

Web API, JavaScript Console, ECMAScript, Scripting Language

What is Web API?

Before we understand what is Web API, let's see what is an API (Application Programming Interface).

As per [Wikipedia's Definition of API](#): In computer programming, an application programming interface (API) is a set of subroutine definitions, protocols, and tools for building software and applications.

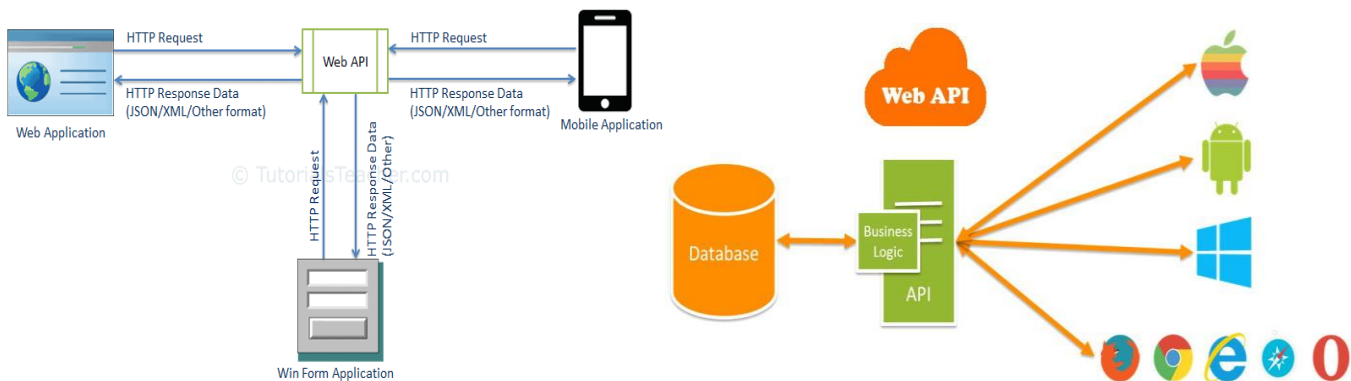
To put it in simple terms, API is some kind of interface which has a set of functions that allow programmers to access specific features or data of an application, operating system or other services.

Web API as the name suggests, is an API over the web which can be accessed using HTTP protocol. It is a concept and not a technology. We can build Web API using different technologies such as Java, .NET etc. For example, Twitter's [REST APIs](#) provide programmatic access to read and write data using which we can integrate twitter's capabilities into our own application.

ASP.NET Web API

The ASP.NET Web API is an extensible framework for building HTTP based services that can be accessed in different applications on different platforms such as web, windows, mobile etc. It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view.

It is like a webservice or WCF service but the exception is that it only supports HTTP protocol



Web API: It is a part of ASP.NET Framework to build and deploy the REST services on HTTP protocol and accessible to wide range of devices.

ASP.NET MVC API: When I am talking about ASP.NET MVC API, It means, it is also an API but creating with ASP.NET MVC application. I mean to say both view and data in the same project.

When you develop your web application using the MVC, many developers get confused

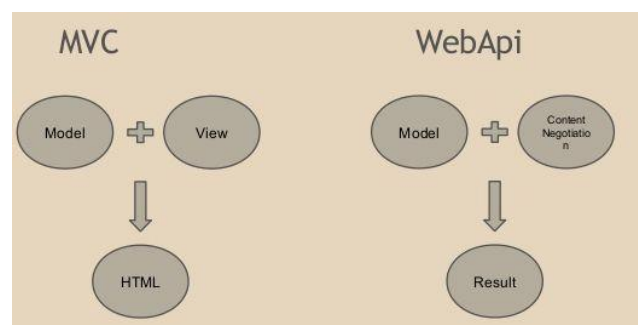
between Web API and ASP.NET MVC. They are confused about which one will be better and how.

View and Data

As we all know Web API is used to create the fully REST service which is exposed only on HTTP protocol, so only HTTP enabled client is able to access the Web API service. Web API only returns data not view. There is not any graphical user interface to represent the data. Client can get the data and present it on their view. But if you create the API using the ASP.NET MVC then you

will be able to create view as well as return data. It means ASP.NET MVC APIs return data and represent this data on their view.

So, if you are going to create API which will be going to return only data then you need to prefer Web API otherwise it is good to go with ASP.NET MVC API.



Content Negotiation

Web API supports one of the most important features in API world and that is Content Negotiation. Web API decides and return best response format data that is acceptable by client very easily. The data format can be XML, JSON, ATOM or any other format data. But with ASP.NET MVC API, if we want the data in JSON, you need to cast your data.

