# Announcements

- Welcome to CS-151: Intro to Programming with Python!
- Things to do:
  - Access the course webpage at `http://www.willamette.edu/~jjrembold/classes/cs151/cs151/`
  - Read over the full syllabus
  - Get yourself a copy of the book
  - Make sure you can access and sign into Campuswire
  - Bring phone or computer for polling questions in future
- Homework
  - HW1 will be posted today
  - Will work in lab today to make sure you are setup with everything you need and have been introduced to the process

# My Vitals

| | |
|---|---|
| Name: | Jed Rembold |
| Office: | Collins 311 (it is shared) |
| Office Hours: | M,W 3-5pm and open door (basically always) |
| Email: | `jjrembold@willamette.edu` |
| Office Phone: | 503-370-6860 |

# Grading

- Standard 90/80/70 etc grade cut-offs
  - Top 2% gets +'s, bottom 2% get -'s

| | |
|---|---|
| Participation | 5% |
| Homework | 25% |
| Lab work | 15% |
| Midterm | 20% |
| Final Project | 10% |
| Final Exam | 25% |

# Participation

- Scored through participation in class polls
- Generally 1-3 polls per day
- Answering any poll gets you full points for the day (even if you are wrong!)
- Answering correctly gets you bits of extra credit
- Polling website at `http://rembold-class.ddns.net`
- Will start in full on Friday

# Homework

- Homework assignments will be given weekly and will be due on Friday nights
- Will generally be comprised of a few conceptual or theoretical problems and a few coding problems
- Submissions of both will be handled through Github Classroom
  - Introduction to this today in lab
- 10 cumulative late days over the entire semester without penalty, then work only accepted for 50% credit
- Start early and upload often!

# Lab Time

- Have lab in the hour after each class
- I will (try to) budget about 45 minutes of this hour
  - Exercises involving what we discussed in class that day
  - Some pair-programming on occasion
- Will either have you submit finished exercises to Github or, more often, check them off in person
- Remaining time will be your own to work on homework, ask questions, or leave

# Tests and Project

- Just two tests this semester:
  - Midterm on March 20
  - Final on May 8
- Tests will be text based, closed-book and without calculator or computer
- Also a small group project at the end of the semester
  - Will have both written and presentation portions
  - More details will come midway through the semester

# Campuswire

- You should have already received an email invitation to our class on Campuswire
- Forum allows a better medium to ask and answer questions
  - Allows everyone to see and benefit from responses/answers
  - Allows nicely formatted code snippets and equations
- Will be used for general announcements and occasional polling, so make sure to check it on occassion!

# On “Learning to Code”

- You all come from a wide variety of backgrounds
  - Sophomore through Seniors
  - Environmental Science, Psychology, Economics, Philosophy, Mathematics, Physics, Biology, International Studies, and of course Undeclared

# On “Learning to Code”

- You all come from a wide variety of backgrounds
  - Sophomore through Seniors
  - Environmental Science, Psychology, Economics, Philosophy, Mathematics, Physics, Biology, International Studies, and of course Undeclared
- You are all more than capable at succeeding in this class
  - You can give me directions to the store or how to bake a cake. Coding is no different.
  - It is just in a format you aren’t as used to and you are explaining it to computers, which, ironically, are inherently clueless

# On “Learning to Code”

- You all come from a wide variety of backgrounds
  - Sophomore through Seniors
  - Environmental Science, Psychology, Economics, Philosophy, Mathematics, Physics, Biology, International Studies, and of course Undeclared
- You are all more than capable at succeeding in this class
  - You can give me directions to the store or how to bake a cake. Coding is no different.
  - It is just in a format you aren’t as used to and you are explaining it to computers, which, ironically, are inherently clueless
- The only way to improve is through repetition and practice
  - Listening or reading won’t cut it
  - Break things off in small chunks that you can focus on

# A Division of Knowledge

Declarative: Statements of fact. What something *is*.

# A Division of Knowledge

Declarative: Statements of fact. What something *is*.

- This cake is comprised of butter, flour, baking powder, sugar, eggs and milk.
- Willamette is located at 900 State Street.

# A Division of Knowledge

**Declarative:** Statements of fact. What something *is*.

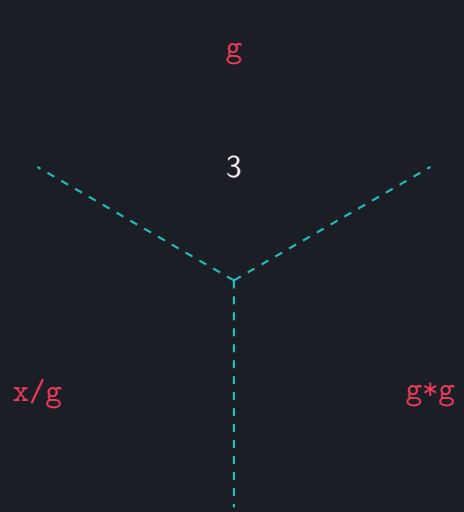
- This cake is comprised of butter, flour, baking powder, sugar, eggs and milk.
- Willamette is located at 900 State Street.

