JavaScript.

The Fundamentals

Topics

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

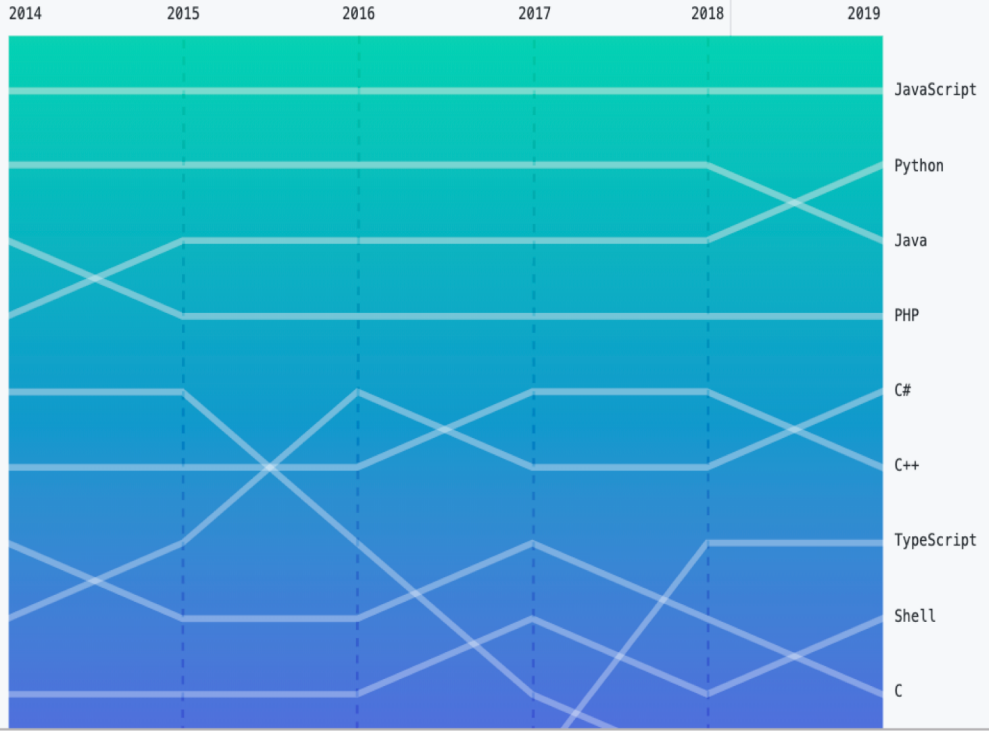
Ref : https://octoverse.github.com/#top-languages

