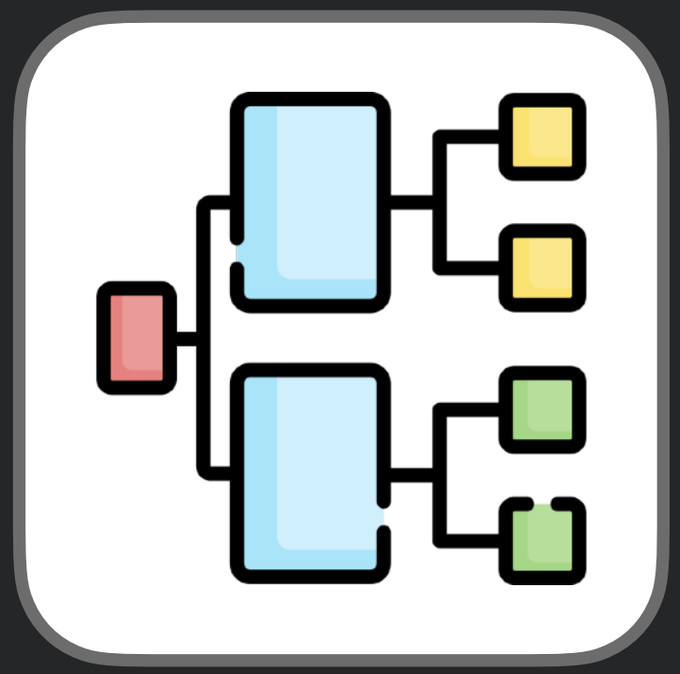
SWE 432 -Web Application Development

Fall 2021

George Mason University

Dr. Kevin Moran

Week 2: Organizing Code & Tools and Testing



Administrivia

- *HW Assignment 1* - Due Before Class Next Week (Sept. 7th)
- *Hands-On Session Schedule* - Now available on Course Webpage

HW Assignment 1

Overview

In this homework assignment, you will download a JSON dataset from the web and write a simple data analytics package in JavaScript to answer eight questions about your dataset.

Assignment Instructions

Step 1: Download a JSON dataset from a website

In this step, you will collect a JSON dataset containing at least 100 rows (i.e., entries) from a website. You are free to choose whatever data source you'd like. Note that some, but not all, data sources may first require you to obtain an API key by creating an account with the data provider. You should not choose any API that requires you to authenticate using oAuth, as we have not yet covered oAuth.

You may, but are not required to, choose a data set from one of the following:

- Open Data DC
- Public APIs (many of these are APIs for performing computation and are NOT datasets, check carefully)
- DC Metro

After choosing an API, you should collect a dataset containing at least 100 records in JSON format. It's fine if your dataset contains more than 100 records. If you dataset is very large (> 10,000 records), you may wish to choose a subset of the dataset to enable you to test your code more quickly in the following steps.

HW Assignment 1

Step 2: List eight (8) questions you will answer about your dataset

Now that you've found a dataset, what insights can you extract from this dataset? In this step, you will write a list of eight (8) questions about your dataset. Each question should describe a statistic to compute from your dataset.

For example, if your dataset is city demographic data, you might have the following questions:

1. What is the average age of residents?
2. What is the average year over year growth rate?
3. What is the fastest growing city?
4. What is the median population density?
5. What is the city with the highest population density?
6. Which is the average age of small cities with less than 100,000 people?
7. What is the city with the oldest population?
8. What is the city with the least amount of new home construction per capita?

In order to more easily satisfy the requirements of step 3, you are encouraged to have a diversity of question types.

HW Assignment 1

Step 3: Implement a JavaScript program to answer your questions

In this step, you will now create a JavaScript program to compute the answers to your eight questions using your JSON dataset. For each of the eight questions, your program should output to the JavaScript console (1) the question and (2) the answer. For example, if your question was “What is the fastest growing city?”, your program should write to the console: “What is the fastest growing city? Springfield”

Your program must use all of the following JavaScript features:

- **Variable declarations**
  - Let statement
  - Const statement
- **Functions**
  - Arrow function
  - Default values
  - Array.map()
- **Loops**
  - For of statement

HW Assignment 1

Your program must use all of the following JavaScript features:

- **Variable declarations**
  - Let statement
  - Const statement
- **Functions**
  - Arrow function
  - Default values
  - Array.map()
- **Loops**
  - For of statement
  - For in statement
- **Collections**
  - Instance of a Map or Set collection (only 1 is required)
- **Strings**
  - Template literal
- **Classes**
  - Class declaration
  - Constructor
  - Using an instance variable with this

HW Assignment 1

Submission instructions

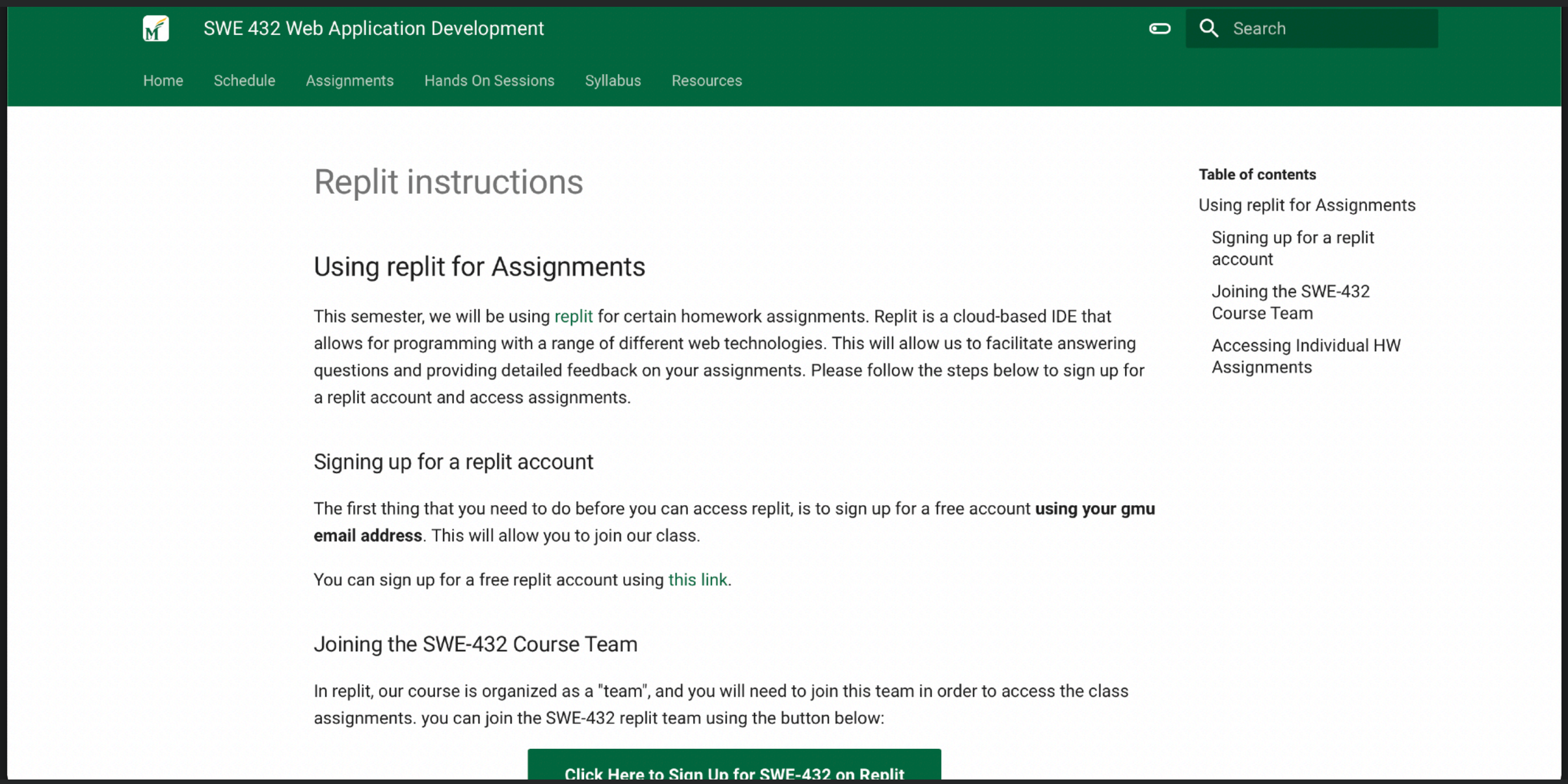
- Submit your HW through **replit**. Please follow these instructions for signing up for the replit account and accessing HW1. You should be able to complete this project using only the replit web interface. However, if you would like to work locally on your machine, you can code using your preferred environment and then upload the final `.js` file through replit.

The HW assignment submission should consist of one `.js` file containing:

1. A comment with your full name
2. A comment containing a URL where your JSON dataset from step (1) can be found
3. Your JSON dataset from step (1) (or a subset of the dataset that is at least 100 records)
4. Your program from step (3)

Click Here to Access the Assignment via Replit

Signing Up For Replit

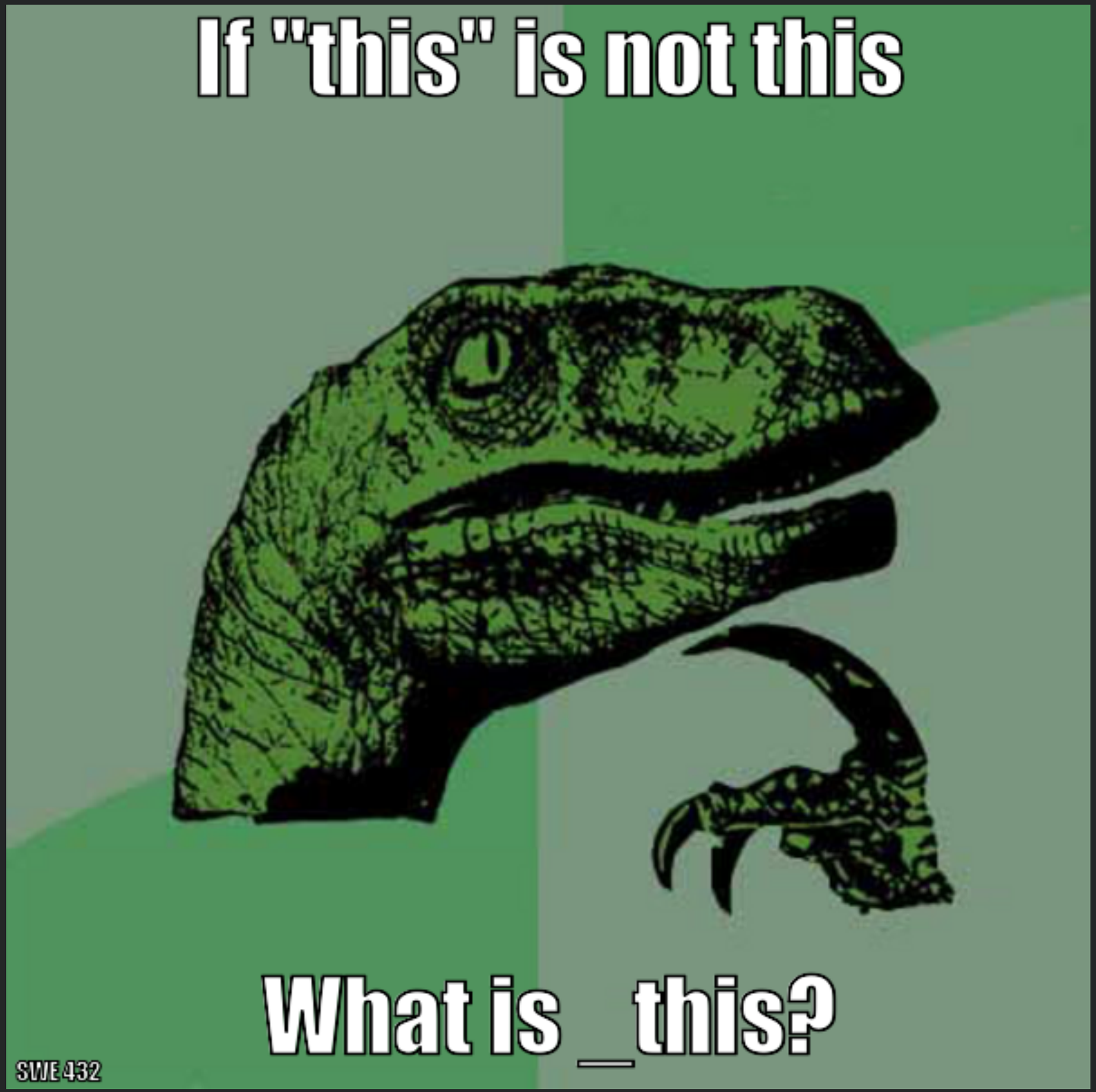


Class Overview

Class Overview

- *Part 1 - Organizing Code in Web Apps:* How can we build comprehensible and maintainable web apps?
- *10 minute Break*
- *Part 2 - Javascript Tools and Testing:* Exploring Node and Testing Strategies
- *Part 3 - In-Class Activity:* Closures and Testing with JEST

Organizing Code in Web Apps


If "this" is not this
What is _this?
SWE 432

Today

- Some basics on how and why to organize code (SWE!)
- Closures
- Classes
- Modules

For further reading:

http://stackoverflow.com/questions/111102/how-do-javascript-closures-work

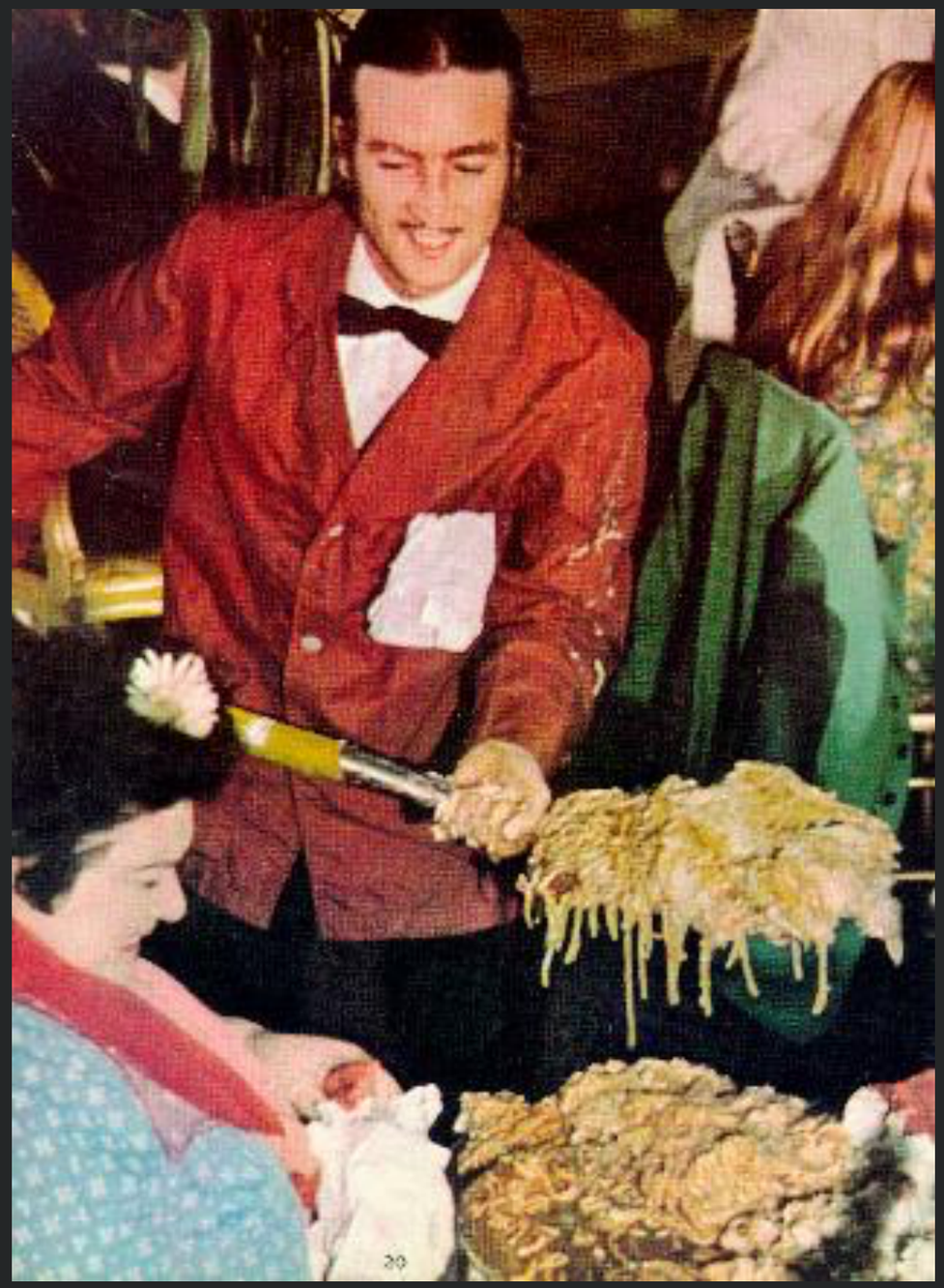
Running Javascript

- More on this next time
- Some options for now
  - a pastebin (e.g., JSFiddle)
  - an IDE (e.g, VSCode, Webstorm)
    - Webstorm is free for students:
      - https://www.jetbrains.com/student/

