CSCI 1103 Computer Science 1 Honors

CSCI 2103 Functional Programming

Fall 2022

Robert Muller

Boston College

Home

https://github.com/BC-CSCI1103/f22

Today

- What this course is about
- Course administration
- Next time: setup & logistics

Teaching Assistant Staff

Liam Murphy
Head Teaching Assistant

OH: Thu 2PM-4PM

Nikki Lockwood

OH: Mondays 2PM - 3PM,
Wednesdays 5PM - 7PM,
Fridays 2PM - 3PM

CSCI 2103 Functional Programming

- Some problem sets more involved than 1103 problem sets: Binary Search Trees, Balanced Search Trees, Parsing
- Some additional material on λ-calculus & other functional programming languages
- Labs are optional
- Path Options --- must let staff know at end of 2nd week

2103 Path Options

- Follow 1103 with several alternate psets and additional readings/videos
- Advanced Data Structures, Prefix Trees, Hash Array Map Tries (HAMTs), etc
- Theory – deeper study of both untyped λ-calculus and various typed λ-calculi

Cousins

- CS 3110 Data Structures and Functional Programming @ Cornell

- CS 51 Abstraction and Design in Computation @ Harvard
- 15-150 Functional Programming @ CMU
- COS 226 Functional Programming @ Princeton
- CIS 1200 Programming Languages and Techniques @ UPenn

What CSCI 1103 is About

Three interwoven themes:

1. Learning about information & computation
2. Developing an important **skill**: how to **code**
3. An introduction and gateway to computer science

Learning how to code

- Application of logic in problem solving (*math-ish*)

- *Clear, concise, beautiful expression of ideas/algorithms (english/poetry-ish)*

Learning how to code

- Have an idea? **You** can build it!
- Empowering in almost any field
- Interesting and really fun
- Learn by doing

Required Work

- Two 75-minute meetings each week 12PM; Some will be lectures, some may be workshop-style
- One 50-minute lab each week, taught by TAs
- Ten programming projects, time requires varies but expect 8-10 hours of work each week
- Two midterms and a final exam

Take-Aways

- By the end of the semester:
  - You'll have a reasonably robust understanding of computation;
  - You'll be **skilled**, able to "think computationally" able to develop code;
  - You'll have a better understanding of computer science as a field and prospective career path.

Take-Aways

- By the end of the semester:
  - You'll be able to pick up other languages such as Python or Java easily;
  - You'll be very well-prepared for CS2 and the rest of the CS curriculum.

Required Background

- High School algebra
- Familiarity with basic trigonometry and geometry also helpful.
- No programming experience required.
- A taste for building things also helpful.

Learned to code in HS?

- Java, Python, C, C++ … : Primacy of commands, expressions are secondary

- OCaml & FP: Primacy of expressions, commands are secondary

Computation and Calculation

Three Aspects of Computation

1. Simplification
2. Abstraction & Composition
3. Abstraction & Representation

$$434 + 58 = 492$$

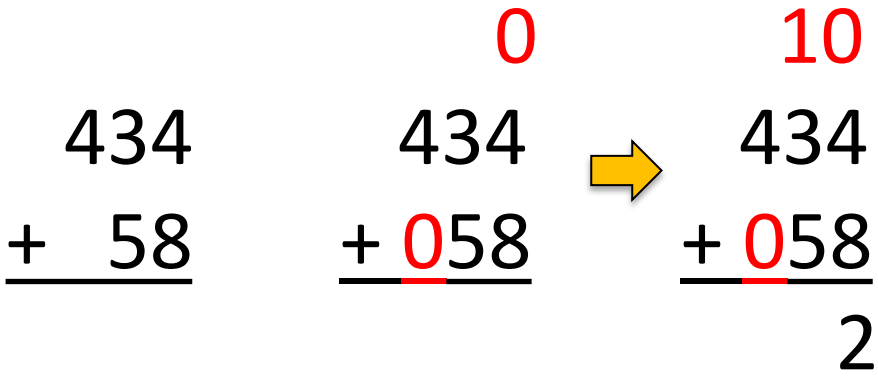
positional addition *al gorithm*

```
  434
+  58
-----
```

positional addition *al gorithm*

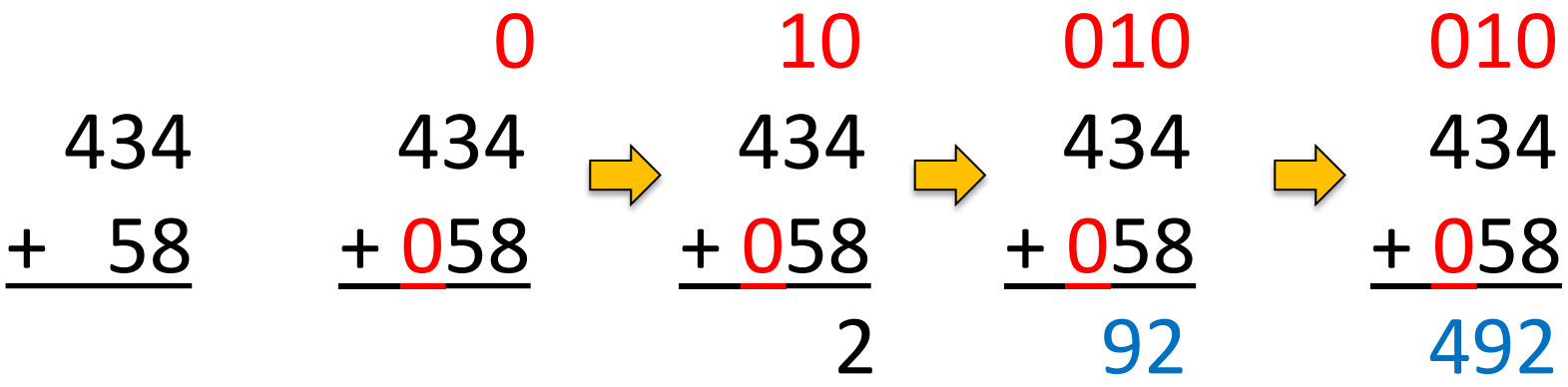
```
  434      434
+  58    + 058
-----    -----
```

positional addition *al gorithm*



positional addition *al gorithm*

For the given input data, 434 and 58, the addition algorithm requires 3 units of work



Units of Work

- Common to take addition, subtraction, comparison, etc as atomic units of work even though some operations are more costly than others

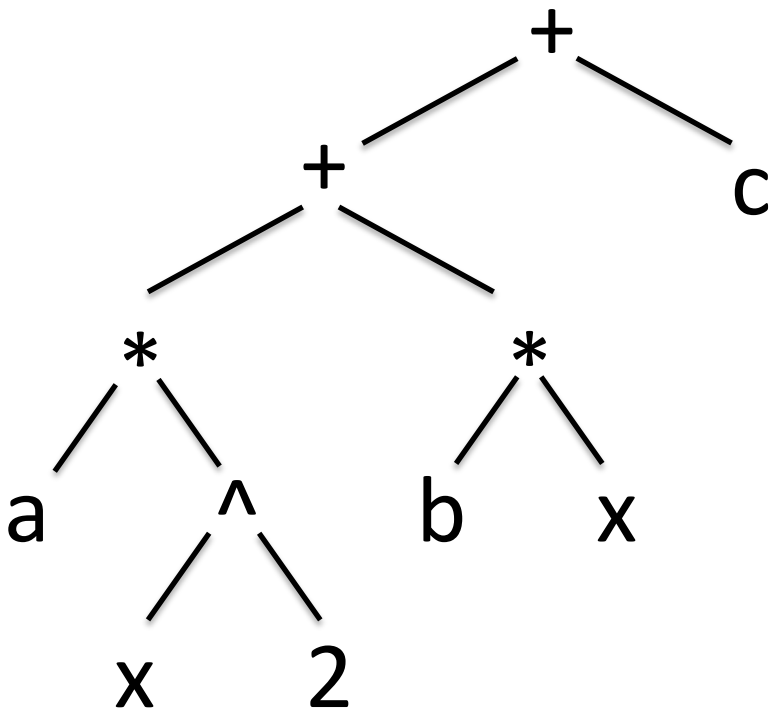
Simplification

In middle school we learned about algebraic expressions:

$$ax^2 + bx + c$$

Where a, b and c are **constants** and x is a **variable**. We learned to solve for roots, how to factor them, we learned properties of their curves, etc.