Top languages

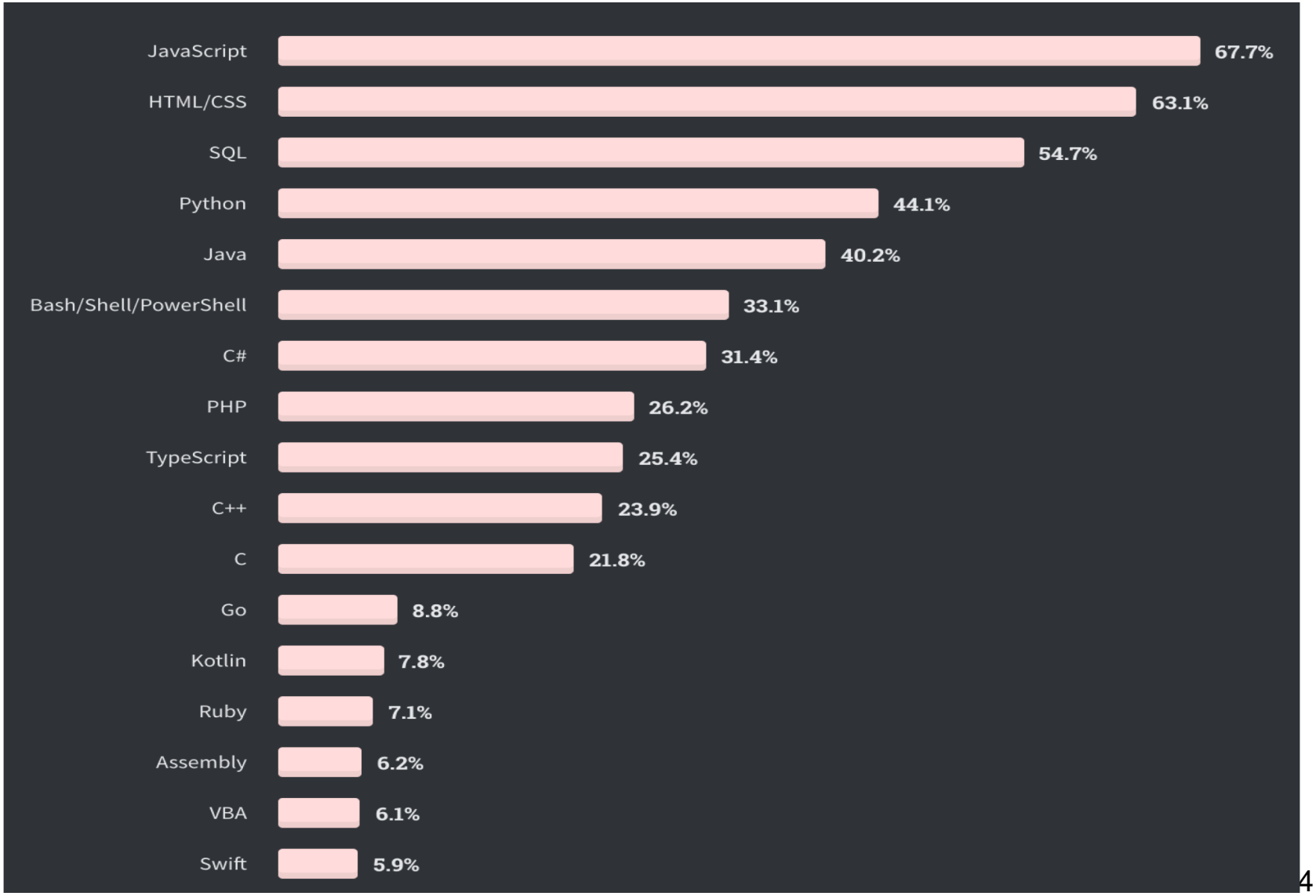
In the last year, developers collaborated in more than 370 primary languages on GitHub.

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.*



Ref.: https://insights.stackoverflow.com/survey/2020#technology



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 – Abandonned.**
  - **ES5 - 2009; ES6 - 2015 (ES2015); ES7 – 2016 (ES2016) ……**
- **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 transpilitation needs.**

JavaScript - Data representation.

