File Formats and Standalone Python Programs

Jennifer Helsby, Eric Potash

Computation for Public Policy

Lecture 7: January 26, 2016

`computationforpolicy.github.io`

Announcements

- HW2 is online: `https://computationforpolicy.github.io/assignments/02.html`
- HW1 is graded
  - Average: 95
- Slight modification of schedule:
  - Census and survey data lecture on Thursday

Today

- Dealing with other file formats than CSV
    - JSON, Excel, Stata, SAS, ...
- Writing standalone Python code
    - Modules and packages
- Debugging
- PEP8 style

File Formats

Excel

- Read Excel files into pandas DataFrames:

```
df = pd.read_excel(‘my_file.xls’, sheetname=’Sheet1’)
```

- Write pandas DataFrames as Excel sheets (do not recommend):

```
pd.to_excel(‘my_file.xls’, sheetname=’Sheet1’)
```

Stata

- Write a dataframe df into a Stata file:

```
df.to_stata('stata.dta')
```

- Read a pandas dataframe df from a Stata file:

```
df = pd.read_stata('stata.dta')
```

SAS

- Read a pandas dataframe from a SAS xport (.XPT) file:

```
df = pd.read_sas('sas_xport.xpt')
```

- No support currently for writing to SAS

Object Serialization

- Process of cloning data structures directly to a file
- Formatted such that it can be reconstructed later
- Usually language specific

Pickle

- Python standard format for object serialization
- Pickle file extension is usually .pkl
- Uses the `pickle` standard library module:

```
import pickle
```

Why use Pickle

- Advantage:
  - Save arbitrary Python objects (e.g. the result of a time-intensive analysis) for later

- Disadvantage:
  - Not good for transferring between languages

Pickling

```
import pickle

student_names = [‘Alice’, ‘Bob’, ‘Eve’]

with open(‘savemahstuff.pkl’, ‘wb’) as f:

    pickle.dump(student_names, f)
```

Unpickling

```
import pickle

with open(‘savemahstuff.pkl’, ‘rb’) as f:

    student_names = pickle.load(f)
```

Pickling Protip

```
to_save = {‘student_names’: student_names,
           ‘data_assignment1’: data_assignment1}’
with open(‘savemahstuff.pkl’, ‘wb’) as f:
    pickle.dump(to_save, f)
```

JSON

- Javascript Object Notation
- Saves data in a human-readable format
- Similar in syntax to a Python dict
- Can store:
  - ints, floats, arrays, None (`null`), bool (`true`, `false`), strings (“”)
- Commonly used in web programming

JSON: Example

```
{"Mustafa Abdul Qawi Abdul Aziz al Shamyri": {"tweet": "Mustafa
Abdul Qawi Abdul Aziz al Shamyri from Yemen has been in Guantanamo
Bay for 13 years four months.", "country": "Yemen",
"time_in_gitmo": "13 years four months."},

"Hamidullah": {"tweet": "Hamidullah from Afghanistan has been in
Guantanamo Bay for 11 years 11 months.", "country": "Afghanistan",
"time_in_gitmo": "11 years 11 months."}}
```

Why use JSON

- Advantage:
    - Saving and sharing data between languages, especially in web programming contexts

- Disadvantage:
    - Simple structure; can’t store complicated data objects

How to serialize data into JSON format

```
import json

student_names = [‘Alice’, ‘Bob’, ‘Eve’]

with open(‘savemahstuff.json’, ‘wb’) as f:

    json.dump(student_names, f)
```

How to serialize data into JSON format

```
import json

with open(‘savemahstuff.json’, ‘rb’) as f:

    student_data = json.load(f)
```

DataFrames to JSON and vice versa

- Reading JSON files:

```
df = pd.read_json('myfile.json')
```

- Writing dataframe to JSON:

```
df.to_json('dataframe.json')
```

Other data formats

- XML
- For large datasets:
    - Loading entire files on the hard drive can become slow
        - Use a database (future lectures)

