Software Engineering in the Arts and Humanities

JavaScript, continued

September 16, 2019

Functions

```
function hello()
{
    alert('Hello, world!');
}
```

Functions

```
function ()
{
    alert('Hello, world!');
}
```

Arrow Functions

```
() => {
    alert('Hello, world!');
}
```

Arrow Functions

```
(x) => {
    alert(x);
}
```

Arrow Functions

```
x => {
    alert(x);
}
```

Arrow Functions

```
x => x * 2
```

jQuery

jQuery

- Before ES6 became popular, a common library known as jQuery was often used by developers because it made DOM traversal a bit more concise (albeit at the expense of being more cryptic.)

jQuery

- Before ES6 became popular, a common library known as jQuery was often used by developers because it made DOM traversal a bit more concise (albeit at the expense of being more cryptic.)

- Largely it has fallen out of favor since ES6 has made streamlining improvements, but you'll still often find it. In particular, Bootstrap's JS components rely on jQuery.

jQuery

- Before ES6 became popular, a common library known as jQuery was often used by developers because it made DOM traversal a bit more concise (albeit at the expense of being more cryptic.)

- Largely it has fallen out of favor since ES6 has made streamlining improvements, but you'll still often find it. In particular, Bootstrap's JS components rely on jQuery.

- Documentation and downloads at https://jquery.com/

Local Storage

- `localStorage.getItem(key);`
- `localStorage.setItem(key, value);`

Asynchronicity

- In most of the examples we've talked about, the JavaScript code has been running top-to-bottom as it's encountered.

Asynchronicity

- In most of the examples we've talked about, the JavaScript code has been running top-to-bottom as it's encountered.

- Normally this isn't a problem, but it can be a problem if one of the functions we need to execute might take a long time (e.g., a network call).

Asynchronicity

```
const data = fulfillRequest();

console.log(data);
...
```

Asynchronicity

```
const data = fulfillRequest();
```

```
console.log(data);
...
```

Asynchronicity

```
const data = fulfillRequest();
🕐🕐
console.log(data);
...
```

Asynchronicity

```
const data = fulfillRequest();
 🕐🕑🕒
console.log(data);
...
```

Asynchronicity

```
const data = fulfillRequest();
 🕐🕑🕒🕓🕔🕕🕖🕗🕘🕙🕚🕛
console.log(data);
...
```

Asynchronicity

```
const data = fulfillRequest();
 🕐🕑🕒🕓🕔🕕🕖🕗🕘🕙🕚🕛
console.log(data);
...
```

Asynchronicity

```
fulfillRequest()
.then(data => data.parse())
.then(results => console.log(results))
...
```

Asynchronicity

```
fulfillRequest().then(data => data.parse()).then(results => console.log(results))
```

...

Asynchronicity

```
fulfillRequest()
.then(data => data.parse())
.then(results => console.log(results))
...
```

Asynchronicity

```
fulfillRequest()
.then(data => data.parse())
.then(results => console.log(results))
...
```

If you see a structure like this somewhere, this is indicative of what's known as a JavaScript **promise**, a mechanism for ensuring orderly execution of asynchronous code.

Additional Requests

- Using JavaScript, it is possible for our code to make supplementary HTTP requests without reloading the page.

Additional Requests

- Using JavaScript, it is possible for our code to make supplementary HTTP requests without reloading the page.

- This technique is commonly known as Ajax, and you may have done it before using `XMLHttpRequests`.

Additional Requests

- Using JavaScript, it is possible for our code to make supplementary HTTP requests without reloading the page.

- This technique is commonly known as Ajax, and you may have done it before using `XMLHttpRequests`.

- In ES6, one of the main mechanisms we'll use to achieve this with a promise is `fetch()`.

APIs

API

- **A**pplication **P**rogramming **I**nterfaces are "contracts" of a sort between a client (us) and, usually, a data provider, to give our applications the ability to access data that may be useful to us in some way.

API

- **A**pplication **P**rogramming **I**nterfaces are "contracts" of a sort between a client (us) and, usually, a data provider, to give our applications the ability to access data that may be useful to us in some way.

- In this course, we'll be using APIs from many different service providers and creating projects that leverage data from those providers.

API

- **A**pplication **P**rogramming **I**nterfaces are "contracts" of a sort between a client (us) and, usually, a data provider, to give our applications the ability to access data that may be useful to us in some way.

- In this course, we'll be using APIs from many different service providers and creating projects that leverage data from those providers.

- Learning to parse API docs will be a crucial skill!

Fixer.IO

Google Books