History + Motivation

"Back in my day before ES6 we didn't have your fancy modules"

Spaghetti Code



```
window.onload = function () {
    eqCtl = document.getElementById('eq');
    currNumberCtl = document.getElementById('currNumber');
};

var eqCtl,
    currNumberCtl,
    operator,
    operatorSet = false,
    equalsPressed = false,
    lastNumber = null;

function add(x,y) {
    return x + y;
}

function subtract(x, y) {
    return x - y;
}

function multiply(x, y) {
    return x * y;
}

function divide(x, y) {
    if (y == 0) {
        alert("Can't divide by 0");
        return 0;
    }
    return x / y;
}

function setVal(val) {
    currNumberCtl.innerHTML = val;
}

function setEquation(val) {
    eqCtl.innerHTML = val;
}

function clearNumbers() {
    lastNumber = null;
    equalsPressed = operatorSet = false;
    setVal('0');
    setEquation('');
}
```

```
function setOperator(newOperator) {
    if (newOperator == '=') {
        equalsPressed = true;
        calculate();
        setEquation('');
        return;
    }

   if (!equalsPressed) calculate();
    equalsPressed = false;
    operator = newOperator;
    operatorSet = true;
    lastNumber = parseFloat(currNumberCtl.innerHTML);
    var eqText = (eqCtl.innerHTML == '') ?
        lastNumber + ' ' + operator + ' ' :
        eqCtl.innerHTML + ' ' + operator + ' ';
    setEquation(eqText);
}

function numberClick(e) {
    var button = (e.target) ? e.target : e.srcElement;
    if (operatorSet == true || currNumberCtl.innerHTML == '0') {
        setVal('');
        operatorSet = false;
    }
    setVal(currNumberCtl.innerHTML + button.innerHTML);
    setEquation(eqCtl.innerHTML + button.innerHTML);
}

function calculate() {
    if (!operator || lastNumber == null) return;
    var currNumber = parseFloat(currNumberCtl.innerHTML),
        newVal = 0;
  switch (operator) {
        case '+':
            newVal = add(lastNumber, currNumber);
            break;
        case '-':
            newVal = subtract(lastNumber, currNumber);
            break;
        case '*':
            newVal = multiply(lastNumber, currNumber);
            break;
        case '/':
            newVal = divide(lastNumber, currNumber);
            break;
    }
    setVal(newVal);
    lastNumber = newVal;
}
```

```
window.onload = function () {
    eqCtl = document.getElementById('e
    currNumberCtl = document.getElemer
};

var eqCtl,
    currNumberCtl,
    operator,
    operatorSet = false,
    equalsPressed = false,
    lastNumber = null;

function add(x,y) {
    return x + y;
}

function subtract(x, y) {
    return x - y;
}

function multiply(x, y) {
    return x * y;
}

function divide(x, y) {
    if (y == 0) {
        alert("Can't divide by 0");
        return 0;
    }
    return x / y;
}

function setVal(val) {
    currNumberCtl.innerHTML = val;
}

function setEquation(val) {
    eqCtl.innerHTML = val;
}

function clearNumbers() {
    lastNumber = null;
    equalsPressed = operatorSet = fals
    setVal('0');
    setEquation('');
}

function setOperator(newOperator) {
    if (newOperator == '=') {
        equalsPressed = true;
        calculate();
        setEquation('');
        return;
    }

    if (!equalsPressed) calculate();
    equalsPressed = false;
    operator = newOperator;
    operatorSet = true;
    lastNumber = parseFloat(currNumberCtl.innerHTML);
    var eqText = (eqCtl.innerHTML == ''
        lastNumber + ' ' + operator +
        eqCtl.innerHTML + ' ' + operat
    setEquation(eqText);
}

function numberClick(e) {
    var button = (e.target) ? e.target
    if (operatorSet == true || currNum
        setVal('');
        operatorSet = false;
    }
    setVal(currNumberCtl.innerHTML + b
    setEquation(eqCtl.innerHTML + butt
}

function calculate() {
    if (!operator || lastNumber == nul
    var currNumber = parseFloat(currNu
        newVal = 0;
    switch (operator) {
        case '+':
            newVal = add(lastNumber, c
            break;
        case '-':
            newVal = subtract(lastNumb
            break;
        case '*':
            newVal = multiply(lastNumb
            break;
        case '/':
            newVal = divide(lastNumber
            break;
    }
    setVal(newVal);
    lastNumber = newVal;
}
```

```
function setOperator(newOperator) {
    if (newOperator == '=') {
        equalsPressed = true;
        calculate();
        setEquation('');
        return;
    }

   if (!equalsPressed) calculate();
    equalsPressed = false;
    operator = newOperator;
    operatorSet = true;
    lastNumber = parseFloat(currNumberCtl.innerHTML);
    var eqText = (eqCtl.innerHTML == '' ?
        lastNumber + ' ' + operator + ' ' :
        eqCtl.innerHTML + ' ' + operator + ' ';
    setEquation(eqText);
}

function numberClick(e) {
    var button = (e.target) ? e.target : e.srcElement;
    if (operatorSet == true || currNumberCtl.innerHTML == '
        setVal('');
        operatorSet = false;
    }
    setVal(currNumberCtl.innerHTML + button.innerHTML);
    setEquation(eqCtl.innerHTML + button.innerHTML);
}

function calculate() {
    if (!operator || lastNumber == null) return;
    var currNumber = parseFloat(currNumberCtl.innerHTML),
        newVal = 0;
    switch (operator) {
        case '+':
            newVal = add(lastNumber, currNumber);
            break;
        case '-':
            newVal = subtract(lastNumber, currNumber);
            break;
        case '*':
            newVal = multiply(lastNumber, currNumber);
            break;
        case '/':
            newVal = divide(lastNumber, currNumber);
            break;
    }
    setVal(newVal);
```

Bad Code "Smells"

Bad Code "Smells"

- Tons of not-very related functions in the same file
- No/bad comments
- Hard to understand
- Lots of nested functions

Bad Code "Smells"

- Tons of not-very related functions in the same file
- No/bad comments
- Hard to understand
- Lots of nested functions

```
fs.readdir(source, function (err, files) {
          if (err) {
          console.log('Error finding files: ' + err)
          } else {
          files.forEach(function (filename, fileIndex) {
                    console.log(filename)
                    gm(source + filename).size(function (err, values) {
                                        if (err) {
                                        console.log('Error identifying file size: ' + err)
                                        } else {
                                        console.log(filename + ' : ' + values)
                                        aspect = (values.width / values.height)
                                        widths.forEach(function (width, widthIndex) {
                                                  height = Math.round(width / aspect)
                                                  console.log('resizing ' + filename + 'to ' + height +
                                                  this.resize(width, height).write(dest + 'w' + width +
                                                                      if (err) console.log
                                                                      })
```

Code Smell Research

When and Why Your Code Starts to Smell Bad
(and Whether the Smells Go Away)

Michele Tufano[1], Fabio Palomba[2], Gabriele Bavota[3]
Rocco Oliveto[4], Massimiliano Di Penta[5], Andrea De Lucia[2], Denys Poshyvanyk[1]
[1]The College of William and Mary, Williamsburg, VA, USA [2]University of Salerno, Fisciano (SA), Italy,
[3]Università della Svizzera italiana (USI), Switzerland, [4]University of Molise, Pesche (IS), Italy,
[5]University of Sannio, Benevento (BN), Italy
mtufano@email.wm.edu, fpalomba@unisa.it, gabriele.bavota@usi.ch
rocco.oliveto@unimol.it, dipenta@unisannio.it, adelucia@unisa.it, denys@cs.wm.edu

Abstract—Technical debt is a metaphor introduced by Cunningham to indicate "not quite right code which we postpone making it right". One noticeable symptom of technical debt is represented by code smells, defined as symptoms of poor design and implementation choices. Previous studies showed the negative impact of code smells on the comprehensibility and maintainability of code. While the repercussions of smells on code quality have been empirically assessed, there is still only anecdotal evidence on *when* and *why* bad smells are introduced, what is their *survivability*, and *how* they are *removed* by developers. To empirically corroborate such anecdotal evidence, we conducted a large empirical study over the change history of 200 open source projects. This study required the development of a strategy to identify smell-introducing commits, the mining of over half a million of commits, and the manual analysis of 10K of them. Our findings mostly contradict common wisdom, showing that most of the smell instances are introduced when an artifact is created and not as a result of its evolution. At the same time, 80% of smells survive in the system. Also, only 9% of the 20% of removed instances are removed as a direct consequence of refactoring operations.

Index Terms—Code Smells, Empirical Study, Mining Software Repositories

When and Why Your Code *Sta*

Michele Tufano*, Fabio Palomba†, Gabriele Bavota‡, [illegible]
Massimiliano Di Penta¶, Andrea De Lucia†, Denys P[illegible]
*The College of William and Mary, Williamsburg, VA, USA - §University of [illegible]
‡Free University of Bozen-Bolzano, Italy - †University of S[illegible]
¶University of Sannio, Benevento, Italy

***Abstract*—In past and recent years, the issues related to managing technical debt received significant attention by researchers from both industry and academia. There are several factors that contribute to technical debt. One of these is represented by code bad smells, *i.e.*, symptoms of poor design and implementation choices. While the repercussions of smells on code quality have been empirically assessed, there is still only anecdotal evidence on *when* and *why* bad smells are introduced. To fill this gap, we conducted a large empirical study over the change history of 200 open source projects from different software ecosystems and investigated when bad smells are introduced by developers, and the circumstances and reasons behind their introduction. Our study required the development of a strategy to identify smell-introducing commits, the mining of over 0.5M commits, and the manual analysis of 9,164 of them (*i.e.*, those identified as *smell-introducing*). Our findings mostly contradict common wisdom stating that smells are being introduced during evolutionary tasks. In the light of our results, we also call for the need to develop a new generation of recommendation systems aimed at properly planning smell refactoring activities.**

I. Introduction

Technical debt is a metaphor introduced by Cunningham to indicate "*not quite right code which we postpone making it right*" [18]. The metaphor explains well the trade-off between delivering the most appropriate but still imperfect product, in the shortest time possible [12]. While the repercussions of "[illegible] quality have been empirically [illegible] of empirical [illegible] focus [illegible]

metric values [33], [34], static code [illegible] of software changes [36]. While these [illegible] accurate and complete identification [illegible] smells, most of them work by "taking [illegible] system or by looking at recent changes, [illegible] snapshot-based recommendation to the develop[illegible] do not consider the circumstances that could hav[illegible] smell introduction. In order to better support dev[illegible] planning actions to improve design and source code q[illegible] is imperative to have a contextualized understanding of th[illegible] cumstances under which particular smells occur. However[illegible] the best of our knowledge, there is no comprehensive empirica[illegible] investigation into *when* and *why* code smells are introduced in software projects. Common wisdom suggests that maintenance activities and pressure to deliver [illegible] prioritizing time-to-market over code [illegible] causes of such smells. Generally, [illegible] has always been consid[illegible] "software aging" [illegible] Broadly [illegible]

Transactions on Software Engineering

removed [14]. This represents an obstacle [illegible] tive and efficient management of technic[illegible] understanding the typical life-cycle of c[illegible] the actions undertaken by developers t[illegible] is of paramount importance in the conc[illegible] tools for developers' support. [illegible] understanding of the ph[illegible] recommenders a[illegible]

Design Goals

- Within a component
  - Cohesive
  - Complete
  - Convenient
  - Clear
  - Consistent
- Between components
  - Low coupling

Cohesion and Coupling

- Cohesion is a property or characteristic of an individual unit
- Coupling is a property of a collection of units
- High cohesion GOOD, high coupling BAD

- Design for change:
  - Reduce interdependency (coupling): You don't want a change in one unit to ripple throughout your system
  - Group functionality (cohesion): Easier to find things, intuitive metaphor aids understanding

Design for Reuse

- Why?
  - Don’t duplicate existing functionality
  - Avoid repeated effort
- How?
  - Make it easy to extract a single component:
    - Low ***coupling*** between components
    - Have high ***cohesion*** within a component

Design for Change

- Why?
  - Want to be able to add new features
  - Want to be able to easily ***maintain*** existing software
    - Adapt to new environments
    - Support new configurations
- How?
  - Low ***coupling*** - prevents unintended side effects
  - High ***cohesion*** - easier to find things

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components		

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	Classes

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	Classes
Entire libraries		

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	Classes
Entire libraries	Packages	

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	Classes
Entire libraries	Packages	Modules

Classes

- ES6 introduces the `class` keyword
- Mainly just syntax - still not like Java Classes

Classes

- ES6 introduces the `class` keyword
- Mainly just syntax - still not like Java Classes

Old

```
function Faculty(first, last, teaches, office)
{
  this.firstName = first;
  this.lastName = last;
  this.teaches = teaches;
  this.office = office;
  this.fullName = function(){
    return this.firstName + " " + this.lastName;
  }
}
var prof = new Faculty("Thomas", "LaToza", "SWE432", "ENGR 4431");
```

Classes

- ES6 introduces the `class` keyword
- Mainly just syntax - still not like Java Classes

Old

```
function Faculty(first, last, teaches, office)
{
  this.firstName = first;
  this.lastName = last;
  this.teaches = teaches;
  this.office = office;
  this.fullName = function(){
    return this.firstName + " " + this.lastName;
  }
}
var prof = new Faculty("Thomas", "LaToza", "SWE432", "ENGR 4431");
```

New

```
class Faculty {
    constructor(first, last, teaches, office)
    {
        this.firstName = first;
        this.lastName = last;
        this.teaches = teaches;
        this.office = office;
    }
    fullname() {
        return this.firstName + " " + this.lastName;
    }
}
var prof = new Faculty("Thomas", "LaToza", "SWE432", "ENGR 4431");
```

Classes - Extends

`extends` allows an object created by a class to be linked to a "`super`" class. Can (but don't have to) add parent constructor.

```
class Faculty {
    constructor(first, last, teaches, office)
    {
        this.firstName = first;
        this.lastName = last;
        this.teaches = teaches;
        this.office = office;
    }
    fullname() {
        return this.firstName + " " + this.lastName;
    }
}
```

```
class CoolFaculty extends Faculty {
    fullname() {
        return "The really cool " + super.fullname();
    }
}
```

