Python Workshop - ANU (CBE)

Jupyter Notebooks and Python Programming Basics and Fundamentals

Matthew McKay [matthew.mckay@anu.edu.au]

June 2017

Agenda ...

1. Jupyter Notebooks
2. Intro to Python Programming
3. Python Packages
    - matplotlib
    - NumPy
    - Pandas
4. Resources

Jupyter

Quick Review

1. Notebook Basics
2. Modal Editing
3. Running Code
4. Text Editor Features (Syntax Highlighting etc.)
5. Tab Completion
6. Object Introspection
7. Working with the shell
8. Working with Files
9. First Python Program

Everyone is able to run Jupyter?

Jupyter Notebooks

See notebook **intro-to-jupyter.ipynb**

Python Basics

1. Introductory Example
2. Basic Structure of a Python Program
3. Variables and Assignment
4. Data Types
    - Booleans
    - Numbers (integers, floats, fractions and complex numbers)
    - Strings
    - Bytes (and byte arrays)
    - Lists
    - Dictionaries
    - Sets
    - Tuples

Python Basics

See notebook **intro-to-python.ipynb**

Order of Operations

Python uses math conventions to determine the order of operations

1. Parentheses
2. Exponentiation
3. Multiplication and Division
4. Addition and Subtraction

Note: Operators that share precedence are then evaluated from left to right.

Using parentheses is good programming practice to improve clarity.

How are numbers stored by a computer?

Binary Number System

Is a base 2 number system with digits 0, and 1

Example: 1011_2

Very useful when using Boolean Logic (True and False).

Decimal Number System

Is a base 10 number system with digits: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9

Conversion:

$$1011_2 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 8 + 0 + 2 + 1 = 11_{10}$$

Others: Hexa-Decimal, Octal

Numbers and Precision ...

The way computers store numbers is important when using math

Computers have **finite** resources and can only represent ranges of values.

Example:

Signed 8-bit Integer can represent values up to $2^7 - 1$

Unsigned 8-bit Integer can represent values up to $2^8 - 1$

Python provides a number of conveniences when working with numbers unlike some other languages.

e.g. Integers are limited by memory

Floating Point Numbers

Floating point numbers are often approximate values with varying degrees of precision

Example:

```
In [8]: 1/3
Out[8]: 0.3333333333333333

In [9]: import math

In [10]: math.pi
Out[10]: 3.141592653589793

In [11]: format(math.pi, '0.20g')
Out[11]: '3.1415926535897931'

In [12]: format(math.pi, '0.2g')
Out[12]: '3.1'

In [13]: format(math.pi, '0.2f')
Out[13]: '3.14'
```

Floating Point Numbers

Comparison can be a bit tricky ...

Example:

```
In [15]: 1/3 == 1/3
Out[15]: True

In [16]: 0.3 == 0.1 + 0.1 + 0.1
Out[16]: False

In [17]: 0.1 + 0.1 + 0.1
Out[17]: 0.30000000000000004

In [18]: import math

In [19]: math.isclose(0.1+0.1+0.1, 0.3)
Out[19]: True
```

Python Floating Point Limitations

https://docs.python.org/3.5/tutorial/floatingpoint.html

Python Fundamentals

1. Syntax
   - Whitespace
   - Line Continuation
   - Commenting
2. Mutable and Immutable Objects
3. String Formatting
4. Conditional Logic
5. Iteration
6. List Comprehensions
7. Functions
8. Recursion
9. Exceptions

Python Fundamentals

See **intro-to-python-two.ipynb**

Syntax - Whitespace

Python uses whitespace as a **delimiter** for code blocks

```
for␣i␣in␣range(0,4,2):
␣␣␣␣print("i␣=␣%s"%i)␣␣␣#This␣code␣belongs␣to␣first␣loop
␣␣␣␣for␣j␣in␣range(2):
␣␣␣␣␣␣␣␣print("j␣=␣%s"%j)␣␣␣#This␣code␣belongs␣to␣second␣loop
␣␣␣␣␣␣␣␣print("i+j␣=␣%s"%(i+j))
```

and is therefore very important.