Creating Python Programs, Modules, Packages

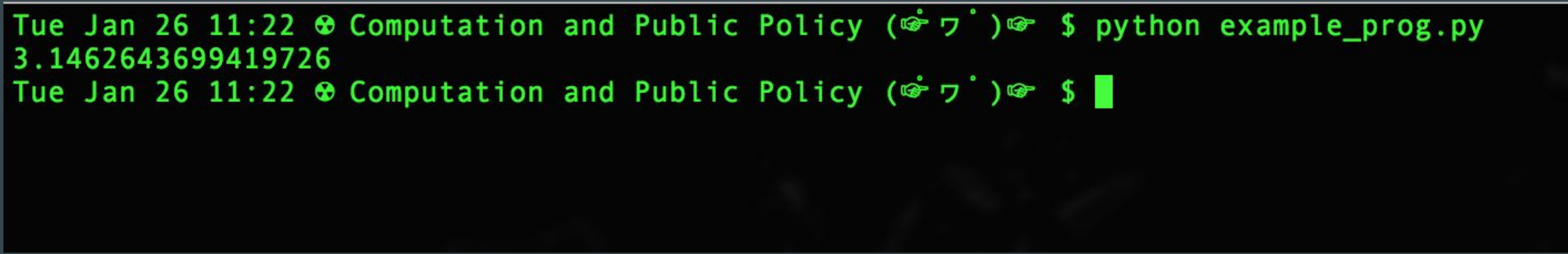
Why create standalone programs and modules

- Create standard tools
- Easier to re-run
- e.g. Create an analysis pipeline that you can run directly from the command line

Example Python Program

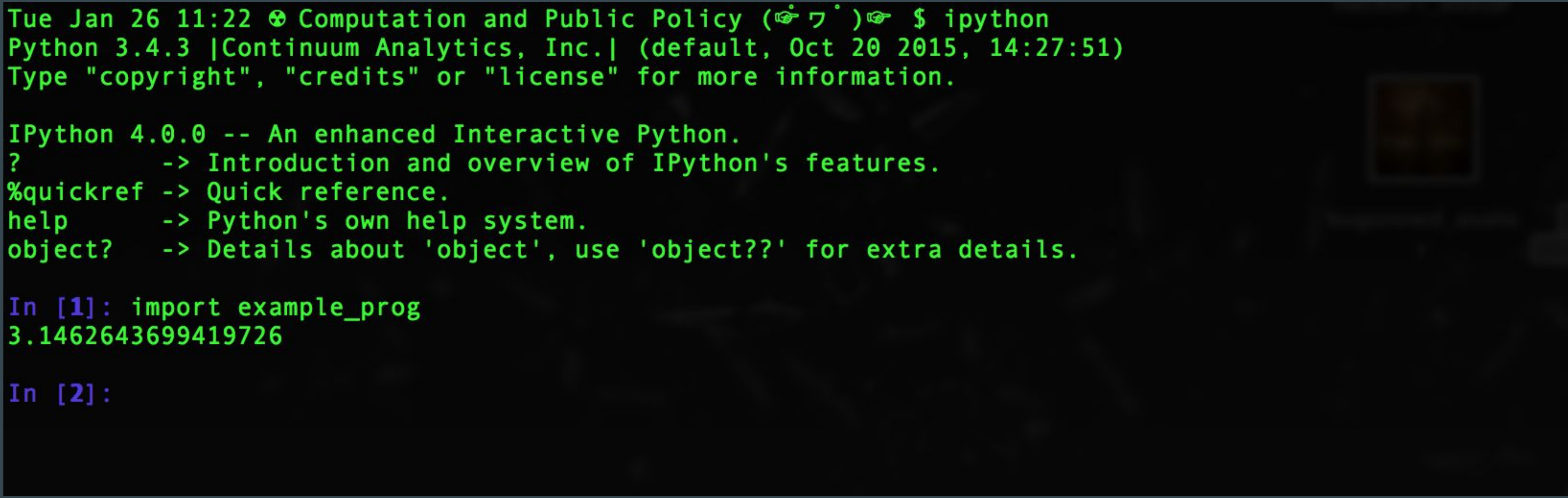
```
import math

def sum_of_sqrts(in_nums):
    sum_tot = 0
    for num in in_nums:
        sum_tot += math.sqrt(num)
    return sum_tot

to_calc = [2, 3]
print(sum_of_sqrts(to_calc))
```

