Module 1: Course Overview, Programming Languages & Paradigms, and Data Structures & Algorithms

Edgardo Molina, PhD | Lead Instructor

CUNY Tech Prep 2016-2017

Outline

Course Overview

Programming Languages, Paradigms, and Patterns

Data Structures and Algorithms (DS & Alg)

Course Overview

Course Overview

- Weekly Meetings and Assigments
    - Alternating **Lectures** and **Labs** (*roughly*, see Syllabus Schedule)
    - Bi-Weekly HackerRank Challenges
    - Bi-Weekly Project Progress and Assignments
- Projects
    - You will ideate, design, and develop a Demo application
    - You will work in team of 3

Course Overview

- Lecture Meetings will cover
    - Full Stack Application development
    - Security concerns in applications
    - Development lifecycle topics
    - Intro to advanced topics
- Labs will cover
    - HackerRank Coding Challenges - *Solutions Review*
        - Q&A for related Data Structure and Algorithms topics
    - Project development time (with TA and Instructor assistance)

Lectures Goal

Why we do things is as important as how we do things.

> *The goal of lectures is to teach you Full Stack Web Application development in JavaScript. We will cover* ***how*** *we build applications, but we will equally focus on* ***why*** *we choose to build things a certain way. This will lead us to cover and compare alternative technologies and methods for building our applications.*

Labs Goal

Succeed at coding challenges

Build awesome projects

Grading

This program covers a lot of material and will keep you busy

Don’t sweat scores, the point is to practice and learn!

Our grading scale:

- work submitted on-time (PASS)
- work not submitted (FAIL)

Course Material and Resources

All lecture notes will be available in (or linked to from) the class github repository:

`https://github.com/medgardo/ctp2016`

All course communication will occur via email and the course slack:

`https://ctp2016.slack.com/`

Programming Languages, Paradigms, and Patterns

Topics

Compiled vs Interpreted Languages

Static vs Dynamic Typing

“Strong” vs “Weak” Typing

Duck Typing

Concurrency (Asynchronous programming)

Procedural vs Event-driven Programming

Why?

Why are we covering these topics if we're building Full Stack Web Applications?

Why?

Why are we covering these topics if we're building Full Stack Web Applications?

> *The first tech stack choice you will make is to pick a programming language!*
>
> *Did you pick the right tool for the job?*
>
> *And what is the criteria for this choice?*

Language Choice Criteria

My boss (or teacher) told me so!

> *You may be right, but this is not a good response. Especially if you want to make a case for a change.*

Developer knowledge, vendor lock in

> *All practical reasons...*

But what if you had complete choice, which language would you pick?

Language Choice Criteria

Developer performance vs Application performance

JS, Python, Ruby

> *Speed up developer <–> Slower app performance*

C/C++, Java

> *Slow down developer <–> Faster app performance*

Why do these opinions exist?

What are example applications for each case?

Language Choice Criteria

Strength of the language in terms of: libraries, community support

- Leverage domain knowledge
- Package repository and development activity
- Going against the grain: OS in JavaScript, or 3D Game in SQL
- It can be done, but at a cost! Is it critical to your bottomline.
    - (*Things like this are done by companies*)

The following topic discussions will help us classify and understand language strengths formally...

