CIS192: Python Programming

Introduction

Harry Smith

University of Pennsylvania

January 18, 2017
Outline

1 Logistics
   - Rooms and Office Hours
     - Grading
     - Class Materials

2 Python
   - What is Python?
   - The Basics

3 Wrapping Up the First Class
What’s CIS 192?

- CIS 19X courses
  - 0.5 Credits each
  - Designed to teach practical skills
  - **Intended to be lightweight and highly functional.**

- CIS 192: Python Programming
  - Powerful scripting language used in academia and industry
  - Simple to read and write

- Take this class if
  - You have some programming experience
  - You are relatively new to Python
Class: CIS 192 Python Programming
  ▶ Listed as CIS 192 201
  ▶ Room: Towne 309
  ▶ Time: Wednesdays, 12:00 - 1:30pm

Instructor: Harry Smith
  ▶ Undergrad Junior in Logic
  ▶ Call me Harry! (Not a professor!)

TAs
  ▶ Alex Frias
Class: CIS 19X Shared Lecture
- Room: Towne 100
- Time: Tuesdays, 6:00 - 7:30pm
- General, useful info: command line intro, Git, overview of the Internet, HTML/CSS (new this year!)

Instructor: Joe Devietti (Listed as Swapneel Sheth)
Outline

1 Logistics
   • Rooms and Office Hours
   • Grading
   • Class Materials

2 Python
   • What is Python?
   • The Basics

3 Wrapping Up the First Class
Grade Breakdown

- **Homework: 70%**
  - 1 per week some time before the end of the semester
  - Programming assignments
  - These will range from collections of exercises on a week’s material to mini-projects with time given to work in class with instructor assistance.
  - Will be looking for a lot of feedback here throughout the semester.

- **Final Project: 25%**
  - Anything you want
  - Individually or in a pair

- **Participation: 5%**
  - Attendance/Piazza/Office Hours

- **Homework Info**
  - Drop one homework (lowest grade)
  - Late homeworks accepted up to 24 hours late with 20% penalty.
Homeworks

- Python 3 (3.42 or latest stable)
- Submit on Canvas!
- Graded for correctness (80%) and style (20%)
- Due Sunday nights at 11:59pm
- HW1 due next Sunday

Academic Integrity
- The Office of Student Conduct
- Don’t copy-paste code from other people
- Don’t have mid-level discussions
  - High-Level: What are the pros/cons of using Python for X?
  - Low-Level: What is the syntax for decorators?
  - Mid-Level: How did you do HW 1?

If you are going to do "research" for a problem...
- Do not copy and paste code from a StackOverflow response
- Do not look at or copy material that directly solves a question
- In the case of reasonable online research, **you must absolutely cite sources.**
- When in doubt, consult me or Alex.
Outline

1 Logistics
   - Rooms and Office Hours
   - Grading
   - Class Materials

2 Python
   - What is Python?
   - The Basics

3 Wrapping Up the First Class
- No text book!
- Python Official Documentation
- In-class slides and code (available on CIS 192 website)
- Piazza
Programming Environment

- Unix system recommended
  - Ubuntu Virtual Machine / Eniac
- Editor
  - Can use anything you want (Sublime Text, Atom, vim, emacs, etc.)
  - IDEs also available: PyDev for Eclipse, PyCharm
  - *Set your editor to interpret tabs as four spaces*
    - Python is whitespace-sensitive
- Limited support for students not using the above.
Outline

1 Logistics
   • Rooms and Office Hours
   • Grading
   • Class Materials

2 Python
   • What is Python?
   • The Basics

3 Wrapping Up the First Class
Easy to Learn

I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"

I DUNNO... DYNAMIC TYPING? WHITESPACE?
COMING 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.
Easy to Use

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—
Easy to Abuse

Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 5 2014, 20:42:22)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

>>> int(3.00)
3

>>> int = 5
>>> int(3.00)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not callable

>>>
History

- Designer: Guido van Rossum
  - Benevolent Dictator For Life (BDFL)
- Multi-Paradigm: Object-Oriented, Functional, Imperative...
- Strongly and Dynamically Typed
- Whitespace delimited blocks
- Garbage Collected
Philosophy

