Web Applications

CSE183

Fall 2020

JavaScript III

Baskin Engineering UC SANTA CRUZ

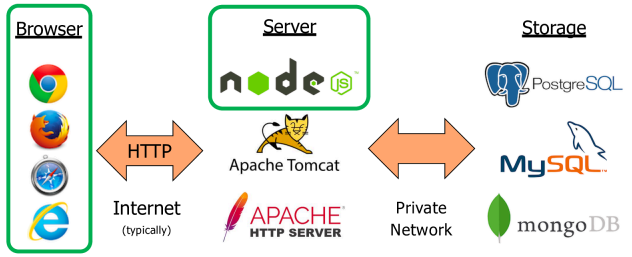
Notices

- **Assignment 3** due 23:59 **Thursday October 22**

Today's Lecture

- JavaScript Recap
- ECMAScript Review
- ECMAScript New Features
- Quiz 1 Solutions
- Questions

Full Stack Web Applications



JavaScript - Overview

- High-level
  - Heavily abstracted from hardware details
- Interpreted
  - Not compiled, executed by a platform-dependent run-time environment
- Dynamic
  - Undertakes compiler-like operations at runtime
- Untyped / Dynamically Typed
  - Any variable can hold any type of data
- Prototype-based
  - Object-oriented behaviors are re-used (inherited) from existing objects (prototypes)
- Has first-class functions
  - Functions are objects and can be manipulated as such
- Programming Models
  - Imperative, Functional, Object-oriented

ECMAScript

- Standardised Version of JavaScript
  - Updated every 12 months
  - Straightforward process for getting new features into the language
- Transpilers:
  - Write in up-to-date style but allow execution by any interpreter
    - i.e. modern JS in, old JS out
  - E.g. Babel https://babeljs.io/

    Given `var name = 'Bob';`
    then `console.log(`Hello ${name}`);`
    becomes `console.log('Hello ' + name);`
    or `console.log('Hello '.concat(name));`
- Front-end Frameworks embrace new language features
  - Reacj.js - Uses modern ECMAScript features
  - Angular - Uses TypeScript (JavaScript with static type checking)

Some ECMAScript Features

- Already encountered:
  - `let`, `const`, `class`
- Additional common features:
  - Default Parameters
  - Rest Parameter
  - Spread Operator
  - Destructuring Assignment
  - Template Literals
  - For Loops
  - Modules
  - Collection Abstractions
  - Asynchronous Operations

Default Parameters

- Named parameters initialized with default values
- Old way:

```
function oldWay(a,b) {
  a = a || 2;
  b = b || 'Hello';
  return b + a;
}
```

- New way:

```
function newWay(a = 2, b = 'Hello') {
  return b + a;
}
```

- Same output from:

```
console.log(oldWay());
console.log(newWay());
console.log(oldWay(undefined));
console.log(newWay(undefined));
```

Rest Parameter …

- Represent indefinite number of arguments as named array
- Old way:
  - No parameters listed, but available as `arguments` array

```
function oldWay() {
  var a = arguments[0];
  var b = arguments[1];
  return a + b;
}
```

- New way:
  - Additional parameters placed in named array

```
function newWay(a,...argv) {
  var b = argv[0];
  return a + b;
}
```

- Same output from:

```
console.log(oldWay(1,2));
console.log(newWay(1,2));
```


Spread Operator …

- Expand iterate-able expressions (arrays, strings) into argument lists or elements
- Given:

```
function sum(x, y, z) {
  return x + y + z;
}
const numbers = [1, 2, 3];
```

- Old way:

```
var s = sum.apply(null, numbers);
```

- New way:

```
let s = sum(...numbers);
```

- Same output from:

```
console.log(s);
```


Destructuring Assignment

- Unpack values from arrays, or properties from objects, into distinct named variables
- Examples:

```
var a = arr[0];
var b = arr[1];
var c = arr[2];
```

```
let [a,b,c] = arr;
```

```
var fname = obj.fname;
var lname = obj.lname;
bar title = obj.title
```

```
let {fname, lname, title} = obj;
```

```
function oldWay(person) {
  var fname = person.fname;
  var age = person.age;
  return fname + age;
}
```

```
function newWay({fname, age}) {
  return fname + age;
}
```

- Same output from:

```
console.log(oldWay({fname: 'Bob', age: 24}));
console.log(newWay({fname: 'Bob', age: 24}));
```


Template Literals

- String literals supporting embedded expressions
- Given:

```
var theirName = 'Bob';
var yourName = 'Patsy';
```

- Old way:

```
var str = 'Hello ' + theirName +', my name is ' + yourName + '.';
```