Web API also supports self-hosting with any ASP.NET application. So, if you think that your client application is not in ASP.NET MVC then you should always prefer to create your API in Web API. But if you use ASP.NET MVC to create API then you cannot use self-hosting feature with this application



Logging messages to the console is a very basic way to diagnose and troubleshoot minor issues in your code. But, did you know that there is more to console than just log? In this article, I'll show you how to print to the console in JS, as well as all of the things you didn't know console could do.

Firefox Multi-line Editor Console

If you've never used the multi-line editor mode in Firefox, you should give it a try right now! Just open the console, Ctrl+Shift+K or F12, and in the top right you will see a button that says "Switch to multi-line editor mode". Alternatively, you can press Ctrl+B.

This gives you a multi-line code editor right inside Firefox.

console.log

Let's start with a very basic log example.

```
let x = 1
console.log(x)
```

Type that into the Firefox console and run the code. You can click the "Run" button or press Ctrl+Enter.

In this example, we should see "1" in the console. Pretty straightforward, right?

Multiple Values

Did you know that you can include multiple values? Add a string to the beginning to easily identify what it is you are logging.

```
let x = 1
console.log("x:", x)
```

But what if we have multiple values that we want to log?

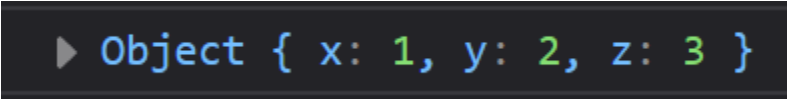
```
let x = 1
let y = 2
let z = 3
```

Instead of typing `console.log()` three times we can include them all. And we can add a string before each of them if we wanted, too.

```
let x = 1
let y = 2
let z = 3
console.log("x:", x, "y:", y, "z:", z)
```

But that's too much work. Just wrap them with curly braces! Now you get an object with the named values.

```
let x = 1
let y = 2
let z = 3
console.log( {x, y, z} )
```



Console Output

You can do the same thing with an object.

```
let user = {
  name: 'Jesse',
  contact: {
    email: 'codestackr@gmail.com'
  }
}
console.log(user)
console.log({user})
```

The first log will print the properties within the user object. The second will identify the object as "user" and print the properties within it.

If you are logging many things to the console, this can help you to identify each log.

Variables within the log

Did you know that you can use portions of your log as variables?

```
console.log("%s is %d years old.", "John", 29)
```

In this example, `%s` refers to a string option included after the initial value. This would refer to "John".

The `%d` refers to a digit option included after the initial value. This would refer to 29.

The output of this statement would be: "John is 29 years old."

Variations of logs

There are a few variations of logs. There is the most widely used `console.log()` . But there is also:

```
console.log('Console Log')
console.info('Console Info')
console.debug('Console Debug')
console.warn('Console Warn')
console.error('Console Error')
```

These variations add styling to our logs in the console. For instance, the `warn` will be colored yellow, and the `error` will be colored red.

Note: The styles vary from browser to browser.

Optional Logs

We can print messages to the console conditionally with `console.assert()` .

```
let isItWorking = false
console.assert (is It Working, "this is the reason why")
```

If the first argument is false, then the message will be logged.

If we were to change `isItWorking` to `true`, then the message will not be logged.

Counting

Did you know that you can count with console?

```
for(i=0; i<10; i++){
  console.count()
}
```

Each iteration of this loop will print a count to the console. You will see "default: 1, default: 2", and so on until it reaches 10.

If you run this same loop again you will see that the count picks up where it left off; 11 - 20.

To reset the counter we can use `console.countReset()` .

And, if you want to name the counter to something other than "default", you can!

```
for(i=0; i<10; i++){
  console.count('Counter 1')
}
console.countReset('Counter 1')
```

Now that we have added a label, you will see "Counter 1, Counter 2", and so on.

And to reset this counter, we have to pass the name into `countReset`. This way you can have several counters running at the same time and only reset specific ones.

Track Time

Besides counting, you can also time something like a stopwatch.

To start a timer we can use `console.time()`. This will not do anything by itself. So, in this example, we will use `setTimeout()` to emulate code running. Then, within the timeout, we will stop our timer using `console.timeEnd()`.

```
console.time()
setTimeout(() => {
  console.timeEnd()
}, 5000)
```

As you would expect, after 5 seconds, we will have a timer end log of 5 seconds.

We can also log the current time of our timer while it's running, without stopping it. We do this by using `console.timeLog()`.

```
console.time()

setTimeout(() => {
  console.timeEnd()
}, 5000)

setTimeout(() => {
  console.timeLog()
}, 2000)
```

In this example, we will get our 2 second `timeLog` first, then our 5 second `timeEnd`.

Just the same as the counter, we can label timers and have multiple running at the same time.

Groups

Another thing that you can do with log is group them. ?