**Imperative:** Recipes for information. "How to" knowledge.

1. Whisk together flour and baking soda in one bowl.
2. Beat butter and sugar together, then add eggs.
3. Slowly add flour mixture while continuing to mix.
4. Divide batter between pans and cook at 350 degrees for 30 minutes

# A Numeric Example

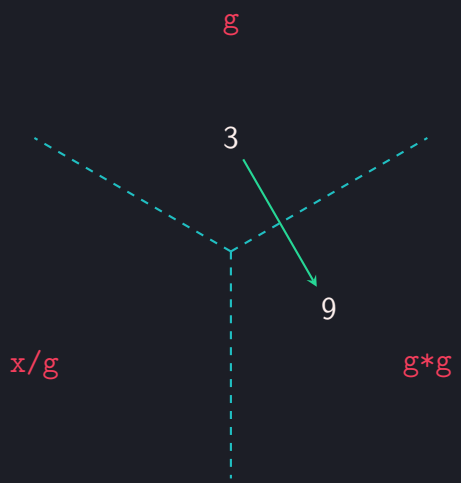
- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough



# A Numeric Example

- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough

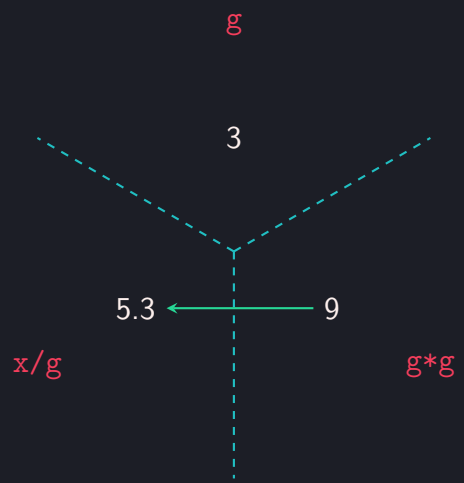
Finding $\sqrt{16}$



# A Numeric Example

- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough

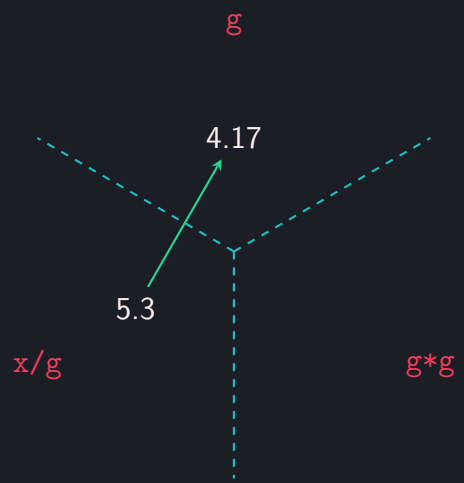
Finding $\sqrt{16}$



# A Numeric Example

- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough

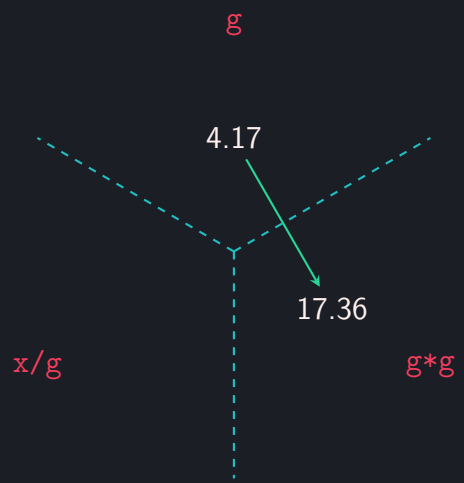
Finding $\sqrt{16}$



# A Numeric Example

- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough

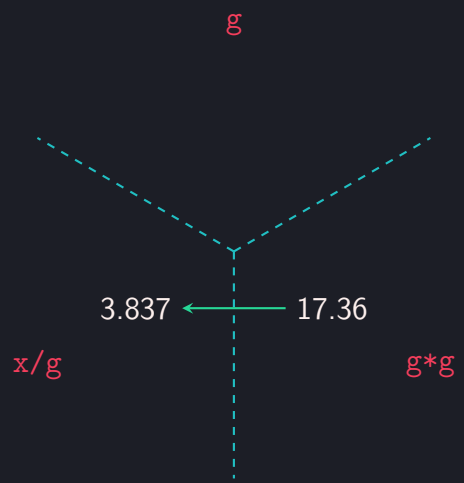
Finding $\sqrt{16}$



# A Numeric Example

- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough

