Introduction to Python

CSCI 4448/5448: Object-Oriented Analysis & Design

Lecture 8

Learning Objectives

- Students will be able to…
  - Understand why Python is suited to OO designs
  - Find support for their own Python language exploration
  - Use Python for class projects and future designs

Why Python?

- Python: Development Speed
- Python's syntax emphasizes code readability – estimated that 10% of code required for a given application vs. C
- Built-in memory management
- Python has vast library support for UI and M2M/IoT communication protocols
- Python's interactive interpreter allows rapid test of features while programming
- Python has a complete (?) implementation of object-oriented design elements
- IEEE 2018 language survey places Python (#1) above C (#4) [1]
- C++ and Java are #2 and #3

“Pythonic” Code Style

- When working in Python, and looking at examples, you will see the term “Pythonic”
- Pythonic is code that is as simple and readable as possible
- Pythonic code follows the basics of Python code style [4]
- Consider using the pep8 tool to check your code for style issues [5]
- More information on Python style is available [6]

Zen of Python: Tim Peters [7]

- Beautiful is better than ugly.
- Explicit is better than implicit.
- **Simple is better than complex.**
- **Complex is better than complicated.**
- Flat is better than nested.
- Sparse is better than dense.
- **Readability counts.**
- Special cases aren't special enough to break the rules.
  Although practicality beats purity.
- **Errors should never pass silently.**
- Unless explicitly silenced.

Zen of Python: Tim Peters [7] continued…

- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch*.
- Now is better than never.
- Although never is often better than *right* now.
- **If the implementation is hard to explain, it's a bad idea.**
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

- You can see the Zen of Python in a Python interpreter by entering:
- >>> import this
- *Dutch – refers to Guido van Rossum, Python’s creator, who is Dutch.

Python: Functional, Procedural…

- Functional – Direct evaluation of mathematical functions (like R or Mathematica):

```
>>> 1+2
3
```

- Procedural – Structured programs based on modules and procedure calls (like C):

```
def add_me(x,y):
     print(x+y)
add_me(3,4)
# Output: 7
```

Python: …and Object-Oriented

Object-oriented – Data operations via objects with defined methods; inheritable classes (like Java and C++):

```
class MyMath:
      input_error = “Bad input”
    def error(self):
        print(self.input_error)
    def adder(x,y,self):
        print(x+y)

m = MyMath()
print(m.input_error)
# Output: Bad input
print(m.error())
# Output: Bad input
print(m.adder(3,4))
# Output: 7
```

Python: …and Object-Oriented

Object-oriented – Data operations via objects with defined methods; inheritable classes (like Java and C++):

```
class MyMath:
      input_error = “Bad input”
    def error(self):
        print(self.input_error)
    def adder(x,y,self):
        print(x+y)

m = MyMath()
print(m.input_error)
# Output: Bad input
print(m.error())
# Output: Bad input
print(m.adder(3,4))
# Output: 7
```

Object-Oriented Python

- Unlike Java, Python can handle multiple inheritance (see code)
- Duplicate method names are parsed based on inheritance order to prevent Diamond issue (linearized inheritance chain)
- Some issues with Python OO:
  - Type safety
  - Changing objects or attributes at runtime
  - Data privacy
  - __new__, __init__, __del__ issues
  - No interfaces (but Python 3 introduced annotations for abstract methods, etc.)

```
# Python example to show working of
# multiple inheritance
class Base1:
    def __init__(self):
        self.str1 = “I’m base 1"
        print "Base1"

class Base2:
    def __init__(self):
        self.str2 = “I’m base 2"
        print "Base2"

class Derived(Base1, Base2):
    def __init__(self):

        # Calling constructors of Base1
        # and Base2 classes
        Base1.__init__(self)
        Base2.__init__(self)
        print "Derived"

    def printStrs(self):
        print(self.str1, self.str2)
```

Common Python Syntax

From the author of Python Crash Course [8]:
A set of Python syntax cheat sheets for most Python coding

Beginner's Python Cheat Sheet

Variables and Strings

Variables are used to store values. A string is a series of characters, surrounded by single or double quotes.

