CSC324 Principles of Programming Languages

Lecture 1

September 14/15, 2020

Welcome to CSC324!

Introduction - Instructors

Lisa Zhang (Course Coordinator)

- Email: lczhang@cs.toronto.edu
- Office Hours: Th/Fri 1pm-2pm and by appointment
- Prefers to be called “Lisa”, “Prof Zhang”, “Prof Lisa”

Andi Bergen

- Email: andi.bergen@utoronto.ca
- Office Hours: Wed 3pm-5pm and by appointment
- Prefers to be called “Andi”, “Prof Bergen”, “Prof Andi”, “Dr. Bergen”

Introduction - You!

Survey – About you

- Mostly 3rd year (73%)
- Mostly use own computer (46% windows, 38% mac/linux)
  - But the lab machines are still available!
- Interests: games, piano, netflix, guitar, reading, biking, gardening, anime, baking, cars, cats, ...

Survey – why CSC324?

- To learn something new
- I really enjoy deconstructing and learning the fundamentals of things

Survey – why CSC324?

- To learn something new
- I really enjoy deconstructing and learning the fundamentals of things
- The course was recommended to me by an upper year student and I like to learn new things
- I hope that CSC324 will make me a better programmer
- Just for fun and for credits
- Honestly, I don't remember why I signed up for it

Survey – why CSC324?

- To learn something new
- I really enjoy deconstructing and learning the fundamentals of things
- The course was recommended to me by an upper year student and I like to learn new things
- I hope that CSC324 will make me a better programmer
- Just for fun and for credits
- Honestly, I don't remember why I signed up for it
- post requirement, pre-req for CSC384/CSC488

Survey – why CSC324?

- To learn something new
- I really enjoy deconstructing and learning the fundamentals of things
- The course was recommended to me by an upper year student and I like to learn new things
- I hope that CSC324 will make me a better programmer
- Just for fun and for credits
- Honestly, I don't remember why I signed up for it
- post requirement, pre-req for CSC384/CSC488
- I enjoyed learning how computers work at a low level in CSC209 and CSC258
- To learn SQL

Survey – Online learning concerns

- The existential dread that comes with the entirety of my academic, personal and social life coming out of the same 12x10 rectangle.

Survey – Online learning concerns

- The existential dread that comes with the entirety of my academic, personal and social life coming out of the same 12x10 rectangle.
- I'd prefer if lectures could be recorded and viewed later as well as being live.
- Accessibility (lectures, exams, course resources, etc.)
- Is a webcam necessary?

Survey – Online learning concerns

- The existential dread that comes with the entirety of my academic, personal and social life coming out of the same 12x10 rectangle.
- I'd prefer if lectures could be recorded and viewed later as well as being live.
- Accessibility (lectures, exams, course resources, etc.)
- Is a webcam necessary?
- technical problems occur during tests/exams

Survey – Online learning concerns

- The existential dread that comes with the entirety of my academic, personal and social life coming out of the same 12x10 rectangle.
- I'd prefer if lectures could be recorded and viewed later as well as being live.
- Accessibility (lectures, exams, course resources, etc.)
- Is a webcam necessary?
- technical problems occur during tests/exams
- I am just worried that I won't manage my time

Survey – Online learning concerns

- The existential dread that comes with the entirety of my academic, personal and social life coming out of the same 12x10 rectangle.
- I'd prefer if lectures could be recorded and viewed later as well as being live.
- Accessibility (lectures, exams, course resources, etc.)
- Is a webcam necessary?
- technical problems occur during tests/exams
- I am just worried that I won't manage my time
- We absolutely need to have some official platform for students to communicate, like Piazza...
- It would be really hard to get to know the professor and TA's

Survey – What you wanted Andi/Lisa to know

- I am looking forward to a fun semester :)
- Please no ProctorU...
- I need an accessibilities accomodation...
- Will the exam be online as the rest of the semester?
- I would like to know what OS will be the easiest to work with for this course since I can choose between Windows or Linux.
- Can we join whichever lecture / tutorial section we want or do we have to stay in our assigned slots?

Type in the chat...

What programming languages have you learned so far?

Type in the chat...

What programming languages have you learned so far?

```
# Python
x = 5       # programs consists of *statements*
y = f(x)    # that manipulates the values of variables
y += 1      # (or manipulating *states*)
```

Type in the chat...

