BST 273: Introduction to Programming

Eric Franzosa (franzosa@hsph.harvard.edu)

Kevin Bonham (kbonham@broadinstitute.org)

http://franzosa.net/bst273

Outline for today's class

- Course overview
- Introduction to programming & Python
- Computer setup

Course Overview

Syllabus

- Everything I’m about to go over is covered in the course syllabus
- Syllabus is available in the “Course Documents” module on Canvas
  - We will visit Canvas later in the lecture
  - **Who doesn’t already have access to Canvas?**

Course overview

- BST 273 is a half-semester introduction to computer programming
    - Meetings Tuesdays and Thursdays (TR), 11:30am-1pm in this room (FXB G13)
- In-class activities, but no separate lab component
- Intended for students who have <u>never programmed before</u>
    - Experience running commands in computing environments (R, MATLAB) OK
    - Otherwise talk to me
- Entry-point for other courses with a programming prerequisite

Course Staff

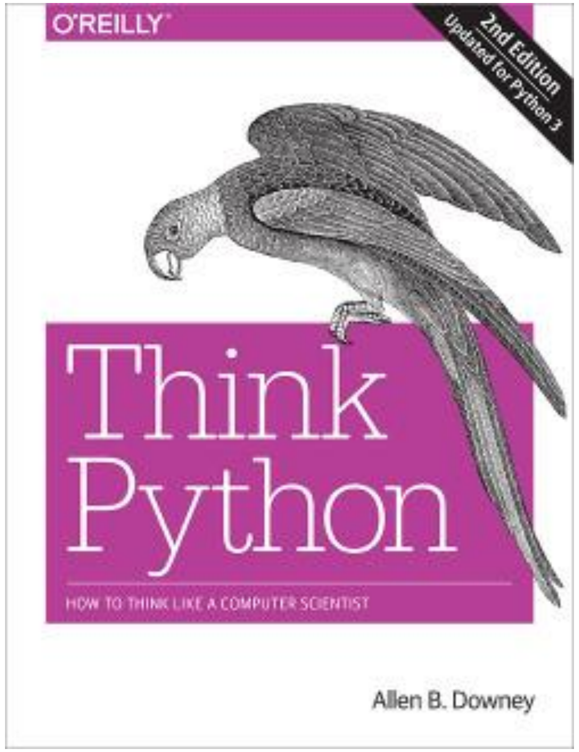
- Instructors (2):
  - Eric Franzosa
  - Kevin Bonham
  - Calling us “Eric” and “Kevin” is fine
- Teaching Assistants (3):
  - Shirley Liao
  - Emma Thomas
  - Marina Cheng
- Contact us through Canvas or via the email addresses from the syllabus
  - If emailing, please include “BST 273” in the subject line

Course Schedule

Week	Date	Day	Unit	Lecture
0	09/04/2018	T	**Fundamentals**	Orientation
0	09/06/2018	R	**Skills**	Working on the command line
1	09/11/2018	T	**Fundamentals**	Variables, scalar data types and methods
1	09/13/2018	R	**Fundamentals**	Collection data types and iteration
2	09/18/2018	T	**Fundamentals**	Conditional logic and flow of control
2	09/20/2018	R	**Fundamentals**	Working with modules, examples with file I/O
3	09/25/2018	T	**Fundamentals**	Writing functions, references vs. data
3	09/27/2018	R	**Fundamentals**	Making an executable script
4	10/02/2018	T	**Skills**	Version control and intro final projects
4	10/04/2018	R	**Skills**	Testing, debugging, getting online help
5	10/09/2018	T	**Special topics**	Interacting with external programs
5	10/11/2018	R	**Special topics**	Regular expressions
6	10/16/2018	T	**Special topics**	Scientific computing with Python
6	10/18/2018	R	**Special topics**	Object-oriented Python
7	10/23/2018	T	**Special topics**	Parallelism and workflows in Python
7	10/25/2018	R	**Special topics**	Next steps for developing as a programmer

Textbooks / Readings

- Think Python 2nd Edition by Allen B. Downey
  - **Required**
  - Available in its entirety online at https://greenteapress.com/wp/think-python-2e/
  - Available for purchase in-print if desired (not required)
  - Readings will be listed per-lecture on Canvas
- Additional online readings will be linked from Canvas



Course structure

- Five homework assignments (13% × 5 = 65%)
- Final project (25%)
- Participation (10%)

