

JavaScript.

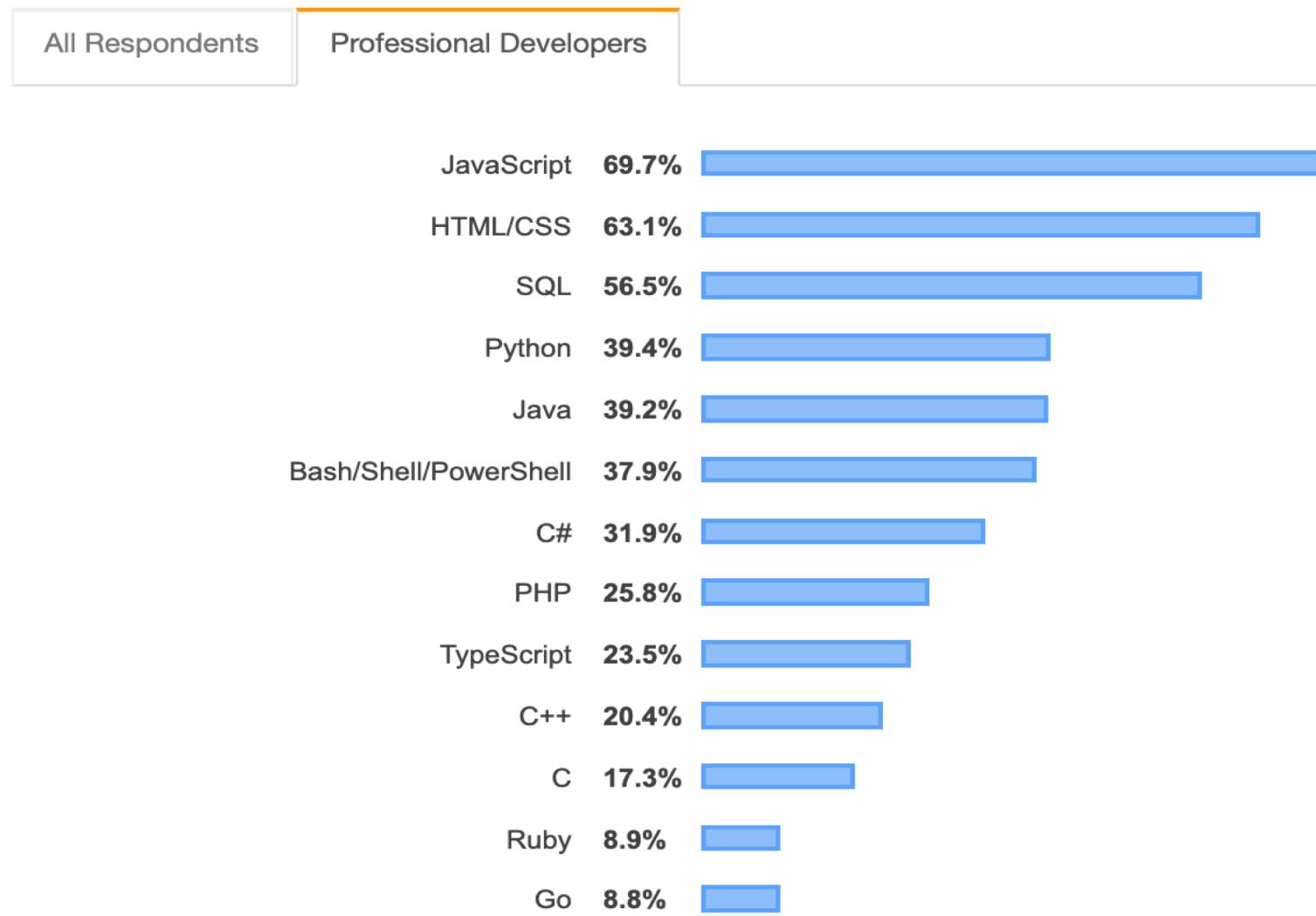
The Fundamentals

Topics

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

Ref. [https://insights.stackoverflow.com/survey/2019?](https://insights.stackoverflow.com/survey/2019/)