Hello world

```
print("Hello world!")
```

Hello world with a variable

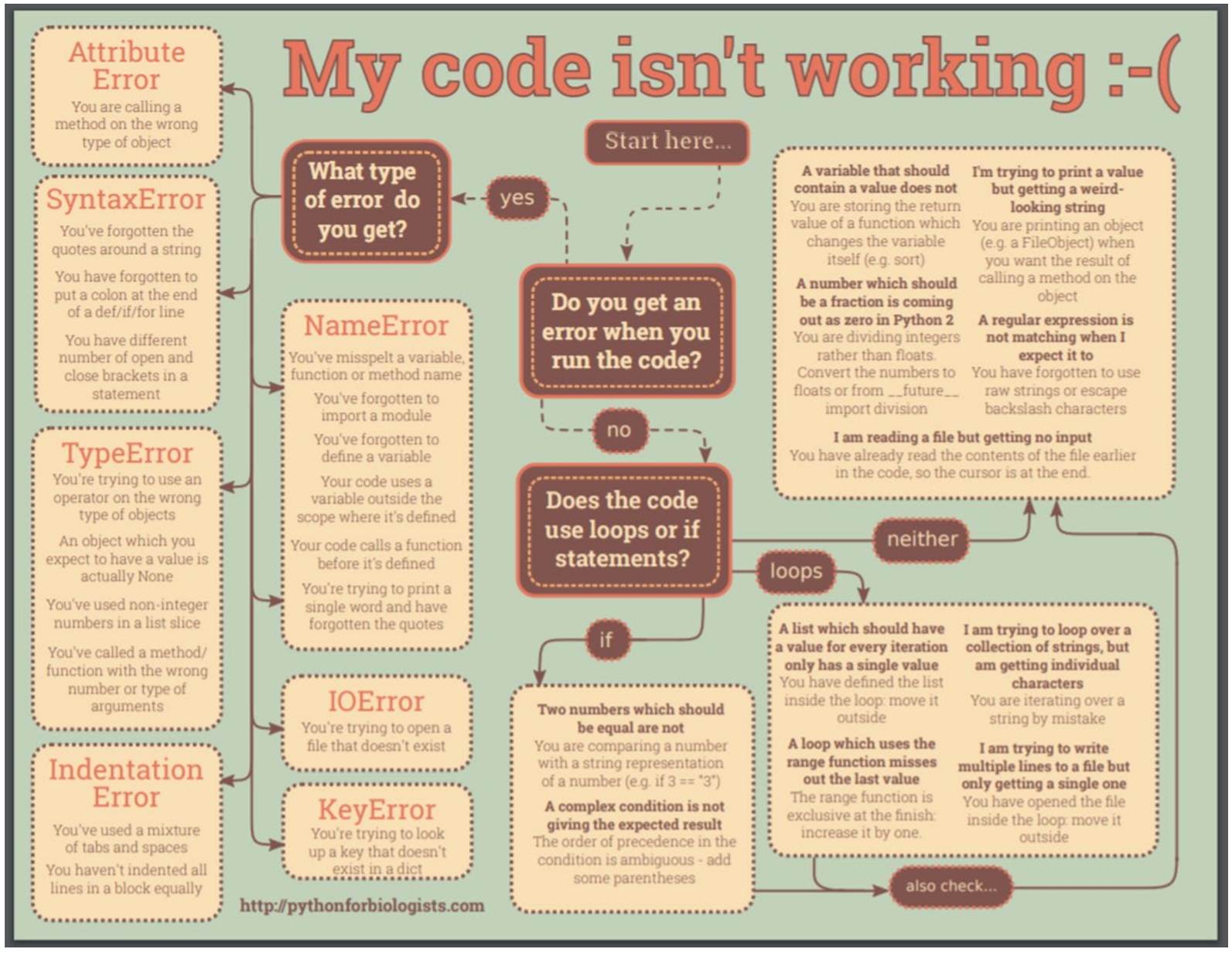
```
msg = "Hello world!"
print(msg)
```

Concatenation (combining strings)

```
first_name = 'albert'
last_name = 'einstein'
full_name = first_name + ' ' + last_name
print(full_name)
```

Common Python Errors

From Python for Biologists [9]:
A nice site for an introductory Python tutorial as well



Useful Python Libraries

- Requests – “HTTP for Humans”, eases messaging via HTTP packets [10]
- Pillow – a fork of the PIL image manipulation library [11]
- Beautiful Soup – parser for XML and HTML [12]
- Twisted – event-driven networking and web server [13]
- NumPy, SciPy, SymPy – math/scientific libraries [14][15][16]
- Matplotlib – 2D plotting library [17]

- Python Rule of Thumb – There’s already a library for it.
- More suggestions at 20 Python Libraries You Can’t Live Without [18]