- **The Zen of Python**
  - Beautiful is better than ugly
  - Explicit is better than implicit
  - Simple is better than complex
  - Complex is better than complicated
  - Readability Counts

- **Other ideas**
  - There should be one obvious way to do it
  - Clarity over marginal efficiency
  - We’re all consenting adults here
Outline

1. Logistics
   - Rooms and Office Hours
   - Grading
   - Class Materials

2. Python
   - What is Python?
   - The Basics

3. Wrapping Up the First Class
REPL

- Read Evaluate Print Loop (a.k.a. an interpreter)
- Type “Python” at the terminal

```bash
ryin@Raymonds-MBP:~$ python3
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 5 2014, 20:42:22)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print('sup everyone')
sup everyone
>>> 5 + 6
11
>>> 
```

- Test out language behavior here
- Get information with `dir()`, `help()`, `type()`
Identifiers, Names, Variables

- All 3 mean the same thing
- Variable naming convention
  - Functions and variables: lower_with_underscore
    - `my_num = 5`
  - Constants: UPPER_WITH_UNDERSCORE
    - `SECONDS_PER_MINUTE = 60`
- \( x = 1 \)
- \( y = x \)
- \( x = 'a' \)
Binding

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

- \( x = 1 \)
- \( y = x \)
- \( x = 'a' \)

Diagram:

- Node \( X \) connected to \( x = 1 \)
- Node \( Y \) connected to \( x = 'a' \)
Objects

- Python treats all data as objects
- Identity
  - Memory address
  - Does not change
- Type
  - Does not change
- Value
  - Mutable: value can be changed (e.g. `[1, 2]`)
  - Immutable: value cannot be changed after creation (e.g. `(1, 2)`)
- Equality
  - Use `is` for referential equality (do x and y point to the same object?)
  - Use `==` for structural equality (are x and y equal based on object’s `__eq__` method?)
Types

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

- **True** and **False**
- **Boolean operators:** `and`, `or`, `not`
- Any object can be tested for truth value for use in conditionals, or as operands of the above Boolean operations.
- **"Falsy"**
  - `None`
  - `0`
  - `0.0`
  - Any empty string/sequence/collection ([ ], (), etc.)
- Almost everything in Python is **True**.
Comparisons

- `<`, `<=`, `>`, `>=`, `==`, `!=`, `is`, `is not`
- Chainable binary operators
  - This means that `x < y <= z` and `x < y and y <= z` are equivalent.
- `x is y simplifies to True when x and y refer to the same object.`
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 characters: r"\n"
Conditionals

- One `if` block
- Zero or more `elif` blocks
- Zero or one `else` block
Sequences

- **Immutable**
  - Strings, Tuples
- **Mutable**
  - Lists
- **Operations**
  - `len()`
  - Indexing
  - Slicing
  - `in`
  - `not in`
Range

- Immutable sequence of numbers
- \texttt{range(stop)}, \texttt{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 (`for item in my_list:`)
  - Iterate over an object

- While loops (`while some_condition:`)
  - Continues as long as condition holds

- Both
  - `else`: executes after loop finishes
    - For loops: the sequence has been exhausted
    - While loops: the condition has been made False.
  - `break`: stops the loop and skips the else clause
  - `continue`: starts the next iteration of the loop
Functions

- Functions are first class
  - They’re objects, too!
  - Can pass them as arguments
  - Can assign them to variables
- Define functions with a `def`
- `return` keyword to return a value
- `pass`: Use this to finish a function that is empty so it compiles.
- If a function reaches the end of the block without returning, it will return `None` (null)
Importing Modules

- 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 *`
Last thoughts

- Confirm that you are on Canvas/Piazza
- Fill out the anonymous survey on Piazza/in your email
- If you are registered for the class and are preparing to drop, please come talk to me
- If you are not registered for the class, please come talk to me
- I have Office Hours immediately following this class in DRL 2W11
- Homework 1 to be released shortly (or is already released) and will be due January 28.