We start a group by using `console.group()`. And we end a group with `console.groupEnd()`.

```
console.log('I am not in a group')

console.group()
console.log('I am in a group')
console.log('I am also in a group')
console.groupEnd()

console.log('I am not in a group')
```

This group of logs will be collapsible. This makes it easy to identify sets of logs.

By default, the group is not collapsed. You can set it to collapsed by using `console.groupCollapsed()` in place of `console.group()`.

Labels can also be passed into the `group()` to better identify them.

Stack Trace

You can also do a stack trace with `console`. Just add it into a function.

```
function one() {  
  two()  
}  
function two() {  
  three()  
}  
function three() {  
  console.trace()  
}  
one()
```

In this example, we have very simple functions that just call each other. Then, in the last function, we call `console.trace()`.

```
console.trace()  
  
three      debugger eval code:8  
two        debugger eval code:5  
one        debugger eval code:2  
<anonymous> debugger eval code:10
```

Console Output

Tables

Here's one of the most mind-blowing uses for `console`: `console.table()`.

So let's set up some data to log:

```
let devices = [  
  {  
    name: 'iPhone',  
    brand: 'Apple'  
  },  
  {  
    name: 'Galaxy',  
    brand: 'Samsung'  
  }  
]
```

```
}  
]
```

Now we'll log this data using `console.table(devices)`.

```
console.table() debugger eval code:11:9
```

(index)	name	brand
0	iPhone	Apple
1	Galaxy	Samsung

Console Output

But wait – it gets better!

If we only want the brands, just `console.table(devices, ['brand'])`!

```
console.table() debugger eval code:2:9
```

(index)	brand
0	Apple
1	Samsung

Console Output

How about a more complex example? In this example, we'll use `jsonplaceholder`.

```
async function getUsers() {  
  let response = await fetch('https://jsonplaceholder.typicode.com/users')  
  let data = await response.json()  
  
  console.table(data, ['name', 'email'])  
}  
  
getUsers()
```

Here we are just printing the "name" and "email". If you `console.log` all of the data, you will see that there are many more properties for each user.

Style ?

Did you know that you can use CSS properties to style your logs?

To do this, we use `%c` to specify that we have styles to add. The styles get passed into the second argument of the log.


```
console.log("%c This is yellow text on a blue background.", "color:yellow; background-color:blue")
```

You can use this to make your logs stand out.

Clear

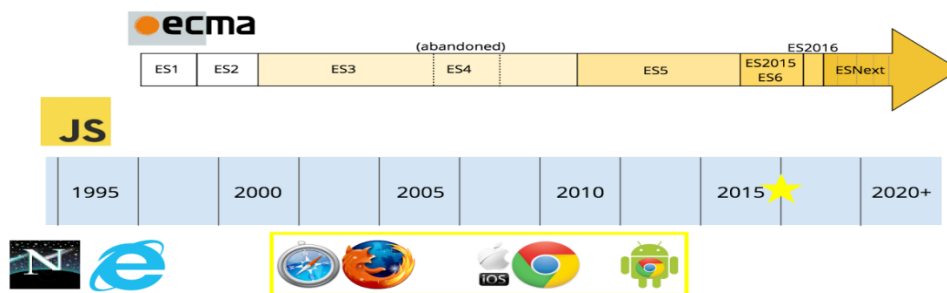
If you are trying to troubleshoot an issue using logs, you may be refreshing a lot and your console may get cluttered.

Just add `console.clear()` to the top of your code and you'll have a fresh console every time you refresh. ?

Just don't add it to the bottom of your code, lol.

ECMASCRIPT

Before going to see about **ECMASCRIPT** first we see the origin and evolution of **ECMASCRIPT**.



EVOLUTION OF ECMASCRIPT

WHAT IS ECMASCRIPT?

1. JavaScript was originally named JavaScript in hopes of capitalizing on the success of Java.
2. After that Netscape submitted JavaScript to ECMA International for Standardization.
3. Then it results in a new language standard, known as **ECMAScript**.
4. ECMA script is a language based on JavaScript.
5. International ECMA organization ensures that JavaScript written for one company browser would work in all companies' browsers and also maintain the standard for the language.
6. ECMAScript is the standard name for JavaScript. Many developers are fear for using new features of ECMA that are why the new features of ECMAScript can be added to JavaScript engines so developers can hold of them with extra features.

ECMAScript is often abbreviated as **ES**. **ES** is followed by number that it refers to the edition of **ECMAScript**.

Since there are eight editions of **ECMAScript** published.

ES3, ES5, ES6, ES7, ES8, ES2015, ES2016, ES2017.

Beside JavaScript, other languages implemented **ECMAScript**, including:

- **Action Script** (the Flash scripting language), which is losing popularity since Flash will be officially discontinued in 2020
- **J Script** (the Microsoft scripting dialect), since at the time JavaScript was supported only by Netscape and the browser wars were at their peak, Microsoft had to build its own version for Internet Explorer

but of course JavaScript is the **most popular** and widely used implementation of ES.