Classes - `static`

`static` declarations in a `class` work like in Java

```
class Faculty {
    constructor(first, last, teaches, office)
    {
        this.firstName = first;
        this.lastName = last;
        this.teaches = teaches;
        this.office = office;
    }
    fullname() {
        return this.firstName + " " + this.lastName;
    }
        static formatFacultyName(f) {
        return f.firstName + " " + f.lastName;
    }
}
```

Modules (ES6)

- With ES6, there is finally language support for modules
- Module must be defined in its own JS file
- Modules **`export`** declarations
  - Publicly exposes functions as part of module interface
- Code **`imports`** modules (and optionally only parts of them)
  - Specify module by path to the file

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Or name all of the exports at once

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};

export {getFaculty as aliasForFunction, someVar};
```

Or name all of the exports at once

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};

export {getFaculty as aliasForFunction, someVar};
```

Or name all of the exports at once

Can rename exports too

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};

export {getFaculty as aliasForFunction, someVar};


export default function getFaculty(i){...
```

Or name all of the exports at once

Can rename exports too

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Or name all of the exports at once

Can rename exports too

```
export {getFaculty as aliasForFunction, someVar};
```

```
export default function getFaculty(i){...
```

Default export

Modules (ES6) - Import Syntax

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

- Import default export, binding to specified name

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

- Import default export, binding to specified name

```
import theThing from "myModule";
theThing()... -> calls getFaculty()
```

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

- Import default export, binding to specified name

```
import theThing from "myModule";
theThing()... -> calls getFaculty()
```

- Import all exports, binding to specified name

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

- Import default export, binding to specified name

```
import theThing from "myModule";
theThing()... -> calls getFaculty()
```

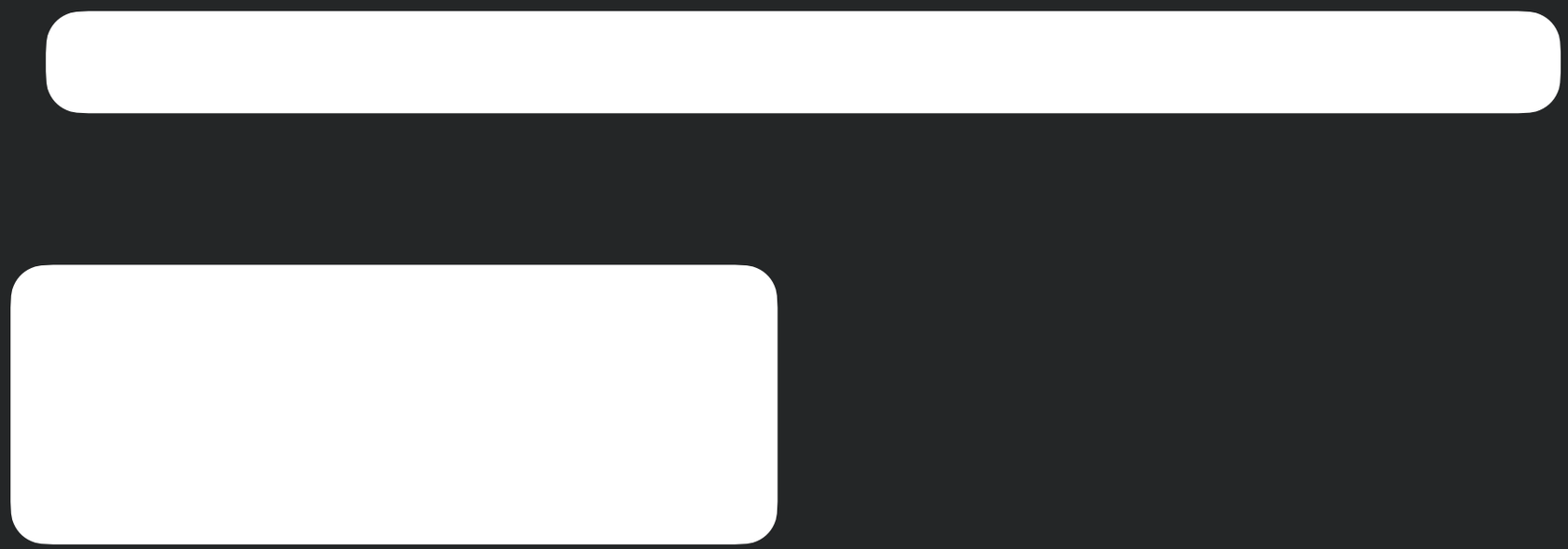
- Import all exports, binding to specified name

```
import * as facModule from "myModule";
facModule.getFaculty()...
```

Patterns for using/creating libraries

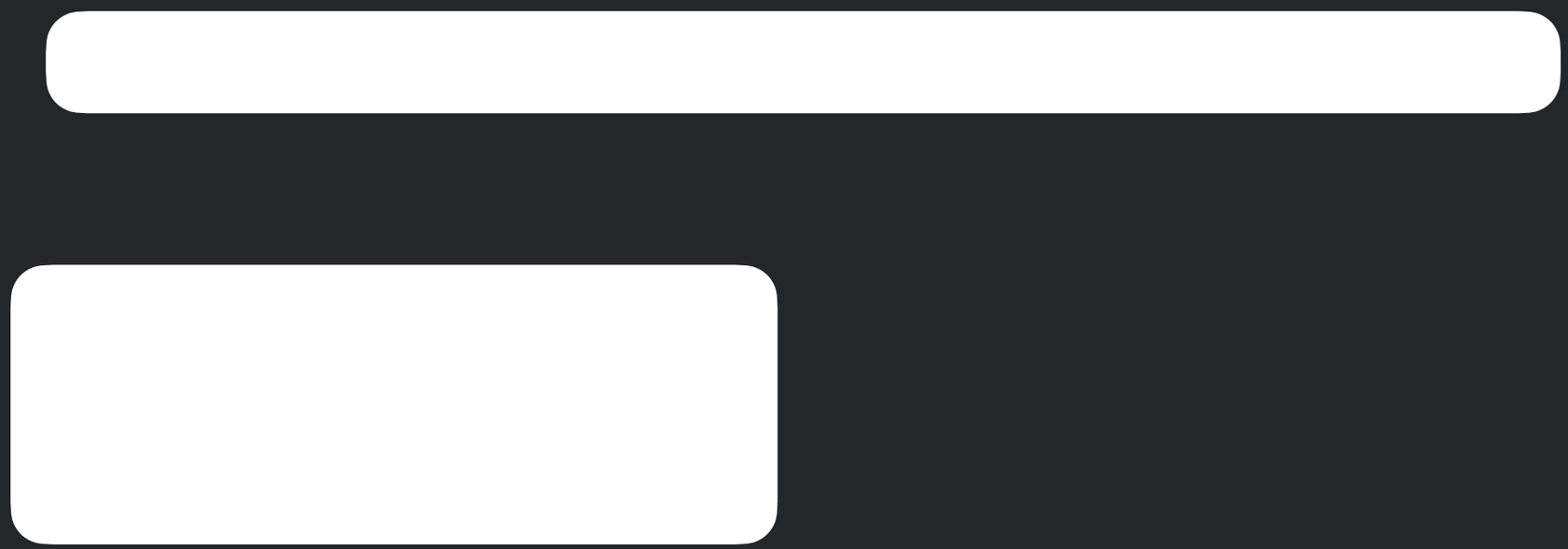
- Try to reuse as much as possible!
- Name your module in all lower case, with hyphens
- Include:
  - README.md
  - keywords, description, and license in package.json (from npm init)
- Strive for high cohesion, low coupling
  - Separate models from views
  - How much code to put in a single module?
- Cascades (see jQuery)

Cascade Pattern



Cascade Pattern

- aka “chaining”

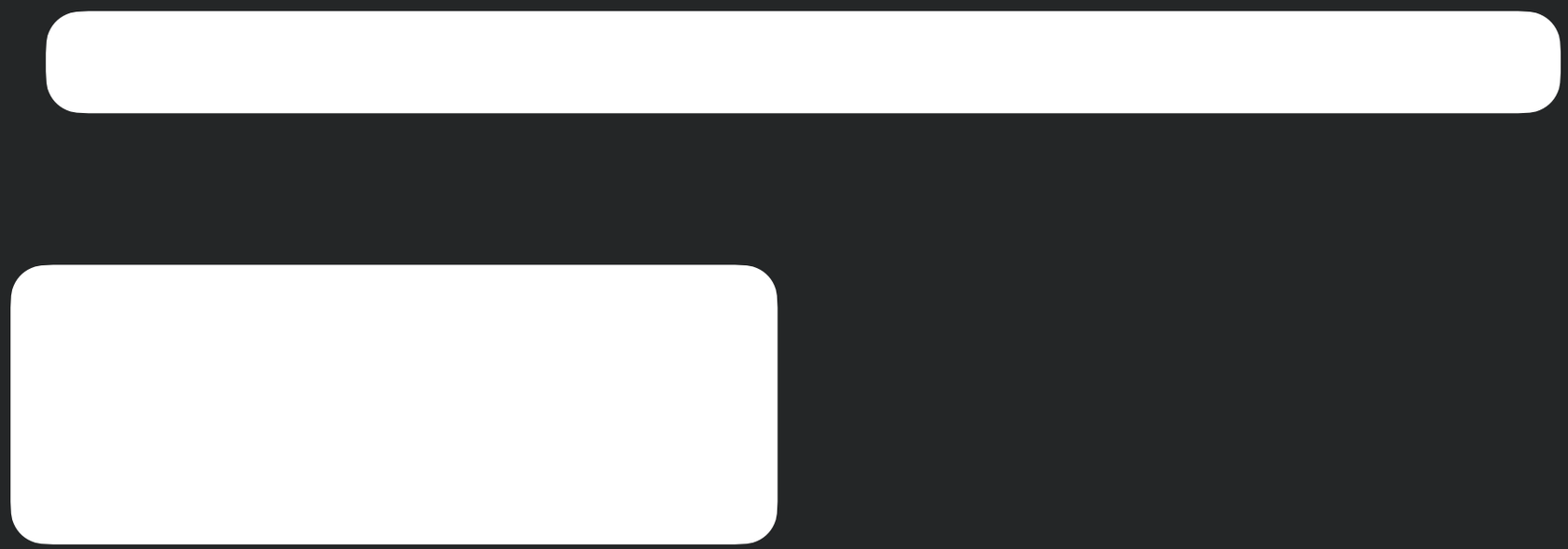


Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object

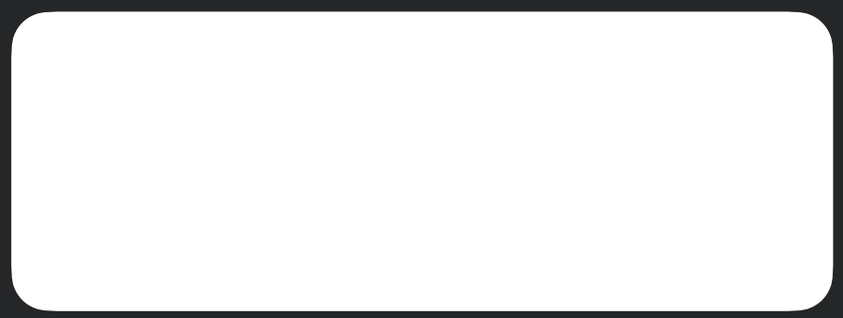
Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily



Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence



Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

- Example (jQuery):

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

- Example (jQuery):

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
    - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

- Example (jQuery):

```
$(“#wrapper")
```

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

- Example (jQuery):

```
$(“#wrapper")
        .fadeOut()
```

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

- Example (jQuery):

```
$(“#wrapper")
        .fadeOut()
        .html(“Welcome")
```

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
    - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

- Example (jQuery):

```
$(“#wrapper")
        .fadeOut()
        .html(“Welcome")
        .fadeIn();
```

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
- Closure is that function and a ***stack frame*** that is allocated when a function starts executing and ***not freed*** after the function returns

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Function called: stack frame created

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

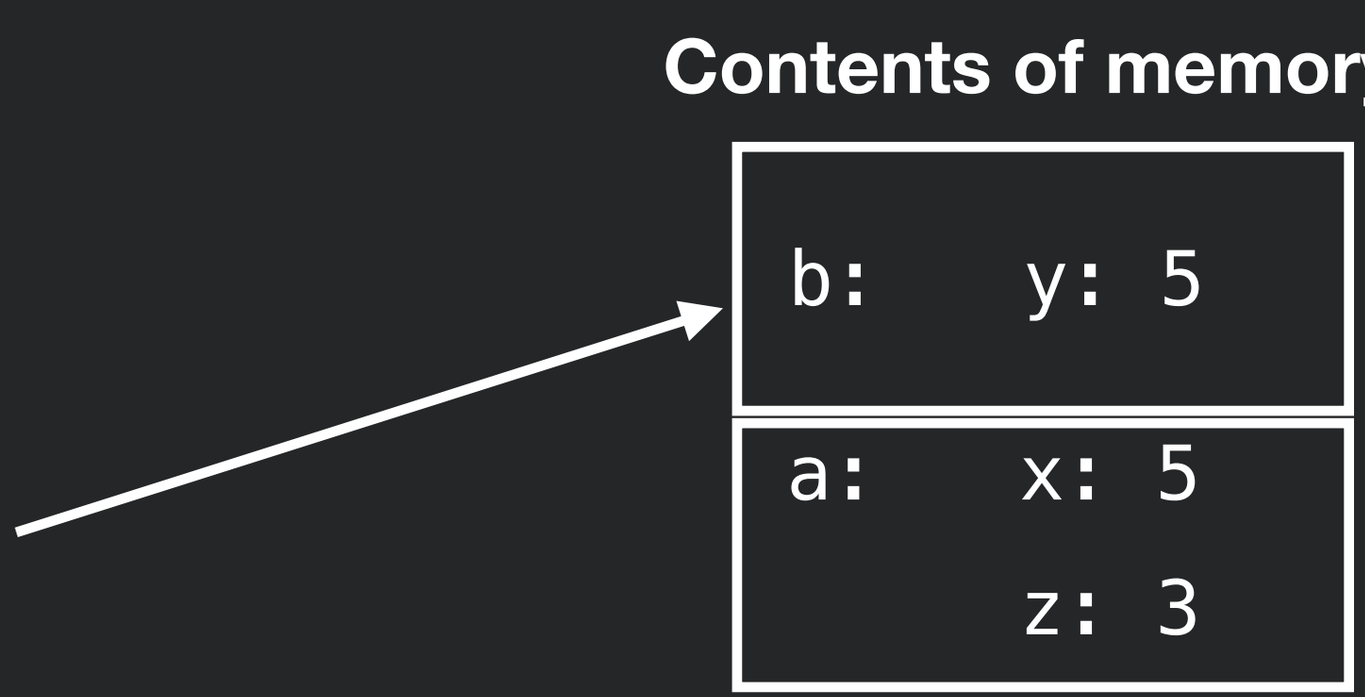
Contents of memory:

Stack frame

Function called: stack frame created

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

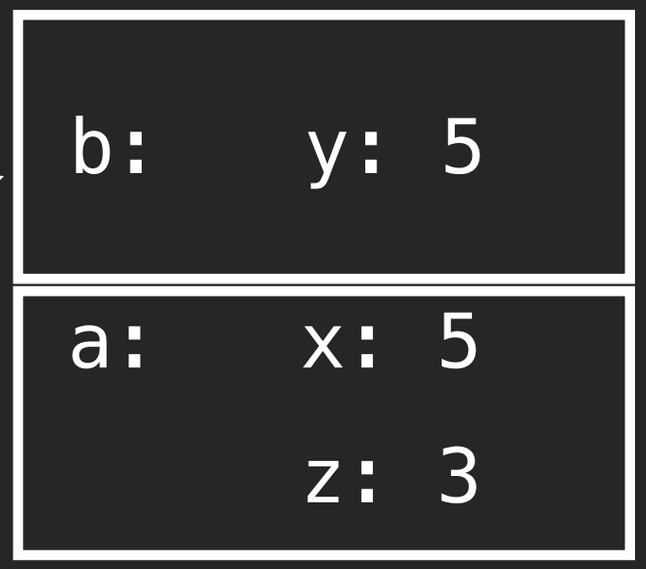


Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y) {
    console.log(y);
}
a();
```

