Admin stuff

- A1 will be available tonight. Deadline is May 29st 5pm
  - No push access after this date

- Submit your assignment on **Gradescope.ca** by linking it to your Github

CSCC09 Programming on the Web
Interactive frontends with JavaScript

Cho Yin Yong

UNIVERSITY OF
TORONTO
SCARBOROUGH

Last Lecture

- Create a professional page layout with HTML/CSS with 12-column layout
- Learn how to make websites responsive

Javascript

- Dynamic and weakly typed (typescript kind of fixes that)
- Interpreted, Just-in-time (JIT) compiled programming language (more on that in CSCC24)
- **First class functions (*functions are variables*)**

Use the latest version of javascript - es2020

- The last big change to Javascript was ES6
- https://caniuse.com/?search=es6
- https://caniuse.com/?search=es5

Always learn the latest - people always make tools to compile es6 code to es5.

Basic Syntax

```
// this is a comment
/*
 * this is a
 * multiline comment
 */
console.log('Hello world')
console.warn('Houston, we have a problem')
```

Useful Primitives (immutable)

```
const a = ‘hello’  // string

const b = 10 // number

let c = true // boolean

const d = undefined

const e = null
```

Arrays (mutable)

```
const myArray = [1, 2, 3, 4]

myArray[0]  // 1
```

Objects (mutable)

```
const myObject = {

  firstName: “Cho Yin”,

}

myObject.lastName = “Yong”
```

If-else statements

```
if (myCondition && myOtherCondition) {
  // do something
} else if (myCondition || anotherCondition) {
  // do something else
} else {
  // do something else
}
```

Functions

```
function myFunctionName(myParam) {


}
```

Array manipulation - useful for lab and assignment!

```
const myArray = [1, 2, 3]

myArray.push(4)

myArray.splice(1, 1)

console.log(myArray)
```

Anonymous/Arrow Functions

```
const myFunction = function() {}

const myOtherFunction = (myParam) => {}

const result = myFunction()
```

const and let

```
var x = 0 // old syntax, don't use this

const a = 0

a = 1 // ERROR

let b = 0

b = 1 // OK
```

When to use `const` vs when to use `let`?

Functions can be variables

```
const a = [1, 2, 3, 4]
a.forEach((element) => {
  console.log(element);
})
const b = a.map((e) => {
  return e + 1;
})
```

Functions can be variables

```
function forEach(array, fn) {
  for (let i = 0; i < array.length; i++) {
    // the function is called for each item in array
    fn(array[i]);
  }
}

forEach([1, 2, 3, 4], (e) => console.log(e))
```

Functions can be variables

```
function map(array, fn) {
  const output = [];
  for (let i = 0; i < array.length; i++) {
    output.push(fn(array[i]));
  }
  return output;
}
```

Tricks

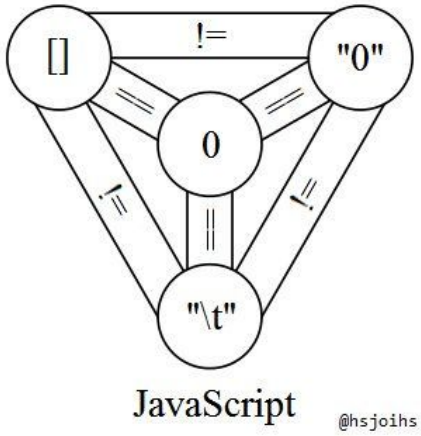
- Use optional chaining operator on if statements to simplify:
  `if (item?.attr) vs if (item && item.attr)`
- To convert a string into an int, you can do `+'5'`
- When checking for null or undefined, use Nullish coalescing operator (??)

Don’ts

Double equals (==)

Use triple equals almost always (===)

The Javascript holy trinity



== will convert variable values to the same type before comparison.

More JavaScript help

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

Github Copilot Chat

Javascript in the Browser

JavaScript in the browser

- Event-driven programming model
  - Execution flow determined on events that happen on the browser
- Updates a page via the “Document Object Model”
- No access to other programs
- Cannot execute arbitrary OS commands
- Cannot access other tabs
- ~~No access to filesystem (only upload forms)~~ new!

