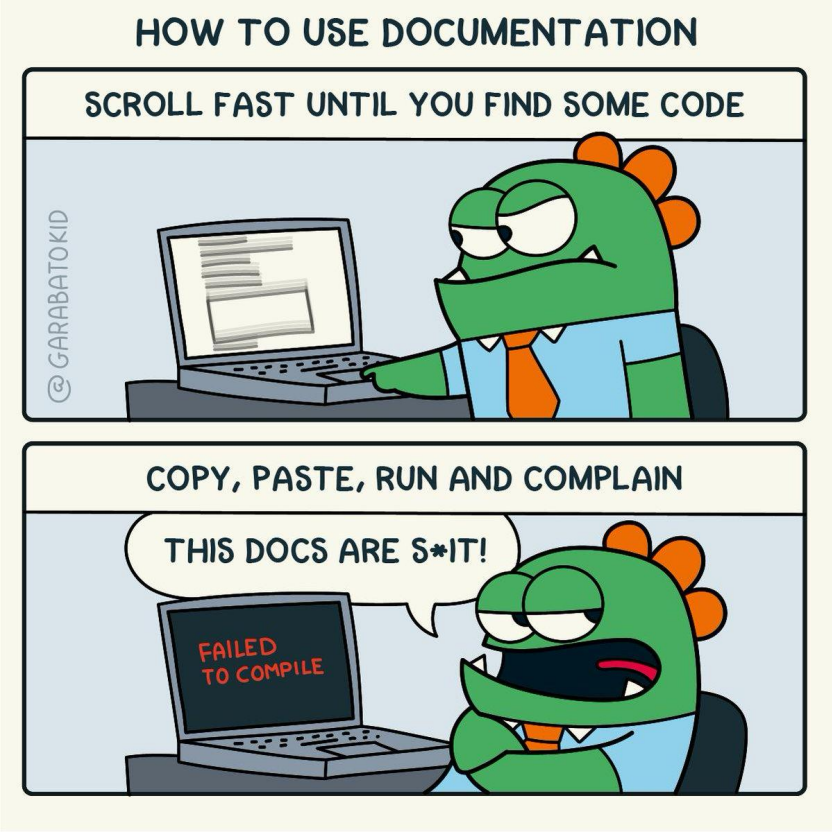
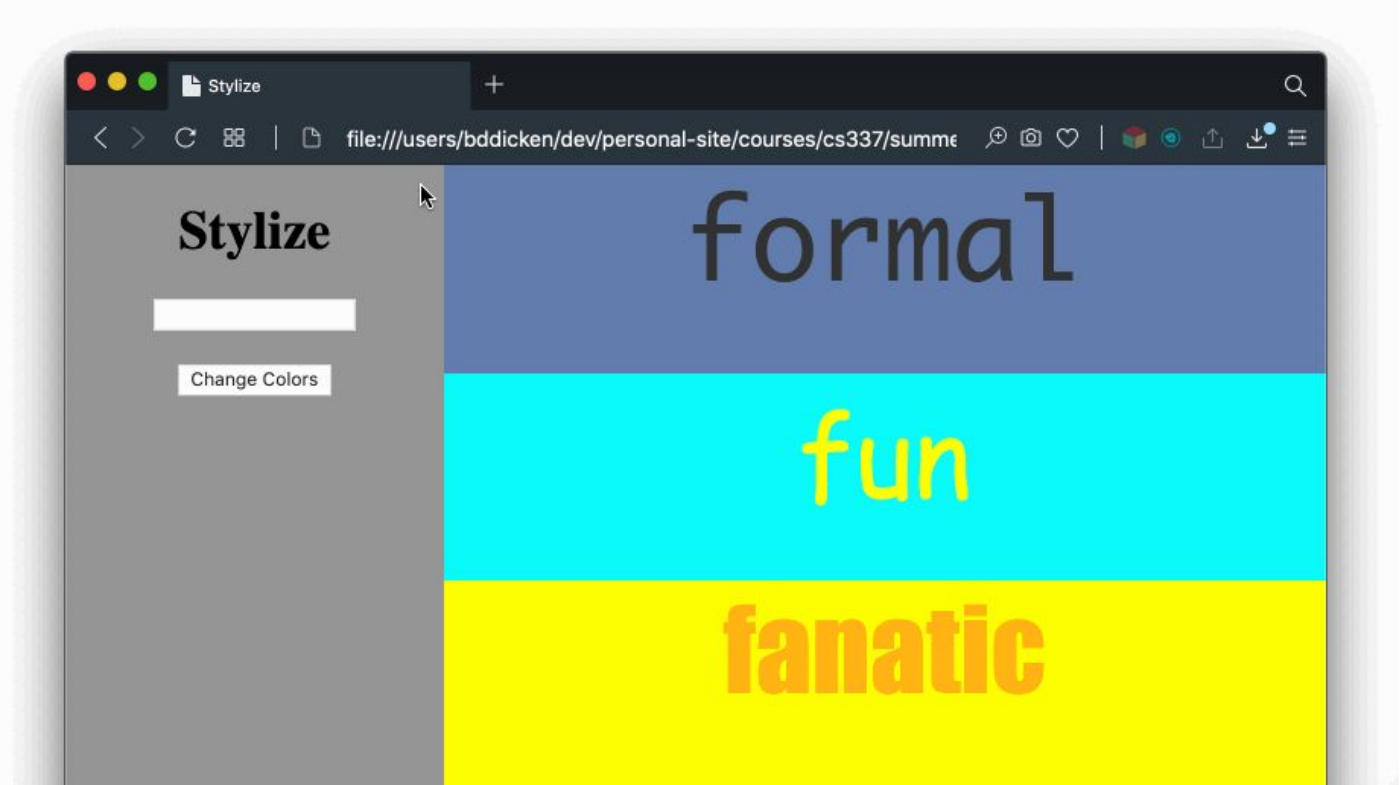


CSc 337

DOM, Events, Lambda + Higher-Order Functions

Benjamin Dicken

Lets build this simple app



What will we need?