ECMAScript version

The current ECMAScript version is **ES2018**. It was released in June 2018.

Edition	Official name	Date published
ES9	ES2018	June 2018
ES8	ES2017	June 2017
ES7	ES2016	June 2016
ES6	ES2015	June 2015
ES5.1	ES5.1	June 2011
ES5	ES5	December 2009
ES4	ES4	Abandoned
ES3	ES3	December 1999
ES2	ES2	June 1998
ES1	ES1	June 1997

ECMAScript versions

Let's dive into the specific features added to JavaScript since ES5. Let's start with the ES2015 features.

let and const

Until ES2015, **var** was the only construct available for defining variables.

```
var a = 0
```

If forget to add **var** it leads to be assigning a value to an undeclared variable, and the results might vary.

If don't initialize the variable when declare it, it will have the **undefined** value until assign a value to it.

```
var a // type of a === 'undefined'
```

Also declare multiple variables at once in the same statement:

```
var a = 1, b = 2;
```

Using let

let is a new feature introduced in **ES2015** and it's essentially a block scoped version of **var**. Its scope is limited to the block, statement or expression where it's defined.

Modern JavaScript developers might choose to only use **let** and completely discard the use of **var**.

Using `const`

Variables declared with **var** or **let** can be changed later on in the program, and reassigned. Once a **const** is initialized, its value can never be changed again, and it can't be reassigned to a different value.

```
const a = 'test'
```

Arrow Functions

Visually, **function** is a simple, which allows you to write functions with a shorter syntax, from:

```
const myFunction = function()
```

```
{  
  //...  
}
```

This can be rewritten as:

```
const myFunction = () =>
```

```
{  
  //...  
}
```

Implicit return

Arrow functions allow you to have an implicit return: values are returned without having to use the **return** keyword.

This function works when there is one line statement in the function body,

```
const myFunction = () => 'test'  
myFunction() // 'test'
```

Enhanced Object Literals

In ES2015 Object Literals gained superpowers.

Simpler syntax to include variables

```
const something = 'y'  
const x = {  
  something  
}
```

New Object methods

ES2015 introduced several static methods under the Object namespace:

1. **Object.is()** determines if two values are the same value
2. **Object.assign()** used to shallow copy an object
3. **Object.setPrototypeOf** sets an object prototype.

Map

A Map data structure allows us to associate data to a key.

Before ES6

Before its introduction, people generally used objects as maps, by associating some object or value to a specific key value:

```
const car = {}  
car['color'] = 'red'  
car.owner = 'Flavio'  
console.log(car['color']) //red  
console.log(car.color) //red  
console.log(car.owner) //Flavio  
console.log(car['owner']) //Flavio
```

Enter Map

ES6 introduced the Map data structure, providing us a proper tool to handle this kind of data organization.

A Map is initialized by calling:

```
const m = new Map()
```

Add items to a Map

add items to the map by using the set method:

```
m.set('color', 'red')  
m.set('age', 2)
```

Get an item from a map by key

And you can get items out of a map by using get:

```
const color = m.get('color')  
const age = m.get('age')
```

Delete an item from a map by key

Use the `delete()` method:

```
m.delete('color')
```

Delete all items from a map

Use the `clear()` method:

```
m.clear()
```

Check if a map contains an item by key

Use the `has()` method:

```
const hasColor = m.has('color')
```

Find the number of items in a map

Use the `size` property:

```
const size = m.size
```

Initialize a map with values

initialize a map with a set of values:

```
const m = new Map([[ 'color', 'red'], [ 'owner', 'Flavio'], [ 'age', 2]])
```

Map keys

Just like any value (object, array, string, number) can be used as the value of the key-value entry of a map item, any value can be used as the key, even objects.

String padding

The purpose of string padding is to add characters to a string, so it reaches a specific length.

ES2017 introduces two string methods: `padStart()` and `padEnd()`.

```
padStart(targetLength [, padString])
```

```
padEnd(targetLength [, padString])
```

Sample usage:

padStart()	
'test'.padStart(4)	'test'
'test'.padStart(5)	' test'
'test'.padStart(8)	' test'
'test'.padStart(8, 'abcd')	'abcdtest'
padEnd()	
'test'.padEnd(4)	'test'
'test'.padEnd(5)	'test '
'test'.padEnd(8)	'test '
'test'.padEnd(8, 'abcd')	'testabcd'

Regular Expression improvements

ES2018 introduced a number of improvements regarding Regular Expressions.

ESNext

What's next? ESNext.

ESNext is a name that always indicates the next version of JavaScript.