Syntax - Whitespace

Whitespace is however ignored in parentheses, brackets, and simple expressions

```
>>>m2␣=␣[[1,2,3],
␣␣␣␣␣␣␣␣␣[4,5,6],
␣␣␣␣␣␣␣␣␣[7,8,9]]
>>>m2
[[1,␣2,␣3],␣[4,␣5,␣6],␣[7,␣8,␣9]]
```

Syntax - Line Continuation

```
>>> a = 1 + 2 + 3 + 4 + 5 + 6 + 7
    + 8 + 9 + 10 + 11 + 12 + 13
  File ''<ipython-input-73-24d78ad3af91>'', line 3
    + 8 + 9 + 10 + 11 + 12 + 13
    ^
IndentationError: unexpected indent
```

A line can continue using a **backslash**

```
>>> a = 1 + 2 + 3 + 4 + 5 + 6 + 7 \
    + 8 + 9 + 10 + 11 + 12 + 13
>>> a
91
```

Syntax - Comments

Line Comments

```
# This is a Line Comment, Anything written here is ignored
a = 2
print(a) # This prints variable a (and is a bad comment)
```

Block Comments

```
"""
This is technically a docstring.
Anything written between these are ignored by the python interpre
but it is really a docstring!
"""
```

String Formatting

How do you construct strings in Python using variables?

```
>>> name = ”Matt”
>>> greeting = ”Hello ” + name + ”! Nice to meet you.”
>>> print(greeting)
Hello Matt! Nice to meet you.
```

This works - but can be more concise to use the % operator

String Formatting

Better to use string formatting

```
>>> name = "Matt"
>>> print("Hello %s! Nice to meet you."%name)
Hello Matt! Nice to meet you.
```

Pass a **tuple** for multiple arguments and will be unpacked across the string

```
>>> name = "Matt"
>>> day = "Tuesday"
>>> print("Hello %s! Nice to meet you.\nToday is %s"%(name,day))
Hello Matt! Nice to meet you.
Today is Tuesday
```

String Formatters

The following are the **basic** string formatters

%s	Format as a string. All types match a %s target
%d	Format as a Decimal (base-10 integer).
%f	Format as a Floating Point

There are others but these basics go a long way

Advanced String Formatters

Advanced String Formatters are available and can be very useful when working with **templates** etc.

```
>>> #Example Dictionary Substitutions
>>> reply = """
    Greetings...
    Hello %(name)s!
    Your age is %(age)s
    """
>>> values = {'name': 'Matt', 'age': 103}
>>> print(reply % values)
Greetings...
Hello Matt!
Your age is 103
```

Mutable and Immutable Objects

Mutable objects in python are those whose state can change

1. lists
2. dictionaries

Immutable objects in python are those whose state cannot be changed without the creation of a new object

1. numbers
2. strings
3. tuples

Conditional Logic

Using Boolean expressions to control the flow of a program

Relational Operators

```
x == y    # x is equal to y
x != y    # x is not equal to y
x > y     # x is greater than y
x < y     # x is less than y
x >= y    # x is greater than or equal to y
x <= y    # x is less than or equal to y
```

Reference: https://lectures.quantecon.org/py/python_essentials.html#comparisons-and-logical-operators

Conditional Statements

There are three main ways to write conditional logic expressions

```
if x > 0:
    print("x is > 0")
```

```
if x > 0:
    print("x is > 0")
else:
    print("x is <= 0")
```

```
if x > 0:
    print("x is > 0")
elif x == 0:
    print("x is = 0")
else:
    print("x is < 0")
```

Combining Conditions

Conditional statements can be combined using

1. **and**
2. **or**
3. **not**

```
if x >= 0:
    if x <= 10:
        print(”X is greater than 0 AND less than or equal to 10”)
```

can be written

```
x >= 0 and x <= 10
```

Iteration

The while loop:

```
#Collatz Conjecture
while n != 1:
    print(n)
    if n%2 == 0:
        n = n/2
    else:
        n = n*3+1
```

Iteration

The for loop:

```
for i in [1,'A',2,'B']:
    print(i)
    print(type(i))
```

List Comprehensions

See **intro-to-python-two.ipynb**

Functions

Functions are very useful for collecting a sequence of instructions

```
def collatz(n):
    seq = []
    while n != 1:
        seq.append(n)
        if n%2 == 0:
            n = n/2
        else:
            n = n*3+1
    return seq
```

Functions

Why use functions?

1. Incredibly useful to reuse code when performing the same operation many times
2. Can make your program much easier to read by breaking big tasks into many small tasks
3. Easier code to read is easier to debug
4. Can import your functions into other programs without rewriting them.

Functions: Basic Syntax

See **intro-to-python-two.ipynb**

Recursion

Functions and **Conditional Statements** can be combined to produce recursive loops when a function calls itself.

See intro-to-python-two.ipynb

References:

`http://openbookproject.net/thinkcs/python/english3e/recursion.html`

Python Libraries ...

Visualizing Data

1. matplotlib
2. line charts, bar charts, histograms
3. Other packages

Numeric Computing

1. The numpy package
2. Arrays
3. Matrices
4. Linear Algebra
5. Solving Systems of Equations

Working with Data (Pandas)

Basic Plotting

Most common library is **matplotlib**

`https://lectures.quantecon.org/py/matplotlib.html`

matplotlib

matplotlib is a versatile Python plotting package.

