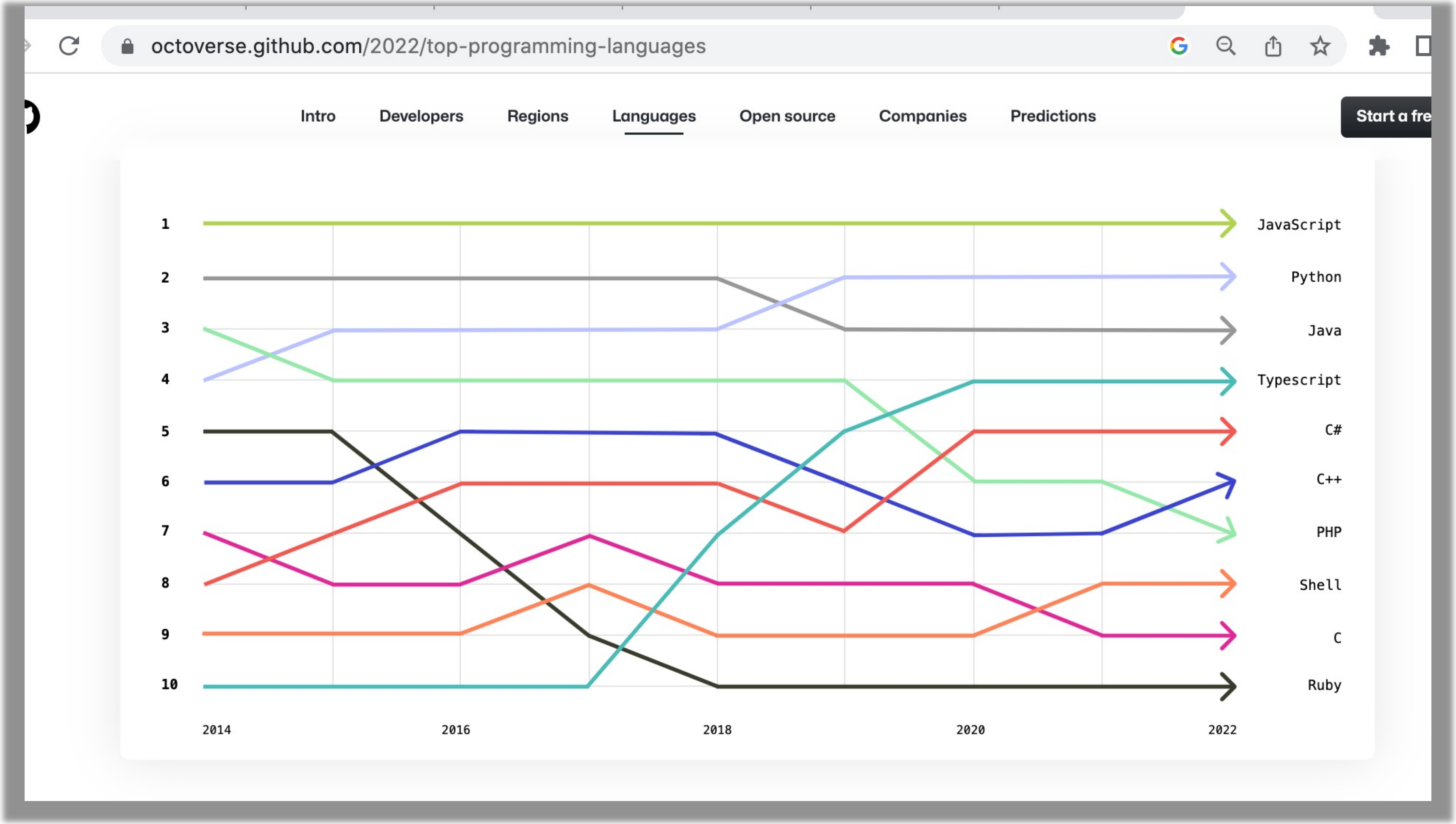
