

2020 Stack Overflow Developer Survey (65,000 Developers 186 countries)

Most Popular Technologies

Programming:

- (1) JavaScript
- (2) HTML/CSS
- (3) SQL
- (4) Python
- (5) Java

Web Framework:

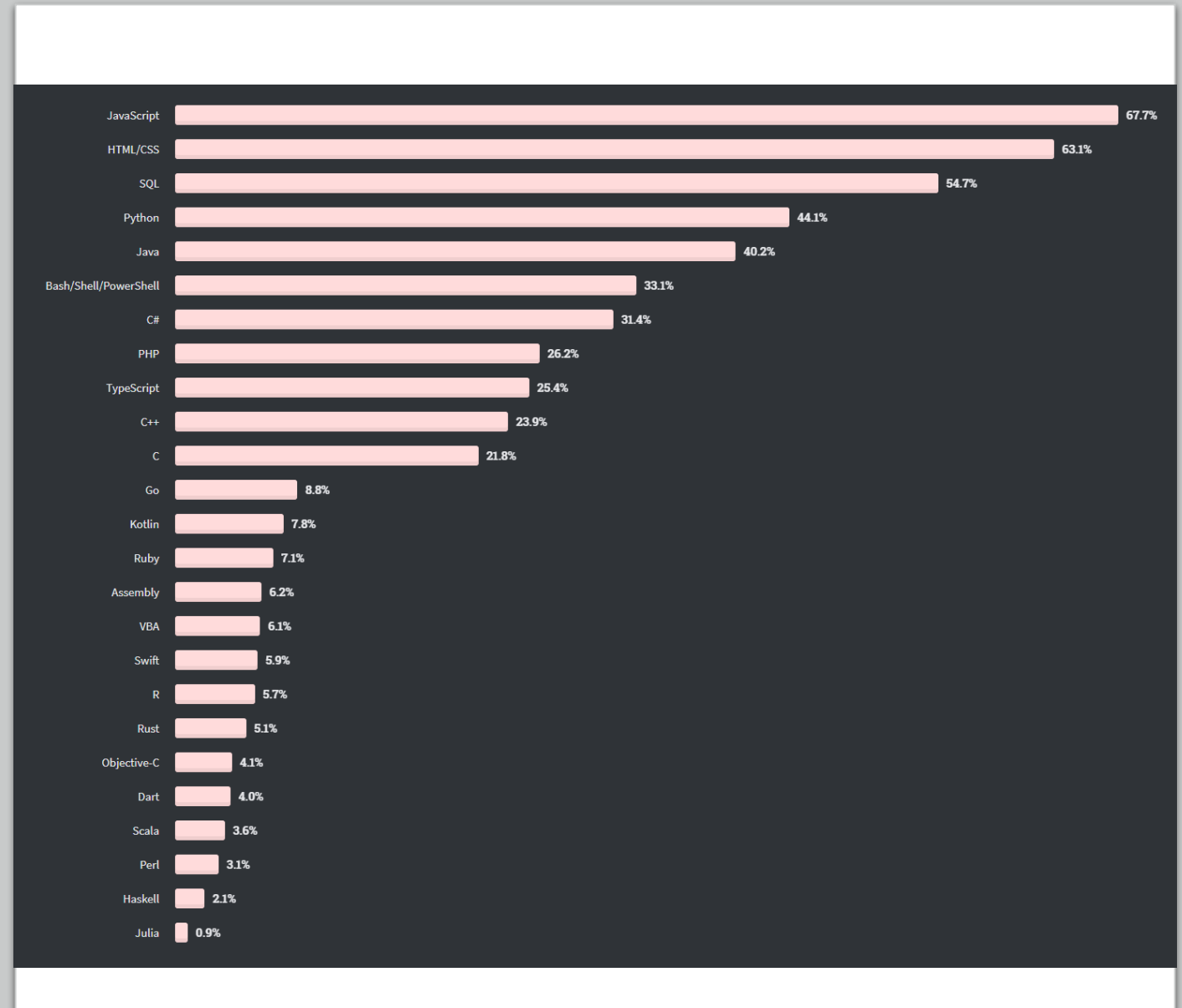
- (1) jQuery
- (2) React.js
- (3) Angular
- (4) ASP.NET
- (5) Express

Frameworks, Libraries, and Tools

- (1) Node.js
- (2) .NET
- (3) .NET Core
- (4) Pandas
- (5) TensorFlow

Database:

- (1) MySQL
- (2) PostgreSQL
- (3) Microsoft SQL Server
- (4) SQLite
- (5) MongoDB



