Unit 1.2
Python programming

(Execute "Basic Python.ipynb" along)

IST 718 – Big Data Analytics

Daniel E. Acuna

http://acuna.io

Resources

- If you are coming from R: http://mathesaurus.sourceforge.net/matlab-python-xref.pdf (http://mathesaurus.sourceforge.net/matlab-python-xref.pdf)
- Python Essential Training in Lynda.com: https://www.lynda.com/Python-tutorials/Python-Essential-Training/614299-2.html (https://www.lynda.com/Python-tutorials/Python-Essential-Training/614299-2.html)
- Python for Data Science: https://www.lynda.com/Python-tutorials/Python-Data-Science-Essential-Training/520233-2.html (https://www.lynda.com/Python-tutorials/Python-Data-Science-Essential-Training/520233-2.html)
- Learn Python the Hard Way (book): https://learnpythonthehardway.org/python3/ (https://learnpythonthehardway.org/python3/)
- Learn Git and Github (today!): https://www.lynda.com/Git-tutorials/Up-Running-Git-GitHub/409275-2.html (https://www.lynda.com/Git-tutorials/Up-Running-Git-GitHub/409275-2.html)

Install

- If you haven’t yet, install Anaconda Python https://docs.continuum.io/anaconda/install/ (https://docs.continuum.io/anaconda/install/)
- Check that it is installed when you open your terminal by running:

```
>> python --version
Python 3.5.3 :: Anaconda 4.4.0 (x86_64)
```

- Create the following script:

```
print(“Hello, World”)
```

- Save as `hello_world.py`
- Run `python hello_world.py`

Different ways of interacting with Python: script

- One good editor for Python is Visual Studio Code (https://code.visualstudio.com/ (https://code.visualstudio.com/))
- Also PyCharm provides an academic license (need `.edu` email): https://www.jetbrains.com/pycharm/ (https://www.jetbrains.com/pycharm/)
- You can put your code in a file and then execute it with Python like before
- Or you can use `ipython` in the terminal
- Or you can use `jupyter notebook`

Jupyter Notebook Demo

Learn:

1. Jupyter notebook architecture
2. Cell
3. Cell type (e.g., markdown, code)
4. Executing code in a cell
5. Show line number
6. Command and edit mode
7. Delete cell
8. Insert cell
9. Block comment
10. Change notebook name
11. Save notebook
12. Validate tests (only visible tests)
13. Download notebook
14. Upload notebook

Python is Powerful and Easy to Understand

```
adjectives = ['awesome','good','excellent','awful','terrible']
count = 0
for word in adjectives :
    if word[0] == 'a':
        print(word)
        count += 1
print ("Found ", count ," adjectives that start with 'a'")
```

Can you figure out what does this program do? Take a minute.

Python Language Basics

Data Types

- Python has the expected data types like int, float, bool, str
- Python also contains types for collections of data:
    - tuple: immutable collection
    - list: all-purpose container for any data types
    - set: a container for unordered, unique values
    - dict: for storing maps of values (key/value pairs)
- Python is *loosely-typed*. Meaning you can change types later.

Demo: Data Types

Quick Check: What Type is it?

- 4.5
- "4.5"
- [4.5]
- { 'x' : 4.5 }
- (4.5)

- int
- str
- float
- bool
- tuple
- dict
- list

Lists

- A ***list*** is a mutable collection of any data type
- Lists are defined using square brackets
- Python list functions: [https://docs.python.org/3/tutorial/datastructures.html (https://docs.python.org/3/tutorial/datastructures.html)](https://docs.python.org/3/tutorial/datastructures.html)
- List items are indexed starting at 0
- Slice notation can be used to obtain a sub-list of values `x[2:4]` equal to a list of `X[2]` and `x[3]`
- `split()` makes a list from a string
- `range()` produces a sequential list of int

```
adjectives = ['awesome','good','excellent','awful','terrible']
temperatures = [87, 56, -4, 70, 92, 60, 38, 87, 64, 71]
coordinates = [(-34, 77), (45, -82)]
```

Demo: Python Lists

Quick Check: Lists

Code:

```
X = [1,1,2,3]
Y = X + [4]
Z = Y[1:3]
```

What is:

```
1. Y?
2. Z?
3. Y[:3]?
4. len(Y)?
5. X[2]?
```

Program Flow Control: IF

- The `if` statement is used to branch your code based on a Boolean (`True`/`False`) expression.

```
if boolean_expression:
  # statements when true
else:
  # statements when false
```

Program Flow Control: IF ladder

Use `elif` to make more than one decision in your `if` statement

```
if boolean_expression1:
    # statements when exp1 true
elif boolean_expression2:
    # statements when exp2 true
else:
    # statements when false
```

Program Flow Control: WHILE

The `while` statement is a loop. It is used to repeat statements as long as a Boolean expression is true.

```
while boolean_expression:
    # statements to
    # repeat while true
```

Program Flow Control: FOR

The `for` statement is a loop. It is used to iterate over a list of items.

```
for variable in list:
    # statements to
    # do for each item
```

Infamous Fizz Buzz interview question

"Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz"."