Formal definitions

Compiled Languages

Source code is first converted into a machine code executable

The executable consists of machine instructions that run on a CPU

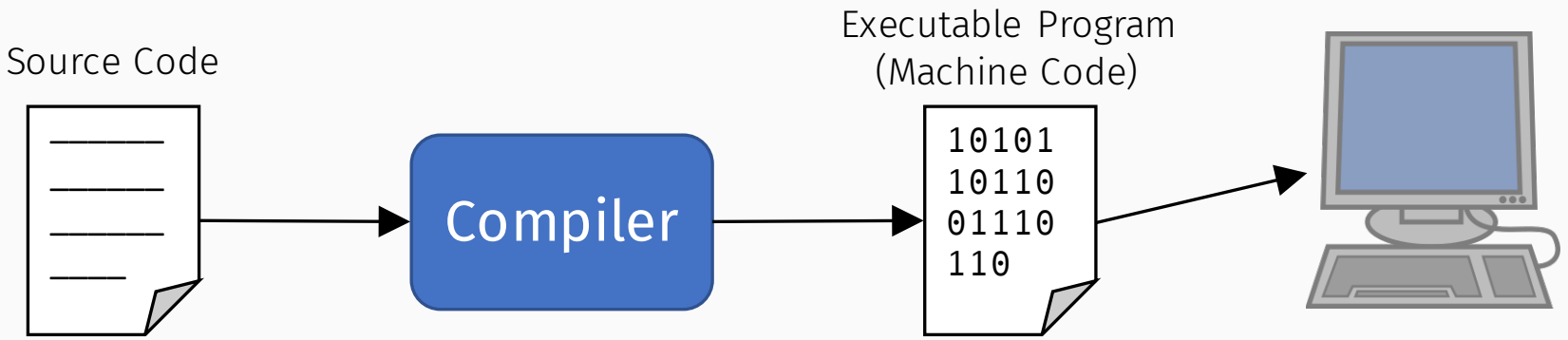

Figure 1: Compilation process

Formal definitions

Interpreted Languages

Source code is converted into machine instruction line-by-line "everytime" the program is run.

Figure 2: Interpretation process

Compiled vs Interpreted Languages

In practice

Java (compiled, loosely)

> *Source Code is compiled to an intermediate format (Bytecode)*
>
> *Bytecode runs on a Virtual Machine (the JVM)*

JavaScript (interpreted)

> *Source code is converted to machine code "on the fly".*
>
> *Example interpreter: V8 Engine in Node.js or Web Browsers*

