Javascript Basics

Intro
Variables
Data Types

leandro.frigerio@digitalcareerinstitute.org

What is Javscript

JavaScript is a **cross-platform**, **object-oriented** scripting language developed by Netscape. JavaScript was created by Netscape programmer **Brendan Eich**.

It was first released under the name of LiveScript as part of Netscape Navigator 2.0 in September 1995. It was renamed JavaScript on December 4, 1995. As JavaScript works on the client side, It is mostly used for client-side web development.

JavaScript is designed for use on web pages and closely integrated with HTML. JavaScript can create applications which run in the browsers such as IE, Opera, FireFox, Google Chrome and other. Netscape submitted JavaScript to ECMA International for standardization resulting in the standardized version named **ECMAScript**.

ECMA

By 1996, JavaScript's importance grew so much that it was handed over to an international standards scripting language body called ECMA (European Computer Manufacturers Association), which is responsible for the development and upkeep of this language to this day.

As a result, the scripting language was officially given the name '**ECMAScript**', however people still call it JavaScript. **ES5** (a more modest update than the ill-fated ES4, and originally named ES3.1) was released in 2009, and the Ecma community once again piled on a wishlist of new JS features for ES6. This resulted in another protracted consensus process, spanning an interminable six years, and practically doubled the size of ECMAScript.
Every year, ECMA releases whichever features are deemed ready. **ES6** was officially published as ES2015, and since then, ES2016 and ES2017 have delivered only incremental changes, with continuous support from browser vendors.

JS Feature

- JavaScript is a lightweight, cross-platform, object-oriented computer programming language
- JavaScript is one of the tree core technologies of web development
- JavaScript is most commonly used as a part of webpages & webapps
- Today, JavaScript can be used in different places:
  - Client-side: JavaScript was traditionally only used in the browser
  - Server-side: Thanks to node.js, we can use JavaScript on the server as well
- **JavaScript is what made modern web-development possible:**
  - Dynamic effects and interactivity
  - Modern web applications that we can interact with

CONTENT

PRESENTATION

DYNAMIC EFFECTS/
PROGRAMMING

NOUNS

`<p></p>`
means "**paragraph**"

ADJECTIVES

`p {color: red;}`
means "the paragraph text is **red**"

VERBS

`p.hide();`
means "**hide** the paragraph"

So what can it really do?

The core JavaScript language consists of some common programming features that allow you to do things like:

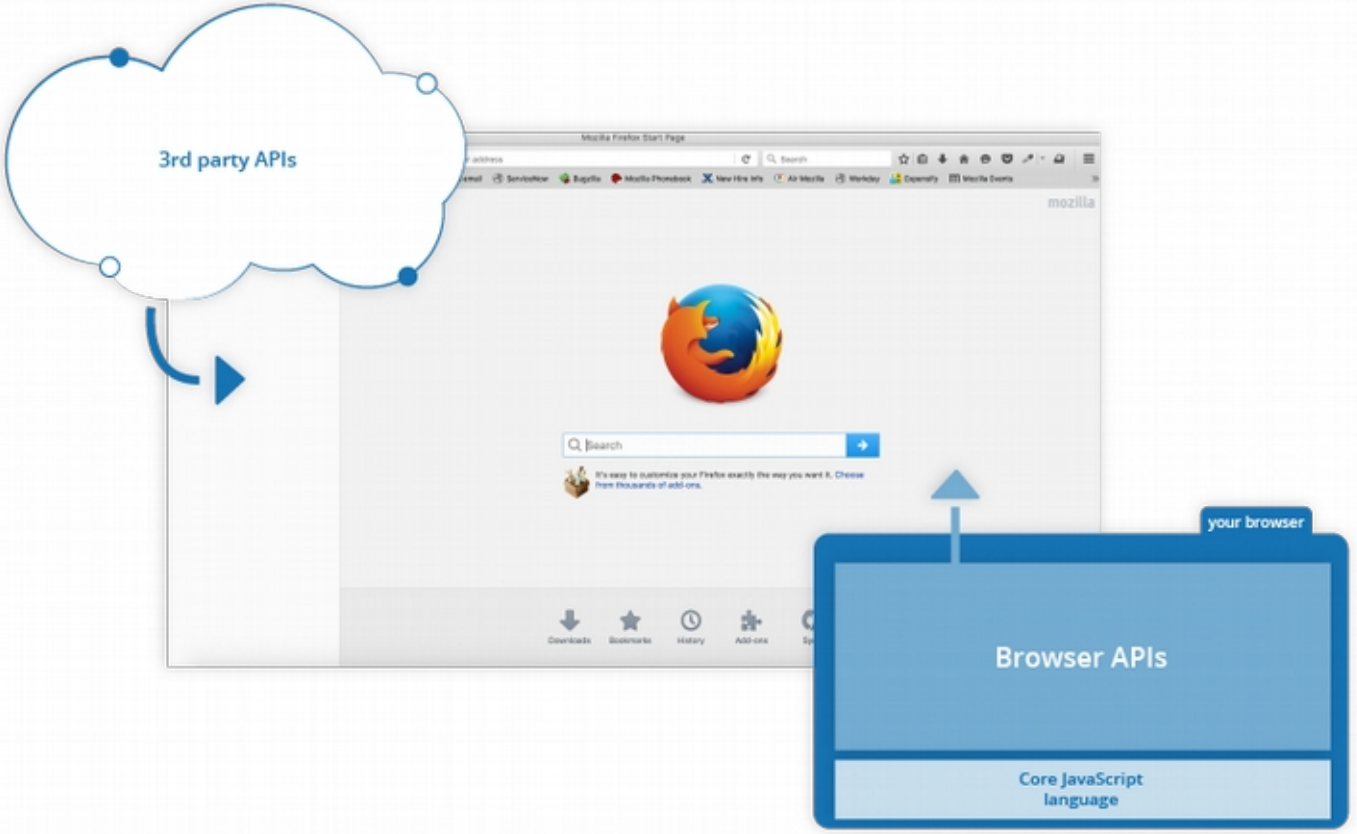
- Store useful values inside **variables**.
- **Operations** on pieces of text (known as "strings" in programming) and numbers.
- Running code in response to certain **events** occurring on a web page.

What is even more exciting however is the functionality built on top of the core JavaScript language. So-called **Application Programming Interfaces** (APIs) provide you with extra superpowers to use in your JavaScript code.

APIs are ready-made sets of code building blocks the Developer can use.

API

They generally fall into two categories.



APIs

- Third party APIs are not built into the browser by default, and you generally have to grab their code and information from somewhere on the Web. For example: Facebook API, Google API, Github API, Reddit API, etc…
- Browser APIs are built into your web browser, and are able to expose data from the surrounding computer environment, or do useful complex things. For example:
  - The DOM (Document Object Model) API allows you to manipulate HTML and CSS, creating, removing and changing HTML, dynamically applying new styles to your page, etc.
  - The Geolocation API retrieves geographical information.
  - The Canvas and WebGL APIs allow you to create animated 2D and 3D graphics.
  - Audio and Video APIs like HTMLMediaElement and WebRTC allow you to do really interesting things with multimedia

HTML, CSS and JavaScript

JavaScript runs in your web browser alongside HTML and CSS, and can be added to any web page using a <**script>** tag.

The script element can either contain JavaScript directly (internal) or link to an external resource via a src attribute (external).