$ax^2 + bx + c$



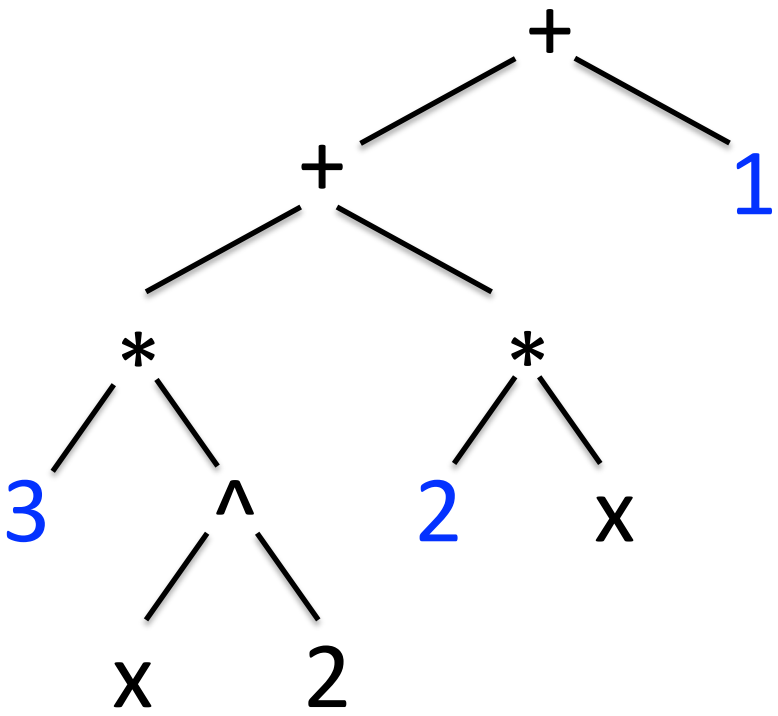
Simplification

For example, plugging in constants 3 for a, 2 for b and 1 for c we get:

$$3x^2 + 2x + 1$$

Three fixed constants and one variable x.

$3x^2 + 2x + 1$

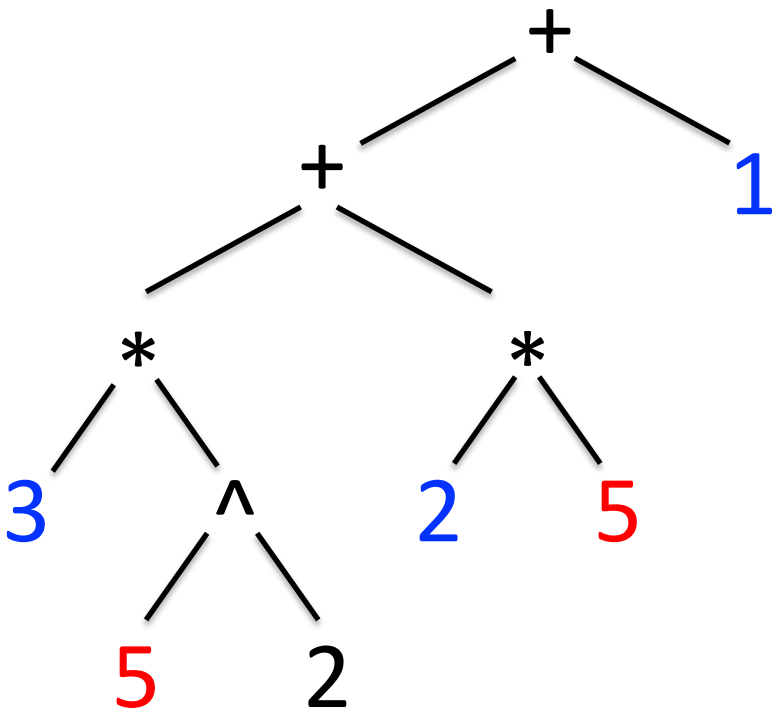


Simplification

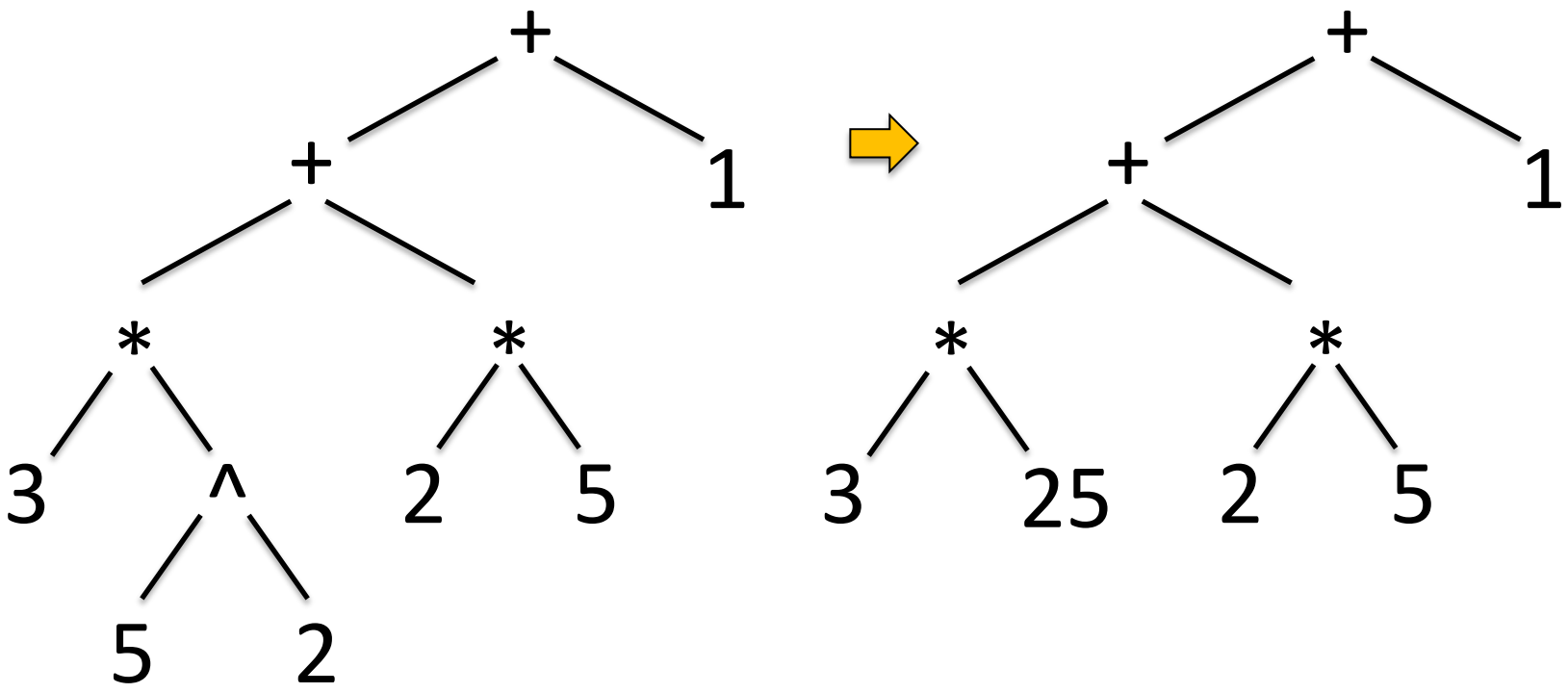
We can plug a number in for variable x and simplify. Say 5:

$$3 \bullet 5^2 + 2 \bullet 5 + 1$$

$3*5^2 + 2*5 + 1$



$3*5^2 + 2*5 + 1$



Simplification

$3 \bullet 5^2 + 2 \bullet 5 + 1$

➔ $3 \bullet 25 + 2 \bullet 5 + 1$

➔ $75 + 2 \bullet 5 + 1$

➔ $75 + 10 + 1$

➔ $85 + 1$

➔ 86

Simplification

$3 \bullet 5^2 + 2 \bullet 5 + 1$

➔ $3 \bullet 25 + 2 \bullet 5 + 1$

➔ $75 + 2 \bullet 5 + 1$

➔ $75 + 10 + 1$

➔ $85 + 1$

➔ 86 ⟵ A <u>value</u>

Simplification

$3 \bullet 5^2 + 2 \bullet 5 + 1$

$\rightarrow 3 \bullet 25 + 2 \bullet 5 + 1$

$\rightarrow 75 + 2 \bullet 5 + 1$

$\rightarrow 75 + 10 + 1$

$\rightarrow 85 + 1$

$\rightarrow 86$

5 units of work in 5 steps

Parallel Simplification

$3 \bullet 5^2 + 2 \bullet 5 + 1$

5 units of work In **3** steps

→ $3 \bullet 25 + 10 + 1$

→ $75 + 11$

→ 86

Abstraction

Algebraic expressions packaged up as *functions*:

$$f(x) = 3x^2 + 2x + 1$$

We can take this as a *definition* of function f.

Function Definitions and Uses

Euler's notation for **uses**, **calls** or **applications** of function f:

f(5) f(2 + 2)

1. Simplify the argument to value V,
2. plug the value V in for x,
3. simplify the result.

Simplification

f(2+2) ➔ f(4)

➔ $3 \bullet 4^2 + 2 \bullet 4 + 1$

➔ $3 \bullet 16 + 2 \bullet 4 + 1$

➔ $48 + 2 \bullet 4 + 1$

➔ $48 + 8 + 1$

➔ $56 + 1$

➔ 57

Functions and Code

- Roughly speaking, a software application is a collection of functions.
- In HS algebra our functions usually worked with real numbers.
- In programming, there are lots and lots of interesting **types** of inputs for our functions.

