CIS192 Python Programming
Introduction

Dan Gillis
University of Pennsylvania

January 20th, 2016
Outline

1 About this Course
   - People
   - Curriculum

2 Logistics
   - Grading
   - Office Hours
   - Software

3 Python
   - What is Python
   - The Basics
   - Dynamic Types
People

Instructor: Robert Rand

TAs: Adel Qalieh, Dan Gillis, Harry Smith
What are CIS 110-121?

- Core computer science courses at Penn.
- Teach fundamentals of programming and critical concepts like recursion and data structures.
- Much more important than this course.
- But maybe less fun.
Let’s learn Python!

Part One - Python Basics (∼6 weeks)
- How to program in Python.
- Enough to add “Python” to your resume.

Part Two - Fun with Python (∼8 weeks)
- Cool stuff we can do in Python (eg. Machine Learning, Web Apps)
- Check out previous iterations of course.
- Send me ideas (suggestion box is open all semester)
What is the CIS 19x Shared Lecture?

- Learn CS skills with Swapneel Seth.
- Some of these will be used in the course (particularly command line arguments).
- No homework
- Meets on Tuesdays
Outline

1 About this Course
   - People
   - Curriculum

2 Logistics
   - Grading
   - Office Hours
   - Software

3 Python
   - What is Python
   - The Basics
   - Dynamic Types
Homeworks: 65% of grade.
Final Project: 30% of grade.
Participation: 5% of grade.
Homework

- Due Sunday at midnight, eleven days after class
- One “Late Week” extension
  - Subsequent late homeworks will be penalized 10%
  - Homework that is more than one week late will not be accepted.
- Submit via Canvas
Will be graded for correctness, efficiency and style.

Correctness: We will often include “test” files for unit testing - use them and add your own tests.

Style: Be sure to follow the PEP-8 style guidelines - you can add an automatic checker to most IDEs.

Efficiency: Think before coding and try to avoid unbounded recursion and highly nested loops.
General: Code of Academic Integrity
  - [http://www.upenn.edu/academicintegrity/ai_codeofacademicintegrity.html](http://www.upenn.edu/academicintegrity/ai_codeofacademicintegrity.html)

Specifics:
  - Discuss with up to one partner.
  - Write your own code.
  - Do not look up sources that directly solve the problem.
  - Cite partner and sources consulted (if any).
Final Project

- Work with up to one partner
  - You can search for partners on Piazza
- Proposal due March 13th
- Final Projects due April 27th
- Project Demos will be scheduled for during the department-wide Demo Day and Reading Days.
- Start thinking about a project now!
Office Hours

- Robert Rand: Friday 2:30 - 4:30pm, Levine 513 and by appointment.
  - Not including this Friday
- Adel Qalieh: Monday 3-5pm, Location TBD.
- Dan Gillis: Thursday 4-6pm, DRL 4N30
- Harry Smith: Tuesday 3-5pm, Moore 212
Programming Environment

- PyDev for Eclipse Recommended
  - Emacs/vim are also good
  - Plenty of alternatives for various OSs
- *Set your editor to interpret tabs as four spaces*
  - Python is whitespace-sensitive
Use Piazza to find project partners and discuss lectures/homeworks with your peers.
Outline

1. About this Course
   - People
   - Curriculum

2. Logistics
   - Grading
   - Office Hours
   - Software

3. Python
   - What is Python
   - The Basics
   - Dynamic Types
Easy to Learn

I learned it last night! Everything is so simple!
Hello world is just print "Hello, world!"

I dunno... dynamic typing? whitespace?
Come join us! Programming is fun again!
It's a whole new world up here!
But how are you flying?

I just typed import antigravity
That's it?

... I also sampled everything in the medicine cabinet for comparison.
But I think this is the python.
Skating uphill like this is amazing. Years of gliding downhill and pushing uphill, and now suddenly it's gliding both ways.

It's like going from C to Python. You don't realize how much time you were spending on the boring parts until you don't have to do them anymore.

But coding C or assembly makes you a better programmer. Maybe the boring parts build character.

Yeah... but it depends how you want to spend your life. See, my philosophy is—
for thing in something:
    if type(thing) == str:
        string_handle()
    elif thing == None:
        ...

Read Evaluate Print Loop

Type “Python” at the terminal

Test out language behavior here

Get information with `dir()`, `help()`, `type()`
## 2.7 vs. 3.4

<table>
<thead>
<tr>
<th>Python 2.7</th>
<th>Python 3.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>print &quot;hello world!&quot;</td>
<td>print(&quot;hello world!&quot;)</td>
</tr>
<tr>
<td>1/2 = 0</td>
<td>1/2 = 0.5</td>
</tr>
<tr>
<td>1 // 2 = 0</td>
<td>1 // 2 = 0</td>
</tr>
<tr>
<td>No Unicode support</td>
<td>Unicode support</td>
</tr>
<tr>
<td>More popular</td>
<td>Gaining popularity</td>
</tr>
<tr>
<td>Better library support</td>
<td>Good library support</td>
</tr>
</tbody>
</table>
from __future__ import absolute_import, division, print_function

Changes Python 2.7’s behavior to mimic 3.x’s
Best of both worlds
Add to the top of every .py file.
Static Types

What are static types?