Compiled vs Interpreted Languages

Many modern languages are actually in between: Java, C#, Python

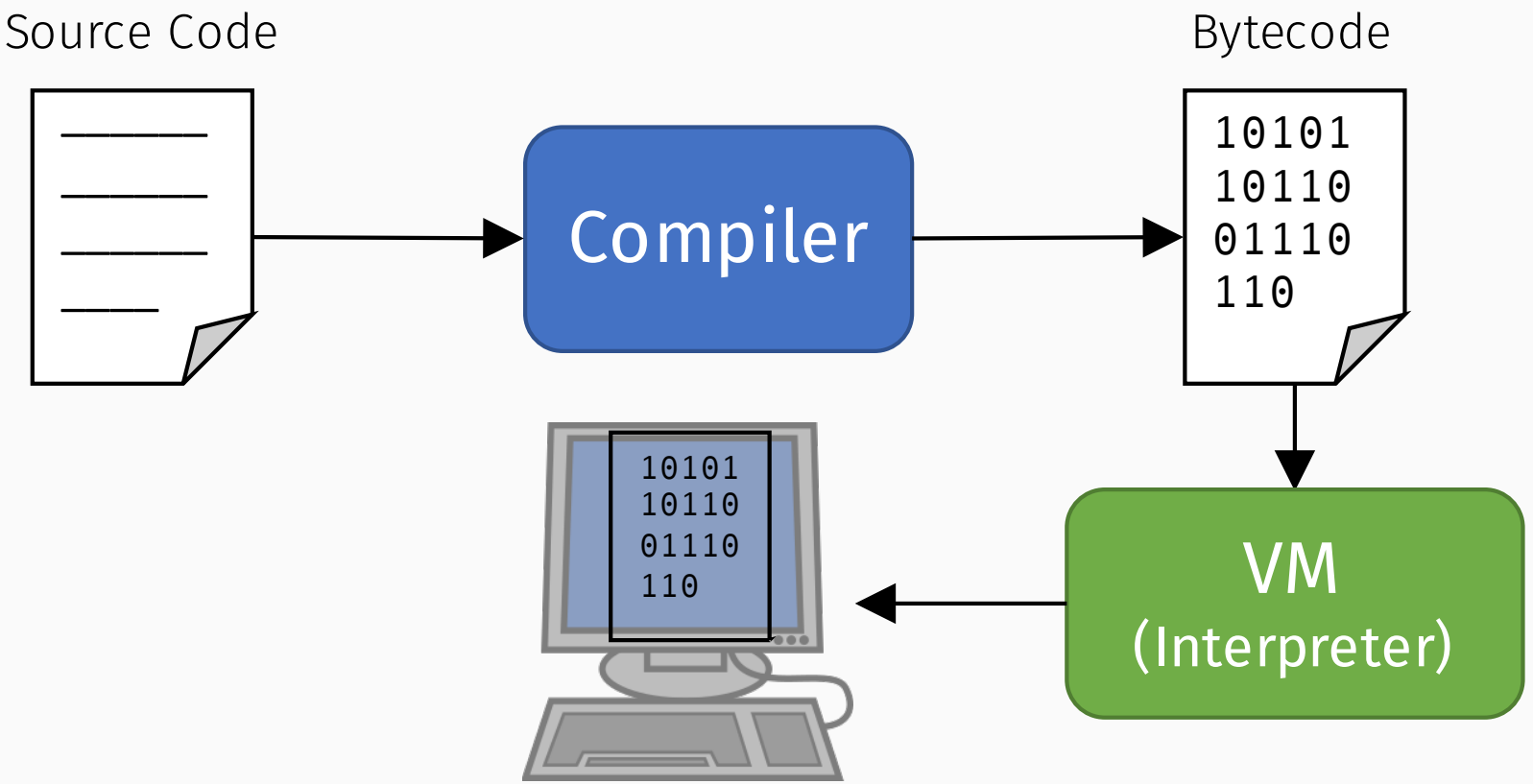

Figure 3: Bytecode interpretation

Compiling

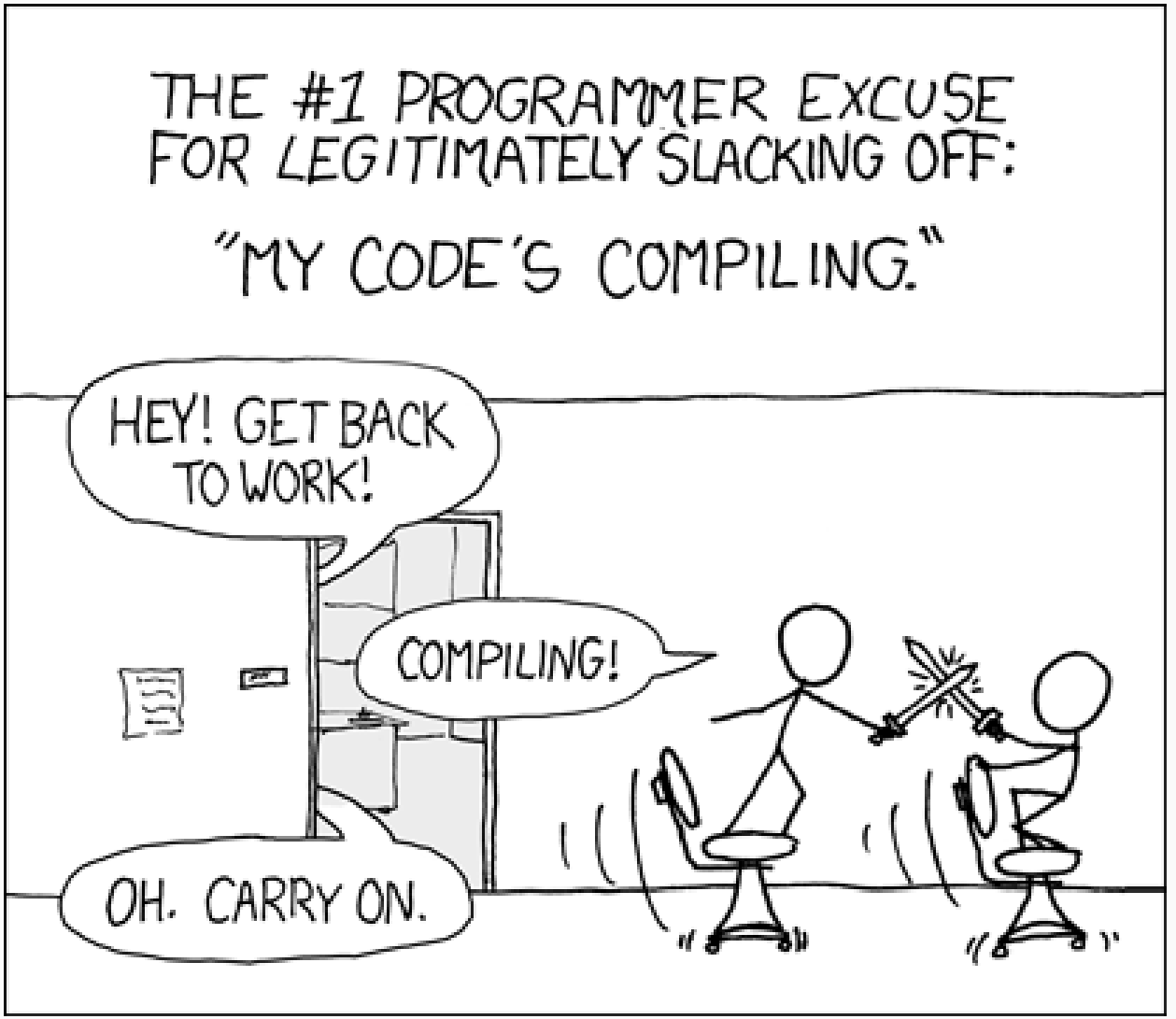

Figure 4: https://xkcd.com/303/

Static vs Dynamic Typing / “Strong” vs “Weak” Typing

Typing refers to the ability to determine a *value’s*, *variable’s*, *expression’s*, or *function’s* data type.

When that determination happens … influences the style of programming.

Statically typed languages

Type checking happens at compile-time.

Dynamically typed languages

Type checking happens at run-time.

Static vs Dynamic Typing / “Strong” vs “Weak” Typing

Further, we have the informal concept of “Strong” vs “Weak” Typing.

This refers to whether the language automatically casts/converts types for you or not.

“Strong” typing

Requires the programmer to explicitly cast one type into another

“Weak” typing

The runtime will implicitly cast one type into another (aka coercion)

Static vs Dynamic Typing / “Strong” vs “Weak” Typing

Static Typing in Java

```
int num = 34;
num = ”23”;
```

Static Typing in Java

```
int num = 34;
num = ”23”;
```

we get the *compilation error*:

```
simple.java:8: error: incompatible types: String cannot
be converted to int
      num = ”23”;
            ^
```

because `num` has a static type that cannot change.

Static vs Dynamic Typing / “Strong” vs “Weak” Typing

Dynamic Typing in JavaScript

```
var num = 34;
num = ”23”;
```

Dynamic Typing in JavaScript