What programming languages have you learned so far?

```
# Python
x = 5       # programs consists of *statements*
y = f(x)    # that manipulates the values of variables
y += 1      # (or manipulating *states*)
```

The languages you learned so far probably follow the **imperative programming** paradigm

There are other paradigms

Functional Programming

- Express computation using *expressions* that evaluates to a value
- No manipulatable states, no mutable variables!
- Programs are easier to reason about (e.g. prove correctness, write interpreter)

There are other paradigms

Functional Programming

- Express computation using *expressions* that evaluates to a value
- No manipulatable states, no mutable variables!
- Programs are easier to reason about (e.g. prove correctness, write interpreter)

Logic Programming

- Express computation using *constraints* that we wish to satisfy
- Useful for finding answers in a large search space
- Natural to express problems like sudoku, automatic programming by example

What we will do in CSC324 (Surface Level)

We will learn the programming languages **Racket** and **Haskell**, and create a few small languages of our own!

Q: What is a programming language?

Q: Are these programming languages?

1. Program takes two numbers as input, and adds them.

Q: Are these programming languages?

1. Program takes two numbers as input, and adds them.
2. Program takes two numbers and a binary operation (+, -, etc. . .) as input, and applies the operation to the numbers.

Q: Are these programming languages?

1. Program takes two numbers as input, and adds them.
2. Program takes two numbers and a binary operation (+, -, etc. . .) as input, and applies the operation to the numbers.
3. Same as #2, but allow users to save intermediate computation (e.g. as variables).

Q: Are these programming languages?

1. Program takes two numbers as input, and adds them.
2. Program takes two numbers and a binary operation (+, -, etc. . .) as input, and applies the operation to the numbers.
3. Same as #2, but allow users to save intermediate computation (e.g. as variables).
4. Same as #3, but allow users to define and use functions.

Q: Are these programming languages?

1. Program takes two numbers as input, and adds them.
2. Program takes two numbers and a binary operation (+, -, etc. . .) as input, and applies the operation to the numbers.
3. Same as #2, but allow users to save intermediate computation (e.g. as variables).
4. Same as #3, but allow users to define and use functions.
5. Same as #4, but the program is a web application with a drag-and-drop user interface.

Greenspun's tenth rule of programming

Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.

What we will do in CSC324 (More deeply)

We will define/analyze/modify the **syntactic** and **semantic** features of programming languages, and create small languages of our own.

Ideal Result: Recognize when a programming problem is a "programming languages" problem.

. . . and along the way, expand your definition of programming.

Course Logistics

Course software

- Racket
  - We recommend the IDE **DrRacket**
- GHC Haskell
  - No particular IDE for Haskell (try Visual Studio Code)
- MarkUs for assignment submission
- Quercus for quizzes/tests/course website
- Piazza for discussion board
  - Use Piazza for course content related questions
  - Use email (Lisa) for personal/logistics questions

Course Components: readings, quizzes, lectures

Posted Readings: Complete before lecture

Weekly Quizzes (5%): Quercus Quizzes, ideally attempt once before lecture

- Officially due Wednesday 10pm, up to 5 attempts
- No late quizzes will be accepted

Lectures: Synchronous (live) lectures on BbCollaborate, will be recorded

Course Components: exercises and projects

Lab Exercises (28%): Weekly lab exercises (9 total)

- TAs will help you get started during the labs on BbCollaborate
- Lab exercise due most Saturdays 10pm
- First exercise does not count towards your grade

Projects (27%): Two projects, can work with a partner

- Project 1 in Haskell (week 5), Project 2 in Racket (week 12)
- The entire week will be devoted to the projects, and the lectures will become OH

Token system for late penalty for Exercise/projects:

- Each student gets 10 tokens, each worth 12 hours
- Max 2 tokens can be used per exercise, 4 tokens per project
- Both partners must have tokens to submit a project late

Course Components: tests and exam

Tests (20% or 25%): Four tests during your labs, each 30 minutes

- Weeks 3, 6, 8 and 10
- If you miss a test, the weight will be shifted to the final exam
- If you miss a second test, you will take an oral makeup for the second test

Final Exam (15% or 20%): Online TBD

We'll take the weighting (20% + 20% or 25% + 15%) that gives you a better grade.