Run from Command Line



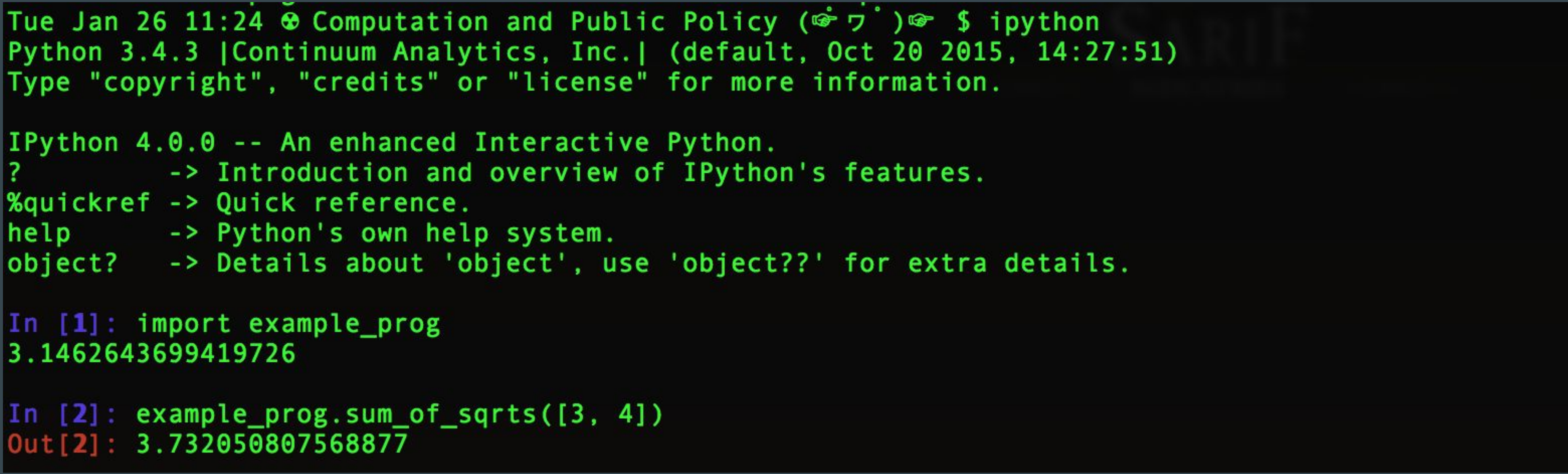
```
Tue Jan 26 11:22 ☢ Computation and Public Policy (☞ﾟヮﾟ)☞ $ python example_prog.py
3.1462643699419726
Tue Jan 26 11:22 ☢ Computation and Public Policy (☞ﾟヮﾟ)☞ $ █
```

Run from Python Interpreter



```
Tue Jan 26 11:22 ☢ Computation and Public Policy (☞ﾟヮﾟ)☞ $ ipython
Python 3.4.3 |Continuum Analytics, Inc.| (default, Oct 20 2015, 14:27:51)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import example_prog
3.1462643699419726

In [2]:
```

Use as Library

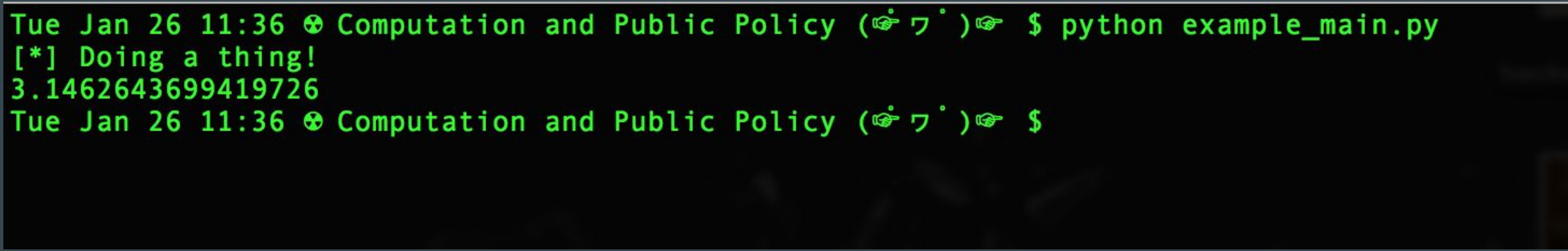


```
Tue Jan 26 11:24 ☢ Computation and Public Policy (☞ ﾜ ﾟ)☞ $ ipython
Python 3.4.3 |Continuum Analytics, Inc.| (default, Oct 20 2015, 14:27:51)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import example_prog
3.146264369941972б

In [2]: example_prog.sum_of_sqrts([3, 4])
Out[2]: 3.732050807568877
```