Contents of memory:



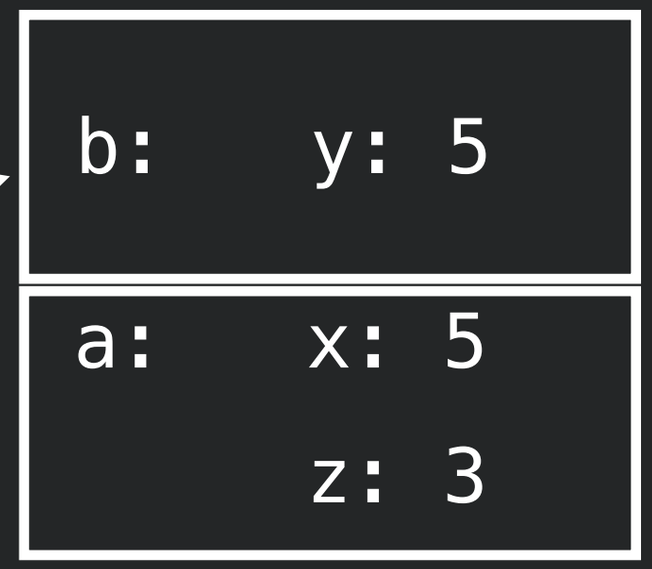
Stack frame

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y) {
    console.log(y);
}
a();
```

Contents of memory:

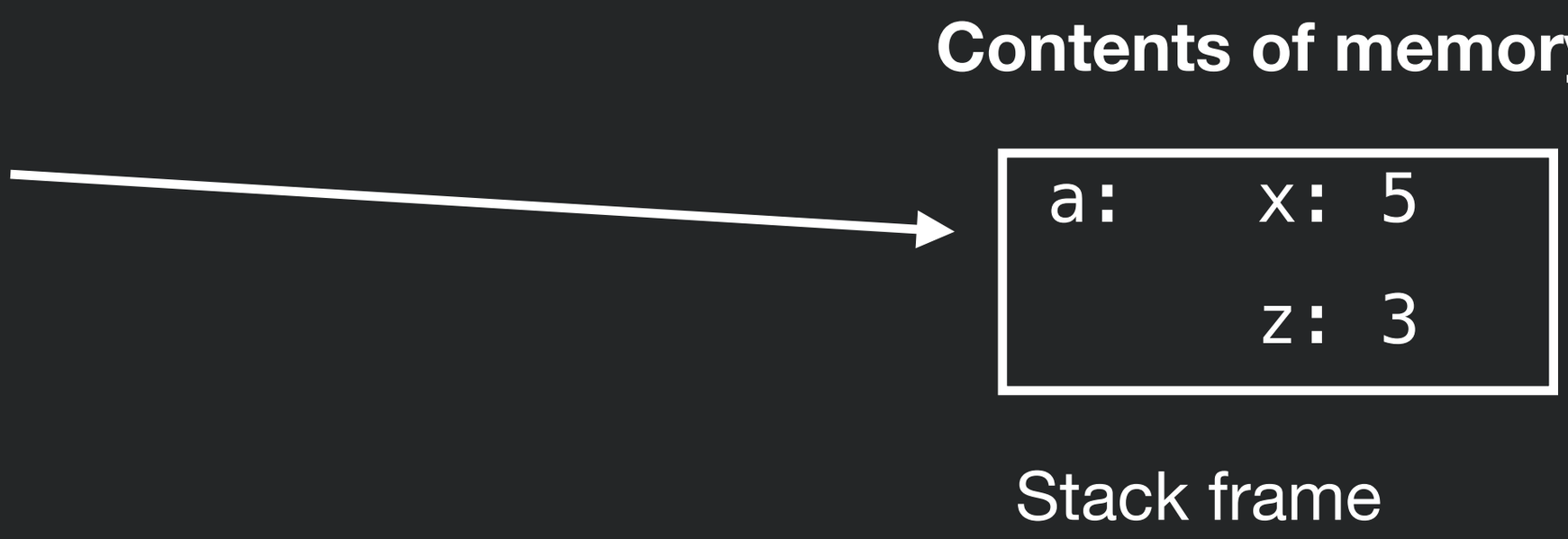


Stack frame

Function called: stack frame created

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code



Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Contents of memory:

Stack frame

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Contents of memory:

Stack frame

Function called: stack frame created

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

```
var x = 1;
function f() {
    var y = 2;
    return function() {
            console.log(x + y);
            y++;
    };
}
var g = f();
g();             // 1+2 is 3
g();             // 1+3 is 4
```

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

```
var x = 1;
function f() {
     var y = 2;
     return function() {
               console.log(x + y);
               y++;
     };
}
var g = f();
g();                // 1+2 is 3
g();                // 1+3 is 4
```

This function attaches itself to x and y so that it can continue to access them.

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();          // 1+2 is 3
g();          // 1+3 is 4
```

This function attaches itself to x and y so that it can continue to access them.

Closures

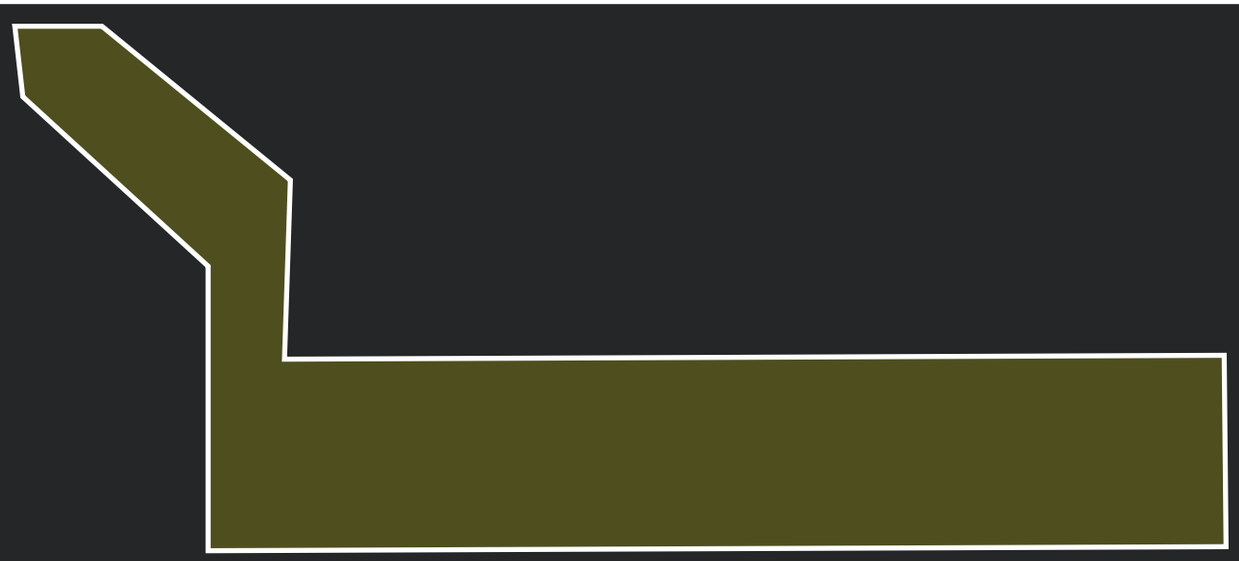
- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

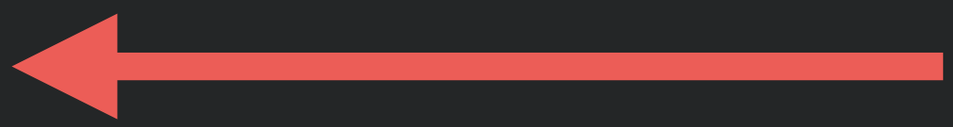
```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();          // 1+2 is 3
g();          // 1+3 is 4
```

This function attaches itself to x and y so that it can continue to access them.