Including javascript

Inline

```
<button onclick="console.log('hello world')">
```

Embedded: specified in the header (<head>)

```
<script type="text/javascript">
   console.log('hello world')
</style>
```

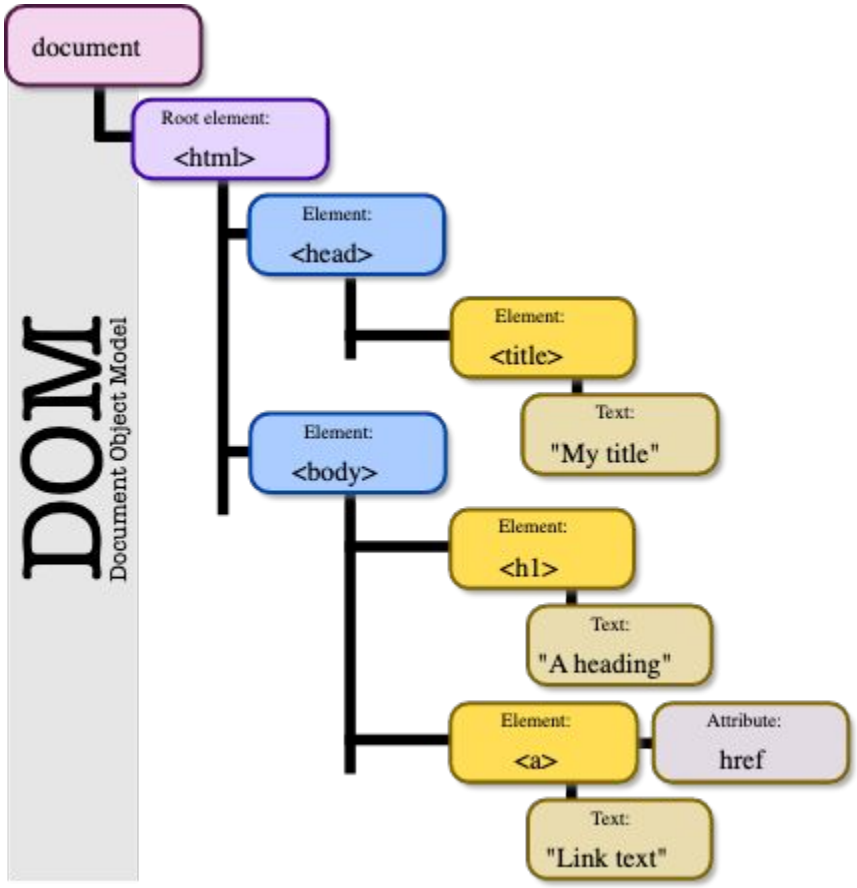
Separate file: at the end of the body

```
<script src="js/main.js"/>
```

Rule specific for C09 vanilla JS. Each "framework" has its own rules - should respect that.

Document Object Model

- **Tree** representation of an HTML document, accessible via Javascript
- Add, change, remove any HTML elements and attributes
- Change any CSS styles



DOM Node accessors - selecting DOM element

```
document.getElementById("id")
document.getElementByTagName("p");
document.getElementByClassName("class");
document.querySelector("#username");  # preferred
document.querySelectorAll(".dropdown .dropdown-item"); # preferred


document.querySelector(".dropdown .dropdown-item");  # problem
```

DOM Methods (for element x) - Updating

x.innerHTML	The HTML inside of x
x.content	The content inside of x
x.attributes	The attributes of x
x.style	Css of x
x.append(element), x.prepend(element)	prepend/append element to x
x.appendChild	Insert a child node to x
x.removeChild	Remove a child node from x
document.createElement('div')	Create a div

We will not use this

Web Events

- Events that are built into the browser
- Anything that you can think of
- https://developer.mozilla.org/en-US/docs/Web/Events

```
document.querySelector("#dontClickMe").addEventListener("click",  function() {
  alert("You clicked me!");
});
```

Global variables!

window	The current browser window
history	Browser back and forward URLs
localStorage/sessionStorage	A simple browser persistent storage API
document	To access the DOM

setTimeout, setInterval

- setTimeout: Run something after x milliseconds.
- setInterval: Run something every x milliseconds.

```
setTimeout(function() {}, 1000)
```

Good browser Javascript practices

Strict mode: `"use strict"`

- Force the browser to validate Javascript against the standard
- Dynamically raises errors (or warnings) in the console when the code is not compliant with the standard
- Example...

