempStr+=" "+arrOfWords[index];
60.      }
61.      $('#msg').text(tempStr);
62.
63.  });
64.</script>
65.
66.</body>
67.</html>
```

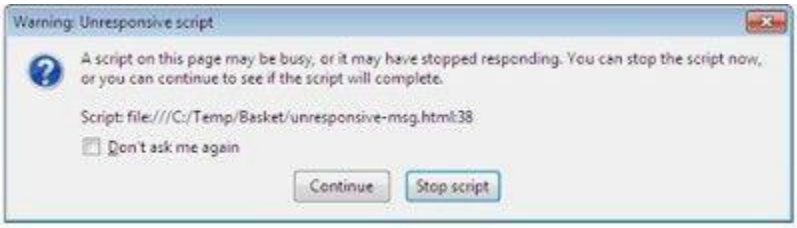
Download this code and run this in the webserver, for example http://localhost/webworker/index.html

It looks like this:




Enter any number, say 100 in the text box and click “Get Words – Problem” and see the output of random words, for example:


This button will work if you want to generate the number of words in the text box, say 1000000 (it may be less at your machine). But if you enter a higher value, say 10000000, then after clicking “Get Words – Problem” it'll throw the following message after some time:

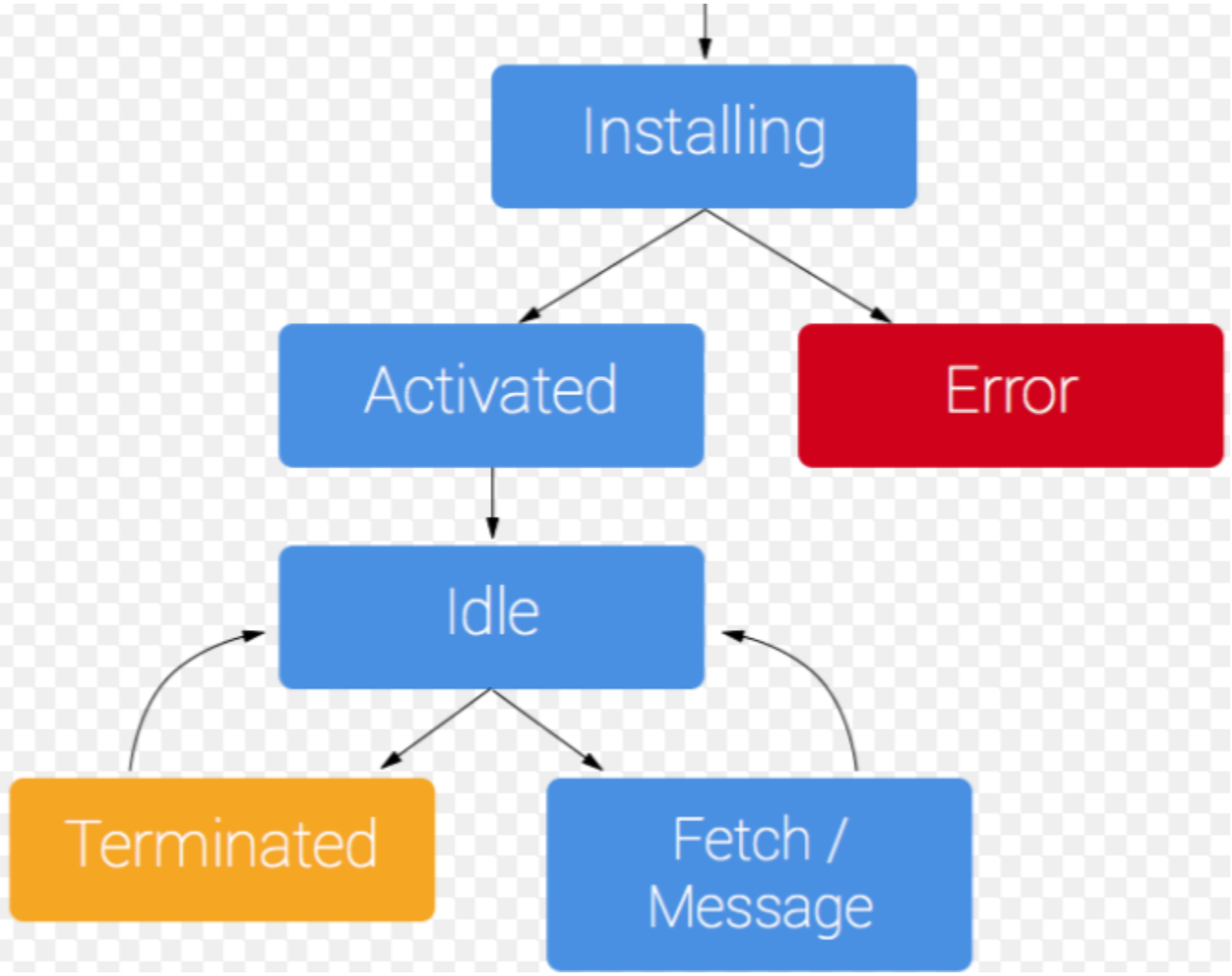



**Solution**

To solve these problems, let's leverage the WW provided by HTML5. Enter the

same number in the text box 10000000 and click “Get Words – Solution” and it'll work. We'll review Web Worker life cycle, i.e.,

- spawning a worker (installing, activate)
- idle
- post message
- terminate
- error




**Spawning a worker**

To create a Web Worker is simple and you need a new JavaScript file which contains code that you want WebWorker to execute. Hence, below code will create a new object from Worker.

```
1. $('#btnWordsSol').click(function()
2.
3. var worker=new Worker('solution.js');
```

**Communicating with a Web Worker**

To use these Web Workers in real world you need to establish a communication. These messages could be simple string, objects. The preceding line will load the script located at “worker.js” and execute it in the background. You need to call the Worker() constructor with the URI of a script to execute in the Worker thread as in the following:

```
1. worker.postMessage(num);
2.
3.  worker.onmessage=function(event) {
4.    if(event.data.percent!=null)
5.     $('#prg').val(event.data.percent)
6.    $('#msg').text(event.data.showText);
7.   }
```

If you want to get data from the Worker (for example, the output of the processed information, notifications, and so on) then you should set the Worker's onmessage property to an appropriate event handler function. The onmessage event is callback that receives the value from the background JavaScript. It also sends data back via postMessage.

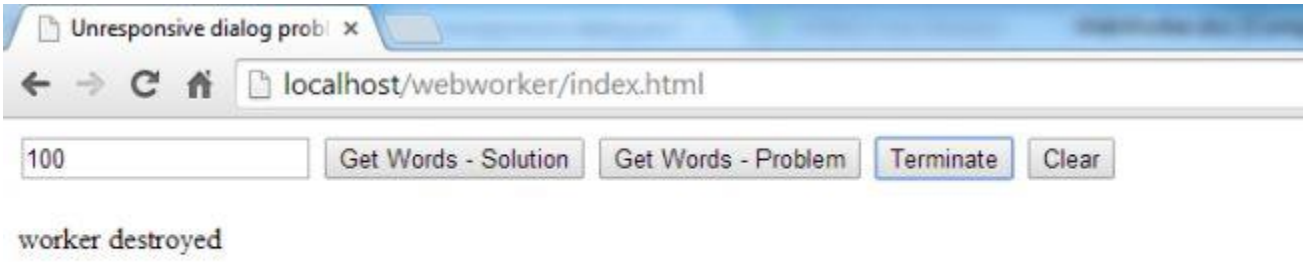
**Capturing error**

Web Workers support onerror event. The onerror event captures a JavaScript error that occurs in the background JavaScript.

```
1. worker.onerror=function(event) {
2.   $('#msg').text(event.lineno+ ' : '+ event.message);
3.  }
```

If you want to terminate WW in between then you can click the Terminate button that will kill between the worker processes as in the following:

worker.terminate();




You can use self or this keyword to access onmessage. It's sending data back to problem.html via self.postMessage.

So, communication between the foreground thread running problem.html and the background thread running solution.js is done via postMessage.

**Theory of WW**

WW runs JavaScript via background thread. Don't be surprised, JavaScript now has background thread execution capability. We know JavaScript is single-threaded and scripts execute sequentially.

But, it's a real thread, spawned by the OS, that executes in the background.

**Features available to WW**

Due to the multi-threaded behavior, a WW only has access to the following subset of JavaScript features:

- the navigator object
- the location object
- XMLHttpRequest
- setTimeOut/clearTimeOut/setInterval/clearInterval
- importScripts
- spawning other WW

**Limitations of WW**

It has some limitations apart from it's multithreading advantage, i.e., it can't access:

- DOM elements
- window object
- document object
- parent object

# JavaScript JSLint v1

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript. I hope this series on JavaScript is giving you good understanding and you are enjoying most parts of it.

We will begin this part by reading this phrase "The Software shall be used for Good, not Evil." - Wikipedia

We will discuss in this section about how to write good code. Our code shall be clean, beautified and other programmers are able to read & understand it; rather debug it. In order to achieve that we will look into process of linting and then apply in context of JavaScript.

**Lint**

It is a tool, which is used to flag code quality. It takes source code and find issues in it. This tool is available for many programming languages like C, PHP etc. Linting is a process of running a program that will analyze code for potential errors.

As JavaScript is so popular and widely used, a lint for JavaScript is needed. In JavaScript, I am a fan of JSLint. This tool was developed by Douglas Crockford http://www.crockford.com/. Let us deep dive into this tool.

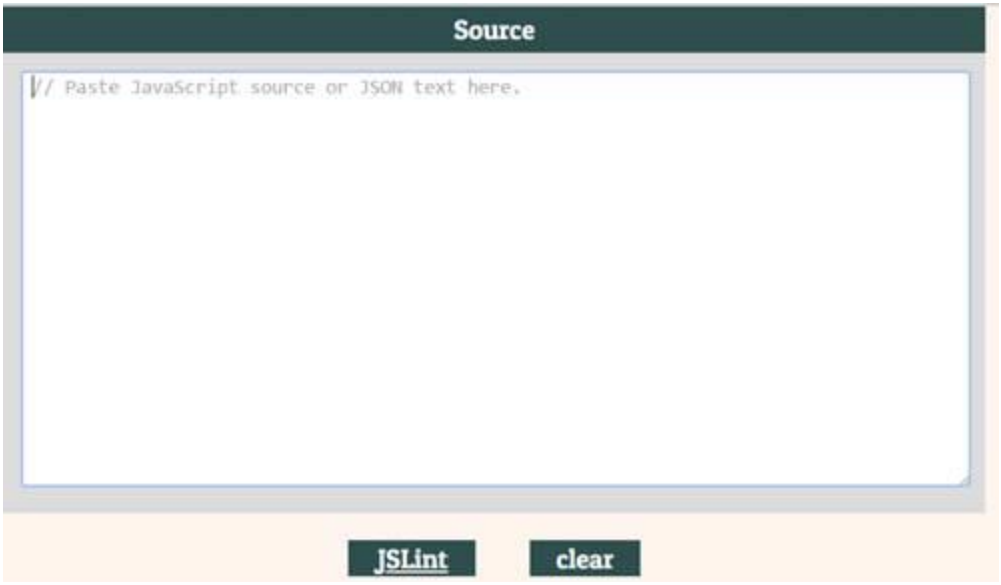
**JSLint**

JSLint is a code quality tool for JavaScript. There are various advantages of this tool:

- Provide another pair of eyes at your code
- Another layer of language on top of JavaScript
- Helps you write better JavaScript
- Reject the code which claims that it is perfectly fine

- Reduce error and give perfect code

  URL: http://www.jslint.com/




You can use an online version of JSLint and paste your JavaScript code & test it.

Here is a simple f1() function and we will validate in JSLint

```
'use strict';

function f1() {
    var x = 90;
    return x;
}
```

The above will validate perfectly fine and this code will sail perfectly. To see how JSLint helps let me modify code & generate warnings:

```
function f1(){
    var x=90;
    return x;
}
```

| Warnings | |
|---|---|
| Expected one space between ')' and '{'. | line 1 column 14 |
| `function f1(){` | |
| Expected ' "use strict": ' before 'var'. | line 2 column 5 |
| `var x=90;` | |
| Expected one space between 'x' and '='. | line 2 column 10 |
| `var x=90;` | |
| Expected one space between '=' and '90'. | line 2 column 11 |
| `var x=90;` | |

It is so fantastic to do statistical analysis of your code so quickly! If we review four warnings, three are related to 'whitespace'. One grabs my attention is

Expected '"use strict": 'before 'var'. line 2 column 5

**What is 'use strict'?**

Ans: We have not talked about 'use strict' rule so far. Nevertheless, this is an instruction that JavaScript code should be executed in 'strict mode'. This directive was introduced in ECMAScript 5. You can define scope of 'use strict' to the beginning of a script (global scope) or a function but you cannot keep it within block statements enclosed in {} braces. Below picture could explain it better.

```
'use strict';
function f1(){

    var x=90;
    return x;
}
```

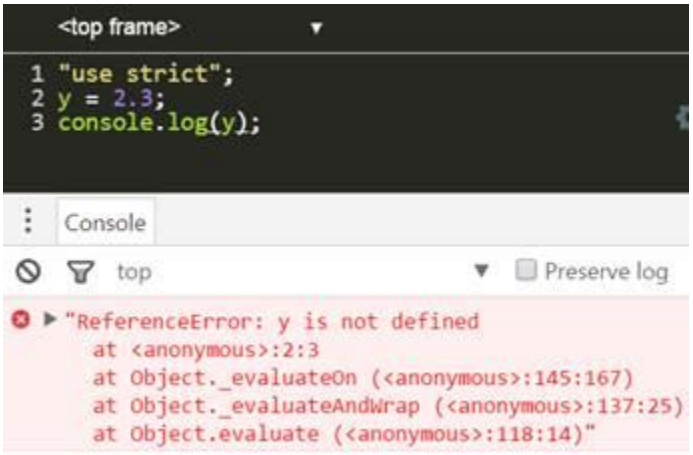
Global scope

```
function f1(){
    'use strict';
    var x=90;
    return x;
}
```

Function scope

Advantages of script mode are:

- Avoid created undeclared variable
- Cannot duplicate variable
- Avoid mistyping a variable to avoid global variable declaration




In above example ‘y’ was not declared but referred, hence it caused below error ‘y is not defined’. Switch gear back to JSLint now ï?Š

**Other examples where JSLint could help in finding issues in code:**

- Declaring variable again: In JavaScript if you declare variable again it does not pose any problem, as we know JavaScript uses ‘hoisting’, closures model. As good programmers, we shall not re-declare or duplicate variable declaration in same scope. Take a look at below code:

```
1. // fahrenheit to celsius code
2. 'use strict';
3.
4. function toFahreheit(celsius)
5. {
6.     var celsius = celsius;
7.     var F = celsius * 9 / 5 + 32;
8.     return F;
9. }
```

**Upon execution, JSLint will give below message:**

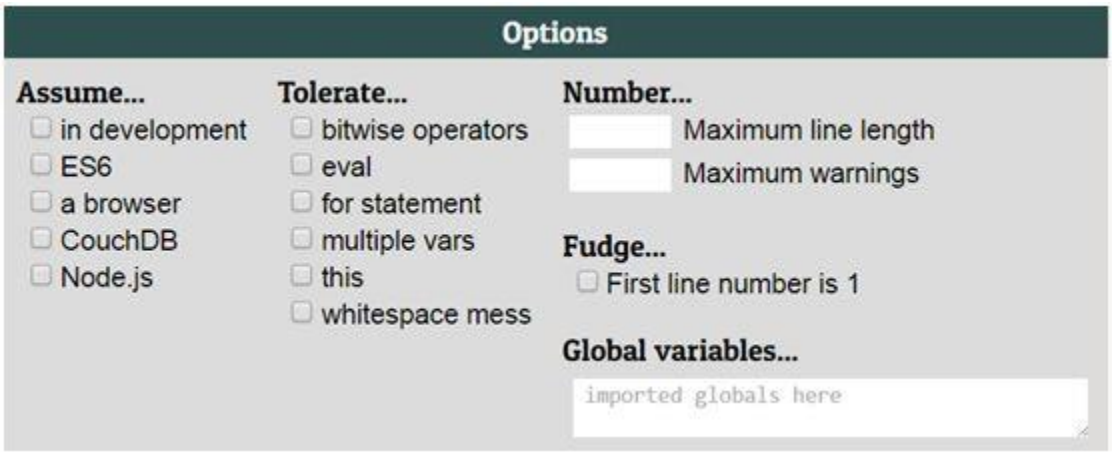
## JavaScript JSLint v2

JavaScript is a language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript. I hope this series on JavaScript is giving you a good understanding and you are enjoying most parts of it.

This is a continuation of the last part on JSLint, we will start with Options and command line utility of JSLint. Connect the dots where we left now, but read the last part (link) before reading this one otherwise you will feel nausea, drowsiness and feel unhappy.

Go to URL: http://www.jslint.com/

You can fiddle with JSLint and customize your warnings via Options below:



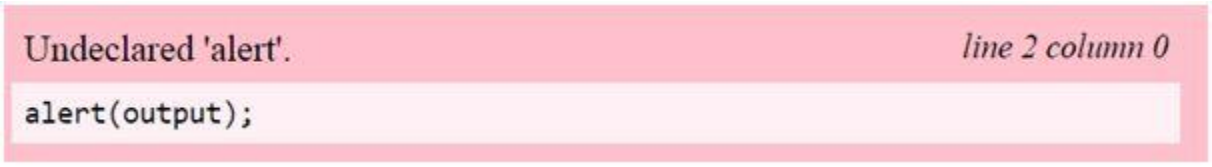

I am going to show commonly used options here:

- Assume => in development: select this when you are using debugging code, troubleshooting and code includes statements like console, alert etc.

```
1. var array = [2,4,16];
2.
3. var output = array.map(Math.sqrt);
```

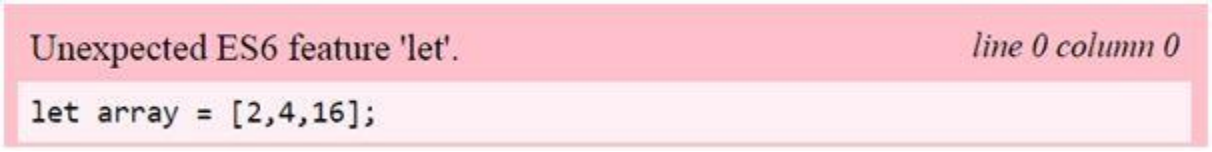
```
4.
5. alert(output);
```

If 'in development' is unchecked, you will get below warning:




- Assume => ES6 checkbox: if you're using ES6 features in your code and you try it without selecting this checkbox it'll give you warning, ex-

```
1. let array = [2,4,16];
2.
3. var output = array.map(Math.sqrt);
```

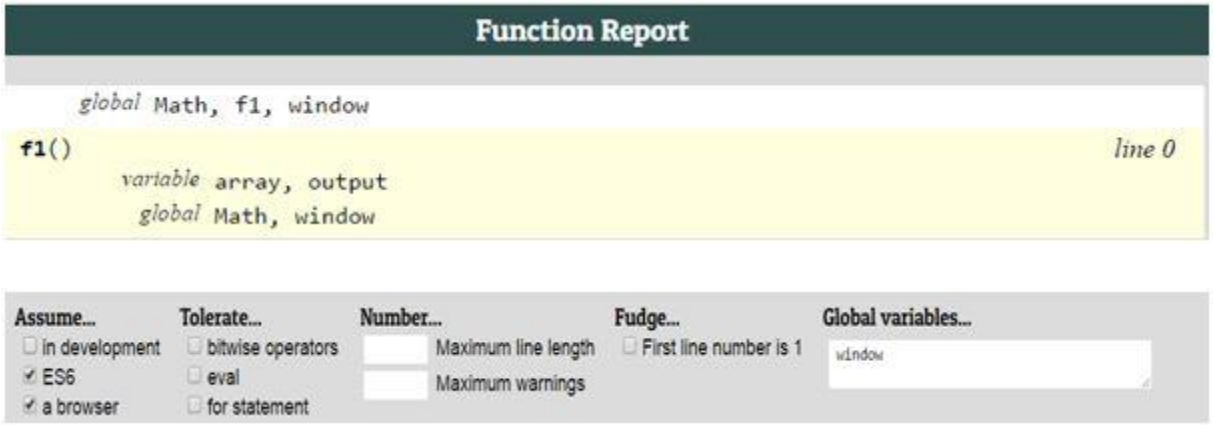



- Assume =>a browser: your code is leveraging global variables accessible in browser ex- window, document then you can click this checkbox and mention these specifically in global variables, refer snapshot below:

**Code:**