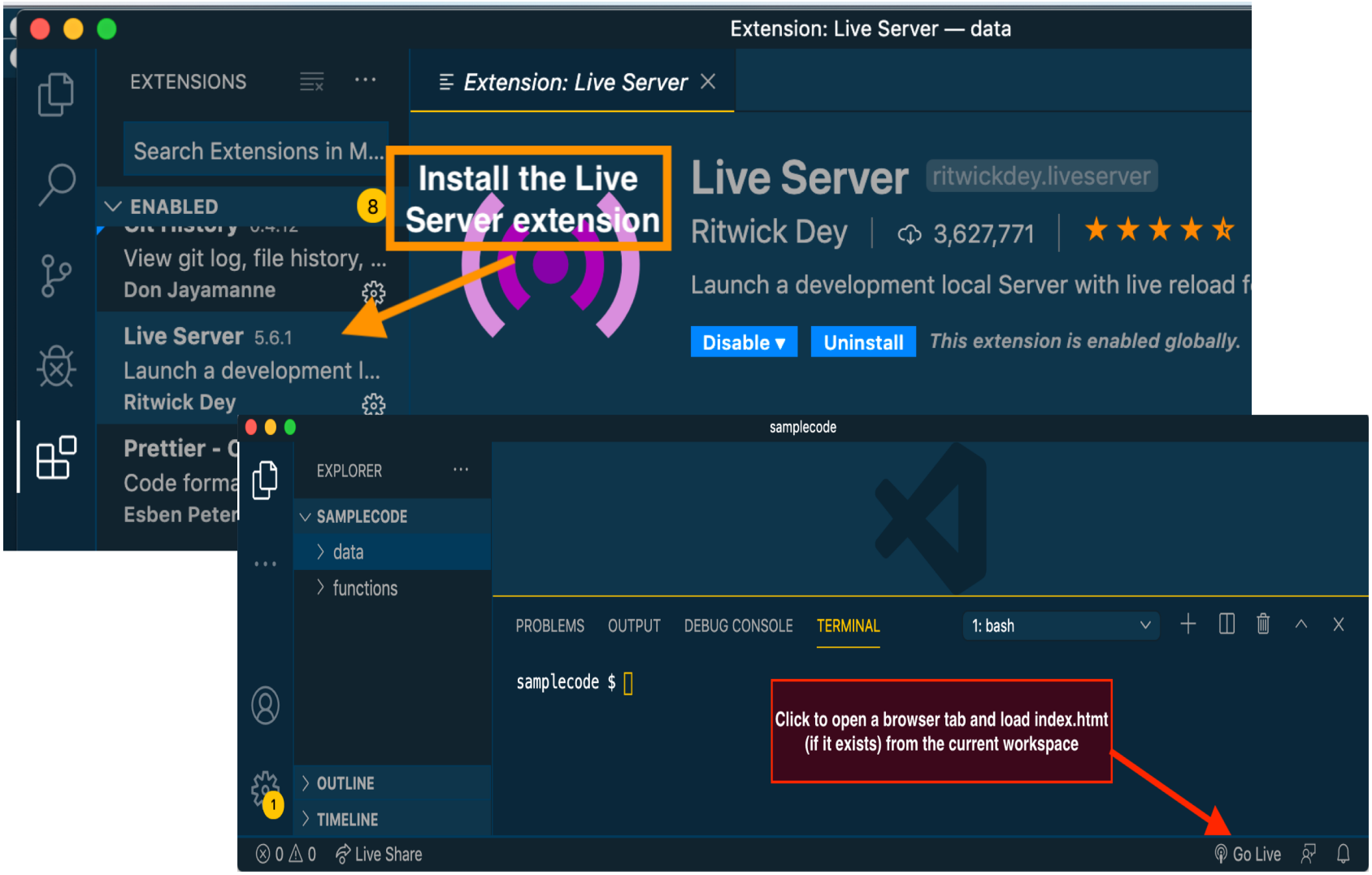
JavaScript Data Types.

- **Data types:**
  1. **Primitives: number, string, boolean, null, undefined.**

  2. **Everything else is an object.**

- **JS is a dynamically typed language.**

Demo setup


Extension: Live Server — data
EXTENSIONS
Extension: Live Server
Search Extensions in M...
Install the Live Server extension
ENABLED
8
View git log, file history, ...
Don Jayamanne
Live Server 5.6.1
Launch a development l...
Ritwick Dey
Live Server
ritwickdey.liveserver
Ritwick Dey
3,627,771
Launch a development local Server with live reload f
Disable
Uninstall
This extension is enabled globally.
samplecode
EXPLORER
SAMPLECODE
data
functions
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: bash
samplecode $
Click to open a browser tab and load index.htmt (if it exists) from the current workspace
OUTLINE
TIMELINE
Live Share
Go Live

Primitive types.