Scoping Problem

In the browser, all Javascript files share the same execution environment i.e they share the same scope.

- Naming conflicts
- Unintended side consequences to event listeners

```
<script src=”js/one.js”/>

<script src=”js/two.js”/>

<script src=”js/three.js”/>
```

Scoping Problem

```
function getChirps() {
  return []
}
// somewhere later in the file
getChirps()
```

one.js

```
function getChirps() {
  return [{"content": "hello world"}]
}
```

two.js

```
getChirps()
```

three.js

What gets called?

Closure: encapsulate and export the namespace

```
const apiService = (function() {

  "use strict";

  const module = {};

  module.getChirps = function() {}


  function privateFunction() {}

  return module;

}());
```

Example usage:

```
apiService.getChirps()
```

```
(function() {
  function getChirps() {}
})();
```

Encapsulate and add some private functions

```
const $ = (function() {

  "use strict";

  let module = {}

  function myFunction() {}

  return module;

}());
```

Example usage:

```
$.myFunction() <- this will throw an error
```

DOM takes time to load

If the DOM has not loaded, you are attaching event listeners to nothing...

```
window.onload = function() {}
```

```
window.addEventListener('DOMContentLoaded', (event) => {});
```

Both can be used, but what is the difference between the above?

Quick Chirper example

Create “chirps” and append them to the DOM.

Structuring browser JavaScript code

index1.js

Imperative programming.

When an event happens (on submission of HTML form), perform some updates directly to the DOM.

The DOM forms a **source of truth** of the information. We direct manipulate the UI.

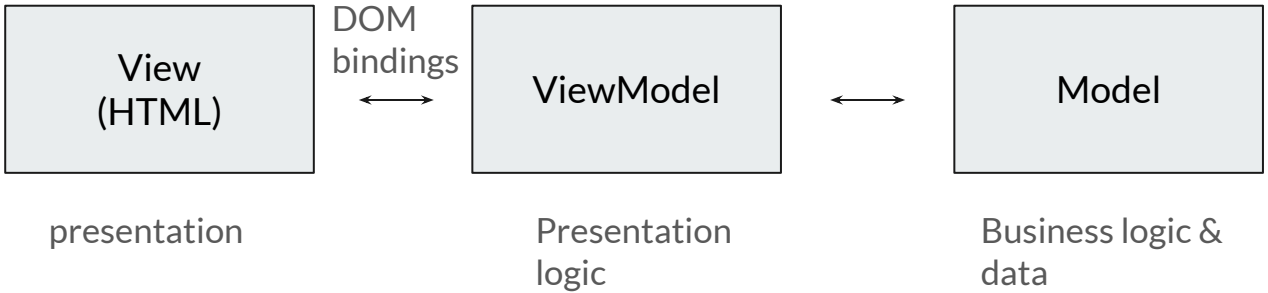
This is a common pattern 10 years ago with jQuery.

index2.js

Taking out business logic to another file to keep separation of concerns. (i.e. separate manipulation of UI with keeping track of items)

Model-View-Viewmodel (MVVM)

- Model: Frontend API Service (api-service.js)
- ViewModel: Frontend Controller (index2.js)
- View: View (index.html)



Key Concepts

- The View does not know the existence of the Model
- The API Service does not know the existence of anything else
- The Controller is the piece that links both elements together. Controller never stores data.

index3.js

Manage application state using “global variables”. Use global variables as the source of truth to re-render UI always.