The current ECMAScript version is **ES2018**. It was released in June 2018.

Historically JavaScript editions have been standardized during the summer, so we can expect **ECMAScript 2019** to be released in summer 2019.

So at the time of writing, ES2018 has been released, and **ESNext** is **ES2019**

Proposals to the ECMAScript standard are organized in stages. Stages 1–3 are an incubator of new features, and features reaching Stage 4 are finalized as part of the new standard.

ECMAScript is a term commonly used by developers for coding standard but most of us don't know how did it come into the picture and why do we need it?

Imagine a world without a set of rules for coding. Everyone would be writing their own codes, will have their own set of rules and when a new person enters the world of software development, he/she will never know which set of rules they need to follow. The situation will become chaotic!!

The same incident happened with Javascript. When Javascript was first created by Netscape then there was a war going on between all the browser vendors in the market. Microsoft implemented its own version of javascript in Internet Explorer and Mozilla implemented its own. Similarly, other browser vendors implemented their own versions.

All this created a huge problem for the developers. One version ran fine on Netscape but was a total waste on Internet Explorer or Firefox.

To solve the cross-browser compatibility, Javascript was standardized by the ECMA international and that's the reason it got the name ECMAScript. All browsers eventually implemented ECMAScript (though it took a lot of time).

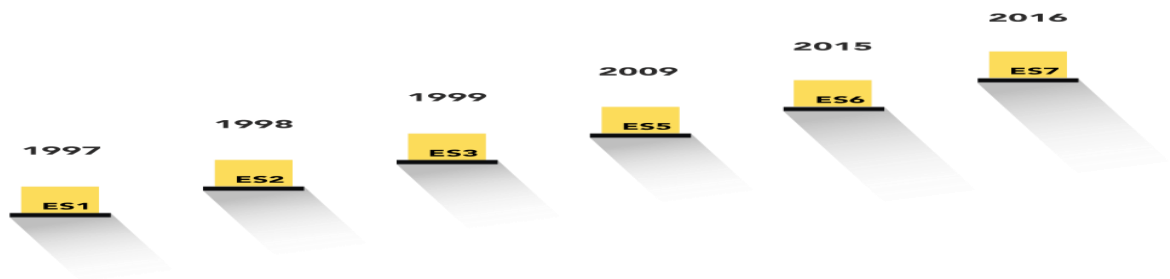
How is Javascript related to ECMAScript

JavaScript is actually the term most of the developers use for ECMAScript. Although there are other implementations available for ECMAScript like – JScript (Microsoft) and ActionScript (Adobe), but JavaScript has proven to be the best-known implementation of ECMAScript since it was first published.

Jeff Atwood (founder of Stack Overflow) coined the term “Atwood’s Law,” which states:

“Any application that can be written in JavaScript, will eventually be written in JavaScript. “

It’s more than ten years now, and Atwood’s statement still lingers on. JavaScript is continuing to gain more and more adoption. The “next generation” of Javascript is something known as ES6 (The 6th edition, officially known as ECMAScript 2015).



6th Edition – ECMAScript 2015

The 6th edition, officially known as ECMAScript 2015 is different because it introduces new syntax. Infact, a lot of new syntaxes.

Browser support for ES6 is still incomplete. However, we can still use ES6 features by using a [pre-processor like Babel](#) to cross-compile our JavaScript back to ES5 compatible code for older browsers, so there’s no reason to put off learning about it.

This update added arrow functions, promises, let and const, classes and modules and a lot of new features, but defines them semantically in the same terms as ECMAScript 5 strict mode. The complete list of new features can be found on <http://es6-features.org>

Some of the new features that are introduced in ES6 –

Default Parameters

Old way	New way
<pre>function (height, color) { var height = height 150; var color = color 'fair'; ... }</pre>	<pre>function(height = 150, color = 'fair') { ... }</pre>

We do not need to worry about parameters taking 0 or undefined value since we can define defaults in the parameters' list now. This also saves the overhead of sending parameters to each function call when we wish to use the defaults.

Block-Scoped Let and Const

Old way	New way-1	New way-2
<pre>function calcAmount () { var amount = 0; { var amount = 1; { var amount = 100; } } return amount; } //amount is 100</pre>	<pre>function calcAmount () { let amount = 0; { let amount = 1; { let amount = 100; } } return amount; } //amount is 0</pre>	<pre>function calcAmount () { const amount = 0; { const amount = 1; { const amount = 100; } } return amount; } //amount is 0</pre>

'let' is a new 'var' which restricts the scope of the variable to a block instead of the whole function. With 'const', things are easier. It's just an immutable entity, and at the same time block-scoped like let.