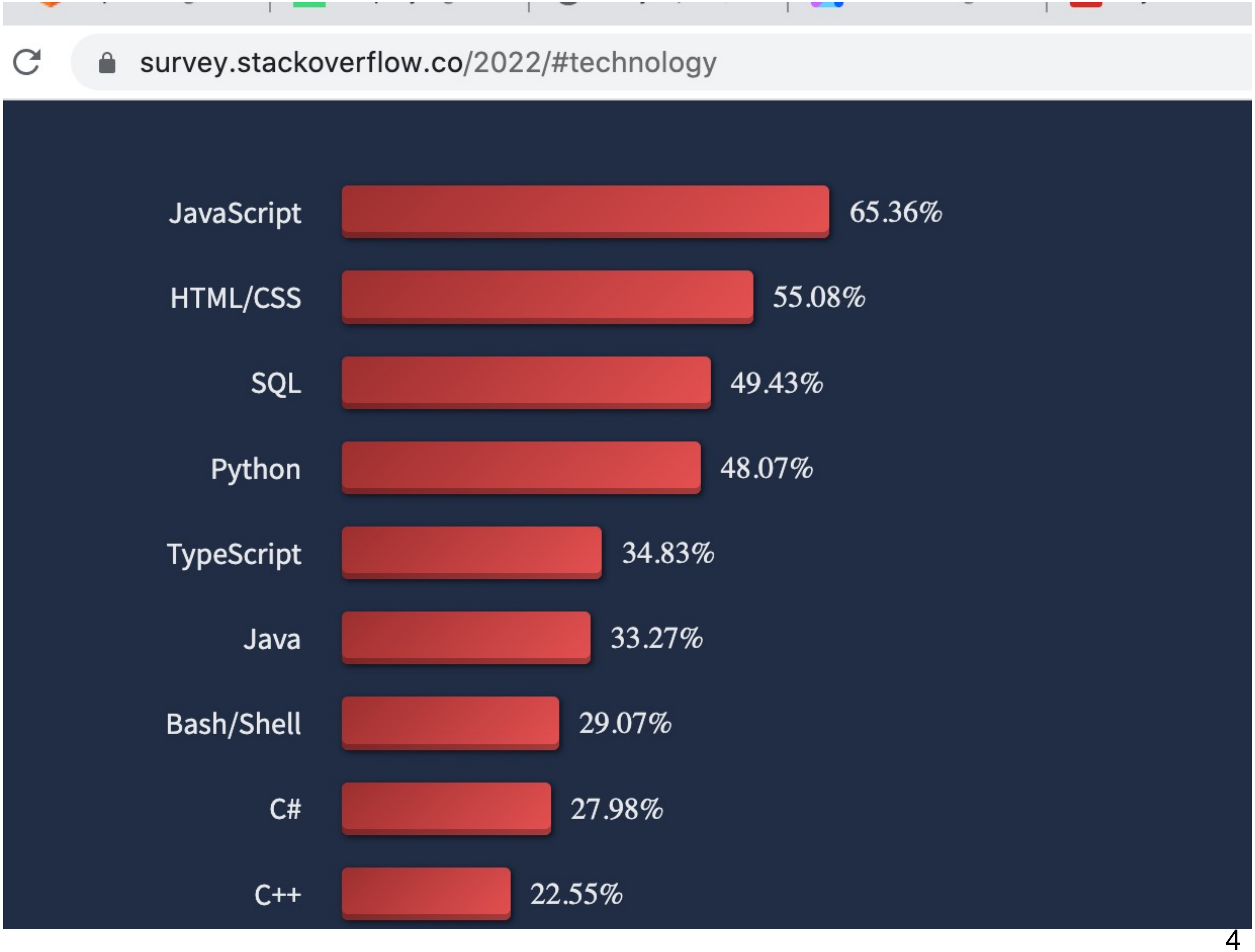
JavaScript.

