Școala
informală
de IT

Promises

Agenda

- Recap
- Callback Functions (revisited)
- Javascript Engine: How It Works
- Promises
- Fetch API

Recap

Recap

- API
- AJAX
- JSON

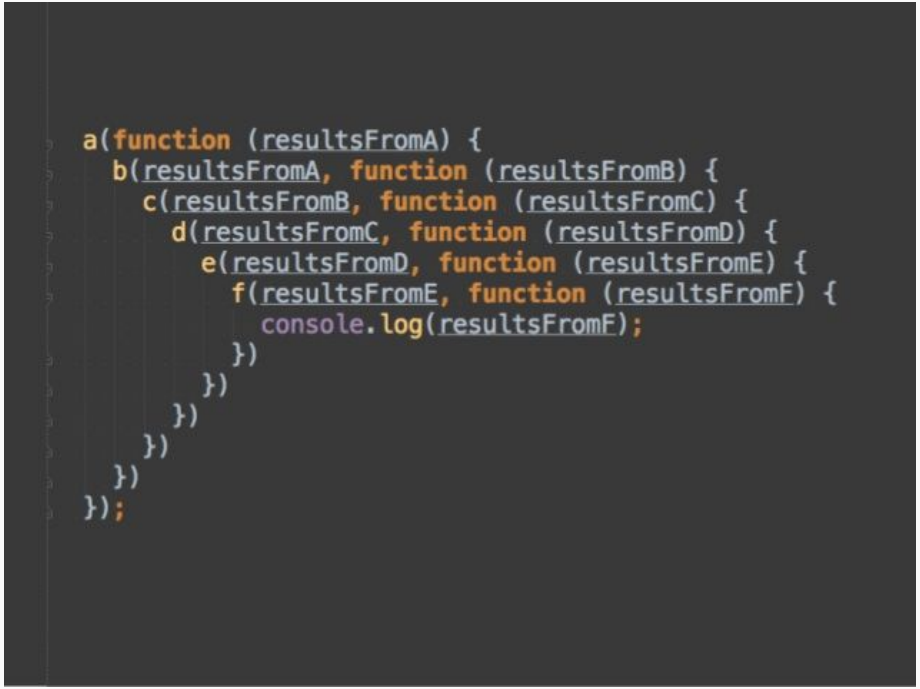
Callback Functions (revisited)

Callback Functions

- Javascript is single threaded and follows the async model
- A callback function
  - is a function that is passed to another function as a parameter (also known as a higher-order function)
  - it is called (or executed) in the future (when it's invoked by that other function), not immediately

Callback Hell / The Pyramid of Doom

- One of the biggest problems of callbacks: the chaining of different asynchronous activities
- You end up calling function after function to pass around values
- The result is an unmaintainable "callback hell"
  - => PROMISES



```
a(function (resultsFromA) {
  b(resultsFromA, function (resultsFromB) {
    c(resultsFromB, function (resultsFromC) {
      d(resultsFromC, function (resultsFromD) {
        e(resultsFromD, function (resultsFromE) {
          f(resultsFromE, function (resultsFromF) {
            console.log(resultsFromF);
          })
        })
      })
    })
  })
});
```

Javascript Engine: How It Works

Event Loop

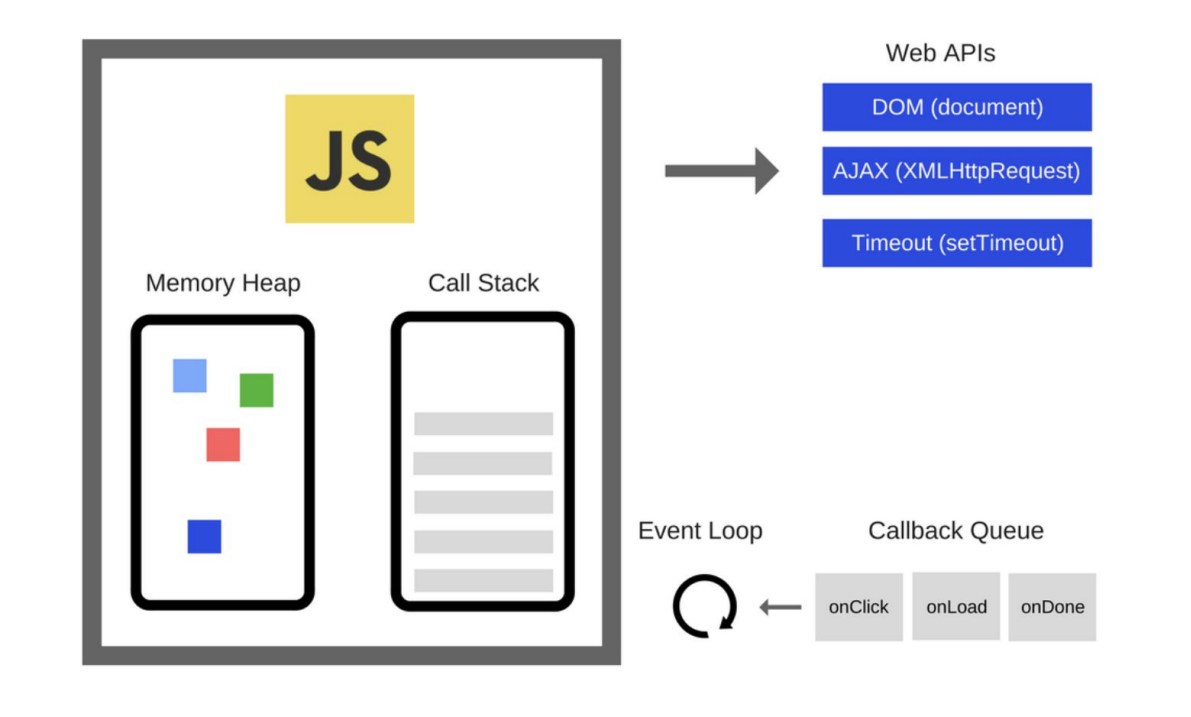


Image Source

Promises

Promise Definition

- A promise is an object that may produce a value some time in the future
- It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.
- Analogies:
  - Promises from parents
  - Buying a salad or a sandwich from Panemar