<script> alert("Hello, world."); </script>

or

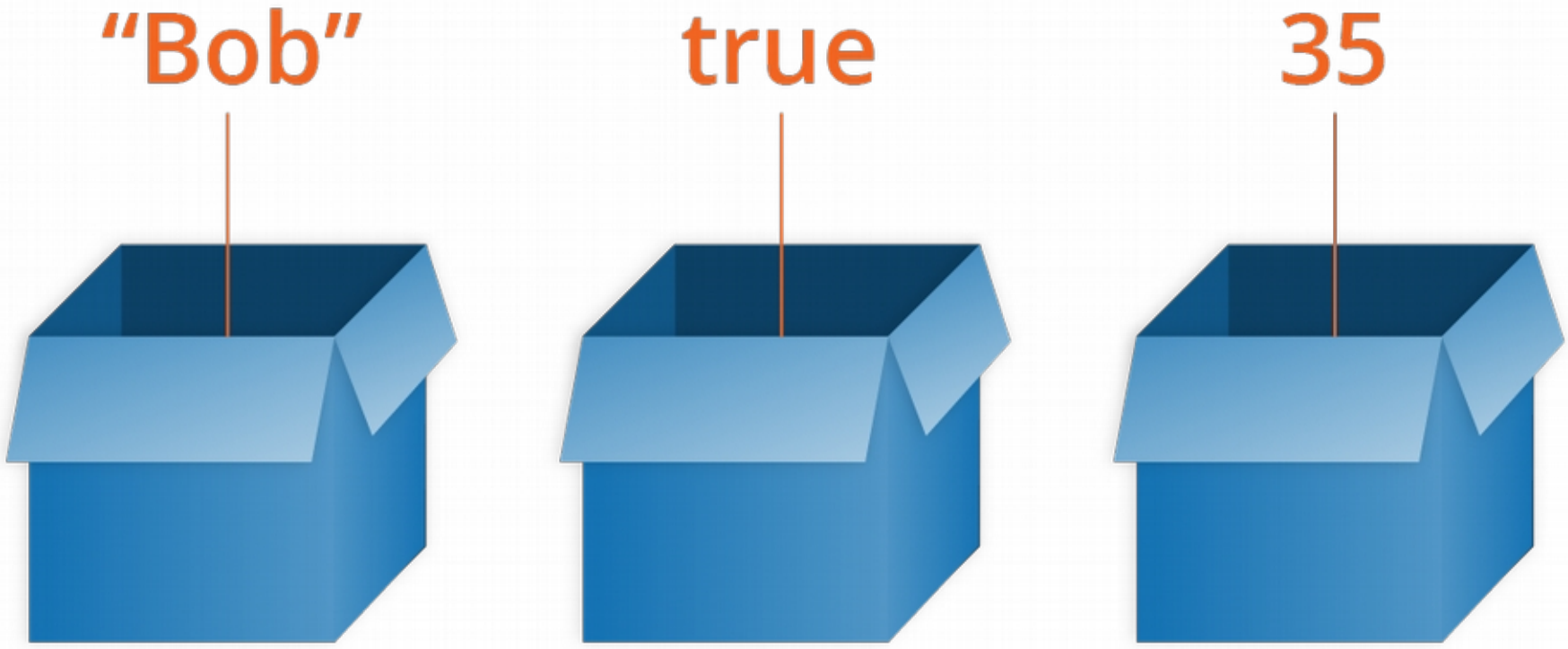
<script src="assets/js/main.js"></script>

The browser runs JavaScript code line-by-line, starting at the top of the file or script element and finishing at the bottom

Interacting with your code

- A web page with HTML/JS
- The command line interpreter of our Web Console
  https://developer.mozilla.org/en-US/docs/Tools/Web_Console
- JSBIN
  https://jsbin.com/?js,console

Variables are containers



Variables

Variable: A container in which we can store a value in order to use it later again in our code

Storing data so we can use it later is one of the most important things when writing code!

A variable is a “named storage” for data.

A variable has a name and a value.

```
var name = “Leandro”;
let age = 41;
const birthdate = ‘16.07.1977’;
```

Declaration and Initialization

- **Declaration**
  To use a variable you've first got to create it

After the declaration, the variable has no value.
(Technically it has the value of undefined)

- **Initialization**

Once you've declared a variable, you can initialize it with a value. You do this by typing the variable name, followed by an equals sign (=), followed by the value you want to give it.

You can declare and initialize a variable at the same time.

Once a variable has been initialized with a value, you can change (or update) that value by simply giving it a different value.

Constant

Variables declared using **const** are called “constants”. They cannot be changed. An attempt to do it would cause an error: you can't reassign the constant! When you are sure that the variable should never change, you should use **const**

```
const myBirthday = '16.07.1977';
```

There is a widespread practice to use constants as aliases for difficult-to-remember values that are known prior to execution.

```
const COLOR_RED = "#F00";
```

The difference between var and let

You probably already asked: “do we need two keywords for defining variables??” or “ Why have var and let?” :)

The reasons are somewhat historical. Back when JavaScript was first created, there was only **var**. So **let** was created in modern versions of JavaScript, a new keyword for creating variables that works somewhat differently to var, fixing its issues in the process.