The Fundamentals

Topics

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


octoverse.github.com/2022/top-programming-languages
Intro
Developers
Regions
Languages
Open source
Companies
Predictions
Start a fre
1
2
3
4
5
6
7
8
9
10
2014
2016
2018
2020
2022
JavaScript
Python
Java
Typescript
C#
C++
PHP
Shell
C
Ruby



survey.stackoverflow.co/2022/#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.**

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.

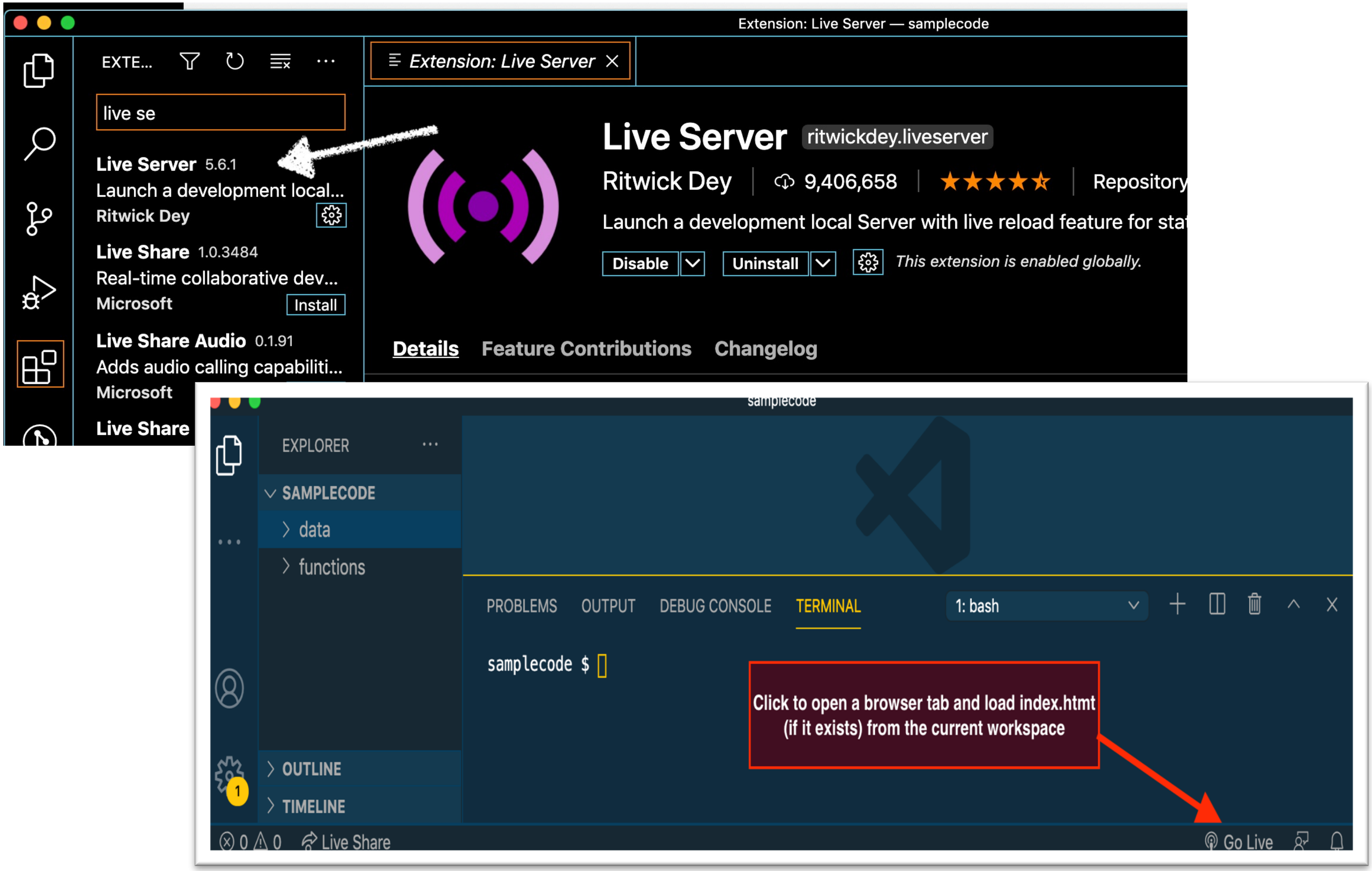
JavaScript Data Types.

- **Data types:**
  1. **Primitives: number, string, boolean, null, undefined.**
     - **undefined – a variable with no defined value.**

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

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

Demo setup (VS Code)