Programming, Scripting, and Markup Languages



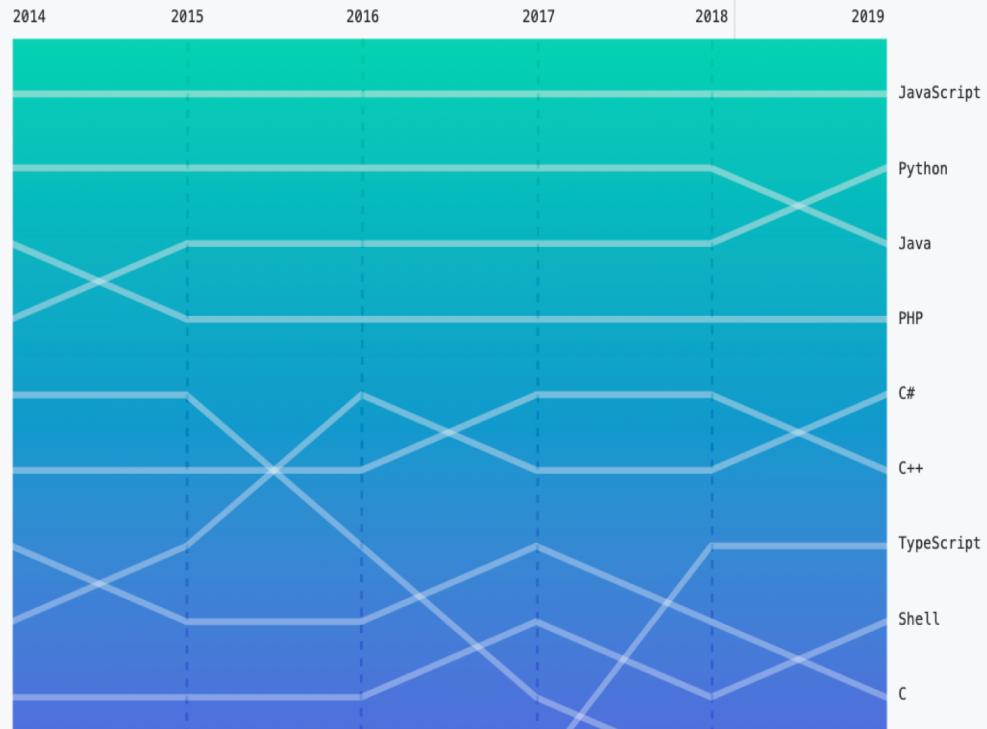
Ref : <https://octoverse.github.com/#top-languages>

Top languages

Top languages over time

This year, C# and Shell climbed the list. And for the first time, Python outranked Java as the second most popular language on GitHub by repository contributors.*

In the last year, developers collaborated in more than 370 primary languages on GitHub.



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.

Demo setup

The screenshot shows the VS Code interface with the following elements:

- EXTENSIONS** tab in the top left.
- Extension: Live Server — data** tab in the top right.
- Search Extensions in Microsoft Store** input field.
- ENABLED** section:
 - Git History 0.4.12** by Don Jayamanne (8 updates)
 - Live Server 5.6.1** by Ritwick Dey (Launch a development local Server with live reload f...)
 - Prettier - Code formatter 3.18.0** by Esben Petersen (Code formatter using pr...)
- Install the Live Server extension** callout box with an orange arrow pointing to the **Live Server** extension entry in the list.
- Live Server** extension details:
 - Ritwick Dey** | **3,627,771** | ★★★★★
 - Launch a development local Server with live reload f...
 - Disable** ▾ **Uninstall** *This extension is enabled globally.*
- EXPLORER** view showing a folder named **DATA** containing files:
 - JS 01_primitives.js**
 - JS 02_objects.js**
 - JS 03_dynamic_objects.js**
 - JS 04_1_nested_objects.js**
 - JS 04_2_nested_objects.js**
 - JS 05_arrays.js**
 - index.html**
- OUTLINE** view at the bottom.
- Go Live** button at the bottom right.
- 1** notification badge in the bottom left corner.