Promise States

- 3 states:
  - Pending - initial state, not fulfilled, nor rejected
  - Fulfilled / Resolved - the operation completed successfully
  - Rejected - the operation failed
- Settled promise = not pending (either fulfilled or rejected)

How promises work

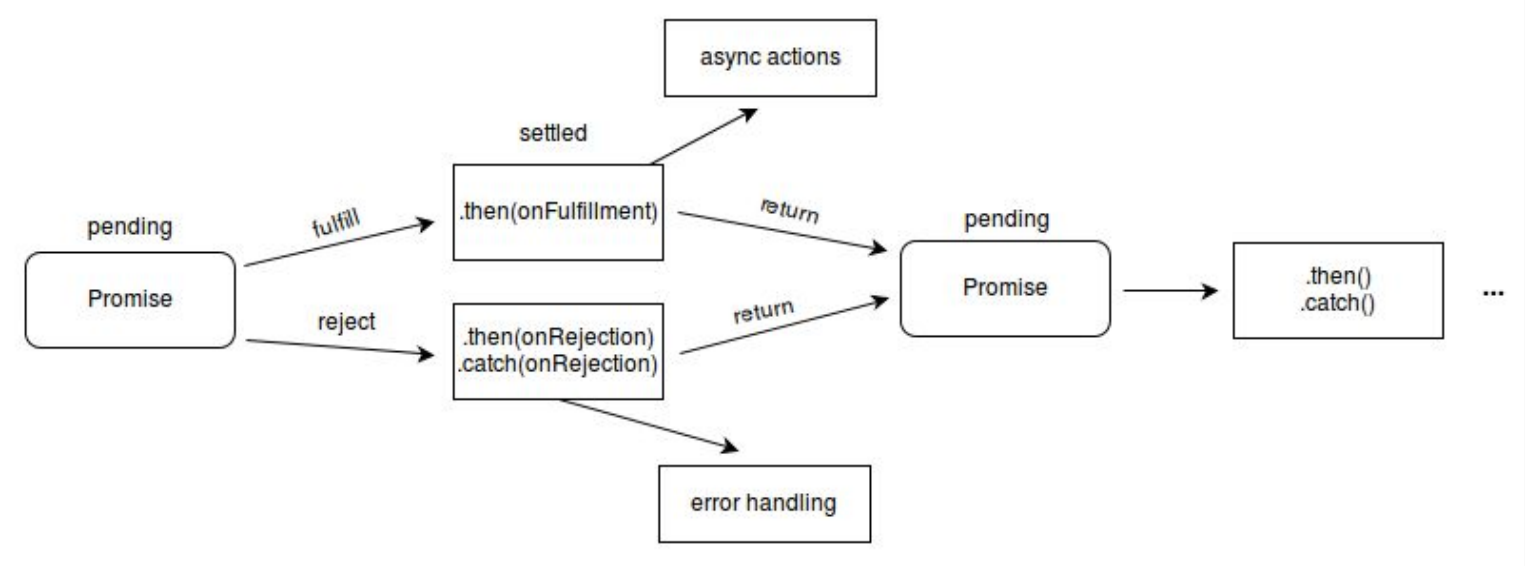
- **Essentially, a promise is an object to which you attach callbacks, instead of passing callbacks into a function. (MDN)**

```
function doSomethingAsync(params, callback) {                    // old
    processParams(); // do something here with the params
    callback(); // call the callback
}
doSomethingAsync(params, callback);

function doSomethingAsync(params) {                              // new
    return processParams(); // processParams should return a promise
}
doSomethingAsync(params).then(callback);
```

Promise Chaining

- .then(onFulfilled, onRejected)
  - Returns a promise in the pending status
- .catch(onRejected)



Let’s Practice

Practice

- Using the previous code, separate the API calls into their own file
- Use promises to:
    - Avoid the callback hell
    - Handle errors properly

Fetch API

Verifying if response is successful (Fetch API)

```
fetch(root + '/posts/1', { method: 'GET'})
  .then(function(response) {
    if (response.ok) {
      return response.json();
    }
    throw new Error('Network response was not ok.');
  })
  .then(function(jsonResp) {
    console.log(jsonResp);
  })
  .catch(function(error) {
    console.log("There was a network error", error);
  });
```

Resources

https://www.quora.com/What-is-callback-hell

http://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/

https://scotch.io/tutorials/javascript-promises-for-dummies

https://medium.com/javascript-scene/master-the-javascript-interview-what-is-a-promise-27fc71e77261

https://dev.to/azizhk110/javascript-promise-chaining--error-handling?utm_content=buffer50085&utm_medium=social&utm_source=facebook.com&utm_campaign=buffer

https://medium.com/cameron-nokes/4-common-mistakes-front-end-developers-make-when-using-fetch-1f974f9d1aa1

https://devhints.io/js-fetch

WATCH: https://www.youtube.com/watch?v=8aGhZQkoFbQ