80,000 respondents

Programming, scripting, and markup languages

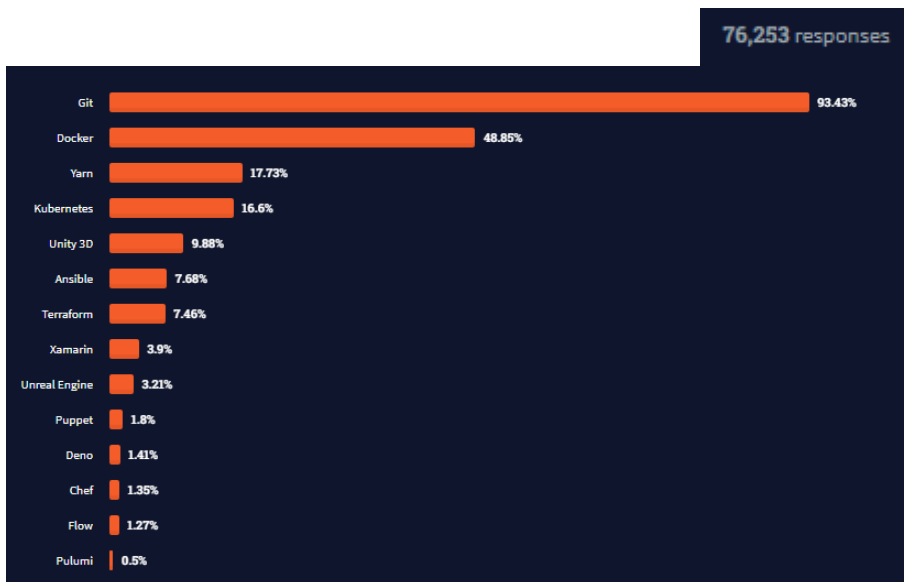
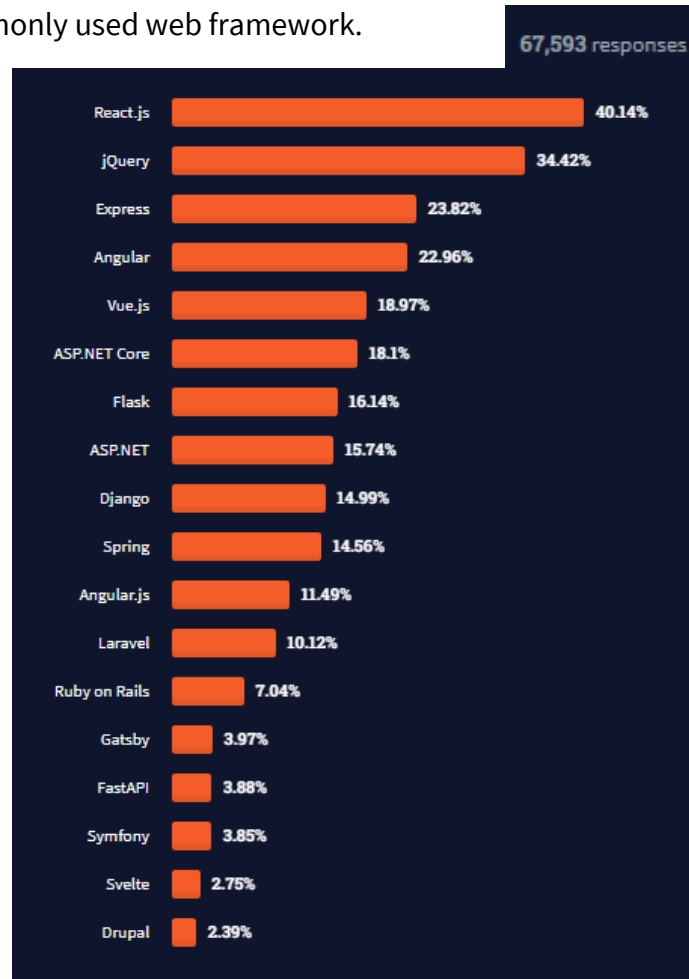
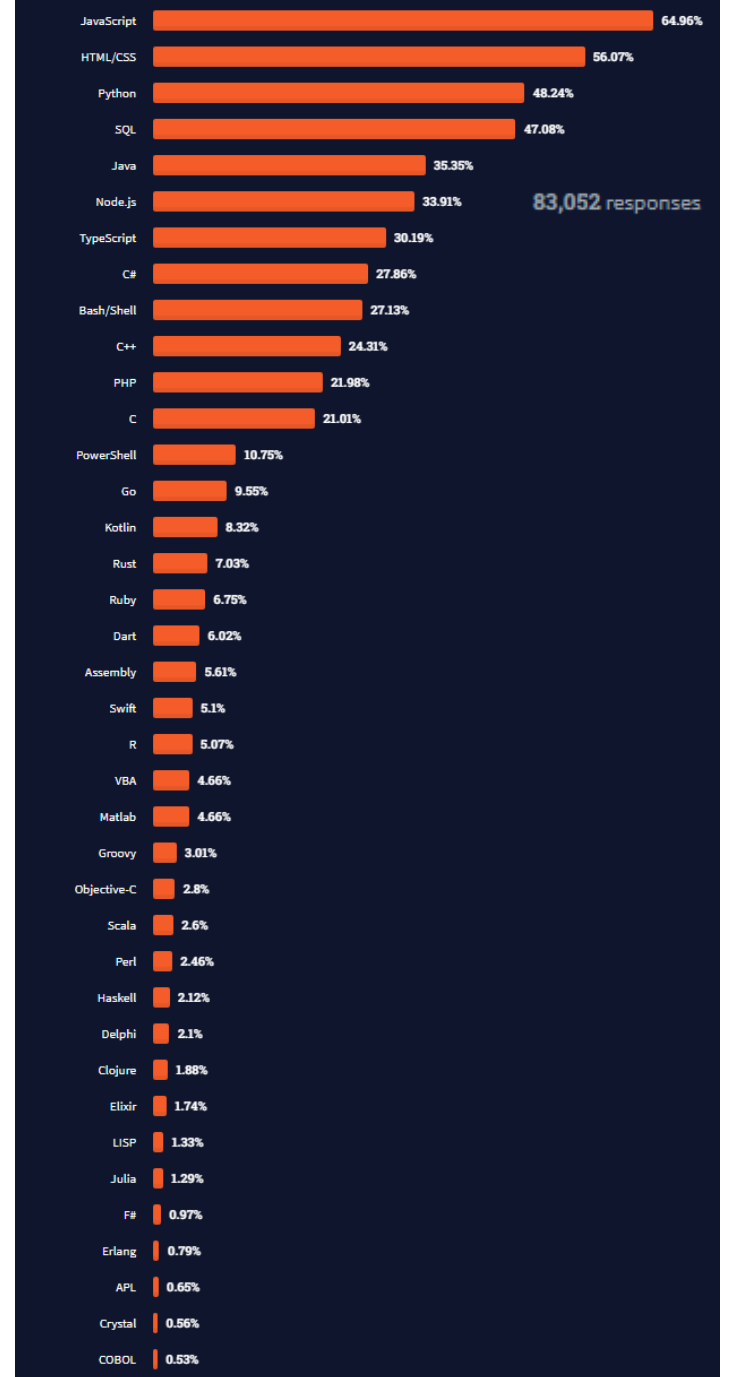
JavaScript completes its **ninth year** in a row as the most commonly used programming language. For most developers, programming is web programming. Python traded places with SQL to become the third most popular language.