Opens a browser tab and loads index.html from current workspace

- Ref data/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) ;

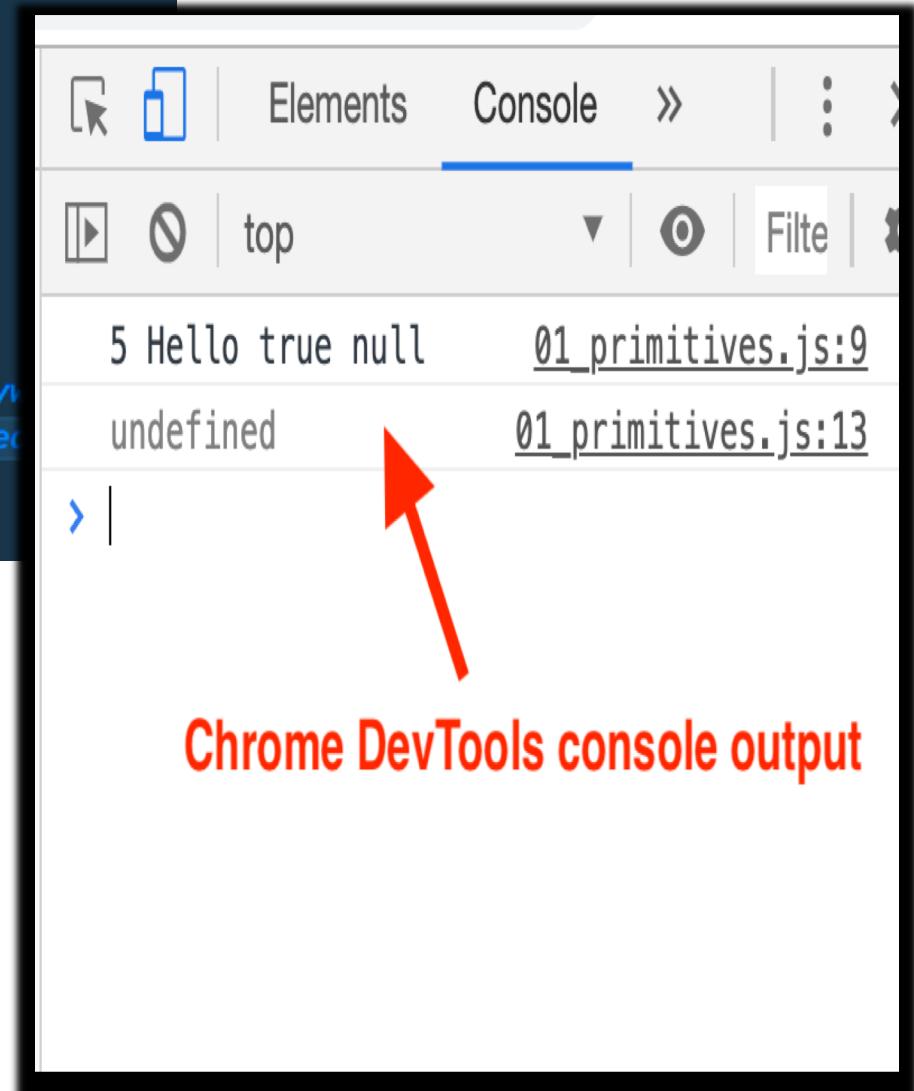
```

```

index.html ×
index.html > html > body > script
1  <!DOCTYPE html>
2  <html>
3  >  <head>...
5  </head>
6  <body>
7    <h1>JS Data</h1>
8    <script src = "./01_primitives.js"></script>
9  </body>
10 </html>

```

Primitive types.



Primitive types (Basic syntax).

```
let foo1 = 5 ;
```

- **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. +, =, *, -, [] (subscript) etc
 - Some rules about where they can appear in a statement.
- Semicolon (;) – statement terminator.
 - **Optional**.
 - **Babel puts them back in - ASI**.
 - **When omitted, be careful with multi-line 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, if, function, class
 - Same as Java

Objects.

- The fundamental structure for representing complex data.
- A unit of composition for data (or STATE).
- Literal syntax:

{ <key1> : <value1>, <key2> : <value2>, }

e.g.

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

- 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) , array or function.

Manipulating Object properties.

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

Object characteristics.

- **Objects are dynamic:**
 - Properties can be inserted and removed at run-time (JS is dynamic).
 - [**Ref. 03_dynamic_objects.js**](#),
- **Objects can be nested.**
 - A property value may be an object structure.
 - [**Ref. 04_1_nested_objects.js**](#)
- **A property value can be a variable reference.**
 - [**Ref. 04_2_nested_objects.js**](#)

Object extras.

- **Objects declared with `const` ARE MUTABLE.**
 - `const` cannot be reassigned, but its internal ‘value’ is mutable.
- **`Object.keys(objRef)` – get all keys in an object structure.**
- **`Object.values(objRef)` – get all values in an object structure.**
- **The ‘in’ operator – Does an abject have a certain key? e.g. ‘name’ in me**
- **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 05_arrays.js**

Array data structure.

- In JS, arrays are really just ‘special’ objects.
 - Index converted to a string for subscript notation:
nums[2] becomes nums[‘2’]
- An array ‘objects’ has special properties built-in:
 - length property, e.g. const len = nums.length
 - 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.**

String templates (ES6)

- **String concatenation (ES5):**