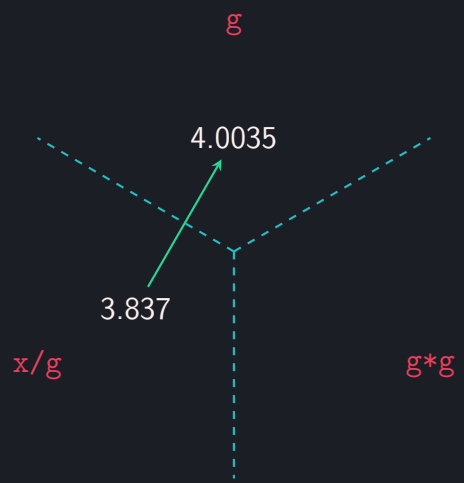
Finding $\sqrt{16}$



# A Numeric Example

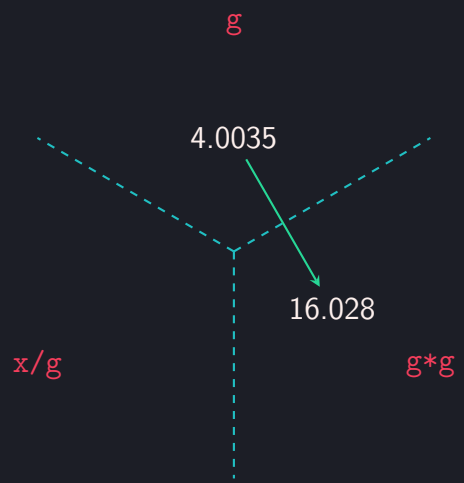
- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough

Finding $\sqrt{16}$



# A Numeric Example

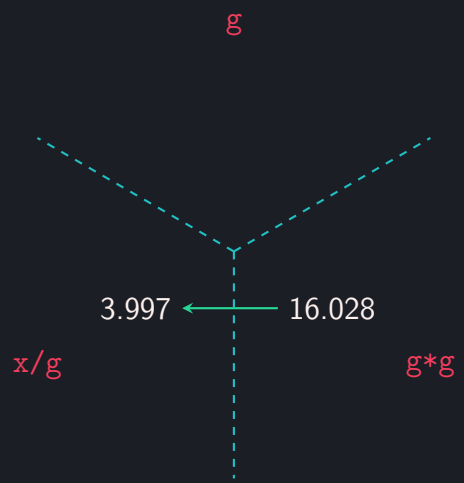
- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough



# A Numeric Example

- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough

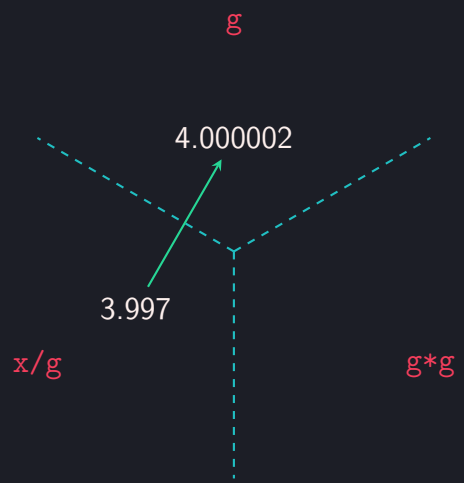
Finding $\sqrt{16}$



# A Numeric Example

- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough

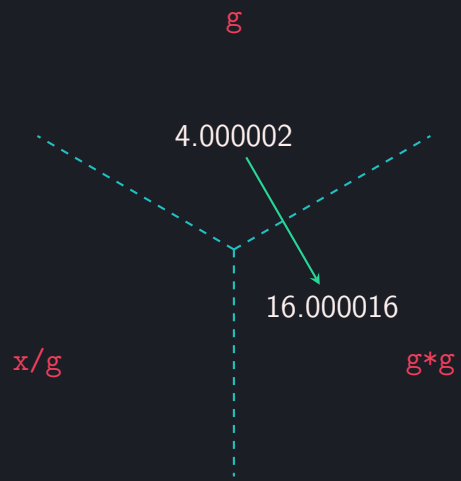
Finding $\sqrt{16}$



# A Numeric Example

- A square root of a number `x` is `y` if `y*y=x`.
- Recipe to find `y`:
  1. Start with a guess, `g`
  2. If `g*g` is close enough to `x`, stop and say `g` in the answer.
  3. Otherwise make a new guess by average `g` and `x/g`.
  4. Using the new guess, repeat until close enough

Finding $\sqrt{16}$



# Delicious Recipes

- What comprised our recipe?