Web frameworks

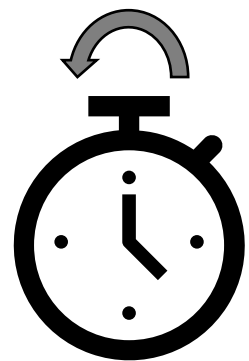
This year, **React.js** surpassed jQuery as the most commonly used web framework.

Other tools

Over 90% of respondents use **Git**, suggesting that it is a fundamental tool to being a developer.



https://insights.stackoverflow.com/survey/2021?_ga=2.244540421.1483398750.1627920327-1168534977.1627920327



JavaScript History

Asst.Prof. Dr. Umaporn Supasitthimethee

ผศ.ดร.อุมภาพร สุภสีทธิเมธี



JavaScript History

- **1995** - JavaScript is a programming language that was created by **Brendan Eich** who was working for *Netscape*.
- **1997** - JavaScript 1.1 proposal was submitted to the European Computer Manufacturers Association (ECMA).

ECMAScript

- The formal specification of the JavaScript language specified in the document ***ECMA-262***
- ***ES1, ES2, ES3,...ESX*** are a different version of the ***ECMAScript*** specification

<https://en.wikipedia.org/wiki/ECMAScript>

* Started from ES6, version of the ECMAScript start naming the versions based on the year of published specification, for example, ES2015 (ES6), ES2016 (ES7), ...

JavaScript

ES5 (2009) is fully supported by most modern browser in early 2016

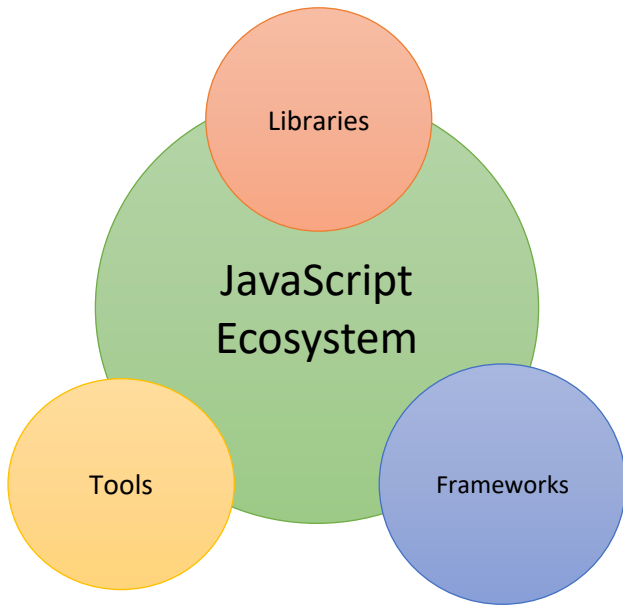
- Higher-order iteration functions (map, reduce, filter, foreach);
- JSON support;
- Better reflection and object properties;

ES6 (ES2015) provide a greatly improved developer experience

- Classes
- Modules
- Iterators
- Generators
- Promises
- Arrow functions

From 2016 to 2019, a new edition of the ECMAScript standard was published each year, but the scope of changes was much smaller than the 5th or 6th editions

ES11 (ES2020), officially known as ECMAScript 2020, was published in June 2020















JavaScript EcoSystem

Asst.Prof. Dr. Umaporn Supasitthimethee

ผศ.ดร.อุมาพร สุขสีทธิเมธี

The different aspects of JavaScript

- Front-End:  React,  Angular,  Vue.js,  svelte,  jQuery
- Back-End:  node.js  Deno
- Web Framework: Express
- Mobile:  React Native,  Apache Cordova  Ionic
- Desktop:  Electron
- Database:  mongoDB MongoDB