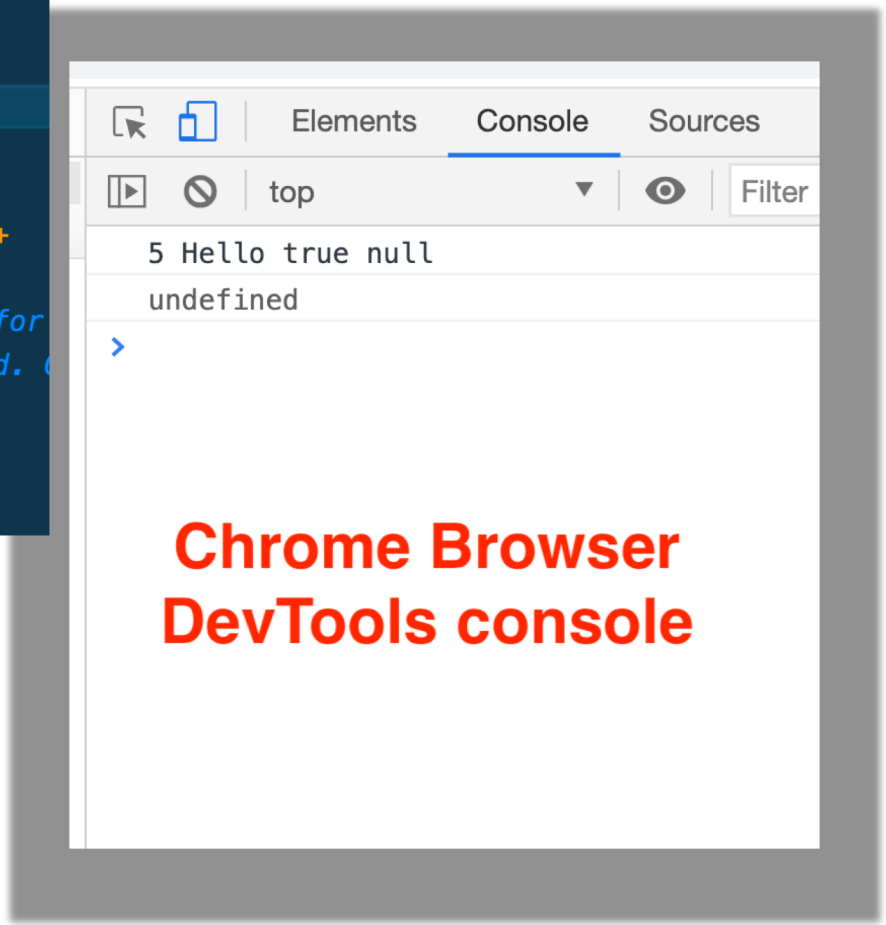
- **Ref dataSamples/***01_primitives.js:*

01_primitives.js

```
4   let foo1 = 5;
5   let foo2 = "Hello";
6   let foo3 = true;
7   let foo4 = null;
8   const Pi = 3.14;
9   console.log(foo1 + " " + foo2 + " " +
10                  foo3 + " " + foo4);
11  foo1 = 3; // Reassign foo1. No need for
12  foo2 = 10; // JS is dynamically typed.
13  let foo5;
14  console.log(foo5);
15  // Pi = 3.141592  // ERROR
```

