

JavaScript.

The Fundamentals

Topics

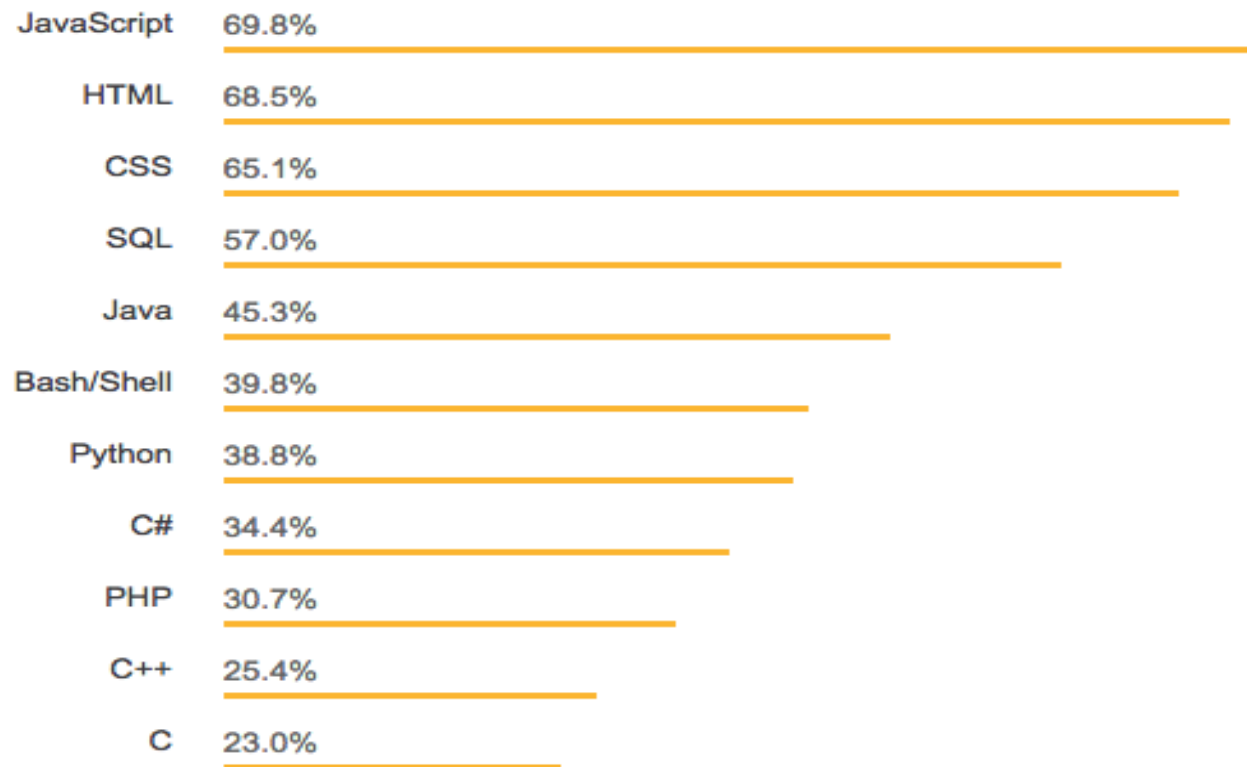
- **Background**
- **Data (State) representation**
 - **All about objects**
- **Behaviour (Logic) representation**
 - **All about functions**