JS Introduction to JavaScript

Asst.Prof. Dr. Umaporn Supasitthimethee

ผศ.ดร.อุมาพร สุภสีทธิเมธี



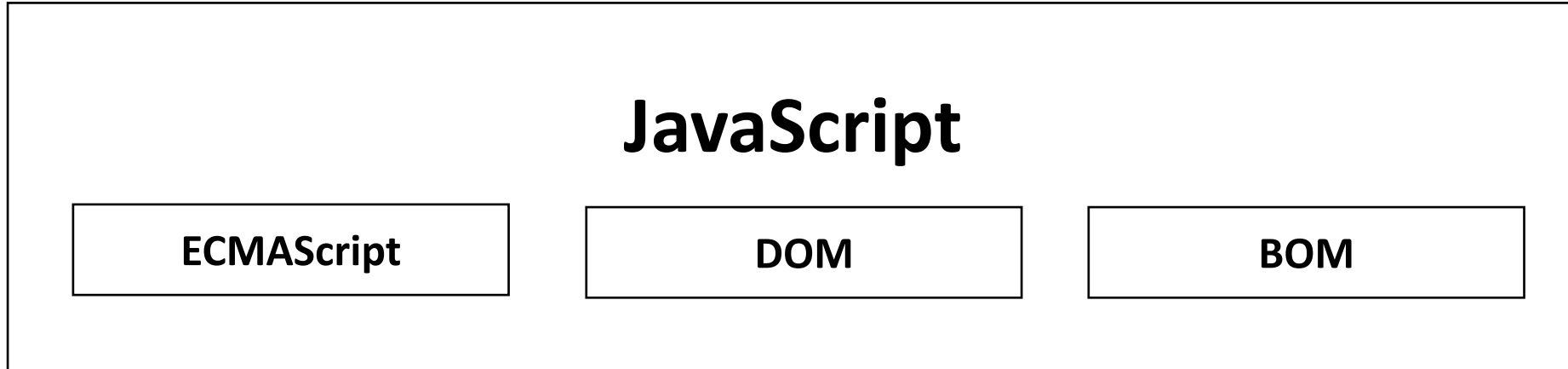


Introduction to JavaScript

- JavaScript is the programming language of the web.
- The overwhelming majority of websites use JavaScript, and all modern web browsers—on desktops, tablets, and phones
- Over the last decade, Node.js has enabled JavaScript programming outside of web browsers, and the dramatic success of Node means that JavaScript is now also the most-used programming language among software developers.
- JavaScript is completely different from the Java programming language.



JavaScript



DOM: The Document Object Model

Map out an entire page as a hierarchy of nodes

BOM: The Browser Object Model

Deals with the browser window and frames

Chromium

open source browser project

Web Browser

Chromium-based browser:  Google Chrome  Microsoft Edge  Opera



Safari is a graphical web browser developed by Apple, based on the WebKit engine.



Mozilla Firefox, or simply **Firefox**, is a free and open-source web browser developed by the Mozilla Foundation and its subsidiary, the Mozilla Corporation. Firefox uses the Gecko layout engine to render web pages.

Chrome V8



open source JavaScript engine project

JavaScript Engine

Chrome V8:  Google Chrome  Microsoft Edge  Opera



JavaScriptCore: A JavaScript interpreter and JIT originally derived from KJS. It is used in the WebKit project and applications such as Safari.



SpiderMonkey: A JavaScript engine in Mozilla Gecko applications, including Firefox.

JavaScript Development Environment

In Web Browser

-  Google Chrome
-  Microsoft Edge
-  Safari
-  Firefox
-  Opera

Outside Web Browser (based on Chrome V8 JavaScript Engine)



Node.js: a JavaScript runtime built on Chrome's V8 JavaScript engine.



Deno: a simple, modern and secure runtime for JavaScript and TypeScript that uses Chrome's V8 and is built in Rust.



Demo JavaScript

In and Outside Web Browser

```
console.log("I am JavaScript.");
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="MyFirstScript.js"></script>
</head>
<body>
  <h1>Hello, This is my HTML page with JavaScript.</h1>
</body>
</html>
```



JavaScript Language Features

- Interpreted Language
- Single Threaded, do one operation at one time
- Dynamically and weakly typed language
- Support Object Oriented Programming (Prototyped-based)



Basic JavaScript Statements

- Semicolon in the end of statement is an optional
 - `let x=10;`
 - `let y=20`
- Statement can take up multiple lines
- Comment
 - `//Single Line Comment`
 - `/* ... */ Single or Multiple Lines Comment`
- Console Printing
 - `Console.log (variable);`



Literals

- 15 // The number twelve
- 1.5 // The number one point two
- "Hello World" // A string of text
- 'Hi' // Another string
- ` "I' am a student", I said ` // Another string
- true // A Boolean value
- false // The other Boolean value
- null // Absence of an object

Escape sequences can be used in JavaScript: \n, \t, \\, \b, ...



Identifiers

- **Identifiers** are used to name constants, variables, properties, functions, and classes and to provide labels for certain loops in JavaScript code.
- A JavaScript identifier **must begin with a letter, an underscore (_), or a dollar sign (\$)**. Subsequent characters can be letters, digits, underscores, or dollar signs. (Digits are not allowed as the first character so that JavaScript can easily distinguish identifiers from numbers.)
- JavaScript is a **case-sensitive language**. This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.