```
1. function f1() {
2.
3.    var array = [2,4,16];
4.
```

```
5.    var output = array.map(Math.sqrt);
6.
7.    window.alert(output);
8.
9.  }
```




**Function report**

It display an output showing about global objects and objects leveraged inside our function.

**Question:** What is the biggest problem in JavaScript?

**Answer:** I am going to share my opinion that it is not possible to determine at compile time if you misspell name of variables. The risk is it will create global undefined variables. As outcome, developers will spend a lot of time troubleshooting and debugging code. If front-end code grows exponentially then code smells in absence of right design and best practices.

**FYI:** Other advantage of JSLint is that it can read JSON also.

**Command line support**

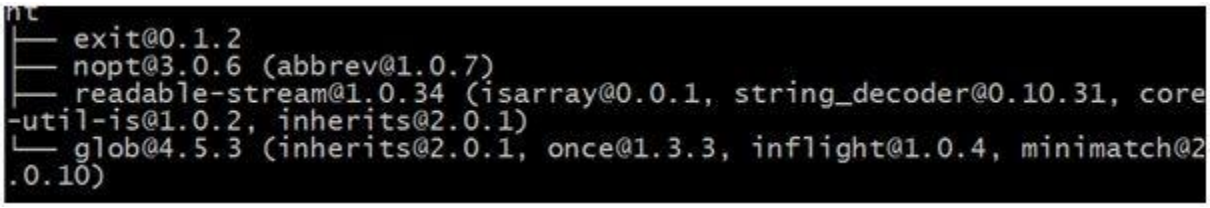
As developers we love command prompt. Good news is JSLint is available as node module and we will learn how to lint our JS files locally.

Assumption: I believe you have NodeJS installed, if not please refer my part number 6 of this series. We will first install jslintJSLint npm package in existing NodeJS repository.

**Steps:**

- Open GIT bash and goto NodeJS repository.
- Run below command to install jslint.

  npm install jslint -g




- Run jslint from command line to lint your code.

  Example lets revisit our script we wrote on JSlint.com and save as 'script.js'

```
1. 'use strict';
2.
3. function f1() {
4.
5.   var x ={a:1, b:2};
6.
7.   alert(x);
8.
9. }
```

```
$ jslint script.js

script.js
 #1 Missing space between '=' and '{'.
    var x ={a:1, b:2}; // Line 3, Pos 12
 #2 Missing space between ':' and '1'.
    var x ={a:1, b:2}; // Line 3, Pos 15
 #3 Missing space between ':' and '2'.
    var x ={a:1, b:2}; // Line 3, Pos 20
 #4 'alert' was used before it was defined.
    alert(x); // Line 4, Pos 5
```

Above Warnings we have seen at web application are now appearing on command line. The benefit of command line is integration with your code and during continuous integration & deployment.

- We could pass Options in command line utility as well, but before that check syntax:

```
$ jslint
No files specified.
Usage: C:\Users\Sumit Jolly\AppData\Roaming\npm\node_modules\jslint\bin
\jslint.js [--anon] [--ass] [--bitwise] [--browser] [--closure] [--colo
r] [--config] [--continue] [--debug] [--devel] [--edition] [--eqeq] [--
es5] [--evil] [--filter] [--forin] [--indent] [--json] [--maxerr] [--ma
xlen] [--newcap] [--node] [--nomen] [--on] [--passfail] [--plusplus] [-
-predef] [--properties] [--regexp] [--rhino] [--sloppy] [--stupid] [--s
ub] [--terse] [--todo] [--undef] [--unparam] [--vars] [--version] [--wh
ite] [--windows] [--] <scriptfile>...
```

- In order to remove above Warnings we could pass on Options as following:

```
$ jslint --white true --devel true script.js

script.js is OK.
```

# XMLHttpRequest API

The Web offers a variety of APIs to build your app or website. You could access using JavaScript code. These days the browser offers many APIs to developers. In this part, we will explore XMLHttpRequest API.

**XMLHttpRequest (XHR)**

It is basic of client / server communication via Browser. This API provides an easy way to get data from URL without full refresh. It was originally developed by Microsoft, and adopted by others like Google, Apple, and Mozilla. The advantage is to update parts of a web page, without full refresh or reloading the whole page.

XHR is used by many popular frameworks like AngularJS, e.g., $http is a service which operates on this. If you read AngularJS code.

You will find $http service description:

> * @description
>
> * The `$http` service is a core Angular service that facilitates communication with the remote
>
> * HTTP servers via the browser's [XMLHttpRequest]
>
> * object or via [JSONP]

**Example**

In 1st example we will create XHR object and download jQuery by sending GET request,

```
1. varmyRequest = new XMLHttpRequest(); //myRequest is an object created by XHR API
```

```
2. console.log(typeof(myRequest)); // object
3. myRequest.open('GET', 'https://code.jquery.com/jquery-2.2.3.min.js');
4. myRequest.send();
5. myRequest.onreadystatechange = function()
6. {
7.     if (myRequest.readyState == 4 && myRequest.status == 200)
8.     {
9.         console.log(myRequest.responseText);
10.    }
```

Hence, we will get jQuery in myRequest.responseText. Great!

Let us understand nitty gritties of few components in above example.

Open method syntax:

```
open(method: string, url: string, async?: boolean, user?: string
, password?: string): void
```

- **method:** GET, POST
- **url:** Location of API
- **async:** True means asynchronous or false means synchronous
- **user:** If API needs authentication
- **password:** If API needs password

After open method is used we call send() to trigger the call at server but what if we change our code as below:

```
1. varmyRequest = new XMLHttpRequest(); //myRequest is an object created
   by XHR API
2. console.log(typeof(myRequest)); // object
3. myRequest.open('GET','https://code.jquery.com/jquery-2.2.3.min.js');
4. myRequest.send();
5. console.log(myRequest.responseText); // may not get data in responseText
   and there will be no error in console.
```

**Question:** What is the benefit of onreadystatechange after we called send method?

**Answer:** The reasons to use onreadystatechangeevent is to ensure data in responseText. To understand this, we will understand readyState property has different value between 0 to 4:

0: Request not initialized
1: Server connection established
2: Request received
3: Processing request
4: Request finished and response is ready

To try this modify program like below:

```
1.  varmyRequest = new XMLHttpRequest();
2.  myRequest.onreadystatechange = function()
3.  {
4.     switch (this.readyState)
5.     {
6.        case 0:
7.           console.log('request not initialized ');
8.           break;
9.        case 1:
10.          console.log('server connection established ');
11.          break;
12.       case 2:
13.          console.log('request received ');
14.          break;
15.       case 3:
16.          console.log('processing request ');
17.          break;
18.       case 4:
19.          console.log('request finished and response is ready ');
20.          console.log(myRequest.responseText);
```

```
        break;
    }
}
myRequest.open('GET', 'https://code.jquery.com/jquery-2.2.3.min.js');
myRequest.send();
```

**Output**

```
server connection established
request received
processing request
request finished and response is ready
/*! jQuery v2.2.3 | (c) jQuery Foundation | jquery.org/license */
!function(a,b){"object"==typeof module&&"object"==typeof module.exports?modu
requires a window with a document");return b(a)}:b(a)}("undefined"!=typeof w
```

It has not printed “case 0: console.log('request not initialized ');” because when you create a new XHR request it’s state is already 0, so there is no change that is triggered.

Other Events

- **abort:** To terminate XHR request if its sent
- **progress:** You can monitor progress while result is downloading
- **error:** Incase of any error generated

**Other Properties**

- **responseType:** It could vary depending upon response sent by the server, ex- json, document, text, blob.
- **responseURL:** This will be the final URL after redirects.
- **timout:** Number of milliseconds a request can take before terminated.

We understood one HTTP verb GET, the other common HTTP verb is POST. Posts differentiate, as you are required to pass requestHeader depending upon request ex- you are submitting a form:

```
myRequest.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
myRequest.send("fname=sumit&lname=jolly");
```

**Command line utility**

I am always excited to do things from command line.

**Question:** Is there a way we can leverage command line utility to test?

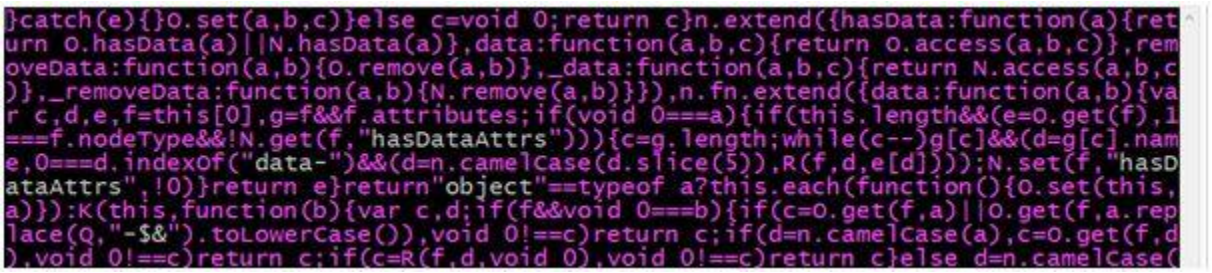
**Answer:** Yes, use curl command. You can download curl utility and run it:

*curlhttps://code.jquery.com/jquery-2.2.3.min.js*


&I am saving in jQuery.js via redirection operator '>'.

You can run VIM editor to review jQuery.js:

$ vi jQuery.js



## Web Storage API

The Web offers variety of APIs to build your app or website. You could access using JavaScript code. These days browsers offer many APIs to developers. In the last part, we read about XMLHttpRequest API and in this part, we will understand Storage APIs available.
Before I dig into storage we will understand key-value pair.

### Key-Value pair (KVP)

KVP stores two values together. It is a set of two linked data items; a unique key and some value. This is frequently used in hash, map, dictionary, JSON, configuration files etc. Example – below representation of KVP could be translated in the form of Object in JavaScript.

| *key* | *value* |
|---|---|
| width | 300 |
| height | 200 |
| title | Menu |

```
1. var obj = {width: 300, height:200, title: 'Menu'};
```

Now let's proceed with Storage API

### Web Storage API

When we think about storage we think of Database, Files etc. With HTML5 web storage provides a client-side method for storing information.

The two ways within Web storage are:

- **sessionStorage:** maintains a storage as long as browser is open, including page reloads and restores. It gets clear when session ends.

- **localStorage:** same thing like sessionStorage, but persists even when the browser is closed / opened again. This means it has no expiration time.

**Note:** There was a way in Firefox 9, i.e., globalStorage but it was deprecated after HTML5 introduced this local storage.

**Access client-side storage**
These are available via Window.sessionStorage & Window.localStorage. Both of these provide access to create, get, modify, delete client-side storage depending upon session or local store.

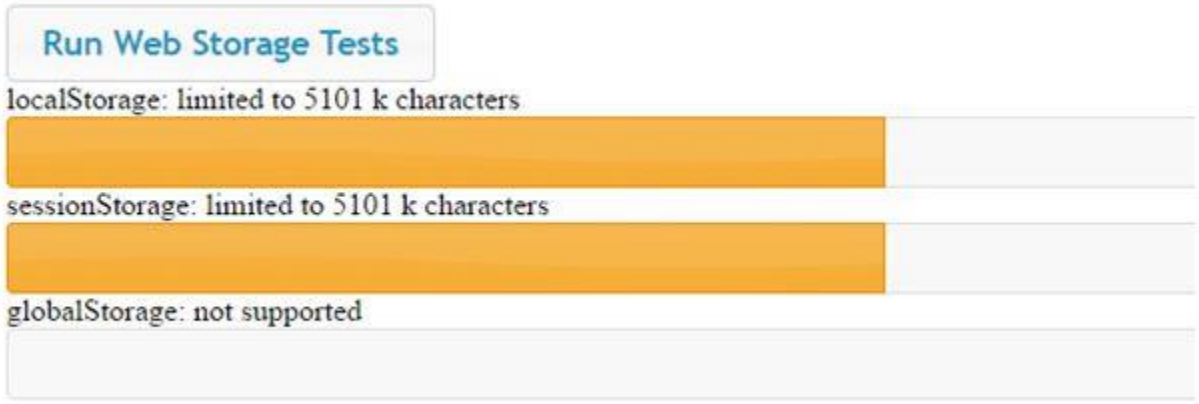
**Facts about client-side storage**

**Question:** How much storage does browser provide?

**Answer:** Generally, 5MB, 5MB is available for sessionStorage& localStorage respectively.

**Question:** How can we check the available storage in our browser?

**Answer:** You can leverage utility to run these tests at your browser and it will show you the output. I conducted test at my browser Chrome 50 & Firefox 45 and both gave same result, i.e., 5 MB.




**A misconception**

Please note 5MB does not mean 5 million characters are allowed. Developers think that each character occupies 1 byte, which is correct for some programming languages. But JavaScript occupies 2 bytes each character as it stores string on basis on UTF-16.

**localStorage**

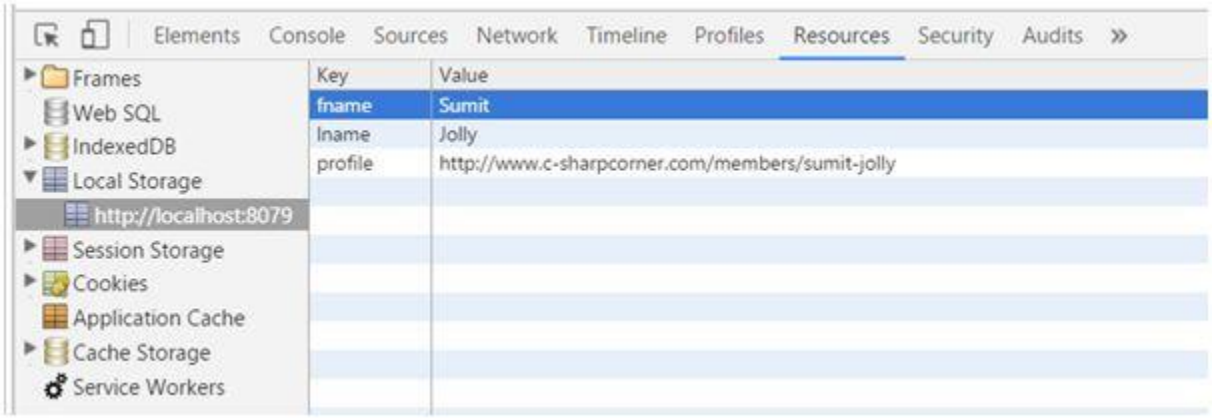
**Write localStorage**

Below code will write data into localStorage and using setItem() function:

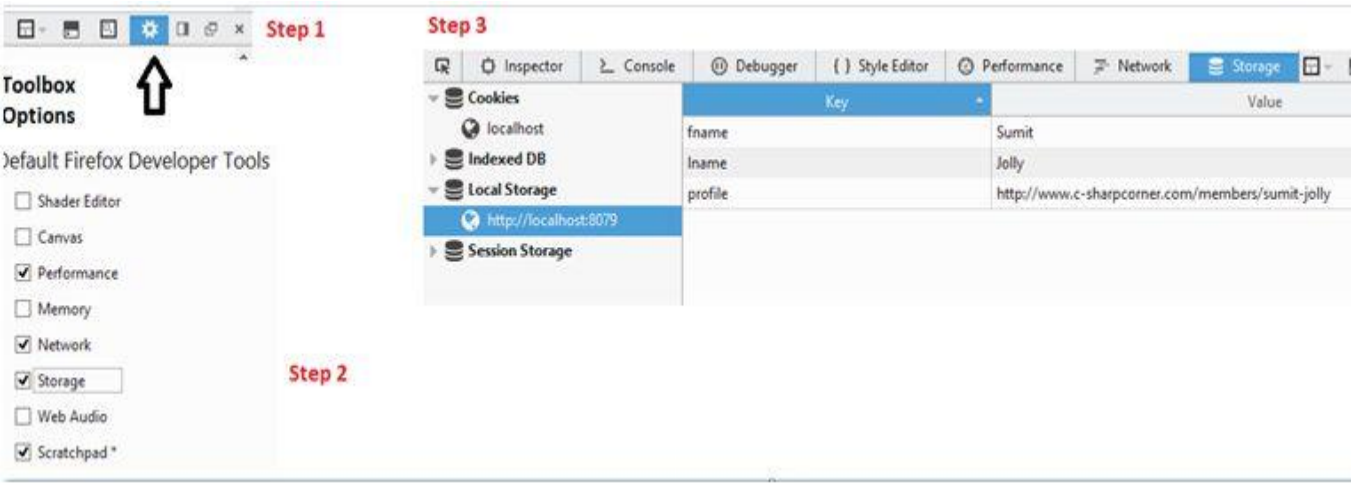
```
1. <script>
2.    localStorage.setItem("fname", "Sumit");
3.    localStorage.setItem("lname", "Jolly");
4.    localStorage.setItem("profile", "http://www.c-
   sharpcorner.com/members/sumit-jolly");
5. </script>
```

Configure a website in local Web server and run
URL: *http://localhost:8079/test.html*

You can see '**Local Storage**' content in browser after clicking Resources in Chrome.




If you run on Firefox then it will also create a Local Storage but view is disabled by default. To enable it goto '**Toolbox options**' and click Storage checkbox.




Aforesaid, localStorage has no expiration and even you close browser it'll stay there. You can try this and would be able to see data in localStorage of browser.

**Read localStorage**

You can retrieve data from localStorage using getItem() function. I prefer iterating to KVP.

**Example:**

```
(function() {
  for (var i = 0; i < localStorage.length; i++){
    console.log(localStorage.getItem(localStorage.key(i)));
  }
})(); // self-invoking function will give below output
```

**Output**

*Sumit*
*Jolly*

http://www.c-sharpcorner.com/members/sumit-jolly

There are many other functions and properties you can explore like removeItem, clear, length.

**sessionStorage**

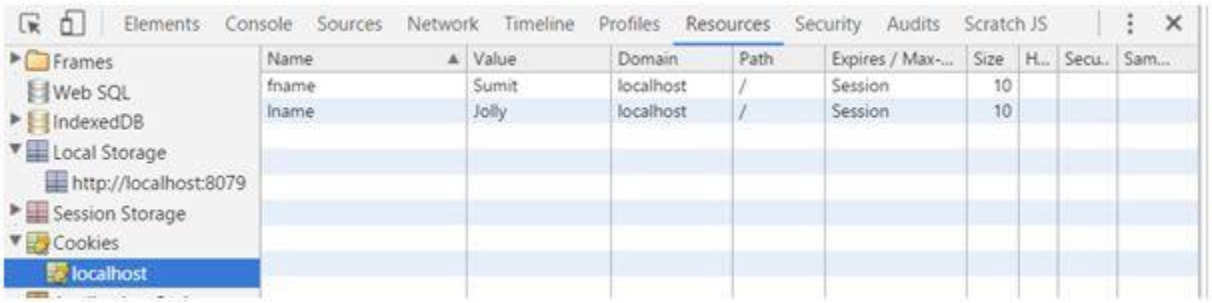
It is same as localStorage with the difference that it is associated with session only. After you close browser window it is gone. I believe if you have more security aspect than go for sessionStorage rather localStorage.

**Cookies**
For many years, cookies are used to remember user information. So whenever a user visits page his/her information is stored in a cookie. It is also saved as KVP.

A simple cookie can be created like:

```
document.cookie = "fname=Sumit";
document.cookie = "lname=Jolly";
console.log(document.cookie); // fname=Sumit; lname=Jolly
```

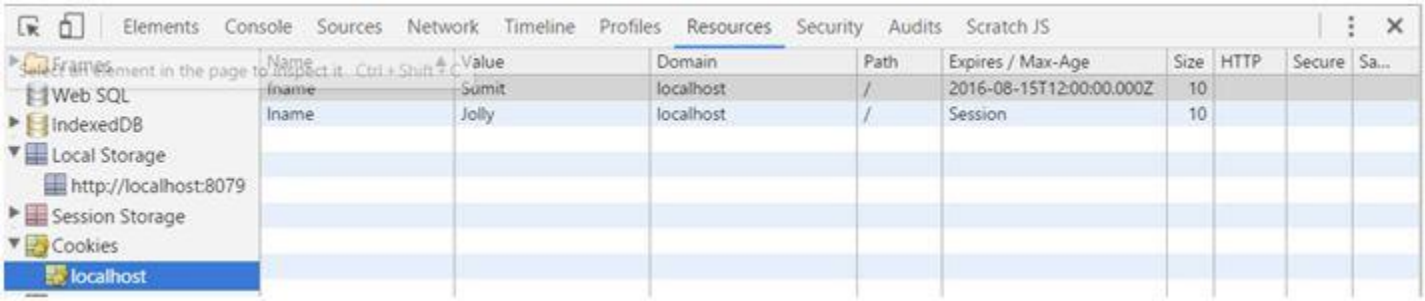
You can check it in browser Resources Developer Tools.




Notice an expiry date is set to Session. The reason we have not defined an expiry date to a cookie, so default is session.

You could do via setting an expiry date alongwith.