Code

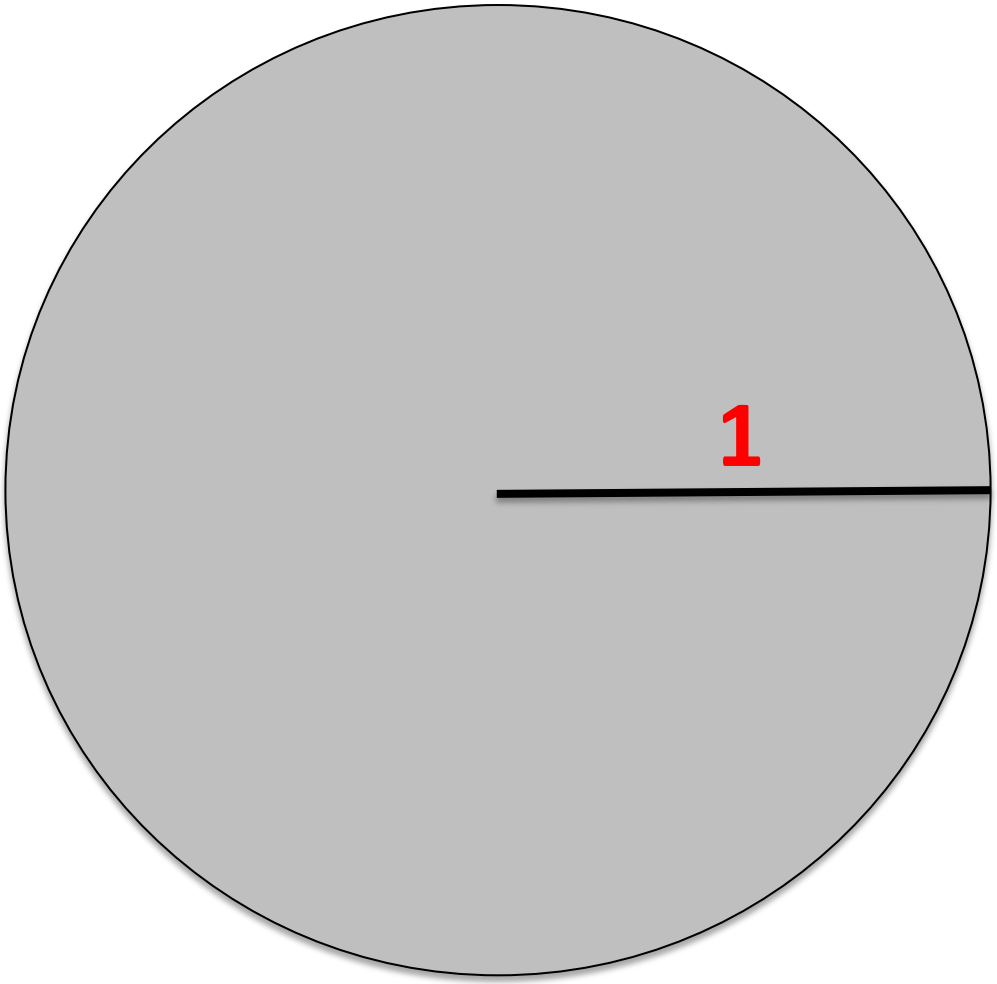
OCaml:

```
let f x = 3 * x ** 2 + b * x + c
```

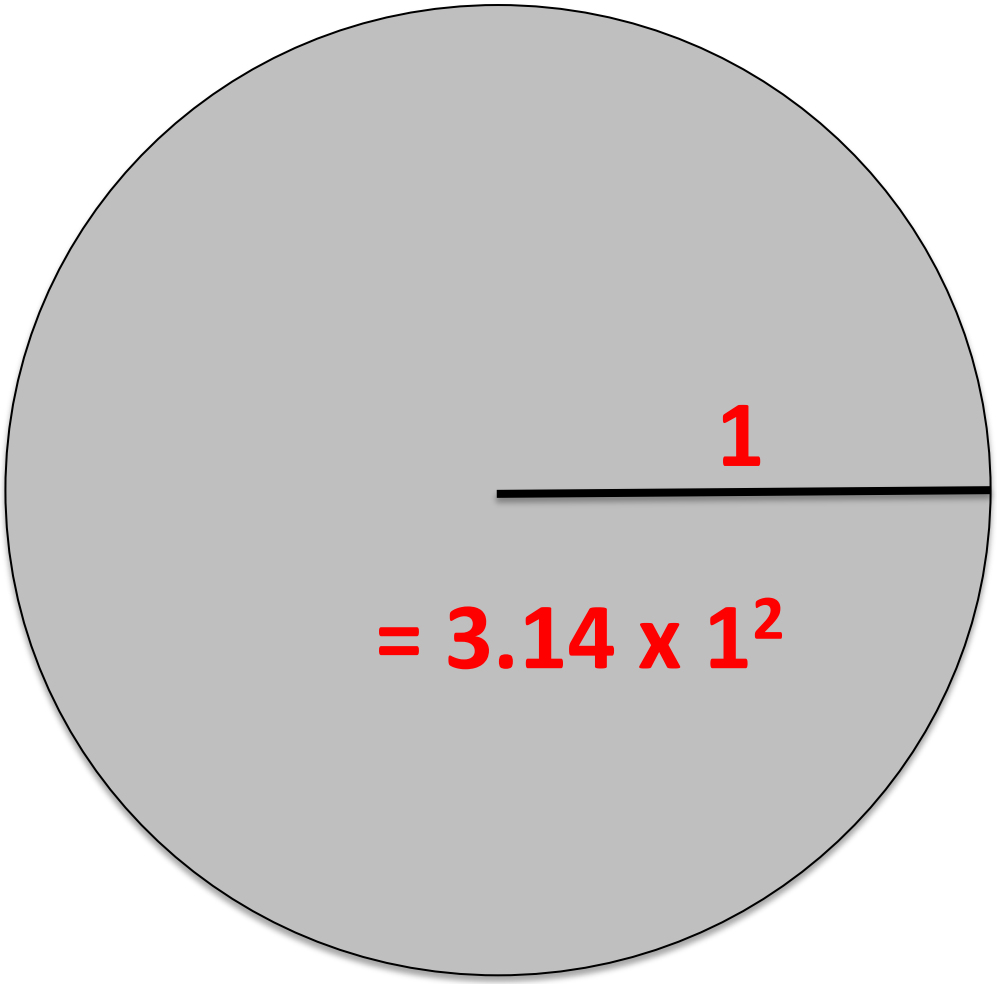
Python:

```
def f(x):
    return 3 * x ** 2 + b * x + c
```