- **`onclick`** and **`oninput`** event
- Layout with CSS (absolute?)
- Use CSS to customize fonts, colors, etc
- Randomization (colors)

Javascript Function

- **Lambda Function**
- **Higher Order Function**
- What are these?

Fancy Names, Simple Meaning

- **Lambda Function** - A function with no name (identifier)

- **Higher Order Function** - A function that either *takes* a function as a parameter, and/or *returns* a function

Fancy Names, Simple Meaning

- **Lambda Function** - A function with no name (identifier)

  . . . why is *this* useful ?

- **Higher Order Function** - A function that either *takes* a function as a parameter, and/or *returns* a function

  . . . why is *this* useful ?

Passing Around Functions

- In Js, one can save functions to variables, pass them to other functions, return functions
- Why and how?

```
function takeActionOnElements(action, theData) {
  for (var i = 0; i < theData.length; i++) {
    action(theData[i]);
  }
}

function printValue(value) { console.log(value); }

function printDouble(value) { console.log(2 * value); }

nums = [1, 5, 2, 9]
takeActionOnElements(printValue, nums);
takeActionOnElements(printDouble, nums);
```

```
function takeActionOnElements(action, theData) {
  for (var i = 0; i < theData.length; i++) {
    action(theData[i]);
  }
}

nums = [1, 5, 2, 9]

takeActionOnElements((v) => { console.log(v); }, nums);
takeActionOnElements((v) => { console.log(2*v); }, nums);
```

```
function showAlert(a) {
  alert(a + " X");
}

function justPrint(a) {
  console.log(a + " Y");
}

function pickOne() {
  var seconds = Math.floor(new Date().getTime());
  if (seconds % 2 == 0) { return showAlert; }
  else { return justPrint; }
}

let supriseMe = pickOne();

supriseMe(1000);
```

What does this do?

```
function completeInSequence(funcs) {
  for (i in funcs) {
    task = funcs[i];
    task(5);
  }
}

completeInSequence([
  (x) => {console.log("hello there." + x)},
  (y) => {
    var x = [1,2,3,4];
    for (i in x) {
      console.log(x[i] + '' + y);
    }
  }
]);
```

What does this do?

Practical Uses in Web Programming

- The **`window.setInterval`** and **`window.setTimeout`** functions are higher-order!
  - Can pass in a named function or lambda function.
- Functions are often used as event handlers.
  - “When event **X** happens, call function **Y** to handle it!

```
element.onclick = myFunction;
// or
element.onclick = () => { window.alert("click!"); }
```

Array Functions

- Map - Create new array from existing array with a transformation applied to each element
- Reduce - Produce a single object / element from the elements of a array based on a rule given by a function
- Filter - Produce a new array from an existing array, filtering out some based on a rule
- Some - Check if an array contains one or more elements that match a rule given by a function

Practical Uses in Web Programming

- Also will be *very* useful when we get to server-side programming
- Much of the code you write for the server side will involve higher-order functions (callbacks and promises)

```
nums = [7, 9, 1, 2, 1, 8, 4, 2]

maximum = nums[0];
for (i in nums)
  maximum = maximum > nums[i] ? maximum : nums[i];

minimum = nums[0];
for (i in nums)
  minimum = minimum < nums[i] ? minimum : nums[i];

sum = 0;
for (i in nums)
  sum += nums[i];

mul = 1;
for (i in nums)
  mul *= nums[i];

last = nums[nums.length-1];

console.log(maximum, minimum, sum, mul, last);
```

What does this do?

```
nums = [7, 9, 1, 2, 1, 8, 4, 2]

maximum = 0;
for (i in nums)
  maximum = maximum > nums[i] ? maximum : nums[i];

minimum = 0;
for (i in nums)
  minimum = minimum < nums[i] ? minimum : nums[i];

sum = 0;
for (i in nums)
  sum += nums[i];

mul = 1;
for (i in nums)
  mul *= nums[i];

last = nums[nums.length-1];

console.log(maximum, minimum, sum, mul, last);
```

ICA

```
nums = [7, 9, 1, 2, 1, 8, 4, 2]

maximum = 0;
for (i in nums)
  maximum = maximum > nums[i] ? maximum : nums[i];

minimum = 0;
for (i in nums)
  minimum = minimum < nums[i] ? minimum : nums[i];

sum = 0;
for (i in nums)
  sum += nums[i];

mul = 1;
for (i in nums)
  mul *= nums[i];

last = nums[nums.length-1];

console.log(maximum, minimum, sum, mul, last);
```

Rewrite the code to reduce duplication. Use lambda / higher-order functions

```
nums = [7, 9, 1, 2, 1, 8, 4, 2]

function action(a, all) {
  var soFar = 0;
  for (i in all)
    soFar = a(soFar, all[i]);
  return soFar;
}

maximum = action((s,v) => s > v ? s : v, nums);
minimum = action((s,v) => s < v ? s : v, nums);
sum     = action((s,v) => s + v, nums);
mul     = action((s,v) => s == 0 ? v : s * v, nums);
last    = action((s,v) => v, nums);

console.log(maximum, minimum, sum, mul, last);
```