- A form of integrated specification
- Enforced at compile time.
- Verbose
- Python does not have these.
Dynamic Languages are Static Languages

While reviewing some of the comments on my post about parallelism and concurrency, I noticed that the great fallacy about dynamic and static languages continues to hold people in its thrall. So, in the same “everything you know is wrong” spirit, let me try to set this straight: a dynamic language is a straightjacketed static language that affords less rather than more expressiveness. If you’re one of the lucky ones who already understands this, congratulations, you probably went to Carnegie Mellon! For those who don’t, or think that I’m wrong, well let’s have at it. I’m not going to very technical in this post; the full technical details are available in my forthcoming book, Practical Foundations for Programming Languages, which is available in draft form on the web.

So-called dynamic languages (“so-called” because I’m going to argue that...
Dynamic Types

- Basically a system of tags.
- Let’s see how they work in practice.
Identifiers, Names, Variables

- All 3 mean the same thing
- [A-Za-z0-9_] First character cannot be a number
- Variable naming convention
  - Functions and variables: lower_with_underscore
  - Constants: UPPER_WITH_UNDERSCORE
x = 1
y = x
x = 'a'

Diagram:

```
X --> 1
```
* x = 1
* y = x
* x = 'a'

Graph:

- **X** connected to 1
- **Y** connected to 1
x = 1
y = x
x = 'a'

\[
\begin{array}{c}
\text{X} \\
\text{Y}
\end{array}
\quad
\begin{array}{c}
\text{a} \\
1
\end{array}
\]
Types

- Every object has a type
- Inspect types with `type(object)`
- `isinstance(object, type)` checks type hierarchy
- Types can be compared for equality but usually want `isinstance()`
- Some types:
  - int, float, complex
  - str, bytes, tuple
  - list, bytearray
  - range, bool, None
  - function
Objects

- Python treats all data as objects
- **Identity**
  - Memory address
  - Does not change
- **Type**
  - Does not change
- **Value**
  - Mutable $\rightarrow [1,2]$
  - Immutable $\rightarrow (1,2)$
Math

- **Literals**
  - Integers: 1, 2
  - Floats: 1.0, 2e10
  - Complex: 1j, 2e10j
  - Binary: 0b1001, Hex: 0xFF, Octal: 0o72

- **Operations**
  - Arithmetic: + - * /
  - Power: **
  - Integer division: //
  - Modulus: %
  - Bitwise: << >> & | ^
  - Comparison: <, >, <=, >=, ==, !=

- **Assignment Operators**
  - += *= /= &= ... 
  - No ++ or --
Strings

- Can use either single or double quotes
- Use single to show double flip-flop """" $\rightarrow$ ' and """" $\rightarrow$ "
- Triplequote for multiline string
- Can concatenate strings by separating string literals with whitespace
- Strings are not unicode
  - One of the major differences between Python 2 and Python 3
- Prefixing with r means raw. No need to escape: r’\n’
Conditionals

- One `if` block
- Zero or more `elif` blocks
- Zero or one `else` block
- Booleans: `True` `False`
Sequences

- **Immutable**
  - Strings, Tuples, Bytes

- **Mutable**
  - Lists, Byte Arrays

- **Operations**
  - `len()`
  - Indexing
  - Slicing
  - `in`
  - `not in`
Range

- List of numbers
  - Stores all values in memory

- `range(stop), range(start,stop)
  range(start,stop,step])`

- start defaults to 0
- step defaults to 1
- All numbers in [start,stop) by incrementing start by step
- Negative steps are valid
- Alternate function: `xrange`
  - Immutable sequence of numbers
  - Memory efficient: Calculates values as you iterate over them
Loops

- For each loops
  - Iterate over an object

- While loops
  - Continues as long as condition holds

- Both
  - else: executes after loop finishes
  - break: stops the loop and skips the else clause
  - continue: starts the next iteration of the loop
Functions

- Functions are first class
  - Can pass them as arguments
  - Can assign them to variables

- Define functions with a `def`

- `return` keyword to return a value

- If a function reaches the end of the block without returning
  It will return `None` (null)
Imports

- Allow use of other python files and libraries
- imports: `import math`
- Named imports: `import math as m`
- Specific imports: `from math import pow`
- Import all: `from math import *`