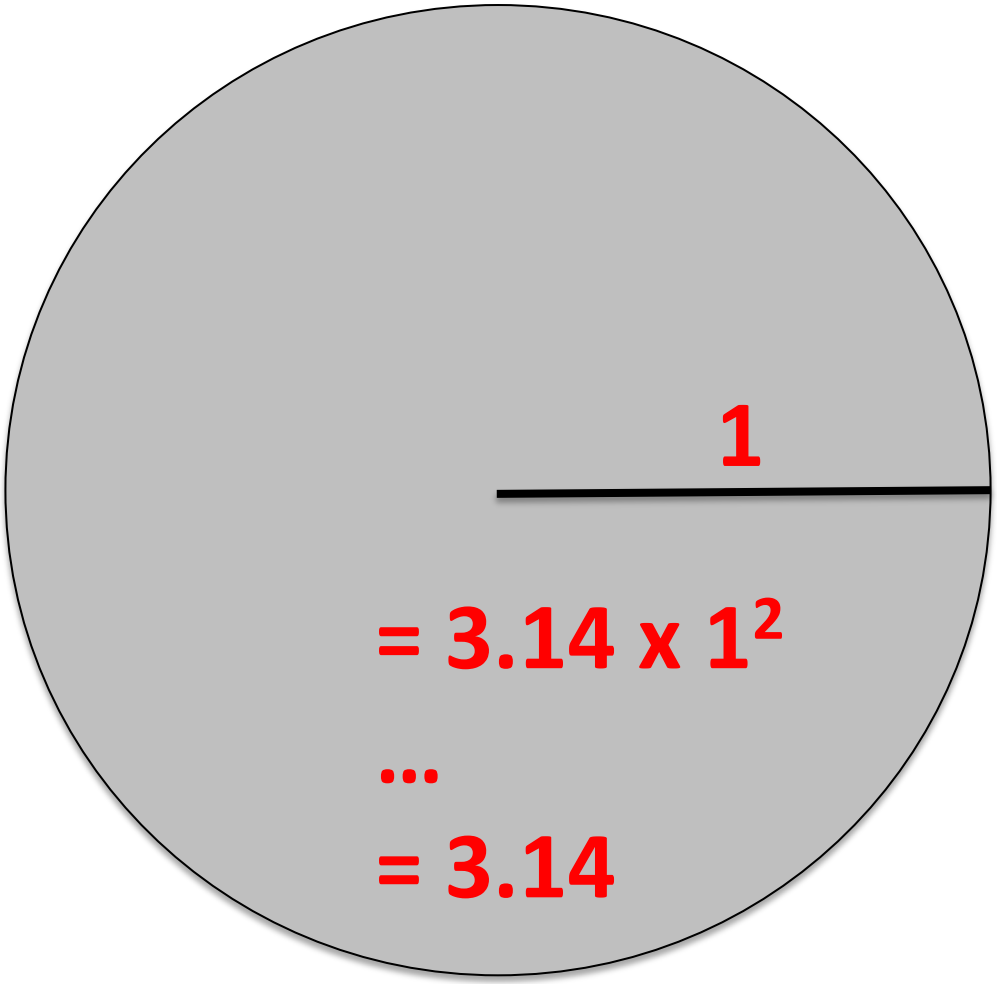
Example: area of unit circle



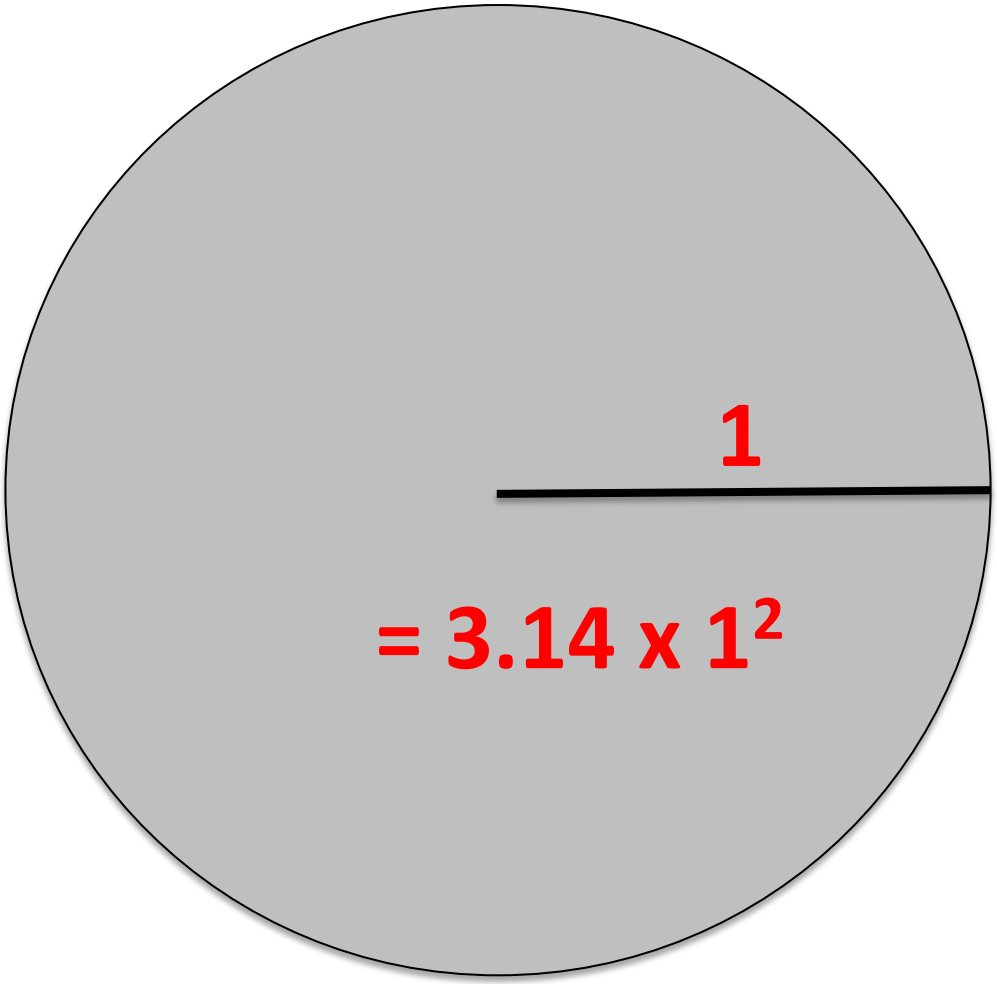
Example: area of unit circle



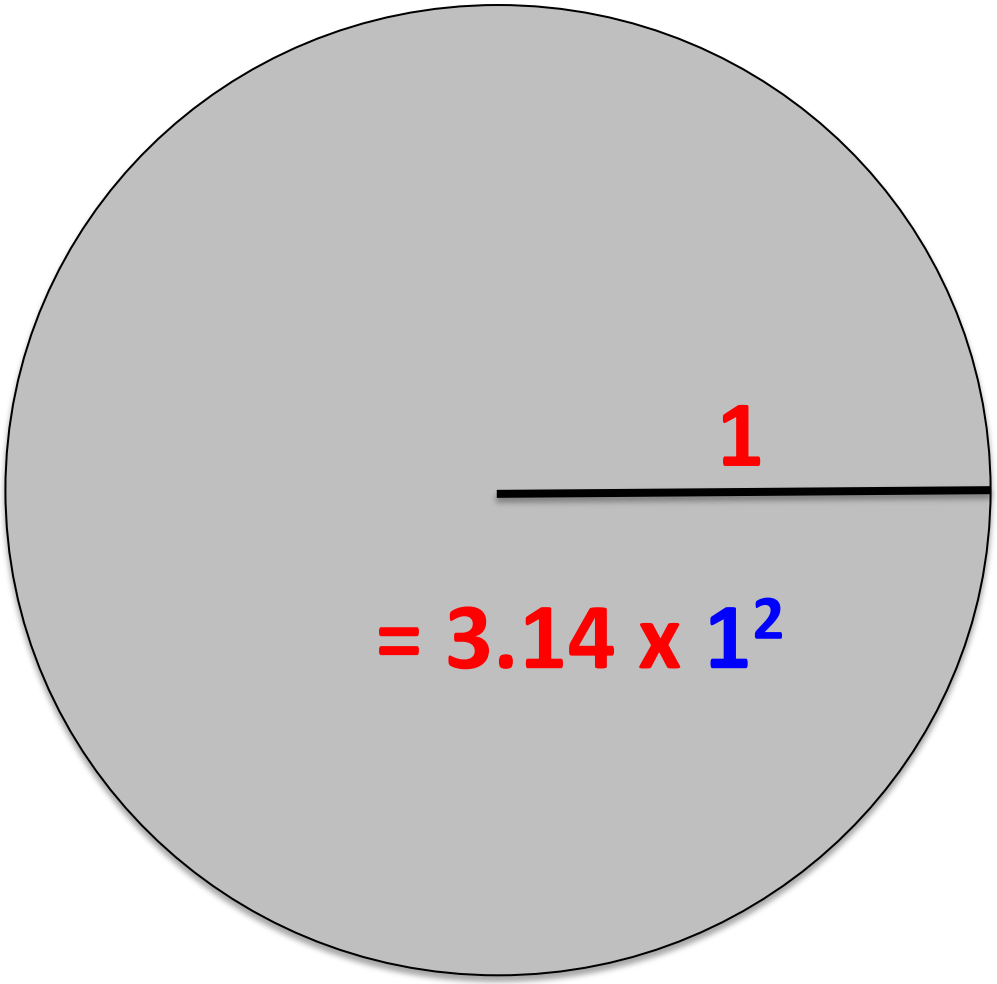
Example: area of unit circle



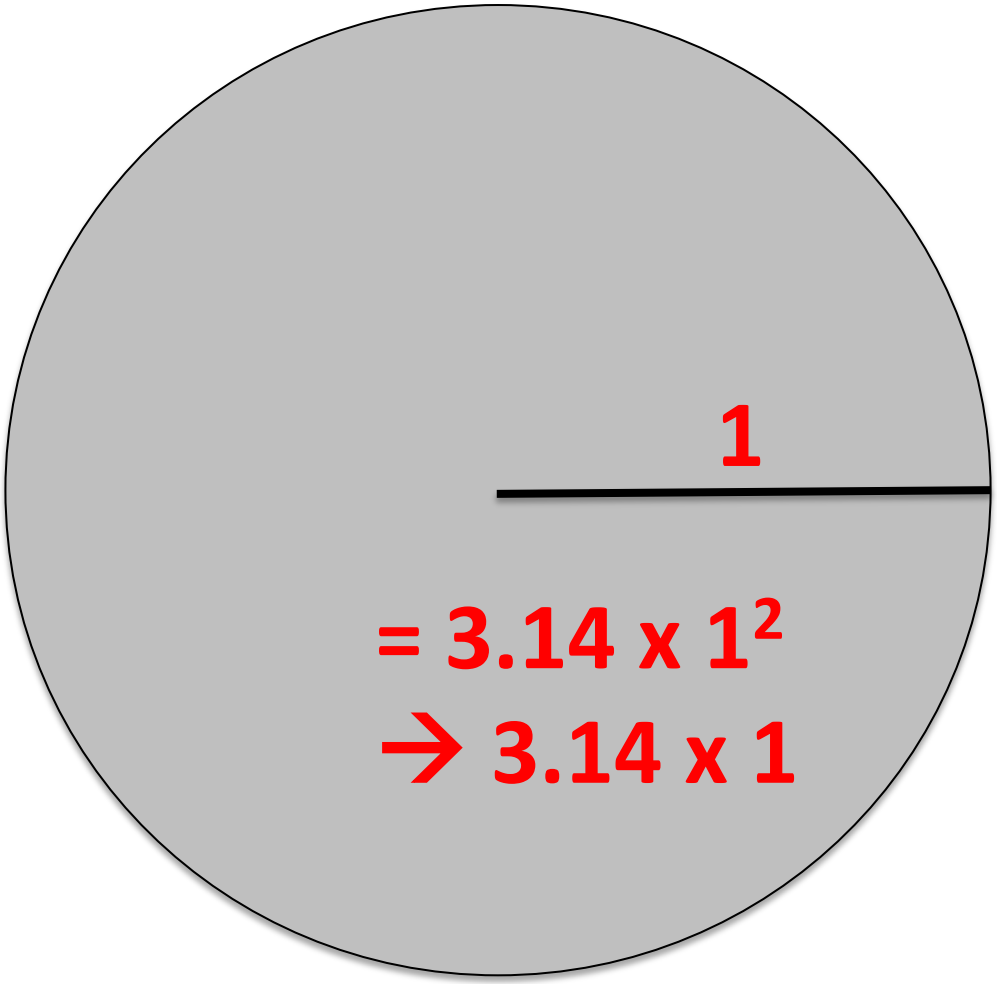