Idea: if we keep global variables up to date, the UI will reflect the variables always.

```
const state = {
   chirps: [],
 };
```

```
state.chirps.find((c) => c.id === chirp.id).content = formProps.chirp;
updateChirpList();
```

```
ChirpService.addChirp(formProps.chirp);
state.chirps = ChirpService.getChirps();
updateChirpList();
```

```
state.chirps.forEach(function (chirp) {
  const newChirp = createChirpComponent(chirp);
  document.querySelector("#chirpsList").prepend(newChirp);
});
```

index3.js

Use of event listeners (observer pattern)

```
const onEditButtonClickedListeners = [];
```

```
newChirp.querySelector(".edit").addEventListener("click", function () {
     // show edit form of the current ChirpComponent
     onEditButtonClickedListeners.push((chirpId)=> {
       if (chirpId !== chirp.id) {
         // hide edit form of the current ChirpComponent
       }
     });
     onEditButtonClickedListeners.forEach((listener)=> listener(chirp.id));
});
```

index4.js / meact.js

An attempt to use vanilla JS to replicate basic building blocks of React.

- useState()
- useEffect()

What to look out for:

- Implementation of Observer pattern

Chrome Devtools Demos

- console
- debugger

JSON

JSON - Javascript Object Notation

Serializes a Javascript object into a string format

Deserializes into a Javascript object

Breaking down “JSON”

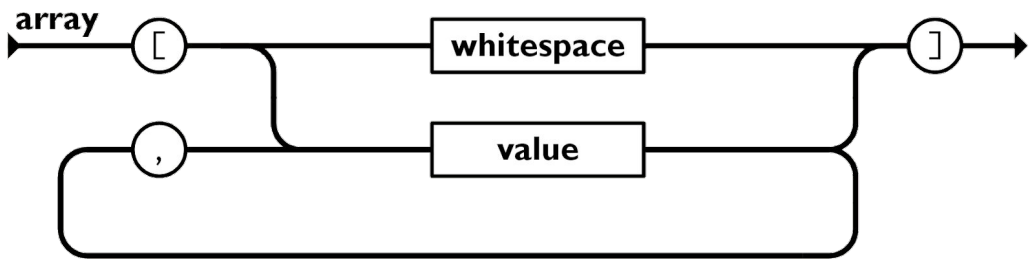
A “Valid JSON value” is a recursive structure:

- string (eg. `“cscc09”`, `“is”`, `“awesome”`)
- number (eg. `1`, `2`, `42`, `69`)
- boolean (true, false)
- array of “Valid JSON values”
- object of string to “Valid JSON values”

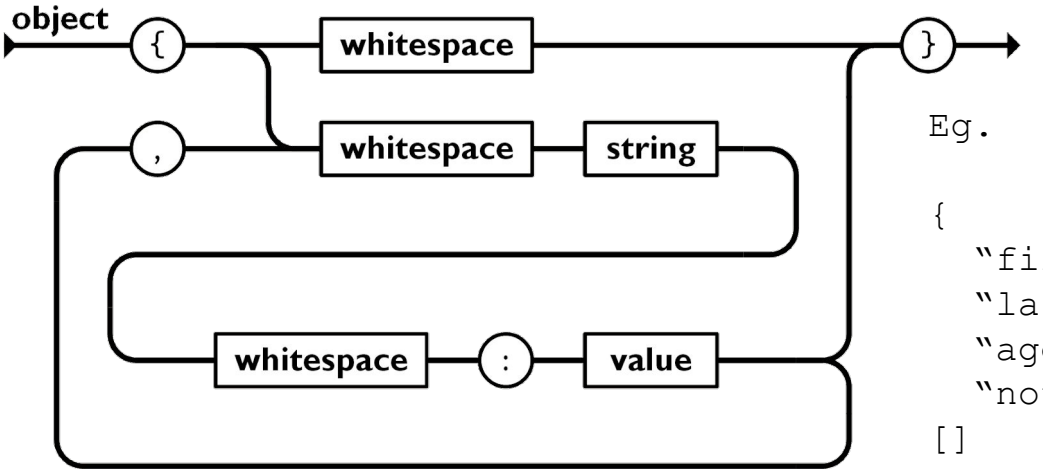
Array

An indexed list.

Eg. `[1, 2, 3, true, false]`
Eg. `["cscc09", 42]`



Object (key value pairs)



Eg.

```
{
  “first_name”: “Donald”,
  “last_name”: “Trump”,
  “age”: 77,
  “notably_positive_features”:
[]
}
```

Why do we care?

`localStorage` can only handle strings and numbers, not objects or arrays.

Serialize: `JSON.stringify(obj)`

Deserialize: `JSON.parse(jsonStr)`