

Introduction to JavaScript

For web browsers

Julius Putra Tanu Setiaji

NUS Hackers Hackerschool, 2018

NUS Hackers



<http://nushackers.org>

Hackerschool
Friday **Hacks**
Hack & Roll
NUS **Hackerspace**

About Me

Hi! I'm Julius. My GitHub is
`https://github.com/indocomsoft`

A Year 1 Computer Science Undergraduate who loves hacking
and building systems.

I took CS1101S taught in JavaScript and have been doing web
development intensively for the past 2 years.

(Not so important) I also enjoy Aerospace Engineering, Music
Theory and History (my favourite games are KSP and EU4 hit me up if you play those too)

Table of Contents

Introduction

Primitive Data Types

Variable, Data Structures, Flow Control

HTML DOM

- Selecting an element

- Playing audio file

- Adding/Removing a CSS class

- Events and Listeners

Putting everything together

Required Software

- **Google Chrome** (<http://chrome.google.com/>)
- **Sublime Text 3** (<http://www.sublimetext.com/3>) or any decent text editor

Materials can be found at

https://drive.google.com/drive/folders/1gaiBcnkGRwZ3w3z1j8H_CZqpV6iPcQmz?usp=sharing

Code snippets at

<https://hackmd.io/0bxMdu7SS0ijP0089-G75w?view>

JavaScript Drum Kit

- A sneak peek on what we will be building today.
- Do raise your hands if you're lost!

Why and What is Javascript?

- HTML & CSS defines a webpage's structure and style statically.
- JavaScript allows more dynamic aspect of the web:
 - User interaction
 - Modifying the webpage
 - Communicating with a server
- Javascript is:
 - dynamic and weakly-typed
 - multi-paradigm (prototype-based object-oriented, imperative, functional, event-driven)

Short History

- It was first included by Netscape Navigator in 1995.
- It has since been standardised by Ecma Int'l.
- Consequently, the standard is called ECMAScript.
- There are several editions of the standard:
 - ECMAScript 5.1
 - ECMAScript 6 (ES6, also called ES2015)
 - ECMAScript 7 (ES7, also called ES2016)
- For the purpose of today's Hackerschool, we will focus more on ES6.

Resources

- Mozilla Developer Network (<https://developer.mozilla.org/en-US/docs/Web/JavaScript>) offers one of the best documentation of JavaScript.
- Even Microsoft is redirecting its web docs to MDN¹

¹<https://blogs.windows.com/msedgedev/2017/10/18/documenting-web-together-mdn-web-docs/>

Following along

- All modern web browsers have an integrated JavaScript interpreter. You can run codes in this presentation by using the console.
- For Firefox and Chrome, go to **Developer Tools** (keyboard shortcut: Ctrl+Shift+I or Command + Option + I)

Data Types

There are 6 primitive data types in ES6:

- Null
- Undefined
- **Number**
- **String**
- Symbol
- **Boolean**

3 Important Primitive Data Types

- **Number** = A numeric data type in the double-precision 64-bit floating point format

```
1  typeof 1101          // returns "number"
2  typeof 5.00         // returns "number"
3  typeof Math.PI      // returns "number"
```

- **String** = sequence of characters

```
1  typeof "asd"        // returns "string"
2  typeof 'asd'        // returns "string"
```

- **Boolean** = only 2 possible values: true and false

```
1  typeof true         // returns "boolean"
2  typeof false        // returns "boolean"
```

Variable Declaration

- Traditionally, variables are declared using `var`.
- However, since ES6, there are 2 more ways to declare variable, `let` (allows reassignment) and `const` (prevents reassignment).
- The difference is in scoping². Generally, I would advise using `let` and `const`.

```
1  var name = "Julius"
2  let mood = "happy"
3  const birthyear = 1997
4  name = "indocomsoft" // OK
5  mood = "excited"      // OK
6  birthyear = 2001      // Error
```

²`var` is function-scoped while `let` and `const` are block-scoped

Array

- **Array** is an ordered collection of data.

```
1  // Empty array
```

```
2  []
```

```
3
```

```
4  let arr = [1, 2, 3, "a", true]
```

```
5  a[0] // 1
```

```
6  a[3] // "a"
```

```
7  a[4] // true
```

- There are many built-in Array methods. Look them up at MDN!