Python Strings

- Python has an extensive library of functions for manipulating strings: https://docs.python.org/3/library/stdtypes.html?highlight=upper#string-methods (https://docs.python.org/3/library/stdtypes.html?highlight=upper#string-methods)

```
today = "wednesDay"
print(today.capitalize(), today.upper(), today.index("ne"))
```

```
>> Wednesday WEDNESDAY 3
```

Functions

- Use the `def` keyword to define a custom function.
- Use `return` to send a value back to the function caller.

```
def function_name (arguments):
  # statements in function
  return expression_value
```

Demo: Functions

- Fibonacci seq

$$f(n) = f(n-2) + f(n-1)$$
$$f(1) = f(2) = 1$$

Dictionaries

- Dictionaries are Key-Value pairs
- Defined with curly brackets **{ }**
- Values are indexed by their key
- `KeyError` when key does not exist
- Circumvent with `.get()`

```
personal_info = {'Name':'John', 'Age':34, 'Height': 68.4}
print(personal_info['Age'])
```

```
>> 34
```

Demo: Dictionaries

Quick Check: Dictionaries

```
Code:
S = {'name' : 'bob', 'gpa'  : 3.4 }
S['major'] = 'IM&T'
```

What is:

```
1. S['gpa']?
2. S['Name']?
3. S[0]?
4. S.get(0,1)?
```

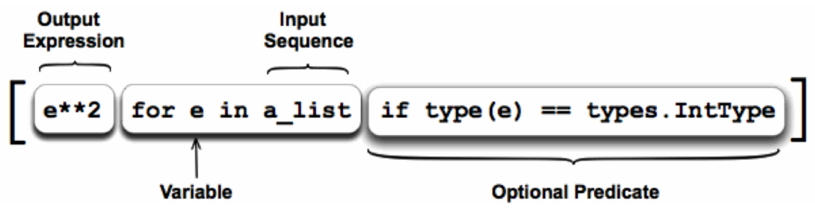
Help Me Code!

- Help me write a Python program to read in a string and output a dictionary of words and their frequencies.
- NOTE: Be sure to strip out the punctuation!

Comprehensions

- A list comprehension consists of the following parts:
  - An Input Sequence.
  - A Variable representing members of the input sequence.
  - An Optional Predicate expression.
  - An Output Expression producing elements of the output list from members of the Input Sequence that satisfy the predicate.

```
a_list = [1, '4', 9, 'a', 0, 4]
squared_ints = [e**2 for e in a_list if type(e) == types.IntType]
print(squared_ints) # [1, 81, 0, 16]
```



Other comprehensions

- Similarly we can create tuple, sets, and dictionary comprehensions

```
(a for a in range(10)) # tuple
{a for a in range(10)} # set
{k: v for k, v in [(1, 2), (3, 4)]} # dictionary
```

- Nested list comprehension

```
[[(i,j) for i in range(10)] for j in range(10)]
```

Demo: Lists of Dictionaries & List Comprehensions

For Representing Tables of Data

Activity

- Implement matrix multiplication:
    1. Create a function that takes two lists of rows representing a matrix each
    2. Return a list of rows that represents the multiplication of both
- Implement triplet matrix representation:

$$\mathbf{A} = \begin{bmatrix} 0 & 2 & 0 \\ 3 & 0 & 0 \end{bmatrix} \implies$$

```
A = [{'i': 1, 'j': 2, 'v': 2}, {'i': 2, 'j': 1, 'v': 3}]
```

- Implement matrix multiplication when representation is in triplets

Values and references

- Some types are held “by value” in a variable

```
x = 2
y = x
y = 2
x?
```

- Other types are held “by reference” in a variable

```
x = [1, 2]
y = x
y[0] = -1
x?
```

- All but the simplest types are held by reference

Python syntax is really expressive

Try:

```
a, b = (2, 3)
1 < 3 < 5
```

Advanced topics

Functions (1)

- It is easy to define functions:

```
def f():
  return 5
```

- Functions have their own type

```
type(f)?
```

- You can access hidden parameters of a function object: `f.__`
- You can define the help of a function (called a "doc") by putting a triple double quote box just below a function
- Try to run `?f` after the doc definition

Functions (2)

- You can have several types of arguments:

```
def f(a, b=0, c=3):
  return (a + b)*c
```

- These all work:

```
f(1), f(1, 2), f(1,c=5), f(1, 2, 3), f(1, c=3, b=2)
```

But this doesn't work

```
f(1, b=3, 3)
```

Functions (3)

- Be careful with mutable default arguments:

```
def f(x=[]):
    x.append(1)
    return x

f()
[1]
f()
??
```

- How to fix it?