Homework assignments

- Five assignments total (each 13% of final grade, 65% total)
- Weekly starting next week and excluding last two weeks
    - i.e. Final Project work replaces homeworks here
- Published Mondays on Canvas
- Due via electronic Canvas hand-in the following Friday by 11:59pm
- Each homework will be a Python script
- More formatting details during next Tuesday's lecture
    - (Once first assignment is published)

Final project

- 25% of final grade (∼2 homeworks)
- Complete and document a Python script to solve a problem in data analysis
- A number of options will be provided, or you can design your own
    - Options + signups will go out the third-to-last week of class
    - Must seek instructor approval if designing your own (details to follow)
- Final project work will go on during last two weeks of class
- Due Friday October 26th 11:59pm (end of last class week)

Participation

- 10% of final grade
- This class has an extensive hands-on, in-class component
  - We expect you to be here and participate
- Attendance will be quantified using Canvas “Quizzes”
  - **No right or wrong answers, not graded, but must submit during class**
  - Practice “quiz” today re: office hours will have a longer submission window
- Breakdown
  - Augmented by e.g. asking/answering questions in class
  - Full credit (10%): 0-1 unexplained absences
  - Medium credit (5-9%): 2-3 unexplained absences
  - Low credit (0-4%): 4+ unexplained absences

Late-work policy

- ***Please hand in assignments on time***
- If 1 day late, assignment will be graded out of a maximum of 90%
- If 2 days late, maximum of 75%
- If 3 days late, maximum of 50%
- If 4+ days late, no credit
- Extensions may be granted if requested with reason at least 24 hours in advance of the assignment deadline

Collaboration policy

- **DON'T**
  - Look at / copy another student's <u>assignment</u> code
  - Show your <u>assignment</u> code to another student
  - Post <u>assignment</u> code online (in the Canvas Discussion Board or elsewhere)
  - Treated as violations of the Academic Integrity policy (linked in full via Syllabus)
- **DO**
  - Seek help for assignment code during Instructor/TA office hours
  - Work with other students on <u>in-class</u> programming activities
  - Discuss general concepts with other students
  - Consult instructors if you have questions about the OK-ness of your collaboration

Other class policies

- Please bring a laptop with you to class for in-class programming
    - If this poses a problem, please talk to us
- Audits are OK if there's room – priority goes to registered students
    - Contact me to be added to Canvas as a "guest"
- We know it's lunch time, but please don't eat during class
    - If you bring a drink, please keep it off the tables to avoid computer spills

Office hours

- Instructor Office Hours
  - Currently Fridays, 11am-12pm, SPH2 rm. 434
  - I will be there at the above time <u>this Friday</u> for general course questions
  - Some room to negotiate on time if this is universally bad (see Canvas poll)
- TA Office Hours
  - To be scheduled via Canvas poll
  - 1 hour per TA per week
  - Biased toward the end of the week (closer to homework hand-in)
- Fill out Canvas poll ASAP
  - Would like to have office hours finalized by next class

Questions?

(franzosa@hsph.harvard.edu)

Look at Canvas

Philosophy of Programming

Learning to Program

- Why do it?
  - Make easy tasks easy
  - Make hard tasks possible
  - Improve accuracy and efficiency in your work
  - It’s empowering!
- What does it take?
  - Learn to identify problems that computers can solve
  - Learn to describe those problems in a way that computers can understand
  - Learn a programming language to translate those descriptions into code

Learning to Program

- Why do it?
  - Make easy tasks easy
  - Make hard tasks possible
  - Improve accuracy and efficiency in your work
  - It’s empowering!
- What does it take?
  - Learn to identify problems that computers can solve (**not too bad**)
  - Learn to describe those problems in a way that computers can understand (**harder**)
  - Learn a programming language to translate those descriptions into code (**not too bad**)
  - ***Analogous to learning spelling/grammar vs. learning to write well***

How computers “think”

- Computers are well-suited to solving problems that can be expressed as transformations of data (converting input data into output data)
- These transformations are **algorithms**: predefined rules or calculations we apply to data in pursuit of solving problems
- The goal of programming is to translate an algorithm so a computer can understand it and apply it to arbitrary data

Input Data	Output Data
`List of numbers`	`Sum of numbers`
`Pattern and text`	`Locations of pattern in text`
`Atmospheric data`	`Weather prediction`

How computers “think” (pros and cons)

- Computers work very quickly, performing millions of calculations per second
  - Computers are fast, even when programmed naively
- Computers do exactly what you tell them to do*
  - They don’t make their OWN mistakes
- Computers don’t read between the lines / have good intuition
  - They do only what you tell them to do explicitly
- Computers do exactly what you tell them to do*
  - They will follow YOUR mistakes without question, often without telling you

An example with sorting

Consider some unsorted numbers (**input data**):

25	4	11	9	1	8	10	2	2	6

I'm sure you could tell me that the sorted version (**output data**) is this:

1	2	2	4	6	8	9	10	11	25

But how did you get there?

