



Ansh Bhawnani
Python Beginner's Course
Bitten Tech



Python

- ▶ Simple
 - ▶ Python is a simple and minimalistic language in nature
 - ▶ Reading a **good** python program should be like reading English
 - ▶ Its Pseudo-code nature allows one to concentrate on the problem rather than the language
- ▶ Easy to Learn
- ▶ Free & Open source
 - ▶ Freely distributed and Open source
 - ▶ Maintained by the Python community
- ▶ High Level Language - memory management
- ▶ Portable - *runs on anything c code will



Python

- ▶ Interpreted
 - ▶ You run the program straight from the source code.
 - ▶ Python program → Bytecode → a platform's native language
 - ▶ You can just copy over your code to another system and it will auto-magically work! *with python platform
- ▶ Object-Oriented
 - ▶ Simple and additionally supports procedural programming
- ▶ Extensible - easily import other code
- ▶ Embeddable - easily place your code in non-python programs
- ▶ Extensive libraries
 - ▶ (i.e. reg. expressions, doc generation, CGI, ftp, web browsers, ZIP, WAV, cryptography, etc...) (wxPython, Twisted, Python Imaging library)



Timeline

- ▶ Python was conceived in the late 1980s.
 - ▶ Guido van Rossum, Benevolent Dictator For Life
 - ▶ Rossum is Dutch, born in Netherlands, Christmas breather, big fan of Monty Python's Flying Circus
 - ▶ Descendant of ABC, he wrote glob() func in UNIX
 - ▶ M.D. @ U of Amsterdam, worked for CWI, NIST, CNRI, Google
 - ▶ Also, helped develop the ABC programming language
- ▶ In 1991 python 0.9.0 was published and reached the masses through alt.sources
- ▶ In January of 1994 python 1.0 was released
 - ▶ Functional programming tools like lambda, map, filter, and reduce
 - ▶ comp.lang.python formed, greatly increasing python's userbase





Timeline

- ▶ In 2000, Python 2.0 was released.
 - ▶ Introduced list comprehensions similar to Haskell's
 - ▶ Introduced garbage collection
- ▶ In 2001, Python 2.2 was released.
 - ▶ Included unification of types and classes into one hierarchy, making Python's object model purely Object-oriented
 - ▶ Generators were added (function-like iterator behavior)
- ▶ In 2008, Python 3.0 was released.
 - ▶ Broke Backward compatibility
 - ▶ Latest version is 3.7
 - ▶ Course will be based on Python3



Python types

- ▶ Str, unicode - 'MyString', u'MyString'
- ▶ List - [69, 6.9, 'mystring', True]
- ▶ Tuple - (69, 6.9, 'mystring', True) immutable
- ▶ Set/frozenset - set([69, 6.9, 'str', True])
frozenset([69, 6.9, 'str', True]) -no duplicates & unordered
- ▶ Dictionary or hash - {'key 1': 6.9, 'key2': False} -
group of key and value pairs



Python types

- ▶ Int - 42- may be transparently expanded to long through 438324932L
- ▶ Float - 2.171892
- ▶ Complex - $4 + 3j$
- ▶ Bool - True or False



Python semantics

- ▶ Each statement has its own semantics, the `def` statement doesn't get executed immediately like other statements
- ▶ Python uses duck typing, or latent typing
 - ▶ Allows for polymorphism without inheritance
 - ▶ This means you can just declare `"somevariable = 69"` don't actually have to declare a type
 - ▶ `print "somevariable = " + toString(somevariable)` strong typing , can't do operations on objects not defined without explicitly asking the operation to be done



Python Syntax

- Python uses indentation and/or whitespace to delimit statement blocks rather than keywords or braces
- ```
if __name__ == "__main__":
 print "Salve Mundo"
```

*# if no comma (,) at end '\n' is auto-included*

## • **CONDITIONALS**

- ```
if (i == 1): do_something1()  
elif (i == 2): do_something2()  
elif (i == 3): do_something3()  
else: do_something4()
```



Conditionals Cont.