Object

- **Object** is a data structure containing data and instructions (fields and methods).

```

1  // Empty object
2  {}
3
4  // Literal object
5  let car = { "brand": "Tesla", "model": "X",
6             ↪ "production_year": 2015 }
7  car["brand"]           // "Tesla"
8  car.model              // "X"
9  car["production_year"] // 2015
10 car.production_year    // 2015
11 car.name               // undefined

```

Function

- **Function** is a code snippet.

```

1  function plusOne(x) {
2      return x + 1;
3  }
4  plusOne(2)                                // Returns 3
5
6  let plusOne = (x) => x + 1; // Arrow functions
7  plusOne(2)                                // Returns 3
8
9  // Functions can be passed around
10 let op = (f, v) => f(v);
11 op(plusOne, 5);                            // Returns 6

```


if - else if - else Flow Control

- Logical operators: && (and), || (or), ! (not)
- Comparison operators: == (equality), != (inequality), === (identity/strict equality), !== (non-identity/strict inequality), >, >=, <, <=.

```
1  let x = 10;  
2  if (x < 10) {  
3      console.log("smaller")  
4  } else if (x > 10) {  
5      console.log("larger")  
6  } else {  
7      console.log("equal")  
8  }
```

Truthy and Falsy

- Values that translate to true and false respectively.
- List of falsy values:

```
1  if (false)
2  if (null)
3  if (undefined)
4  if (0)
5  if (NaN)
6  if ( '')
7  if ( "")
```

- Other values are by definition truthy

```
1  let me = { "name": "Julius", "age": 21 }
2  if (me.address) console.log("address exists!")
3  else console.log("address is missing")
```

JavaScript is dynamic and weakly typed!

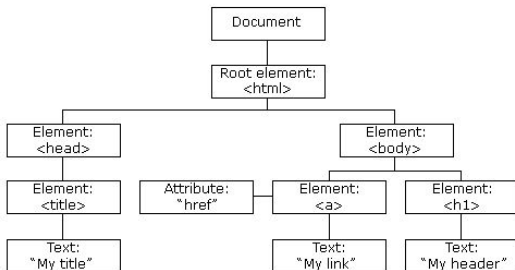
- Be careful! JavaScript was designed to not throw error as far as it could. So, given an ambiguous instruction, it will try to guess what you really meant.
- A case in point: WAT
<https://www.destroyallsoftware.com/talks/wat>

Brief Review on HTML & CSS

- HTML defines a document's structure
- CSS defines a document's style

The HTML DOM

- HTML DOM (Document Object Model) is the Web API that allows JavaScript to dynamically change a webpage.
- In JavaScript, the API can be accessed using the `document` object.
- A HTML Document can be represented as a tree:



Selecting an element

- Use the `document.querySelector`³ function.


Syntax

```
1 | element = document.querySelector(selectors);
```

Parameters

`selectors`

A [DOMString](#) containing one or more selectors to match. This string must be a valid CSS selector string; if it isn't, a `SYNTAX_ERR` exception is thrown. See [Locating DOM elements using selectors](#) for more about selectors and how to manage them.

 **Note:** Characters which are not part of standard CSS syntax must be escaped using a backslash character. Since JavaScript also uses backspace escaping, special care must be taken when writing string literals using these characters. See [Escaping special characters](#) for more information.

Return value

A [Element](#) object representing the first element in the document that matches the specified set of [CSS selectors](#).

If you need a list of all elements matching the specified selectors, you should use [querySelectorAll\(\)](#) instead.

³<https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector>

Example of `document.querySelector`

```
1 document.querySelector("audio[data-key='65']");
```

What does this do?

- `document.querySelector` will select the first element.
- CSS selector string `"audio[data-key='65']"`
 - an element with an audio tag
 - whose data-key attribute is 65

The <audio> tag and data-* attribute

```
<audio data-key="65" src="sounds/clap.wav"></audio>
```

- The <audio> tag is used to embed sound content in documents, containing one or more audio sources specified in the src attribute or with a <source> elements.
- data-* attribute is a new feature introduced in HTML5. It is for extensibility purposes, allowing us to store extra information on standard HTML elements.

Playing audio file

- An `<audio>` element provides a method to play the audio it contains: `audioElement.play()`;
- Thus, to play audio, we can do this:

```
1 let clap =  
  ↪ document.querySelector("audio[data-key='65']");  
2 clap.play();  
3 let hihat =  
  ↪ document.querySelector("audio[data-key='83']");  
4 hihat.play();  
5 let kick =  
  ↪ document.querySelector("audio[data-key='68']");  
6 kick.play();  
7 // So on and so forth
```

DRY! (Don't Repeat Yourself)

- Remember your CS1010/CS1101S! Abstraction!

```
1 function playSound(keyCode) {  
2   let audio =  
    ↪ document.querySelector("audio[data-key='" +  
    ↪ keyCode + "']");  
3   audio.play();  
4 }  
5  
6 playSound(65);
```

- Now try running `playSound(65);` in rapid succession. The same audio waits until it is finished before playing again!