Example: area of unit circle



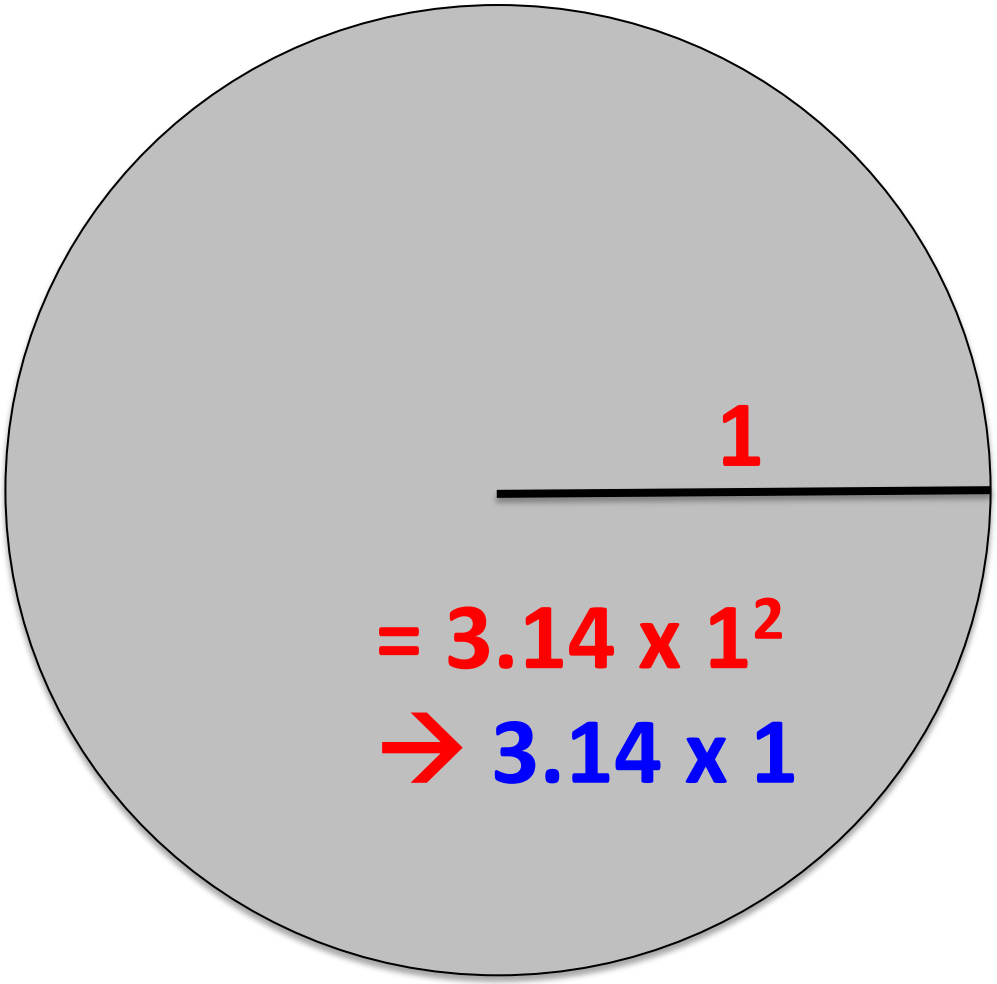
Example: area of unit circle



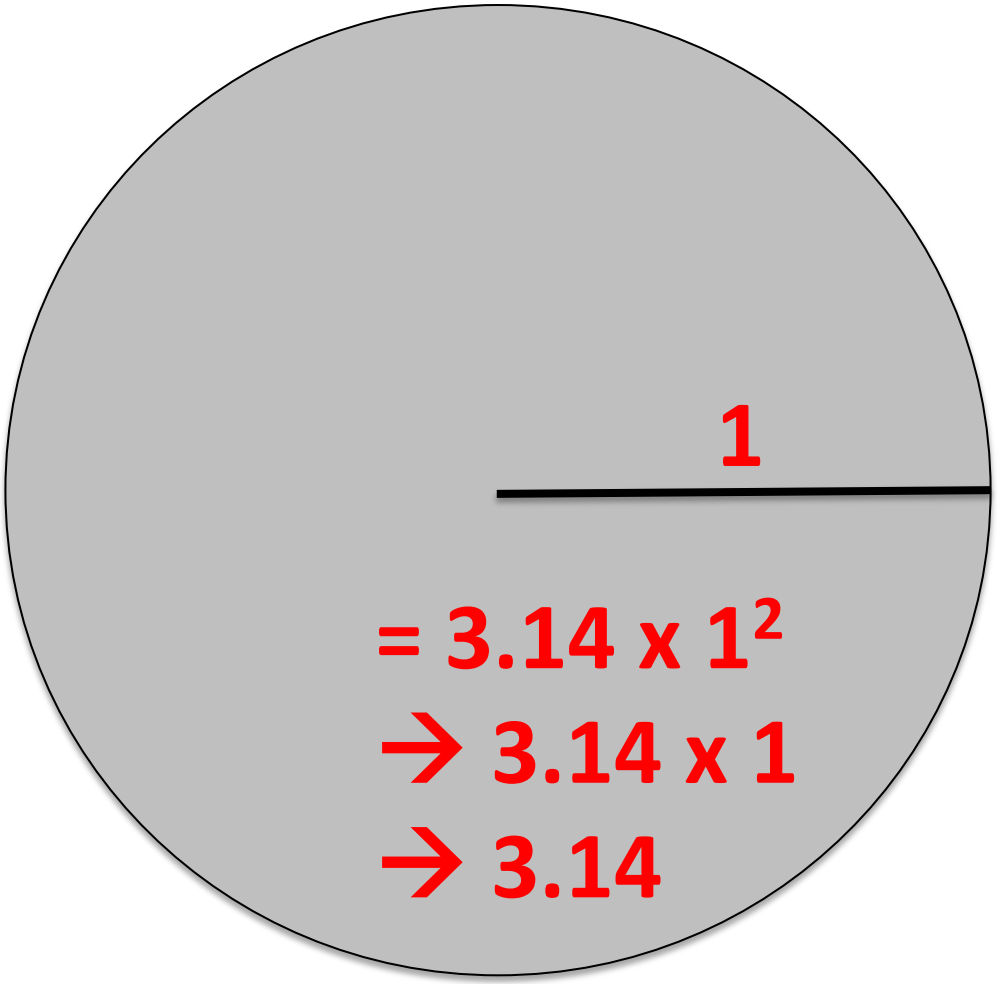
Example: area of unit circle



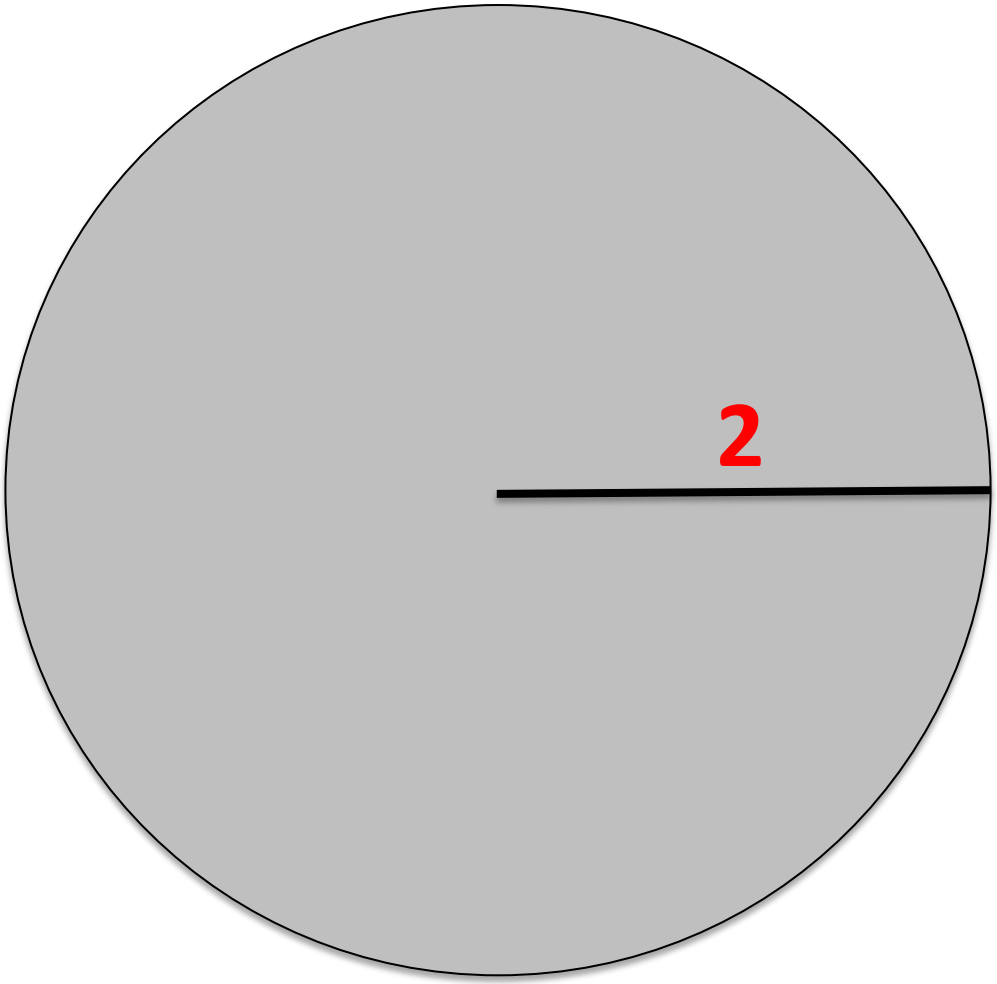
Example: area of unit circle



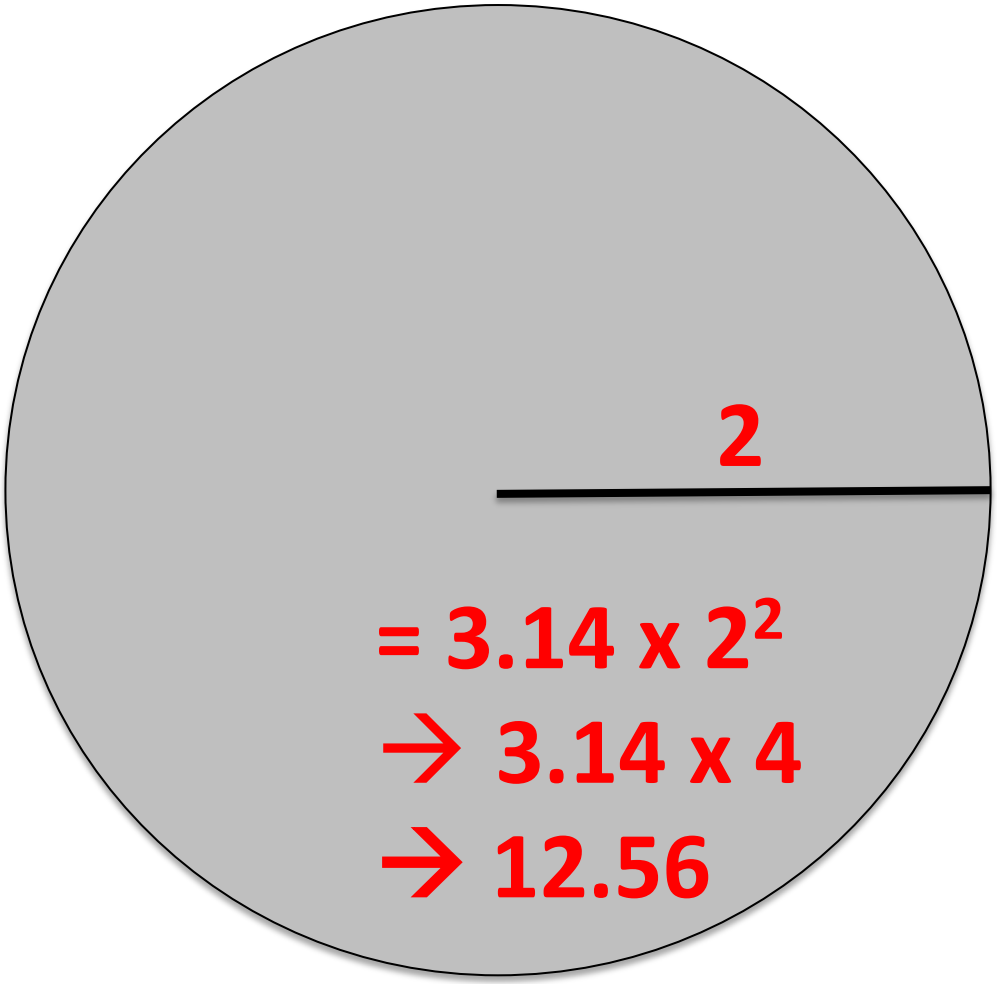