Template Literals

Old way	New way
<pre>var name = My name is ' + first + ' ' + last; var url = 'http://localhost:5000/api/' + id;</pre>	<pre>let name = `My name is \${first} \${last}`; let url = `http://localhost:5000/api/\${id}`;</pre>

It determines the way to output or append variables in a string. Now we have a simple bash-like syntax that makes the code look prettier.

Multi-line Strings

Old way	New way
<pre>var story = 'Then took the other, \n\t' + 'as just as fair, And having \n\t' + 'perhaps the better claim, \n\t' + 'Because it was grassy and \n\t' + 'wanted wear.\n\t';</pre>	<pre>let story = `Then took the other, as just as fair, And having perhaps the better claim, Because it was grassy and wanted wear.`;</pre>

When writing a multiple line string, ES6 save a lot of typing efforts by its new syntax that uses 'backticks'.

Destructuring Assignment

Old way	New way
<pre>// req.body has username and password var username = req.body.username; var password = req.body.password;</pre>	<pre>// req.body has username and password let {username, password} = req.body;</pre>

It provides intuitive and flexible destructuring of Arrays and Objects into individual variables during assignment so we do not need to write extra code for the assignment task.

Arrow Functions

Old way	New way
<pre>var that = this; \$('.btn').click(function(event){ that.callFunction(); })</pre>	<pre>\$('.btn').click((event) => { this.callFunction(); })</pre>

Arrows are shorthand for functions using the `=>` syntax. They are syntactically similar to C#, Java 8 and CoffeeScript arrows. Arrows would make your `'this'` behave properly, i.e., `'this'` will have the same value as in the context of the parent function – it won't mutate.

Promises

Old way	New way
<pre>function showMsg (msg, timeout, callback) { setTimeout(function () { callback(msg); }, timeout); }</pre>	<pre>function showMsg (msg, timeout) { return new Promise((resolve, reject) => { setTimeout(() => resolve(`\${msg}`), timeout); }) }</pre>

Promises provide the representation of a value that may be made asynchronously available in the future. We can also combine one or more promises into new promises without worrying about the order of the underlying asynchronous operations.

Classes

Old way	New way
<pre>var Shape = function (id, x, y) { this.id = id; this.move(x, y); }; Shape.prototype.move = function (x, y) { this.x = x; this.y = y; };</pre>	<pre>class Shape { constructor (id, x, y) { this.id = id this.move(x, y) } move (x, y) { this.x = x this.y = y } }</pre>

ES6 provides more intuitive, OOP-styled and boilerplate-free classes. These classes encourage the prototype-based object-oriented pattern which brings support for inheritance, constructors, static methods and more.

Modules

Old way	New way
<pre>// lib/math.js LibMath = {}; LibMath.sum = function (x, y) { return x + y; }; LibMath.pi = 3.141593; // app.js var sum = LibMath.sum, pi = LibMath.pi; console.log("2π = " + sum(pi, pi));</pre>	<pre>// lib/math.js export function sum (x, y) { return x + y; } export let pi = 3.141593; // app.js import { sum, pi } from "lib/math"; console.log("2π = " + sum(pi, pi));</pre>

ES6 includes a built-in module system which provides support for exporting values from modules and importing values to modules within the javascript code and without polluting the global namespace.

Spread Operator

Old way	New way
<pre>var array1 = ["hello", true, 3]; var array2 = [1, 2].concat(array1);</pre>	<pre>let array1 = ["hello", true, 3]; let array2 = [1, 2, ...array1];</pre>

The spread operator is an interesting way to build new arrays based on the values of existing arrays. It can:

- **Copy an array**
- **Concatenate arrays**
- **Insert new items into arrays**

7th Edition – ECMAScript 2016

The 7th edition, officially known as ECMAScript 2016, was finalized in June 2016. New features include the exponentiation operator () and Array.prototype.includes.**

Array.prototype.includes

Old way	New way
<pre>let arr = ['react', 'angular', 'vue']; if (arr.indexOf('react') !== -1) { console.log('Can use React'); }</pre>	<pre>let arr = ['react', 'angular', 'vue']; if (arr.includes('react')) { console.log('Can use React'); }</pre>

The `includes()` method determines whether an array includes a certain element or not, returning true or false as accordingly.

Exponentiation Operator

Old way	New way
<code>Math.pow(7,12)</code>	<code>7 ** 12</code>

The exponentiation operator works same as exponentiation in mathematics, it returns the result of raising first operand to the power second operand. For example – 5^{10} or 7^{12}

Conclusion

There are a number of really great language-centric reasons to start writing your code in ES6 and ES7 now, but there isn't enough room in a single blog post to enumerate them or go into the nuances of how they will make your life better.

