

UMassAmherst

Manning College of Information
& Computer Sciences

Programming Methodology

Lab 10: Asynchronous Programming

Wednesday, April 17th, 2024



Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises
- Discussion leaders will walk around and answer questions
- Solutions to exercises will be reviewed as a class
- Attendance taken at the end

Reminders

- Homework 6 is due tonight at 11:59pm
 - Come to [office hours](#) for help!
- Get in touch with your team for HW7 if you haven't already!
- HW7 is due Thursday, May 2nd at 11:59pm

Today's Goals

- Practice working with asynchronous programming

Promises: then()

`Promise.prototype.then(onFulfill, onReject)` (onReject is optional)

- When the promise resolves (fulfilled/rejected), the corresponding handler is called
 - argument passed to `onFulfill` is *fulfillment value*
 - argument passed to `onReject` is *rejection reason*

Promises: then()

`.then(...)` returns a new Promise, `p`

If the handler returns (the function passed to `.then()`):

- **A value:** `p` gets fulfilled with the value
- **A pending promise, `a`:** the resolution of `p` is subsequent to the resolution of `a`
 - If `a` fulfills, then `p` fulfills with the same value
 - If `a` rejects, then `p` rejects with the same value
- **A resolved promise:** `p` is fulfilled/rejected with the resolved Promise's fulfillment/rejection value respectively

Promises: `resolve()`, `reject()`, `catch()` UMassAmherst

Manning College of Information
& Computer Sciences

`Promise.resolve(value)`: “resolves” a value to a Promise

if value is a Promise, value is returned

otherwise, resolve returns a Promise fulfilled with that value

`Promise.reject(reason)`: returns a Promise that is rejected with a given reason.

Best practice: reason is an Error for debugging and error catching

`Promise.prototype.catch(onReject)`:

`p.catch(onReject)` is the same as `p.then(undefined, onReject)`

(no handler for fulfill, just for reject)

Promises: `all()`, `any()`, `race()`

`Promise.all(promiseArr)`: returns a Promise that asynchronously:

- fulfills when all promises in the array fulfill (or when `promiseArr` is empty)
- rejects when any of the promises rejects, with the first reject reason

`Promise.any(promiseArr)`: returns a Promise that asynchronously:

- fulfills when any promise in the array fulfills, with its fulfillment value
- rejects when all of the promises reject (or when `promiseArr` is empty), with an `AggregateError` of the reject reasons

`Promise.race(promiseArr)`: returns a Promise that settles with the eventual state of the first Promise in `promiseArr` that settles

`Promise.allSettled(promiseArr)`: returns a Promise that always fulfills with an array of objects describing the outcome of each input promise:

```
{ status: "fulfilled" | "rejected", value?: someType, reason?: errType }
```


Exercise 1: Promises

Write a function `getObjsWithName` that takes in an array of URLs and returns a promise of an array of objects. The array of objects should contain all objects with the property “name” found at every URL that points to JSON data.

Assume (1) All urls point to valid JSON data.

(2) Assume the data is an array of objects.

Hint: You may use “in” to check properties

Exercise 1: Promises

Write a function `getObjsWithName` that takes in an array of URLs and returns a promise of an array of objects. The array of objects should contain all objects with the property “name” found at every URL that points to JSON data.

`fetch()` will return a promise, which may resolve or reject
=> we get an array of Promises

Use `response.json()` to get JSON representation of the GET response.
=> `json()` returns a Promise

Exercise 1: Promises

... returns a Promise of an array of objects. The array of objects should contain all objects with the property “name” found at every URL that points to JSON data.

Wrap result in `Promise.allsettled`.

=> Call to `Promise.allsettled` always fulfills with an array of objects.

```
{ status: "fulfilled" | "rejected", value?: someType, reason?: errType }
```

Use `filter` and `in` to check if object has property `name`.

For each url, we have an of array of objects with “name” property

Need to combine results from all urls into one, i.e, go from `Object[][]` => `Object[]`

=> flatten the result into a single array using `flat()`

Testing your function (included with starter code)

UMassAmherst

Manning College of Information
& Computer Sciences

Write a function `getObjsWithName` that takes in an array of URLs and returns a promise of an array of objects. The array of objects should contain all objects with the property “name” found at every URL.

```
const urls: string[] = ['https://api.github.com/users/umass-compsci-220/repos',  
  'https://api.github.com/users/umass-cs-230/repos'];  
  
//Printing the “name” property of objects  
getObjsWithName(urls)  
  .then(obs => obs.map(obj => obj["name"]))  
  .then(console.log);  
  
["public-materials", "umass-compsci-220.github.io", "230-code-examples", "230-lab-bank-simulator",  
  "230-lab-bash-scripts", "230-lab-binary-bomb", "230-lab-bits-and-bytes", "230-lab-cache",  
  "230-lab-escape", "230-lab-grep-wc", "230-lab-threads", "230_gtest", "binary_bomb_project",  
  "huffman_project"]
```

Exercise 2: Compose Functions

UMassAmherst

Manning College of Information
& Computer Sciences

Write a function that takes an array of **asynchronous** functions and returns a function that is their composition.

Example: Composition of 3 **synchronous** functions

`composeFunctions([f, g, h])`

Composition of functions is a function: `(x) => h(g(f(x)))`

Apply first function to the input `f(x)`

Apply second function to the result of the first `g(f(x))`

Apply third function to the result of the second `h(g(f(x)))`

Exercise 2: Compose Functions

UMassAmherst

Manning College of Information
& Computer Sciences

Write a function that takes an array of **asynchronous** functions and returns a function that is their composition.

Exercise asks composition of asynchronous functions.
=> each function takes in a value and returns a Promise.

The returned function should compose the functions in the array, applying them successively, left to right, and return a promise that resolves to the result or rejects with the reason of the first promise that rejects.

Example: Composition of 3 asynchronous functions

```
composeFunctionsAsync([f, g, h])
```

```
(x) => f(x).then(g).then(h)
```

where `f(x)` returns a Promise

then apply `g` to its fulfillment value, and produce another Promise

then apply `h` to that fulfillment value, and produce a Promise with the result

Testing your function (included with starter code)

UMassAmherst

Manning College of Information
& Computer Sciences

```
let getJSON = composeFunctionsAsync([fetch,
    (fetchres : Response) => fetchres.ok ? fetchres.json() Promise.reject("Error with fetching")])

//Print the first objects "owner" property?
getJSON('https://api.github.com/users/umass-cs-230/repos').then(res => res[0]["owner"]).then(console.log);
{
  "login": "umass-cs-230",
  "id": 1156163,
  "node_id": "MDEyOk9yZ2FuaXphdGlvbjExNTYxNjM=",
  "avatar_url": "https://avatars.githubusercontent.com/u/1156163?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/umass-cs-230",
  ...
  "type": "Organization",
  "site_admin": false
}
```