A Little Python – Part 1

Introducing Programming with Python

Preface

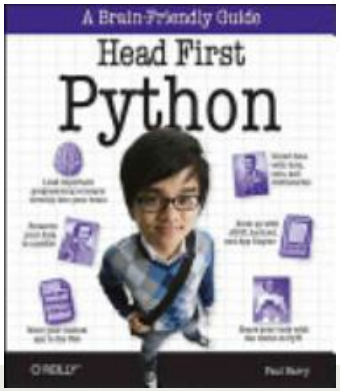
- Not a complete course in a programming language
  - Many details can't be covered
  - Need to learn as you go
- My programming 'style' is not considered 'pythonic'
  - I program like an overly careful C programmer
  - "Python" people generally hate my code
    - I'll do my best to give examples in good style here

Learning ANY Programming Language

- Syntax
  - What is a valid statement
  - What statements have meaning
- Variables, data types, data structure
- Control flow, branching, testing, loops, iteration
- Input/Output, I/O, read/write files
- Procedures, subroutines
- Objects, encapsulation of code + data

Additional Resources

- Online Courses/Tutorials
  - http://docs.python.org/tutorial/index.html
  - http://www.codecademy.com/tracks/python
    - Only 13 hours – do it this weekend!!!
- Learning Python book (5th edition) ~ $50 ($30)
  - Great if you know another language
- Head First Python ~ $35 ($25)
  - Good for real beginners
  - Although it's now "Python 3"



Why Python?

- Interpreted (vs compiled)
- Interactive

```
[Tiki:~] dwmc% python
Python 2.7.10 (default, Aug 22 2015, 20:33:39)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Just “try it” in the interpreter
  - If it works in the interactive interpreter – then it works

Try this …

- Launch python
  - Get a command line
  - Type "Python"
- Print output

```
>>> print "This is a string of text"
This is a string of text
```

- Assign values to variables

```
>>> five = 1234
>>> abc = "this is some text"
>>> alist = [1, 2, 'c', abc, None]
```

Try this …

- Output the values of variables

```
>>> print five
1234
>>> print abc
this is some text
>>> print a_list
[1, 2, 'c', 'this is some text', None]
>>>
```

Another Quick Example

- Python interpreter continuations
- To show how this works we define a simple function

```
>>> def counter(i):
...     k = 0
...     while( k<i ):
...             print k
...             k += 1
...
>>> counter(5)
```

Basic Data Types

- Empty value - `None`
- Boolean
- Strings
- Numbers
  - Integer values
  - Real numbers, Rational values, Floating point values, Decimal values

Basic Data Types - None

- The empty value
  - Special value, `None`
    - This is not the same as other empty values:
      - "" (empty string) or
      - [] (empty list) or
      - {} (empty dictionary)

Basic Data Types - Boolean

- Boolean - truth values (two of them)
  - `True`
  - `False`
- Boolean is the result of a comparison
  - `"abc" == 1` `(False)`
  - `"1" == 1` `(False)`
  - `5 == 5.0` `(True)`

Basic Data Types - Strings

- Strings are sequences of characters
- Assigning string values

```
>>> foo = "this is a string"
>>> bar = 'this is also a string'
>>> special = '''python allows
... multi line
... string contants'''
>>>
```

- Unicode strings
  - Important for web work
  - The ‘u’ designation makes a string unicode
  - More on this later

Basic Data Types - Integers

- Assigning integer values

```
>>> x = 123
>>> y = 111
>>> z = 1
>>> print x+z
124
>>> print x+y
234
>>> print y+z
112
>>>
```

- Integers support arbitrary, dynamic size

Basic Data Types - Float

- Assigning real/floating point values

```
>>> n = 123
>>> k = 1.11
>>> i = 1.3
>>> print n+k
124.11
>>> print n+i
124.3
>>> print i+k
2.41
>>>
```

- Float support arbitrary, dynamic size, automatic conversion

Operations

- Operations are the way you change variable values, compare, or manipulate values
- You've seen several "operators" already in prior examples – can you describe them or name them?

Operations

- You've seen …
- Assignment Operator – assign a value to a variable, copy the value of one variable to another variable
  - What character(s)?
- Addition Operator – add two values
  - What character(s)?
- Less Than Comparison Operator – test whether a value is less than another
  - What character(s)?
- Equality Comparison Operator – test whether two values are equal
  - What character(s)?

Operations

- Some operations

```
x = y
x or y
x and y
not x
x + y, x - y, -x
x * y, x / y
(), [], {}
x==y, x<y, x>y, x<=y, x>=y, x!=y
x in y, x not in y, x is y, x is not y
```

- Try a few of these

Example operations

```
>>> n = 124
>>> m = 2
>>> k = 10.5
>>> i = 1.3
>>> n / m
62
>>> n * m
248
>>> n / i
95.384615384615387
>>> n // i
95.0
>>> (i==k)
False
>>> (i<=k)
True
>>> (i!=k)
True
```

More example operations

```
>>> foo = "this is a string"
>>> bar = "this is a string"
>>> foo+bar
'this is a stringthis is a string'
>>> foo-bar
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>> foo*bar
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'
>>> foo in bar
True
>>> foo is bar
False
>>> bar in foo
True
```

Sidebar – the "import" statement

- "import" is a mechanism to extend what python can do
  - Adds features that are not "built in"
  - Adds things that don't get used all the time
  - Temporary - only added while the interpreter keeps running
- You will see this in many examples and it can be confusing at the start – watch for it

Sidebar – Example "import"

- Example features that are not built in
  - Random numbers
  - Operating system specific features (mac/linux/windows)
- Try some