```
var num = 34;
num = ”23”;
```

it works!

The *variable* `num` was allowed to change the data type it represented.

Static vs Dynamic Typing / “Strong” vs “Weak” Typing

“Strong” Typing in Java

```
String s = ”23”;
int x = 40;
res = x-s;
```

Static vs Dynamic Typing / “Strong” vs “Weak” Typing

“Strong” Typing in Java

```
String s = ”23”;
int x = 40;
res = x-s;
```

we get the *compilation error*:

```
simple.java:21: error: bad operand types for
binary operator ’-’
      res = x-s;
              ^
  first type:  int
  second type: String
```

because the String **s** is not casted (or coerced) to an int. (**res**’s type does

Static vs Dynamic Typing / “Strong” vs “Weak” Typing

“Weak” Typing in JavaScript

```
var j = "23";
var k = 3;
var r = j - k;
```

“Weak” Typing in JavaScript

```
var j = "23";
var k = 3;
var r = j - k;
```

it works!

Because JavaScript coerced `j` into a number.

Static vs Dynamic Typing / “Strong” vs “Weak” Typing

Consider the JavaScript code:

```
var person = { talk: () => { console.log("Hi"); } }
var duck = { quack: () => { console.log("Quack"); } }

onlyDucks(person);
```

How should we implement the function `onlyDucks(...)`?

Static vs Dynamic Typing / “Strong” vs “Weak” Typing

Consider the JavaScript code:

```
var person = { talk: () => { console.log("Hi"); } }
var duck = { quack: () => { console.log("Quack"); } }

onlyDucks(person);
```

How should we implement the function `onlyDucks(...)`?

Duck Typing

If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.

