Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU44000 Internet Applications

Week 2 Lecture 1

Conor Sheedy

A note on Web Assembly (Wasm)

JavaScript is Interpreted,

- Just-in-Time (JIT) Compiled
- dynamically optimised when it is downloaded to the browser
- Executes very fast,
- but still slow for games, image processing etc

Asm.js

  - Predecessor to wasm, 2013
  - Subset of JavaScript
  - C programmes could be transpiled to asm.js

A note on Web Assembly (Wasm)

Web Assembly, 2017

- programs written in an assembly language (for a VM)
  - The Virtual Machine is written to have many of the features of modern processors
- could be downloaded and executed in the browser at near-native speeds
- It assumes an architecture which is quite similar to modern processor architecture
  - Registers, stacks, instructions …
- JavaScript can call WASM modules and vice-versa
- Original usage, speed sensitive sections of JavaScript compiled to wasm
- Compilers becoming available for C/C++,Rust, Go, Python, Ruby, Elixir
- Opens up client-based programming but also risks further fragmentation in the ecosystem

The Document Object Model

DOM

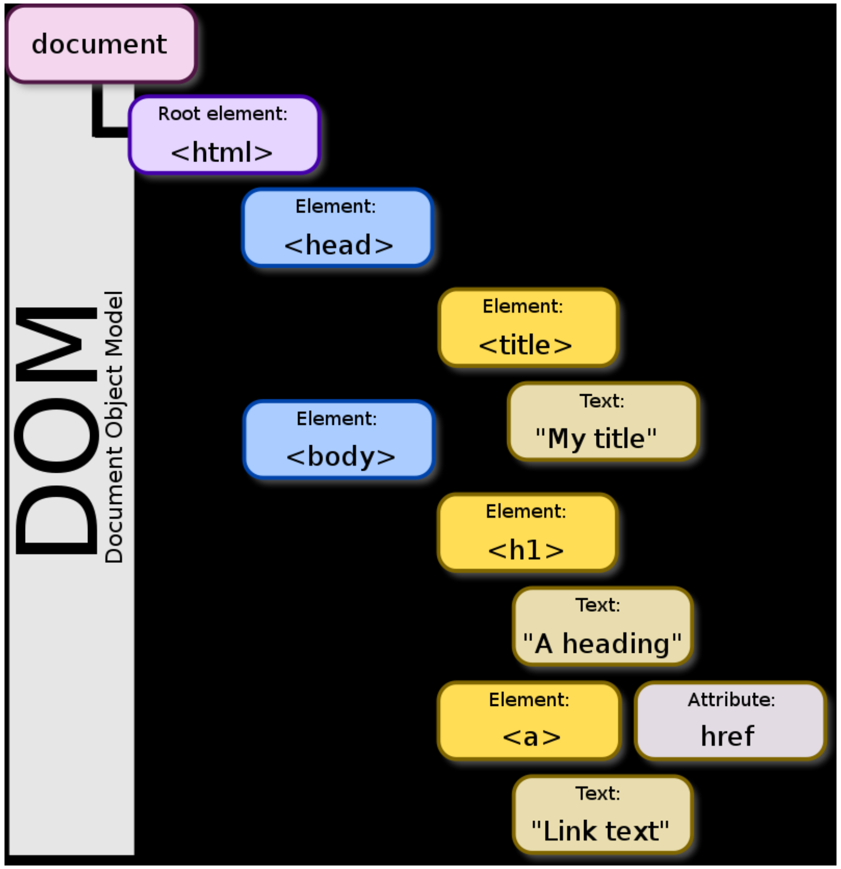
- Microsoft IE,1996
- during the Browser Wars
- to allow ‘client-side’ interactivity