An example with sorting

Consider some unsorted numbers (**input data**):

25	4	11	9	~~1~~	8	10	~~2~~	2	6

An **algorithm** for sorting (that works well for us humans) is to iteratively find, copy, and eliminate the smallest remaining number…

25	4	4	4	1	1	1	1	1	1

25	4	4	4	4	4	4	2	2	2

Output data:

1	2	2	4	6	8	9	10	11	25

An example with sorting

- "Keep finding the smallest number" is a generic algorithm
    - It will work on any numbers (ties, fractions, etc.)
    - It will work on arbitrarily large lists of numbers
- Note how the algorithm defined a simple but explicit procedure ("keep finding the smallest number") and repeated it until we had a complete solution
    - This is a common theme in algorithms / programming
    - Unlike us, a computer can repeat simple steps without getting tired / making mistakes
- Practice decomposing intuitive procedures into generic algorithms
    - We'll do something with this on the first homework

Programming vs. Computer Science

- Computer Science is concerned with, among other things, finding the *best* algorithm to solve a given problem
  - With "best" usually defined as "fastest" or "requiring the fewest steps"
- The "keep finding the smallest number" algorithm is not particularly efficient because it requires us to repeat a lot of work
  - E.g. repeatedly considering/rejecting the first number, 25, as the smallest
- There are faster search algorithms out there, but…

First Rule of Programming:

First get it right – worry about speed later (or never)

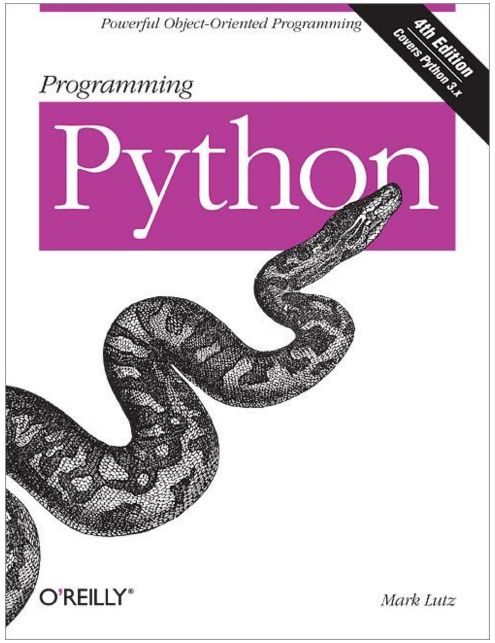
Questions?

(franzosa@hsph.harvard.edu)

Python

Introduction to Python

- We'll be learning to program in Python in this course
- Python exists today in two major flavors
  - Python 2.x (getting old)
  - Python 3.x (the best place to get started)
  - Aside from a couple of things, they are superficially very similar
- Invented by Dutch programmer Guido van Rossum c. 1991
- Named after Monty Python, not the snake
- Python programmers sometimes called "Pythonistas"
  - Mostly by themselves…



Introduction to Python

- Python is a “high-level” programming language
  - Designed to be easier for humans to read than computers
  - Emphasis on words over symbols in code
  - White space used to denote blocks of code (rather than symbols)
- Python is an interpreted programming language
  - Computer directly follows your code, without pre-compiling to something else
- Large “Standard Library” (built-in code) + 1,000s of installable packages

Second Rule of Programming:

Re-using working code is “appropriately lazy”

Ex. Python `sorted( )` function

Introduction to Python

- Python favors speed/ease of development over speed of execution
  - Good for solving personal research questions (run-once scripts)
  - Good for solving objectively “fast” problems (seconds of compute)
  - Good for “stitching” results from highly optimized code
- Blazingly fast compared to manual computation
- Slow parts can be sped up (optimized) later if needed
  - We’ll talk about fast numerical computing in Python later in the course
- Used across many industries and academic fields

Introduction to Python

- Python bears a striking resemblance to "pseudocode": a language-agnostic way of representing computer algorithms (often in publications)