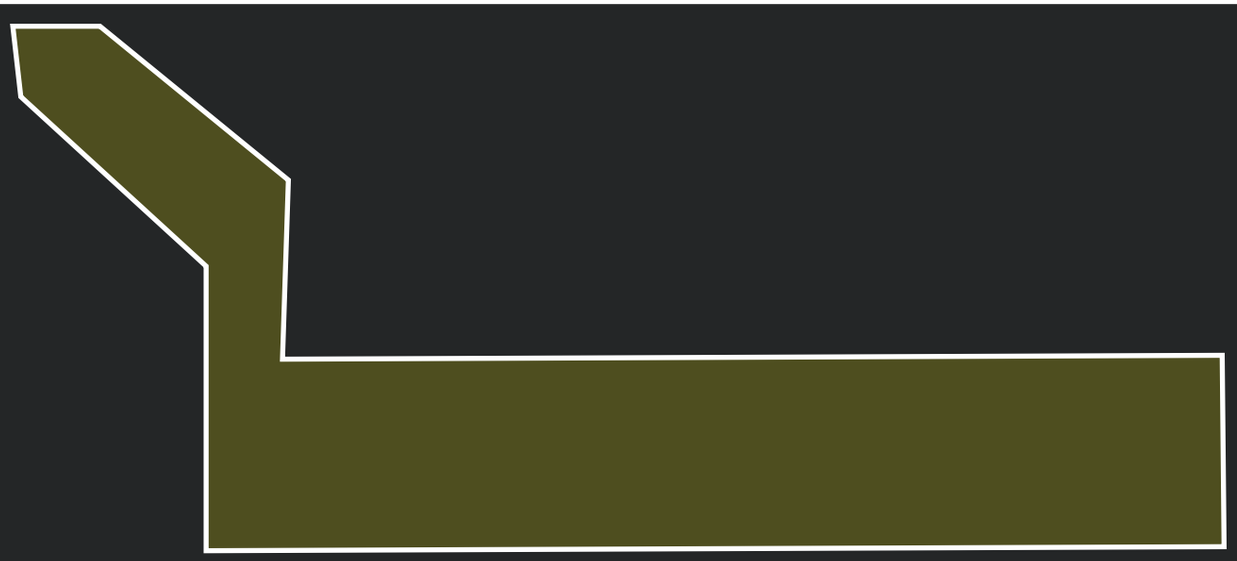
It “**closes up**” those references

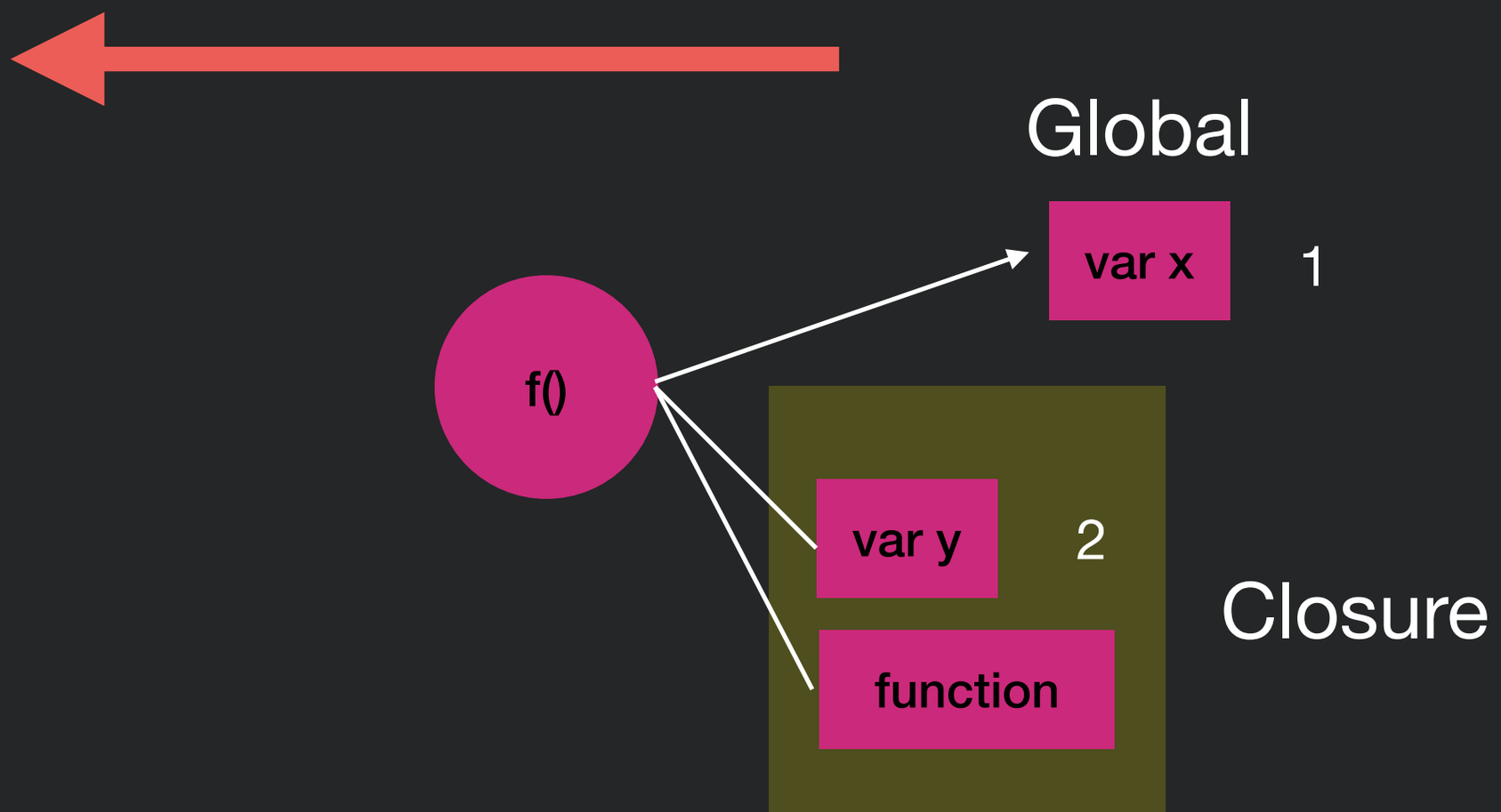
Closures





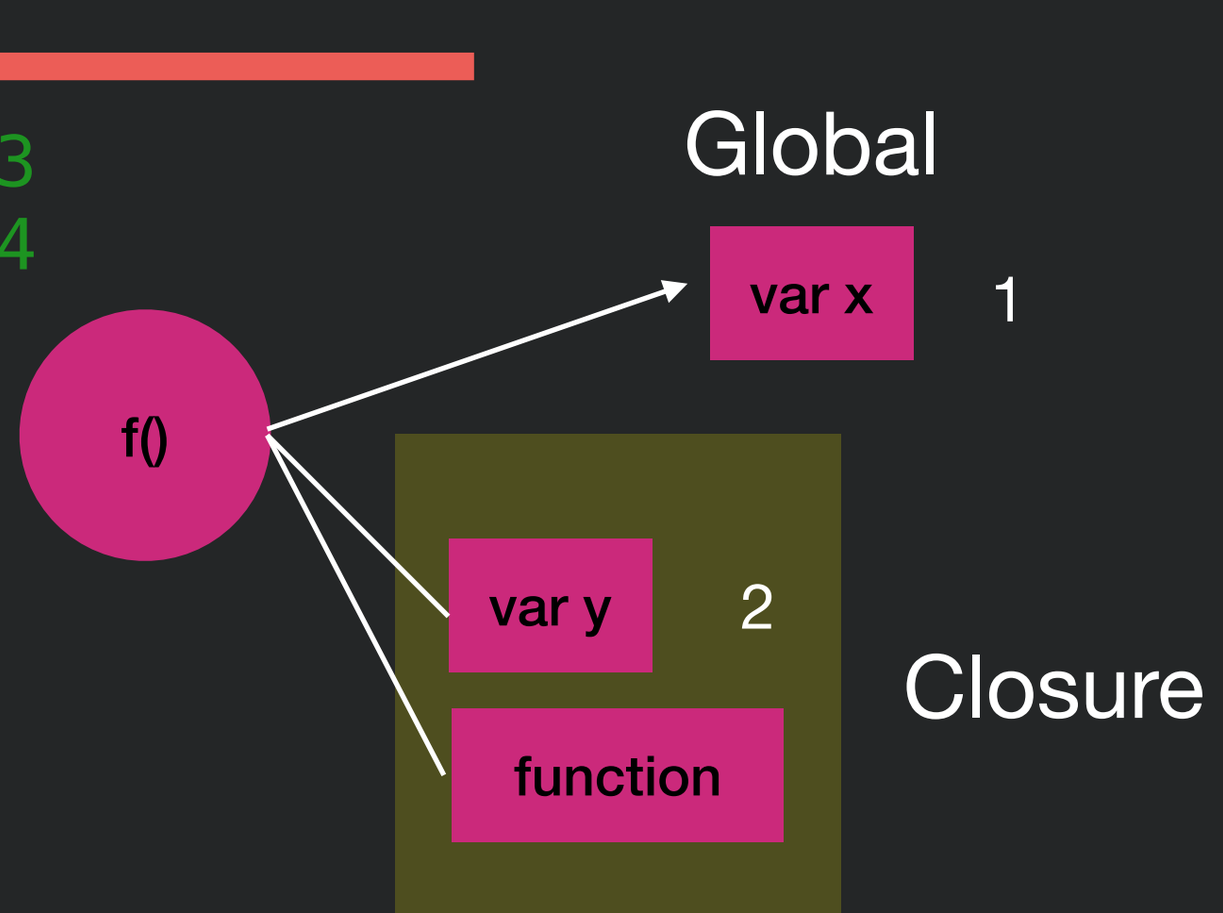
Closures



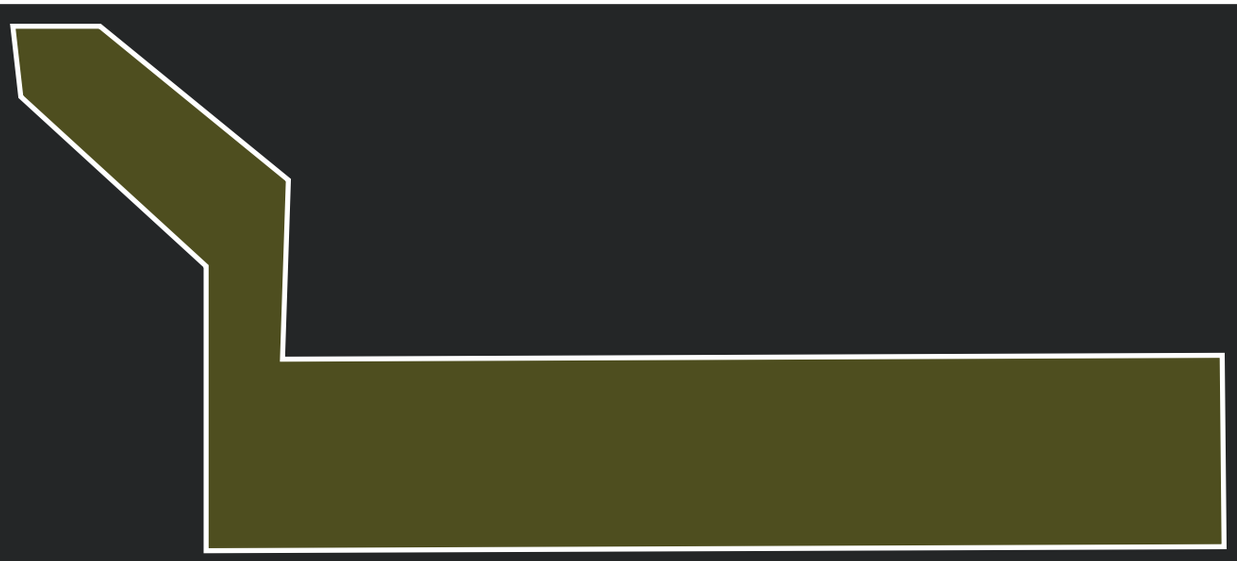


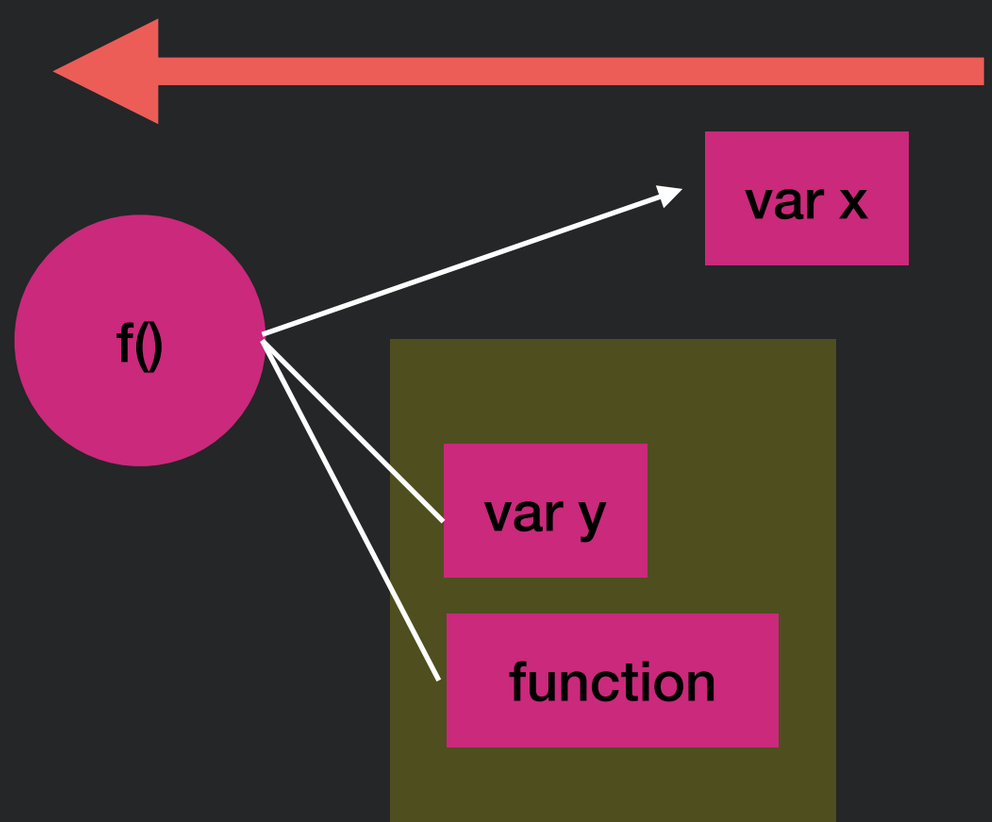
Closures

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();          // 1+2 is 3
g();          // 1+3 is 4
```

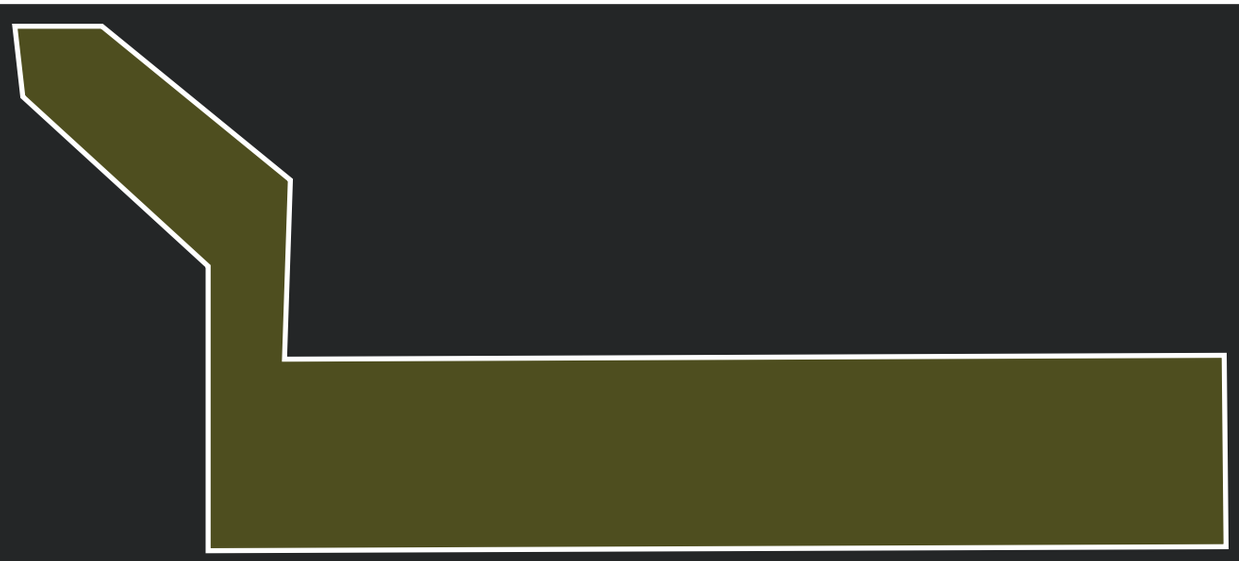


