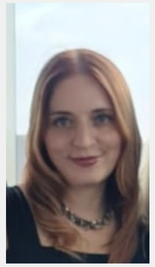
COM3504/6504 The Intelligent Web

Lecture 3: Async programming



Dr Vitaveska Lanfranchi
v.lanfranchi@sheffield.ac.uk
https://www.sheffield.ac.uk/dcs/people/academic/vitaveska-lanfranchi

Mr. Andy Stratton
a.stratton@sheffield.ac.uk
https://www.sheffield.ac.uk/dcs/people/academic/andrew-stratton

Dr Fatima Maikore
F.Maikore@sheffield.ac.uk
https://www.sheffield.ac.uk/dcs/people/research-staff/fatima-maikore

Let’s start

- Why real-time communication?
- What are the main technologies involved
    - Async JavaScript
        - callbacks
        - timeouts
        - Promises

Real-time communication

- Ability to get resources when they are available and add new ones when we want to
- Examples of real-time applications
  - real-time messaging
  - Notifications
  - Live-streaming
- To build a real-time web application it is important to consider
  - Scalability
  - Security
  - **Async programming**
  - **Data Format**
  - **Bi-directional communication**

Scalability and Node.JS

- Node is event-based
- A single process can execute multiple requests
  - handling a huge number of simultaneous connections with high throughput

Security and Node.JS

- Allows to integrate with existing security layers to support SSO (Single Sign-on)
  - identity and access management (IAM) that enables users to securely authenticate with multiple applications and website
  - Module called passport
- Allows control access to data and systems based on the user's role.

Data Formats and Node.JS

- Streaming data consists of a series of time-stamped data packets.
- Node.JS provides an application programming interface to work with streams and allows creating readable and writable data streams
  - module called stream

ASYNC PROGRAMMING

What is async programming

- Async programming is a strategy whereby the program is able to control occurrences of events outside the normal flow of the main thread
  - i.e. some code is executed when an event occurs rather than when the flow of programming requires it
- Typical example is the click of a button
  - this raises an event (buttonClick) which is intercepted by the program
  - to execute some code

Why async programming?

- Many Web API features use asynchronous code
  - especially those that access or fetch resources from external devices,
    - e.g. files from the network, data from databases, video streams from a web cam, etc.
- You should learn how because
  - computational resources are scarce,
  - battery power is scarce
  - computing is very expensive

How is it done in JavaScript?

- Async operations are stored into an event queue
- The queue is dealt with after the main thread has finished processing
  - not in parallel as Js is single threaded!!
- The queued operations are completed as soon as possible
- They return their results to the JavaScript environment
- 4 methods
  - Callbacks (classic)
  - Timeouts and Intervals (delayed and repeated execution)
  - Promises (new)
  - Await/async (newer)

CALLBACKS

Callbacks

- JavaScript runs code sequentially in top-down order
  - However there are cases where you want your code to run only after something happens and not sequentially
- A callback is a function that is called only when another function is performed AND finished
- It is a function passed as parameter to another function otherFunction, and is called (or executed) inside the otherFunction.
  - So if the function takes time to finish, the callback is called only at the end

Callbacks – how can they be used?

- Callbacks can be used to perform an action at the end of an asynchronous action
  - i.e. an action that is executed outside the current flow
    - e.g. an action that requires to go to the server to get data
- When you pass a callback function as a parameter you pass just the name of the function without ()

Example

```
var numbers = [1, 2, 3, 4, 5, 6, 7];

function isOddNumber(number) {
    return number % 2;
}

var oddNumbers = numbers.filter(isOddNumber);
console.log(oddNumbers);
```

Explanation

- I have an array of numbers and want to check which ones are odd
- I use the method filter that allows me to filter an array based on a condition
    - The filter method returns a new array
- I create a function isOddNumber(number)
- I pass isOddNumber as parameter to numbers.filter

TIMING EVENTS

JavaScript timing methods

- JavaScript has a series of methods that allow to execute functions in a specified timeframe
- They are ideal for running asynchronous functions
- And for callbacks

setTimeout()

- setTimeout() calls a function or evaluates an expression after a specified number of milliseconds.
- It is a method of the window object
- It is executed only once

setTimeout() – example with callback

```
<button onclick="myFunction()">Try it</button>
<script>
var myVar;
function myFunction() {
  myVar = setTimeout(alertFunc, 3000);
}
function alertFunc() {
  alert("Hello!");
}
</script>
```

setInterval()