Dynamic HTML

- Netscape Navigator 4.0, 1997
- Similar to DOM

DOM level 1 standard, 1998

- W3C

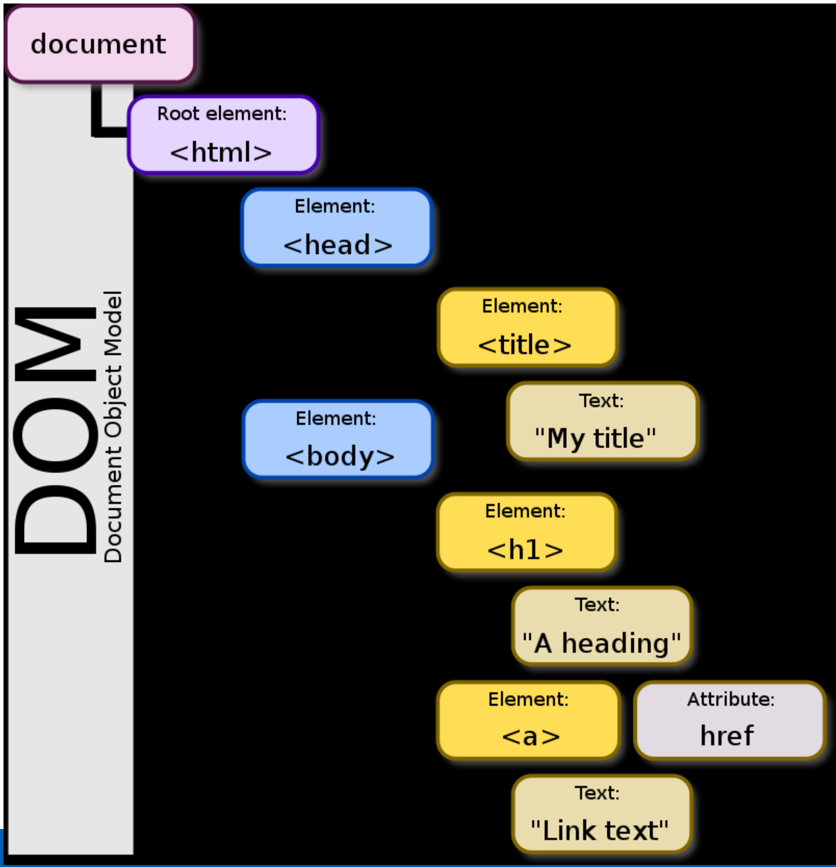


- still issuing standards based on WHATWG (Web Hypertext Application Technology Working Group) snapshots
  - WHATWG control HTML, https://html.spec.whatwg.org/multipage/

The Document Object Model

DOM

- When a browser loads a document (HTML/XML) it parses it and builds a tree in the object ‘document’ and then ‘renders’ it
- JavaScript has full access to add/change elements in this and react to events from it
- The jQuery library built on this in 2006
- other frameworks built on top of that again
- Try changing things in the DOM here: https://developer.chrome.com/docs/devtools/dom/
    - E.g. ‘force state’
    - document.bgColor= "red"



DOM Example

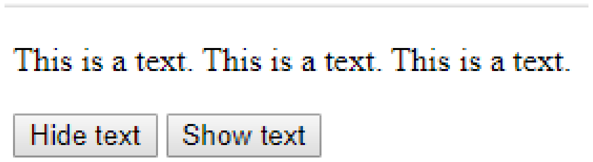
```
<!DOCTYPE html>
<html><body>

<p id="p1">
This is a text.
This is a text.
This is a text.
</p>

<input type="button" value="Hide text"
onclick="document.getElementById('p1').style.visibility='hidden'">

<input type="button" value="Show text"
onclick="document.getElementById('p1').style.visibility='visible'">

</body>
</html>
```



This is a text. This is a text. This is a text.

Defining functions in Javascript

Functions are first-class objects in javascript

- They can be assigned to variables and passed and returned from functions

```
// different ways to define and pass
// functions
function f1add(x,y) { return x+y }

f2add = function (x,y) {return x+y }

f3add = (x,y) => { return x+y;}  // fat arrow

console.log("1 and 2 is "+ f1add(1,2) )
console.log("2 and 3 is "+ f2add(2,3) )
console.log("3 and 4 is "+ f3add(3,4) )
```

The fat-arrow syntax '=>' reflects how frequently
anonymous functions are used

2nd example show .forEach – which takes a fn as arg and applies it to all elements of an array

```
var array1 = ['a', 'b', 'c'];
array1.forEach(function(element) {  console.log(element);});
// expected output: "a“
// expected output: "b“
// expected output: "c"
```

Callbacks

Very many actions in JavaScript involve initiating an action and passing a function that will be called when the action completes

- This is a Call-back function
- Can detract from readability
- Example shows tying JavaScript functions to button ‘events’

```
<!DOCTYPE html>
<html>
<body>
<!button with a call-back function !>

<h1> Buttons and Call-Backs</h1>

<button onclick="myfunction()">
Go On!
</button>
<script>
function myfunction() {alert('you pressed my button')}
</script>

<button onclick="getElementById('demo').innerHTML=Date()">
What is the time?
</button>
<p id="demo"></p>
</body>

</html>
```

Callbacks, Promises and Async.. Await

Promises and Async represent an improvement to the the original way Callbacks were used.

An excellent 45 min video on the topic can be found here:
https://www.youtube.com/watch?v=gB-OmN1egV8