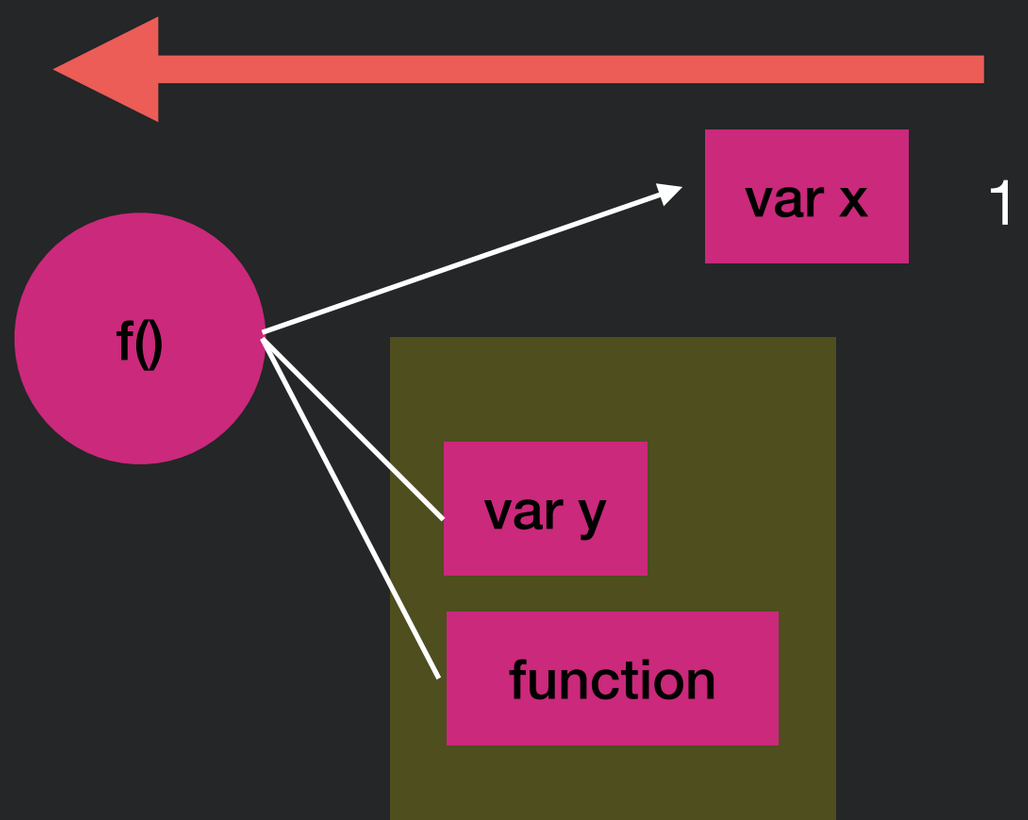
Closures



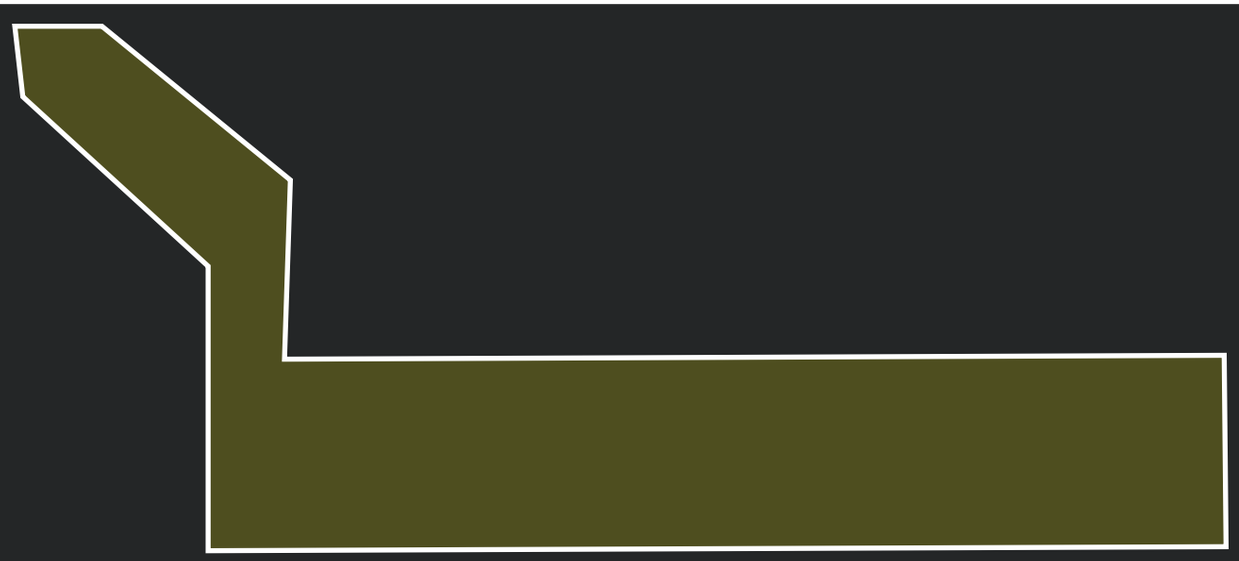


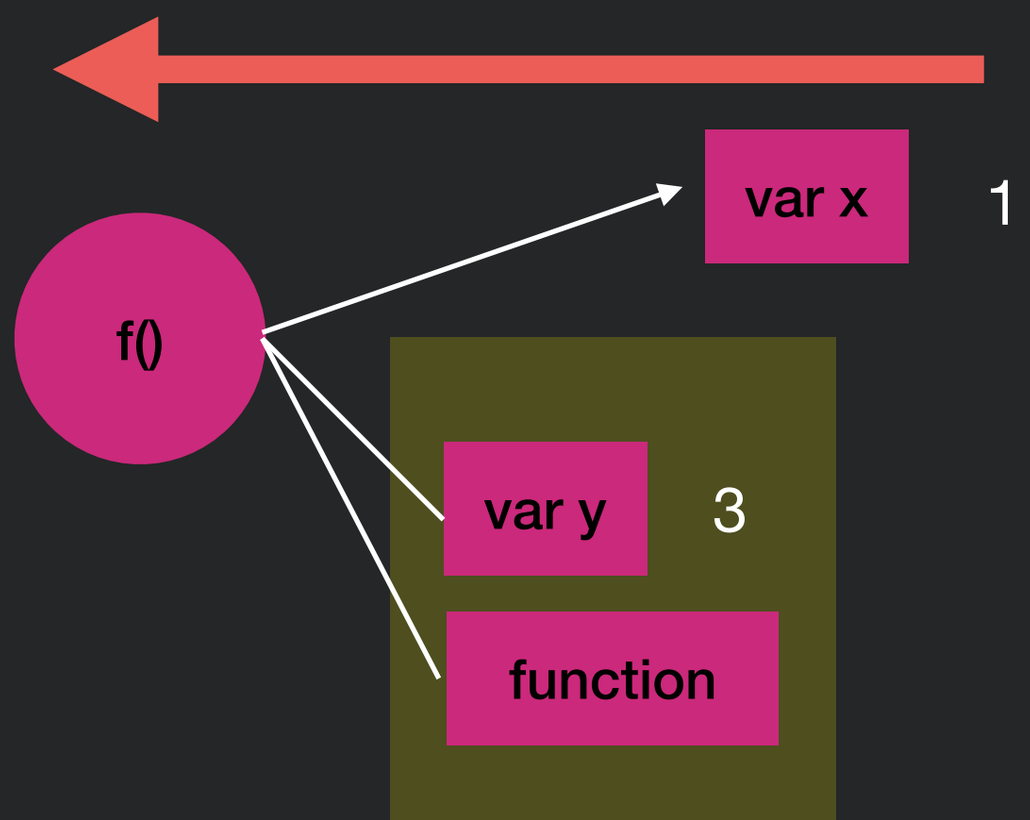
Closures





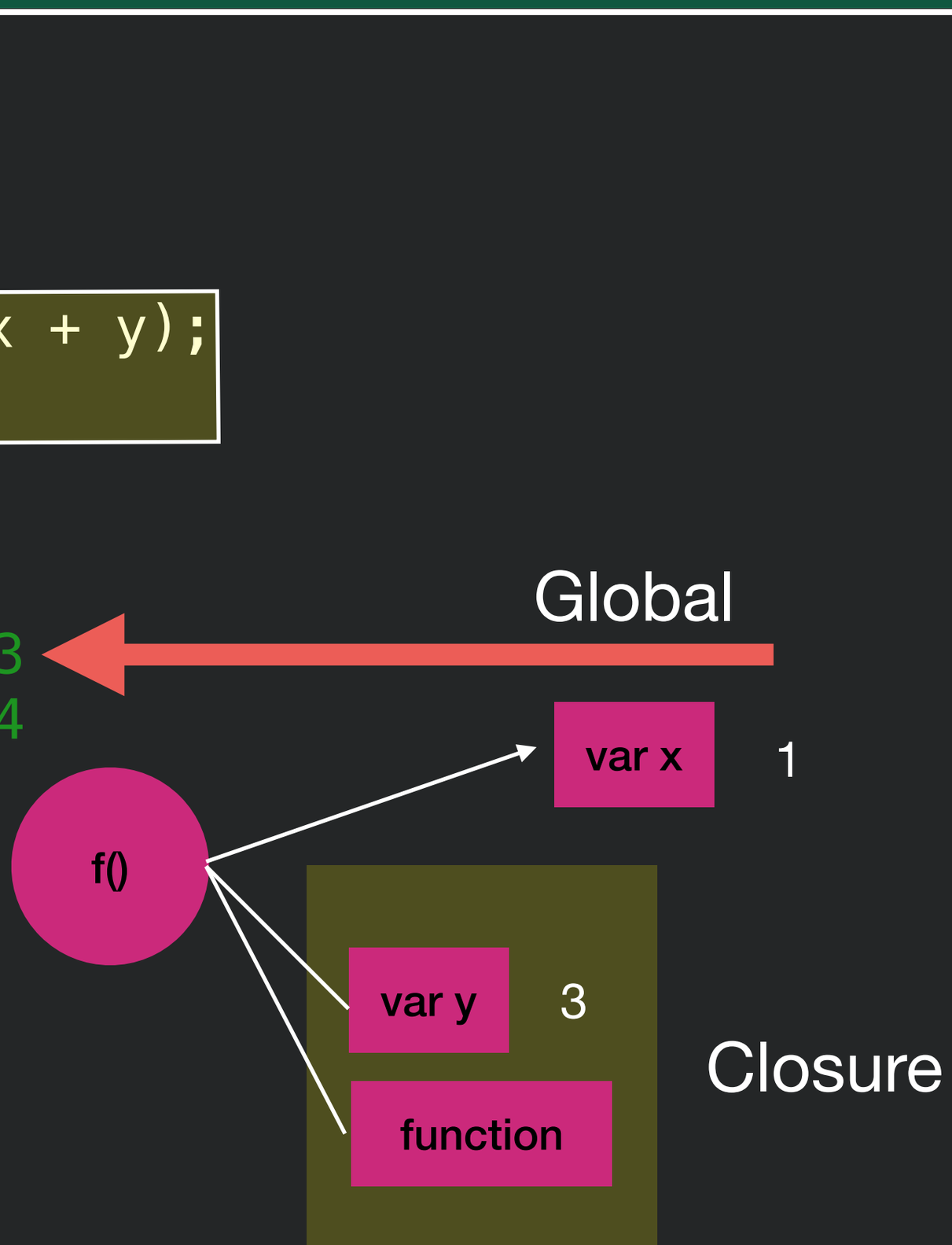
Closures



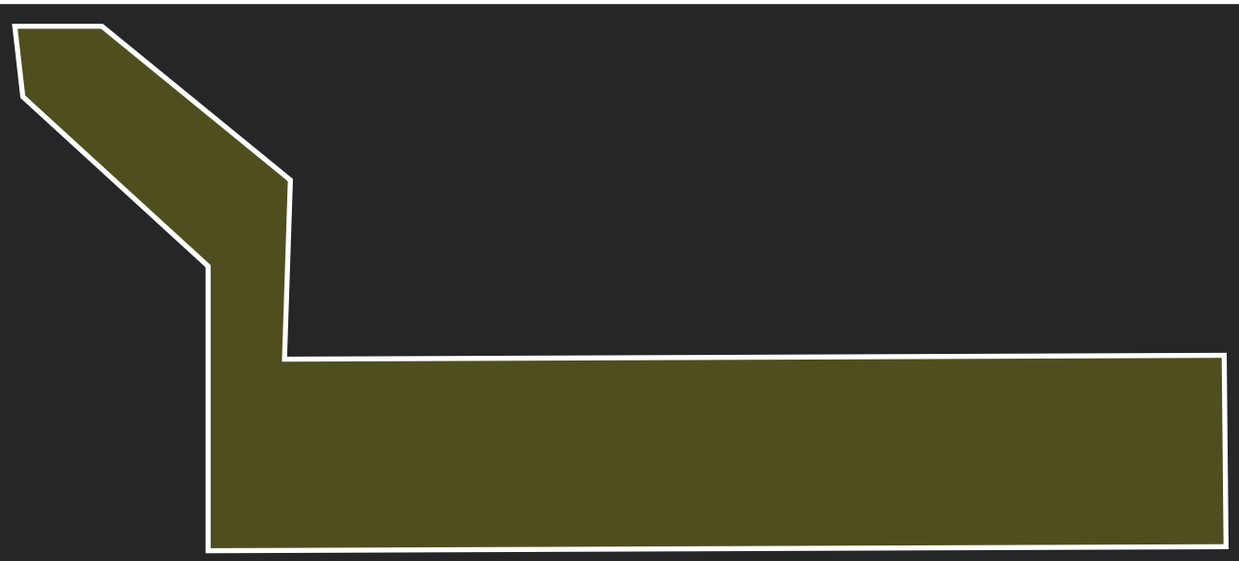


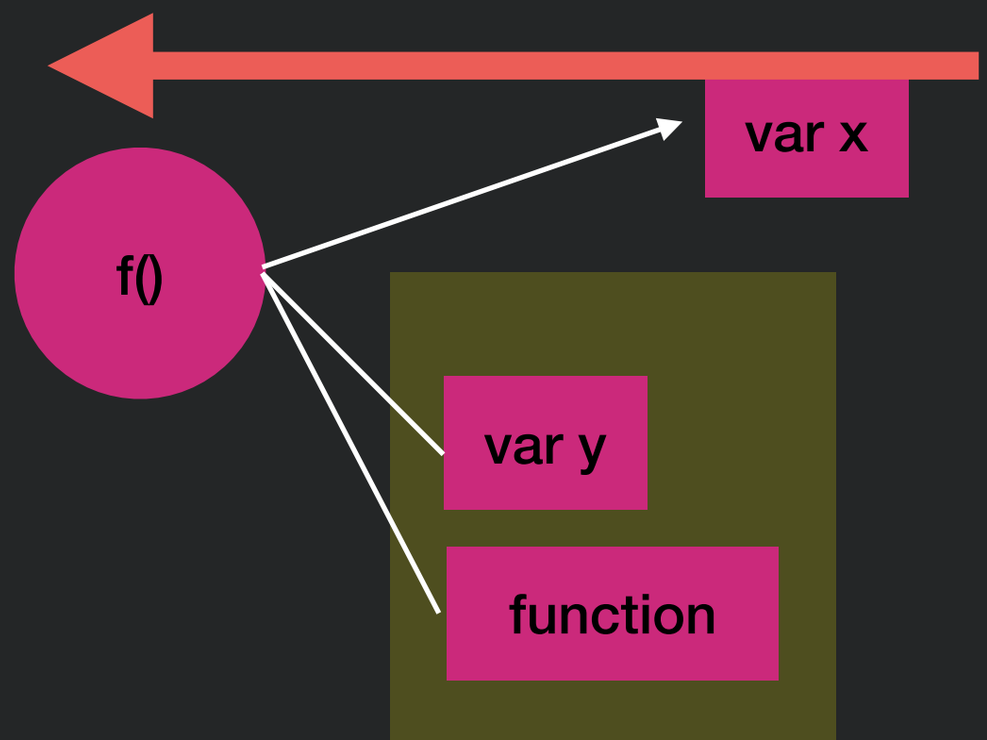
Closures

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();                // 1+2 is 3
g();                // 1+3 is 4
```