Starting audio before the previous play finishes

- How do you solve this? Use the HTML DOM, `HTMLMediaElement.currentTime`⁴!

```
1 function playSound(keyCode){
2   let audio =
  ↪ document.querySelector("audio[data-key='" +
  ↪ keyCode + "']");
3   audio.currentTime = 0; // Add this
4   audio.play();
5 }
6
7 playSound(65);
8 playSound(65);
```

⁴<https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement/currentTime>

CSS class

- Recall how we apply styles to HTML documents: by including a CSS stylesheet, and then adding appropriate class attributes to the HTML elements.
- For example, each key in the drum kit has class key

```
<div data-key="65" class="key">  
  <kbd>A</kbd>  
  <span class="sound">clap</span>  
</div>  
  
.key { /* various styles */  
.playing {  
  transform: scale(1.1);  
  border-color: #ffc600;  
  box-shadow: 0 0 1rem #ffc600;  
}
```

Adding/Removing a CSS class

- Now, what we want to do is to add a CSS class playing when the audio is playing, and then remove the class when the key has been scaled up.
- This can easily achieved through a HTML DOM method `Element.classList`⁵

```
1      let clapKey =  
    ↪   document.querySelector("div[data-key='65']");  
2      clapKey.classList.add('playing');  
3      clapKey.classList.remove('playing');
```

⁵<https://developer.mozilla.org/en-US/docs/Web/API/Element/classList>

Events and Listeners

- As mentioned in slide 4, JavaScript is event-driven.
- Analogy:

| (Events) When the customers re-requests for... | (Listeners) Inform these people... |
|---|---|
| Spaghetti | Chef |
| Washroom | Toilet manager |
| Pizza | Pizza Hut, Canadian Pizza, Domino's |

- Event:** signal from the browser that something has happened. The browser then conveys this signal to all **listeners** of that event.

Callback functions as Listener

- In JavaScript, we have callback functions as listeners that is invoked whenever an event occurs.
 - To register a function as a listener, we use the HTML DOM function `document.addEventListener(eventType, callback)`
- ```
1 document.addEventListener('keydown', () => {
2 console.log(event);
3 });
```
- `console.log()` is the equivalent of print in other languages.

## Putting everything together

When user hits the key:

1. Play the sound associated with the key
2. At keypress, add `.playing` class to the `<div>` associated with the key
3. When it has been completely scaled, remove `.playing` class from `<div>`



## Play the sound associated with they key

```
1 function playSound(keyCode){
2 let audio =
 ↪ document.querySelector("audio[data-key='" +
 ↪ keyCode + "']");
3 audio.currentTime = 0;
4 audio.play();
5 }
```

## At keypress, add .playing class

```
1 function playSound(keyCode) {
2 let audio =
 ↪ document.querySelector("audio[data-key='" +
 ↪ keyCode + "']");
3 let key = document.querySelector("div[data-key='" +
 ↪ keyCode + "']");
4 key.classList.add('playing');
5 audio.currentTime = 0;
6 audio.play();
7 }
```

## Filter for Bad Input

- data-key is represents the ASCII code of the keys.
- If the key pressed is not any of the keys in the HTML document, then do nothing.

```

1 function playSound(keyCode) {
2 let audio =
 ⇨ document.querySelector("audio[data-key='" +
 ⇨ keyCode + "']");
3 let key = document.querySelector("div[data-key='" +
 ⇨ keyCode + "']");
4 if (audio !== null) {
5 key.classList.add('playing');
6 audio.currentTime = 0;
7 audio.play();
8 }
9 }
```

## Listeners on multiple elements

- We can do so using `document.querySelectorAll`<sup>6</sup> and `Array.forEach`<sup>7</sup>

```
1 let keys = document.querySelectorAll('.key');
2 keys.forEach(key =>
 ↪ key.addEventListener('transitionend', event
 ↪ => {
 ↪ event.target.classList.remove('playing');
 ↪ }));
```

---

<sup>6</sup><https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll>

<sup>7</sup>[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/forEach](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach)

## Talk to us!

- **Feedback form:** <https://tinyurl.com/HS2018JS>
- **Upcoming hackerschool:**  
Introduction to Machine Learning Part 1