CIS192 Python Programming
Introduction

Robert Rand
University of Pennsylvania

August 27, 2015
1 Logistics
   - Grading
   - Office Hours
   - Shared Lecture
   - Software

2 Python
   - What is Python
   - The Basics
   - Dynamic Types
Grading

- Homeworks: 65% of grade.
- Final Project: 30% of grade.
- Participation: 5% of grade.
Homework

- Due Sunday at midnight, ten days after class
- One late week
- Can discuss with a partner
- Code must be your own
- Cite partner / all references used
- Submit via Canvas
Final Project

- Work with up to one partner
- Proposal due Nov 8th
- Demo ready version due last day of class
- Final version due last day of exams
Office Hours

- **Instructor:** Robert Rand
  - Thursday 1:30 - 3:30pm, Levine 513

- **TA:** Joseph Cappadona
  - Tuesday 3:00 - 5:00pm, DRL 4E9

- **TA:** Raymond Yin
  - Friday 3:00 - 5:00pm, Towne 303
Class: CIS 19X Shared Lecture

- Room: Towne 100
- Time: Tuesdays 6:00 - 7:30
- Only a few meetings per semester
- Includes Unix skills, version control

Instructor: Swapneel Sheth
Programming Environment

- PyDev for Eclipse Recommended
  - Emacs is also good
- Make sure your editor interprets `TAB` as four spaces
  - Python is whitespace-sensitive!
Use Piazza to find project partners and discuss lectures/homeworks with your peers.
Outline

1 Logistics
   - Grading
   - Office Hours
   - Shared Lecture
   - Software

2 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):
        ...

MacBook-Pro:~ Reuven$ python
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World!"
Hello World!

- 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><code>print &quot;hello world!&quot;</code></td>
<td><code>print(&quot;hello world!&quot;)</code></td>
</tr>
<tr>
<td><code>1/2 = 0</code></td>
<td><code>1/2 = 0.5</code></td>
</tr>
<tr>
<td><code>1 // 2 = 0</code></td>
<td><code>1 // 2 = 0</code></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.
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 be 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_undercore
  - Constants: UPPER_WITH_UNDERSCORE
- \( x = 1 \)
- \( y = x \)
- \( x = 'a' \)
Binding

- $x = 1$
- $y = x$
- $x = 'a'$
Binding

- $x = 1$
- $y = x$
- $x = 'a'$
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 """" -> ' ' and """"
- Triplequote for multiline string
- Can concat strings by separating string literals with whitespace
- All strings are unicode
- 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

- Immutable sequence of numbers
  - `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
- 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 *`