Most Common Structure of a Python Program

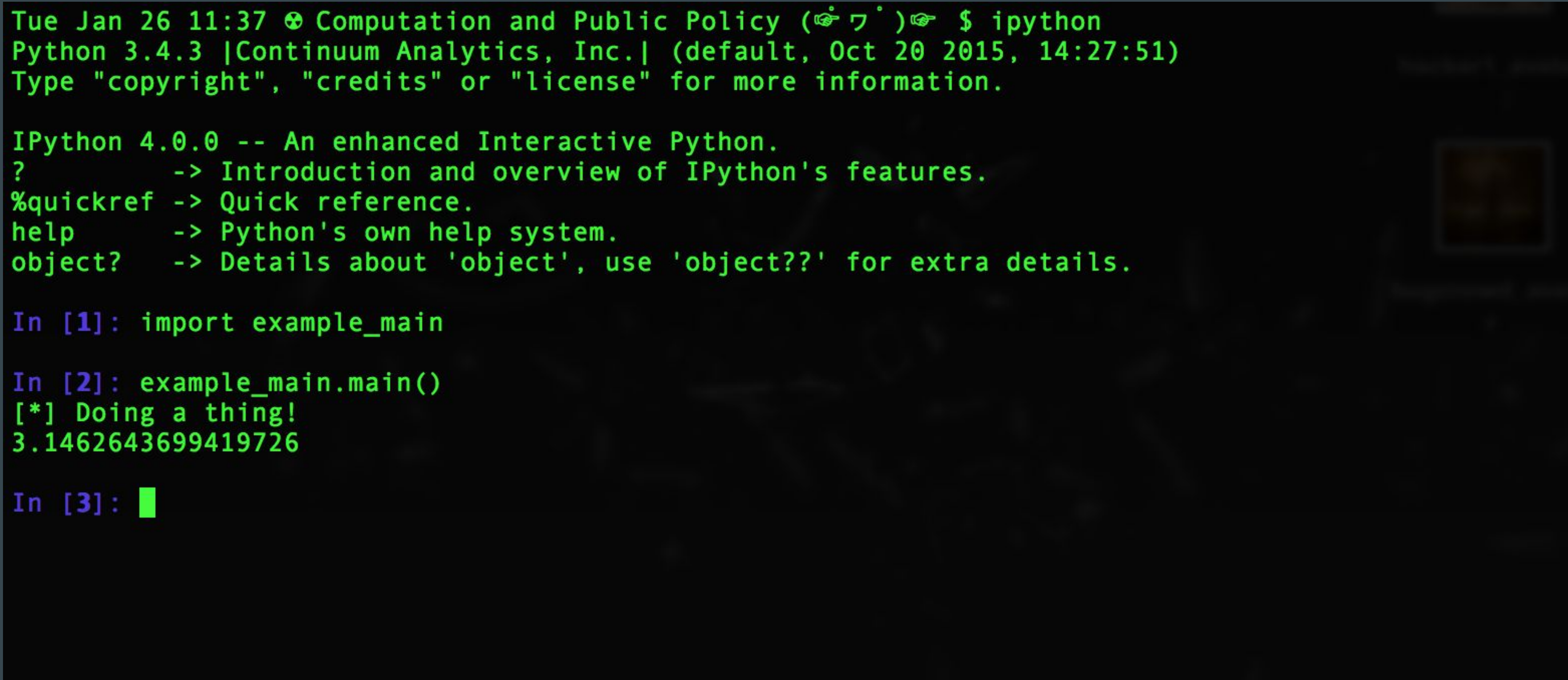
```
import math

def sum_of_sqrts(in_nums):
    sum_tot = 0
    for num in in_nums:
        sum_tot += math.sqrt(num)
    return sum_tot

def main():
    print('[*] Doing a thing!')
    to_calc = [2, 3]
    print(sum_of_sqrts(to_calc))

if __name__=='__main__':
    main()
```