```
1. document.cookie = "fname=Sumit; expires=Mon, 15 Aug 2016 12:00:00 UT
   C";
```




Here are the differences between three storage types we discussed:

| localStorage | sessionStorage | Cookies |
|---|---|---|
| Not added | Not added | Added to every HTTP request |
| No expiry | date Expire with session | Has both features i.e., session and you can define expiry date |
| Limit of 5MB | Limit of 5MB | Cookies give you a limit of 4096 bytes |

<=”” b=”” style="outline: none; color: rgb(51, 51, 51); font-family: "Open Sans", sans-serif; font-size: 15px; font-style: normal; font-variant: normal; font-weight:

normal; letter-spacing: normal; line-height: 21px; orphans: auto; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 1; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255);">Summary

Now the world is going towards fat client and thin server. With browsers becoming more capable of faster processing of data and powered with various resources, think of client before server for good performance of your web application.

## Application Cache API

JavaScript is language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In the last part we learned about Storage API available in browser. Now, we will understand cache strategies and offline cache capability of application. Let me start with explanation of short anecdote “First view”.

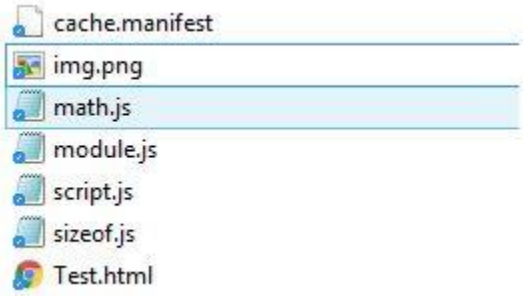
### First view

First view means when you launch a new application and users will be hitting it for the first time. This testing shall ideally be done by clearing your browser cookies and cache to test how ‘first time’ user will have an experience of the site.

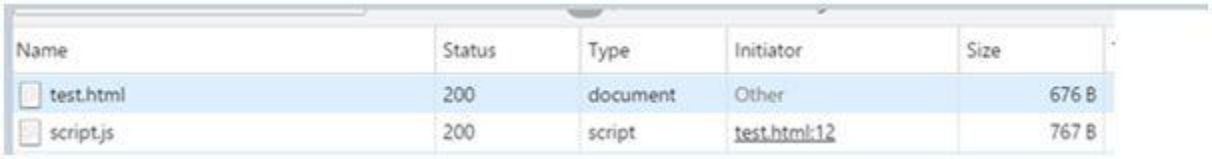
### Cache

Cache exists in your processor in the form of L1, L2 cache for faster processing. Here we are referring to browser caching. I have created an application in IIS server.

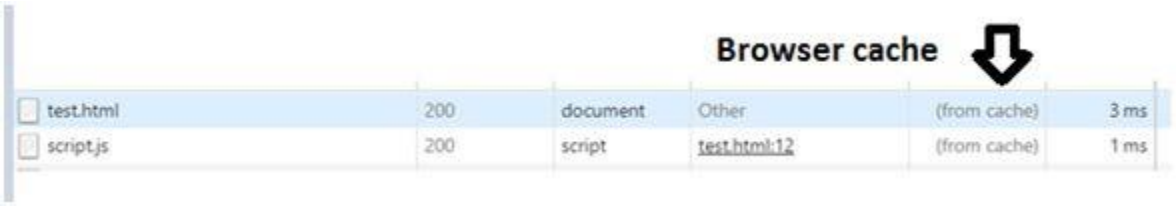
For now, let us focus on few files test.html, img.png and script.js. This site is setup at*http://localhost:8079/test.html* [however you can configure any other port]




When I access site first time, it's first view and rendered as:




Next time when I refresh I get below view because it's rendered from browser's cache.




Browser caches static content like css, html, images so the webpage loads faster for repeated visitor. Browser usually reserves certain amount of disk space, like 12MB for caching. There are many ways for cache control in modern browsers.

**HTTP cache headers**

There are two primary cache headers, Cache-Control and Expires.

**Cache-Control:** you can specify public OR private.

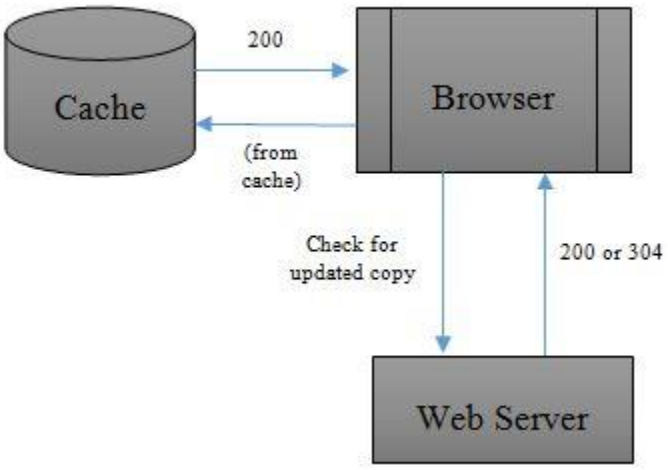
- **Public:** Can be cached by any proxies in between

- **Private:** Can be cached only by the end-client and not by proxies in between

You can also mention expiry date along with using max-age in seconds, ex-

*Cache-Control:public, max-age=31536000 //31536000 = 1 year*

Expiry date tells browser when to retrieve this resource from the server. It is also called as conditional requests. The process is well explained by below diagram:




Here is a sample application you can test at Heroku and review request & response headers

URL: *http://http-caching-demo.herokuapp.com/?cache=true*

**Cache-Control:** public, max-age=30
**Connection:** keep-alive
**Content-Length:** 379
**Content-Type:** text/html;charset=utf-8
**Date:** Mon, 16 May 2016 09:40:07 GMT
**Expires:** Mon, 16 May 2016 09:40:37 GMT

URL: http://http-caching-demo.herokuapp.com/?etag=true

```
Connection: keep-alive
Content-Length: 0
Date: Mon, 16 May 2016 09:43:00 GMT
Etag: "15f0fff99ed5aae4edffdd6496d7131f"
```

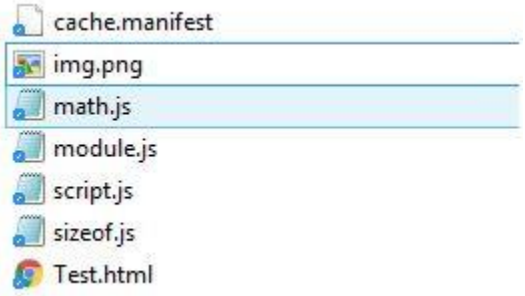
**Offline capability**

As the web became advanced the need for applications to work in offline mode arose. Offline? Yes, it is possible via Application Cache interface. It gives a lot of advantages, these are:

- Offline browsing
- Server load gets reduced
- Speed improves because network payload is avoided
- User can navigate around incase server / site goes down

**Cache manifest file**

Revisit our code files I created for caching example. You can see cache.manifest file. It is an instruction to browser to cache resources mentioned in the file to work offline.




**Structure of manifest file**

This has three sections:

- **CACHE MANIFEST –** Files which will be cached after they are downloaded from the web server, ex- jpg, png, html
- **NETWORK –** Which will not be cached, ex- aspx, asp, php files
- **FALLBACK –** If a particular file / page is inaccessible then leverage file mentioned under this section. It is fallback option.

cache.manifest file: I am caching three static files here:

*CACHE MANIFEST*
*/img.png*
*/test.html*
*/script.js*

**Refer manifest file**

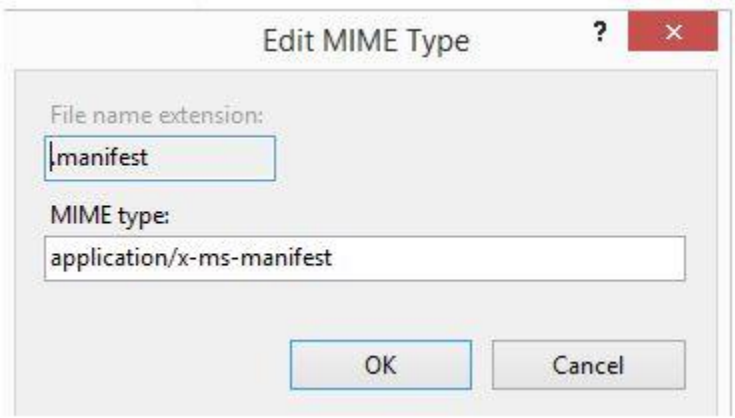
You have to refer cache.manifest file in HTML tag, ex, in test.html.

```
<!DOCTYPE html>
<html manifest="cache.manifest">

<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-16">
    <meta name="robots" content="noindex, nofollow">
    <meta name="googlebot" content="noindex, nofollow">
    <title>JS</title>
</head>

<body style="background-color:white ">
    <script src="script.js"></script>
    <imgsrc="img.png">
        <script>
        </script>
</body>

```

```
18.</html>
```

### Configure manifest file at Web Server

The server has to send manifest file to client so this extension shall be registered at server. In IIS add below MIME type:

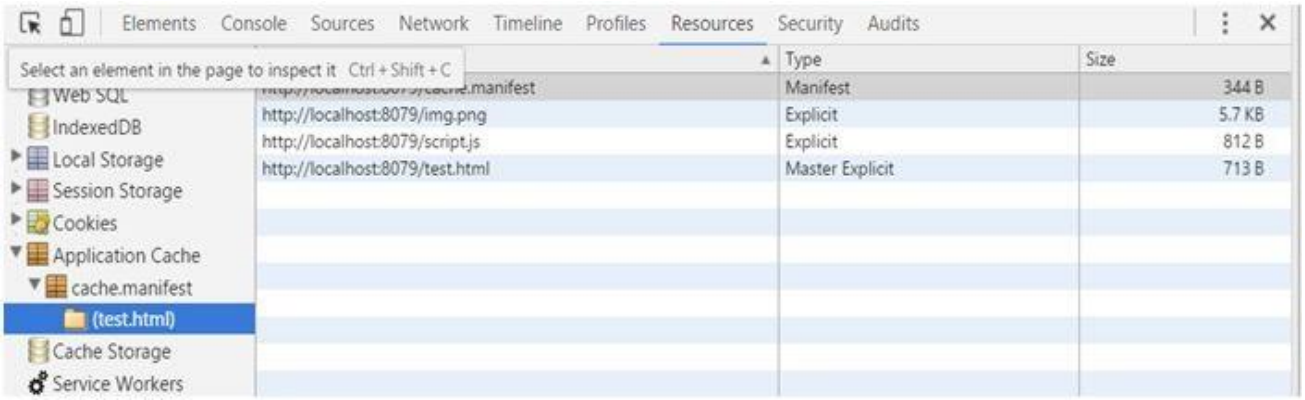



### Run website

Now you run your website and review messages generated at Console window

```
Creating Application Cache with manifest http://localhost:8079/cache.manifest
Application Cache Checking event
Application Cache Downloading event
Application Cache Progress event (0 of 3) http://localhost:8079/test.html
Application Cache Progress event (1 of 3) http://localhost:8079/img.png
Application Cache Progress event (2 of 3) http://localhost:8079/script.js
Application Cache Progress event (3 of 3)
Application Cache Cached event
```

You could see cache.manifest sent by server and it is giving instruction to browser to cache three files. The same could be reflected by monitoring at Resources tab.




The files mentioned in cache.manifest file are available in Application cache.

**Use Application Cache**

Steps to use / test offline browsing capability,

- Go to IIS and Stop the website
- Run the site again and it'll work fine

# JavaScript WebSocket

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In this part, we will understand WebSocket, which is a new revolution in client server communication. Modern browsers support this protocol. The server shall also support WebSocket thus a handshake between client & server is possible.

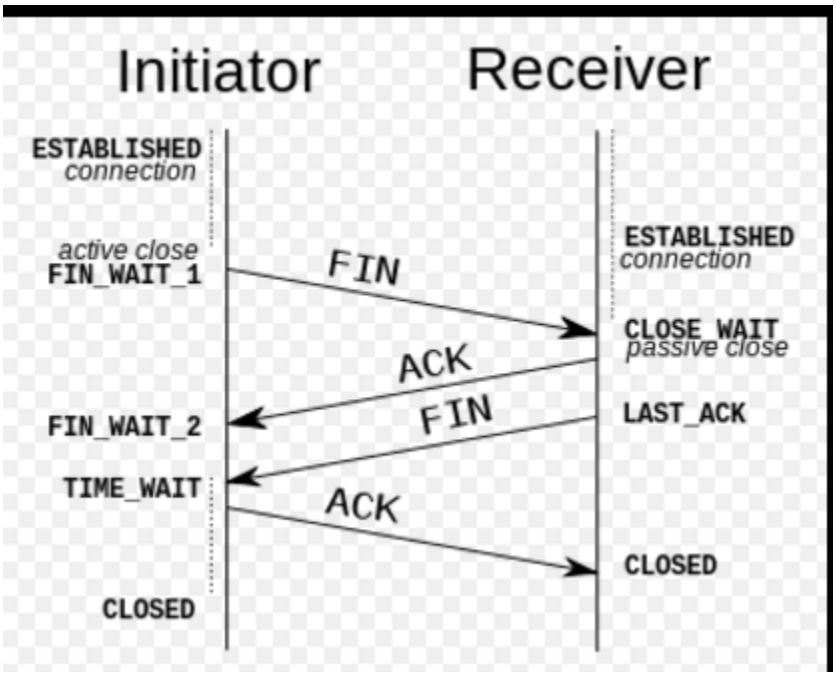
### WebSocket

It is based on ws schema and establishes full duplex connection between client & server. At client-end, it is a browser and it can work with any server capable of running WebSocket protocol.

### Advantages of WebSocket

- **Faster than TCP**

  Quick response from server where you are using HTTP connection to connect. TCP connection lifecycle is SYN, SYN/ACK, ACK and then send a GET request. TCP header contains 10 mandatory fields and an optional field. TCP has close ends at both sides – Refer useful diagram from Wiki.




However, WebSocket simply receive the response with a small header.

- **Better than other methods like polling, AJAX**

  For duplex communication, developers used technique like polling to constantly synchronize with server. This technique was never effective for duplex communication. Hence, WebSocket came into picture.

- **Easy to use**

  It is easy to use at both ends. At server, it is independent of platform too.

**Disadvantages of WebSocket**

- **Open connection**

  The disadvantage is that it keeps the connection open on the server for the duration of the time user is interacting with the page. Therefore, it consumes more resources on the server.

- **Single channel communication**

  If your app does not need too much duplex communication with the server then you can think of using other approaches like long polling, AJAX over WebSocket.

**WebSocket server**

I suggest to use /download any server supporting WebSocket. In IIS, you can also enable WebSocket via adding Application Development features. For testing, I want to use WebSocketd server like UNIX.

**Steps to run server:**

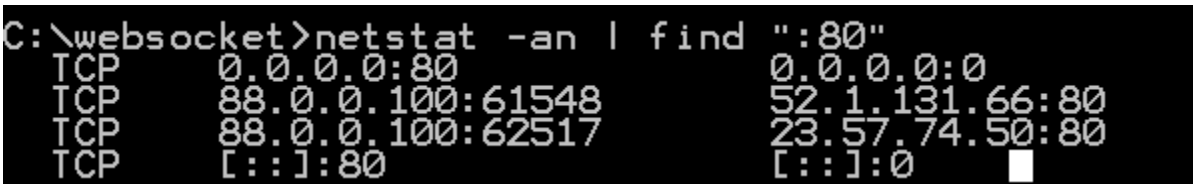
- Download WebSocketd from URL.
- Extract zip file in a folder like C:\websocket
- The server is platform independent and can run any program
- Create a server program in C# and build to generate myProgram.exe,

```
1. using System;
2. using System.Threading;
3.
4. //myProgram.cs
5. class Counter
6. {
7.    static void Main()
8.    {
9.       for (int i = 1; i <= 10; i++)
10.      {
11.         Console.WriteLine(i);
12.         Thread.Sleep(500);
13.      }
14.   }
15.}
```

- You can copy myProgram.exe into c:\websocket for convenience
- Open command shell and goto c:\websocket and run below command

- websocketd.exe -- port 80 myProgram.exe


It gave above error because port 80 is already in use. We can validate by using netstat command.




So I will use port 8082 to run WebSocket server.

websocketd.exe -- port 8082 myProgram.exe


**WebSocket client object**

With this API, you can send and receive messages to a server without having to poll the server.

**Receiving messages**

After connection is established, we can receive messages using onmessage event.

```
1. var ws = new WebSocket('ws://localhost:8082/');
2. // this will establish connection with server at port 8082
3. //notice ws: this is new URL schema for WebSocket connection
```

```
4. ws.onmessage = function(event) {
5. console.log('Count is: ' + event.data);
6. }; // this will receive output from server in event.data
```

Console Output

```
OnOpen
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

Server Ouput

```
C:\websocket>websocketd.exe --port 8082 myProgram.exe
Mon, 16 May 2016 19:51:09 +0530 | INFO    | server      |  | Serving using applica
tion    : .\myProgram.exe
Mon, 16 May 2016 19:51:09 +0530 | INFO    | server      |  | Starting WebSocket se
Mon, 16 May 2016 19:57:06 +0530 | ACCESS  | session     |  url:'http://localhost:80
82/' id:'14634088261499232OO' remote:'::1' command:'.\myProgram.exe' origin:'htt
p://localhost:8079' | CONNECT
```

The above circled in red runs myProgram.exe and send data to client which is shown in Console Output of browser

### Sending messages

After connection is established via onopen event, we can send messages using send method.

```
1. ws.onopen = function () {
2. console.log('OnOpen');
3. ws.send('my message');
4. }
```

### Log errors

Use onerror event to trap error generated from WebSocket,

```
1. ws.onerror = function (error) {
2. console.log('WebSocket Error ' + error);
3. };
```

### ReadyState

If you remember, we have ready state in XHR API. Similarly, WebSocket also maintains connection state. It starts from 0 to 3,

- 0 | Connecting – the connection is not yet open
- 1 | Open – the connection is open & ready to communicate
- 2 | Closing – the connection is closing
- 3 | Closed – the connection is closed

**Close**

To close an open connection use close method, like ws.close();

# Parse JSON

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In this part, we will understand JSON in detail as a Data Interchange format. Douglas Crockford as “The Fat-Free Alternative to XML” introduced it. We will also understand difference with XML and how could we access it using JavaScript.

### JSON (JavaScript Object Notation)

JSON is an easier-to-use alternative to XML. As we know, XML has been there for many years as a data interchange format. XML was better than SGML. We will compare JSON & XML shortly, but before it is good to understand the JSON in detail.

### JSON structure

It is built on two structures:

- A collection of name/value pairs.
- An ordered list of values like array, list

### JSON syntax

- The data is written as name/value pairs, like JavaScript object properties, e.g.,
  1. {“fname”:”sumit”}
- JSON objects are inside curly braces {}, separated by “,”

  1. {“fname”:”sumit”, “lname”:”jolly”}
- Arrays are written inside square brackets

```
1. {'students': [
2. {'name':'Ravi','year':'1st'},
3. {'name':'Abhishek','year':'1st'},
```

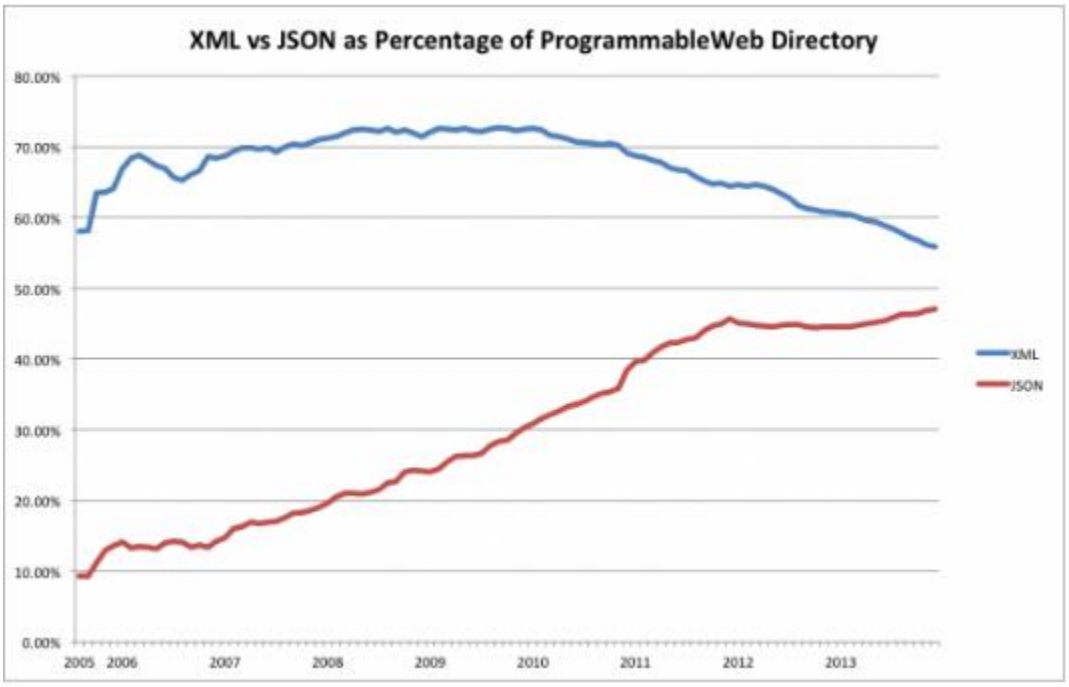
```
4.  {'name':'Neha','year':'2nd'}]};
```

**Comparison between JSON and XML**

It is obvious to compare the two formats:

| JSON | XML |
|---|---|
| JSON is less verbose | XML has lot of baggage and has lot of opening and closing tags |
| Easy readable format by humans | Complex to read large XML files |
| JSON is data-oriented | XML is document-oriented |
| JSON notation is built in JavaScript | XML needs extra library to process it |
| JSON is just a data interchange format | XML is more than interchange format, it's like a language |
| No namespace concept, each object is its own namespace. So names can be repeated | XML supports namespaces and prefixes |

Each has its own advantages and disadvantages, but JSON is definitely rising for last many years. There is popular chart depicting usage of XML vs JSON released by http://www.programmableweb.com/,




**JavaScript JSON parsing**

JavaScript contains window.JSON object for parsing JavaScript Object Notation and converting values to JSON. There are two methods associated with it:

**JSON.parse()**

It can be used to convert a JSON text into Object, e.g.

```
1. var json = '{"result":true,"count":1}',
2. obj = JSON.parse(json);
3. console.log(obj);
```

**Syntax**

*JSON.parse(text[, reviver])*

**text:** the string to parse a JSON

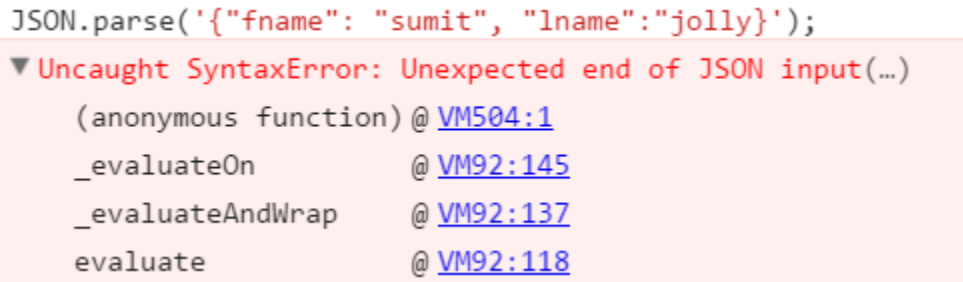
**reviver (optional)**

It is a function to screen or modify values which are being parsed. It is invoked on each data item being parsed, like an iterator. It takes two arguments, like KVP name and value. You need to address topmost value separately in reviver call. Example,

```
1. JSON.parse('{"fname": "sumit", "lname":"jolly"}', function(k, v) {
2. if (k === '') { return v; } // if topmost value, return it,
3. console.log(v);
4. return v.toUpperCase();
5. }); // output will be name in upper case
```

**Syntax Error**

In case there is any error in string or reviver then a syntax error is thrown example- I removed on double-quote from lname value.