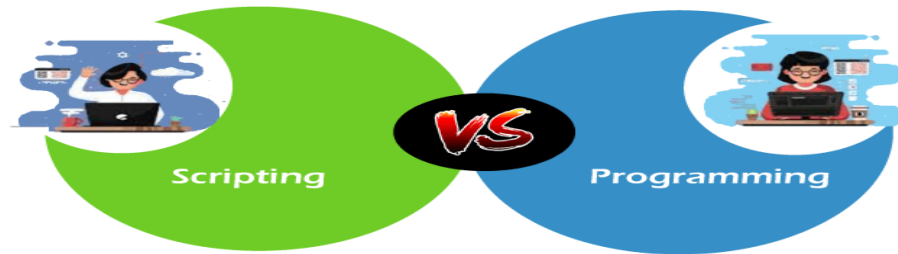
The point isn't *"all the cool kids are doing it."*

The point is that *"a large number of the top libraries used by developers to build applications these days are already written in ES6."*

In my next blog, I will be writing about ES2017 and ES.next features. So stay tuned!!

Scripting Vs. Programming | Difference between Scripting and Programming

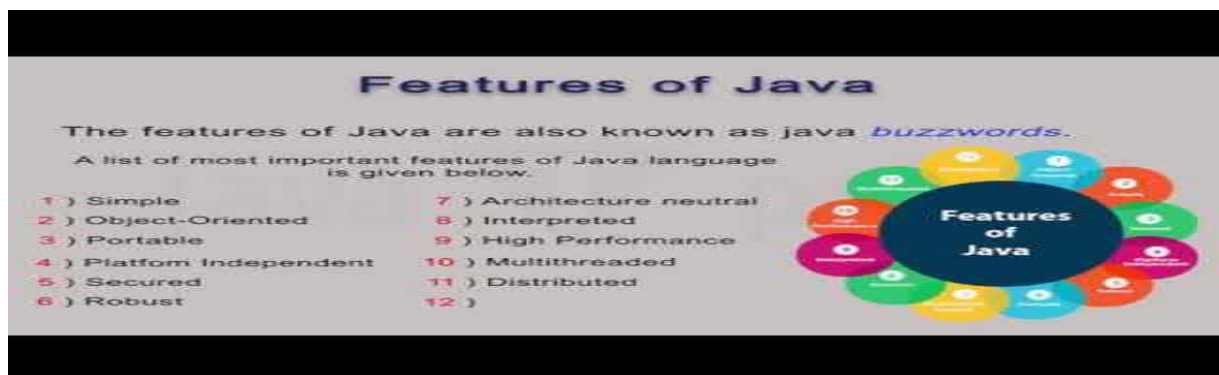
Most Often, people think of the terms scripting and programming as similar to each other and use them synonymously, even though they are very different from each other. However, due to the advancements in computer languages, these differences are becoming blurred and less important, but to be a good programmer, it is very important to understand the basic differences between the two.



Both scripting and programming are computer languages, with all the scripting languages being programming languages, but all programming languages are not scripting languages. *The basic difference between a scripting language and a programming language is that scripting languages do not need an additional step of compilation and rather they are interpreted, whereas programming languages are compiled and hence need a compilation step to convert the high-level language to machine code.*

To better understand the differences, we should understand why scripting languages are evolved? In earlier days, programming languages were built to create software and applications such as *Microsoft Excel, MS Word, Internet Explorer*, etc. But as time passed, the demand to upgrade the programming languages has increased, and programs needed a way to add new functionalities, and hence scripting languages come into existence.

In this topic, we will discuss more details about scripting languages and programming languages, along with their differences.



What is a Programming Language?

A **programming language** is a combination of words and symbols that is used to write programs, and these programs are set of instructions. Therefore, we can say, "*A programming language is a way by which programmers communicate with computers through the set of instructions known as code/program.*" Programming languages are compiled languages, which means the source code is compiled to convert it to machine code.

As we know, computers work on bits (0 and 1) and cannot understand human languages such as English; hence programming languages are implemented. Programming languages are the computer languages that are used in computers to provide instruction and implement algorithms. Each programming language contains its own set of rules for writing the code, and such rules are known as **Syntax**. Thus, to learn and write code in one programming language, we need to know its syntax. These languages enable the developers to create **desktop applications, web applications, mobile applications**, implement machine learning algorithms, and many more tasks. Some popular programming languages are **C++, C, Pascal, COBOL, Java** (*But java is compiled and interpreted as firstly its source code is compiled into byte-code, and then interpreted at runtime*).

Advantages

- These are building blocks for other computer languages.
- These are well suited for large projects.

Applications of Programming languages

- Programming languages are mainly used to create different software and applications such as **MS Excel, PowerPoint**, etc.
- These are used for transforming the data, for example, solving a set of equations from a set of conditions.

What is a Scripting Language?