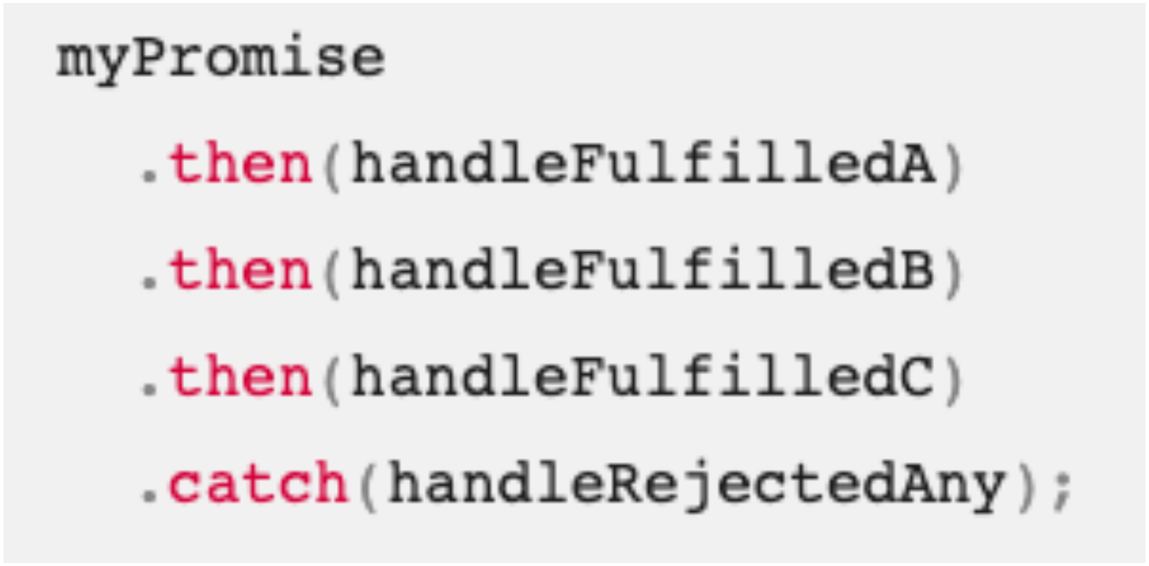
Promise

- Is an object representing the eventual completion or failure of an asynchronous operation.
- Callbacks for success and failure cases are attached to an object using the .then() method
- A Promise can be in one of three states
  - Pending
  - Fulfilled
  - rejected

Promises

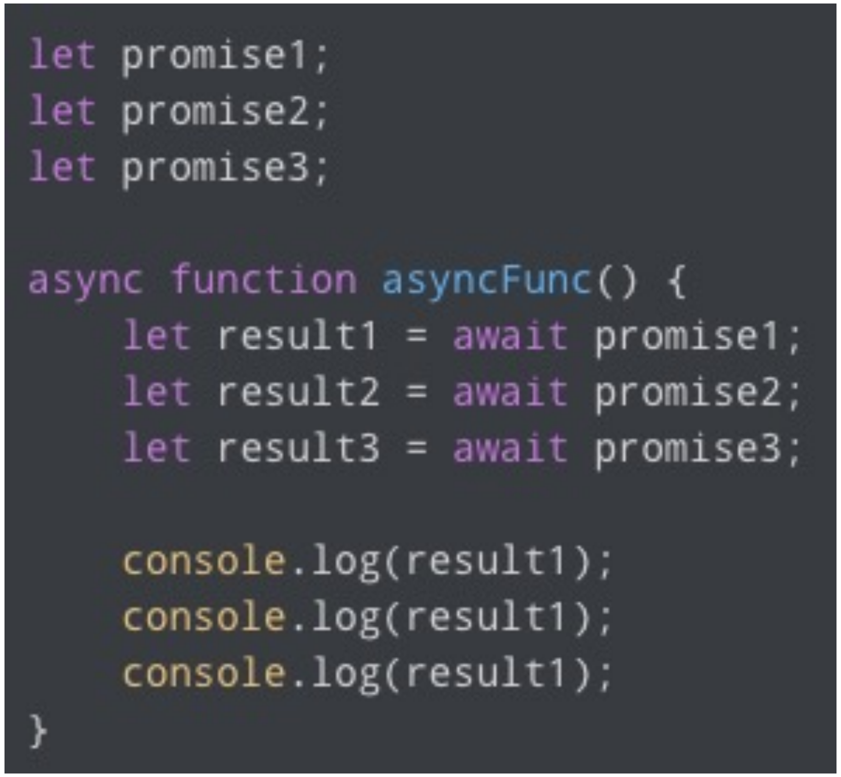
Promises

- Can be chained
- Resulting in a nice syntax for asynchronous events
  - E.g.



```
myPromise
   .then(handleFulfilledA)
   .then(handleFulfilledB)
   .then(handleFulfilledC)
   .catch(handleRejectedAny);
```

Async.. Await



```
let promise1;
let promise2;
let promise3;

async function asyncFunc() {
    let result1 = await promise1;
    let result2 = await promise2;
    let result3 = await promise3;

    console.log(result1);
    console.log(result1);
    console.log(result1);
}
```

Uses Promises

- Another improvement to syntax
- Results in more readable code
  - E.g.

AJAX

If a browser-based Javascript program needed to get information from a server

- the primary way to do this was a page refresh
- Very cumbersome for many type of application e.g. moving around on Google maps

Cured with a JavaScript object called XMLHttpRequest object

- which allowed a script to asynchronously pull other pages at any time
- First implemented by Microsoft in 1999, but not universally deployed until 2002
- Spawned a class of Internet Application called “Asynchronous Javascript and XML (AJAX)”
- Used by Gmail(2004), Google Maps(2005)