```
console.log( foo1 + '' + foo2 + '' + foo3 + '' + foo4) ;
```

- **String template:**

```
console.log(` ${foo1} ${foo2} ${foo3} ${foo4} `) ;
```

- **Use backquote (`) to enclose template, not single quote.**
- **Interpolation: Embed variable / expressions using \${ }.**
 - Expression is evaluated and result inserted into string
- **Multi-line strings.**

String templates (ES6)

```
1 const name = 'Diarmuid' ;
2 const height = 5.8 ;
3 const toMeters = function(feet) { return (feet*0.3048).toFixed(2) }
4 let text =
5   `My name is ${name} and my height is ${toMeters(height)} meters`;
6 console.log(text);
7 // Multi-line strings
8 text = (
9   `My name is ${name}
10  and my height is ${toMeters(height)} meters
11`);
12 console.log(text);
13 // Embedded quotes in a string
14 text =
15   `I'm "amazed" that we have so many quotation marks to choose from!` ;
16 console.log(text);
```

```
My name is Diarmuid and my height is 1.77 meters
My name is Diarmuid
and my height is 1.77 meters

I'm "amazed" that we have so many quotation marks to choose from!
```

JavaScript - Behavior structures

JavaScript functions.

- Fundamental unit of composition for logic (or BEHAVIOUR).
- Function syntax:
 - ES5:
 - Function declarations.
 - Function expressions
 - ES6:
 - Arrow functions.
 - Shorthand arrow functions.
 - Anonymous functions.
- Ref. functions/01_functionBasics.js
- Hoisting (ES5) – all functions moved to the top of the current scope at runtime.
 - Function call can appear before its declaration.

Function characteristics

- Constructor functions – **function for creating objects of a certain type, e.g.**

```
function Person(.....){.....}  
let him = new Person('joe Bloggs', '1 Main Street','m',..... )
```

 - **Same purpose as classes in Java.**
- **Side-effects** – **when a function** “modifies some state variable value(s) outside its local environment”, e.g.
addMiddleName() causes side effects.
salute() does not cause side effects.
 - **Performing I/O also considered a side effect.**
- **Pure function** – **has no side effects; will always return the same result for a given set of parameters.**
 - **Functional programming.**

Higher Order Functions (HOF).

- A function that takes a function as a parameter (and/or returns a function).
 - Function parameter termed a callback.
function someHOF(..., callback,)
 - Callback usually coded as anonymous function using the arrow syntax.
- Case study – The Array HOFs:
 - **forEach()**
 - **filter()**
 - **map()**
 - **reduce()**

Array HOFs – forEach().

- **An alternative to the for-loop, e.g.**

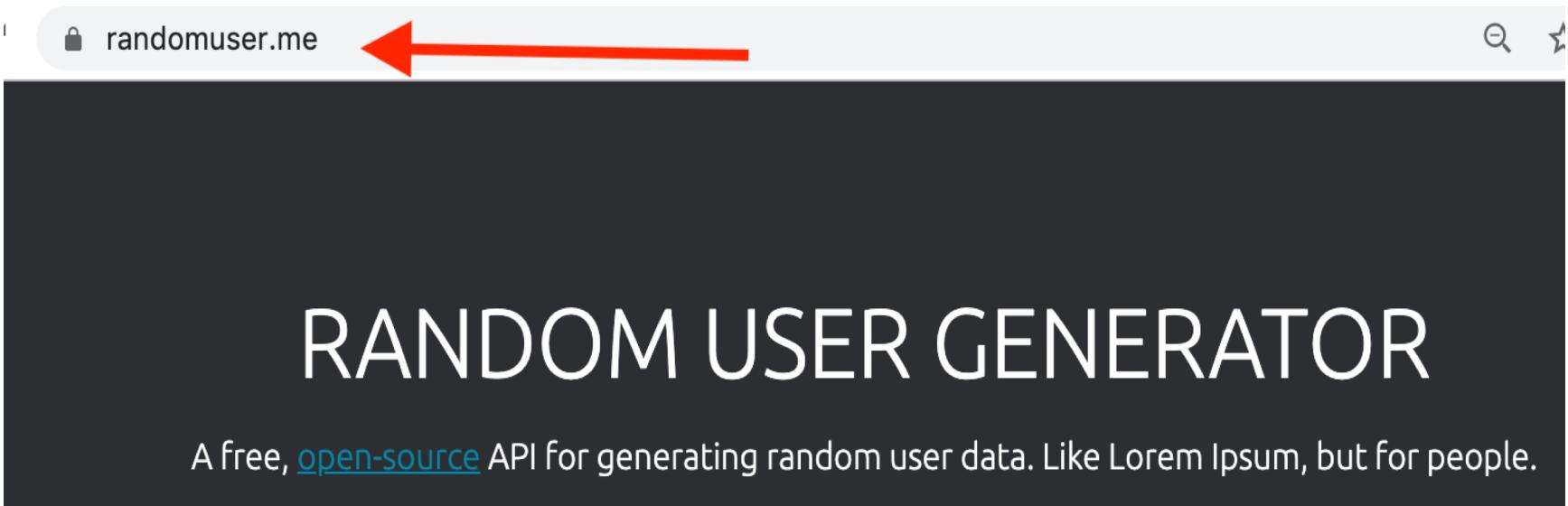
```
nums.forEach(  
  (n, index, array) => {.    // Anonymous function  
    console.log(`${n} at index ${index} of ${array}` )  
  }  
)
```

[**index and array arguments are optional.**]

Output:

```
12 at index 0 of 12,22,5,28  
22 at index 1 of 12,22,5,28  
5 at index 2 of 12,22,5,28  
28 at index 3 of 12,22,5,28
```

Array HOF demos context



- **Open Web API.**
- **Accepts HTTP GET requests, e.g.**
<https://randomuser.me/api/?results=10> - generate 10 user profiles.
- **Returns JSON response.**

Array HOF demos

context



- **Ref functions/webAPI.js**

```
{ "results": [ { "gender": "male", "name": { "title": "Mr", "first": "Jonathan", "last": "Velasco" }, "location": { "street": { "number": 4156, "name": "Calle de Ferraz" }, "city": "Elche", "state": "Castilla la Mancha", "country": "Spain", "postcode": 57833, "coordinates": { "latitude": 39.4322, "longitude": -2.3419 } }, "timeZone": "Europe/Madrid", "offset": "+02:00", "description": "Central European Summer Time" }, { "email": "jonathan.velasco@example.com", "login": { "uuid": "9fa67348-b896-48a7-83a5-c3e844e9f08c", "username": "purplegoose817", "password": "SecureP@ssw0rd", "salt": "42315455", "hash": "5d4bb412a54827c3273455a2a2a2a2a2", "method": "bcrypt", "iteration": 10000, "key": "31337" } } ] }
```

JSON Response formatted using jsonformatter.curiousconcept.com

Array HOF demos.

- **filter()**
 - Select entries from an array, based on some criteria.
 - Selected entries added to a new array.
 - Source array unchanged (Pure).
 - Ref. functions/03_filtering.js
- **map()**
 - Creates a new array with the results of calling a function for every array element.
 - Source array unchanged (Pure)
 - Ref. functions/04_mapping.js

Array HOF demos.

- **reduce()**
 - reduces the array to a single accumulated value. Reduce() executes a provided function for each element of the array (from left-to-right). The return value of the function is stored in an accumulator (result/total). The parameters of reduce() are: accumulator from the previous function call; current array element, current index and full array.

```
sourceArray.reduce(  
  (acc, element, index, array) => {
```

```
  . . . . .
```

```
  return updatedAccumulator
```

```
}, initialAccumulator )the
```

Note initialAccumulator argument.

- **Ref functions/05_reducing.js**

Summary

- **Representing Data / State**
 - **Primitives.**
 - **Objects.**
 - **Dynamic, nested.**
 - **Arrays.**
 - **String templates**
- **Defining Behavior.**
 - **Functions:**
 - **ES5 – Function declarations; Function expressions.**
 - **ES6 – Arrow functions. Shorthand.**
 - **Anonymous functions.**
 - **Higher Order functions.**