Example: area of unit circle



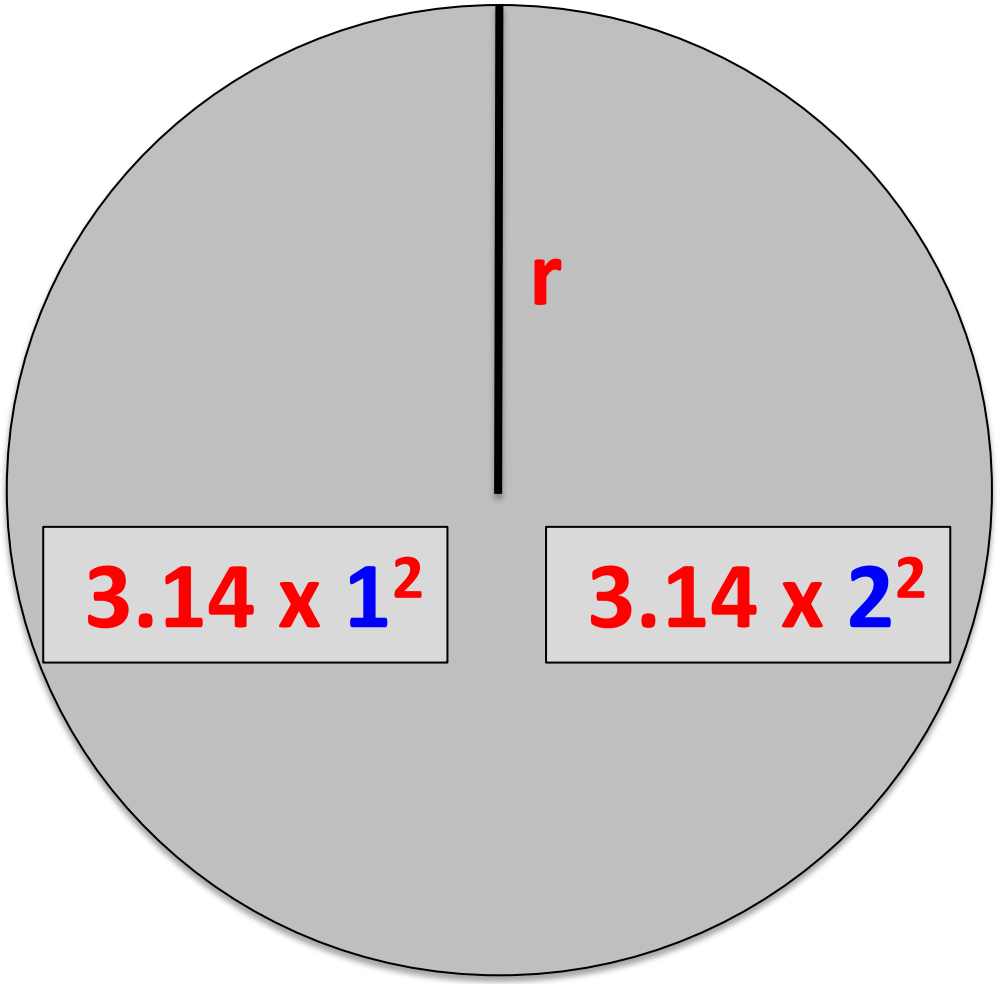
Example: area of circle of radius 2



Example: area of circle of radius **2**

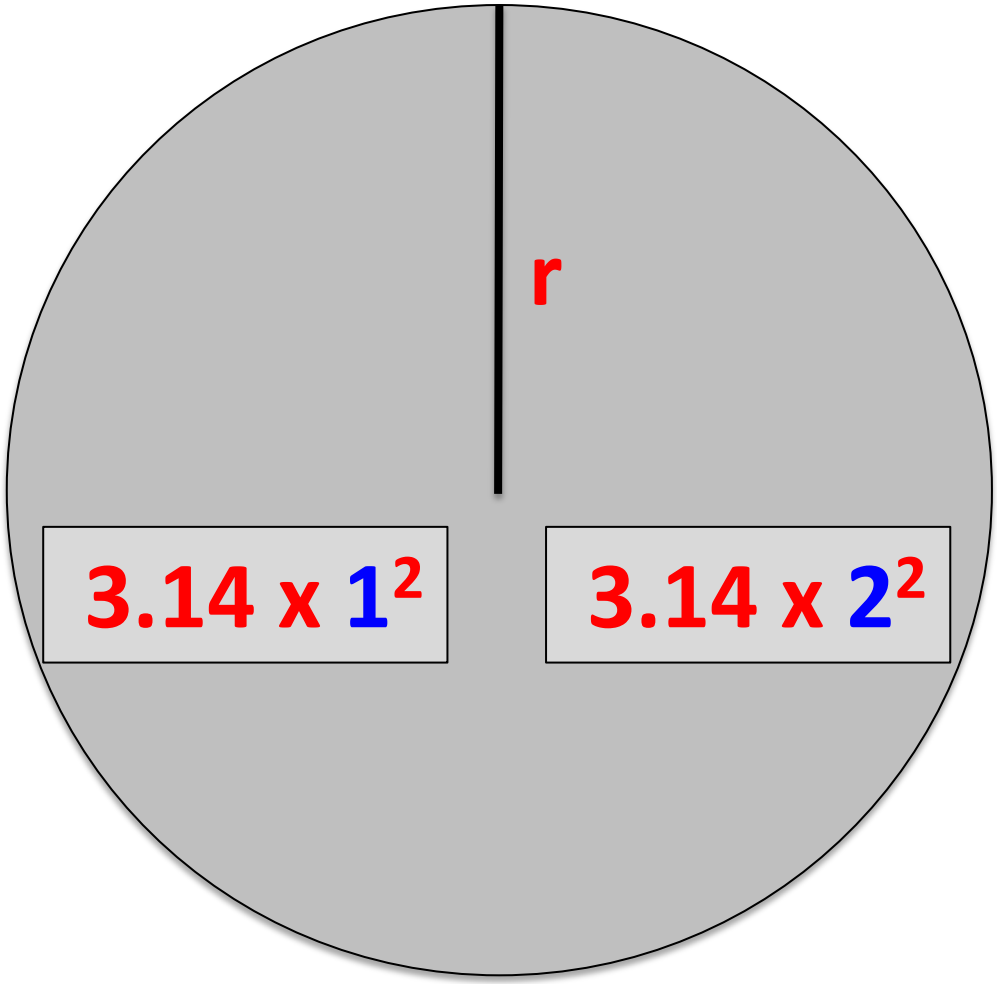


Example: area of circle of radius r