Pip

- Pip is a tool for installing and managing Python packages, such as those found in the Python Package Index (aka PyPI [19])
- Typically, pip is used to install packages into a python environment:
- Pip can also be used to create packages you want to provide to the larger Python community
- Wheels (distribution packages) are relatively new to Python, replacing older “eggs” [20]
- Most Python packages have been converted to Wheels [21]

```
Usage: pip [options]

Commands:
install     Install packages.
uninstall   Uninstall packages.
freeze      Output installed packages in
            requirements format.
list        List installed packages.
show        Show information about installed
            packages.
search      Search PyPI for packages.
wheel       Build Wheel archives for your
            requirements and dependencies.
help        Show help for commands.
```

Python Test Frameworks

- One accepted approach to professional software development is test-first (sometimes test-with) programming
- The goal is to create suites of unit test cases that can be run against code whenever it changes to ensure issues are not introduced
- In a continuous development environment, using a tool such as Jenkins, such unit tests are included in each build execution to ensure project health
- Python supports several test frameworks, including:
  - pytest, nose, and unittest
- For more information and other test frameworks, including for GUIs, see [22]

PyTest Example

- In this example, a test case is defined for func at a value of 3
- The assert clause is checked by the PyTest run, which fails because the assert = 4, not 5 as specified
- Such testing can be performed for function bounds and abnormal values
- The creation of effective unit tests is a skill
- TDD is common in many professional development environments…

```
# content of test_sample.py
def func(x):
    return x + 1
def test_answer():
    assert func(3) == 5
```

```
$ pytest
======= test session starts ========
platform linux -- Python 3.x.y, pytest-
3.x.y, py-1.x.y, pluggy-0.x.y
rootdir: $REGENDOC_TMPDIR, inifile:
collected 1 item

test_sample.py F

======= FAILURES ========
_______ test_answer ________

def test_answer():
> assert func(3) == 5
E assert 4 == 5
E + where 4 = func(3)

test_sample.py:5: AssertionError
======= 1 failed in 0.12 seconds ========
```

Practical Python Issues

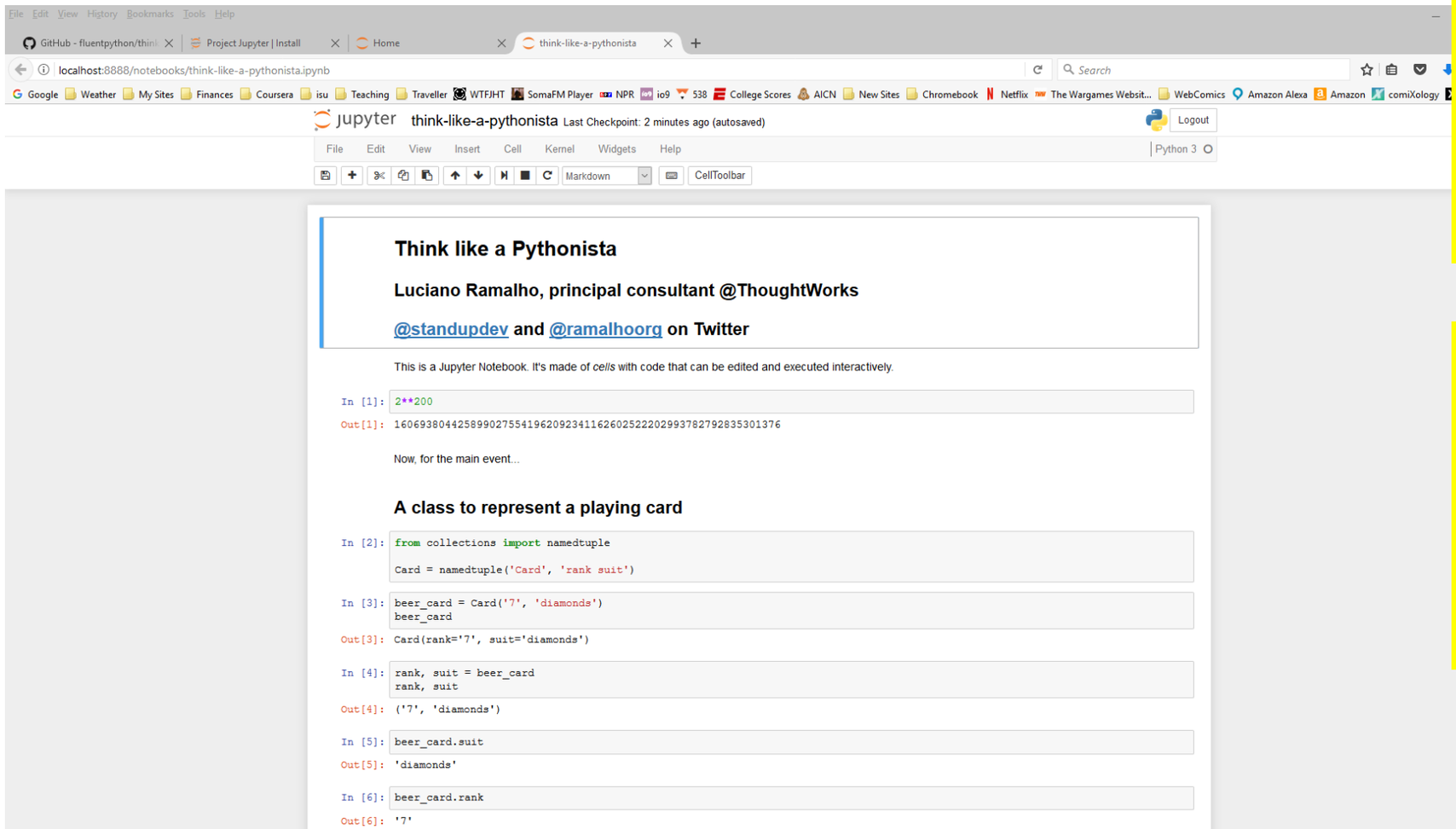
- Include a shebang as the first line in a Python program to run it directly from a Linux shell
- `#!/usr/bin/env python3`
- Now the program can be made executable to run directly from the shell
- `$ ./program.py`
- Use Python docstrings, keywords like __author__ and __copyright__, and comments to document your code for yourself and others
- Examples will be found in homework and projects