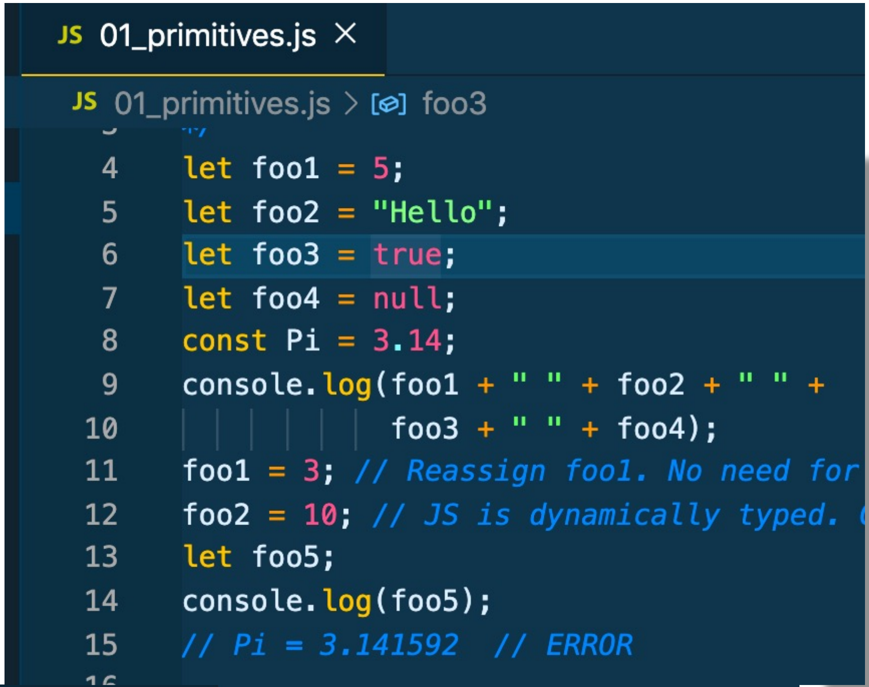


```
samplecode $ 
```

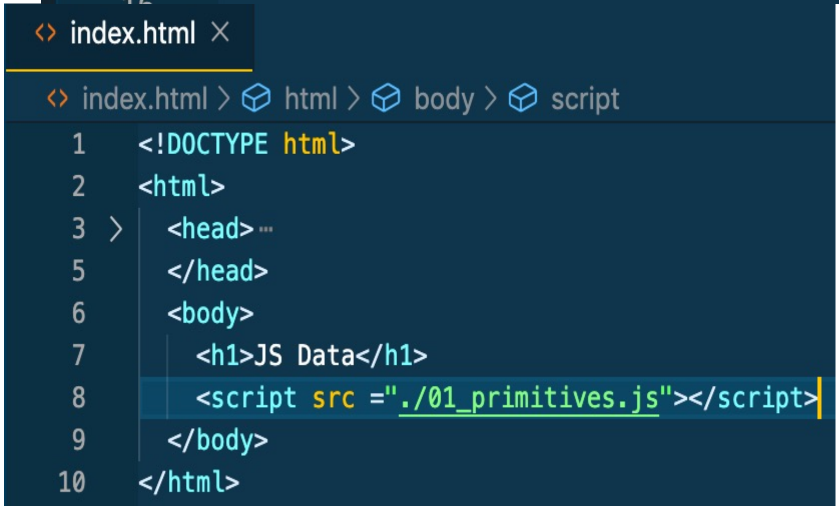
Click to open a browser tab and load index.htmt (if it exists) from the current workspace

Primitive types.

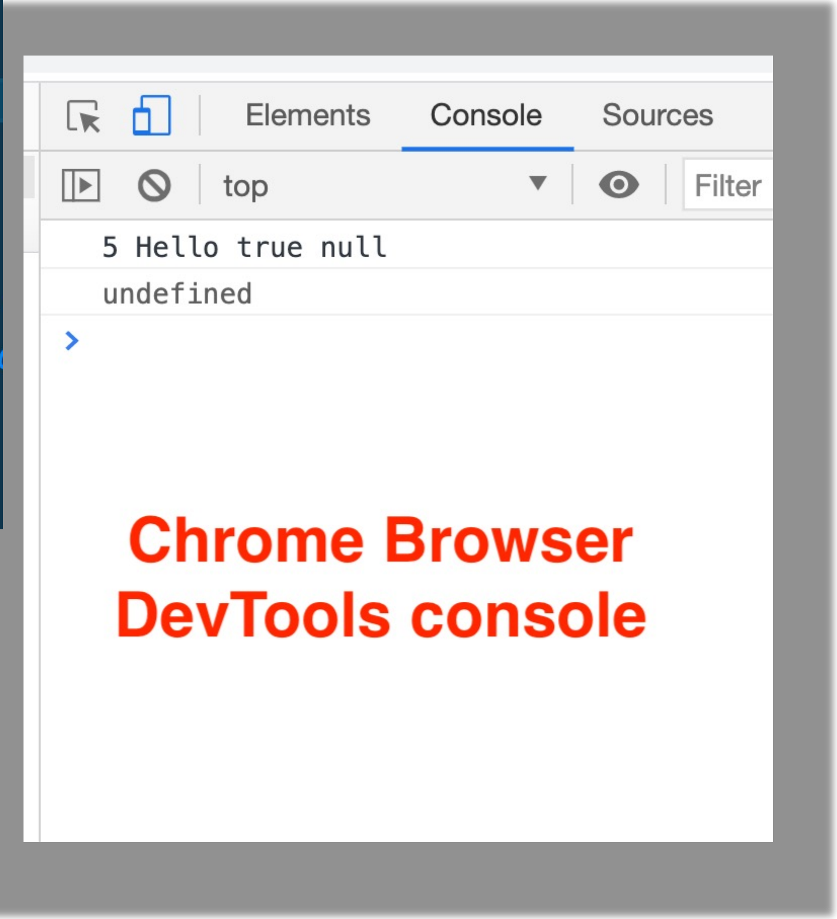
- **Ref archive -** *dataSamples/01_primitives.js:*



```
let foo1 = 5;
let foo2 = "Hello";
let foo3 = true;
let foo4 = null;
const Pi = 3.14;
console.log(foo1 + " " + foo2 + " " +
            foo3 + " " + foo4);
foo1 = 3; // Reassign foo1. No need for
foo2 = 10; // JS is dynamically typed.
let foo5;
console.log(foo5);
// Pi = 3.141592  // ERROR
```



```
<!DOCTYPE html>
<html>
  <head>…
  </head>
  <body>
    <h1>JS Data</h1>
    <script src ="./01_primitives.js"></script>
  </body>
</html>
```



```
5 Hello true null
undefined
>
```