Reserved Words

as	const	export	get	null	target	void
async	continue	extends	if	of	this	while
await	debugger	false	import	return	throw	with
break	default	finally	in	set	true	yield
case	delete	for	instanceof	static	try	catch
do	from	let	super	typeof	class	else
function	new	switch	var			

JavaScript Data Types: numbers, string, boolean, undefined, symbol, object

//script.js

```
let myNum = 0;
console.log(`type of myNum is ${typeof myNum}`);

let myString = 'Good';
console.log(`type of myString is ${typeof myString}`);

let myBool = true;
console.log(`type of myBool is ${typeof myBool}`);

let myUndefined;
console.log(`type of myUndefined is ${typeof myUndefined}`);

let mySymbol = Symbol();
console.log(`type of mySymbol is ${typeof mySymbol}`);

let myNull = null;
console.log(`type of myNull is ${typeof myNull}`);
```

//output

```
type of myNum is number
type of myString is string
type of myBool is boolean
type of myUndefined is undefined
type of mySymbol is symbol
type of myNull is object
```

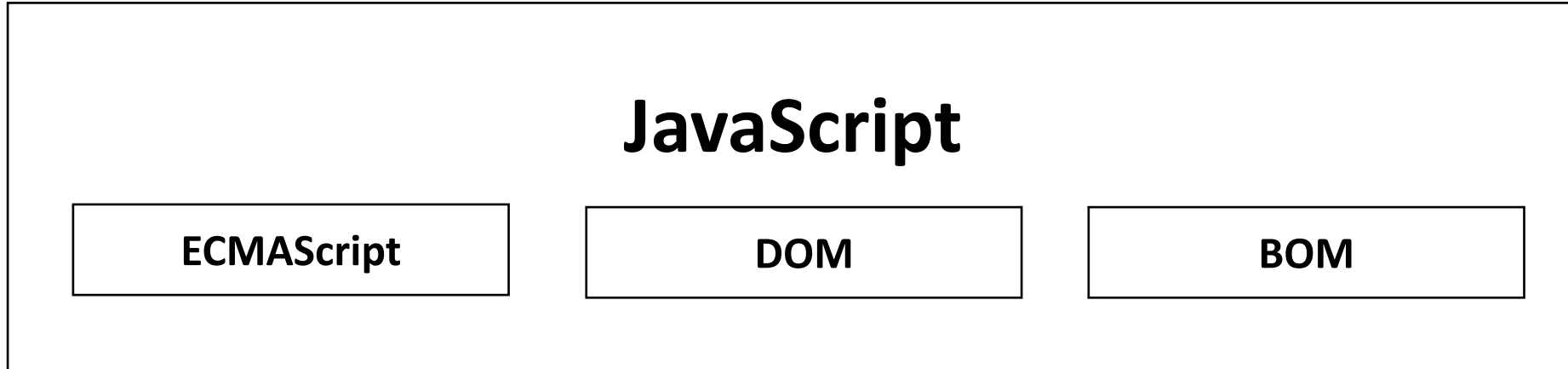


Null and undefined

- `null` is a language keyword that evaluates to a special value that is usually used to indicate the absence of a value.
- Using the `typeof` operator on `null` returns the string `"object"` indicating that `null` can be thought of as a special object value that indicates "no object".
- JavaScript also has a second value that indicates absence of value. The `undefined` value represents a deeper kind of absence. It is the value of variables that have not been initialized and the value you get when you query the value of an object property or array element that does not exist. The `undefined` value is also the return value of functions that do not explicitly return a value and the value of function parameters for which no argument is passed.
- If you apply the `typeof` operator to the `undefined` value, it returns `"undefined"`, indicating that this value is the sole member of a special type.