**JSON.stringify()**

It converts a JavaScript value to JSON string.

**Syntax:**

*JSON.stringify(value[, replacer[, space]])*

**value:** The value to convert to a JSON string

**replacer (optional):** This second argument is used to screen and modify the results like reviver in JSON.parse. It accepts KVP name-value pair:

**space (optional):** We can add number of whitespace characters also,

```
1. var numberJson = ['one', 'two', 'three'];
2. var json = JSON.stringify(numberJson, function(k,v){
3. if (k === '') { return v; }
4. console.log(v);
5. return v.toUpperCase();
6. }, 4);
7. console.log(json); // I added 4 characters as whitespace, so below output wi
   ll be generated
```

```
[
    "ONE",
    "TWO",
    "THREE"
]
```

# Save Coding Time With Lodash.js

JavaScript is a language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In this part, we will explore functional programming aspects of JavaScript using a popular library called Loadash.js. It makes JavaScript easier. I use Lodash.js as Swiss Army knife to help me in breaking complex problems into smaller chunks and faster delivery.

## Introduction

In day-to-day programming, we work on solving business problems and working with arrays, numbers, objects, strings etc. Loadash.js works great with these:

- Arrays, objects & strings
- Testing values
- Creating composite functions

This library is released with MIT license Download.

**Note:** I am going to refer 4.12 of this library.

## Use cases

I will demonstrate benefit of using Lodash.js to simply common problems. It also enhance developer experience to understand and read code better.

**Case 1:** Sort an array and get unique values.

## Naive method

```
1. varary = [2, 4, 2, 5, 4, 7];
2. varsortAry = ary.sort(); // [2, 2, 4, 4, 5, 7]
3. //run two loops to traverse and match each element, e.g., 2 ===2; 2===4
```

```
for (vari = 0; i < sortAry.length; i++)
{
    for (var j = i + 1; j < sortAry.length; j++)
    {
        if (sortAry[i] === sortAry[j]) //equal value and same type operator
        {
            deletesortAry[i]; //delete element and place 'undefined'
        }
    }
}
console.log(sortAry); // [undefined × 1, 2, undefined × 1, 4, 5, 7]
//Now we can remove undefined via using filter function
varfinalAry = sortAry.filter(function(v)
{
    return v != undefined
})
console.log(finalAry); // [2, 4, 5, 7]
```

**Using Loadash.js**

```
varary = [2,4,2,5,4,7];
varsortAry = ary.sort();
console.log(_.uniq(sortAry)); // [2, 4, 5, 7]
Outcome: reduced lot much code by using _.uniq function of Loadash.js
```

**Case 2:** Find a user who based on criteria,

**Naive method**

```
var users = [
{
    'user': 'Peter',
    'age': 36,
    'isMember': true
},
{
```

```
    'user': 'Chris',
    'age': 40,
    'isMember': false
},
{
    'user': 'Alan',
    'age': 20,
    'isMember': true
}];
for (vari in users)
{
    if (users[i].age > 38)
    {
        var user = users[i];
        break;
    }
}
console.log(user); // Object {user: "Chris", age: 40, active: false}
```

**Using Loadash.js**

```
var users = [
{
    'user': 'Peter',
    'age': 36,
    'isMember': true
},
{
    'user': 'Chris',
    'age': 40,
    'isMember': false
},
{
    'user': 'Alan',
    'age': 20,
```

```
    'isMember': true
}];
_.find(users, function(o)
{
    return o.age > 38;
}); //// Object {user: "Chris", age: 40, active: false}
Outcome: reduced lot much code by using _.find
function of Loadash.js
```

**Case 3:** Loop over collection.

**Naive method**

```
for(vari=0;i<users.length;i++){
    console.log(users[i]);
}
```

**Loadash.js**

```
_.times(users.length, function(index){
    console.log(users[index]);
});
Outcome:readability is better with loadash.js
```

**Case 4:** Truncate a string.

I never wrote a code using Naïve method so jumping directly on Lodash.js implementation. This truncate feature is useful when you are dealing with large text of data and want to truncate it.

**Loadash.js**

```
varstr="hello world"; // below will truncate string accordingly
_.truncate(str, {'length':7}); // hell...
```

**Case 5:** Shuffle a response.

If you have noticed in online assessments questions are shuffled for each candidate. Lodash.js offers awesome function to help with.

```
1. var fruits =['apple','orange','banana','pomegranate'];
2. _.shuffle(fruits); // ["orange", "apple", "pomegranate", "banana"]
```

**Note:** The output of _.shuffle could vary because it uses a random permutation Fisher-Yatesalgorithm.

**Case 6:** Gets a random selection.

```
1. var fruits =['apple','orange','banana','pomegranate'];
2. _.sample(fruits); // pomegranate; this output could vary
```

**Case 7:** Map and Reduce functions.

If you remember we explored map and reduce functions in Array methods in part 17.

Loadsh.js provides a convenient approach for both these functions, e.g., get names from users collection and we will use Map function.

```
1. _.map(users, 'user'); // ["Peter", "Chris", "Alan"]
2. //To change all users name into UpperCase
3. _.map(users, function(u){varobj=u.user; return obj.toUpperCase();});
4. //["PETER", "CHRIS", "ALAN"]
```

Reduce function has similar arguments but it applies reduce approach to get a single value. Example - we want to sum number greater than 20.

```
1. _.reduce([10, 15, 20, 35, 30, 45], function (accumulator, value) {
2. if (value > 20)
3. return accumulator + value;
4. else
```

```
5. return 0;
6. }); // sum is 110
```

**Case 8:** String template is very common for substitution of data.

**Naïve method**

```
1. var name='Andriana';
2. var interest='Programming';
3. varstr="Hello, "+name+". Your interest is "+interest;
4. console.log(str); // Hello, Andriana. Your interest is Programming
```

**Loadash.js**

```
1. vartpl = _.template('Hello, <%= name %>. Your interest is <%= interest%>');

2. tpl({
3. name: 'Andriana', interest: 'Programming'});
4. console.log(tpl);//Hello, Andriana. Your interest is Programming
```

# ChakraCore JavaScript Engine By Microsoft

JavaScript is a language of Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

In this part, I am going to explore new JavaScript Engine Chakra by Microsoft.

**Introduction**

ChakraCore is a new JavaScript engine developed by Microsoft. It doesn't support IE9. The cool thing about ChakraCore is that it is OSS (Open Source Software). ChakraCore is the core part of the Chakra JavaScript engine and source code is available at GitHub.

It is part of Microsoft Edge browser and available for any developer as an open source project.

**Alternate to V8 engine**

Currently NodeJS uses V8 engine for compiling JavaScript source code to machine code instead of interpreting in real time. Now with ChakraCore Microsoft is providing an alternative. Microsoft submitted a pull request to Node.js to work with ChakraCore.

Microsoft implemented a V8 API shim (aka ChakraShim) which takes advantage of ChakraCore runtime hosting APIs (JSRT). Many developers supported this as it is going to be big and awesome stuff. Node.js also welcomed this step that Microsoft coming closer to Node.js. We know Microsoft is an early adopter of Node.js.

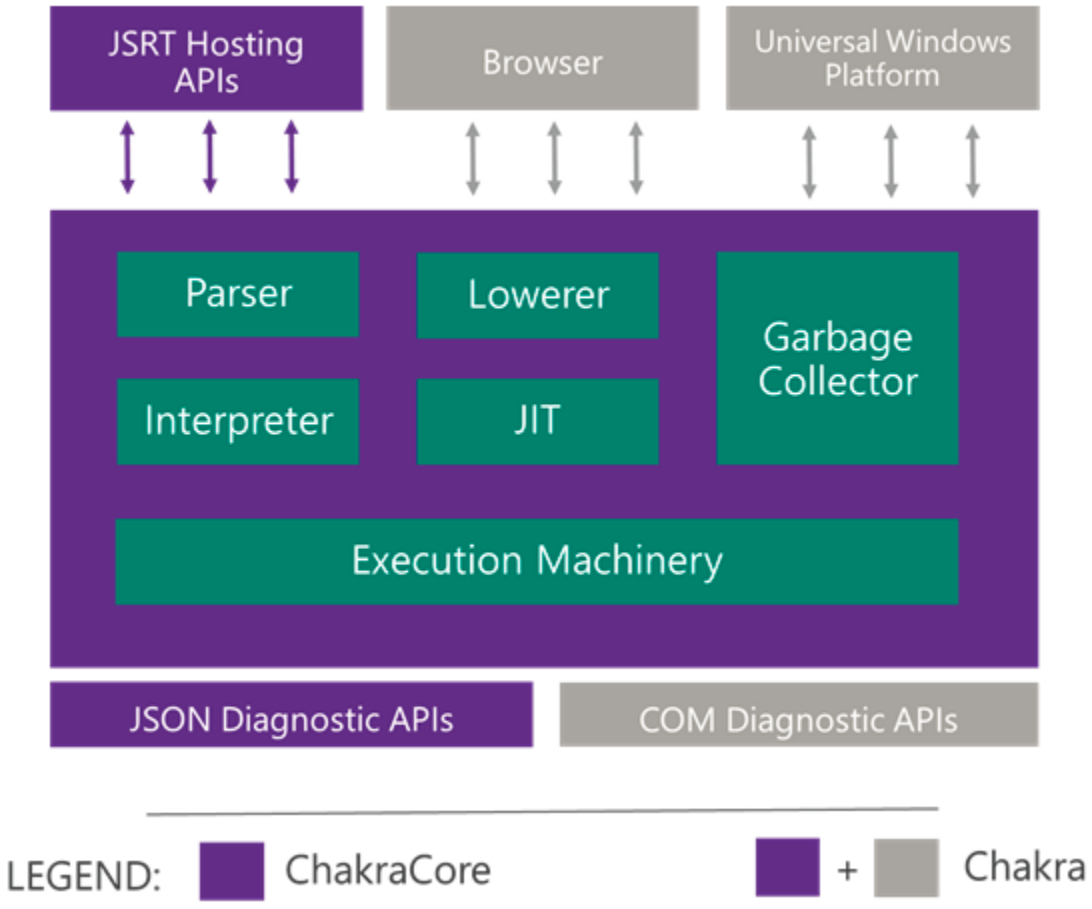
"This pull request enables Node.js to optionally use the ChakraCore JavaScript engine," Guarav Seth

**Roadmap**

The ChakraCore roadmap shows that they are planning to port to Linux, targeting x64 Ubuntu. However, it doesn't talk about MAC OSX platform. To read more detail please goto URL: https://github.com/Microsoft/ChakraCore/wiki/Roadmap

**JSRT (JavaScript Runtime) APIs**

JSRT APIs provide a way to embed ChakraCore into applications. It is part of the Core project. Refer architecture diagram.



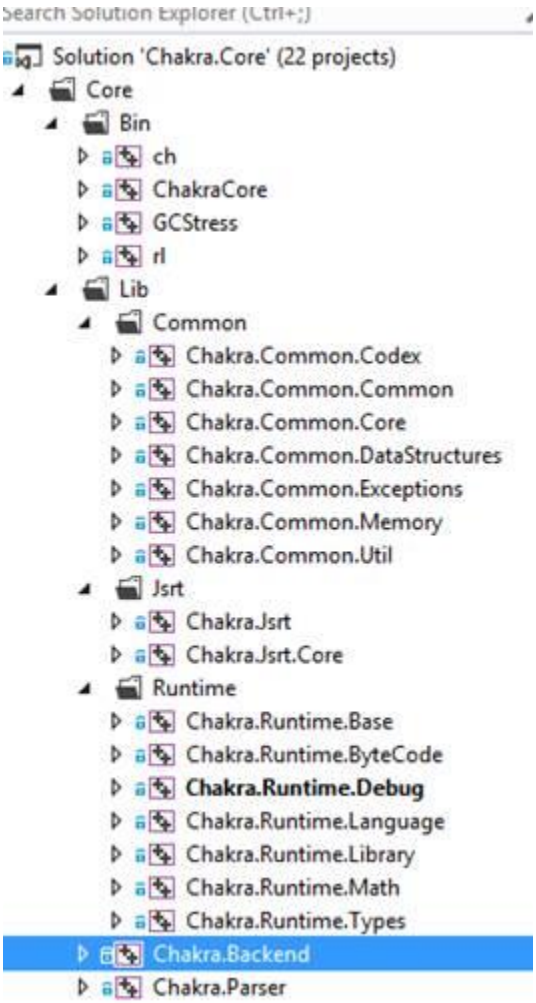

JavaScript engine using the JSRT APIs depends on two key concepts: runtime and execution contexts. It also allows embedding Chakra outside of client-only scenarios also,

- A runtime represents a complete JavaScript execution environment
- Execution contexts are tied to a particular runtime and execute code within that runtime.
- High performance JavaScript engine
- Support Just-in-time (JIT) compilation of JavaScript
- Supports JSRT APIs which allows you to embed in your applications

**Building ChakraCore**

Since it is OSS and available at GitHub you could download source code from GitHub and build using Visual Studio:

- Clone
- Open Build\Chakra.Core.sln in Visual Studio




- Build Solution

**Embedding ChakraCore**

It can be embedded via JSRT APIs and Azure DocumentDB was one of the other programs which uses Chakra to provide JavaScript programmability. DocumentDB is NoSQL document oriented database which is built to support JSON & JavaScript. Therefore, it's good for DocumentDB to leverage Chakra to process JavaScript.

To read more details.

# JavaScript Web App Performance Best Practices

JavaScript is a language of the Web. This part will talk about my observations, learnt during my decade of software development experience with JavaScript.

Our applications should be high performance for a good user experience. Users do not prefer working on slow web applications. Our objective is to make our web apps faster. This part has two sections,

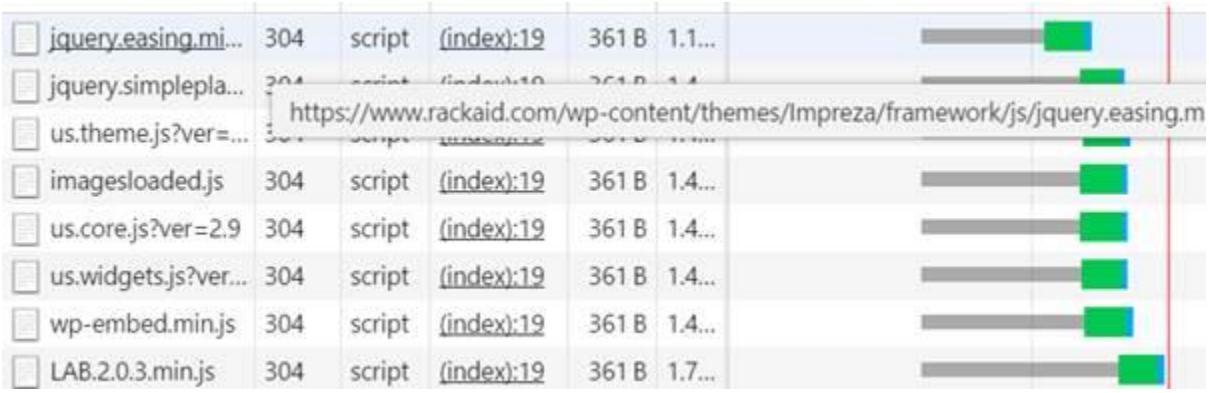
**Section 1:** Best practices about handling JavaScript files to make our web app fast.

**Section 2:** Measure JavaScript performance using APIs.

**Section 1: Best practices**

**Put JavaScript at the bottom?**

Most browsers make multiple connections to a server at the same time. Press F12 (developer tool) and goto Network tab and observe connection stream. Example,




As soon as the browser encounters script tag, browser will not start any other download. JavaScript blocks parallel download. This is suggested by YSlow (a tool by Yahoo). In current times, it is a debatable topic because many developers do not agree.

I think if your web app is more content focused and less dependent on JavaScript then place JavaScript at bottom. To the contrary, if JavaScript as written in your webapp is of priority then put it at top.

**Minify JavaScript**

To compress / minify your JavaScript code is best practice before production release. The smaller the file, better would be download speed and your page load time will be better too. There are various online tools available to minify your JavaScript code.

You can paste JavaScript code and it will compress it to good extent.

Example- my script was compressed by 33%,


There are many command line utilities available also, which you could use, example Grunt Minify files with UglifyJS.

To talk about grunt task runner minify recipe is beyond the scope of this part.

**Combine JavaScript files into one**

During development, we leverage multiple JavaScript files & libraries, e.g.,

```
1. <script src="navbar.js"></script>
2. <script src="component.js"></script>
3. <script src="page.js"></script>
4. <script src="framework.js"></script>
5. <script src="script.js"></script>
```

It is recommended to combine JavaScript files. The advantage is to reduce number of requests at server and faster
webapp. The lesser number of <script> tags are better in production code.

You can leverage grunt concat https://github.com/gruntjs/grunt-contrib-concat to do so

In practice, we combine and minify Javascript files for better performance

**Remove duplicate Javascript files**

Sometimes you may notice developers add duplicate JavaScript files with different versions. One or multiple versions may not be required, but it went unnoticeable. Hence, client is making wasteful HTTP request to the server, e.g.,

```
1. <script type="text/JavaScript" src="library_1.0.17.js"></script>
2. <script type="text/JavaScript" src="library_1.1.0.js"></script>
```

Hence, in the above scenario one library is redundant and adding payload to the network, which slows down the performance of web application. Identify such cases and remove extra files.

**Reduce Cookie Size**

We know HTTP cookies are used to keep information about the user. It is recommended to keep size of cookie small. The reason is cookie gets transferred in each server request and response. In case the cookie is heavy, it increases the size of HTTP request and consumes more time. Be mindful of setting the cookie size and set expiry accordingly.

**Note**: considering security threats like CSRF (in top 10 threat) I prefer session cookies to persistent cookies.

**Section 2: Performance APIs**

You can use Performance interface to access certain functions for measuring performance of web pages and web applications.

**now() method:** browser returns the timestamp, which is DOMHighResTimeStamp. It is a double to store a time value. The unit is milliseconds.

*var currentTime = window.performance.now(); //ex- 4135.333*

**mark() method:** browser creates a timestamp, which is DOMHighResTimeStamp in buffer. The advantage is that we can name a timestamp in memory.

```
1. window.performance.mark('script_started_loading');
2. // script goes here
3. var req = new XMLHttpRequest();
4. req.open('GET', '/data/v1/geoLocation', true); //replace your URL here
5. req.onload = function(e) {
6. do_something(e.responseText); //some anonymous function
7. }
8. req.send();
9. window.performance.mark('script_fully_loaded');
10.// we will mark difference between 'script_fully_loaded' & 'script_started_l
   oading'
```

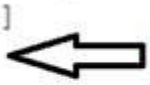
**measure() method:** it calculates elapsed time between two marks. So to calculate difference between two marks you can define code like below:

window.performance.mark('measure1','script_started_loading','script_fully_load
ed');

**getEntriesByName() method:** you can read data created in above using this method

```
performance.getEntriesByName('script_started_loading')
[▼ PerformanceMark                ]
    duration: 0
    entryType: "mark"
    name: "script_started_loading"
    startTime: 595350.535
  ▶ __proto__: PerformanceMark
```

```
performance.getEntriesByName('measure')
[▼ PerformanceMeasure             ]
    duration: 89646.9900000001
    entryType: "measure"
    name: "measure"
    startTime: 595350.535
  ▶ __proto__: PerformanceMeasure
```



# ECMAScript Promises

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

**Introduction**
A Promise is an asynchronous operation that has not completed now, but will complete in the future. ES6 has adopted promises implementation as native APIs. Promise gives us guarantee to return the value in future. Promises are not like events where you need to bind the event with a particular method.

**Creating a promise**
The way to create a Promise is by using "new Promise" constructor, which accepts two callback functions as parameters. The first typically named resolve is a function to call with the future value when promise is ready. The second typically named reject is a function to reject the Promise if it cannot resolve the future value.