Course Policy on Accessibility

Lisa and Andi care about making this course accessible!

- We'll post written notes/exercises early so you can get ahead in anticipation of any issues
- We're happy to provide resources in alternative formats upon request (e.g. raw text instead of pdf)
- There will be some contact hours every day, and we're happy to meet privately upon request

Additional Requests for Extensions

The token system should be useful for most issues.

If you are in a situation that requires additional accommodations, let Lisa know by email as soon as possible and *24 hours before a deadline*.

However, unless there is an unanticipated emergency (e.g. you're stuck in an elevator), we won't make accommodations on the day of the deadline.

Academic Integrity: Is it acceptable to...

- Discuss lab exercise solutions with a classmate?

Academic Integrity: Is it acceptable to...

- Discuss lab exercise solutions with a classmate?
- Discuss lab exercise solutions with classmates during office hours?

Academic Integrity: Is it acceptable to...

- Discuss lab exercise solutions with a classmate?
- Discuss lab exercise solutions with classmates during office hours?
- Have someone else take your test for you?

Academic Integrity: Is it acceptable to...

- Discuss lab exercise solutions with a classmate?
- Discuss lab exercise solutions with classmates during office hours?
- Have someone else take your test for you?
- Ask questions about lab exercises on Piazza?

Academic Integrity: Is it acceptable to...

- Discuss lab exercise solutions with a classmate?
- Discuss lab exercise solutions with classmates during office hours?
- Have someone else take your test for you?
- Ask questions about lab exercises on Piazza?
- Post partial solutions to lab exercises on Piazza?

Academic Integrity: Is it acceptable to...

- Discuss lab exercise solutions with a classmate?
- Discuss lab exercise solutions with classmates during office hours?
- Have someone else take your test for you?
- Ask questions about lab exercises on Piazza?
- Post partial solutions to lab exercises on Piazza?
- Work with someone else on a test?

Academic Integrity: Is it acceptable to...

- Discuss lab exercise solutions with a classmate?
- Discuss lab exercise solutions with classmates during office hours?
- Have someone else take your test for you?
- Ask questions about lab exercises on Piazza?
- Post partial solutions to lab exercises on Piazza?
- Work with someone else on a test?
- Discuss your part of the project with your partner?

Academic Integrity: Is it acceptable to...

- Discuss lab exercise solutions with a classmate?
- Discuss lab exercise solutions with classmates during office hours?
- Have someone else take your test for you?
- Ask questions about lab exercises on Piazza?
- Post partial solutions to lab exercises on Piazza?
- Work with someone else on a test?
- Discuss your part of the project with your partner?
- Discuss your part of the project with someone who isn't your partner?

Academic Integrity: Is it acceptable to...

- Discuss lab exercise solutions with a classmate?
- Discuss lab exercise solutions with classmates during office hours?
- Have someone else take your test for you?
- Ask questions about lab exercises on Piazza?
- Post partial solutions to lab exercises on Piazza?
- Work with someone else on a test?
- Discuss your part of the project with your partner?
- Discuss your part of the project with someone who isn't your partner?
- Work with a classmate on the weekly quizzes? (not the tests)

Academic Integrity Summary

- Do not discuss exercise solutions with classmates, except during office hours and on Piazza.
- Do not discuss project with classmates other than your partner
- Do not post full/partial solutions to exercise/projects/tests on Piazza.
- Please don't "help" others by helping them solve the homework!
- You **can** work with other students on the weekly quizzes (but NOT the tests!)