JavaScript



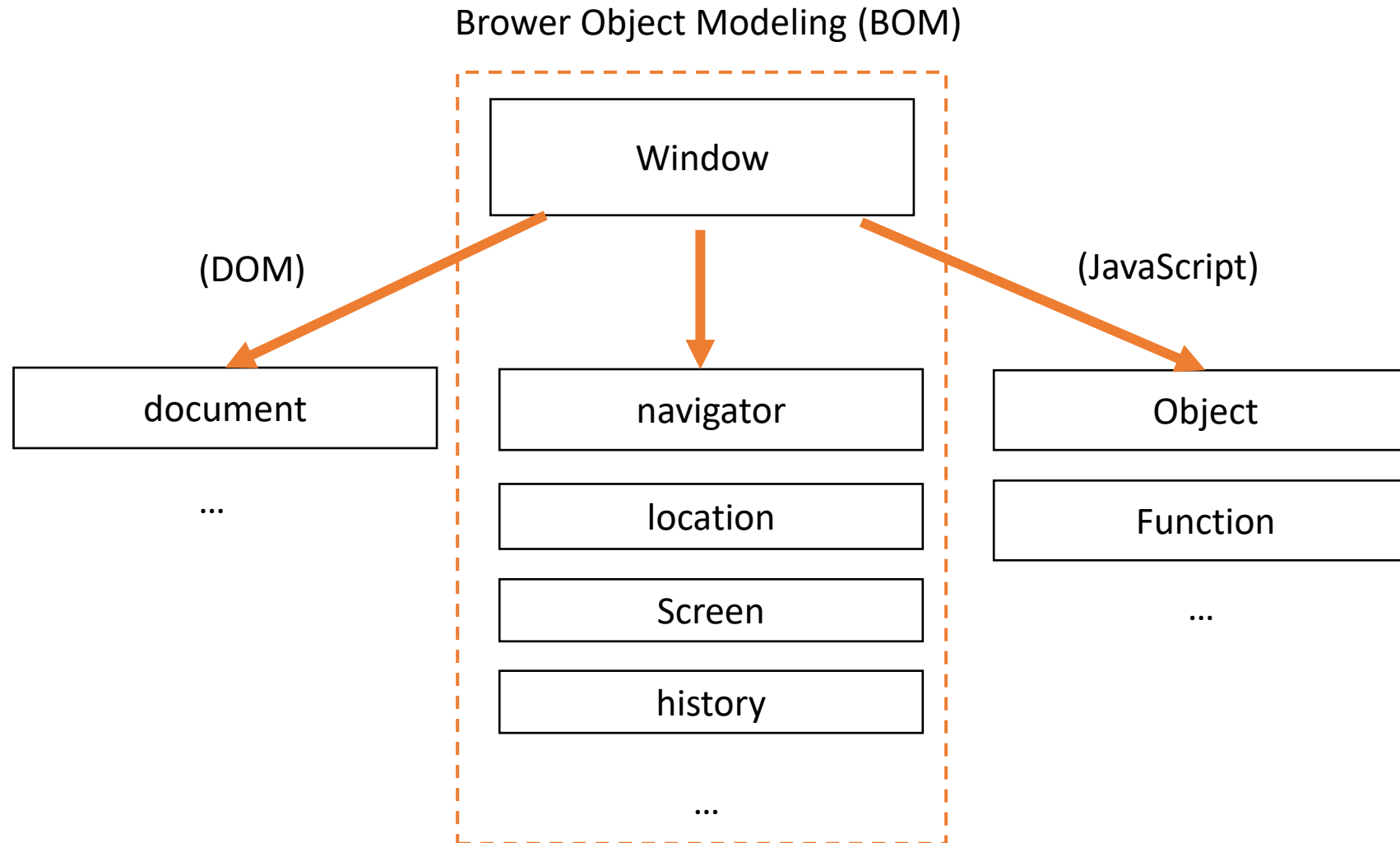
DOM: The Document Object Model

Map out an entire page as a hierarchy of nodes

BOM: The Browser Object Model

Deals with the browser window and frames

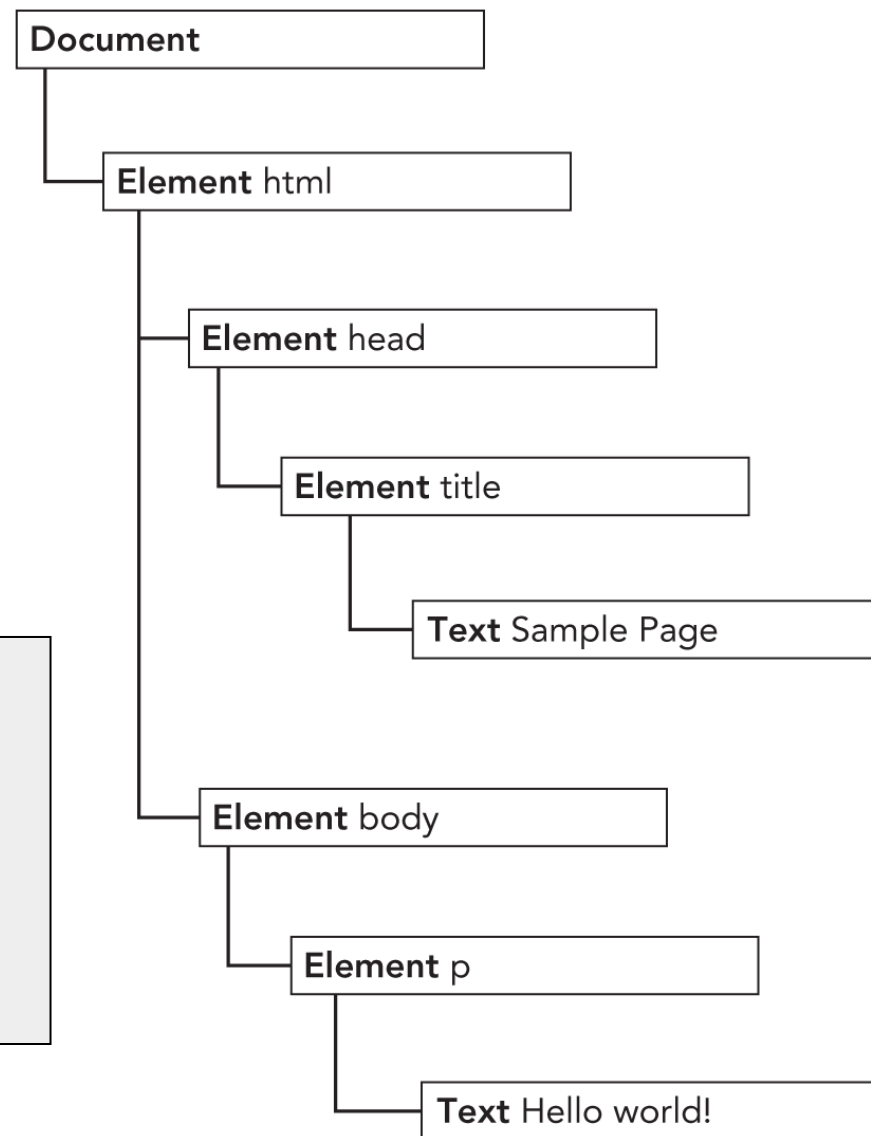
The `Window` interface represents a window containing a DOM document. In a tabbed browser, each tab is represented by its own `Window` object.