# Delicious Recipes

- What comprised our recipe?
  1. A sequence of simple steps

# Delicious Recipes

- What comprised our recipe?
  1. A sequence of simple steps
  2. Some control flow that specified when to execute each step

# Delicious Recipes

- What comprised our recipe?
  1. A sequence of simple steps
  2. Some control flow that specified when to execute each step
  3. Some method of determining when to stop

# Delicious Recipes

- What comprised our recipe?
  1. A sequence of simple steps
  2. Some control flow that specified when to execute each step
  3. Some method of determining when to stop
- The 3 together comprise what we call an algorithm

# Delicious Recipes

- What comprised our recipe?
  1. A sequence of simple steps
  2. Some control flow that specified when to execute each step
  3. Some method of determining when to stop
- The 3 together comprise what we call an algorithm
  - Algorithms are what you use to solve your problem or accomplish your task

# Delicious Recipes

- What comprised our recipe?
  1. A sequence of simple steps
  2. Some control flow that specified when to execute each step
  3. Some method of determining when to stop
- The 3 together comprise what we call an algorithm
  - Algorithms are what you use to solve your problem or accomplish your task
  - Programming or coding is how you communicate those algorithms to the computer

# Rise of the Machine

- Recipes can be bound or connected to a machine in a variety of ways
  - Mechanical, electrical, hydraulic, etc
- Computing machines (computers) fall into two overall categories depending on their capabilities
  - Fixed program computers
    - Algorithm(s) hardcoded into the machine and can't be easily changed
  - Stored program computers
    - The machines executes *and* stores instructions

# Stored Programs

- Some sequence of instructions is *stored* inside the computer
  - Generally built from a predefined set of primitive instructions. Things like:
    - arithmetic
    - logic
    - simple tests
    - moving around data
- Some special program (the interperter) executes each instruction in order
  - Simple tests control the flow of instructions
  - Stop when out of instructions

# How Primitive

- Alan Turing describes with his Turing Machine that any computable problem can be solved with 6 basic primitive operations
  - There do exist problems whose solutions simply can not be computed!

# How Primitive

- Alan Turing describes with his Turing Machine that any computable problem can be solved with 6 basic primitive operations
  - There do exist problems whose solutions simply can not be computed!
- The Church-Turing thesis ensures that if any machine can simulate a Turing Machine, then it is also a Turing Machine
  - Said to be Turing Complete
  - All modern languages are Turing Complete, as well as many other things!
    - Magic: The Gathering ← scientific paper
    - Pokemon Yellow ← creating a midi player from Pokemon Yellow
    - Minecraft
    - Excel
    - Powerpoint ← very worth watching!
    - and more

# How Primitive

- Alan Turing describes with his Turing Machine that any computable problem can be solved with 6 basic primitive operations
  - There do exist problems whose solutions simply can not be computed!
- The Church-Turing thesis ensures that if any machine can simulate a Turing Machine, then it is also a Turing Machine
  - Said to be Turing Complete
  - All modern languages are Turing Complete, as well as many other things!
    - Magic: The Gathering ← scientific paper
    - Pokemon Yellow ← creating a midi player from Pokemon Yellow
    - Minecraft
    - Excel
    - Powerpoint ← very worth watching!
    - and more
- Modern languages have much more convenient primitives to work with

# Communication

- The programming language provides some set of primitive operations
  - For instance $*$ or $+$
- We combine those with literals to describe what we want the machine to do
  - Examples would be numbers: `25`, or strings of characters: `"fishsticks"`.
- Legal combinations of literals and operations are call expressions
  - But what constitutes “legal”?

# Aspects of Languages

- Any language, English or Python, has rules about how you can combine its constituent parts
- Syntax defines how various elements or parts must be arranged to be well formed.
  - In English, proper sentences have subject and predicate, or generally `<noun>`, `<verb>`, `<noun>`.
    - "Billy cow house" ← not syntactically valid
    - "Billy runs home" ← syntactically valid
  - In Python, we commonly have `<literal>`, `<operation>`, `<literal>`
    - `7"hi"` ← not syntactically valid
    - `7*12` ← syntactically valid

# Syntax is not enough

- It is totally possible to have syntactically valid language that still doesn't make sense or have meaning:
  - "Trees drops branches."
- Static semantics defines which syntactically valid expressions have meaning.
  - `12+5` ← syntactically valid and valid static semantics
  - `12+"hi"` ← syntactically valid but static semantic error

# Python > English

- Semantics is the meaning associated with a syntactically correct string of symbols with no static semantic errors.
  - In English, we often can have sentences with multiple meanings:
    - "Flying planes can be dangerous."
    - "I can not praise this student too highly."
  - In programming, an expression will only have a single meaning
    - But it may be different that the programmer intended!

# Where everything breaks...

- Syntax errors:
  - happen often and are easily caught and fixed
- Static semantic errors
  - Some languages will check before running
  - Python doesn't do much before but will check and can catch some while running
- No semantic errors, but different meaning than intended
  - Program crashes and stops running
  - Program runs forever
  - Program gives answer but incorrect or different than expected