- setInterval() calls a function or evaluates an expression at specified intervals (in milliseconds)
- It is a method of the window object
- It is repeatedly executed until stopped
  - clearInterval()
  - Window is closed

setInterval() - example

- setInterval() calls a function or evaluates an expression at specified intervals (in milliseconds)
- It is a method of the window object
- It is repeatedly executed until stopped
    - clearInterval()
    - Window is closed

What will this code output?

```
<script>
   function myFunction()
   {  x=123;
   }
   var x=0;
   myFunction();
   alert(x);
</script>
```

Ex.1 - What will this code output?

```
<script>
   function myFunction()
   {  x=123;
   }
   var x=0;
   myFunction();
   alert(x);
</script>
```

Ex.1 - Explanation

- X is first set sequentially to 0
- Then the function myFunction is called
- X is set to 123
- The alert prints 123

Ex.2 - What will this code output?

```
<script>
function myFunction()
{  x=123;
}
var x=0;
setTimeout(myFunction, 3000);
alert(x);

</script>
```

Ex.2 - Explanation

- X is first set sequentially to 0
- Then the function myFunction is called but on a timeout
- Whilst waiting for myFunction to set the variable of x to 123
  - The alert prints the current value that is 0

Ex.3 - What will this code output?

```
<script>
function myFunction() {
   x=x*2;
   alert(x);
}
var x=1;
setTimeout(myFunction, 3000);
</script>
```

Ex.3 - Explanation

- X is first set sequentially to 1
- Then the function myFunction is called but on a timeout
- In myFunction the value of x is multiplied by 2.
- The alert is inside my function so after the timeout will prints the current value that is 2

Ex.4 - What will this code output?

```
<script>
function myFunction() {
   x=x*2;
   setTimeout(myFunction2, 3000);
   alert(x);
}
function myFunction2() {
   x=1234;
}
var x=1;
setTimeout(myFunction, 3000);
</script>
```

Ex.4 - Explanation

- X is first set sequentially to 1
- Then the function myFunction is called but on a timeout
- In myFunction the value of x is multiplied by 2.
- myFunction2 is called on a setTimout to set the value of x to 1234
- The alert is inside my function so whilst waiting for myFunction2 to set the new value will prints the current value that is 2

Ex.5 - What will this code output?

```
<script>
function myFunction() {
   x=x*2;
   setInterval(myFunction2, 3000);
   alert(x);
}

function myFunction2() {
   x=1234;
}
var x=1;
setTimeout(myFunction, 3000);
</script>
```

A WORLD TOP 100 UNIVERSITY

Ex.5 - Explanation

- X is first set sequentially to 1
- Then the function myFunction is called but on a timeout
- In myFunction the value of x is multiplied by 2.
- myFunction2 is called on a setInterval to repeatedly set the value of x to 1234
- The alert is inside my function so whilst waiting for myFunction2 to set the new value will prints the current value that is 2

JAVASCRIPT PROMISES

What are promises

- Promises are a way to create asynchronous code that allows easy sequencing of async processes
- Following a very familiar structure:
- if x
    - do Y
    - then do W
    - then do Z

Promise status

- A Promise can be:
  - *pending*: initial state
  - *fulfilled*: the operation was completed successfully.
    - Returns a value
  - *rejected*: the operation failed.
    - Returns an error

How promises work

- Declaration
  - it declares a long running computation
  - it declares placeholders for the behaviour to adopt in case of success and in case of error (resolve and reject represent functions passed as parameters
- Consumption
  - it declares the functions actual functions to be used for success/error
- Execution
  - it runs the promise code and calls the actual success/reject function

Promises methods

- Promise.then() takes two arguments
  - a callback for success
  - A callback for failure.
- Promise.catch() deals with the rejected promise
- You can chain promises
- You can combine them with timed operations

Example

```
function myDisplayer(username) {
document.getElementById("demo").innerHTML = username;
}


// declaration
var myPromise = new Promise(function(myResolve, myReject) {
		var x = prompt("What is your name?")
```

Example 2

```
    // consumption
    if (x != null) {
    myResolve(x);
    } else {
    myReject("Error");
    }
    });
//execution
myPromise.then(
function(value) {myDisplayer(value);}
);
myPromise.catch(
  function(error) {myDisplayer(error);}
);
```