index.html

```
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 (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 **– 'foo1' 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).**
- **An object is a set of** key-value **pairs, termed** properties**.**
  - { <key1> : <value1>, <key2> : <value2>, ... }
  - **Key (property name) - an identifier; must be unique within the object structure.**
  - **Value - can be a primitive value, another object (nesting) , array or function.**

e.g.

const me = { firstName: “Diarmuid”, lastName: “O’ Connor” } ;

Manipulating Object properties.

- **Two notations:**
  1. **Dot notation e.g** me.firstName ;
  2. **Subscript notation e.g.** me['firstName’] (Note quotes)
- **Same notations for** changing **a property value, e.g.**

  me.firstName = ‘Jeremiah’ ;

  me[‘lastName’] = ‘O Conchubhair’ ;
- **Subscript notation supports a variable reference as the key:**

  const key = ‘lastName’ ; console.log (‘Surname: ’ + me[key]) ;
- **Objects declared with** *const* **ARE MUTABLE.**
  - *const* **cannot be reassigned, but its internal ‘value’ is mutable.**
- **Ref. dataSamples/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.

- **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 object have a certain key? e.g.**
  ‘name’ in me
- **Ref. 04_2_ nested_objects.js**
- **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) ;

  - **Error prone and cumbersome.**

- **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.**

- **Ref 06_string_templates.js**

JavaScript - Behavior structures

JavaScript functions.

- **Fundamental unit of composition for logic (or BEHAVIOUR).**

- **Function syntax:**
  - **ES5:**
    - **Function declarations.**
    - **Function expressions.**
    - Hoisting **(ES5) – all functions moved to the top of the current scope at runtime.**

  - **ES6:**
    - **Arrow functions.**
      - **Shorthand version.**
  - **Anonymous functions (see later).**
- **Ref. functions/01_functionBasics.js**

Arrow functions

- **A cleaner syntax for creating functions.**

  const name = (parameters) => { ….. Body ……. }
- **The => (arrow) separates function body from its parameters.**
- **Enclose body with curly braces, { …… }.**
  - **Unless body is a single expression (optional).**
- **Enclose parameter list with parentheses, (…).**
  - **Unless only a single parameter (optional).**
- **Omit return token when single-expression body (optional).**

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 a side-effect.**

    salute() does not cause side-effects.
  – **Performing I/O also considered a sid- 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).

- **Definition: A function that takes a function as a parameter (and/or returns a function response).**
  - **Function parameter termed a** callback.

    function someHOF(. . ., callback, ….)
  - **Callback usually coded as an** anonymous **function.**

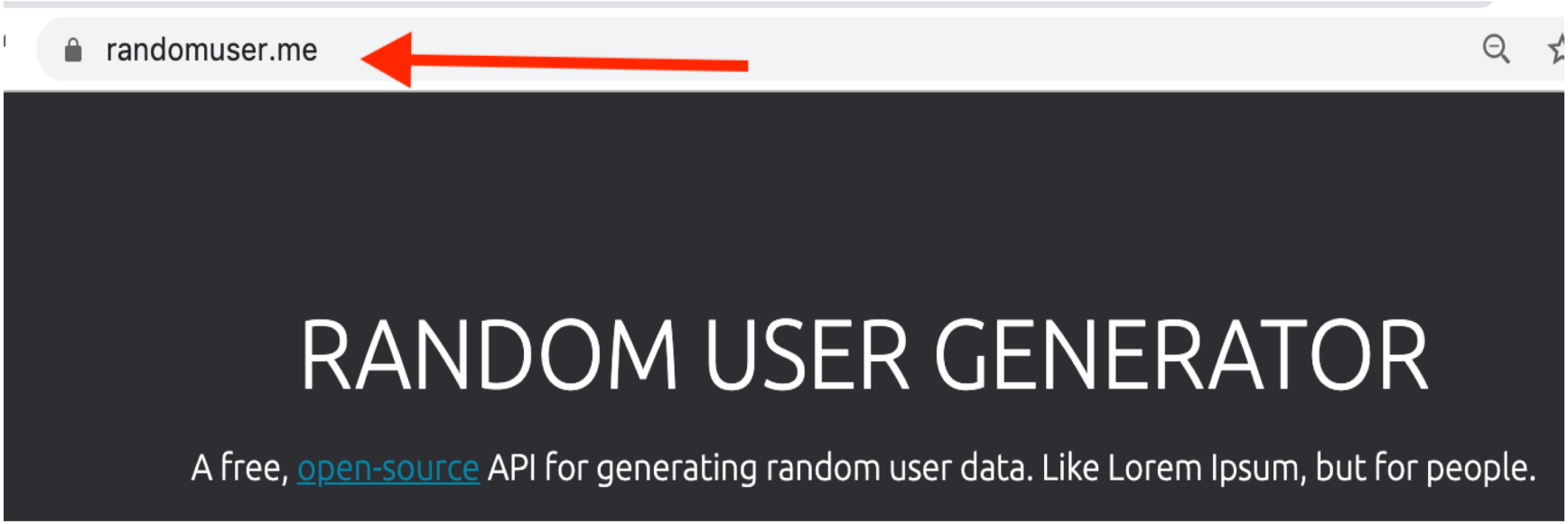
- **Case study – The Array HOFs.**
  - **forEach()**
  - **filter()**
  - **map()**
  - **reduce()**

Array HOFs – forEach().