**Syntax**

As per MDN, below is the syntax:

*new Promise( /* executor */ function(resolve, reject) { ... } );*

**Executor:** A function that will be passed to other functions via the arguments resolve and reject.

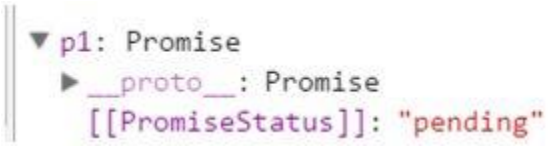
**Sample code**

```
1. var p1 = new Promise(function(resolve, reject)
2. {
3.     if ( /* condition */ )
4.     {
5.         resolve( /* value */ ); // promise fulfilled
6.     }
```

```
7.    else
8.    {
9.       reject( /* reason */ ); // mention reason for rejection
10.   }
11.});
```

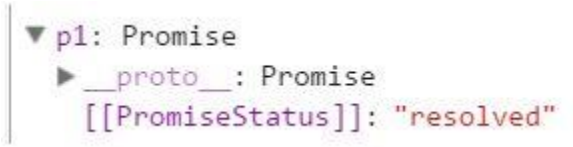
Let us see what happens at Browser after we trigger above code.

- When code executes “new Promise”, it starts from below state

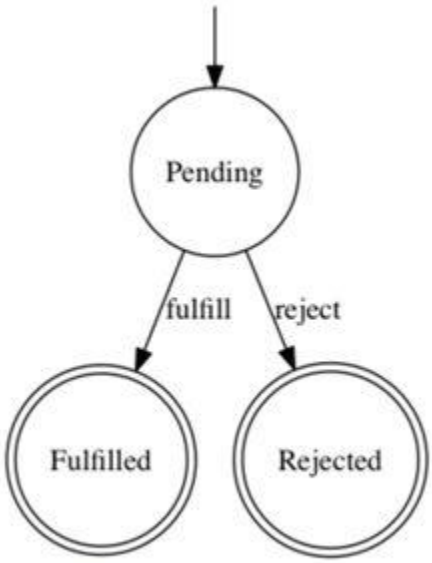



The first state of Promise is “pending”.

- After Promise is fulfilled then it changes to:




The second state of Promise is “fulfilled” or a promise is “rejected” if it’s not fulfilled.



Possible states of a JavaScript promise.

**Summary of Promise states:**

- **pending:** Initial state, not fulfilled or rejected.
- **fulfilled:** Meaning that the operation completed successfully.
- **rejected:** Meaning that the operation failed.

**Prototype promise methods**

After a Promise is created, we need to pass around value of this Promise. In order to consume a Promise value we attach a handler using via below methods.

Two methods of Promise are associated with constructor prototype Promise.prototype these are:

**then() method**

It returns a promise with two arguments, i.e., onFulfilled, on Rejected

- onFulfilled: a callback function called when promise is fulfilled.

- onRejected: a callback function is called when promise is rejected.

Example –it will print “Fulfilled!” because we’ve set *flag = true,*

```
var flag = true;
var p1 = new Promise(function(resolve, reject)
{
    if (flag)
    {
        resolve("Fulfilled!");
    }
    else
    {
        reject("Rejected!");
    }
});
p1.then(function(value)
{
    console.log(value); // Fulfilled!
}, function(reason)
{
    console.log(reason); // Rejected!
});
```

**catch() method**

It returns a promise and deals with rejected cases only. If you pass null or undefined in .then() method first argument it .then() method behaves like .catch() method.

Example- Below code will print “Rejected!”

```
var flag = false;
var p1 = new Promise(function(resolve, reject)
{
    if (flag)
```

```
    {
        resolve("Fulfilled!");
    }
    else
    {
        reject("Rejected!");
    }
});
p1.then(function(value)
{
    console.log(value); // Fulfilled!
}).catch(function(reason)
{
    console.log(reason); // Rejected!
});
```

Example - Chaining promises via .then() method.

As .then(), .catch() methods return Promise so we could chain these [please refer Voice of a Developer JavaScript Chaining part 16 to understand Chaining].

Example- chaining of .then() method and incrementing counter.

```
var counter = 0;
var p1 = new Promise(function(resolve, reject)
{
    resolve(counter);
});
p1.then((value) =>
{
    console.log('Counter: ' + ++counter); // 1
}).then((value) =>
{
    console.log('Counter: ' + ++counter); // 2
}).then((value) =>
```

```
13.{
14.    console.log('Counter: ' + ++counter); // 3
15.}).catch(function(reason)
16.{
17.    console.log(reason);
18.});
```

**Output**

```
Counter: 1
Counter: 2
Counter: 3
```

**Promise methods**

Promise.all() –When you are working with multiple promises, this function is really helpful. Once all Promises are resolved or rejected, it returns Promises.

```
1.  var p1 = new Promise(function(resolve, reject)
2.  {
3.     var counter = 0;
4.     resolve(++counter);
5.  });
6.  var p2 = new Promise(function(resolve, reject)
7.  {
8.     resolve("counter 2");
9.  });
10.var p3 = new Promise(function(resolve, reject)
11.{
12.    setTimeout(resolve("Promise 3"), 5000); //5000 milisecond = 5 sec
13.});
14.Promise.all([p1, p2, p3]).then(function(values)
15.{
16.    console.log(values); // Output: [1, "counter 2", "Promise 3"]
```

```
17.}).catch((val) =>
18.{
19.    console.log(val); // return “rejected”, if any promise fails
20.});
```

Once all Promisesp1, p2, p3 are fulfilled.then() method is called. The above code .then()will execute after 5 seconds when all Promises are resolved (after setTimeout).

If any promise returns reject, then it will return “rejected”.

Promise.race() – It returns a Promise which resolve or reject first. It accepts an iterable of Promises and works like OR condition.

```
1.  var p1 = new Promise(function(resolve, reject)
2.  {
3.     setTimeout(resolve, 1500, "resolved - will not get printed");
4.  });
5.  var p2 = new Promise(function(resolve, reject)
6.  {
7.     setTimeout(reject, 100, "rejected - printed");
8.  });
9.  Promise.race([p1, p2]).then(function(value)
10.{
11.    console.log(value); // not get printed
12.}, function(reason)
13.{
14.    console.log(reason); // rejected - printed
15.    // p6 is faster, so it rejects
16.});
```

**Load XMLHttpRequestfiles**

The use of Promises is by using XMLHttpRequestAPI asynchronously. It loads file in background, example: it will load two JSON files via XHR requests:

**automobile.json**

```
{
   "automobile": [
   {
      "vehicle": "car",
      "engine": "1200cc"
   },
   {
      "vehicle": "bike",
      "engine": "200cc"
   },
   {
      "vehicle": "jeep",
      "engine": "2000cc"
   }]
}
```

**employee.json**

```
{
   "employees": [
   {
      "firstName": "John",
      "lastName": "Doe"
   },
   {
      "firstName": "Anna",
      "lastName": "Smith"
   },
   {
      "firstName": "Peter",
      "lastName": "Jones"
   }]
```

```
15.}
```

**Script.js**

```
1. varary = ['http://localhost/automobile.json', 'http://localhost/employee.jso
   n']
2. var p1 = new Promise(function(resolve, reject)
3. {
4.    letsrcurl = getJSON(ary[0], resolve);
5. });
6. var p2 = new Promise(function(resolve, reject)
7. {
8.    letjson = getJSON(ary[1], resolve);
9. });
10.functiongetJSON(json, resolve)
11.{
12.   varxmlhttp = new XMLHttpRequest(json);
13.   xmlhttp.open("GET", json);
14.   xmlhttp.send();
15.   xmlhttp.onload = function()
16.   {
17.      if (xmlhttp.status === 200)
18.      {
19.         resolve(xmlhttp.responseText);
20.      }
21.   }
22.};
23.Promise.all([p1, p2]).then((val) =>
24.{
25.   document.getElementById('div2').innerHTML = val;
26.});
```

**Output:** it will load data of ‘val’ into ‘div2’ of HTML.

{"automobile":[ {"vehicle":"car", "engine":"1200cc"}, {"vehicle":"bike", "engine":"200cc"}, {"vehicle":"jeep", "engine":"2000cc"} ]},{"employees":[ {"firstName":"John", "lastName":"Doe"}, {"firstName":"Anna", "lastName":"Smith"}, {"firstName":"Peter", "lastName":"Jones"} ]}

## Promises Advantages

- It gives us ability to write async code synchronously.
- You can handle via handler whether its resolved or rejected
- Solve problem of code pyramids, ex-

```
step1(function(value1)
{
    step2(value1, function(value2)
    {
        step3(value2, function(value3)
        {
            step4(value3, function(value4)
            {
                // Do something with value4
            });
        });
    });
});
```

This is solved and simplified via Promise prototypal methods & chaining.

```
var p1 = new Promise().resolve('resolve');
p1.then((value) =>
{
    console.log(value);
}).then((value) =>
{
    console.log(value);
}).then((value) =>
{
    console.log(value);
```

```
}).then((value) =>
{
    console.log(value);
});
```

**Libraries providing Promises**

Promises are supported via different libraries Q, AngularJS, BlueBird etc. There are many libraries providing promises as an abstraction on top of JavaScript Native Promises API, e.g.

**Q.js**

```
var functionA = function()
{
    return "ret A";
};
var functionB = function()
{
    return "ret B";
};
var promise = Q.all([Q.fcall(functionA), Q.fcall(functionB)]);
promise.spread(finalStep);
```

**AngularJS**

```
// Simple GET request example:
$http(
{
    method: 'GET',
    url: '/someUrl'
}).then(function successCallback(response)
{
    // this callback will be called asynchronously
    // when the response is available
}, function errorCallback(response)
```

```
11.{
12.    // called asynchronously if an error occurs
13.    // or server returns response with an error status.
14.});
```

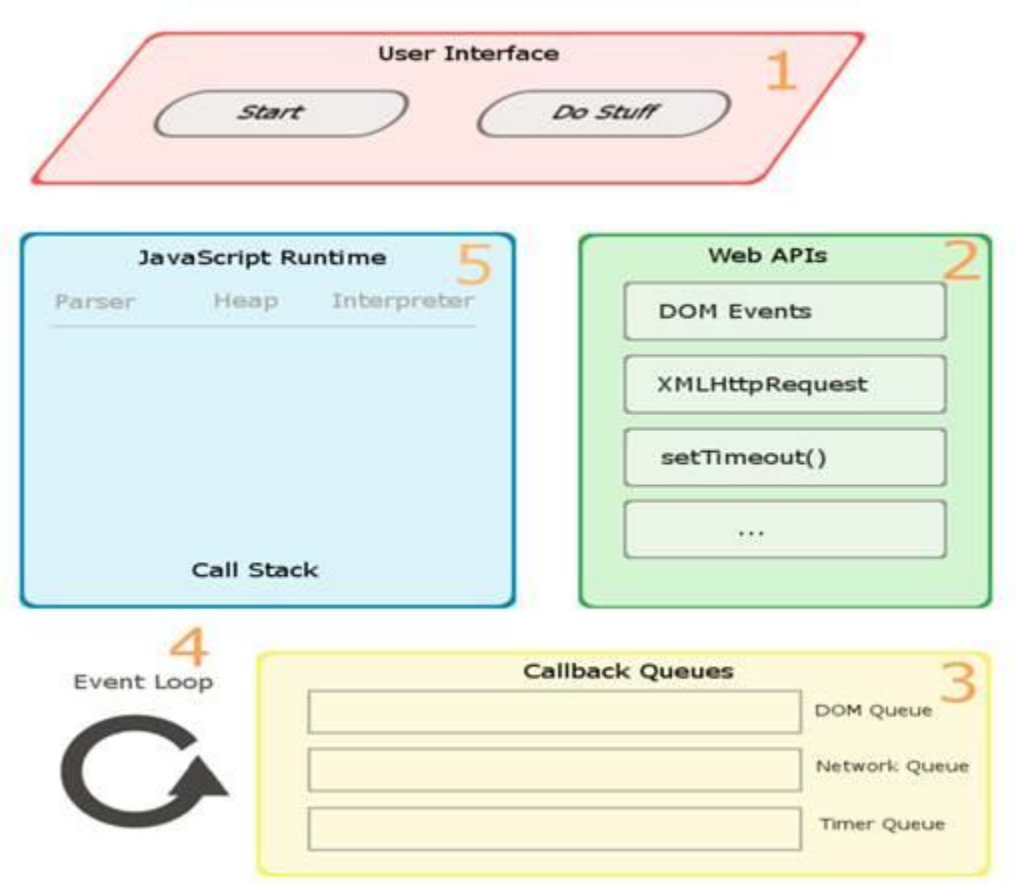
It generates a HTTP request and returns a response.

# Browser Runtime

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

### Browser Runtime

When we think of Browser, generally we think of DOM elements, Events, Images, JavaScript executing at a client-side. Technically, it is more than that. A browser constitutes of various components, please refer diagram from Philip Robert:




I will explain each component in detail now:

### User Interface

User submit a form and clicks a button.

**Web APIs**

This is multi-threaded area of browser that allows many events to trigger at once. JavaScript can access these via WINDOW object on page load. Other examples are DOM document, XMLHttpRequest, setTimeout().

setTimeout is an API provided to us by the browser, it doesn't live in the V8 source. It is not a guaranteed time of execution. It is the minimum time of execution.

**Event Queue**

Any of the web APIs pushes the callback on to the event queue when it is done. It is also called task queue.

**Event loop**

The event loop looks at call stack and task queue and decide which callback to push onto call stack.

**Call Stack**

JavaScript is a single threaded runtime, it has a single call stack and can execute only one piece of code at a time. If there is a piece of code taking long time to execute then it blocks UI. This problem is known as blocking. Every function invocation including event callbacks creates a new stack frame. These stack frames are pushed and popped from the top of the call stack (LIFO concept, last in first out). It executes top most stack frame and return that stack frame is popped out. The call stack is part of JavaScript runtime.

Blocking- means code is slow and therefore preventing execution of statements after it. For example, there is complex FOR loop, looping around 10,000 times.

Until FOR loop is over, next statements will not be executed. The browser cannot render anything until there is some process in the call stack. If you remember, I shared in part 31 (http://www.c-sharpcorner.com/article/voice-of-a-developer-javascript-webapp-performance-best-pra/) that Yahoo YSlow proposed script tag shall be placed at the bottom of webpage.

**Q: How shall we solve this problem of blocking?**

Ans: The solution is asynchronous callbacks. We know that most browsers use single thread for UI and JavaScript, which is blocked by synchronous calls. We know we can run asynchronous requests using XMLHttpRequestAPI too to solve this problem.

**There are other various components inside Call Stack, these are,**

**Parser -** It is the process of analyzing a string of symbols, like part of speech. This takes a program and constructs abstract syntax tree.

**Heap -** we know heap is a part of memory system. In most programming languages, generally there are two type of memories allocation i.e., stack and heap.

| Stack | Heap |
|---|---|
| It is used for static memory allocation | It is used for dynamic memory allocation |
| Stack size is limited dependent various factors like browser type, version | Heap is dependent on size of Virtual memory |
| Fast to access | Slow to access than stack |

There is no separation of memory into stack/heap in Javascript.

You can also find out Heap size available using Memory management API. Example- performance.memory in Chrome console [only available with Chrome as of now]

```
performance.memory
MemoryInfo {totalJSHeapSize: 33100000, usedJSHeapSize: 26000000, jsHeapSizeLimit: 793000000}
```

### Interpreter

It translates high-level instructions into a form where it can execute. However a compiler translates directly to machine language.

We know how to calculate size of variables (refer part 1 http://www.c-sharpcorner.com/article/voice-of-a-developer-javascript-data-types/), but is there a way to calculate size of call stack available?

### Calculate Size of Stack

### Step 1 - Goto console of your browser and run below script

```
1. var size = 0;
2.
3. function inc()
4. {
5.     size++;
6.     inc();
7. }
8. inc();
```

### Output from Chrome

```
> var size=0;
  function inc() {
      size++;
      inc();
  }
  inc();
✖ ▶ Uncaught RangeError: Maximum call stack size exceeded(…)
```

### Output from Firefox

```
InternalError: too much recursion
    var size=0;
    function inc() {
        size++;
        inc();
    }
    inc();
```

### Output from IE

```
var size=0;
function inc() {
    size++;
    inc();
}
inc();
Out of stack space
```

### Step 2

Now you have seen that each browser is complaining about call stack size limit crossed. It's time to know the unknown limit by printing variable 'size' in each browser and we can use navigator object to determine details of the browser too.

### Note

Below output, "Stack size" will depend upon your browser and its version.
*console.log ("Stack size: "+ size + " Browser: "+ navigator.appVersion)*

### Output from Chrome

```
> console.log("Stack size: "+ size + " Browser: "+ navigator.appVersion)
  Stack size: 41847 Browser: 5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36
```

### Output from Firefox

```
> console.log("Stack size: "+ size + " Browser: "+ navigator.appVersion)
  Stack size: 8723 Browser: 5.0 (Windows)
```

**Output from IE**

```
console.log("Stack size: "+ size + " Browser: "+ navigator.appVersion)
undefined
Stack size: 7187 Browser: 5.0 (Windows NT 6.3; WOW64; Trident/7.0; .NET4.0E; .NET4.0C; Tablet PC 2.0; .NET CLR 3.5.30729; .NET CLR 2.0.50727; .NET CLR 3.0.30729; rv:11.0)
```

**Execution JavaScript Runtime**

We talked above about call stack and now we can see how a program is executed internally. Example below script executes as following,

```
1. var firstFunc = function()
2. {
3.     console.log("I'm first!");
4. };
5. var secondFunc = function()
6. {
7.     setTimeout(firstFunc, 3000);
8.     console.log("I'm second!");
9. };
10.secondFunc();
```

**Sequence of execution**

- Initially call stack is empty.
- Then it push two lines of code in call stack.


- setTimeout moves to Web API area of browser and has a callback function firstFunc() which will be executed after 3 seconds.


- In the meantime secondFunc() statement gets into processing at Call Stack,

- Next piece of code is console.log("I'm second!") to pop out from stack and in meantime firstFunc() moved into the event queue.


- Event loop picks firstFunc() from task queue and push into Call Stack for execution.


- firstFunc() has console.log("I'm first!") which will then pushed into stack & executed.

**Output**

I'm second!
I'm first!

# JavaScript Unit Test

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

**Unit tests**

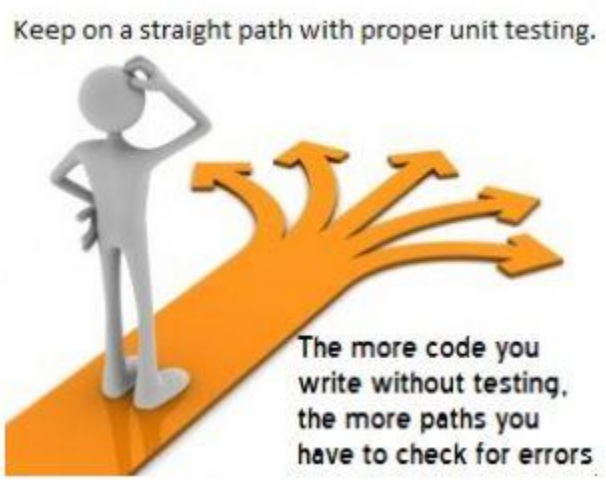
Unit testing is a software development process in which the smallest parts of the Application, called units, are tested. It can be done manually or the process could be automated. Unit Testing is a level of software testing where the individual units/ components of the software are tested. The purpose is to validate that each unit of the software performs as designed.

**Advantages of Unit tests**

I see a lot of advantages in Unit tests, I have listed all of them as per my experience,

- Allows you to make big changes to the code quickly.
- Improves the design of the code, especially with Test-Driven Development.
- Makes it easier to change and refactor code.
- Saves your development time.

Unit tests and TDD go hand-in-hand. The concept of TDD is to first write a failing test, then we write enough code to pass the test and refactor the code. Generally, as a developer, we write fast code that works well, but we don’t think to write Unit tests.




**Source** http://www.juancarbonel.com/

Unit tests are the safety nets and test whether the code is working or not by just running it.

List of all the unit test frameworks are available here.

**Few popular frameworks are:**

- Jasmine
- Mocha
- Chai

**Other concepts**

**Test runner**

A CLI based tool requires you to run your tests. As developers, we shall love command line interface and integrate our scripts with it. In an automation, it is of great help and we can integrate the test runner tool with the various environments too. There are many test runner softwares available. For e.g. Karma, Grunt, Gulp, Nrunner.

**Assertion**

It means to validate the condition or the output. We can determine the output; whether a test makes your code fail or pass depends on the assert condition. Different test suites/automation tools syntax can vary but the purpose is the validation, e.g. Assert.IsTrue (true);

**Pre-requisites**

You need to install NodeJS to enable the unit tests in JavaScript. You can refer to the part given below, by clicking it.

- Jump Start Node.js by Sanjay Kumar.

Let us do Unit testing now. To do that, we will install some node packages, i.e., Jasmine and Karma.

**There are two ways to install things in NodeJS**

- **Global**

  It installs the node modules in the global space. Hence, you do not need to install it every time for different projects. This is done using –g parameter; while installing. If you are creating your Application to test whether it's fine or not, you should have in dev dependencies, which is the next option.