Duck Typing

```
// Check if it quacks
function onlyDucks(thing) {
    if('quack' in thing) { ...
    } else { ...
    }
}

// Or ask for forgiveness
function onlyDucks(thing) {
    try {
        thing.quack();
    } catch {
        console.log("Not a duck :(");
    }
}
```

Learn more about typing

More about JavaScript data types and its weak typing rules

`http://cs.lmu.edu/~ray/notes/javascripttypes/`

[*Advanced*] If you're curious about type systems and language design:

`http://cs.lmu.edu/~ray/notes/types/`

Concurrency (Asynchronous programming)

Concurrency means doing multiple things at once.

Examples:

Handling multiple web page requests

Handling simultaneous keyboard and mouse inputs

We can do this either with a Multi-Threaded or an Event system

Concurrency (Asynchronous programming)

Since we're doing multiple things, we have to be careful about

- Sharing state
- Shared resources (*writing to a file*)
- Blocking
- Synchronizing (*when needed*)

Concurrency (Asynchronous programming)

[*Advanced*] Intro to Concurrency:

`http://cs.lmu.edu/~ray/notes/introconcurrency/`

[*Advanced*] See the Java First Example:

`http://cs.lmu.edu/~ray/notes/trivialcpexamples/`

Procedural vs Event-driven Programming

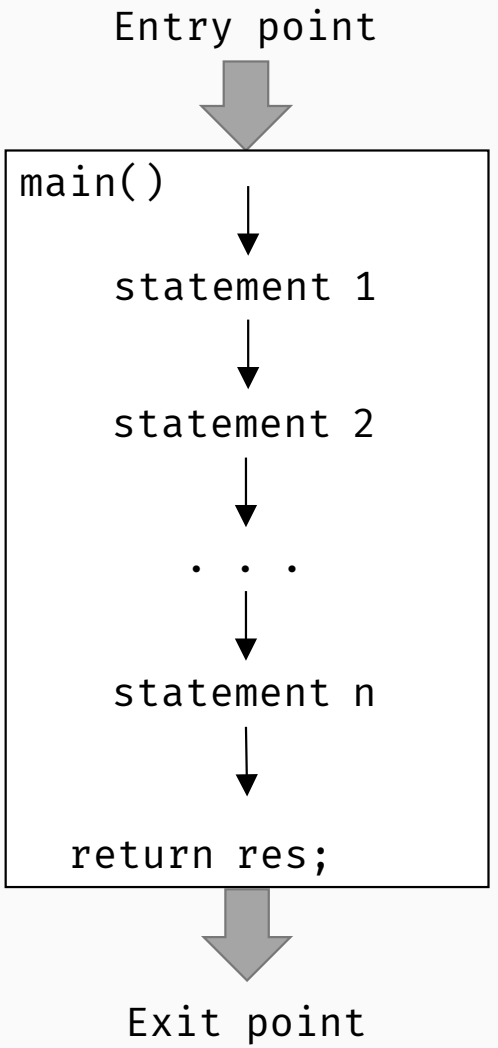

Figure 5: Procedural programming

Procedural vs Event-driven Programming

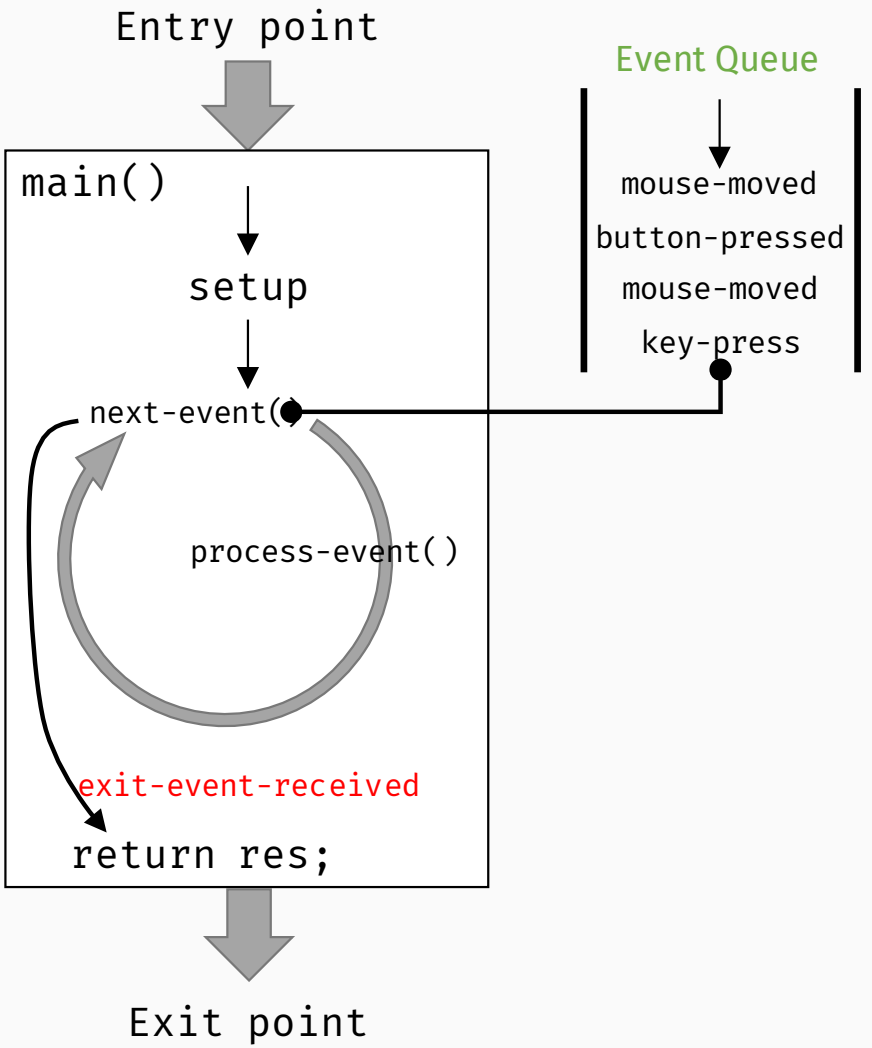

Figure 6: Event-driven programming

Event-driven Programming

Event-loop

Waits for events and applies the corresponding processing function (these are called **callbacks** or **listeners**).

What are events?

Event-driven Programming

Event-loop

Waits for events and applies the corresponding processing function (these are called **callbacks** or **listeners**).

What are events?

Keyboard input, mouse motion and button presses, network connections, timers, etc