```
>>> import math
>>> import random
>>> import aflac
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named aflac
>>> random.random()
0.14584529741440777
>>> random.randint(5,151)
30
```

Operators – Manipulating strings

- Strings have some special operators

```
s = "This sample has CAPS and hypen-ated words"
i = 5
j = 11
s[i]
s[i:j]
len(s)
s.find('is')
s.replace('is','as')
s.split('-')
s.isdigit()
s.lower()
```

- Try a few of these

Example – Manipulating strings

```
>>> s = "this is a string"
>>> s2 ="a-string-with-no-spaces"
>>> s3 = "30"
>>> s[5]
'i'
>>> s[5:6]
'i'
>>> s[5:7]
'is'
>>> s.find('is')
2
>>> s.replace('is','as')
'thas as a string'
>>> s2.split('-')
['a', 'string', 'with', 'no', 'spaces']
>>> s3.isdigit()
True
>>>
```

Generating Output

- Simple output
- print statement (is a print function in Python 3)

```
>>> s1 = "the value of k is"
>>> k = 1.45
>>> print s1,k
the value of k is 1.45
>>> print k,s1
1.45 the value of k is
>>> s2 = "bob's big boy"
>>> print s1,s2
the value of k is bob's big boy
>>> print s1+s2
the value of k isbob's big boy
>>>
```

Program Structure

- Function, procedure, method, subroutine (synonyms)

```
def counter(i):
    k = 0
    while( k<i ):
        print k
        k += 1
```

- Procedures and functions are how we keep some logical control over complex programs
- Few programs can be written as simply a very long list of simple statements
- This is important, we need to dissect this

Program Structure

```
def counter(i):
        k = 0
        while( k<i ):
                print k
                k += 1
```

- def - is a special word – think 'define'
- counter – this is what we are defining – it is a new procedure (or function), we pick the name
- (i) – this is a parenthesized list of parameters that the procedure 'counter' will take

Program Structure

```
def counter(i):
        k = 0
        while( k<i ):
                print k
                k += 1
```

- def is an example of a "block" structure
- In python a "block" is indented
  - Statements that end with a colon ":" indicate a block
  - Everything in the block is indented the same amount
  - Be careful, tab characters and space characters are not the same amount even if they "look" the same visually

Program Structure

```
def counter(i):
          k = 0
          while( k<i ):
                    print k
                    k += 1
```

- Parameters are the values that are given to a procedure (or function) when the procedure is called and executed
  - In this case when `counter(5)` is called the value 5 is assigned to the variable `i` while counter is running
  - If a variable `z=3`, and we called `counter(z)` then the value of `z` is assigned to the variable `'i'` while counter is running.

Program Structure

```
def counter(i):
        k = 0
        while( k<i ):
                print k
                k += 1
```

- This procedure has two statements in the "block"
  - A simple assignment statement
  - And a nested "block"
    - We'll get to "while" statements later

Program Structure

```
def counter(i):
          k = 0
          while( k<i ):
                    print k
                    k += 1
```

- This procedure has two statements in the "block"
  - A simple assignment statement
  - And a nested "block"
    - We'll get to "while" statements later …
    - Our nested block has two statements

Define Simple Procedures

```
def bob():
        print "Bob is great!"

def notBob():
        print "Bob is a fink!"

def liveBob():
        print "Long live Bob!"

def bobParam(superlative):
        print "Bob is a",superlative
```

Procedures can return a value

```
def bob():
        print "Bob is great!"

def notBob():
        print "Bob is a fink!"

def liveBob():
        print "Long live Bob!"

def bobParam(superlative):
        print "Bob is a",superlative

def bobConcat(superlative):
        print "Bob is a",superlative
        return superlative+" "+"dude"
```

Bob Procedure Output

```
>>> bob()
Bob is great!
>>> notBob()
Bob is a fink!
>>> liveBob()
Long live Bob!
>>> varBob("chocolate coated candy!")
Bob is a chocolate coated candy!
>>> BOB()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'BOB' is not defined
>>>
```

Python Programming Environments

- IDLE (sucks)
- Ipython
  - Iron python - very popular, nice "workbook" feature
- Python Additions to Eclipse
  - PyDev
- Komodo Edit (nice)
  - Komodo IDE (expensive)
- Plain text editor
  - Old school

Python Plain old Text Editors

- Mac OS
  - Sublime Text
  - Text Wrangler
  - SubEthaEdit
  - Smultron
  - Komodo Edit

Assignment 1

- Write 5 short programs (some are a single line)

1. Make the python interpreter calculate and print 13! (13 factorial)
2. Make the python interpreter output "Happy New Year!" using 3 different string variables.
3. Define three procedures that each returns one string of "Happy", "New", and "Year!". In the python interpreter execute the three procedures and show what they output.

Assignment 1 - continued

- Write 5 short programs

4. Write a new procedure using the ones you created in the prior problem. Make your new procedure print "Happy New Year!"

5. Write a procedure that takes two parameters and adds them together. The procedure should write output that looks like an addition statement. For example, if the procedure was given the values 3 and 4 the output should be something like: "3 + 4 = 7"