From the Command Line



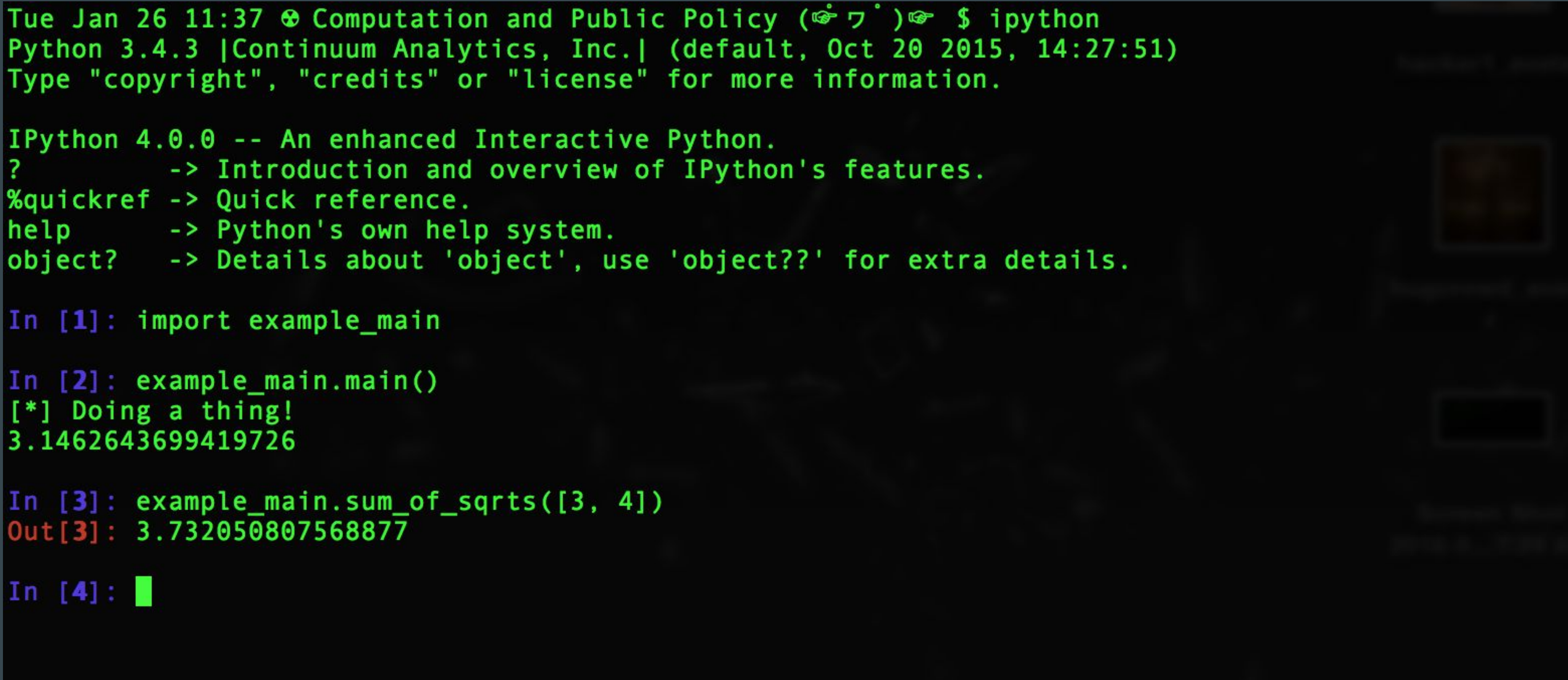
```
Tue Jan 26 11:36 ☢ Computation and Public Policy (☞ﾟヮﾟ)☞ $ python example_main.py
[*] Doing a thing!
3.1462643699419726
Tue Jan 26 11:36 ☢ Computation and Public Policy (☞ﾟヮﾟ)☞ $
```

From the Python Interpreter



```
Tue Jan 26 11:37 ☢ Computation and Public Policy (☞ﾟヮﾟ)☞ $ ipython
Python 3.4.3 |Continuum Analytics, Inc.| (default, Oct 20 2015, 14:27:51)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import example_main

In [2]: example_main.main()
[*] Doing a thing!
3.146264369941972б

In [3]: 
```

Use as Library



```
Tue Jan 26 11:37 ☢ Computation and Public Policy (☞ﾟヮﾟ)☞ $ ipython
Python 3.4.3 |Continuum Analytics, Inc.| (default, Oct 20 2015, 14:27:51)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import example_main

In [2]: example_main.main()
[*] Doing a thing!
3.1462643699419726

In [3]: example_main.sum_of_sqrts([3, 4])
Out[3]: 3.732050807568877

In [4]: 
```

