Programming II - Functions and Modules

GEOG 30323

September 8, 2015

Python refresher

Define…

> - Variables
> - Strings
> - Lists
> - Indexing and slicing

Functions

- If you find yourself doing the same thing frequently, it can be tedious to write the same code over and over!
- As such, you should use **functions** to make your code re-usable

---

- Functions are defined with a `def` statement followed by a colon (`:`)
- They include a series of **parameters** to which you supply **arguments**, which are like variables that operate within your function:

```
def myfunction(parameter1, parameter2, parameter3):
    """Your code goes here"""

# "Calling" the function:

myfunction(arg1, arg2, arg3)
```

Empty functions

- At a basic level, a function can be defined without parameters

Example:

```
>>> def drwalker():
...     print("The best professor!")
...
>>> drwalker()
The best professor!
```

Functions and parameters

Parameters are components of your function that can vary based on user input. Example:

```
>>> def make_big(x):
...     print(x.upper())
...
>>> make_big('abcdefg')
ABCDEFG
>>> make_big('the quick brown fox jumped over the lazy dog')
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG
>>> x = 'hijklmn'
>>> make_big(x)
HIJKLMN
```

The “return” statement

- The output of functions can be assigned to variables if a `return` statement is provided

Example:

```
>>> def add_five(x):
...     return x + 5
...
>>> y = add_five(10)
>>> y
15
```

Python and whitespace

- Python code is organized by **indentation** and **whitespace**
- After function definitions, code should be indented with **four spaces**. In the Jupyter Notebook (and other Python development environments), the Tab key represents four spaces, and it will indent your code automatically
- Code that is not indented properly will cause an error

```
>>> def add_six(x):
... return x + 6
  File "<stdin>", line 2
    return x + 6
         ^
IndentationError: expected an indented block
```

Comments

- *Comments*, preceded by a hashtag (`#`), can be included in your code but are not run
- *Commenting* your code is useful for describing what you are doing, or keeping experimental/old code you don't want to run during the development process

```
# The function 'divide_by_two' divides any number by two  <-- a comment

def divide_by_two(number):
    return number / 2

# This didn't work - remember whitespace issues

# def divide_by_two(number):
# return number / 2
```

Docstrings

- *Docstrings* can be used to describe what functions do
- Docstrings are enclosed in triple double quotes (`"""  """`) and are placed on the line following the function definition

```
def concat_numbers(num1, num2):
    """
    Return a concatenated string from two numbers.

    Parameters:
    -----------

    num1: The first number you'd like to concatenate

    num2: The second number you'd like to concatenate

    """

    return str(num1) + str(num2)
```

Ordering and named arguments

Arguments can be supplied to functions in two ways:

- “Unnamed” in the order specified
- “Named” in any order. Be careful, however, if you mix the two!

```
>>> concat_numbers(7, 3)
'73'
>>> concat_numbers(num2 = 7, num1 = 3)
'37'
>>> concat_numbers(7, num2 = 3)
'73'
>>> concat_numbers(num2 = 7, 3)
  File "<stdin>", line 1
SyntaxError: non-keyword arg after keyword arg
```

Scripts

- Python *script*: organized collection of code stored in a text file with a `.py` suffix
- Generally, scripts authored in a text editor or integrated development environment (IDE)
- IDE that comes packaged with Anaconda: **Spyder**

Scripts

- In IDEs, you can document workflows with scripts and run them interactively:

Editor - C:\Users\kylewalker\Dropbox\Teaching\Geographic data analysis\Preview\geog30323_preview.py

temp.py | geog30323_preview.py

```
1  import numpy as np, pandas as pd, seaborn as sb, matplotlib.pyplot as plt
2
3  names = ['state', 'gender', 'year', 'name', 'count']
4
5  tx = pd.read_csv('namesbystate/TX.TXT', names = names)
6
7  tx['per1000'] = 1000 * (tx['count'] / tx.groupby(['year', 'gender'])['count'].transform(sum))
8
9  tx2013 = tx.query('(year == 2013) & (gender == "F")')
10
11 tx2013['rnk'] = tx2013.per1000.rank(ascending = False)
12
13 tx25 = tx2013[tx2013['rnk'] < 26]
14
15 names = tx25.name.tolist()
16
17 txsub = tx[(tx['name'].isin(names)) & (tx['year'] > 1989)]
18
19 txsub = txsub[['year', 'name', 'per1000']]
20
21 txwide = pd.pivot_table(txsub, values = 'per1000', index = 'name', columns = 'year')
22
23 sb.heatmap(txwide)
24
25 plt.title("Female baby names per 1000 in the SSA data (TX)")
26 plt.xlabel("")
27 plt.ylabel("")
28
29 plt.show()
```

Scripts

- Scripts can also be run from the command line, e.g. `python myscript.py`
- Alternatively, scripts can be used in Python *modules* and *packages*

Modules and packages

- *Module*: file containing variables, functions, etc. that can be *imported* into a Python session with the `import` statement
- *Package*: directory of modules that perform similar tasks (e.g. data visualization, statistics, etc.)
- Thousands upon thousands of Python packages available - that do just about anything!

Creating your own module

Let's try this out!