  [**Note** for this part, I'll show –g option for the installation]

- **Save local**

  It installs the node modules in the project directory. Hence, it is a part of the developer dependencies in Package.json [configuration file of NodeJS dependencies] project. If you're in a hurry to create package.json; then there are various ways you can download a sample using curl command.

  **Sample** package.json

*curl -O*

```
$ curl -O https://gist.githubusercontent.com/bbraithwaite/7432f64ea45e028eb23c/r
aw/2a6e23cee09e7737c30cc790393a9b273070fbcb/package.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   306  100   306    0     0    275      0  0:00:01  0:00:01 --:--:--   306
```

package.json

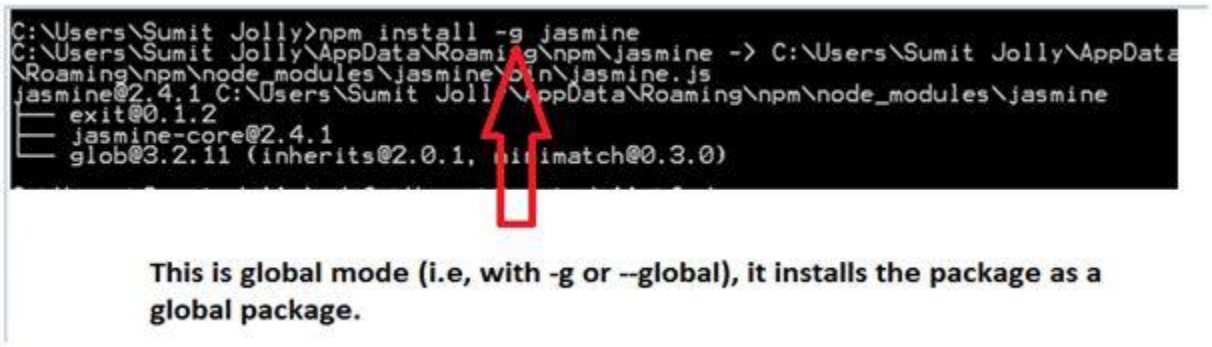
```
{
  "name": "karma-seed",
  "version": "1.0.0",
  "description": "",
  "scripts": {
    "test": "./node_modules/karma/bin/karma start karma.conf.js"
  },
  "devDependencies": {
    "jasmine-core": "^2.2.0",
    "karma": "^0.12.31",
    "karma-chrome-launcher": "^0.1.7",
    "karma-jasmine": "^0.3.5"
  }
}
```

- **Jasmine**

  Jasmine is a test driven development framework to test JavaScript code. For more details, you can see here.

  Now, we will install Jasmine framework, using the command shown below,

  *npm install –g jasmine*




- **Karma**

  Karma is a spectacular test runner for JavaScript. It allows you to execute JavaScript code in the multiple real Browsers. I am emphasizing real Browsers, because it does not run Fake/stubs, but executes tests on the real Browsers. For more details, you can find which Browser, it supports.

**Installing Karma and plugins**

Karma install: fire below command to install karma,

*npm install –g karma*

**Karma-Jasmine**

Karma runner supports many different test frameworks like Jasmine, Chai & Mocha etc. We will install karma-jasmine module,

*npm install -g karma-jasmine*

**Chrome launcher**

Although we can leverage any supported Browser by Karma, I will show the tests via most popular Browser Chrome. Hence, install Chrome-launcher,

*npm install –g karma-chrome-launcher*

**Karma CLI**

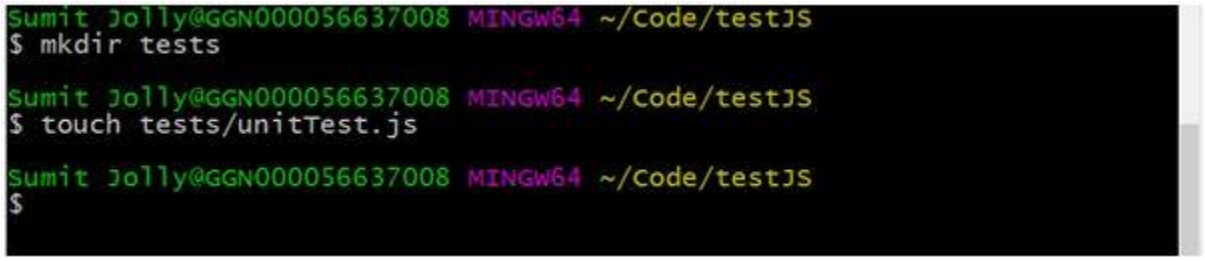
It is a nice interface; from command line to leverage Karma, install,

*npm install –g karma-cli*

**Write Unit Test Program**

- Create a directory, md testJS


- Create tests folder and underneath unitTests.js,




- Open file unitTests.js in your favorite editor and paste the code, shown below:

```
1. describe("UnitTests", function()
2. {
3.     it("First Unit test", function()
4.      {
```

```
5.        expect(2).toEqual(2);
6.    });
7. });
```

Let us dive deep into the various components of this.

**describe**

A test suite begins with describing what accepts the title of the suite and a function. This function is going to be tested.

**it**

Specs are defined, using it. You can write any number of test specifications. A test can be true or false and it depends upon the condition.

**expect**

It takes a value and matches against the expected value.

- **Configure the test runner**

To create this file, quickly download sample karma.conf.js from the location, given below:

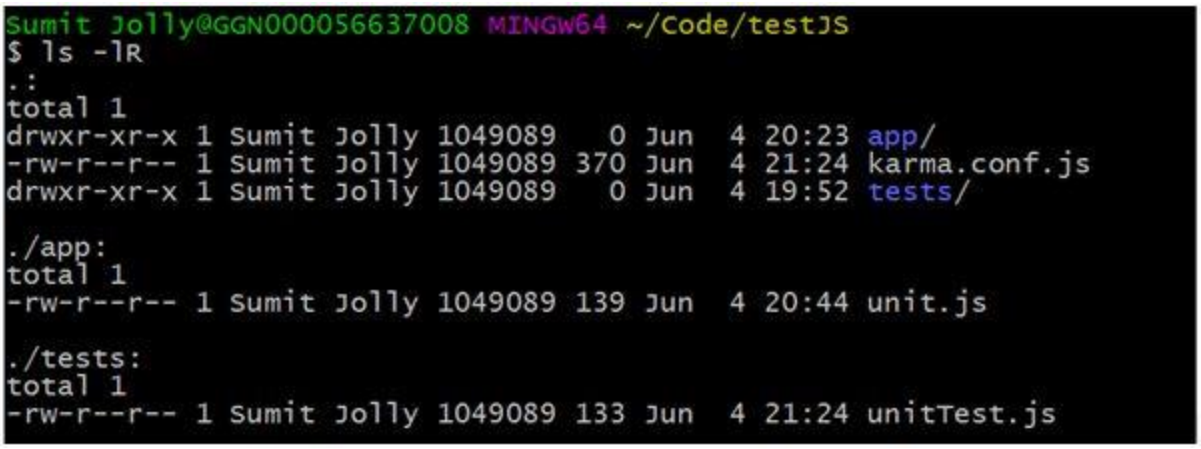
*curl -O*

Review karma.conf.js for now, we'll modify it later.

```
1. module.exports = function(config)
2. {
3.    config.set(
4.      {
5.        basePath: '',
6.        frameworks: ['jasmine'],
```

```
        files: [
          'build/js/**/*.js',
          'build/tests/**/test_*.js'
        ],
        exclude: [],
        preprocessors: {},
        reporters: ['dots'],
        port: 9876,
        colors: true,
        logLevel: config.LOG_INFO,
        autoWatch: true,
        browsers: ['PhantomJS'],
        singleRun: true
    });
};
```

- Create the folder app and add code file unit.js. The folder structure may look like,

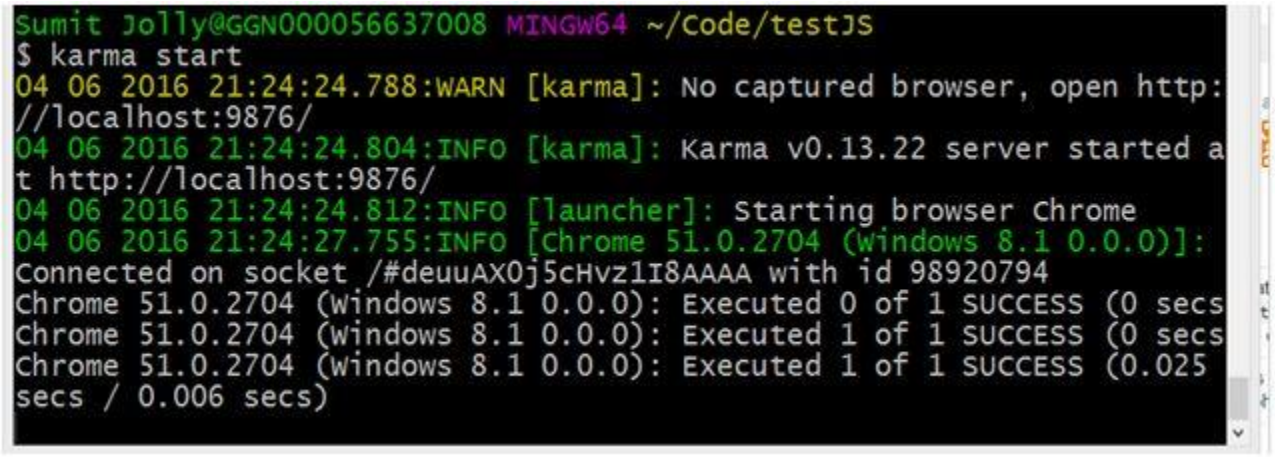



- Modify some key-value pairs, according to your code:

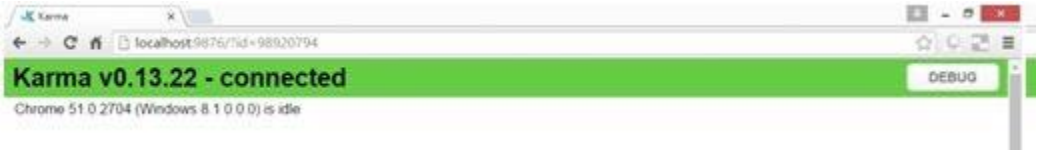
```
1. browsers: ['Chrome'],
2. files: [
3. 'app/*.js',
```

```
4.  'tests/*.js'
5.  ],
```

- Run the basic test using karma. Start and it will also open Chrome Browser and run test in unitTest.js,




Launch Chrome too.




**Info**

You may get an error that Chrome is not properly configured, so please update the path accordingly,

On command prompt, you could see - it runs your test and print,

*Executed 0 of 1 SUCCESS*
*Executed 1 of 1 SUCCESS*

This happens because our expect statement results into true.

**TDD principles**

I am sure that you are familiar with TDD. In TDD, we promote first to write test and then code. Above example was to quickly demonstrate you passing the test. Now, we will ensure the principles of TDD.

- First write the fail test (RED)
- Write the minimum code so the test can pass (GREEN)
- Improve the code (Refactor)
- Revisit unitTest.js and we want to create a global library, where we can perform simple mathematical operations like sum, product. I assumed the global object and two functions with it:

```
describe("UnitTests", function()
        {
    var g = global;
    it("Sum test", function()
      {
       expect(g.sum(2, 3)).toEqual(5);
    });
    it("Product test", function()
      {
       expect(g.product(2, 3)).toEqual(6);
    });
});
```

**Note**

Follow TDD principle, I will first fail my tests.

```
Sumit Jolly@GGN000056637008 MINGW64 ~/Code/testJS
$ karma start
04 06 2016 21:42:39.652:WARN [karma]: No captured browser, open http:
//localhost:9876/
04 06 2016 21:42:39.665:INFO [karma]: Karma v0.13.22 server started a
t http://localhost:9876/
04 06 2016 21:42:39.672:INFO [launcher]: Starting browser Chrome
04 06 2016 21:42:42.109:INFO [Chrome 51.0.2704 (Windows 8.1 0.0.0)]:
Connected on socket /#BHFcEsJqlp4JejCcAAAA with id 48200067
Chrome 51.0.2704 (Windows 8.1 0.0.0): Executed 0 of 1 SUCCESS (0 secs
Chrome 51.0.2704 (Windows 8.1 0.0.0) UnitTests encountered a declarat
ion exception FAILED
        ReferenceError: global is not defined
            at Suite.<anonymous> (C:/Users/Sumit Jolly/Code/testJS/te
sts/unitTest.js:2:13)
            at C:/Users/Sumit Jolly/Code/testJS/tests/unitTest.js:1:1
Chrome 51.0.2704 (Windows 8.1 0.0.0): Executed 1 of 1 (1 FAILED) (0 s
Chrome 51.0.2704 (Windows 8.1 0.0.0): Executed 1 of 1 (1 FAILED) ERRO
R (0.012 secs / 0.003 secs)
```

At terminal, it says, “global is not defined”, that means my test is in RED.

- Next step to write minimum code is to turn it into GREEN,

  Update the code in unit.js, as shown below:

```
var global =
    {
    sum: function(a, b)
    {
      return a + b;
    },
    product: function(a, b)
     {
      return a * b;
    }
};
```

- Again, run karma start command,

```
04 06 2016 21:51:06.398:INFO [watcher]: Changed file "C:/Users/Sumit
Jolly/Code/testJS/tests/unitTest.js".
Chrome 51.0.2704 (Windows 8.1 0.0.0): Executed 0 of 2 SUCCESS (0 secs
Chrome 51.0.2704 (Windows 8.1 0.0.0): Executed 1 of 2 SUCCESS (0 secs
Chrome 51.0.2704 (Windows 8.1 0.0.0): Executed 2 of 2 SUCCESS (0 secs
Chrome 51.0.2704 (Windows 8.1 0.0.0): Executed 2 of 2 SUCCESS (0.012
secs / 0.005 secs)
```

Nice, we can see our both tests, that passed.

- Refactor code by using the “use strict” and replacing var with let keyword inside test will ensure:

unit.js app

```
1  "use strict";
2  var global = {
3      sum: function (a, b) {
4          return a + b;
5      },
6      product: function (a, b) {
7          return a * b;
8      }
9  };
```

unitTest.js

```
1  describe("UnitTests", function () {
2      let g = global;
3      it("Sum test", function () {
4          expect(g.sum(2,3)).toEqual(5);
5      });
6
7       it("Product test", function () {
8          expect(g.product(2,3)).toEqual(6);
9      });
10 });
```

I committed this code at GitHub and from here you can download or clone it.

## Decoupling Your Application

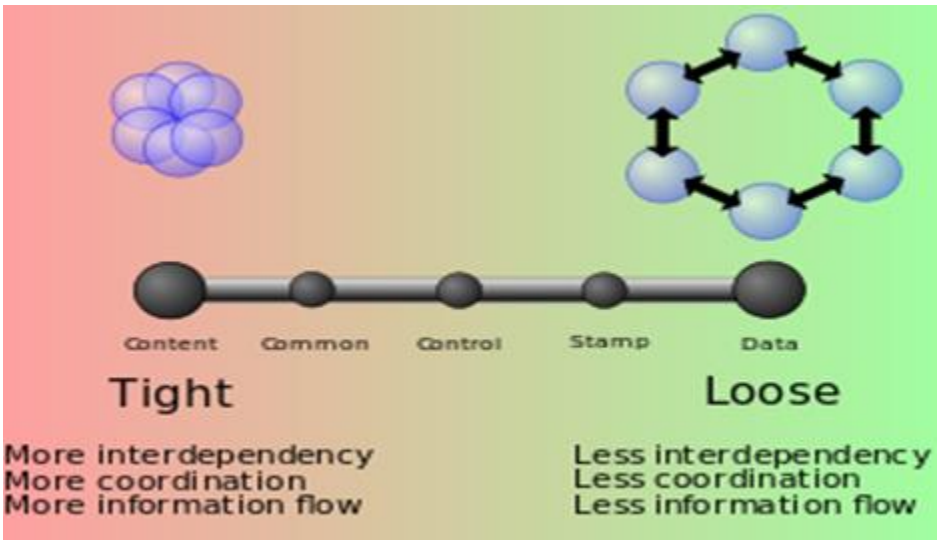
Javascript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript.

**What is coupling?**

In computer systems, coupling refers to the degree of direct knowledge that one component/class has of another. In a way, it is the degree of interdependence between the software modules. Our Application quality metrics depend upon the degree of interdependence components.

**Types of coupling**

Primarily, there are two types of coupling, i.e., loose and tight.




Source: Wikipedia

Loose coupling is a better design, as it promotes single-responsibility and separation of the concerns principle. A loosely coupled module can be consumed and tested independently of other modules. In OOPS programming languages, interface is a powerful tool to use decoupling. Classes communicate via interface

rather than other concrete classes.

In general, tight coupling is bad and not preferred because the modules are dependent on each other. If you want to change one module then it is not easy if it is tightly coupled with the other modules.

**What is modularity?**

An Application is modular, when it is composed of decoupled pieces stored in the modules. As developers, we shall aspire for loose coupling. Hence, our Application will be easy to maintain, update code, and make changes.

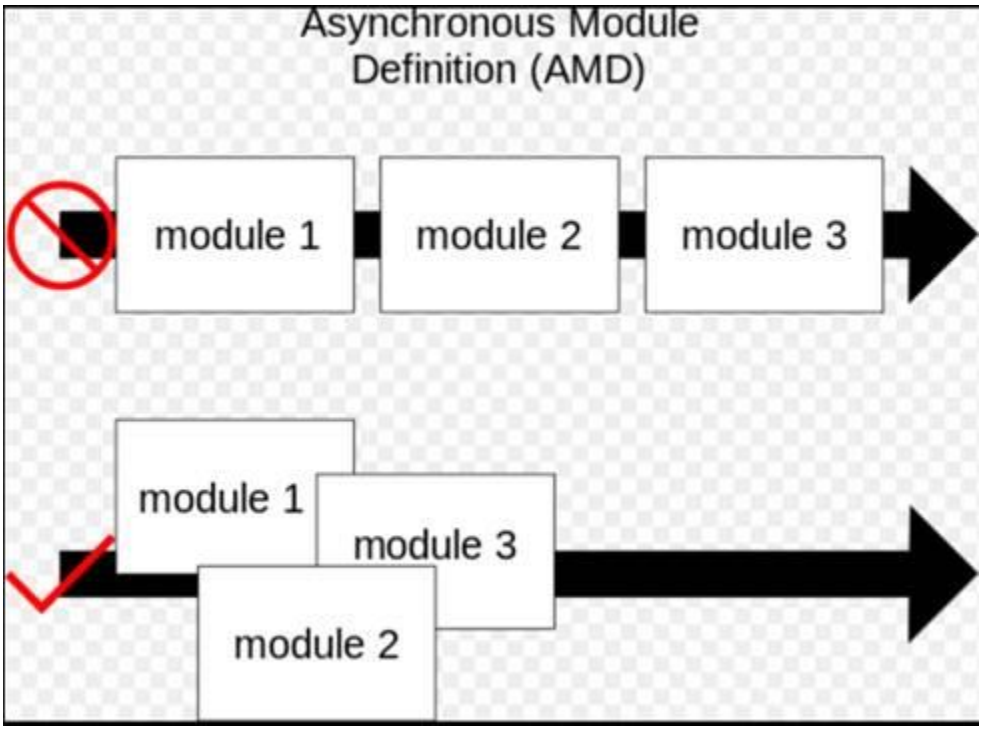
**Module writing in JavaScript**

If you remember, I mentioned in ES6 v2 (part 20), that we have a module available where we can export and import JavaScript functions. The popular concept in loading the module is AMD.

**Asynchronous Module Definition (AMD)**

The overall goal for the AMD format is to provide a solution for the modular JavaScript, that the developers can use today. It loads the module and dependencies asynchronously. There are many advantages associated with AMD and these are:

- Improved performance loads files when require / needed.
- Define dependencies: Allow the developers to define dependencies that must load before a module is executed.
- This pattern is useful, where synchronous loading of the modules incurs performance issues.




Source: Wikipedia

There are various implementations of AMD, i.e., CommonJS, RequireJS, Dojo Toolkit.

**Key concepts with the Modules**

Here are the key concepts.

Loader loads the JavaScript code that handles the logic behind defining & loading modules. You need an AMD loader like dojo.js or require.js

**Define function** Here is the signature of this function.

define (id?, dependencies?, factory);

- Id is optional string literal.

- Dependencies is an array which defines dependencies required by the module. These dependencies must be resolved prior to the execution of the next factory function. It is also optional to say if this module is not dependent on any other module. It loads the dependencies asynchronously.

- Factory Mandatory argument is executed only once.

**Sample**

```
define
([
   'require',
   'dependency'
], function(require, factory)
 {
   'use strict';
});
```

**Require function**

The require() function takes two arguments: an array of the dependencies and a callback function to execute once, all the dependencies are loaded.

