Asynchronous Computing using Async/Await

Prof. Fabio Ciravegna
Dipartimento di Informatica
Università di Torino
fabio.ciravegna@unito.it

- Async functions and the await keyword (ECMAScript 2017) are **syntactic sugar** on top of promises,
  - making asynchronous code easier to write and to read afterwards
    - but it is still asynchronous!
  - They make async code look more like old-school synchronous code

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await

Async keyword

- It takes a normal function and returns a promise

```
function hello() { return "Hello" };

hello();
```

```
async function hello() { return "Hello" };
// or, if you prefer:
//       let hello = async () => { return "Hello" };
```

```
hello().then((value) => console.log(value))
```

Await

- putting away in front of a call to an async function, allows to stop the flow of the code until the promise is fulfilled or rejected
  - very useful as the Js thread can execute some other code / react to other events if necessary

```
fetch('coffee.jpg')
.then(response => response.blob())
.then(myBlob => {
  let objectURL = URL.createObjectURL(myBlob);
  let image = document.createElement('img');
  image.src = objectURL;
  document.body.appendChild(image);
})
.catch(e => {
  console.log('There has been a problem with your fetch operation: ' + e.message)
});
```

UNIVERSITÀ DI TORINO

```
async function myFetch() {
  let response = await fetch('coffee.jpg');
  let myBlob = await response.blob();

  let objectURL = URL.createObjectURL(myBlob);
  let image = document.createElement('img');
  image.src = objectURL;
  document.body.appendChild(image);
}


// when we call it we must take care of any error that any promise in the
// body will throw
myFetch()
.catch(e => {
  console.log('There has been a problem with your fetch operation: ' + e.me
});
```

Error handling

- In case of error, you can:
  - return a condition of error intercepted by the calling function
    - e.g. a specific value
  - throw an error
    - in that case it can be captured by the calling function

The downsides of async/await

- Async/await makes your code look synchronous,
  - and in a way it makes it behave more synchronously
- The await keyword blocks execution of all the code that follows until the promise fulfils
  - exactly as it would with a synchronous operation
  - It does allow other tasks to continue to run in the meantime, but your own code is blocked

Promise.all

- Your code could be slowed down by a significant number of awaited promises
  - Each await will wait for the previous one to finish
    - Whereas sometimes you have tasks that can easily run in parallel
      - you may want the promises to begin processing simultaneously, like they would do if we weren't using async/await
      - also you may want that when one fails, all the others are not completed
  - Promise.all does all this
    - promises run in parallel
    - if one fails, the other fail as well
    - one point of error

UNIVERSITÀ
DI TORINO

Example

```
async function delayAll() => {
  await Promise.all([
      Promise.delay(600),
      Promise.delay(600),
      Promise.delay(600)]); //runs all delays simultaneously
};
```

https://stackoverflow.com/questions/45285129/any-difference-between-await-promise-all-and-multiple-await

```
delayAll()
    .then(function(arrayOfValuesOrErrors) {
        // handling ok case
    })
    .catch(function(err) {
      console.log(err.message); // some coding error in handling happened
    });
```

https://stackoverflow.com/questions/30362733/handling-errors-in-promise-all

Promise.all with an array

- You can use Promise.all to provide a number of promises depending on a value at run time
  - e.g. the promises are contained in an array, e.g.

```
const p1 = getSomeValueAsync();
const p2 = getSomeValueAsync();
const [result1, result2] = await Promise.all([p1, p2]);
console.log(result1 + result2);
```

  - So this means that one of the examples we initially used in the callback from hell can be implemented very easily

```
let promiseList= [];
for (photo in photoList){
    promiseList.add(function(photo){
         photoFile= downloadFile(photo)
         return photoFile;
  };
Promise.all(promiseList). then( values =>
     console.log(values);  // this is an array of values returned by
each promise
```

Errors

- Error handling must be implemented using the reject branch of each promise and then collected with catch by the calling function
  - only if failure in one causes failure in all the others
  - if failure in one is not relevant to the others, then do not reject.
    - just resolve with a form of null value

What you have learned

- You should be confident in using async/await as a way to simplify writing code involving promises
- In week 4 we will use the construct considerably

UNIVERSITÀ DI TORINO

Questions?