CA: User Generation

Client-side programming

John Snel

Topics

Today’s Agenda

- Arrays
- Arrow functions
- Template strings
- JSON
- JS Web APIs
- Lab activity next week
- JQuery (time permitting)

Resources

- Fetch API Introduction
- {JSON} Placeholder
- Random User API

Terms

- Asynchronous: functions running in parallel with other functions are called asynchronous
- A callback: a callback is a function passed as an argument to another function.
- A promise: a promise is a returned object to which you attach callbacks, instead of passing callbacks into a function.

Arrays

The Array object is used to store multiple values in a single variable:

Arrays

The Array object is used to store multiple values in a single variable:

```
const cars = ["Saab", "Volvo", “BMW"];
console.log(cars[0]);

// expected output: “Saab”
```

Arrays

The Array object is used to store multiple values in a single variable:

```
const cars = ["Saab", "Volvo", “BMW"];
document.getElementById("demo").innerHTML = cars[0];

// To the DOM: “Saab”
```

Arrays

The Array object is used to store multiple values in a single variable:

```
const cars = ["Saab", "Volvo", “BMW"];
cars.forEach(element => console.log(element));

// expected output: "Saab"
// expected output: "Volvo"
// expected output: “BMW”
```

Arrow Functions

An **arrow function expression** is a compact alternative to a traditional function expression, but is limited and can't be used in all situations.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Arrow Functions

```
// Traditional Function
function (a){
  return a + 100;
}

// Arrow Function Break Down

// 1. Remove the word "function" and place arrow between the argument and
opening body bracket
(a) => {
  return a + 100;
}

// 2. Remove the body brackets and word "return" -- the return is implied.
(a) => a + 100;

// 3. Remove the argument parentheses
a => a + 100;
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Arrow Functions

```
// Traditional Function
function (a, b){
  return a + b + 100;
}

// Arrow Function
(a, b) => a + b + 100;

// Traditional Function (no arguments)
let a = 4;
let b = 2;
function (){
  return a + b + 100;
}

// Arrow Function (no arguments)
let a = 4;
let b = 2;
() => a + b + 100;
```

Arrow Functions

```
// Traditional Function
function (a, b){
  let chuck = 42;
  return a + b + chuck;
}

// Arrow Function
(a, b) => {
  let chuck = 42;
  return a + b + chuck;
}
```

Arrow Functions

```
// Traditional Function
function bob (a){
  return a + 100;
}

// Arrow Function
let bob = a => a + 100;
```

Template Strings

Template literals (Template Strings)

- Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them.
- They were called "template strings" in prior editions of the ES2015 specification.

Template literals (Template Strings)

```
`string text`

`string text line 1
 string text line 2`

`string text ${expression} string text`

tag`string text ${expression} string text`
```

JSON

What is JSON?

- JSON stands for JavaScript Object Notation
- JSON is a lightweight data interchange format
- JSON is language independent *
- JSON is "self-describing" and easy to understand

** The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.*

https://www.w3schools.com/js/js_json.asp

JSON

This JSON syntax defines an employees object: an array of 3 employee records (objects):

```
{
"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
]
}
```

JS Web APIs

- XMLHttpRequest (XHR)
- AJAX
- Fetch

JS Web APIs

XMLHttpRequest (XHR)

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function (){
 if (xhr.readyState === XMLHttpRequest.DONE {
   console.log(xhr.responseText);
 }
}
```

JS Web APIs

JQuery

```
$.ajax({
  url: ‘https://myapi.me/api/',
  dataType: 'json',
  success: function(data) {
    console.log(data);
  }
});
```

JS Web APIs

Fetch

```
fetch(URL).then(function(response) {
  // check for a good response
}).then(function(response){
  // do something with the response data
}).catch(function(error) {
  // handle the error
});
```

JS Web APIs

Fetch

```
fetch('http://example.com/movies.json')
  .then(response => response.json())
  .then(data => console.log(data));
```

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

Lab Activity on 24/04/2021

- Basic JavaScript Instructions
- Functions, Methods and Objects
- Decisions and Loops
- Document Object Model
- Events