- `if (value is not None) and (value == 1):`
 `print "value equals 1",`
 `print " more can come in this block"`
- `if (list1 <= list2) and (not age < 80):`
 `print "1 = 1, 2 = 2, but 3 <= 7 so its True"`
- `if (job == "millionaire") or (state != "dead"):`
 `print "a suitable husband found"`
 `else:`
 `print "not suitable"`
- `if ok: print "ok"`



Loops/Iterations

- `sentence = ['Marry','had','a','little','lamb']`
`for word in sentence:`
 `print word, len(word)`
- `for i in range(10):`
 `print i`
`for i in xrange(1000):` # does not allocate all initially
 `print i`
- `while True:`
 `pass`
- `for i in xrange(10):`
 `if i == 3: continue`
 `if i == 5: break`
 `print i,`



Functions

- ▶ `def print_hello():` # returns nothing
 `print "hello"`
- ▶ `def has_args(arg1, arg2=['e', 0]):`
 `num = arg1 + 4`
 `mylist = arg2 + ['a', 7]`
 `return [num, mylist]`
 `has_args(5.16, [1, 'b'])` # returns [9.16, [[1, 'b'], ['a', 7]]]
- ▶ `def duplicate_n_maker(n):` #lambda on the fly func.
 `return lambda arg1: arg1*n`
 `dup3 = duplicate_n_maker(3)`
 `dup_str = dup3('go')` # `dup_str == 'gogogo'`



Exception handling

```
▶ try:
    f = open("file.txt")
except IOError:
    print "Could not open"
else:
    f.close()

▶ a = [1,2,3]
try:
    a[7] = 0
except (IndexError, TypeError):
    print "IndexError caught"
except Exception, e:
    print "Exception: ", e
except:      # catch everything
```

```
print "Unexpected:"
print sys.exc_info()[0]
raise # re-throw caught
exception
```

```
try:
    a[7] = 0
finally:
    print "Will run regardless"
```

- Easily make your own exceptions:



Classes

```
class MyVector: """A simple vector
class."""
    num_created = 0
    def __init__(self,x=0,y=0):
        self.__x = x
        self.__y = y
        MyVector.num_created += 1
    def get_size(self):
        return self.__x+self.__y
    @staticmethod
    def get_num_created
        return MyVector.num_created
```

#USAGE OF CLASS MyVector

```
print MyVector.num_created
v = MyVector()
w = MyVector(0.23,0.98)
print w.get_size()
bool = isinstance(v,
    MyVector)
```

Output:

0

1.21



I/O

read binary records from a file

from struct **import** *

fin = **None**

try:

fin = **open**("input.bin", "rb")

s = f.read(8) *#easy to read in*

while (len(s) == 8):

x,y,z = unpack(">HH<L", s)

print "Read record: " \

"%04x %04x %08x"%(x,y,z)

s = f.read(8)

except IOError:

pass

if fin: fin.close()

import os

print os.getcwd() *#get "."*

os.chdir('..')

import glob *# file globbing*

lst = glob.glob('*.txt') *# get list of files*

import shutil *# mngmt tasks*

shutil.copyfile('a.py', 'a.bak')



Threading in Python

```
import threading
theVar = 1
class MyThread ( threading.Thread ):
    def run ( self ):
        global theVar
        print 'This is thread ' + \
            str ( theVar ) + ' speaking.'
        print 'Hello and good bye.'
        theVar = theVar + 1
for x in xrange ( 10 ):
    MyThread().start()
```