It is very general and capable of producing very simple to highly complex plots.

The programmer has a lot of control, but the tradeoff is convenience in defining the plots.

Some recent packages provide subsets of plots that are easier to implement quickly but are typically less versatile tools.

Simple Example

```
import matplotlib.pyplot as plt #Import plotting library
import numpy as np
x = np.linspace(0, 10, 200)    #Generate Some Data
y = np.sin(x)
plt.plot(x, y, 'b-', linewidth=2)  #Request a Plot
plt.show()
```

Using matplotlib

There are two primary ways to use the matplotlib library

1. **pylab** which provides a matlab like interface with convenient plotting functions
2. **object-oriented** interface which provides a high degree of control

Basic Plot Types

1. Line Charts
2. Histograms
3. Scatter Plots

See: **intro-to-matplotlib.ipynb**

Other packages

Some other packages include:

1. **ggplot** `http://ggplot.yhathq.com/`
2. **seaborn** `http://stanford.edu/~mwaskom/software/seaborn/`
3. **pandas** `http://pandas.pydata.org/`
4. **bokeh** `http://bokeh.pydata.org/en/latest/`
5. many others ...

Numerical Computing Reading

NumPy:

https://lectures.quantecon.org/py/numpy.html

Linear Algebra:

https://lectures.quantecon.org/py/linear_algebra.html

Documentation:

1. **NumPy** http://docs.scipy.org/doc/numpy/user/

The NumPy package

NumPy is the key package for scientific computing with Python.

NumPy provides:

1. N-dimensional array object (ndarray)
2. broadcasting functions
3. shape manipulation
4. sorting
5. basic linear algebra
6. random number simulation
7. Fourier transforms
8. basic statistical operations

and is widely used as a foundation by other packages, it is fast and it is stable.

Why is it fast?

Underlying implementation is written in C (mostly) or Fortran.

Python is used as a higher level environment to work productively while the computation occurs in libraries that are statically compiled and execute quickly.

There are other ways to get **speed** but this is the best place to start as it is the foundation of so many Python projects.

NumPy Arrays

```
import numpy as np
a = np.array([1.0, 2.0, 3.0])
b = np.array([[1.0, 2.0, 3.0],
              [4.0, 5.0, 6.0]])
```

How are ndarray objects different to standard Python sequences (lists)?

1. NumPy arrays are homogenous in data type (dtype)
2. NumPy arrays have fixed size when they are created

These limitations come with much improved performance for numerical computing.

NumPy Arrays

See **intro-to-numpy.ipynb** on GitHub

Matrices

https://lectures.quantecon.org/py/linear_algebra.html#matrices

Solving Systems of Equations

https://lectures.quantecon.org/py/linear_algebra.html#solving-systems-of-equations

Pandas

Pandas is the workhorse for data work in Python that is built on top of **NumPy**

Some things that Pandas is very good at:

1. Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
2. Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations
3. Hierarchical labeling of axes (possible to have multiple labels per tick)

Reference: http://pandas-docs.github.io/pandas-docs-travis/

Pandas - Continued

Operations:

1. Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
2. Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
3. Intuitive merging and joining data sets
4. Flexible reshaping and pivoting of data sets

Reference: http://pandas-docs.github.io/pandas-docs-travis/

Pandas - Continued

IO

1. Robust IO tools for loading data from
   - flat files (CSV and delimited),
   - Excel files,
   - databases,
   - and saving / loading data from the ultrafast HDF5 format

Reference: http://pandas-docs.github.io/pandas-docs-travis/

Pandas - Continued

Specialized Data Types: TimeSeries

1. Time series-specific functionality:
    - date range generation and frequency conversion,
    - moving window statistics,
    - moving window linear regressions,
    - date shifting and lagging, etc.

Reference: http://pandas-docs.github.io/pandas-docs-travis/

Working with Data in Python

1. intro-python-and-data-analysis.ipynb
2. pandas-explore-fred-data.ipynb

Other Useful Packages

There are a lot of additional packages that are useful for working with data in Python

1. NetworkX (https://networkx.github.io/)
2. odo (https://github.com/blaze/odo)
3. dask (http://dask.pydata.org/en/latest/)
4. statsmodels (http://statsmodels.sourceforge.net/)
5. Scikit-learn (http://scikit-learn.org/stable/)
6. Beautiful Soup, HTML5lib, ...
7. Matplotlib, Plotly, Bokeh ...

Additional References

The main reference is:

`https://lectures.quantecon.org/py/learning_python.html`

Additional References:

1. “Think Python”, Allen B. Downey, Oreilly Media
2. “Data Science from Scratch”, Joel Grus, Oreilly Media
3. “Python for Data Analysis”, Wes McKinney, Oreilly Media