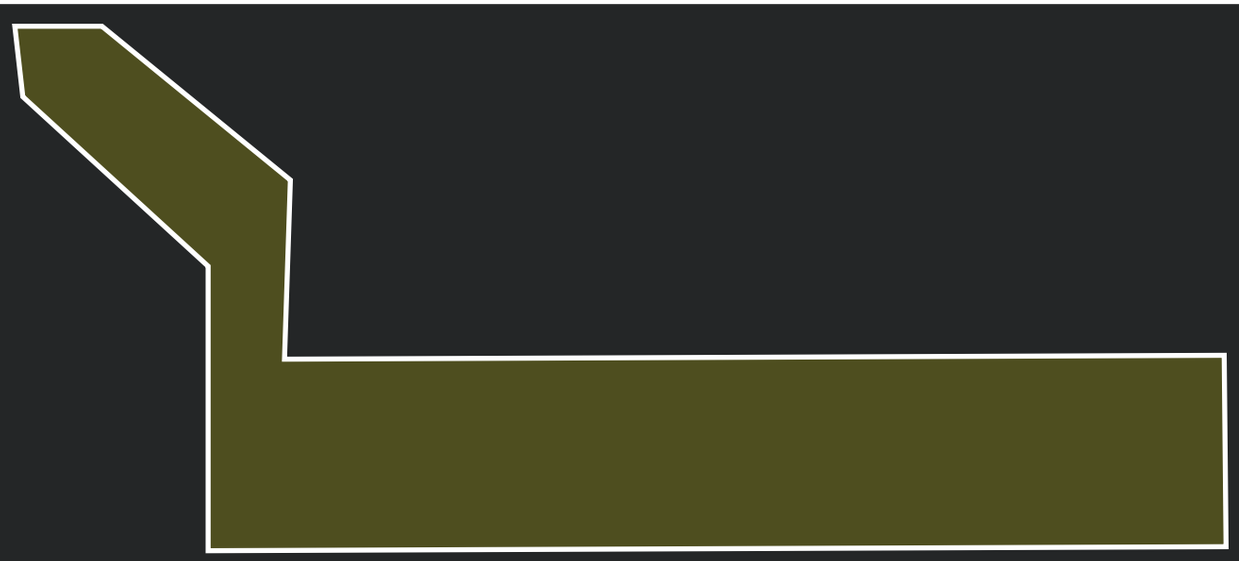


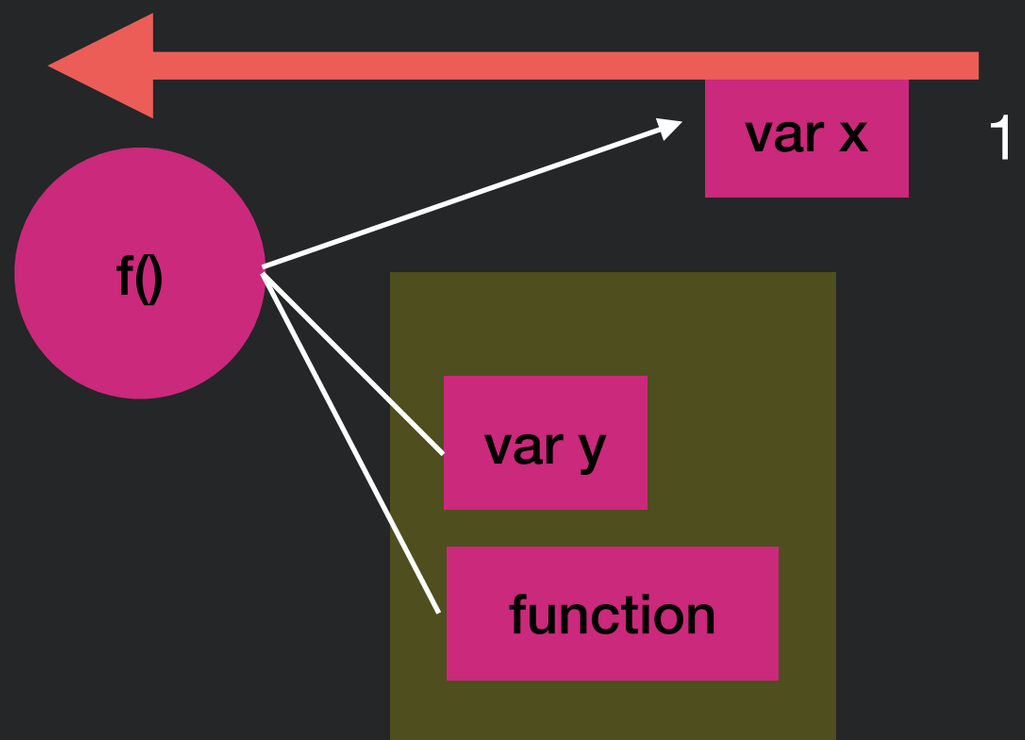
Closures



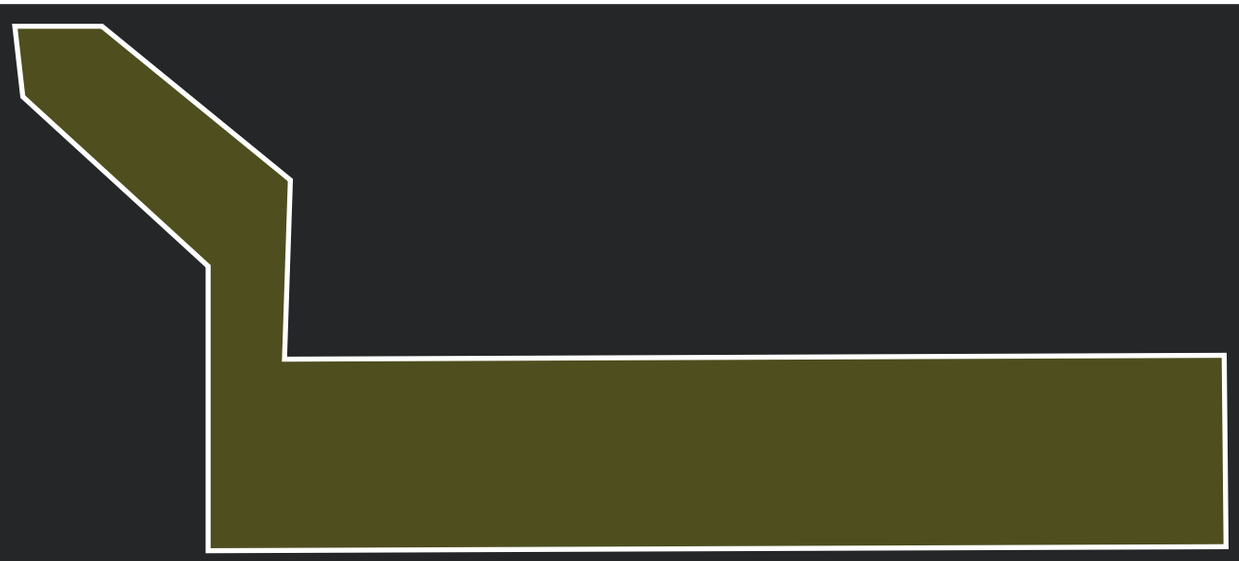


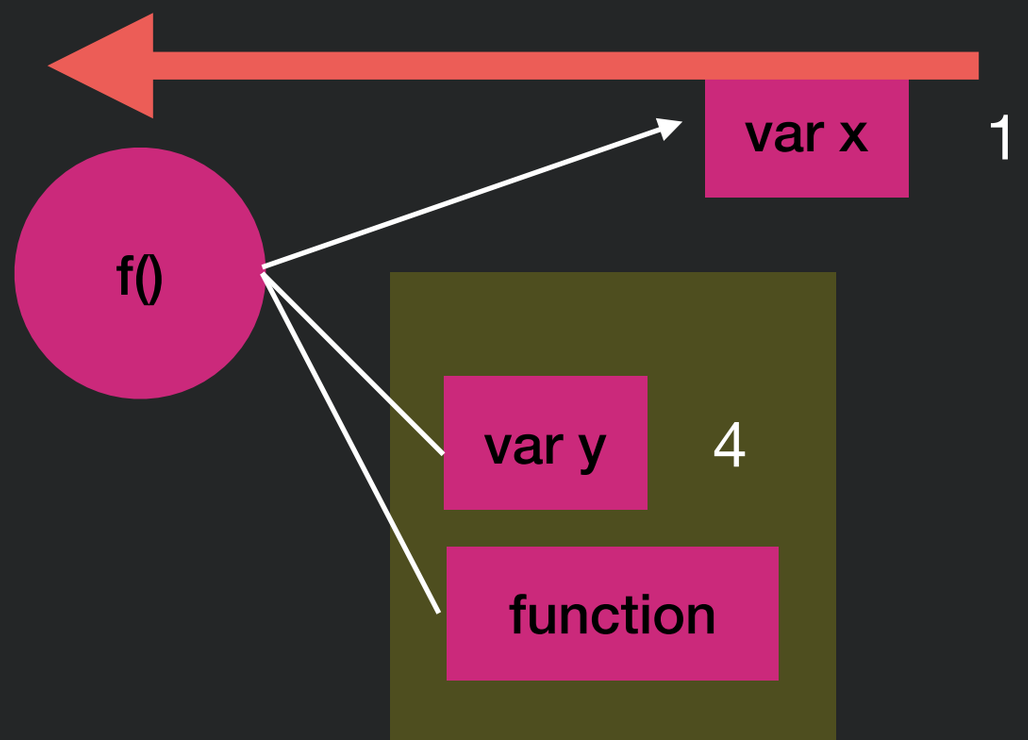
Closures





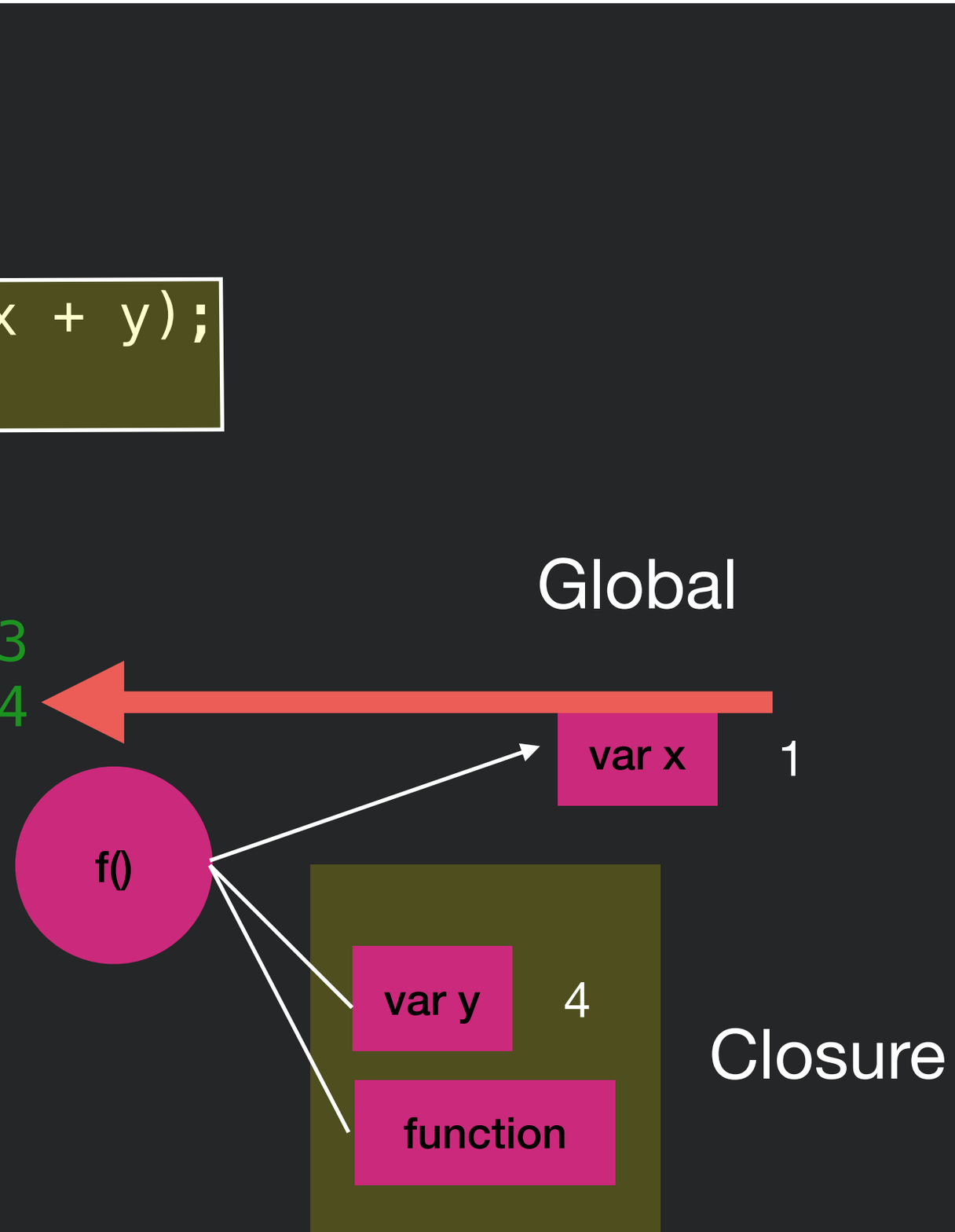
Closures





Closures

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();                // 1+2 is 3
g();                // 1+3 is 4
```



Modules

- We can do it with closures!
- Define a function
  - Variables/functions defined in that function are “private”
  - Return an object - every member of that object is public!
- ***Remember:*** Closures have access to the outer function’s variables even after it returns

Modules with Closures

```
var facultyAPI = (function(){
  var faculty = [{name:"Prof Moran", section: 2}, {name:"Prof
Johnson”, section:1}];

  return {
   getFaculty : function(i)
   {
    return faculty[i].name + " ("+faculty[i].section +")";
   }

 };
})();

console.log(facultyAPI.getFaculty(0));
```

Modules with Closures

```
var facultyAPI = (function(){
  var faculty = [{name:"Prof Moran", section: 2}, {name:"Prof
Johnson”, section:1}];

  return {
   getFaculty : function(i)
   {
    return faculty[i].name + " ("+faculty[i].section +")";
   }

 };
})();

console.log(facultyAPI.getFaculty(0));
```

This works because inner functions have visibility to all variables of outer functions!

Closures Gone Awry

```
var funcs = [];
for (var i = 0; i < 5; i++) {
    funcs[i] = function() { return i; };
}
```

Closures Gone Awry

```
var funcs = [];
for (var i = 0; i < 5; i++) {
    funcs[i] = function() { return i; };
}
```

What is the output of `funcs[0]()`?

Closures Gone Awry

```
var funcs = [];
for (var i = 0; i < 5; i++) {
    funcs[i] = function() { return i; };
}
```

What is the output of `funcs[0]()`?

>5

Closures Gone Awry

```
var funcs = [];
for (var i = 0; i < 5; i++) {
    funcs[i] = function() { return i; };
}
```

What is the output of `funcs[0]()`?

>5

Why?

Closures Gone Awry

```
var funcs = [];
for (var i = 0; i < 5; i++) {
    funcs[i] = function() { return i; };
}
```

What is the output of `funcs[0]()`?

>5

Why?

Closures Gone Awry

```
var funcs = [];
for (var i = 0; i < 5; i++) {
    funcs[i] = function() { return i; };
}
```

What is the output of `funcs[0]()`?

>5

Why?

Closures retain a *pointer* to their needed state!

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  funcs[i] = makeFunction(i);
}
```

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  funcs[i] = makeFunction(i);
}
```

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  funcs[i] = makeFunction(i);
}
```

Why does it work?

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  funcs[i] = makeFunction(i);
}
```

Why does it work?

Each time the anonymous function is called, it will create a ***new variable*** `n`, rather than reusing the same variable `i`

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  funcs[i] = makeFunction(i);
}
```

Why does it work?

Each time the anonymous function is called, it will create a ***new variable*** `n`, rather than reusing the same variable `i`

Shortcut syntax:

```
var funcs = [];
for (var i = 0; i < 5; i++) {
  funcs[i] = (function(n) {
    return function() { return n; }
  })(i);
}
```

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  funcs[i] = makeFunction(i);
}
```

Why does it work?

Each time the anonymous function is called, it will create a ***new variable*** `n`, rather than reusing the same variable `i`

Shortcut syntax:

```
var funcs = [];
for (var i = 0; i < 5; i++) {
  funcs[i] = (function(n) {
    return function() { return n; }
  })(i);
}
```

Exercise: Closures

```
var facultyAPI = (function(){
  var faculty = [{name:"Prof Moran", section: 2}, {name:"Prof
Johnson”, section:1}];

  return {
   getFaculty : function(i)
   {
    return faculty[i].name + " ("+faculty[i].section +")";
   }

 };      C
})();

console.log(facultyAPI.getFaculty(0));
```

Here’s our simple closure. Add a new function to create a new faculty, then call `getFaculty` to view their formatted name.

Javascript Tooling & Testing

Review: JSON: JavaScript Object Notation

Open standard format for transmitting *data* objects.

No functions, only key / value pairs

Values may be other objects or arrays

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: “ENGR 6409”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 6409",
    fullName: {
        firstName: “Alyssa”,
        lastName: “P Hacker”}
};
```

JSON Object

JavaScript Tooling & Testing

- Web Development Tools
- What's behavior driven development and why do we want it?
- Some tools for testing web apps - focus on Jest

An (older) Way to Export Modules

- Prior to ES6, was no language support for exposing modules.
- Instead did it with libraries (e.g., node) that handled exports
- Works similarly: declare what functions / classes are publicly visible, import classes
- Syntax:
  In the file exporting a function or class sum:
  ```
  module.exports = sum;
  ```

  In the file importing a function or class sum:
  ```
  const sum = require('./sum');
  ```

  Where sum.js is the name of a file which defines sum.

Options for Executing JavaScript

- Browser
  - Pastebin—useful for debugging & experimentation
- Outside of the browser (focus for now)
  - node.js—runtime for JavaScript

Demo: Pastebin

```
var course = { name: 'SWE 432' };

console.log('Hello'  + course.name + '!');
```

https://replit.com/@kmoran/SWE-Replit-Demo#script.js

Node.js

- Node.js is a *runtime* that lets you run JS outside of a browser
- We're going to write backends with Node.js
- Download and install it: https://nodejs.org/en/
  - We recommend LTS (LTS -> Long Term Support, designed to be super stable)

Demo: Node.js

```
var course = { name: 'SWE 432' };

console.log('Hello'  + course.name + '!');
```

Node Package Manager

Working with Libraries

"The old way"

```
<script src="https://fb.me/react-15.0.0.js"></script>
<script src="https://fb.me/react-dom-15.0.0.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/
browser.min.js"></script>
```

- What's wrong with this?
  - No standard format to say:
    - What's the name of the module?
    - What's the version of the module?
    - Where do I find it?
  - Ideally: Just say "Give me React 15 and everything I need to make it work!"

A Better Way for Modules

A Better Way for Modules

- Describe what your modules are

A Better Way for Modules

- Describe what your modules are
- Create a central repository of those modules

A Better Way for Modules

- Describe what your modules are
- Create a central repository of those modules
- Make a utility that can automatically find and include those modules

A Better Way for Modules

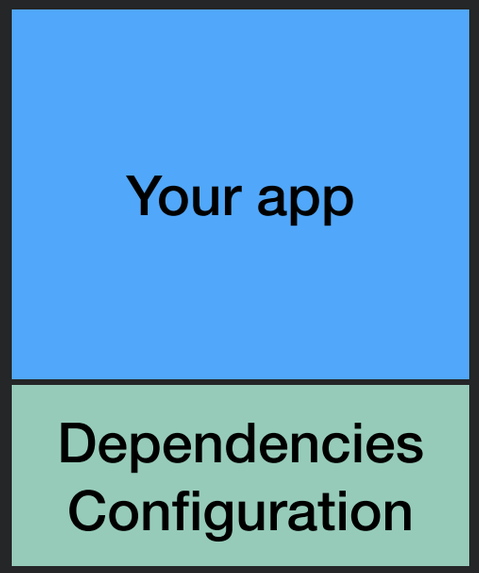
- Describe what your modules are
- Create a central repository of those modules
- Make a utility that can automatically find and include those modules

Your app

Assumes dependencies magically exist

A Better Way for Modules

- Describe what your modules are
- Create a central repository of those modules
- Make a utility that can automatically find and include those modules

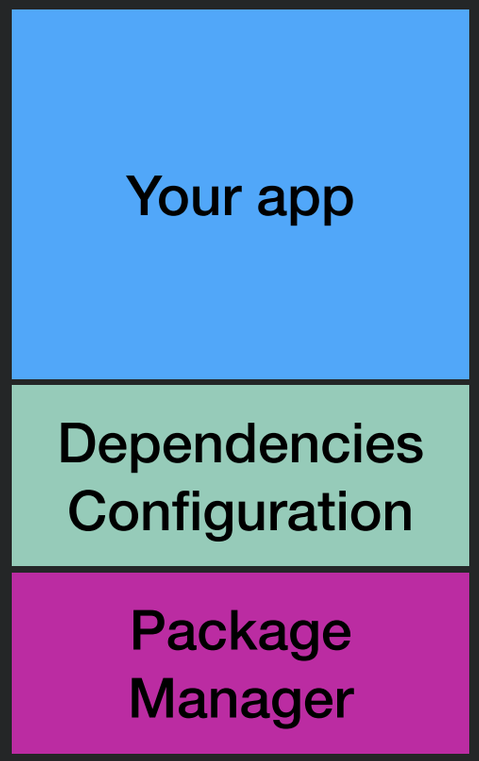


Assumes dependencies magically exist

Declares what modules you need

A Better Way for Modules

- Describe what your modules are
- Create a central repository of those modules
- Make a utility that can automatically find and include those modules