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\far...
This is thread 1 speaking.
Hello and good bye.
This is thread 2 speaking.
Hello and good bye.
This is thread 3 speaking.
Hello and good bye.
This is thread 4 speaking.
Hello and good bye.
This is thread 5 speaking.
Hello and good bye.
This is thread 6 speaking.
Hello and good bye.
This is thread 7 speaking.
Hello and good bye.
This is thread 8 speaking.
Hello and good bye.
This is thread 9 speaking.
Hello and good bye.
This is thread 10 speaking.
Hello and good bye.
C:\Documents and Settings\far...
```




So what does Python have to do with Internet and web programming?

- ▶ Jython & IronPython(.NET ,written in C#)
- ▶ Libraries - ftplib, snmplib, uuidlib, smtpd, urlparse, SimpleHTTPServer, cgi, telnetlib, cookielib, xmlrpclib, SimpleXMLRPCServer, DocXMLRPCServer
- ▶ Zope(application server), PyBloxsom(blogger), MoinMoin(wiki), Trac(enhanced wiki and tracking system), and Bittorrent (6 no, but prior versions yes)



Applications of Python

- ▶ Web Development: Django, Flask, Bottle, Pyramid
- ▶ Scientific and Numeric: SciPy, Pandas, numpy
- ▶ Desktop applications: PyQt, Kivy, PyGUI, PyGTK, WxPython
- ▶ Databases: ODBC, MSSQL, PostgreSQL, Oracle
- ▶ Business: Odoo, Tryton
- ▶ Machine Learning and AI: TensorFlow, scikit-learn, Theano, Caffe, Keras
- ▶ Data Science: Matplotlib, Bokeh, Seaborn, Scrapy
- ▶ PenTesting: Socket, Scapy, Libnmap, Requests, MITMProxy, Spidey, urllib2



Python Interpreters

- ▶ <http://www.python.org/download/>
- ▶ <http://pyaiml.sourceforge.net/>
- ▶ <http://www.py2exe.org/>
- ▶ <http://www.activestate.com/Products/activepython/>
- ▶ <http://www.wingware.com/>
- ▶ <http://pythonide.blogspot.com/>
- ▶ Many more...



Python on your systems

- ▶ Its easy! Go to <http://www.python.org/download/>
- ▶ Download your architecture binary, or source
- ▶ Install, make, build whatever you need to do... plenty of info on installation in readmes
- ▶ Make your first program! (a simple one like the hello world one will do just fine)
- ▶ Two ways of running python code. Either in an interpreter or in a file ran as an executable



Running Python



- Windows XP - double click the icon or call it the command line as such:

from

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\farrin>cd Desktop
C:\Documents and Settings\farrin\Desktop>test.py
Hello World!
C:\Documents and Settings\farrin\Desktop>
```

Python Interpreter

```
Python (command line)
Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hello world'
hello world
>>> x = 'roses'
>>> y = 12
>>> Roy_G_Biv = [('red', 'orange', 'yellow'), 'green', ('blue', ['indigo', 'violet'])
]
>>> MyAwesomeVar = [x, y, Roy_G_Biv]
>>> print MyAwesomeVar
['roses', 12, [('red', 'orange', 'yellow'), 'green', ('blue', ['indigo', 'violet
'])]]
>>> print MyAwesomeVar[0]+' are '+MyAwesomeVar[2:3][0][0][0]
roses are red
>>> print MyAwesomeVar[2:3][0][2][1][1]+'s are '+MyAwesomeVar[2:3][0][2][0]
violets are blue
>>> print str(MyAwesomeVar[1])+' '+MyAwesomeVar[0]+' for my love.'
12 roses for my love.
>>> dict = {'place': 'mantle', 'where': 'above', 'myLove': True}
>>> if(dict["myLove"]): print 'the ' +dict["place"]+' I put them '+dict["where"]

...
the mantle I put them above
>>>
```



Python for the future

▶ Python 3

- ▶ Will not be Backwards compatible, they are attempting to fix “perceived” security flaws.
- ▶ Print statement will become a print function.
- ▶ All text strings will be unicode.
- ▶ Support of optional function annotation, that can be used for informal type declarations and other purposes.



Python: Data Types

Name	Type	Description
Integers	int	Whole numbers, such as: 3 300 200
Floating point	float	Numbers with a decimal point: 2.3 4.6 100.0
Strings	str	Ordered sequence of characters: "hello" 'Sammy' "2000" "楽しい"
Lists	list	Ordered sequence of objects: [10,"hello",200.3]
Dictionaries	dict	Unordered Key:Value pairs: {"mykey": "value" , "name": "Frankie"}
Tuples	tup	Ordered immutable sequence of objects: (10,"hello",200.3)
Sets	set	Unordered collection of unique objects: {"a","b"}
Booleans	bool	Logical value indicating True or False



Python: String Indexing

Character : **h** **e** **l** **l** **o**

Index : **0** **1** **2** **3** **4**

Reverse Index: **0** **-4** **-3** **-2** **-1**



Python: String Slicing

- ▶ Sometimes you may find it necessary to extract a portion of a string from another string.
- ▶ You can use “slicing” notation in Python to extract a span of characters from a string into a new string. We call this new String a "substring"

[start:stop:step]