- New way:

```
let str = `Hello ${theirName}, my name is ${yourName}`;
```

- Same output from:

```
console.log(str);
```

For `of`

- Loop over iterate-able objects
  - E.g. String, Array, Map, Set, …
- Given:

```
var arr = [5,6,7];
var sum = 0;
```

- Old way:

```
for (var i = 0; i < arr.length; i++) {
  sum += arr[i];
}
```

- New way:

```
for (elem of arr) {
  sum += elem;
}
```

- Same output from:

```
console.log(sum);
```


For `in`

- Loop over string-keyed object properties
- Given:

```
var obj = { a: 5, b: 6, c: 7 };
var sum = 0;
```

- Old way:

```
var keys = Object.keys(obj);
for (var i = 0; i < keys.length; i++) {
  sum += obj[keys[i]];
}
```

- New way:

```
for (prop in obj) {
  sum += obj[prop];
}
```

- Same output from:

```
console.log(sum);
```


Modules

- Visibility of variables within source file
  - When "including" one file in another, need a way to restrict access to "private" variables in the included file
- Old way:

```
var exportedName = (function () {
  var x, y, x;   // 'Private' variables
  ...
  return {x: x, y: y};
})();
```

- New way:

```
var x, y, x;     // still private because they're not exported
...
var exportedName = {x: x, y: y};
export exportedName;
```

- Browser: `export` / `import`
- Node.js: `module.exports` / `require()`

Additional Features

- Collection Abstractions
  - `Set`, `Map`, `WeakSet`, `WeakMap`
  - Self study for details
- Asynchronous Operations
  - `async` / `await` and `Promise`
  - We'll cover later in the class

Quiz 1 - Solutions

Select the technology tiers that typically appear in a Full Stack Web Applications with a multitiered architecture.

Browser

Server

Data Storage

Experienced Web App developers consider using the HTML tags <b> and <i> to be bad practice. Explain why this is a valid opinion.

Tags request styling of text (bold/italic) yet it is considered *good* practice to enforce **separation of content** (HTML) **from style** (CSS).

Can be replaced by the semantic and tags

If a CSS Style Sheet does not explicitly set the width and height of an element, how does the browser decide what size to render the element?

Depends on the element type

Defaults to width:auto

Depends on the browser

JavaScript allows var statements to be written anywhere in a function, yet some coding standards require they be placed at the start of functions. Give a reasoned argument as to why this requirement is considered valid.

Var declarations are **hoisted** to the start of functions by the JavaScript runtime regardless of where they are declared in the code.

Insisting they be placed at the start of the function is reasonable as the code as written matches the code as executed and simultaneously avoids confusion and the potential for bugs.

If the current location of a browser tab is http://www.example.com/sales/index.html which contains the links:

Current Stock

Texas Rollup

What are the full URL equivalents of these links?

http://www.example.com/stock/current/index.html (absolute)

http://www.example.com/sales/rollup/texas/index.html (relative)

What would this snippet of JavaScript display on the console? Explain your answer using line numbers for reference.

```
5  let i = 4;
6  (function(i){
7    i /= 6.2;
8    console.log(i);
9  })(12.4);
10 console.log(i);
```

Console output:

2
4

The **closure** of the function includes the argument i but not the global variable i

Line 8 displays the result of dividing the argument i by 6.2

Line 10 displays the value of the global i

What port would the server this URL is hosted by have to be listening on for a browser to connect to it successfully?

https://www.example.com/sales/index.html

443

Is this snippet of JavaScript valid, and if so, what would it display on the console? Explain your answers.

```
11  let arr = ['one','two',,,5,6];
12  console.log([arr[5],arr[3],arr[6]]);
```

Console output: [6, undefined, undefined]

JavaScript arrays are **polymorphic** and **sparse**; hence the code is valid even though the array has missing elements and elements of two different types, namely number and string.

As JavaScript arrays are zero indexed, arr[5] is the number 6, the 6th element in the array.

There is no element at index 3 but JavaScript arrays are sparse so that element has a value of "undefined".

There is no element at index 6 as the length of the array is 6, but again, JavaScript arrays are sparse, so any non-negative index is valid but in this case the value is also "undefined".

Select the statements that are true about fixed and absolute CSS positions.

fixed positions the element with respect to the page

absolute positions the element with respect to its closest ancestor

fixed positioning means the element is fixed in place as scrolling occurs

Describe how invalid characters are dealt with in query parameters and give an example.

Query parameters must be **URL Encoded**. i.e. each invalid character is replaced by %xx where xx is the hexadecimal value of the character.

Example:

https://www.demo.com/customer?name=M&S Stores

Becomes:

https://www.demo.com/customer?name=M%26S%20Stores

Upcoming Lectures

- Wednesday: Document Object Model
- Friday: User Interfaces

- Sometime this week: Assignment 2 Feedback

Tasks

- **Assignment 3** due 23:59 **Thursday October 22**