Example of pseudocode

Algorithm 2: Division

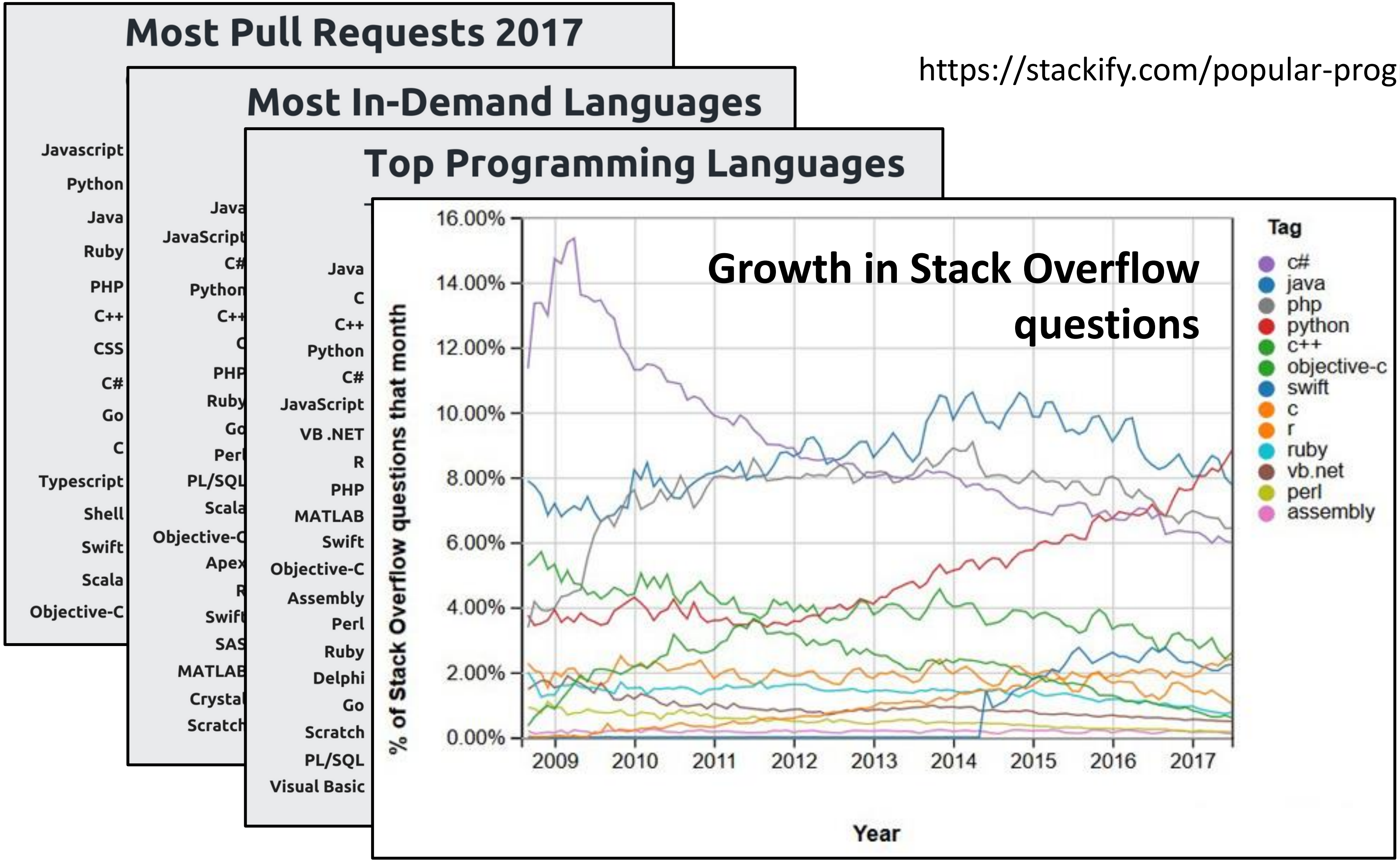
```
function divide (x, y);
Input: Two n-bit integers x and y, where y ≥ 1
Output: The quotient and remainder of x divided by y
if x = 0 then
    return (q, r) = (0, 0)
else
    set (q, r) = divide(⌊x/2⌋, y);
    q = 2 × q, r = 2 × r;
    if x is odd then
        r = r + 1
    end
    if r ≥ y then
        r = r − y, q = q + 1
    end
    return (q, r)
end
```

Example of Python code

```python
def quicksort(list):
    if len(list) <= 1:
        return list
    pivot = list[(len(list)-1)/2]
    list.remove(pivot)
    less = []
    greater = []
    for num in list:
        if num <= pivot:
            less.append(num)
        else:
            greater.append(num)
    return quicksort(less) + [pivot] + quicksort(greater)
```

Not that it's a popularity contest, but…

Images sourced from:
https://stackify.com/popular-programming-languages-2018/



Questions?

(franzosa@hsph.harvard.edu)

Transition to Computer Setup