DOM: The Document Object Model

```
<html>
  <head>
    <title>Sample Page</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

```
const paragraphs =
document.getElementsByTagName("p");
alert(paragraphs[0].nodeName); //p
alert(paragraphs[0].nodeValue); //null
```





let, var, const variables

- One of the features that came with ES6 is the addition of `let` and `const`, which can be used for variable declaration.
- `var` declarations are globally scoped or function/locally scoped.
- The scope is global when a `var` variable is declared outside a function. This means that any variable that is declared with `var` outside a function block is available for use in the whole window.
- All variables and functions declared globally with `var` become properties and methods of the **window** object.
- `var` is function scoped when it is declared within a function. This means that it is available and can be accessed only within that function.

var variables

//script5.js

```
//greeting is globally scope, it exists outside a function
var greeting = 'Hey';

//var variables can be re-declared and updated
var greeting = 'Ho Ho';

function greeter() {
  //msg is function scoped, we cannot access the variable msg outside of a function
  var msg = 'hello';
}

// console.log(msg); //error: msg is not defined
console.log(greeting);
```

var variables can be re-declared and updated

This means that we can do this within the same scope and won't get an error.

```
var year = 'leap';
if (year === 'leap')
  var greeting = 'Hey 366 days'; //re-declared
console.log(greeting);
```

It becomes a problem when you do not realize that a variable greeting has already been defined before.



let variables

- `let` is now preferred for variable declaration.
- JavaScript block of code is bounded by `{}`. A block lives in curly braces. Anything within curly braces is a block.
- `let` is block scoped, a variable declared in a block with `let` is only available for use within that block.
- `let` can be updated but not re-declared.



let can be updated but not re-declared. `//script6.js`

```
/*let variables*/
//greeting is block scope,
let greeting = 'Hey';
//let variables cannot be re-
declared, only can be updated
greeting = 'Ho Ho';
function greeter() {
    //msg is function scoped, we cannot access the
    variable msg outside of a function
    let msg = 'hello';
}
// console.log(msg); //error: msg is not defined
console.log(greeting);

let year = 'leap';
if (year === 'leap')
    greeting = 'Hey 366 days';
console.log(greeting);
```

if the same variable is defined in different scopes, there will be no error. This is because both instances are treated as different variables since they have different scopes.

```
let greeting = 'Hey';
greeting = 'Ho Ho';
function greeter() {
    let greeting = 'Good morning';
    console.log(`greeting in function is ${greeting}`);
}

greeter();
console.log(greeting);
```



const

- Variables declared with the `const` maintain constant values.
- `const` declarations share some similarities with `let` declarations.
- Like `let` declarations, `const` declarations can only be accessed within the block they were declared.
- `const` cannot be updated or re-declared
- Every `const` declaration, therefore, must be initialized at the time of declaration.

//script7.js

```
/*const variables*/  
const greeting = 'Hey';  
//const variables cannot be re-declared  
// const greeting = 'Ho Ho';  
//const variables cannot be updated  
// greeting = 'Hi Hi';
```



Asynchronous vs. Synchronous Programming

- **Synchronous** tasks are performed one at a time and only **when one is completed, the following is unblocked**. In other words, you need to wait for a task to finish to move to the next one.
- **Asynchronous** software design expands upon the concept by building code that allows a program to ask that a task be performed alongside the original task (or tasks), **without stopping to wait for the task to complete**. When the secondary task is completed, the original task is notified using an agreed-upon mechanism so that it knows the work is done, and that the result, if any, is available.

Higher-Order Functions

A “higher-order function” is a function that accepts functions as parameters and/or returns a function.

- JavaScript Functions are **first-class citizens**
 - be assigned to variables (and treated as a value)
 - be passed as an argument of another function
 - be returned as a value from another function

```
//1. store functions in variables
```

```
function add(n1, n2) {  
  return n1 + n2  
}  
let sum = add  
  
let addResult1 = add(10, 20)  
let addResult2 = sum(10, 20)  
  
console.log(`add result1: ${addResult1}`)  
console.log(`add result2: ${addResult2}`)
```

//script2.js

```
//2. returned as a value from another function
```

```
function operator(n1, n2, fn) {  
  return fn(n1, n2)  
}
```

```
//3. Passing a function to another function
```

```
function multiply(n1, n2) {  
  return n1 * n2  
}  
  
let addResult3 = operator(5, 3, add)  
let multiplyResult = operator(5, 3, multiply)  
  
console.log(`add result3 : ${addResult3}`)  
console.log(`multiply result: ${multiplyResult}`)
```

Asynchronous Callback Functions

In JavaScript, a **callback function** is a function that **is passed into another function as an argument**.

This function can then be invoked during the execution of that higher order function.

Since, in JavaScript, functions are objects, functions can be passed as arguments.

//script4.js