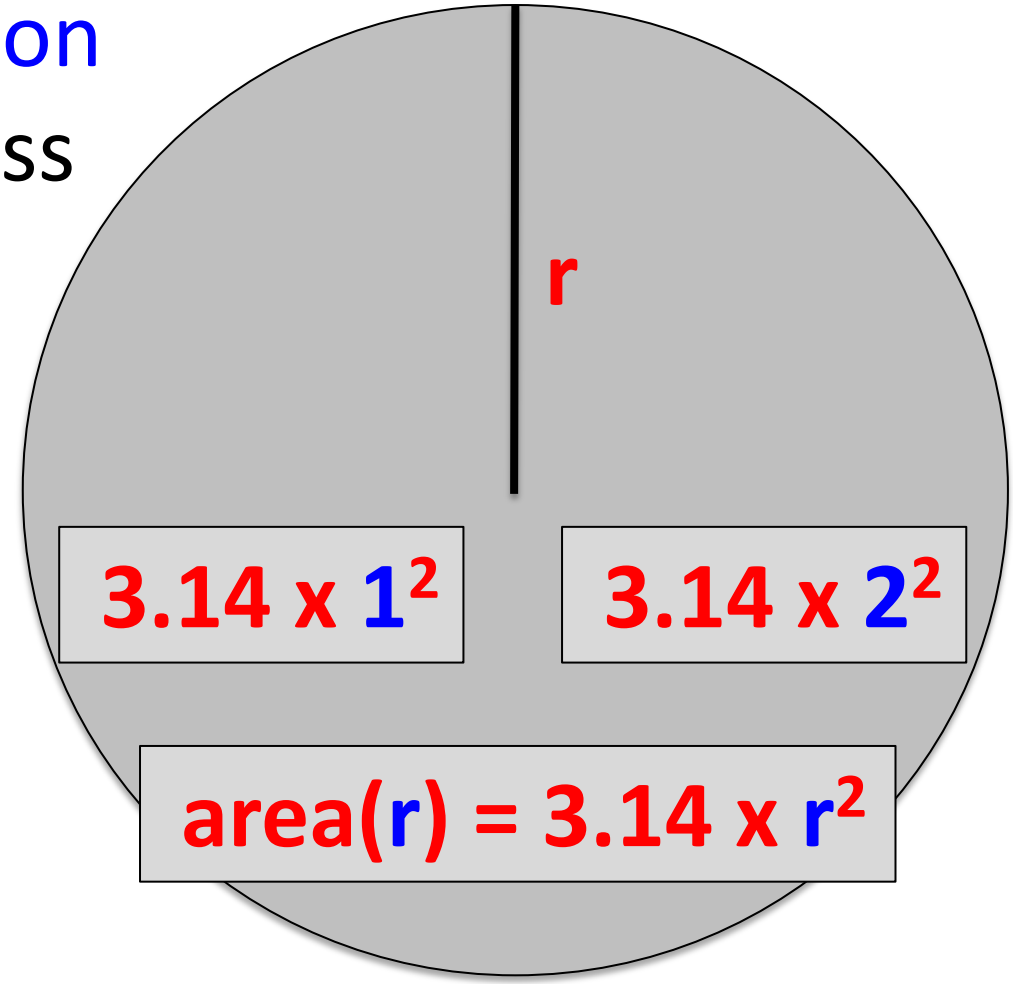
Example: area of circle of radius r

We want to *abstract* with respect to the variation(s).



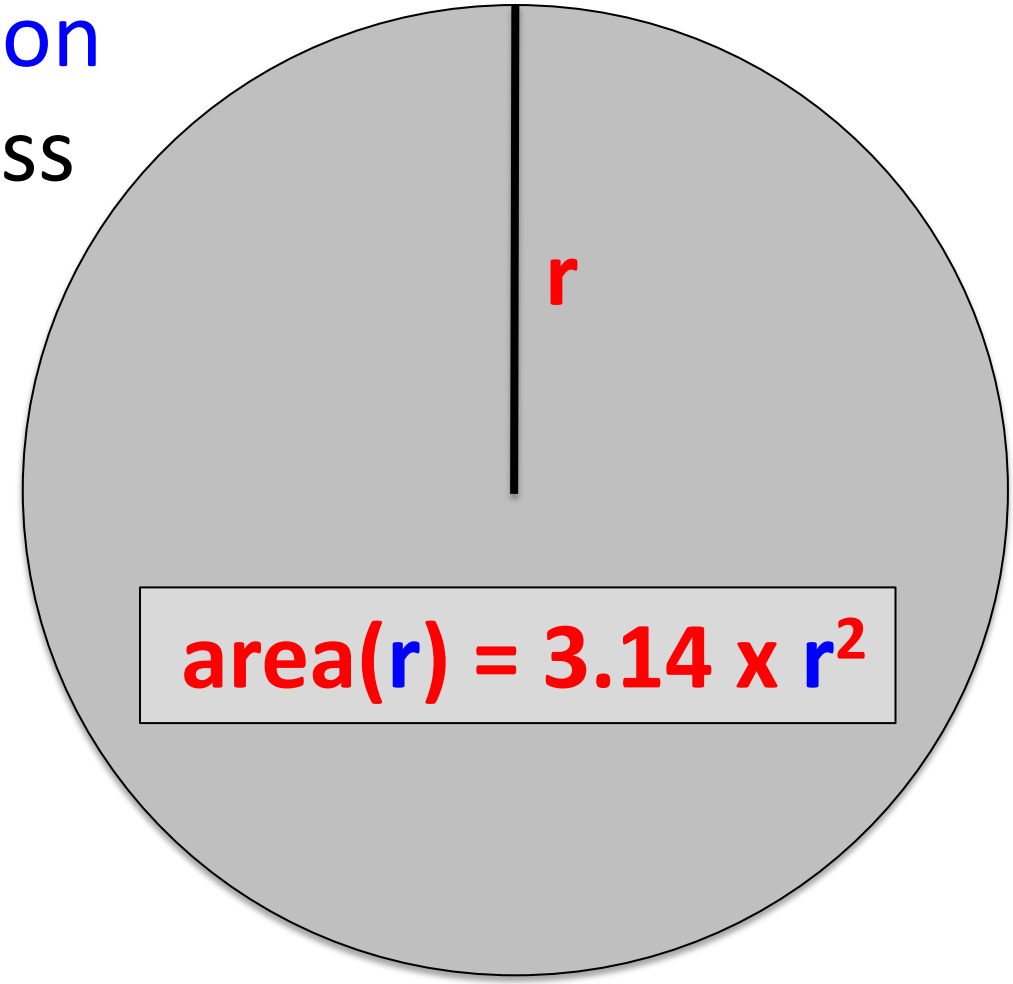
Example: area of circle of radius r

A function definition allows us to express the *abstraction*.