**Sample**

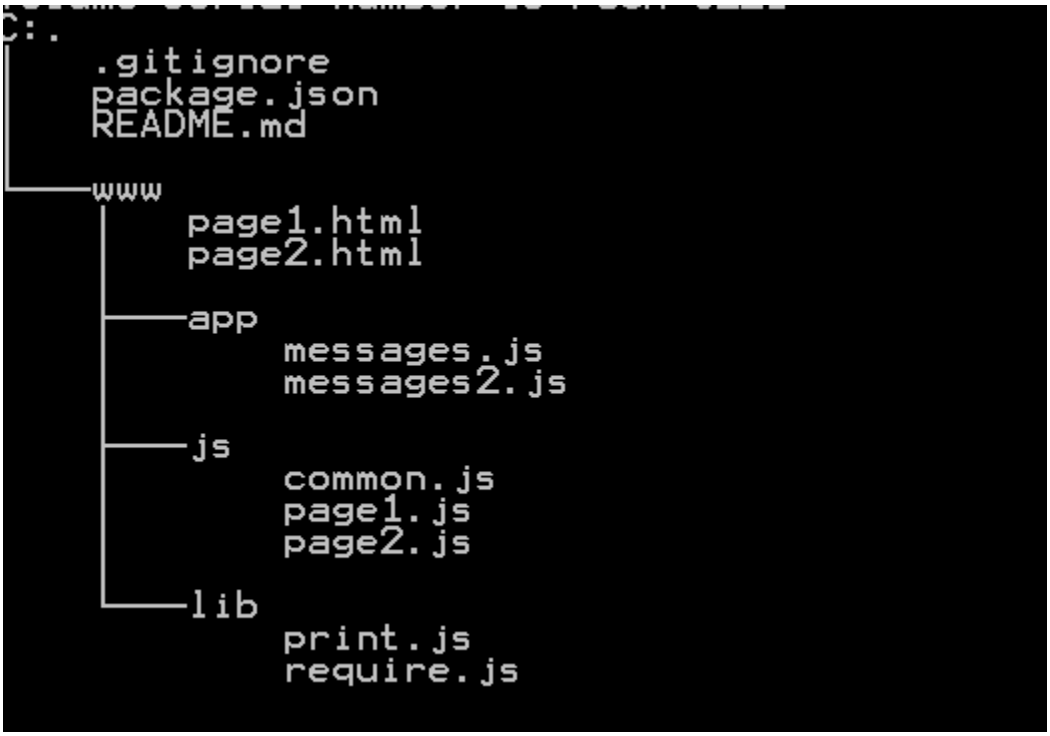
```
require(['calc', 'math'], function(calc, math)
{
   // you code
   calc.getData();
   math.calculate();
});
```

**Play with RequireJS**

RequireJS is the most widely used implementation of AMD. It is a JS file and a

module loader. You can download RequireJS from here.

You can refer to my GitHub repository https://github.com/sumitjolly/create-template and the structure of the project is:




**Download above repository using below command:**

git clone https://github.com/sumitjolly/create-template.git

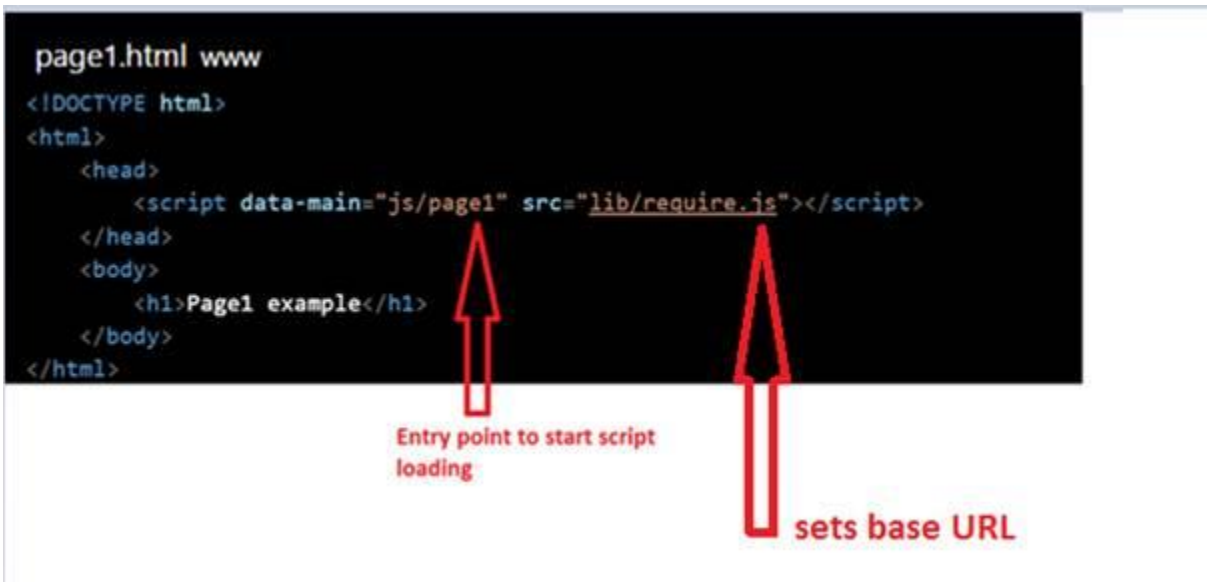
**Project explanation**

The intent is to showcase how RequireJS is useful and how we can write a better code with it. There are various directories, underneath www folder, i.e., app, js, lib. The lib is where, I have kept the require.js file. Let us deep dive into the other details which are as follows:

**Load JavaScript files**

Generally, we use <script src> format to add all our JavaScript files. RequireJS takes a different approach to load the modules. Look at Page1.html,

Page1.html  is an HTML, that includes require.js and pass value js/page1 to the attribute.




**The three components are,**

- *Script* the script tag to load JavaScript
- *data-main* entry point attribute which refer to load a javascript file
- *src* base URL for require.js file

In our directory structure, you can see three files,

- common.js
- page1.js
- page2.js

Now data-main attribute will load js/page1.js.

**Note** RequireJS assumes that file ends with .js. Both js/page1 & js/page1.js work fine.

**Define module**

There is a module that we created messages.js in and defined a dependency in lib/print.js file. The purpose of the messages is to include dependency and return a callback function.

messages.js www\app

```
1   define(['lib/print.js'],function (module1) {
2       module1('Printing ...');
3       return {
4           printConsole: function () {
5               return 'Hello World';
6           }
7       };
8   });
```

print.js file returns a callback print function.

print.js www\lib

```
1  define(function () {
2      return function print(msg) {
3          console.log(msg);
4      };
5  });
```

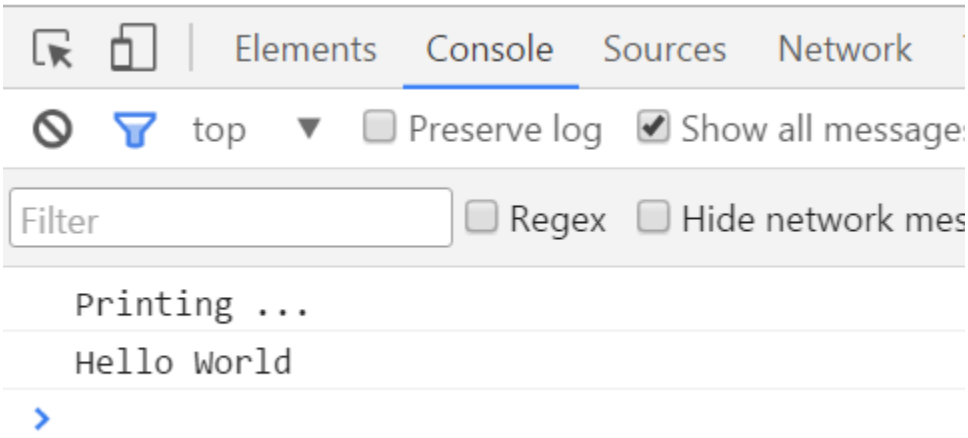
**Calling module**

Page1.js- This page1.js has a dependency on App/messages and uses require() form, that just uses the string name of the module to fetch it.

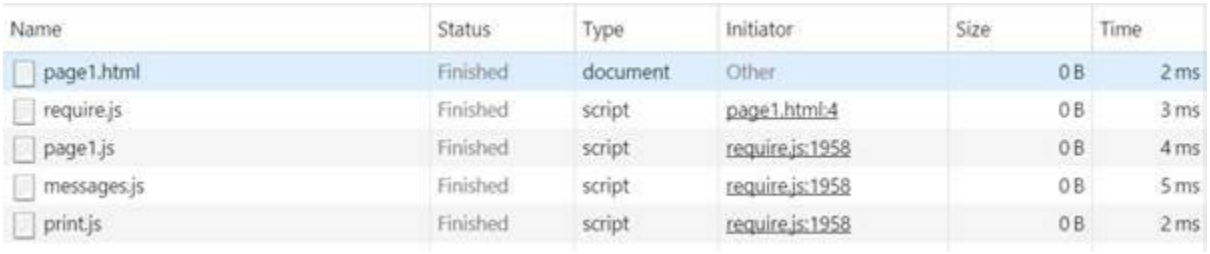
```
page1.js www\js
1   requirejs(['../app/messages'], function(m1){
2       console.log(m1.printConsole());
3   });
4
```

Run www/page1.html in your Browser and review Network & Console Developer (F12).

**Console**




**Network tab**

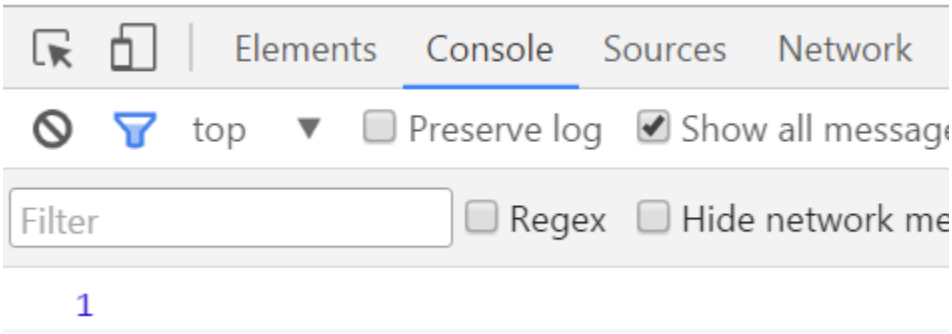


| Name | Status | Type | Initiator | Size | Time |
|---|---|---|---|---|---|
| page1.html | Finished | document | Other | 0 B | 2 ms |
| require.js | Finished | script | page1.html:4 | 0 B | 3 ms |
| page1.js | Finished | script | require.js:1958 | 0 B | 4 ms |
| messages.js | Finished | script | require.js:1958 | 0 B | 5 ms |
| print.js | Finished | script | require.js:1958 | 0 B | 2 ms |

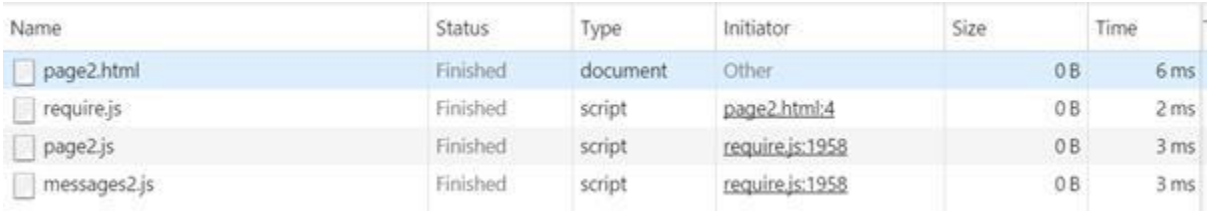
Notice: we can see only relevant JS files are loaded upon executing www/page1.html because only those dependencies are resolved by data-main

You can also try www/page2.html and check Console & Network tab to see page2.html related files only.

**Console**




**Network**



| Name | Status | Type | Initiator | Size | Time |
|---|---|---|---|---|---|
| page2.html | Finished | document | Other | 0 B | 6 ms |
| require.js | Finished | script | page2.html:4 | 0 B | 2 ms |
| page2.js | Finished | script | require.js:1958 | 0 B | 3 ms |
| messages2.js | Finished | script | require.js:1958 | 0 B | 3 ms |

# Adding WinJS To Your Web App

JavaScript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with JavaScript, and what mistakes a developer generally makes and what differences they should be aware of.

### Introduction


WinJS is a Windows library for JavaScript. WinJS provides the controls. You need to make your app professional, equipped with the modern UI controls and include JS, CSS, and HTML all in one place. It is an open-source library by Microsoft and the source code is available at GitHub.

### What's the use of WinJS?

WinJS contains the standard implementation of Win 8 controls which aren't covered by HTML5. It makes it possible to add Windows UI controls in HTML such as:

- AppBar
- DatePicker
- FlipView
- Flyout
- ListView

- Rating
- SemanticZoom
- TimePicker
- ToggleSwitch
- Tooltip
- ViewBox
  and many more…

I found these controls pretty cool as it enhances your webApp UI experience. Microsoft developed this for a universal experience across Win 8 and HTML apps, where you can add WinJS.

**Which frameworks are supported by WinJS?**
WinJS provides support to the three popular JavaScript frameworks via adapters. These are:

- Angular
- React
- Knockout

**What is an adapter?**

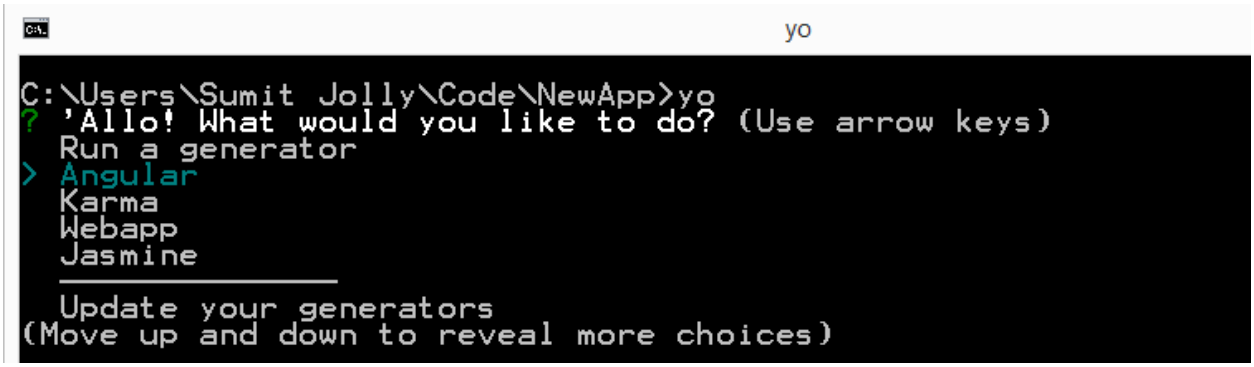
An adapter is a slim layer, which facilitates the usage of WinJS controls with the above three frameworks. Hence, there are different adapters available.

**Create your first WinJS application**

I will leverage the AngularJS adapter to build my first WinJS Application. I use the Yeoman Angular generator to scaffold my Application. Yeoman is a fantastic tool to kick-start new projects. For more information, visit here.

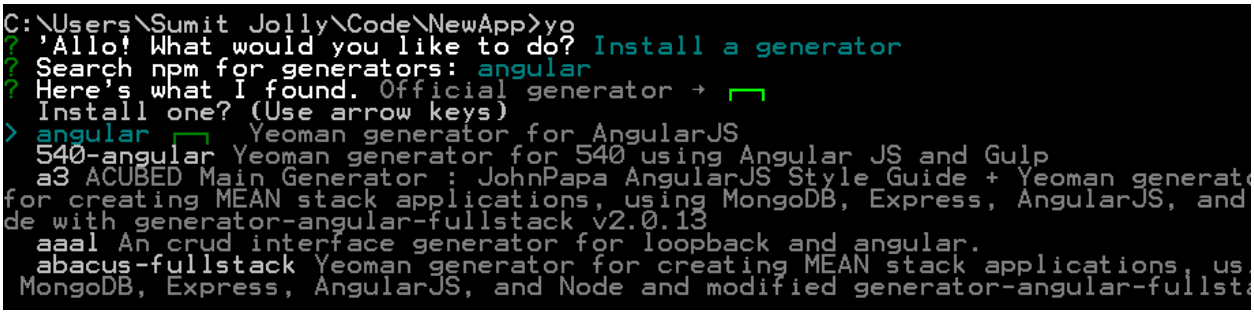
**Setup basic AngularJS webapp**

- Go to command prompt, type YO and it will show you the list of installed generators.




- You must install the Angular generator before you can use it. If you can't find the installed Angular generator, then scroll below to see the option:


You will see the list of matching generators.




Select the top, "Angular," to install it.

- Now, fire YO command again and create a project using "Angular" generator and it will prompt multiple options. Hence, select accordingly.

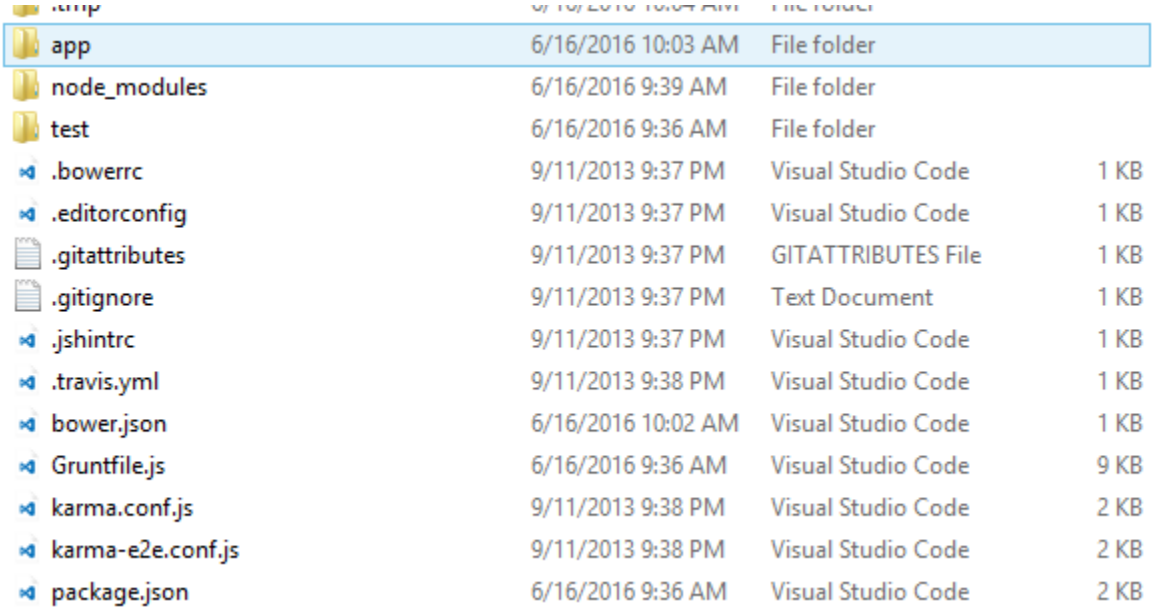
```
C:\Users\Sumit Jolly\Code\NewApp>yo
? 'Allo! What would you like to do? Angular

Make sure you are in the directory you want to scaffold into.
This generator can also be run with: yo angular

[?] Would you like to include Twitter Bootstrap? No
[?] Which modules would you like to include?
 [ ] angular-resource.js
 [ ] angular-cookies.js
>[ ] angular-sanitize.js
```

- Now, you are done with your basic scaffold Application.
- Run the basic Application, using GRUNT command. The default port on which it runs is 9000.

```
Sumit Jolly@GGN0000056637008 MINGW64 ~/Code/NewApp
$ grunt server
```

**Output**


- Go to the directory and the basic structure is ready for you.



| | | | |
|---|---|---|---|
| app | 6/16/2016 10:03 AM | File folder | |
| node_modules | 6/16/2016 9:39 AM | File folder | |
| test | 6/16/2016 9:36 AM | File folder | |
| .bowerrc | 9/11/2013 9:37 PM | Visual Studio Code | 1 KB |
| .editorconfig | 9/11/2013 9:37 PM | Visual Studio Code | 1 KB |
| .gitattributes | 9/11/2013 9:37 PM | GITATTRIBUTES File | 1 KB |
| .gitignore | 9/11/2013 9:37 PM | Text Document | 1 KB |
| .jshintrc | 9/11/2013 9:37 PM | Visual Studio Code | 1 KB |
| .travis.yml | 9/11/2013 9:38 PM | Visual Studio Code | 1 KB |
| bower.json | 6/16/2016 10:02 AM | Visual Studio Code | 1 KB |
| Gruntfile.js | 6/16/2016 9:36 AM | Visual Studio Code | 9 KB |
| karma.conf.js | 9/11/2013 9:38 PM | Visual Studio Code | 2 KB |
| karma-e2e.conf.js | 9/11/2013 9:38 PM | Visual Studio Code | 2 KB |
| package.json | 6/16/2016 9:36 AM | Visual Studio Code | 2 KB |

**Add WinJS in your application**

There are multiple ways to include WinJS in your Application.

- Download WinJS from GIT.
- Refer from CDN.
- Use command prompt tools npm, bower [I prefer this way].
- Fire the commands shown below in your WebApp console location.