Testing Name

```
if __name__ == '__main__':
    print('This program is being run by itself')
else:
    print('This program is being imported from another module')
```

Testing Name

```
if __name__ == '__main__':
    print('This program is being run by itself')
else:
    print('This program is being imported from another module')
```

```
$ python3 using_name.py
This program is being run by itself

$ ipython3
...
[1] import using_name
I am being imported from another module
```

import statements

- Avoid: `from math import *`

  ```
  result = sqrt(to_calc)
  ```

- OK: `from math import sqrt`

  ```
  result = sqrt(to_calc)
  ```

- Ideal: `import math`

  ```
  result = math.sqrt(to_calc)
  ```

import statements

- `import blah` looks:
  - in the current directory for blah
  - Then in the Python path ($PYTHONPATH)
- ImportError will occur if blah is not found

Environmental Variables

- *Environmental variables* are shell variables that keep track of system settings
- Denoted by $ and uppercase: $EXAMPLE_VAR
- View how they are set with:

```
echo $EXAMPLE_VAR
```

Paths

- $PATH: list of directories the shell should look in when searching for programs
- $PYTHONPATH: list of directories Python should look in when searching for Python modules

Adding a new directory to your $PYTHONPATH

```
$PYTHONPATH=$PYTHONPATH:$HOME/mythesis

export PYTHONPATH
```

- Append these lines to `~/.bashrc`
- To have new changes to ~/.bashrc take effect:

```
source ~/.bashrc
```

Making a Python Package

- A *Python package* is a collection of modules
- Create a package by adding an __init__.py file

```
touch __init__.py
```

Folder Structure of a Good Python Project: `mythesis`

`mythesis/`	
`    mythesis/`	Source code
`    docs/`	Documentation
`    tests/`	Tests (if you write tests)
`    img/`	Images
`    README.md`	Main readme file

in the source code directory of `mythesis`

```
mythesis/mythesis/
    __init__.py
    clustering.py
    skyplots.py
```

- Top dir is in your Python path
- Do:

```
import mythesis.clustering

from mythesis import clustering
```

Getting new Python Packages

- On Python 2.7:

```
pip install <packagename>
```

- On Python 3.x:

```
pip3 install <packagename>
```

Python DeBugger (PDB)

Debuggers

- Step through your code as it executes to see where things are going wrong
- Put stops in and inspect memory

Python DeBugger (PDB)

- Import with:

  ```
  import pdb
  ```

- Put stops into your code with:

  ```
  pdb.set_trace()
  ```

- Run code as you usually do
- At stops, you will be dropped to the (Pdb) prompt

PDB Commands

- Single stepping: `s`
- Going to the next breakpoint: `c`
- Quitting PDB: `q`
- Help: `h`
- Print: `p myvar`
- List source: `l`
- Hitting enter will execute the last statement

Post-Mortem Debugging

- Drop to debugger if code hits a snag:

```
python -m pdb myscript.py
```

PEP8 Style

PEP8

- PEP8: style guidelines for Python code
- Not required syntactically
- Makes code easier to read

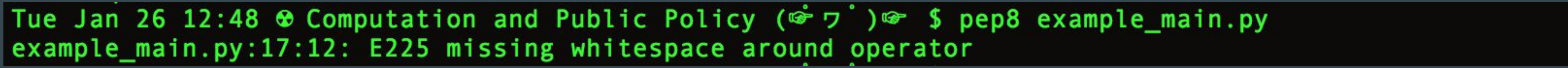
Installing

- Install command line tool:

```
pip install pep8
```

- Have this command check code:

```
pep8 mycode.py
```



```
Tue Jan 26 12:48 ☢ Computation and Public Policy (☞ﾟヮﾟ)☞ $ pep8 example_main.py
example_main.py:17:12: E225 missing whitespace around operator
```

Example PEP8 Rules

- For in-line comments, add two spaces before the comment:

```
num_districts = len(districts)  # Not including district 01
```

- Have whitespace around operators:

```
num_crimes = len(crime_list) + len(homicides_list)
```

- Have two lines between function declarations

A Further Resource

- Learn Python the Hard Way:

`learnpythonthehardway.org`