Ref. <https://insights.stackoverflow.com/survey/2018?>

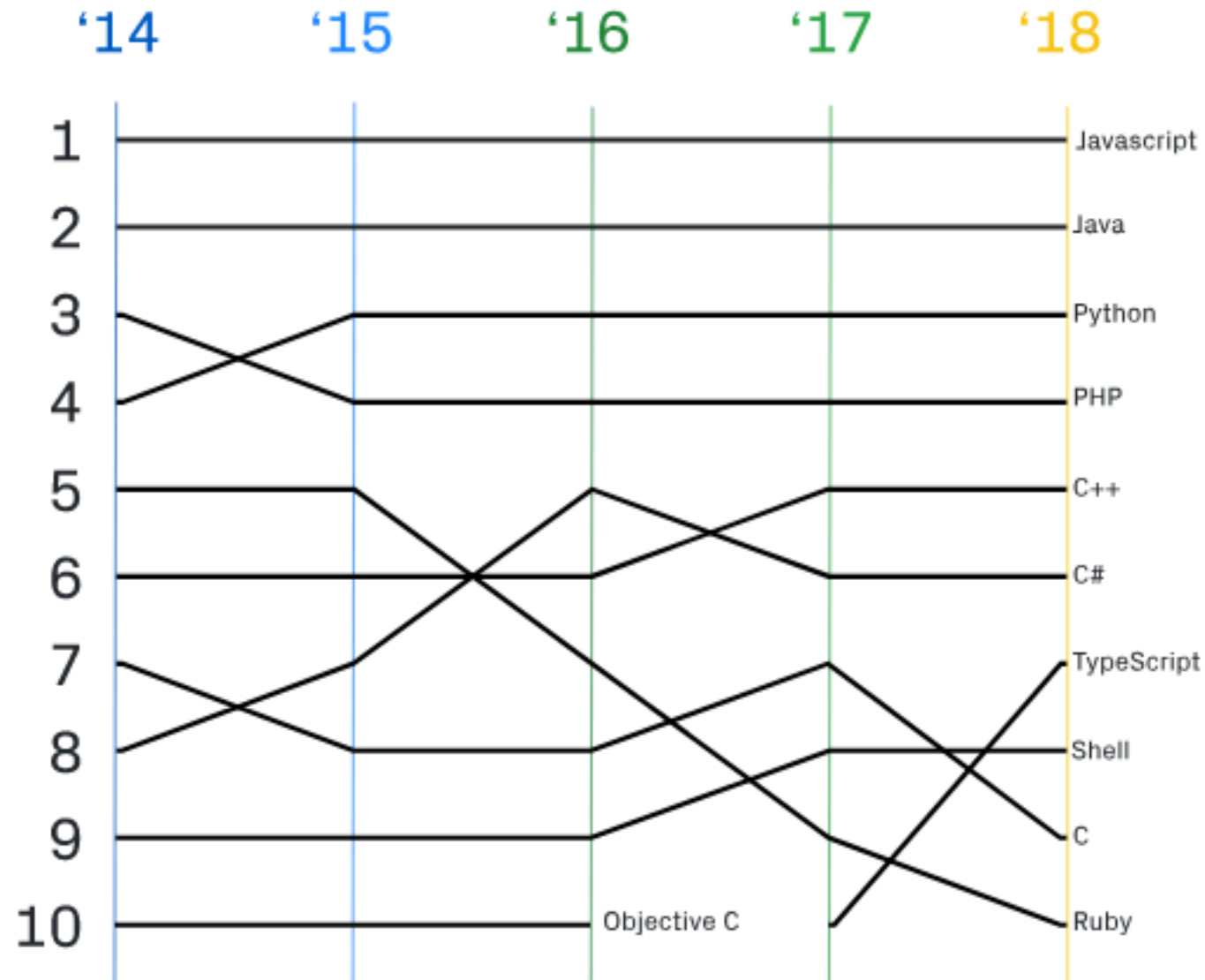
Programming, Scripting, and Markup Languages

All Respondents

Professional Developers



Ref : <https://octoverse.github.com/projects>



Background.

- **Designed by Brendan Eich, at Netscape Corp. (early 1990s).**
 - **Influenced heavily by Java, Self and Scheme.**
- **Named JavaScript to capitalizing on Java's popularity.**
- **Netscape submitted JavaScript to ECMA for Standardization. (ECMA – European Computer Manufacturers Association. Organization that standardizes information)**
- **Resulted in new language standard, known as ECMAScript.**
 - **JavaScript is an implementation of ECMAScript standard.**
 - **ES1: June 1997; ES2: June 1998; ES3: Dec. 1999; ES4: Abandoned**
 - **ES5: 2009; ES6: 2015 (ES2015); ES2016/7**
- **The node.js platform (2009).**
 - **JavaScript on the server-side**
- **Douglas Crockford – 'JavaScript - Volume 1: The Early Years'**

Transpilation (using Babel)

- **Older Browsers cannot execute ES6+ JavaScript.**
 - **Must transpile code first.**
- **Newer browsers incrementally adopting ES6+.**
 - **Same for Node.js platform.**
- **The Babel tool suite.**
 - **One-stop shop for all transpilation needs.**

JavaScript - Data representation.

JavaScript Data Types.