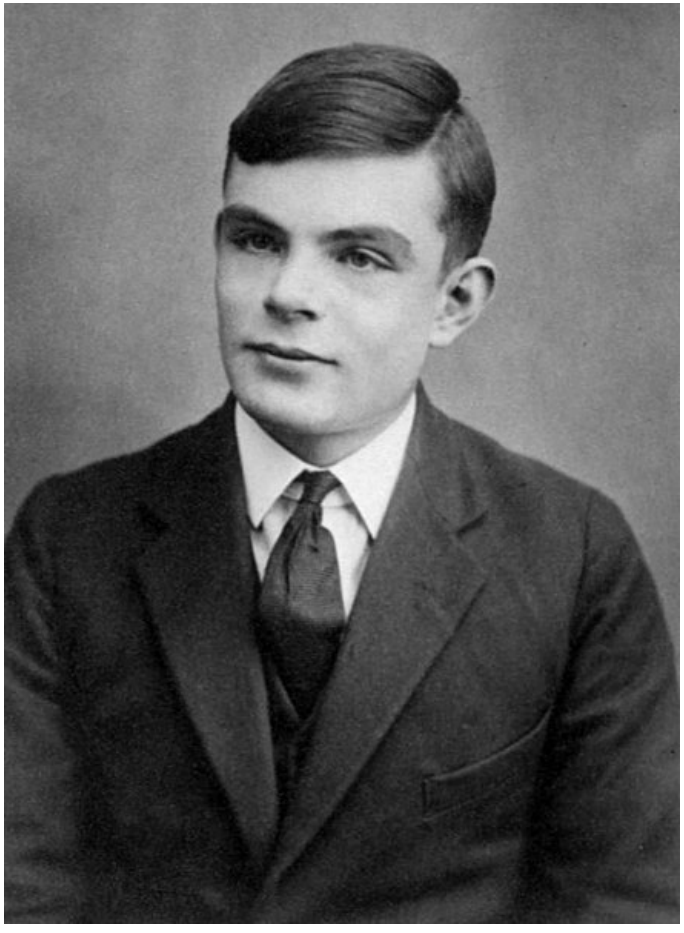
Please don't put others in awkward positions by asking them for help on an exercise or assignment. What you can do to help:

- Discuss examples from lecture and the course materials
- Practice using sample problems
- Ask questions on the discussion board

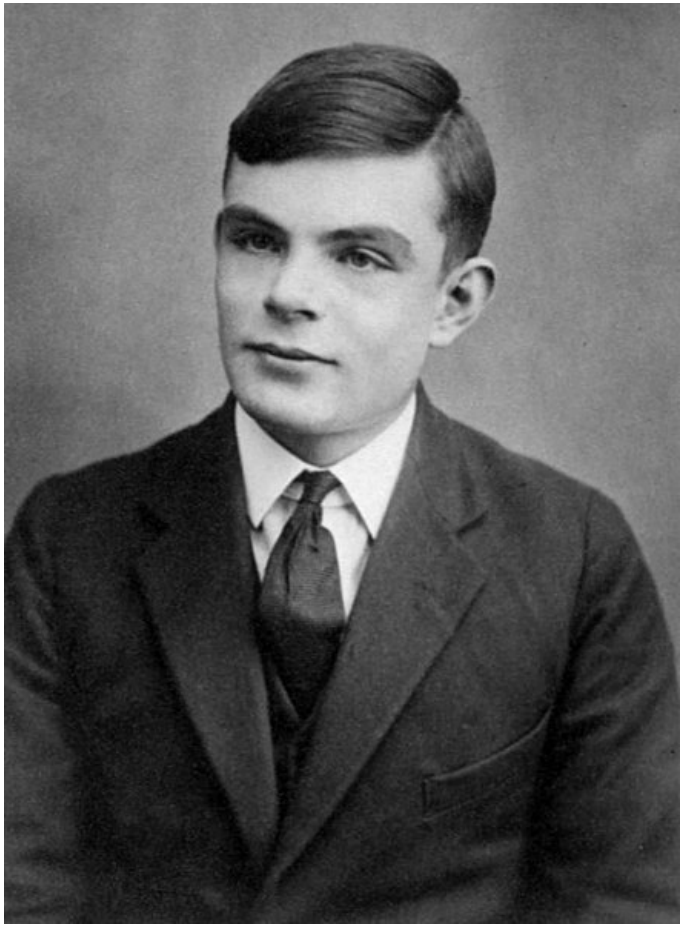
Questions?

The Lambda Calculus

Who are these computer scientists?



Who are these computer scientists?



Alan Turing (left), Alonzo Church (right)

Two models of computation

The **Turing machine** performs computation by

- executing instructions to modify internal memory

The **Lambda-Calculus** performs computation by

- evaluating expressions

The Lambda-Calculus

A program in the lambda-calculus is a single expression (no statements!)

Only three kinds of expressions in the lambda-calculus:

1. Identifier; e.g. x
2. Function expression; e.g. $\lambda x \mapsto x$ (identity function)
3. Function application; e.g. $f\ y$ (applies f with argument y)

"Running" such a program means to evaluate the expression via substitution

Church Turing Thesis

The Turing Machine and the Lambda-Calculus are have the same computational power: any computation that you can perform/express in one model can be performed using the other model.