```
console.log('Hello');  
  
setTimeout(function () {  
    console.log('JS');  
}, 5000);  
  
console.log('Bye bye');
```

//Console

Hello
Bye bye

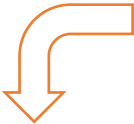
//until 5 seconds
JS

[setTimeout\(\)](#) executes a particular block of code once after a specified time has elapsed.

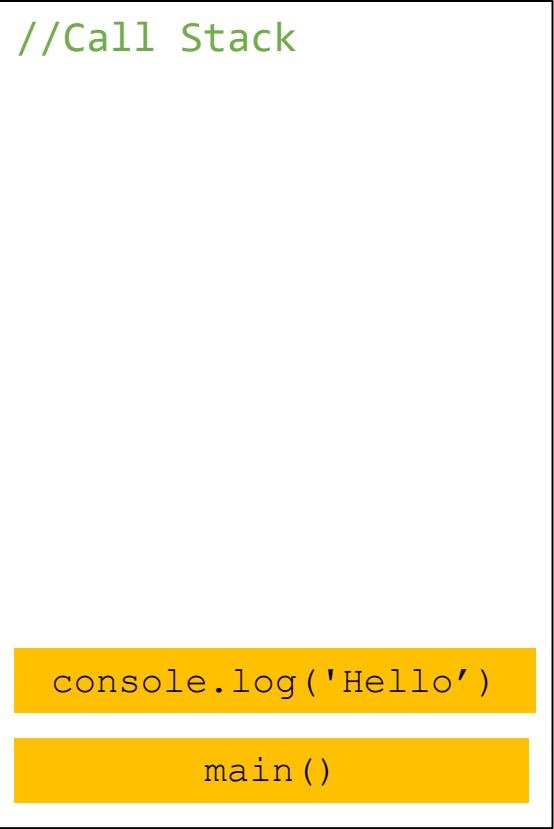


```
console.log('Hello');
setTimeout(function () {
  console.log('JS');
}, 5000);
console.log('Bye bye');
```

```
//Console
Hello
Bye bye
//until 5 seconds
JS
```



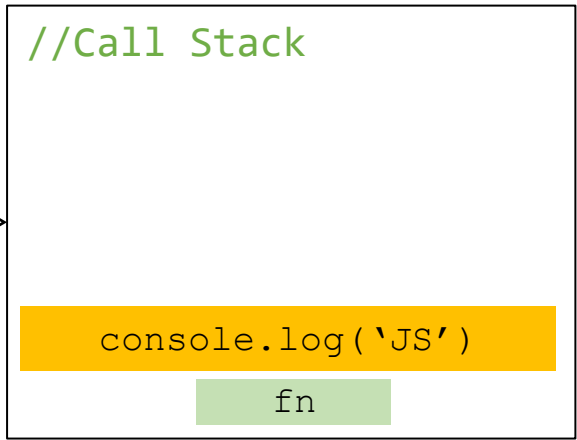
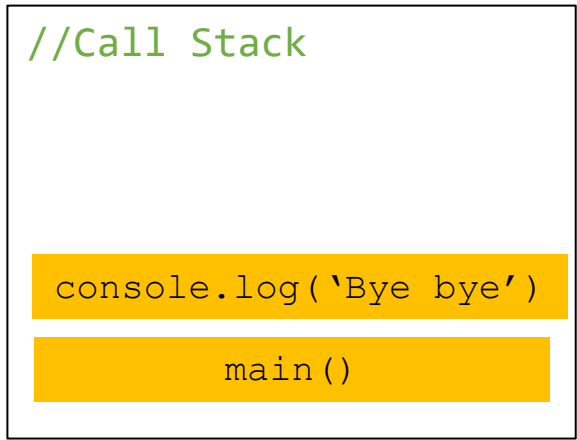
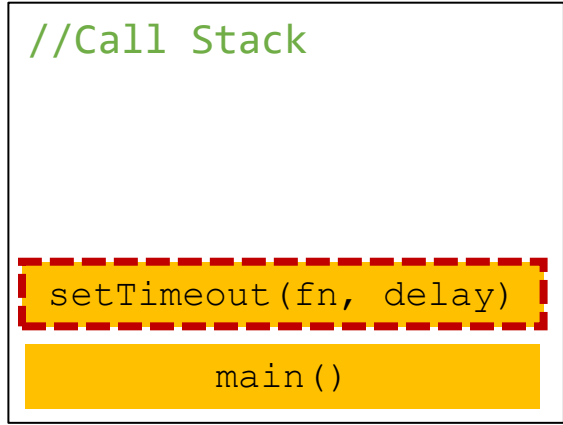
```
console.log('Bye bye')
```



**with Single thread,
JavaScript Runtime
cannot do a
setTimeout while
you are doing
another code**

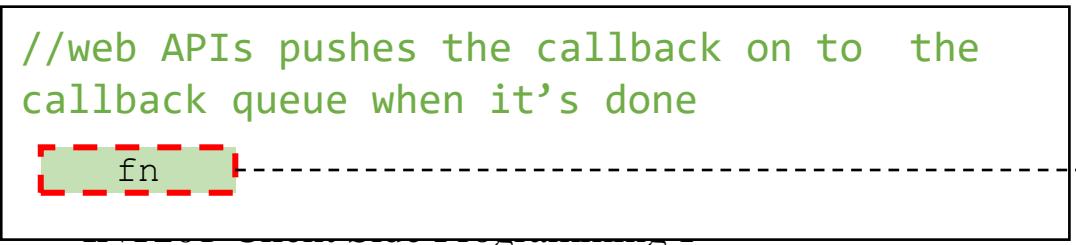
```
console.log('Hello');
setTimeout(function () {
  console.log('JS');
}, 5000);
console.log('Bye bye');
```

```
//Console
Hello
Bye bye
//until 5 seconds
JS
```



Event loop comes in on concurrency, look at the stack and look at the task callback queue. If the stack is empty it takes the first thing on the queue and pushes it on to the stack

callback queue





Vanilla JavaScript

“**Vanilla JavaScript**” is just plain or pure JavaScript without any additional libraries or framework