# Financial Software Engineering
# Lecture 4

Co-Pierre Georg
AIFMRM

July 18, 2018

## Today

1. Javascript
2. Document object model
3. Jquery

## Javascript

## Recap

- From the last lecture we know that
- $\rightarrow$ HTML and CSS represent markup languages that allow us to create and typeset content on a webpage
- $\rightarrow$ JavaScript (JS) represents a way to add interactivity and reactivity to our webpages
- $\rightarrow$ JS represents a complete programming language and we therefore have access to all of the programming functionality we learnt about
- $\rightarrow$ Since JS is an OOP language, we can implement many of the principles we learnt about with Python

## Document object model

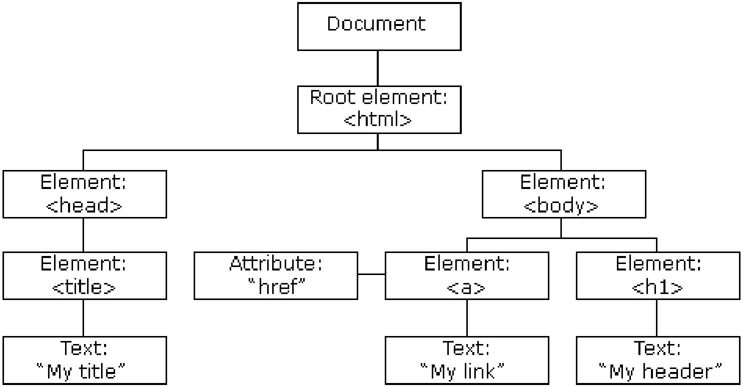
## Document object model

- Up to now, we've used connecting JS to our HTML and CSS and used it to prompt users, take inputs, provide alerts etc.
- Currently however, our JS cannot change our HTML or CSS
- Perhaps we wanted our webpage to react to a user input and change some of it's features accordingly
- To provide this reactive functionality we'll make use of the **Document object model** or DOM
- $\rightarrow$ allows us to use our JS code to <u>interact</u> with HTML and CSS

## Document object model

- When we upload our HTML to the browser, the browser console automatically creates a DOM
- DOM $\rightarrow$ a representation of our HTML code
- Importantly, this representation of our HTML allows us to grab things using JS
- Every box in HTML has a corresponding object in the DOM
- DOM $\rightarrow$ programming interface which converts HTML into a tree structure where each mode represents an object

# Document object model (DOM)



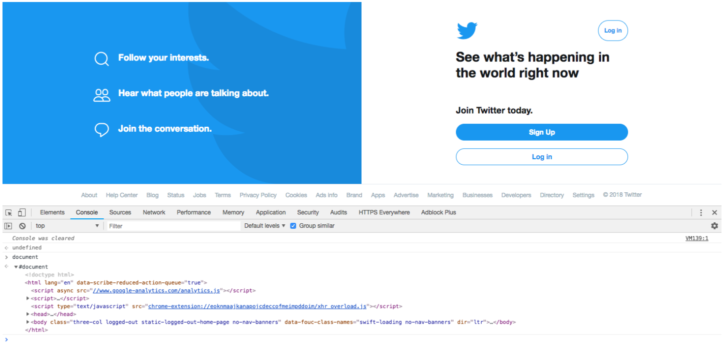
Source: w3schools

## Document object model (DOM)

- By reconstructing HTML tags as JS objects, the DOM allows us to use JS to manipulate HTML and CSS
- JS can now add/change/remove HTML elements, attributes and CSS as well as react to HTML events etc.
- Since we are back in the OOP paradigm, we are now able to work with attributes and objects like before
- The DOM is large; we'll only look at the most commonly used objects and rely on the documentation for everything else