- You'll see references to both Python 3.x and 2.x environments
- Simple answer now: Use Python 3.x, differences from 2.7 are minor
  - Python is up to 3.7.4, but generally any 3.5.x or 3.6.x should be fine

If you haven’t used Python…

- Python is NOT required for OOAD, I will show some code examples in it
- Past this introduction, I will not teach Python coding in class
- If you know other languages, learning to use Python will be easy for you
- Many available environments and distributions for Python
- I recommend two Python tools for your laptop in particular
  - Anaconda – A Python distribution with many standard libraries pre-loaded (including Jupyter below) [23]
  - Jupyter Notebooks – An interactive notebook style editor for executing code snippets in development [24]

Jupyter – Python IDE in a browser



The Jupyter environment is fun for running small Python snippets as you learn the language. Juypter saves your workspace into . Ipynb files.

Python can get very complex and obscure:
The book, Fluent Python by Ramalho, goes into some very esoteric language elements [11]

Python References

- http://dowell.colorado.edu/education-python.html
  - A CU-based Python tutorial
- Dan Bader's tutorials
  - https://dbader.org/
- Google's Python class
  - https://developers.google.com/edu/python/
- Many others…

- If you haven't used Python before, take the opportunity to write a little code and get used to the language… It's fun!

Next Steps

- Piazza, Canvas, Book, …
- Project 1!
- Grad project topic paper!
- Quiz due this Wednesday! See Canvas for details…
  - A new one for this week's topics will go up Fri/Sat…
- Upcoming lectures
  - Next up UML & OO, then OO Patterns…
- Class staff is ready to help!

References

[1] https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages (2-3 not used)
[4] https://docs.python.org/3/tutorial/controlflow.html#intermezzo-coding-style
[5] https://www.python.org/dev/peps/pep-0008/
[6] http://docs.python-guide.org/en/latest/writing/style/
[7] https://www.python.org/dev/peps/pep-0020/
[8] https://github.com/ehmatthes/pcc/releases/download/v1.0.0/beginners_python_cheat_sheet_pcc.pdf
[9] https://pythonforbiologists.com/29-common-beginner-errors-on-one-page/
[10] http://www.python-requests.org/en/master/
[11] https://github.com/python-pillow/Pillow
[12] https://www.crummy.com/software/BeautifulSoup/
[13] https://twistedmatrix.com/trac/
[14] http://www.numpy.org/
[15] https://www.scipy.org/
[16] https://www.sympy.org/en/index.html
[17] https://matplotlib.org/
[18] https://pythontips.com/2013/07/30/20-python-libraries-you-cant-live-without/
[19] https://pypi.python.org/pypi
[20] https://wheel.readthedocs.io/en/latest/
[21] https://pythonwheels.com/
[22] https://wiki.python.org/moin/PythonTestingToolsTaxonomy#GUI_Testing_Tools
[23] https://www.anaconda.com/download/
[24] http://jupyter.org/install.html
[25] https://github.com/fluentpython