```
const sourceArray = [ ……. ]
sourceArray.forEach(
   function(element, index, array) { ……Anonymous function ……}
)
```

- **Calls anonymous function for each element in source array.**
- **An alternative to using for-loop.**
- index **and** array **arguments are optional.**
- **Arrow function style commonly used.**

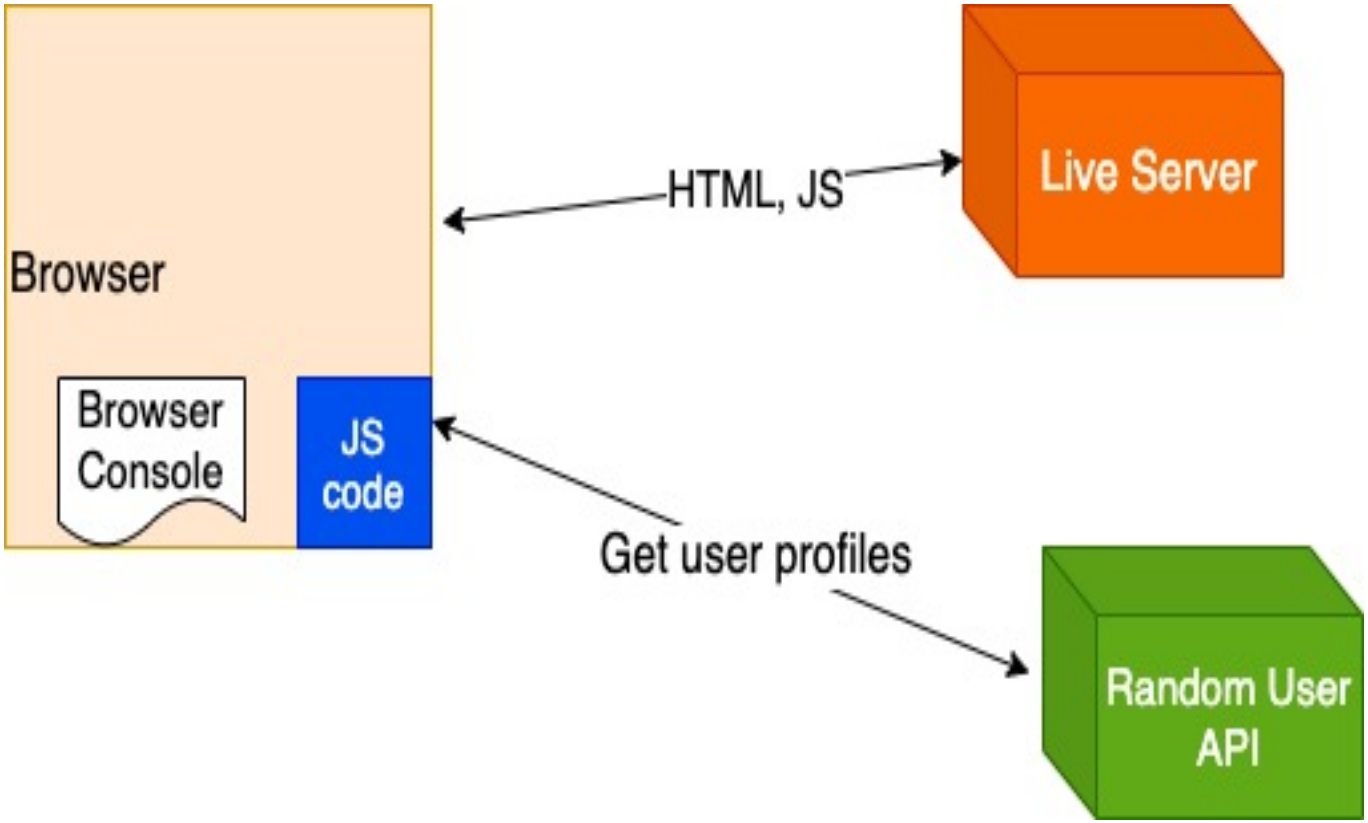
Array HOF demos context



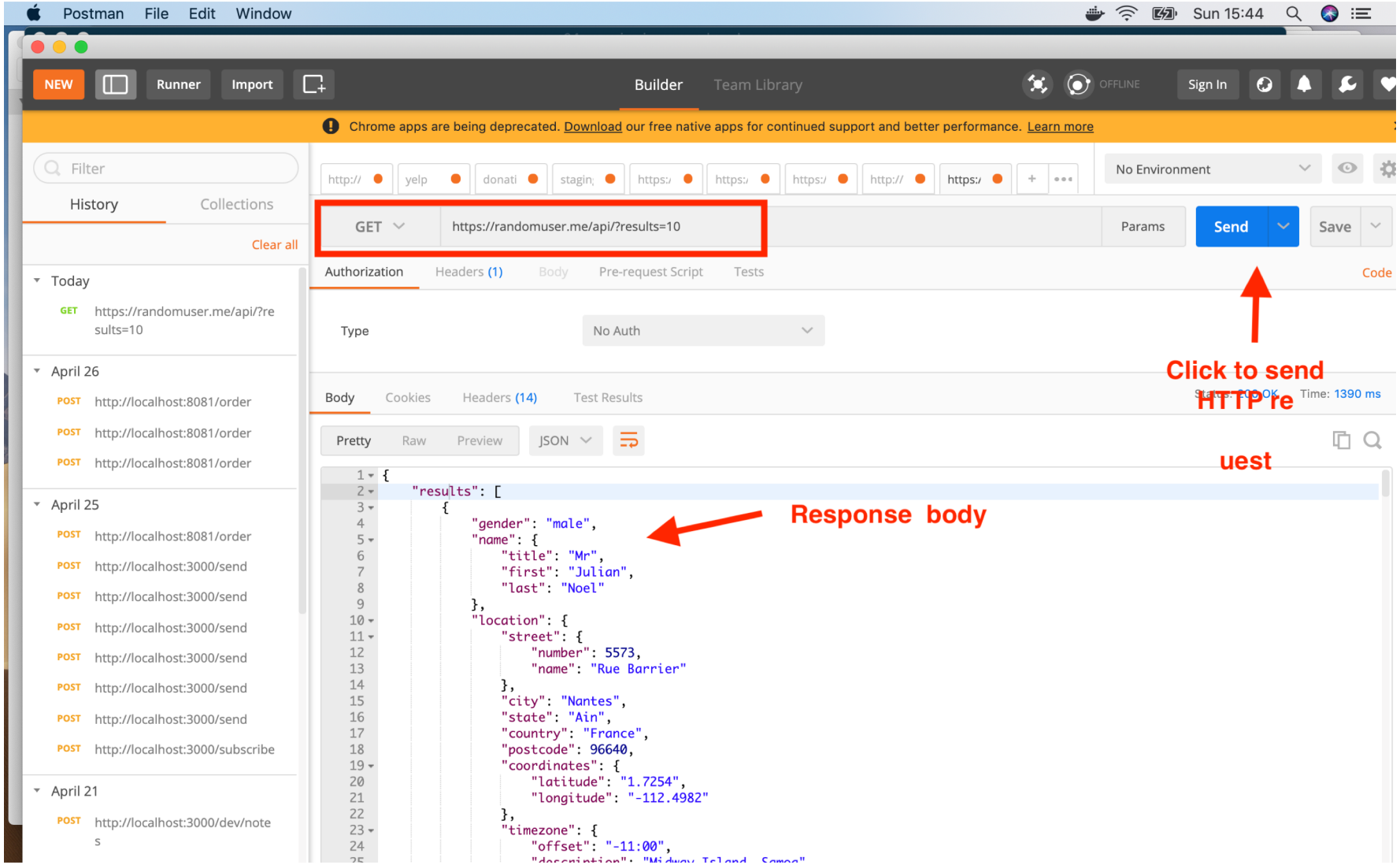
- **Open Web API.**
- **Accepts HTTP GET requests, e.g.**

  https://randomuser.me/api/?results=10 - generate 10 user profiles and returns them in a JSON (Javascript Object Notation) structure.

Array HOF demos context



Use **Postman** to test API
(Postman = Chrome extension or app)



Array HOF demos.

- **Base example.**
  - **fetch() and array.forEach(callback)**
  - **Ref. functions/02_webAPICall.js.**
- **filter(callback).**
  - **Select entries from a source array, based on some criteria.**
  - **Selected entries added to a new array.**
  - **Source array unchanged (Pure).**
  - **Ref. functions/03_filtering.js**
- **map(callback).**
  - **Creates a new array from source – 1-for-1 mapping.**
  - **Source array unchanged (Pure).**
  - **Ref. functions/04_mapping.js**

Array HOF demos.

```
accumulator = sourceArray.reduce(
    (acc, element, index, array) => {.  // Callback
        . . . . . .
        return updatedAccumulator
    }, initialAccumulator )          // Note .
```

- **reduce(……)**
  - **reduces the source array to a** single accumulated value**.**
  - **Source array unchanged (Pure)**
  - **Callback incrementally ‘builds’ accumulator.**
  - **Accumulator passed between callback invocations.**
  - **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.**