Church Turing Thesis

The Turing Machine and the Lambda-Calculus are have the same computational power: any computation that you can perform/express in one model can be performed using the other model.

Church-Turing Thesis: any other reasonable computational model is equivalent to the Turing machine and lambda-calculus.

Church Turing Thesis

The Turing Machine and the Lambda-Calculus are have the same computational power: any computation that you can perform/express in one model can be performed using the other model.

Church-Turing Thesis: any other reasonable computational model is equivalent to the Turing machine and lambda-calculus.

Before we can talk more about the lambda calculus, let's define how we will specify a programming language.

Syntax and Semantics

Specifying a Programming Language

Syntax: the structure of valid programs in a language

Semantics: the meaning of the elements of a language

Grammar

Grammar: formal set of rules specifying the syntax of a language

Example: grammar for arthmetic expressions (infix notation)

```
<expr> = NUMBER
       | '(' <expr> <op> <expr> ')'

<op>   = '+' | '-' | '*' | '/'
```

An expression is syntactically valid iff it can be *generated* by the language's grammar

Example 1: Arithmetic

Assuming the following grammar:

```
<expr> = NUMBER
       | '(' <expr> <op> <expr> ')'

<op>   = '+' | '-' | '*' | '/'
```

Q. Are these expressions syntactically valid?

1. `3`
2. `(3 + (1 + 2))`
3. `(3 + 1 + 2)`
4. `3 + 1`
5. `(/ 1 0)`
6. `(1 / 0)`

Type in chat (e.g. Y/Y/N/Y/N/Y)

Example 2: List of numbers

Assuming the following grammar:

```
<list> = null
       | '(' 'cons' NUMBER <list> ')'
```

Q. Are these expressions syntactically valid?

1. `null`
2. `(cons 3 null)`
3. `(cons 3 (cons 4 null))`
4. `(cons 3 (cons 4 5))`
5. `(cons (cons 4 null) (cons 4 null))`
6. `(cons null null)`

Example 3: Lambda-Calculus Grammar in Racket Syntax

Assuming the following grammar:

```
<expr> = ID
       | '(' 'lambda' '(' ID ')' <expr> ')'
       | '( <expr> <expr> ')'
```

Q. Are these expressions syntactically valid?

1. `x`
2. `(lambda (x) x)`
3. `lambda x x`
4. `(lambda (x y) x)`
5. `((lambda (x) (lambda (y) x)))`
6. `((lambda (x) (x x)) y)`

Semantics

Semantics: the meaning of a program, or how to evalute a program

There are several ways to think about semantics:

- **Denotational Semantics**: the abstract value of a program
- **Operational Semantics**: the evaluation steps used to determine the value of a program

Example: In the language Python,

- Do the expressions `1 + 1` and `2` have the same *denotational semantics*?

Semantics

Semantics: the meaning of a program, or how to evalute a program

There are several ways to think about semantics:

- **Denotational Semantics**: the abstract value of a program
- **Operational Semantics**: the evaluation steps used to determine the value of a program

Example: In the language Python,

- Do the expressions `1 + 1` and `2` have the same *denotational semantics*?
- Do the expressions `1 + 1` and `2` have the same *operational semantics*?

Lambda-Calculus Semantics

A program in the lambda-calculus is a single **expression** (no **statements**!)

"Running" such a program means to **evaluate** the expression

Evaluation by Substitution:

- Identifiers and function expressions are already fully-evaluated
- Function applications are evaluated by substituting the argument for the parameter in the body of the function

$$(\lambda x \mapsto x)hi$$

Evaluated by *substituting hi* for x in the body of the function, giving *hi* as the result.

How powerful is the lambda calculus?

Recall Lambda-Calculus Grammar in Racket

```
<expr> = ID
       | (lambda (ID) <expr>)
       | (<expr> <expr>)
```

Demo: the actual code is beyond the scope of this course

How powerful is the lambda-calculus?

Only using functions and identifiers appears restrictive

But, it turns out that . . .

- We can encode numbers, strings, and other data structures
- We can recover **recursion**
- We can perform any computation that you can in any other programming language

Racket/Haskell

These are much "bigger" language than the lambda-calculus (more language features)

Both are functional programming languages

We won't teach you the syntax of Racket and Haskell. It's up to you to slowly pick them up using the provided notes + resources.