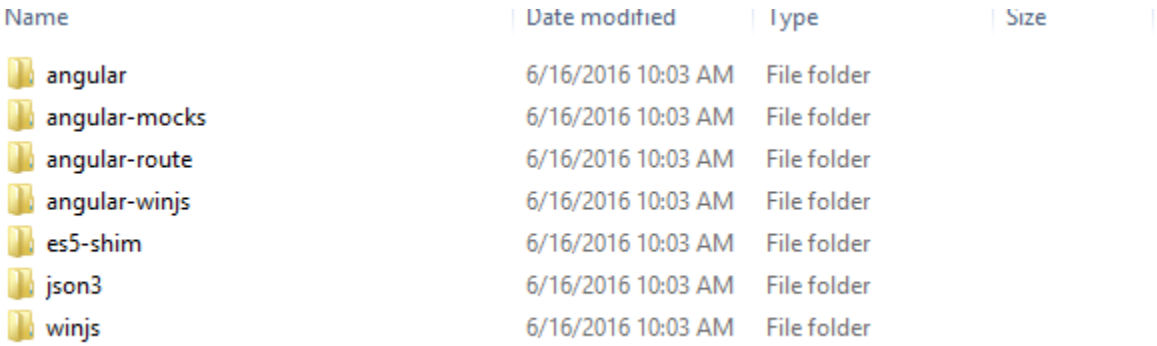
  npm install winjs
  bower install winjs

  This will install WinJS files under app\bower_components folder.

- As I mentioned earlier, we'll add the AngularJS adapter, hence, fire the command shown below:

  bower install angularjs-winjs

Goto directory app\bower_components and see the folder structure, shown below:



| Name | Date modified | Type | Size |
|---|---|---|---|
| angular | 6/16/2016 10:03 AM | File folder | |
| angular-mocks | 6/16/2016 10:03 AM | File folder | |
| angular-route | 6/16/2016 10:03 AM | File folder | |
| angular-winjs | 6/16/2016 10:03 AM | File folder | |
| es5-shim | 6/16/2016 10:03 AM | File folder | |
| json3 | 6/16/2016 10:03 AM | File folder | |
| winjs | 6/16/2016 10:03 AM | File folder | |

- **Modify below files as**

**index.html:** For our WebApp, we shall include all the relevant WinJS and the adapter files:

```
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/angular-route/angular-route.js"></script>
<script src="bower_components/winjs/js/base.js"></script>
<script src="bower_components/winjs/js/ui.js"></script>
<script src="bower_components/angular-winjs/js/angular-winjs.js"></script>
<link href="bower_components/winjs/css/ui-dark.css" rel="stylesheet" />
```

The index.html will be transformed, as shown below:

```
<!doctype html>
<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8]> <html class="no-js lt-ie9"> <![endif]-->
<!--[if gt IE 8]><!-->
<html class="no-js">
<!--<![endif]-->

<head>
  <meta charset="utf-8">
```

```
11.    <meta http-equiv="X-UA-Compatible" content="IE=edge">
12.    <title></title>
13.    <meta name="description" content="">
14.    <meta name="viewport" content="width=device-width">
15.    <!-- Place favicon.ico and apple-touch-
    icon.png in the root directory -->
16.
17.    <!-- build:css(.tmp) styles/main.css -->
18.    <link rel="stylesheet" href="styles/main.css">
19.    <!-- endbuild -->
20.</head>
21.
22.<body ng-app="NewAppApp">
23.    <!--[if lt IE 7]>
24.<p class="browsehappy">You are using an <strong>outdated</strong
    > browser. Please <a href="http://browsehappy.com/">upgrade your
     browser</a> to improve your experience.</p>
25.<![endif]-->
26.
27.    <!--[if lt IE 9]>
28.<script src="bower_components/es5-shim/es5-shim.js"></script>
29.<script src="bower_components/json3/lib/json3.min.js"></script>
30.<![endif]-->
31.
32.    <!-- Add your site or application content here -->
33.    <div class="container" ng-view=""></div>
34.
35.    <!-- Google Analytics: change UA-XXXXX-X to be your site's ID -->
36.    <script>
37.       (function(i, s, o, g, r, a, m)
38.        {
39.           i['GoogleAnalyticsObject'] = r;
40.           i[r] = i[r] || function()
41.           {
42.              (i[r].q = i[r].q || []).push(arguments)
```

```
43.         }, i[r].l = 1 * new Date();
44.         a = s.createElement(o),
45.             m = s.getElementsByTagName(o)[0];
46.         a.async = 1;
47.         a.src = g;
48.         m.parentNode.insertBefore(a, m)
49.       })(window, document, 'script', '//www.google-
    analytics.com/analytics.js', 'ga');
50.
51.       ga('create', 'UA-XXXXX-X');
52.       ga('send', 'pageview');
53.   </script>
54.   <script src="bower_components/angular/angular.js"></script>
55.   <script src="bower_components/angular-route/angular-
    route.js"></script>
56.   <script src="bower_components/winjs/js/base.js"></script>
57.   <script src="bower_components/winjs/js/ui.js"></script>
58.   <script src="bower_components/angular-winjs/js/angular-
    winjs.js"></script>
59.   <link href="bower_components/winjs/css/ui-
    dark.css" rel="stylesheet" />
60.
61.
62.
63.   <!-- build:js({.tmp,app}) scripts/scripts.js -->
64.   <script src="scripts/app.js"></script>
65.   <script src="scripts/controllers/main.js"></script>
66.   <!-- endbuild -->
67.</body>
68.
69.</html>
```

**App.js:** We know this is the starting file of AngularJS and we shall include WinJS module dependency, as shown below:

```
'use strict';
angular.module('NewAppApp', ['winjs', 'ngRoute'])
    .config(function($routeProvider)
    {
        $routeProvider
            .when('/',
            {
                templateUrl: 'views/main.html',
                controller: 'MainCtrl'
            })
            .otherwise
            ({
                redirectTo: '/'
            });
    });
```

**Main.html:** Modify main.html file and include the directive win-rating, as shown below:

```
<win-rating max-rating="maxRating" user-rating="rating"></win-rating>
<div>
    <h2 class="win-h2">Rating: {{rating}}</h2>
    <input class="win-slider" type="range" min="1" max="{{maxRating}}" ng-model="rating" />
</div>
<div>
    <h2 class="win-h2">Maximum Rating: {{maxRating}}</h2>
    <input class="win-slider" type="range" min="1" max="50" ng-model="maxRating" />
</div>
```

**Main.js:** In above main.html, we are using two variable, “rating” and “maxRating,” and we can initialize their value in MainCtrl (controller),

```
1. 'use strict';
2.
3. angular.module('NewAppApp')
4.     .controller('MainCtrl', function($scope)
5.     {
6.         $scope.rating = 1;
7.         $scope.maxRating = 5;
8.     });
```

**Output**


See that the "win-rating" directive has added stars according to the value of the slider.

You can download the code shown above from my GitHub Repository.

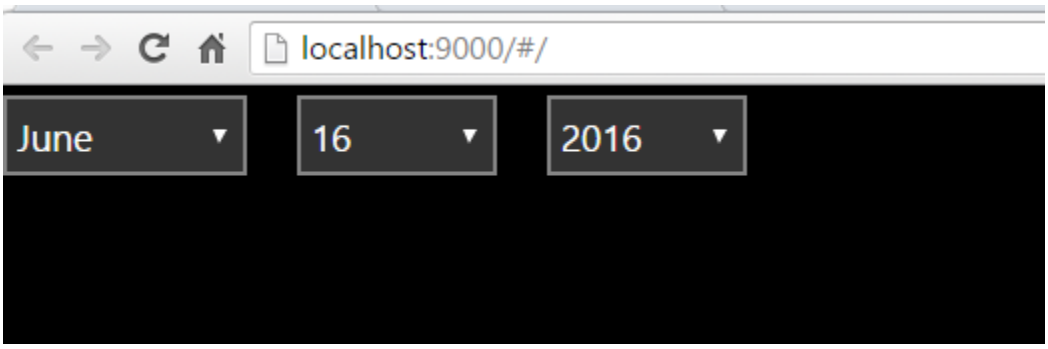
**Note:** Once you clone, please don't forget to run "node install" and "bower install" commands.

There are many other such Win 8 controls which you can try:

- win-date-picker: Add below code in “main.html” and date control will show up,

```
1. <win-date-picker current="date" on-change="dateChanged()"></win-date-picker>
```

**Output**




win-date-picker: Add the code shown below in “main.html” and click the button to see flyout effect.

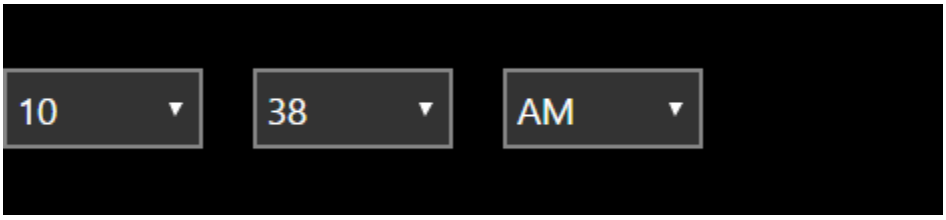
```
1. <button id="flyoutAnchor">Flyout!</button>
2. <win-flyout anchor="'#flyoutAnchor'">Happy coding!</win-flyout>
```

**Output**


win-time-picker: Add the code generated below in “main.html” and see time control.

```
1. <win-time-picker current="time"></win-time-picker>
```

**Output**



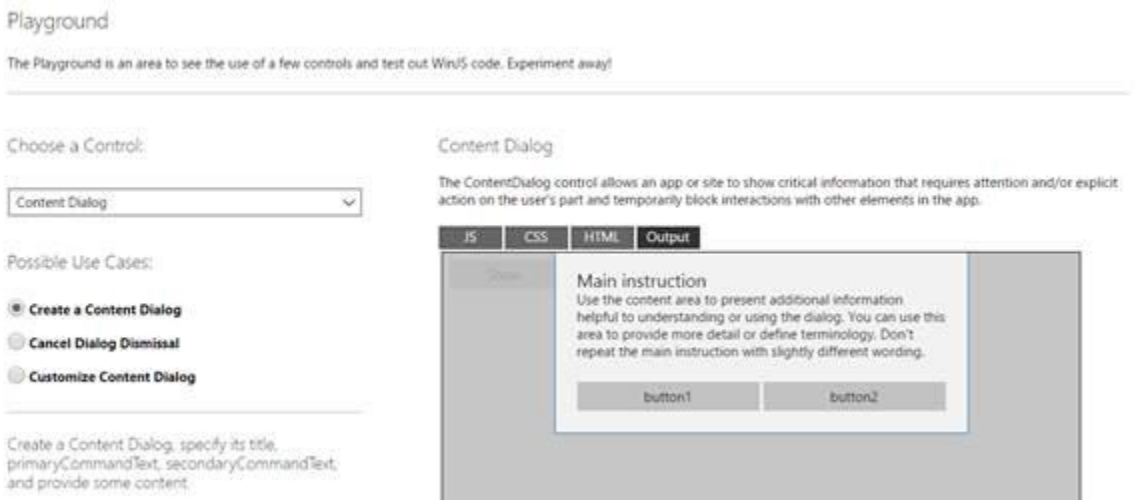

There are many other nice controls available and you can do a lot to enhance your UI experience.

**Playground**

If you would like to try using WinJS without installing the above stuff, you can go to two playgrounds.

- http://winjs.azurewebsites.net/#play
- http://try.buildwinjs.com/playground/

Test the various samples that are provided. For example,



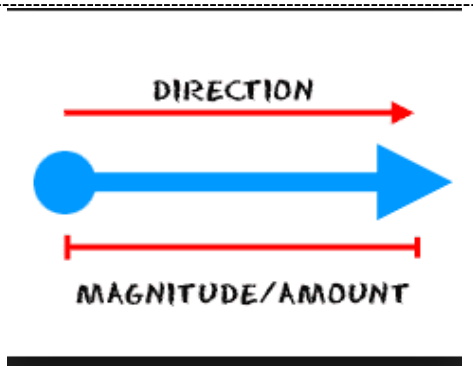
# JavaScript Vector Programming

Javascript is a language of the Web. This part will talk about my observations learned during my decade of software development experience with Javascript. What mistakes a developer generally makes and what differences they should be aware of.

**Flashback**
In computer programming there are two prominent terms, scalar and vector. I am sure these terms must have brought back memories of our computer science/engineering memories.

Computer science is new in comparison to other science streams like Physics or Chemistry. Many computer science concepts are driven by other streams of science. I find roots of Vectors in Physics. Physics describe how any quantity is either a vector or a scalar. Let's look at the differences

**Physics**

| Scalar | Vector |
|---|---|
| Any quantity that can be defined by magnitude (or numerical value) is called scalar. | Any quantity that can be described by magnitude and a direction is called vector. |
| Example: Speed is scalar | Example: Velocity is vector |
|   |   |

**Computer Science**

| Scalar | Vector |
|---|---|
| The term comes from linear algebra, where we differentiate between a single number OR value with matrix. | Vector is dynamic array, which can hold any object, and you can resize it. It has an index and can be accessed by iterator |
| A single value | A one dimensional array |
| Update the value | Add or remove elements |
| Ex- int a = 10; | Ex- var vector = [];<br>Ex- var vector = ['x', 'y']; |

**Vectors in HTML5**

HTML5 implements SVG (Scalable Vector Graphics) via <svg> tag. It is an XML based vector image format for 2D graphics. For example- please refer to this Heart created in SVG,

Source: https://www.svgimages.com/

After you download SVG image then open in any editor and you can see the XML for the above image,

```
<!--?xml version="1.0" standalone="no"?-->
<svg
   xmlns="http://www.w3.org/2000/svg" width="273.000000pt" height="300.000000pt" viewBox="0 0 273.000000 300.000000" preserveAspectRatio="xMidYMid meet">
```

```
    <g transform="translate(0.000000,300.000000) scale(0.100000,-0.100000)" fill="#070707" stroke="none">
        <path style="position: relative;" d="M545 2873 c-167 -35 -350 -177 -419 -324 -37 -78 -66 -211 -66 -304
0 -137 54 -321 132 -451 69 -116 238 -297 453 -484 145 -127 319 -301 398 -397 113 -138 144 -197 197 -363 62 -196 99 -339 107 -414 7 -70 15 -59 38 54
37 189 103 405 161 525 34 71 65 115 149 210 144 162 385 406 465 470 83 66
267 257 326 338 53 71 101 163 135 257 20 58 23 83 23 225 0 142 -2 167 -23
225 -95 266 -234 392 -480 435 -149 26 -260 15 -414 -40 -140 -51 -224 -126
-295 -261 -71 -138 -79 -272 -22 -385 24 -46 106 -121 168 -151 56 -27 165
-30 223 -5 48 20 113 79 133 121 18 37 21 131 7 169 -21 55 -106 106 -176 107
l-40 0 53 -36 c109 -73 108 -163 -3 -238 -69 -47 -140 -32 -209 45 l-38 42 4
76 c8 126 62 207 179 265 91 45 161 59 283 54 88 -3 110 -8 169 -35 113 -54
209 -159 236 -261 39 -143 30 -280 -25 -400 -58 -125 -116 -194 -405 -482
-273 -272 -291 -292 -382 -430 -107 -163 -202 -342 -212 -401 -4 -21 -10 -39
-13 -39 -4 0 -17 32 -30 72 -62 198 -251 469 -478 688 -479 460 -548 561 -561
813 -5 94 -
3 116 17 175 43 129 118 215 236 269 49 22 76 27 164 31 164 6 280
-28 378 -113 125 -108 134 -252 22 -347 -58 -50 -107 -58 -168 -26 -121 62
-120 187 2 253 47 25 46 35 -4 35 -83 0 -157 -62 -181 -151 -11 -40 -10 -55 5
-101 31 -101 119 -159 252 -166 73 -4 88 -1 135 21 30 14 63 34 74 44 74 65
117 180 117 308 0 73 -4 93 -30 147 -33 71 -120 171 -197 228 -140 104 -372
146 -570 103z" id="node1" class="node"></path>
    </g>
    <g transform="translate(0.000000,300.000000) scale(0.100000,-0.100000)" fill="#A2A2A2" stroke="none"></g>
</svg>
```

Ok, the above is complex code. At C-sharpcorner there is a very good series on SVG so you can read and learn SVG from that series:

Before reading further, read the following parts.

- Overview of Scalable Vector Graphics (SVG)
- Scalable Vector Graphics - Rectangle
- Scalable Vector Graphics - Circle
- Scalable Vector Graphics - Ellipse
- Scalable Vector Graphics - Line
- Scalable Vector Graphics - Polyline
- Scalable Vector Graphics - Polygon
- Scalable Vector Graphics - Path 1
- Scalable Vector Graphics - Path 2
- Scalable Vector Graphics - Path 3
- Scalable Vector Graphics - Text 1
- Scalable Vector Graphics - Text 2
- Scalable Vector Graphics - Filters 1
- Scalable Vector Graphics - Filters 2
- Scalable Vector Graphics - Filters 3
- Scalable Vector Graphics - Filters 4

**Create SVG graphics**

You can learn SVG from the above parts and if you want to create SVG quickly then create it using,

- Draw SVG

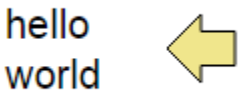
There is a < 1-min interesting exercise to do.

I created a SVG file and you can copy and paste the below in notepad/editor and save as .svg file.

```
1. <svg
2.     xmlns="http://www.w3.org/2000/svg"
```

```
3.    xmlns:xlink="http://www.w3.org/1999/xlink" width="1620px" height="8
   50px" viewBox="0 0 1620 850" preserveAspectRatio="xMidYMid meet" >
4.    <defs id="svgEditorDefs">
5.       <polygon id="svgEditorShapeDefs" fill="khaki" stroke="black" style="ve
   ctor-effect: non-scaling-stroke; stroke-width: 1px;"/>
6.    </defs>
7.    <rect id="svgEditorBackground" x="0" y="0" width="1620" height="850"
   style="fill: none; stroke: none;"/>
8.    <text fill="black" x="219" y="83" id="e1_texte" style="font-
   family: Arial; font-size: 20px;">hello
9.       <tspan x="219" dy="1.25em">world</tspan>
10.   </text>
11.   <path d="M0,-2v-2l-4,4l4,4v-2h4v-
   4Z" stroke="black" id="e2_shape" style="vector-effect: non-scaling-
   stroke; stroke-
   width: 1px;" fill="khaki" transform="matrix(4.25 0 0 4.25 317 88)"/>
12.</svg>
```

Now, open in any browser and it shows,




**Vectors in JavaScript**

In JavaScript you can use Array to leverage vectors and can push or delete elements.

In Node I am a fan of Victorjs which supports many vector-based operations.

- Victorjs

This is JavaScript 2D vector maths library for Node.js and the browser.

This library can be used in both NodeJS and the browser. As we're focusing on frontend I will show you Browser examples.

**Prerequisite**

Please setup frontend webapp using node/npm OR you can download the code shown above from my GitHub Repository.

Add Angular Controller and view either manually OR my preferred way,

- *yo angular:*controller victor

```
$ yo angular:controller victor
   create app\scripts\controllers\victor.js
   create test\spec\controllers\victor.js
```

- *yo angular:*view victor

```
$ yo angular:view victor
   create app\views\victor.html
```

Install VictorJS,


Install javascript libraries,

*bower install victor --save*

Refer to library in Index.html file,

```
<script src="./bower_components/victor/build/victor.js"></script>
```

Add route “/victor” to reach new view in “app.js”,

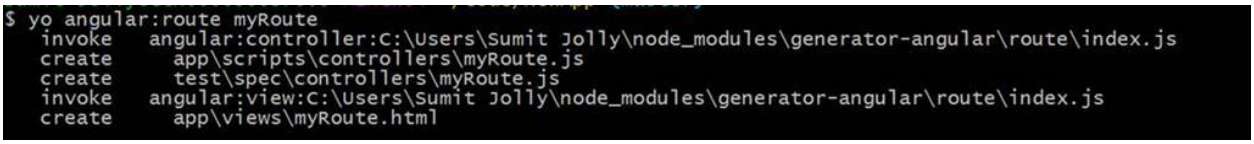
```
'use strict';

angular.module('NewAppApp', ['winjs', 'ngRoute'])
    .config(function($routeProvider)
    {
      $routeProvider
        .when('/',
        {
          templateUrl: 'views/main.html',
          controller: 'MainCtrl'
        })
        .when('/victor',
        {
          templateUrl: 'views/victor.html',
          controller: 'VictorCtrl'
        })
        .otherwise
        ({
          redirectTo: '/'
        });
  });
```

**Note**

I wanted to showcase 3-steps explicitly, otherwise a smart way to add the above three steps, Controller | View | Route, is by firing the below command,

**Example**

*yo angular:*route myRoute




Modify victor.js as,

```
1. 'use strict';
2.
3. angular.module('NewAppApp')
4.     .controller('VictorCtrl', function($scope)
5.     {
6.         var vec = new Victor(10, 20);
7.         $scope.x = vec.x; // will fetch 10
8.         $scope.y = vec.y; // will fetch 20
9.     });
```

Modify victor.html as,

```
1. <p>This is the victor view.</p>
2. <div>Value of X-axis: {{x}}</div>
3. <div>Value of Y-axis: {{y}}</div>
```

Run grunt server and route to “#/victor”,


Similarly, there are many other common vector operations which you can perform using victorJS library like,

- *addX:* to add values of X-axis in two vectors,

```
> var v1 = new Victor(10,20);
undefined
> var v2 = new Victor(20,20);
undefined
> v1.addX(v2);
▶ Victor {x: 30, y: 20}
```

- *subtract:* to subtract values from two vectors
- *toArray:* converts a vector into a one-dimensional array,

```
> Victor(10,20).toArray();
[10, 20]
```