Event order is unknown to the programmer

Event-driven Programming

Event-loop

Waits for events and applies the corresponding processing function (these are called **callbacks** or **listeners**).

What are events?

Keyboard input, mouse motion and button presses, network connections, timers, etc

Event order is unknown to the programmer

JavaScript is event-driven

Both, the web browser and web applications are event-driven systems, making JavaScript a good choice for both

Watch this video

"Philip Roberts: What the heck is the event-loop anyway?"

> *This talk is about the event loop in JavaScript. It focuses on the browser, but the same concept applies to node.js on the backend. (Some topics are advanced and not necessary to understand for this course, in particular beyond 22:20.)*
>
> `http://2014.jsconf.eu/speakers/philip-roberts-what-the-heck-is-the-event-loop-anyway.html`
>
> *Direct Link:*
> `https://www.youtube.com/watch?v=8aGhZQkoFbQ`

More Resources

Learn from the JavaScript creator himself:

`http://javascript.crockford.com/`

Prof. Ray Toal's Lecture Notes

`http://cs.lmu.edu/~ray/`

[Extra] Other programming paradigms

`http://cs.lmu.edu/~ray/notes/paradigms/`

Data Structures and Algorithms (DS & Alg)

Topics for coding challenges and lab review

Seven modules

1. Lists/Vectors with Arrays and Linked Lists
2. Stacks, Queues, and Sets
3. Sorting and Searching
4. Binary Trees, Binary Search Trees
5. Maps, HashMaps, Hashing
6. Graphs
7. Traversals, Depth-first search, Breadth-first search

Visualize the Data Structures and Algorithms

USFCA (University of San Francisco) Visualizations

`https://www.cs.usfca.edu/~galles/visualization/Algorithms.html`

Visualgo.net

`http://visualgo.net/list`

Our First Web App

Clone the CTP-Microblog

Clone and run:

```
https://github.com/medgardo/ctp-microblog
```

Coding Challenge Solutions