## Document object model (DOM)

- DOM can accessed by going to any website console and typing `document`

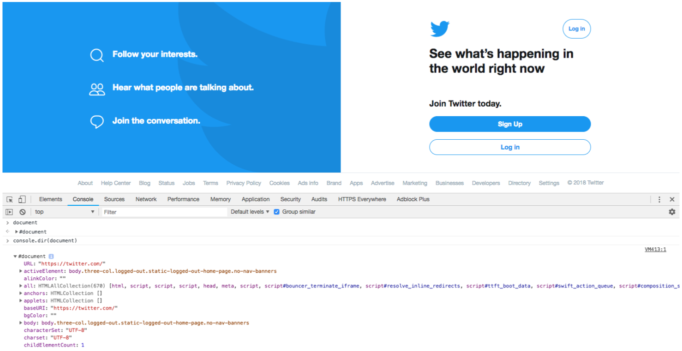

Follow your interests.
Hear what people are talking about.
Join the conversation.
Log in
See what's happening in the world right now
Join Twitter today.
Sign Up
Log in
About Help Center Blog Status Jobs Terms Privacy Policy Cookies Ads info Brand Apps Advertise Marketing Businesses Developers Directory Settings © 2018 Twitter
Elements Console Sources Network Performance Memory Application Security Audits HTTPS Everywhere Adblock Plus
top
Filter
Default levels
Group similar
VM139:1

```
Console was cleared
<· undefined
> document
<· ▾#document
      <!doctype html>
      <html lang="en" data-scribe-reduced-action-queue="true">
        <script async src="//www.google-analytics.com/analytics.js"></script>
       ▸<script>…</script>
        <script type="text/javascript" src="chrome-extension://eoknmaajkanapojcdeccofmeimpddoim/xhr_overload.js"></script>
       ▸<head>…</head>
       ▸<body class="three-col logged-out static-logged-out-home-page no-nav-banners" data-fouc-class-names="swift-loading no-nav-banners" dir="ltr">…</body>
      </html>
>
```

Twitter DOM in the browser console

## Document object model (DOM)

- Then, by typing `console.dir(document)`, we get the DOM as a JS object


Follow your interests.
Hear what people are talking about.
Join the conversation.
Log in
See what's happening in the world right now
Join Twitter today.
Sign Up
Log in
About Help Center Blog Status Jobs Terms Privacy Policy Cookies Ads info Brand Apps Advertise Marketing Businesses Developers Directory Settings © 2018 Twitter
Elements Console Sources Network Performance Memory Application Security Audits HTTPS Everywhere Adblock Plus
top
Filter
Default levels
Group similar

```
> document
<· ▸#document
> console.dir(document)
                                                                                                  VM413:1
  ▾#document
     URL: "https://twitter.com/"
    ▸activeElement: body.three-col.logged-out.static-logged-out-home-page.no-nav-banners
     alinkColor: ""
    ▸all: HTMLAllCollection(670) [html, script, script, script, head, meta, script, script#bouncer_terminate_iframe, script#resolve_inline_redirects, script#ttft_boot_data, script#swift_action_queue, script#composition_s
    ▸anchors: HTMLCollection []
    ▸applets: HTMLCollection []
     baseURI: "https://twitter.com/"
     bgColor: ""
    ▸body: body.three-col.logged-out.static-logged-out-home-page.no-nav-banners
     characterSet: "UTF-8"
     charset: "UTF-8"
     childElementCount: 1
```

Twitter DOM JS object in the browser console

## Document object model (DOM)

- Then, by typing `console.dir(document)`, we get the DOM as a JS object
- Now, we can access attributes like
  - `document.URL`
  - `document.body`
  - `document.links`
- and methods like
  - `document.getElementByID()`
  - `document.getElementByClassName()`
  - `document.querySelector()`

## Methods

- As was the case with HTML, CSS and Bootstrap, a range of functionality exists → documentation is your friend
- We'll look at two specific methods, which you'll likely use most frequently
- getElementByID → allows to select objects based on their HTML id
- querySelector → allows us to select objects based on their CSS style

## Events and event handling

- At this point we're able to use JS to interact with the DOM
- We do this by specifying how this happens <u>beforehand</u>
- As such we've enabled interactivity
- We however are still unable to trigger <u>reactivity</u>
- $\rightarrow$ trigger certain actions, when certain events happen
- These events could be clicks, double clicks, hovers etc.

## Events and event handling

- We are able to implement these event based actions using one of JS's main methods, an **event listener**
- $\rightarrow$ method belonging to the DOM which checks if a specific event has occurred
- Using an event listener, we can now execute certain actions conditional on an event occurring

```
myvariable.addEventListener(event, func)
```

- These events could be clicks, double clicks, hover, drags etc.
- In the event of a click for example

```
head.addEventListener("click", changeColour)
```

## JS, the DOM and event listeners

Let’s look at some examples

## Jquery

## jQuery

- Like other programming languages, JS has a range of libraries
- Up to now we've used base JS, but a range of libraries exist which provide useful functionality
- We'll focus on one, **jQuery**
- $\rightarrow$ a range of methods and objects that simplify interaction with the DOM
- To get jQuery
- $\rightarrow$ link a CDN (just like bootstrap)
- $\rightarrow$ download .js file from the jQuery website

## jQuery vs base JS

- Major advantage of jQuery is it’s use of $ which gives us a convenient implementation of the querySelector method
- Base JS - return all `h1` elements

```
var headers = document.querySelectorAll('h1');
```

- Using jQuery - return all `h1` elements

```
var headers = $('h1');
```

## jQuery events

Let's look at some examples