- **Data types:**
 1. **Primitives:** number, string, boolean, null, undefined.
 2. **Everything else is an object.**
- **JS is a dynamically typed language.**

Primitive types.

- Suppose we have a file *01_primitives.js*:

```
1  /*
2  |   Primitive data types in JS
3  */
4  let foo1 = 5 ;
5  let foo2 = 'Hello' ;
6  let foo3 = true
7  let foo4 = null
8  const foo5 = 22 // Constant
9  console.log( foo1 + ' ' + foo2 + ' '
10 | | | | | + foo3 + ' ' + foo4) ;
11  foo1 = 3 ;    // Reassign; Drop let keyword.
12  foo2 = 10 ;   // JS is dynamically typed.
13  let foo6 ;
14  console.log (foo5) ;
```

- Execute it from the command line using node.js platform – no browser needed.

```
diarmuidoconnor@samplecode $ node 01_primitives.js
5 Hello true null
undefined
diarmuidoconnor@samplecode $
```

Primitive types (Basic syntax).

```
let foo = 20 ;
```

- **let** – keyword to indicate we are declaring ‘something’ (and assigning it a literal value in above case).
 - **Use** `const` when declaring constants (cannot reassign).
- **Identifier** – ‘foo’ is an identifier for the thing being declared.
 - **Lots of rules** about valid format for identifiers (no spaces, don’t start with numeric character, etc)
- **Operator** – e.g. `+`, `=`, `*` (multiply), `-`, `[]` (subscript) etc
 - **Some rules** about where they can appear in a statement.
- **Semicolon (;)** – statement terminator.
 - **Optional.**
 - **Babel** puts them back in - ASI.
 - **Avoid multiline expressions.**

let & const

- **let** – **Declared variable CAN be reassigned**
- **const** – **Declared variable CANNOT be reassigned.**
 - **A Constant.**
 - **Use to clarify intent.**
 - **MUST be initialized on declaration**
- **Both have block scope.**
 - **{ } encloses a block, e.g. for-loop, function, class**
 - **Same as Java**

Objects.

- **The fundamental structure for representing complex data.**
- **A unit of composition for data (or STATE).**
- **Literal syntax:**
$$\{ \langle \text{key1} \rangle : \langle \text{value1} \rangle, \langle \text{key2} \rangle : \langle \text{value2} \rangle, \dots \}$$
- **Objects are a set of key-value pairs, termed properties.**
 - **Key (property name) - an identifier; must be unique within the object structure.**
 - **Value - can be a primitive value, another object (nesting) or array.**

Example:

```
let me = { firstName: "Diarmuid", lastName: "O'Connor" } ;
```

Manipulating Object properties.

- **Two notations:**
 1. **Dot notation e.g** `let fn = me.firstName ;`
 2. **Subscript notation e.g.** `me['firstName']` (Note quotes)
- **Same notations for changing a property value:**
`me.firstName = 'Jeremiah';`
`me['lastName'] = 'O Conchubhair' ;`
- **Subscript notation allows a variable reference as the key:**
`let foo = 'lastName';`
`console.log ('Surname: ' + me[foo]) ;`
`me[foo] = ;`
- **Ref. 02_objects.js**

Object characteristics.

- **Objects are dynamic:**
 - **Properties can be inserted and removed at run-time (JS is dynamic).**
 - **Ref. sample 03,**
- **Nested objects.**
 - **A property value may be an object structure.**
 - **Ref. sample 04_1,**

Object characteristics.

- A property value can be a variable reference.
 - Ref. sample 04_2,
- BY THE WAY:
 - Objects declared with **const** **ARE MUTABLE**.
 - **const** variable cannot be reassigned, but they are mutable.
 - Internally JS stores object keys as strings.
 - Hence the subscript notation – me['address'].

Array data structure.

- **Dfn.: Array is an ordered list of values.**
 - **An object's properties are not ordered.**
- **Literal declaration syntax :**
[<value1>, <value2>,]
- **Values can be of mixed** type (may reflect bad design!).
- **Access elements with subscript notation.**
 - **Subscript termed an index.**
- **Ref sample 05**

Array data structure.

- **In JS, arrays are really just ‘special’ objects.**
 - **Index converted to a string for subscript notation:**
`nums[2] becomes nums['2']`
- **Array objects have special properties built-in:**
 - **Length property, e.g. `let len = nums.length; // 4`**
 - **Utility methods for manipulating elements e.g push, pop, shift, unshift, join etc.**

Nested collections.