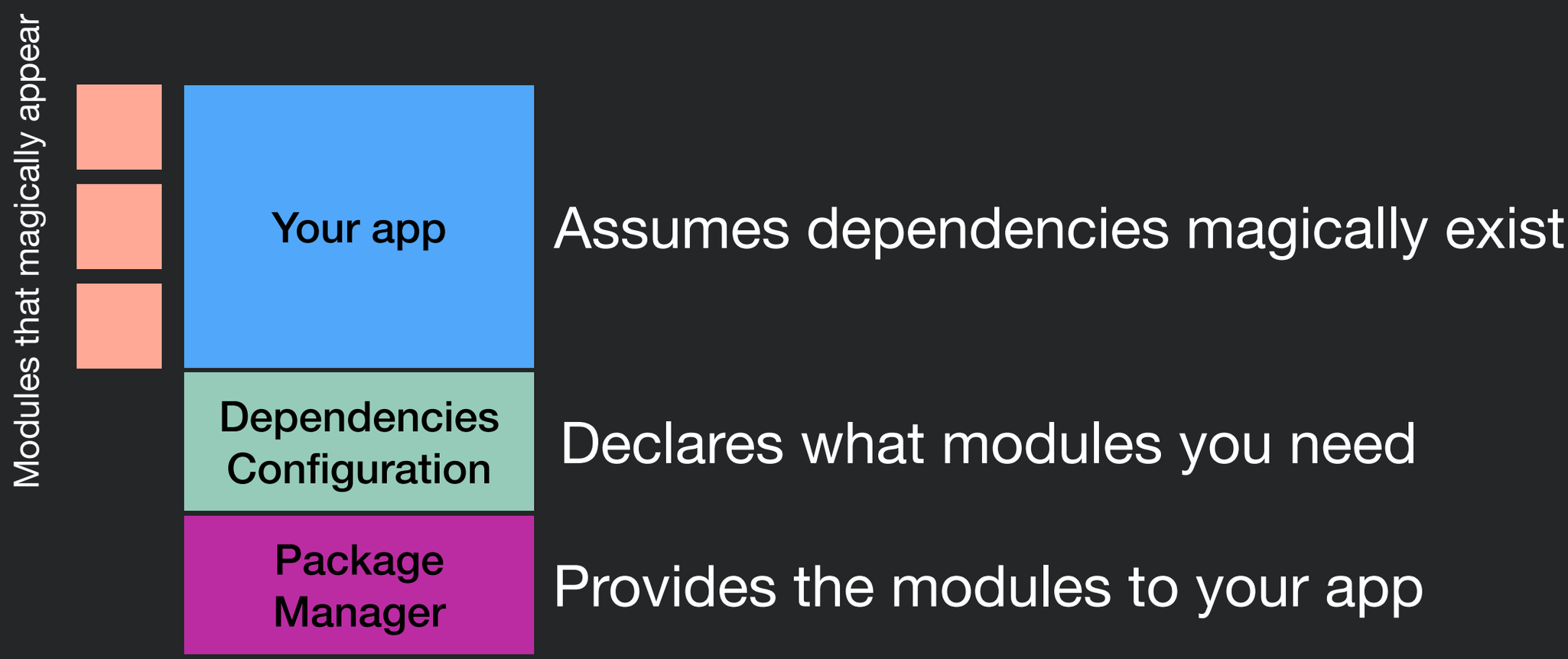
Assumes dependencies magically exist

Declares what modules you need

Provides the modules to your app

A Better Way for Modules

- Describe what your modules are
- Create a central repository of those modules
- Make a utility that can automatically find and include those modules



NPM: Not an acronym, but the Node Package Manager

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”
  - You can run it by saying “node index.js”

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”
  - You can run it by saying “node index.js”
  - “I need express, the most recent version is fine”

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”
  - You can run it by saying “node index.js”
  - “I need express, the most recent version is fine”
- Config is stored in json - specifically package.json

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”
  - You can run it by saying “node index.js”
  - “I need express, the most recent version is fine”
- Config is stored in json - specifically package.json

Generated by `npm` commands:

```
{
  "name": "helloworld",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test
specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.14.0"
  }
}
```

Installing packages with NPM

- `npm install <package> --save` will download a package and add it to your package.json
- `npm install` will go through all of the packages in package.json and make sure they are installed/up to date
- Packages get installed to the `node_modules` directory in your project

Using NPM

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project

```
npm init
```

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
  ```
  npm init
  ```
- Step 2: Declare dependencies
  ```
  npm install <packagename> --save
  ```

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
  ```
  npm init
  ```
- Step 2: Declare dependencies
  ```
  npm install <packagename> --save
  ```
- Step 3: Use modules in your app
  ```
  var myPkg = require(“packagename”)
  ```

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
  ```
  npm init
  ```
- Step 2: Declare dependencies
  ```
  npm install <packagename> --save
  ```
- Step 3: Use modules in your app
  ```
  var myPkg = require(“packagename”)
  ```
- Do NOT include node_modules in your git repo! Instead, just do
  ```
  npm install
  ```

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
  ```
  npm init
  ```
- Step 2: Declare dependencies
  ```
  npm install <packagename> --save
  ```
- Step 3: Use modules in your app
  ```
  var myPkg = require(“packagename”)
  ```
- Do NOT include node_modules in your git repo! Instead, just do
  ```
  npm install
  ```
  - This will download and install the modules on your machine given the existing config!

https://docs.npmjs.com/index

NPM Scripts

- Scripts that run at specific times.
- For starters, we'll just worry about *test* scripts

https://docs.npmjs.com/misc/scripts

```
{
    "name": "starter-node-react",
    "version": "1.1.0",
    "description": "a starter project structure for react-app",
    "main": "src/server/index.js",
    "scripts": {
        "start": "babel-node src/server/index.js",
        "build": "webpack --config config/webpack.config.js",
        "dev": "webpack-dev-server --config config/webpack.config.js --
devtool eval --progress --colors --hot --content-base dist/"
    },
    "repository": {
        "type": "git",
        "url": "git+https://github.com/wwsun/starter-node-react.git"
    },
    "author": "Weiwei SUN",
    "license": "MIT",
    "bugs": {
        "url": "https://github.com/wwsun/starter-node-react/issues"
    },
    "homepage": "https://github.com/wwsun/starter-node-react#readme",
    "dependencies": {
        "babel-cli": "^6.4.5",
        "babel-preset-es2015-node5": "^1.1.2",
        "co-views": "^2.1.0",
        "history": "^2.0.0-rc2",
        "koa": "^1.0.0",
        "koa-logger": "^1.3.0",
        "koa-route": "^2.4.2",
        "koa-static": "^2.0.0",
        "react": "^0.14.0",
        "react-dom": "^0.14.0",
        "react-router": "^2.0.0-rc5",
        "swig": "^1.4.2"
    },
    "devDependencies": {
        "babel-core": "^6.1.2",
        "babel-loader": "^6.0.1",
        "babel-preset-es2015": "^6.3.13",
        "babel-preset-react": "^6.1.2",
        "webpack": "^1.12.2",
        "webpack-dev-server": "^1.14.1"
    }
```

Demo: NPM

Unit Testing

- Unit testing is testing some program unit in isolation from the rest of the system (which may not exist yet)
- Usually the programmer is responsible for testing a unit during its implementation (even though this violates the rule about a programmer not testing own software)
- Easier to debug when a test finds a bug (compared to full-system testing)

Integration Testing

- ***Motivation:*** Units that worked in isolate may not work in combination
- Performed after all units to be integrated have passed all unit tests
- Reuse unit test cases that cross unit boundaries (that previously required stub(s) and/or driver standing in for another unit)

Unit vs Integration Tests

Unit vs Integration Tests

Writing Good Tests

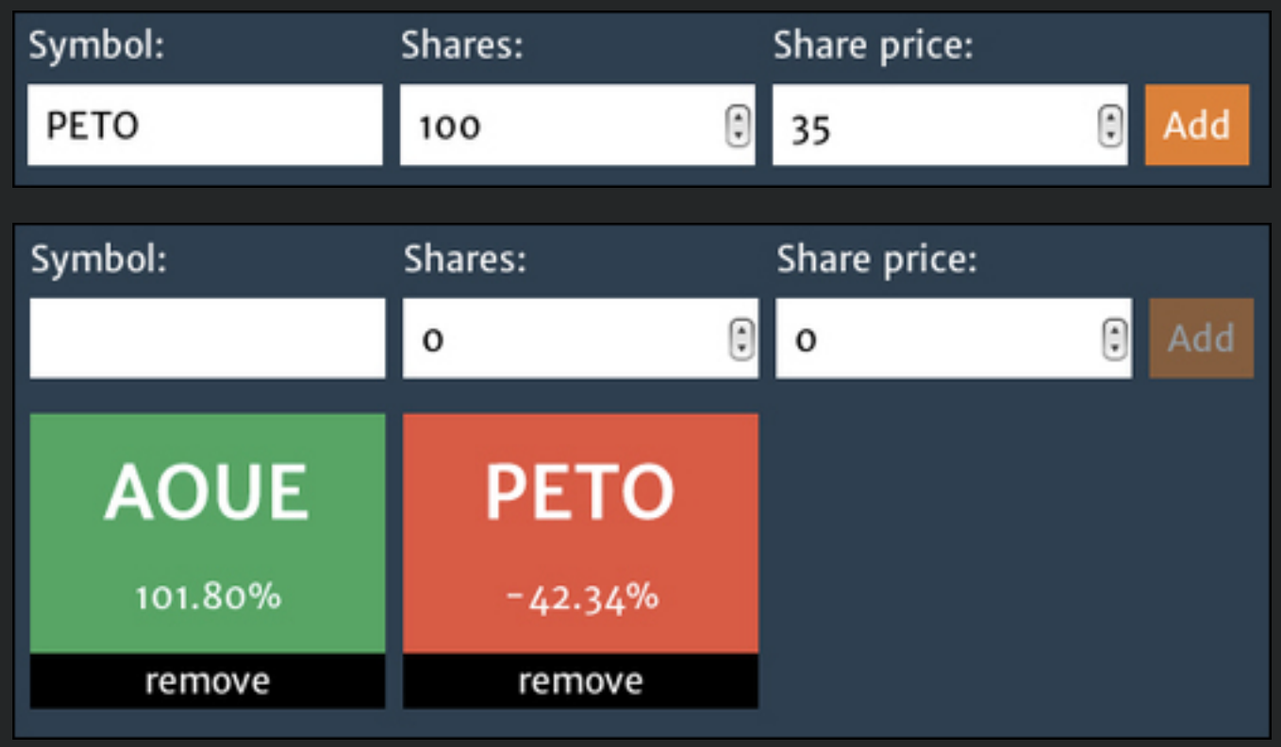
- How do we know when we have tested “enough”?
  - Did we test all of the features we created?
  - Did we test all possible values for those features?

Behavior Driven Development

- Establish *specifications* that say what an app should do
- We write our spec *before* writing the code!
- Only write code if it's to make a spec work
- Provide a mapping between those specifications, and some observable application functionality
- This way, we can have a clear map from specifications to tests

Investment Tracker

- Users make investments by entering a ticker symbol, number of shares, and the price that the user paid per share
- Once the investment has been input, the user can see the current status of their investments
- How do we test this?



Investment Tracker

- What's an investment for our app?
  - Given an investment, it:
    - Should be of a stock
    - Should have the invested shares quantity
    - Should have the share paid price
    - Should have a current price
    - When its current price is higher than the paid price:
      - It should have a positive return of investment
      - It should be a good investment

jestjs.io

Jest 23.3

Search

Docs API Help Blog Aあ English GitHub

Jest

Delightful JavaScript Testing

TRY OUT JEST | GET STARTED | WATCH TALKS | LEARN MORE

★ Star 20,058

Developer Ready

Complete and ready to set-up JavaScript testing solution. Works out of the box for any React project.

Instant Feedback

Fast interactive watch mode runs only test files related to changed files and is optimized to give signal quickly.

Snapshot Testing

Capture snapshots of React trees or other serializable values to simplify testing and to analyze how state changes over time.

Jest Lets You Specify Behavior in *Specs*

Jest Lets You Specify Behavior in *Specs*

- Specs are written in JS

Jest Lets You Specify Behavior in *Specs*

- Specs are written in JS
- Key functions:
  - `describe`, `test`, `expect`

Jest Lets You Specify Behavior in *Specs*

- Specs are written in JS
- Key functions:
  - `describe`, `test`, `expect`
- `Describe` a high level scenario by providing a name for the scenario and function(s) that contains some `test`s by saying what you `expect` it to be

Jest Lets You Specify Behavior in *Specs*

- Specs are written in JS
- Key functions:
  - `describe, test, expect`

- `Describe` a high level scenario by providing a name for the scenario and function(s) that contains some `test`s by saying what you `expect` it to be

- Example:

```
describe("Alyssa P Hacker tests", () => {
    test("Calling fullName directly should always work", () => {
        expect(profHacker.fullName()).toEqual("Alyssa P Hacker");
    });
}
```

Writing Specs

- Can specify some code to run before or after checking a spec

```
var profHacker;
beforeEach(() => {
    profHacker = {
        firstName: "Alyssa",
        lastName: "P Hacker",
        teaches: "SWE 432",
        office: "ENGR 6409",
        fullName: function () {
            return this.firstName + " " + this.lastName;
        }
    };
});
```

Making it work

- Add jest to your project (npm install --save-dev jest)
- Configure NPM to use jest for test in package.json

```
"scripts": {
  "test": "jest"
},
```

- For file `x.js`, create `x.test.js`
- Run `npm test`

Multiple Specs

- Can have as many tests as you would like

```
test("Calling fullName directly should always work", () => {
     expect(profHacker.fullName()).toEqual("Alyssa P Hacker");
 });

test("Calling fullName without binding but with a function ref is undefined", () => {
    var func = profHacker.fullName;
    expect(func()).toEqual("undefined undefined");
});
test("Calling fullName WITH binding with a function ref works", () => {
    var func = profHacker.fullName;
    func = func.bind(profHacker);
    expect(func()).toEqual("Alyssa P Hacker");
});
test("Changing name changes full name", ()=>{
    profHacker.firstName = "Dr. Alyssa";
    expect(profHacker.fullName()).toEqual("Dr. Alyssa P Hacker");
})
```

Nesting Specs

- “When its current price is higher than the paid price:
  - It should have a positive return of investment
  - It should be a good investment”
- How do we describe that?

```
describe("when its current price is higher than the paid price", function() {
    beforeEach(function() {
      stock.sharePrice = 40;
    });
    test("should have a positive return of investment", function() {
      expect(investment.roi()).toBeGreaterThan(0);
    });
    test("should be a good investment", function() {
      expect(investment.isGood()).toBeTruthy();
    });
  });
});
```

Matchers

- How does Jest determine that something is what we expect?

```
expect(investment.roi()).toBeGreaterThan(0);
expect(investment).isGood().toBeTruthy();
expect(investment.shares).toEqual(100);
expect(investment.stock).toBe(stock);
```

- These are “matchers” for Jest - that compare a given value to some criteria
- Basic matchers are built in:
  - toBe, toEqual, toContain, toBeNaN, toBeNull, toBeUndefined, >, <, >=, <=, != , regular expressions
- Can also define your own matcher

Matchers

```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
});

const shoppingList = [
  'diapers',
  'kleenex',
  'trash bags',
  'paper towels',
  'beer',
];

test('the shopping list has beer on it', () => {
  expect(shoppingList).toContain('beer');
  expect(new Set(shoppingList)).toContain('beer');
});
```

Demo: Jest

In Class Exercise: JEST

- Write a factorial program that computes the factorial of x.
- Write a JEST test suite that ensure that this function works correctly.