- “the difference between them is that var is function scoped and let is block scoped.”
- “let gives you the privilege to declare variables that are limited in scope to the block, statement of expression unlike var”
- “var is rather a keyword which defines a variable globally regardless of block scope.”

Var & Let - Redeclaration

- **let** variables cannot be re-declared

  while

- **var** variable can be re-declared in the same scope.

```
'use strict';

var tempVar = "this is a temp variable";
var tempVar = "this is a second temp variable"; //replaced easily

let tempLet = "this is a temp variable";
let tempLet = "this is a second temp variable" //SyntaxError: temp is already declared
```

Var & Let - Functions

let and var variables work the same way when used in a function block.

```
function sampleNamedFunction(){

  let letVariable = "Hey! What's up? I am let variable.";
  var varVariable = "Hey! How are you? I am var variable.";

  console.log(letVariable); //Hey! What's up? I am let variable.
  console.log(varVariable); //Hey! How are you? I am var variable.

}
```

Var & Let - Block

- let variables are usually used when there is a limited use of those variables. Say, in for loops, while loops or inside the scope of if conditions etc. Basically, where ever the scope of the variable has to be limited.

```
for(let i=0;i<10;i++){
  console.log(i); //i is visible thus is logged in the console as 0,1,2,....,9
}
console.log(i); //throws an error as "i is not defined" because i is not visible
```

- For loop using var variable:

```
for(var i=0; i<10; i++){
  console.log(i); //i is visible thus is logged in the console as 0,1,2,....,9
}

console.log(i); //i is visible here too. thus is logged as 10.
```

Best practice anyone?

“For these reasons and more, we recommend that you use let as much as possible in your code, rather than var.

There is no reason to use var, unless you need to support old versions of Internet Explorer with your code (it doesn't support let until version 11; the modern Windows Edge browser supports let just fine).”

The_difference_between_var_and_let

Variable naming

You can call a variable pretty much anything you like, but there are limitations. Generally, you should stick to just using Latin characters (0-9, a-z, A-Z) and the underscore character.

- You shouldn't use other characters (ü) because they may cause errors or be hard to understand for an international audience.
- Don't use underscores at the start of variable names — this is used in certain JavaScript constructs to mean specific things, so may get confusing.
- Don't use numbers at the start of variables. This isn't allowed and will cause an error.

Variable naming / 2

- A safe convention to stick to is so-called "lower camel case", where you stick together multiple words, using lower case for the whole first word and then capitalize subsequent words. We've been using this for our variable names in the article so far.
- Make variable names intuitive, so they describe the data they contain. Don't just use single letters/numbers, or big long phrases.
- Variables are case sensitive — so myage is a different variable to myAge.
- One last point — you also need to avoid using JavaScript reserved words as your variable names — by this, we mean the words that make up the actual syntax of JavaScript! So you can't use words like var, function, let, and for as variable names. Browsers will recognize them as different code items, and so you'll get errors.

Recap Variables

We can declare variables to store data. That can be done using var or let or const.

- **let** – is a modern variable declaration.
- **var** – is an old-school variable declaration. Normally we don't use it at all, but we discussed the differences from let, just in case you need them.
- **const** – is like let, but the value of the variable can't be changed.

Data Types

A variable can contain any data.
There are 7 data types in JavaScript ES6 (6 “primitives” and object) we can store in variables.

- **number** for numbers of any kind: integer or floating-point.
- **string** for strings. A string may have one or more characters, there’s no separate single-character type.
- **boolean** for true/false.
- **null** for unknown values – a standalone type that has a single value null.
- **undefined** for unassigned values – a standalone type that has a single value undefined.
- **symbol** for unique identifiers.
- **object** for more complex data structures.