Functions (4)

```
def f(x=None):
    if x is None:
        x = []
    x.append(1)
    return x

f()
[1]
f()
??
```

Functions (5)

- Pass arguments using a function

```
def f(a, b=2, c=3):
  return [a, b, c]

f(3, 2, 1)
??
f([3, 2, 1])
??
f(*[3, 2, 1])
??
f(*[3], **{'c': 1, 'b': 2})
??
```

Passing functions as arguments

```
def map(f, L):
    return [f(e) for e in L]
```

- Function generator:

```
def n_successor(n=1):
    def f(x):
        return x + n
    return f

successor = n_successor()
successor10 = n_successor(10)
```

Anonymous functions

-
  ```
  (lambda x: x + 1)(2)
  ```

-
  ```
  f  = lambda x: x + 1
  f(2)
  ```

-
  ```
  g = lambda a, b: a + b
  g(1, 2)
  ```

Docstrings

- Each object can have a “help” text attached

```
def f():
    """This function returns 0"""
    return 0

?f
?f.__doc__
```

Formatting strings and other tricks

- a, b = 1, 2

  ```
  print("The value of a is {a} and b is {b}".format(b=b, a=a))
  ```

- You can express conditions with more complicated format:

  ```
  True if a == 1 else False
  ```

- Create index for elements of a list

  ```
  [[index, value] for index, value in enumerate([1, 2, 3, 4])]
  ```

Exceptions

```
try:
    1/0
except ZeroDivisionError as e:
    print("Catched zero division", e)
except:
    print("Catch all other exceptions")
else:
    print("This is executed if everything goes perfectly")
finally:
    print("This is executed no matter what")
```

Generator

- One of the great features in Python

```python
def range_custom(n):
    i = 0
    while i < n:
        yield i
        i += 1

range_custom(3)
next(range_custom(3))
next(range_custom(3))
a = range_custom(3)
next(a)
next(a)
```

Classes and objects

- Much of Python functionality is provided by classes
- Classes define a "template" which will be followed by all objects created from that class
- For example

```python
import pandas as pd
df = pd.DataFrame()
```

- This will create an object of class `DataFrame`
- The class `DataFrame` "inherits" from class `NDFrame`. So it specializes what `NDFrame` does
- You can access all parent classes through `DataFrame.__bases__`

Let’s create a Class

```
class BankAccount():
    def __init__(self):
        self.balance = 0

    def deposit(self, amount):
        self.balance += amount
```

- Create an object from `BankAccount()`
- Objects are accessed “by reference”

Control flow

-
  ```
  if condition:
    pass
  elif another_condition:
    pass
  else:
    pass
  ```

-
  ```
  while condition:
    pass
  ```

Implement a withdraw method in the BankAccount method

```
class BankAccount():
    def __init__(self):
        self.balance = 0

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        # your code here
```

- Add some comments to your code

Implement a function that merges two accounts

```python
def merge_accounts(a1, a2):
    pass
```

- Add the `BankAccount` class definition and the merge definition into a file
- In IPython, make sure you are in the same directory as the file
- Import the module

```python
import module_name
```

- Now you can use the class and function inside!

```python
module_name.BankAccount()
```

Creating a package

1. Now create a folder with the name of package you want to create (e.g., banking)
2. Add `BankAccount` class definition to a file called `objects.py` inside that folder
3. Add `merge_account` function to a file called `operations.py`
4. In a file called `__init__.py` in that folder, import all the functionality of those two files

   ```python
   from .objects import *
   from .operations import *
   ```

5. Now you can import banking and start using the objects and functions with a much better organization of the content!

Create a string representation of an object

- Create a method called `__repr__` and return a string description of the object.
- For example:

```
def __repr__(self):
  return "Bank account with a balance of " \
      + str(self.balance)
```

Other resources

- Programming Foundations: Fundamentals https://www.lynda.com/Programming-Foundations-tutorials/Welcome/83603/90426-4.html (https://www.lynda.com/Programming-Foundations-tutorials/Welcome/83603/90426-4.html)
- Learning Python https://www.lynda.com/Python-tutorials/Up-Running-Python/122467-2.html (https://www.lynda.com/Python-tutorials/Up-Running-Python/122467-2.html)
- You should definitevely learn Github and Git: https://www.lynda.com/Git-tutorials/Up-Running-Git-GitHub/409275-2.html (https://www.lynda.com/Git-tutorials/Up-Running-Git-GitHub/409275-2.html)