Chrome Browser DevTools console

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. archive -** *dataSamples/02_objects.js*

Object characteristics.

- **Objects are dynamic.**
  - **Properties can be inserted and removed at run-time (JS is dynamic).**
  - **Ref. archive -** *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*

- **FYI: Internally JS stores object keys as** strings**.**
  - **Hence the subscript notation** – me[‘address’].

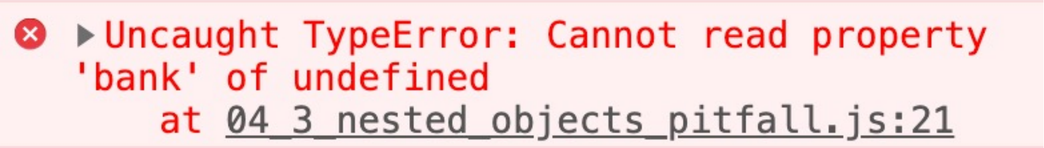
‘Cannot read property of undefined’ error

- **Suppose we access a <u>invalid property of an object</u>.**

  somObject.badProperty → undefined **(not fatal)**

- **Treating the** *undefined* **value as an object is FATAL.**

  e.g. someObject.badProperty.property → Crash!!



```
▶Uncaught TypeError: Cannot read property
'bank' of undefined
    at 04_3_nested_objects_pitfall.js:21
```

- **Ref.** 04_3_ nested_objects_pitfall.js
- **Other variations of this are:**

  *let var1 ; let var2 - var1..propert***yA ;**

  *let var3 = null ; let var4 = var3.propert***yB.**

Array data structure.

- **Dfn.: An array is an** ordered list **of values.**
  - **(An object's properties are unordered.)**

- **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.

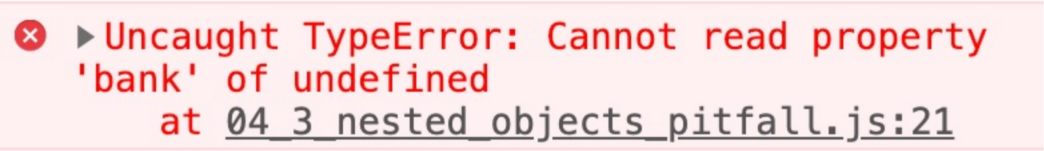
- **FYI: In JS, arrays are really just ‘special’ objects.**
  - **Index converted to a string for subscript notation:**

    nums[2] **becomes** nums[‘2’]

- **An array ‘object’ 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 can form nested data structures.**
- **Ex.:**
  - **An array whose elements are also arrays -** array_outer[3][2]
  - **An array of objects -** array_outer[1].propertyX.
  - **An object with an array property -** objectY.propertyX[5**].**
  - **etc.**
- **Accessing values inside a nested data structure can be a source of error.**



```
Uncaught TypeError: Cannot read property
'bank' of undefined
    at 04_3_nested_objects_pitfall.js:21
```

- **N.B. – Understand a nested data structure's shape before writing code that navigates it.**

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 ${ …. }.**

- **Multi-line strings.**

- **Ref archive -** 06_string_templates.js

Summary

- **Representing Data / State.**
  - **Primitives.**
  - **Objects.**
    - **Set of properties – key-value pairs.**
    - **Dynamic, nested.**
  - **Arrays.**
  - **String templates**

- **That runtime error***: ‘Cannot read property … of undefined***’**