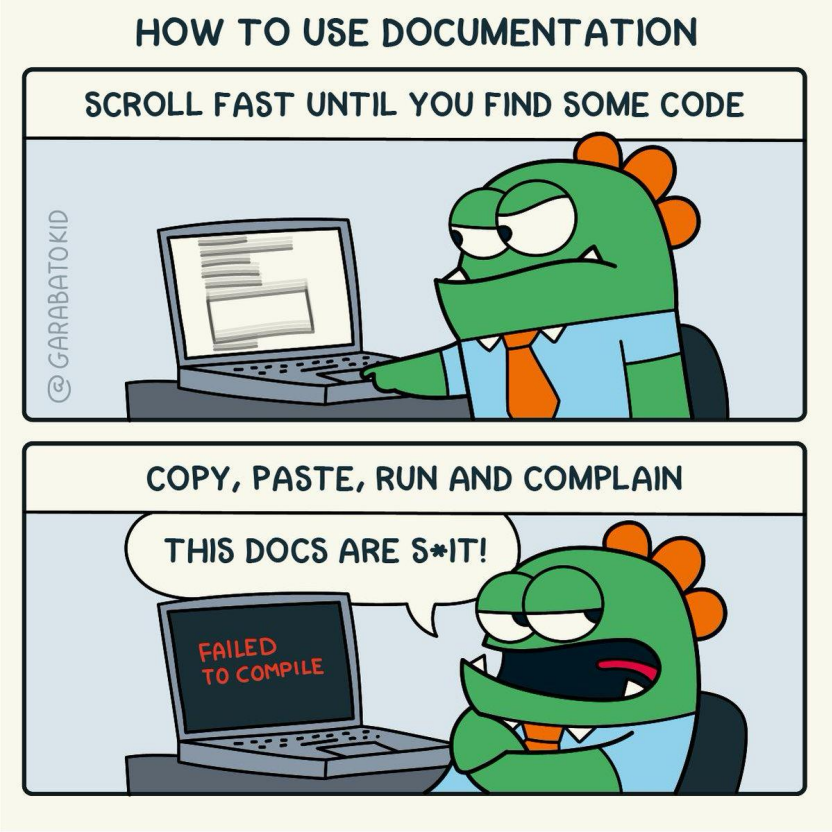


CSc 337 Javascript

Benjamin Dicken

Javascript Function

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

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 ?

ICA

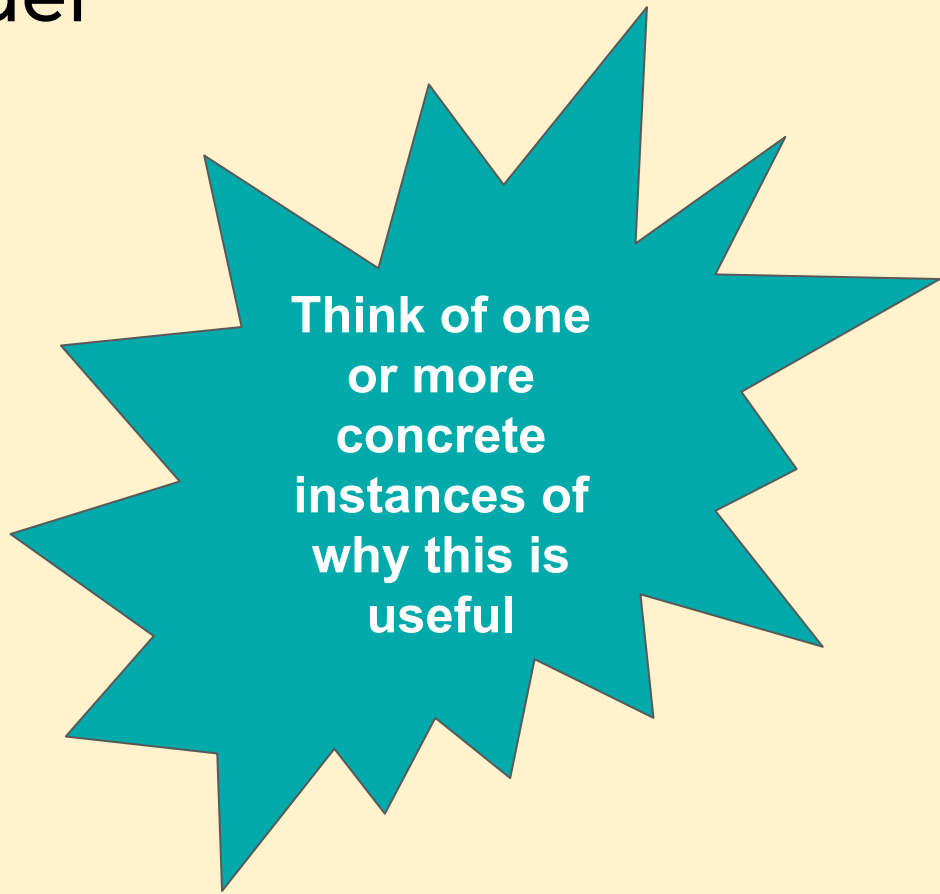
Lambda and Higher-order

- **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 action(a, theData) {
  for (var i = 0; i < theData.length; i++) {
    a(theData[i]);
  }
}

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

nums = [1, 5, 2, 1, 9]

action(printValue, nums);
```

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

nums = [1, 5, 2, 1, 9]

action(function(v) { console.log(v); }, nums);
```

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

nums = [1, 5, 2, 1, 9]

action((v) => { console.log(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?

```
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);
```

ICA

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

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

What will it print?