. . . but let's look a little at Racket syntax.

Function Expressions (aka anonymous function definition)

Racket: `(lambda (<param> ...) <body>)`

- `(lambda (x)   (/ (+ x 2) (- x 2)))`
- `(lambda (x)   (equal? x 2))`
- `(lambda (x y) #f)`
- `(lambda (x y) (not (equal? x y)))`

Haskell: `\<param> ... -> <body>`

- `\x -> (2 + x) / (2 - x)`
- `\x -> x == 2`
- `\x y -> False`
- `\x y -> x /= y`

Function Applications

Racket: `(<function> <args> ...)`

- `((lambda (x) (+ x 1)) 5)` – brackets are important!
- `((lambda (x y) (equal? x y)) 0 1)`
- `((lambda (x y) (= x y)) 0 #f)` – throws an error

Haskell: `<function> <args> ...`

- `(\x -> x + 1) 5` – brackets are for order or operation
- `(\x y -> x == y) 0 1`
- `((\x y -> x == y) 0 1)`

Name Bindings (aka variables!)

Binding identifiers to values gives us two conveniences:

1. Saving the value of subexpressions.
2. Recursively referring to a function name.

Global Name Bindings

Racket: `(define <id> <expr>)`

- `(define x 5)`
- `(define add1 (lambda (x) (+ x 1)))`
- `(define (add1 x) (+ x 1))` – alternative syntax for functions

Haskell: `<id> = <expr>`

- `x = 5`
- `add1 x = x + 1`

Local Name Bindings

Racket: `(let* ([<id> <expr>] ...) <body>)`

```
(define (f x y)
  (let* ([e1 (+ x 1)]
         [e2 (- y 2)])
        (* e1 e2)))
```

(Note: There are subtle differences between `let`, `let*` and `letrec`)

Haskell:

```
let <id> = <expr>
    ...
in
    <body>
```

Pure Functions

A **pure** function is one whose behaviour is solely determined by the *values* of its parameters, and that returns a value and does nothing else.

- No manipulation of global variables or states
- No reading / writing of files
- No reading / writing to database
- No randomness

Referential Transparency

Identifier is **referentially transparent** if it can be substituted by its value without changing the meaning of the program.

For this reason, **an identifier cannot be bound more than once**!

No mutation!

Racket Symbols

A **symbol** is a quoted name

Example:

- `'a`
- `'name`
- `'symbol`

Symbols are **not strings**, and *cannot* be decomposed into characters!

Lists in Racket

In Racket, a list is really a **linked list**, with a first-node and a reference to the rest of the list.

Example:

- `null` or `'()` is the empty list
- `(cons 1 null)` is the list containing 1
- `(cons 1 (cons 2 (cons 3 null)))` is the list containing 1, 2, 3
- `(list 1 2 3)` is also the list containing 1, 2, 3
- `(list 'a 2 #f)` is the list containing `'a`, 2, `#f` – list elements can be different types

(Similar in Haskell)

List Quoting

- `'(a b c)` is the list containing the symbols `'a`, `'b`, and `'c`
- `'(1 2 3)` is the list containing the numbers 1, 2, 3
- `'((a b) (1 2 3))` is a two element list, where:
  - the first list contains the symbols `'a`, `'b`
  - the second list contains the numbers 1, 2, 3
- `'()` is the empty list

List Unpacking

```
> (define lst (list 1 2 3))
> (first lst)
1
> (rest lst)
'(2 3)
> (car lst) ; like `first`
1
> (cdr lst) ; like `rest`
'(2 3)
```

Recursion on Lists

We can operate on the first element of a list, and then recursively on the rest of the list

```
(define (add1-list lst)
  (if (null? lst)
    (list)
    (cons (+ 1 (first lst))
          (add1-list (rest lst)))))
```

This is an example of **structural recursion**.

What to do next

1. Review week 1 notes
2. Complete week 1 quiz; ask questions on Piazza
3. Install Racket & Haskell, or use the lab machine
4. Attend the labs this week (optional)
5. Attend office hours if you have any questions, or just to say hi!
6. Complete and submit exercise 1
7. Read week 2 notes before class next week
8. Attempt week 2 quiz once