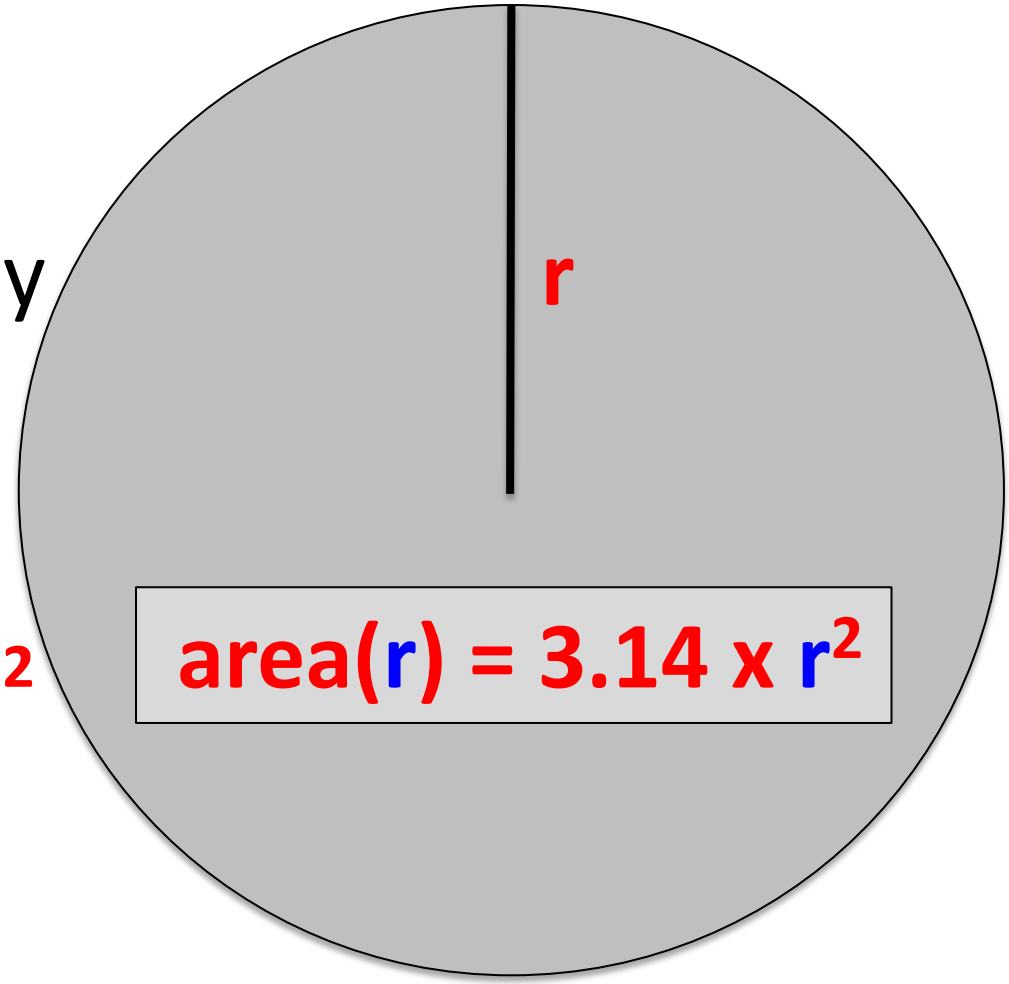
Example: area of circle of radius r

A function definition allows us to express the *abstraction*.



Example: area of circle of radius r

Our function definition can now be *used* or *called* by providing an input.



area(3) → 3.14 x 3²

→ 3.14 x 9

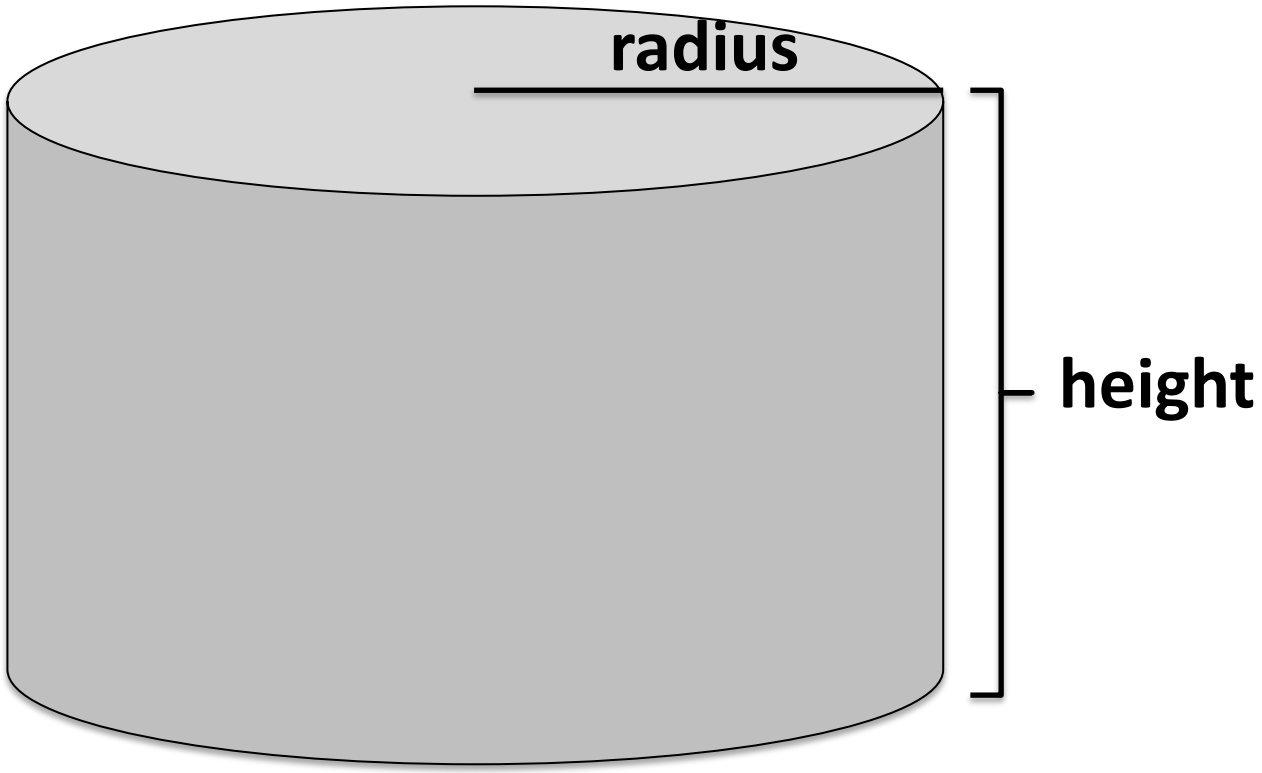
→ 28.26

In OCaml

```
# let area radius = Lib.pi *. radius ** 2.0

area : float -> float
```

Example: volume of a cylinder



In OCaml

```
let area radius = Lib.pi *. radius ** 2.0

area : float -> float

let volume radius height =
   (area radius) *. height

volume : float -> float -> float
```

Tools

- Programming language: OCaml

- System: Unix
- Collaboration: git & GitHub, Piazza
- Code Editor: Visual Studio Code
- Administration: Canvas

Why OCaml?

- Computation can be approached from either a mathematical or mechanical perspective
- From the former, coding is a *natural extension of algebra*

Why OCaml?

- OCaml emphasizes the most important ideas:
  - expression reduction/simplification,
  - functions, abstraction & composition
  - modularity

CS1 and CS2

- A principal theme of CS1 is mastering the art of expressing algorithms as functions, **procedural abstraction.**

- A principal theme of CS2 is mastering the art of writing new types, (values and functions), **data abstraction**.

What to Expect

- You'll often/usually be in a state of confusion
- You'll probably think you're the only one (you're not)
- You may sometimes feel anxious and hopeless to find a solution – hang in there!
- You'll probably experience elation when what you're attempting to build actually works (!)

Problem Set 0

Course Admin

Tour of course website

Resources

- Extensive lecture notes
- Most of our material is covered in lecture, background reading in *OCaml from the Beginning*.
- Office hours, Piazza, the internet, your colleagues

Collaboration Policy – CMU Rule

- By all means, hash things out with your peers. Find a whiteboard, draw diagrams, write formulas, pseudo-code etc, haggle, argue, laugh, eat cold pizza …

- BUT, when all is said and done, all of your joint work **must be erased** and each collaborator must be able to **write their own code** from the understanding they developed together.

- It's good practice to identify your study-mates in comments at the top of your code files.

Class Rules

- We may use laptops from time to time, but **laptops, tablets, phones are only permitted when explicitly specified**

- If you've come down with something like COVID or the flu or are otherwise sick, do not come to class or lab.

Grading

- 200 point scale
  - 110 points for problem sets
  - 60 points for exams, 2 midterms and final
  - 30 points for consistent course participation in lab, lecture & the class Piazza forum

How to Succeed in CS 1103

- Start problem sets *right away*!
- Pay careful attention to detail.
- Seek help when you need it.
- Show up consistently, participate in class, ask questions.