Dynamic Typing

A variable can at one moment be a string and later receive a numeric value:

```
let message = "hello";
message = 123456;
```

Programming languages that allow such things are called **“dynamically typed”**, meaning that there are data types, but variables are not bound to any of them.

https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing

Numbers

- The number type serves both for integer and floating point numbers.
- You can store numbers in variables, either whole numbers like 30 (also called integers) or decimal numbers like 2.456 (also called floats or floating point numbers).
- you don't include quotes
  ```
  let myAge = 41;
  ```
- Infinity represents the mathematical Infinity ∞. It is a special value
  ```
  alert( 1 / 0 ); // Infinity
  ```
- NaN represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for instance:
  ```
  alert( "Hello World" / 2 );
  ```

Strings

Strings are pieces of text.

```
let dolphinGoodbye = 'So long and thanks for all the fish';
```

- A string in JavaScript must be quoted.
- In JavaScript, there are 3 types of quotes:
  - Double quotes: "Hello".
  - Single quotes: 'Hello'.
  - Backticks: `Hello`.

Double and single quotes are “simple” quotes. There’s no difference between them in JavaScript.
Backticks are “extended functionality” quotes. They allow us to embed variables and expressions into a string by wrapping them in ${variableName}

Template literals (Template strings)

Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them. They were called "template strings" in prior editions of the ES2015 specification.

```
let name = "John";

// embed a variable

alert( `Hello, ${name}!` ); // Hello, John!

// embed an expression

alert( `the result is ${1 + 2}` ); // the result is 3
```

Note: this can only be done in backticks. Other quotes do not allow such embedding! You'll see "the result is ${1 + 2}"

Boolean (logical type)

The boolean type has only two values: **true** and **false**.

These are generally used to test a condition, after which code is run as appropriate. So for example, a simple case would be:

```
let iAmAlive = true;
```

Whereas in reality it would be used more like this:

```
let test = 6 < 3;
```

This is using the "less than" operator (<) to test whether 6 is less than 3. As you might expect, it will return false, because 6 is not less than 3!

Null and Undefined

- In JavaScript **null** is not a “reference to a non-existing object”
- **Null** t’s just a special value which has the sense of “nothing”, “empty” or “value unknown”
- The meaning of **Undefined** is “value is not assigned”
- If a variable is declared, but not assigned, then its value is **undefined**
- Technically, it is possible to assign **null** and **undefined** to any variable.

  Normally, we use **null** to write an “empty” or an “unknown” value into the variable, and **undefined** is only used for checks, to see if the variable is assigned or similar.

Objects and Symbols

The **object** type is special. All other types are called “primitive”, because their values can contain only a single thing (be it a string or a number or whatever). In contrast, **objects** are used to store collections of data and more complex entities.

```
let dog = { name : 'Spot', breed : 'Dalmatian' };
```

In programming, an object is a structure of code that models a real-life object. You can have a simple object that represents a box and contains information about its width, length, and height, or you could have an object that represents a person, and contains data about their name, height, weight, what language they speak, how to say hello to them, and more.

The **symbol** type is used to create unique identifiers for objects.

Arrays

An **array** is a single object that contains multiple values enclosed in square brackets and separated by commas.

```
let myNameArray = ['Chris', 'Bob', 'Jim']
```

```
let myNumberArray = [10, 15, 40];
```

Once these arrays are defined, you can access each value by their location within the array.

```
myNameArray[0]; // should return 'Chris'
myNumberArray[2]; // should return 40
```

The typeof operator

The **typeof** operator returns the type of the argument. It's useful when we want to process values of different types differently, or just want to make a quick check.

It supports two forms of syntax:

- As an operator: typeof x.
- Function style: typeof(x).

Recap Data Types

There are a few (7) different types of data we can store in variables.

- 6 are primitives, their values can contain only a single thing
- 1 is Object, can store collections of data and more complex entities

The typeof operator allows us to see which type is stored in the variable.

Reference

- MDN - First Steps - What is JS
- MDN - Variables
- https://javascript.info/first-steps
- https://javascript.info/types
- https://codepen.io/leandro_berlin/pen/dwYWod
- https://codepen.io/leandro_berlin/pen/REWLbJ
- https://auth0.com/blog/a-brief-history-of-javascript/