The Python namespace

- When you declare variables, define functions, import modules, etc., you are adding objects to the *Python namespace*

- To remove objects from the Python namespace, use the `del` statement

Imports and the namespace

- Imported modules can be referenced in multiple ways:

```
# All of these are equivalent

import mymodule
mymodule.add_abc(mymodule.tcu)

import mymodule as mm

mm.add_abc(mm.tcu)

from mymodule import *

add_abc(tcu)
```

Built-in packages

- Many packages are included in `stdlib` , the standard library that ships with Python
- Popular modules: `re` for regular expressions; `os` for operating system functions; `random` for random-number generation; and many more. Full list: https://docs.python.org/2/library/ (https://docs.python.org/2/library/)

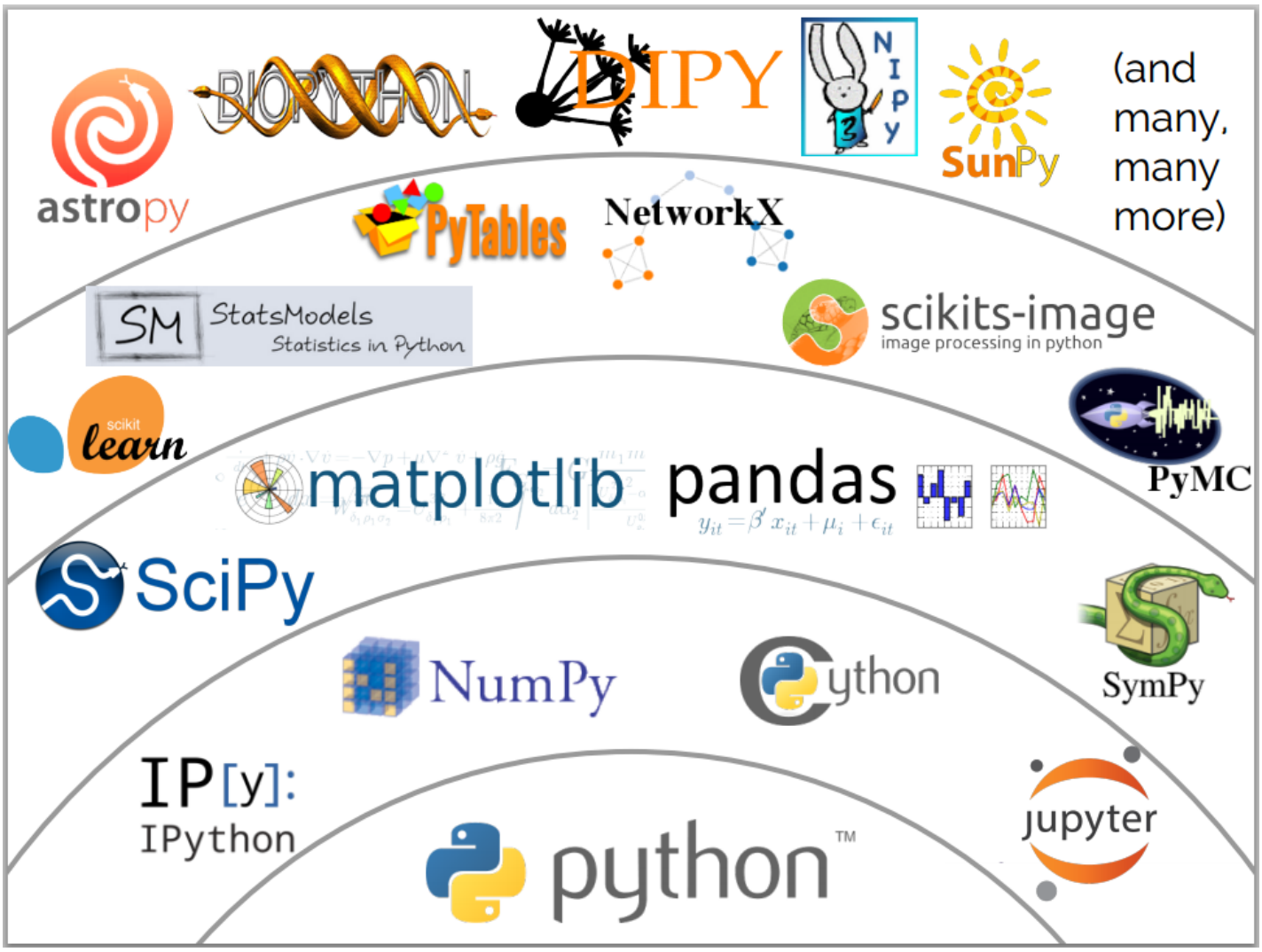
Conda

- Your Anaconda Python installation ships with over 330 packages for scientific computing
- New packages added to Anaconda can be installed with `conda install` from the command line
- To update all your packages, run `conda update --all`

PyPI and pip

- Third-party Python packages are generally found at the Python Package Index (PyPI): https://pypi.python.org/pypi (https://pypi.python.org/pypi)
- Generally, these packages can be installed from the command line with `pip` , the recommended tool for installing packages from PyPI
- Example usage: `pip install newpackage`
- To upgrade packages, use the `--upgrade` option: `pip install --upgrade newpackage`

The PyData ecosystem



Source: Jake VanderPlas, SciPy 2015 Keynote (https://speakerdeck.com/jakevdp/the-state-of-the-stack-scipy-2015-keynote)

GitHub

- Our course materials are hosted on GitHub - so what is GitHub, exactly?
- Let's take a tour: https://github.com/ (https://github.com/)

Open-source software

- Developed, in large part, by the user community
- Source code is *open*
- Software is generally free to purchase and update; some services under a “freemium” model
- Reliant on the user community for support

How open-source software works

- Example: my R package, `tigris` : https://github.com/walkerke/tigris (https://github.com/walkerke/tigris)