- **Arrays and objects are collection types.**
- **They can be nested.**
- **Ex.:**
 - **An array where elements are also arrays** - `array_outer[3][2]`
 - **An array of objects** - `array_outer[1].propertyX`.
 - **An object with a property whose value is an array** - `objectY.propertyX[5]`.
 - **Etc.**

JavaScript - Behavior structures

JavaScript functions.

- **Fundamental unit of composition for logic (or BEHAVIOUR).**
function <func_name>(<parameters>) {
 <body of code>
} // end of function
 - **Some functions don't require parameters.**
 - **Function body is executed by calling it with arguments.**
 - **Declarations are “hoisted” to the top of the current scope.**
 - **Function call can appear before its declaration**
- **Ref. sample 06.**

JavaScript functions.

- **Functions can be created using:**
 1. **A declaration (above examples).**
 2. **An expression.**
 3. **A method (of a custom object).**
 4. **Anonymously.**
- **Can be called as:**
 1. **A function (above examples).**
 2. **A method of an object.**
 3. **A constructor.**

Function Expressions.

- **Defined using the syntax:** `let name = function(...) { ... }`
- **Invoked the same way as function declarations:**
`name(argument1, argument2, ...) ;`
- **No “hoisting”.**
- **Useful for dynamically created functions.**
- **Ref sample 6**

Function Result.

- **Typically, functions perform some logic AND return a result.**
- **A function with no explicit return statement, returns 'undefined'.**
- **Return type can also be a function.**
 - **Termed Higher Order Functions (HOF)**

Methods.

- **Method** – A function associated with an object property.
- **Syntax:** let objX = {
 methodY : function(....) { },
 };
- **Invoking a method:** objX.methodY(....)
- **The ‘this’ special variable** – used by a method to access other properties of the containing object
- **Syntax comparison:**
 - **Function:** computeTotal(person)
 addMiddleName(person,'Paul')
 - **Method:** person.computeTotal()
 person.addMiddleName('Paul')

Anonymous functions.

- **A function without a name:** `function(...) { }`
- **Mainly used for “callbacks”** – when a function appears as a parameter for another function, which the latter calls.
 - Also termed Higher Order Functions (HOF)
 - Very common pattern.
- **Simple Example:**
 - The *forEach* array method.
 - More elegant way of processing an array than for-loops.
 - The `setTimeout` system function.
- **Be careful using ‘this’ inside an anonymous function.**
- **[Any type of function (declaration, expression) can be used as a callback, not just anonymous functions.]**

Constructor functions.

- **The object literal syntax is not efficient for** creating multiple objects of a common ‘type’.
 - **Efficiency = Amount of source code.**

```
let customer1 = { name 'Joe Bloggs',  
  address : '1 Main St',  
  finances : { . . . . . },  
  computeTotal : function () { . . . . . },  
  adjustFinance : function (change) { . . . }  
};  
let customer2 = { name 'Pat Smith',  
  address : '2 High St',  
  finances : { . . . . . },  
  computeTotal : function () { . . . . . },  
  adjustFinance : function (change) { . . . }  
};  
let customer3 = . . . . .
```

Constructors solve
this problem

Constructors.

- **Constructor** - Function for creating an object of *a custom type*.
 - Custom type examples: Customer, Product, Order, Student, Module, Lecture.
 - Idea borrowed from class-based languages, e.g. Java.
 - No classes in Javascript (prior to ECMAScript 6 - June 2015)
- **Convention: Capitalize constructor name to distinguish it from 'ordinary' functions.**
function Customer(. . .) { }
- **Constructor invocation must be preceded by the new operator.**
let customer1 = **new** Customer(.);

Constructors.

- **What happens when a constructor is called (with new) ?**
 - 1. An empty object is created, ie. { }.**
 - 2. The this variable is set to this object.**
 - 3. The constructor function is executed, as normal.**
 - 4. The (default) return value is the object referenced by this.**

Summary

- **Representing Data / State**
 - **Primitives.**
 - **Objects.**
 - **Dynamic, nested.**
 - **Arrays.**
- **Defining Behaviour.**
 - **Functions:**
 - **Declarations.**
 - **Expressions.**
 - **Anonymous.**
 - **Constructor.**