Example in ES6

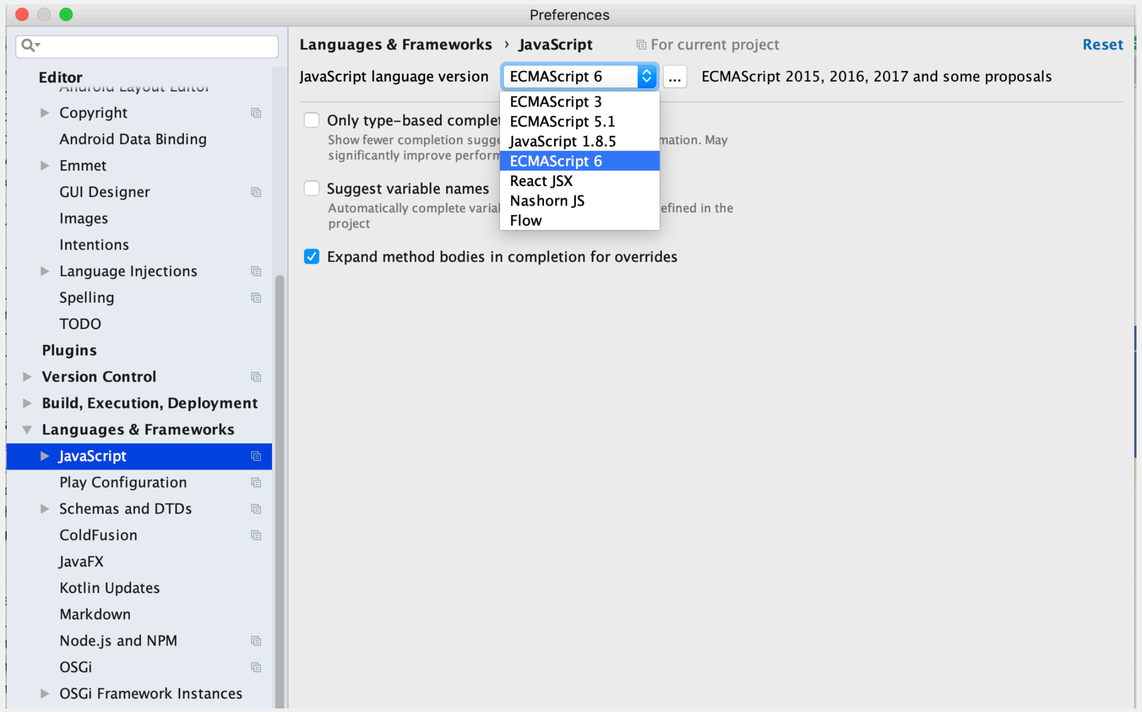
```
let myPromise = time => new Promise((resolve) => setTimeout(resolve, time));

myPromise(3000).then(() => alert('Hello'));

myPromise.catch(
   (error) => console.log(error.message)
);
```

IntelliJ errors with ES6?

- Make sure ES6 is enabled in IntelliJ
  - Preferences -> Languages and Frameworks -> Javascript



Preferences

Editor
- Copyright
- Android Data Binding
- Emmet
- GUI Designer
- Images
- Intentions
- Language Injections
- Spelling
- TODO

Plugins

Version Control

Build, Execution, Deployment

Languages & Frameworks
- JavaScript
- Play Configuration
- Schemas and DTDs
- ColdFusion
- JavaFX
- Kotlin Updates
- Markdown
- Node.js and NPM
- OSGi
- OSGi Framework Instances

Languages & Frameworks › JavaScript — For current project — Reset

JavaScript language version: ECMAScript 6 — ECMAScript 2015, 2016, 2017 and some proposals

- ECMAScript 3
- ECMAScript 5.1
- JavaScript 1.8.5
- ECMAScript 6
- React JSX
- Nashorn JS
- Flow

☐ Only type-based complet

☐ Suggest variable names

☑ Expand method bodies in completion for overrides

Async and await

- Javascript has a keyword async
  - takes as input a function and makes it asynchronous
  - returns a promise
- The keyword await before a function makes the function wait for a promise
- Keeps syntax tidy
- You could for example use it to verify user data from a database

Async and await example

```
const getUserData = async () =>  {
    const response = await prompt('what is your name');
    document.getElementById("username").innerHTML = response;

}
getUserData();
```

Questions

?