"A scripting language is a type of programming language which does not require explicit compilation step, and it is designed for a runtime system to automate the execution of tasks." For example, a JavaScript program is not needed to be compiled before we run it. These are also known as very high-level programming languages because of working at a high level of abstraction.

Scripting languages support "**script**," which is small program written for a specific runtime environment. These are interpreted at runtime rather than compiled. It means, to convert the source code to machine code, scripting languages use an interpreter, not the compiler. As scripting language is not compiled so as we write something meaningful, we can run it immediately.

The scripting language refers to dynamic high-level, general-purpose interpreted languages such as Python, Perl, etc. Thus, a scripting language can automate different environments such as *application softwares, webpages, text editors, operating system shells, computer games, etc.*

Advantages

- It is an easy and quick process to learn coding in Scripting language, and for this, much knowledge of web technology is not needed.
- In scripting languages, a wide variety of libraries is available that enable the developers to develop new applications.
- With the help of scripting languages, we can add visualization interfaces and combinations to web pages. Most of the latest web pages need scripting languages for creating enhanced web pages, fascinating UI, and many more.
- There are less number of data structures and variable to be used, which make it highly efficient.

- These are Less code-intensive as compared to traditional programming languages.

Applications of Scripting Language

- These are used to automate a specific task in a program.
- These are useful to extract information from a dataset.

Key differences between Programming and Scripting language

- **Definition**

A *programming language* is a computer language which is used to communicate with computers using a set of instructions.

A *scripting language* is a type of programming language that supports scripts, which are small programs mainly used to automate the execution of a specific function in a specific runtime environment.

- **Interpretation**

Programming languages use compiler and do not require to be interpreted by another language or application; hence these languages run independently and do not depend on the parent program.

In contrast, scripting languages are interpreted within another program; for example, JavaScript has to be combined within HTML, then interpreted by the web browser.

- **Design**

Programming languages are specifically designed to facilitate the developer with complete code and software development, whereas scripting languages are specifically designed to make programming faster and simpler.

- **Development**

Development of software/Application or coding using programming languages is difficult as lots of lines of code is needed for a task. Whereas in scripting languages, coding is easier as it needs only a few lines of code to perform a task.

Therefore, *development time in programming languages is high due to more coding, whereas development time in a scripting language is less due to less coding.*

- **Types/Categorisation**

Programming languages are categorized into mainly five categories:

1. The first generation,
2. Second generation,
3. Third generation,
4. Fourth generation,
5. and Fifth generation languages.

On the other hand, Scripting languages are categorized into two categories

1. **Server-side scripting languages and**
2. **client-side scripting languages.**

- **Conversion into machine code**

As programming languages use a compiler, hence the complete program is converted into machine code in one shot. Whereas Scripting languages use an interpreter, hence the program is converted into machine code line by line.

- **Speed**

The programming languages are faster in speed because of using a compiler, which usually runs faster as it finds all the errors at once after analyzing the program.

In contrast, Scripting languages are slow as they use an interpreter that analyses a program line by line. Every time it detects an error, it stops further execution until the error gets removed.

- **Examples**

Some popular examples of programming languages are C, C++, Java, [Scala](#), COBOL, etc. Some popular examples of Scripting languages are [Perl](#), [Python](#), [JavaScript](#), [PHP](#), [Ruby](#), etc.

Comparison table between Programming Language and Scripting Language

Programming Language	Scripting Language
A programming language is a computer language that is used to communicate with computers using a set of instructions.	A scripting language is a type of programming language designed for a runtime system to automate the execution of tasks.
It is compiled language or compiler-based language.	It is interpreted language or interpreter-based language
It is used to develop an application or software from scratch.	It is used to combine existing components and automate a specific task.
It runs or executes independently and does not depend on the parent (exterior) program.	It runs or executes inside another program.
It uses a compiler to convert source code into machine code.	It uses an interpreter to convert source code into machine code.
As it uses a compiler, hence the complete program is converted into machine code in one shot.	As it uses an interpreter, hence the program is converted into machine code line by line.
These languages are required to be compiled.	There is no need for compilation.

It is comparatively difficult to write code in a programming language, and it requires numerous lines of code for each task.

The development time in programming languages is high as more lines are required.

There is the high maintenance cost.

All programming languages are not scripting languages

It generates a .exe file.

Usually, programming languages do not support or provide very little support for user interface designing, data types, and graphic designing.

Some popular examples are C, C++, Java, Scala, COBOL, etc.

It is comparatively easy to write code in the scripting language, and it requires few lines of code for each task.

The development time in a scripting language as a smaller number of lines are required.

There is less maintenance cost.

All scripting languages are programming languages

It does not create a .exe file.

Scripting languages provide great support to user interface design, data types, and graphic design.

Some popular examples are Perl, Python, JavaScript, etc.