Python Scripting for System Administration

Rebeka Mukherjee
Department of Computer Science and Engineering
Netaji Subhash Engineering College, Kolkata

E-mail: *rebekamukherjee@gmail.com*

Workshop On Women In Free Software and Fedora Women's Day
(15 July 2016)

Quick Facts

- Python is a widely used high-level, general purpose, interpreted, dynamic programming language
- Its Implementation was started in December 1989 in CWI (*Centrum Wiskunde& Informatica)* in Netherlands
- Invented by **Guido van Rossum**, early 90s (Feb‟91)
- It is successor of ABC Programming Language
- Python 2.0 was released in October 2000, with many new features including a full garbage collector and Unicode. Python 3.0 was released in December 2008.
- Open sourced from the beginning

Installing & Running Python

- Python is pre-installed on most Unix systems
- The pre-installed version may not be the most recent one (2.7.9 and 3.4.3 as of July '14)
- Download from **http://python.org/download/**
- Python comes with a large library of standard modules
- There are several options for an IDE: IDLE, Vim, Emacs, or your favorite text editor

Python Basics

Hello World

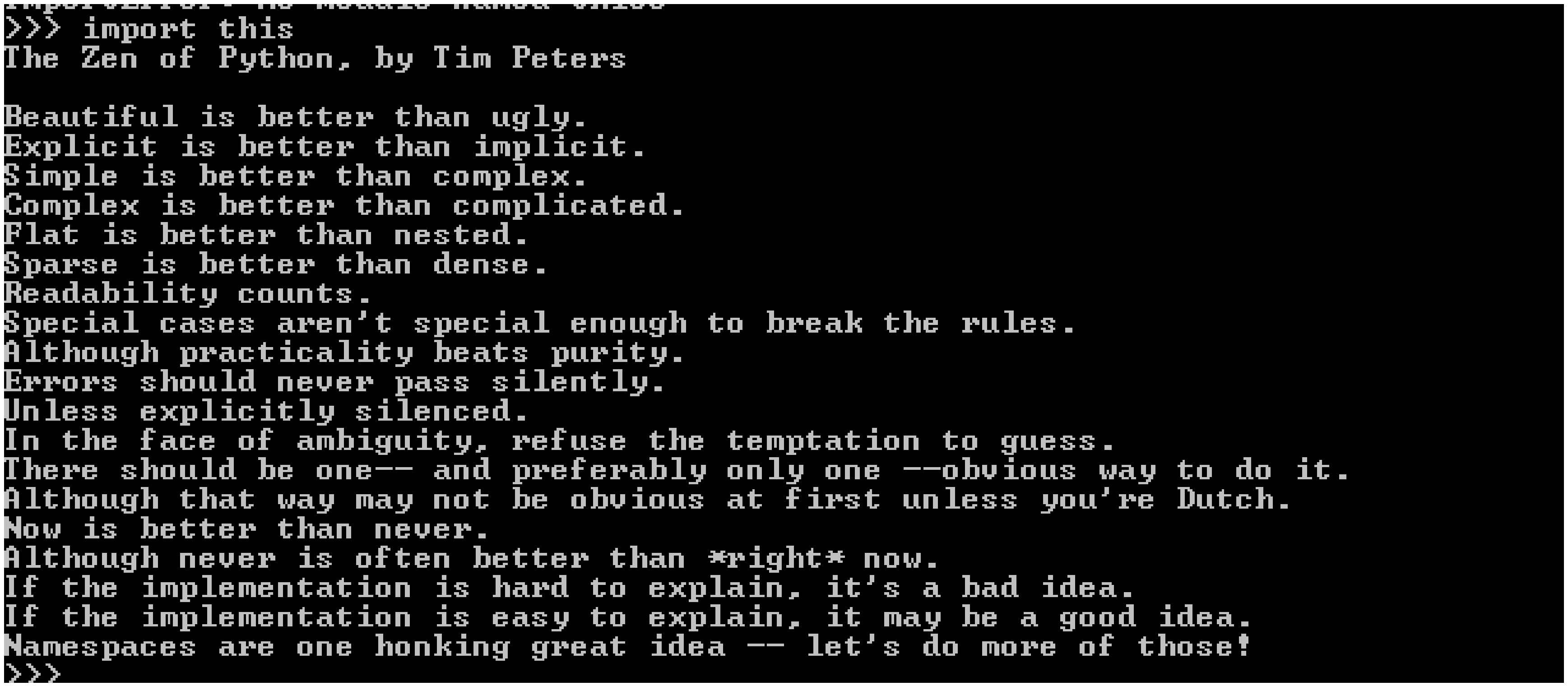
- Open a terminal window and type “python”
- Python prompts with >>>
- At the prompt type ‘hello world!’

```
>>> print ‘hello world!’
hello world!
```

- To exit Python: CTRL-D

Python Philosophy

>>> import this



```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Python Interactive Shell

```
>>> a = 5
>>> b = 6
>>> a + b
11

>>> 2+3*5
17

>>> 2**8
256

>>> help (len)
Return the number of  items of a sequence or collection.
```

Printing and Documentation

```
>>> print ‘this will print’
this will print

>>> #this will not print
>>>

>>> name = ‘Rebeka’
>>> print ‘Hello ’ + name + ‘!’
Hello Rebeka!

>>> a = ‘I can program in Python’
>>> len (a)
23
```

Variables

- Variables are not declared, just assigned
- The variable is created the first time you assign it a value
- Variables are references to objects. The type information is with the object, not the reference
- Everything in Python is an object
- A reference is deleted via garbage collection after any names bound to it have passed out of scope

Variables

```
>>> y
Traceback(most recent call last):
    File "<stdin>", line 1, in <module>
NameError: name ‘y' is not defined
>>> y = 3
>>> y
3

>>> x, y = 2, ‘hello’
>>> x
2
>>> y
‘hello’
```

Data Types

Object Type	Examples literals/ creation
Numbers	1234, 3.141, 2+3j, 001101, Decimal(), Fraction()
Strings	‘Spam’, “Bob’s”, b'a\x01c‘, u'sp\xc4m‘(unicode)
Lists	[1,[2,three],4.5], list(range(10))
Tuples	(1, ‘spam’, 4, ‘U’), tuple(‘spam’)
Dictionaries	{‘user’: ‘scott’, ‘pswd’:12345}, dict=(hours=10)
Files	Open(‘wave.txt’), open(r 'C:\bin', 'wb')
Sets	set('abc'), {'a', 'b', 'c'}
Other Data type	Boolean

Basic Operators

- Arithmetic operators: +, -, *, /, //, %
    - **+** is also used for concatenation
    - **-** is also used for set difference
    - ***** is also used for string repetition
    - **//** is also used for floor division
    - **%** is also used for string formatting

- Assignment operator: =

- Comparison operator: ==

- Logical operators: and, or, not

Numbers

```
>>> 3+1, 3-1          # Addition, Subtraction
(4, 2)

>>> 4*3, 4/2          # Multiplication, Division
(12, 2)

>>> 5%2, 4**2         # Modulus (Remainder), Power
(1, 16)

>>> 2+4.0, 2.0**4     #Mixed type conversions
(6.0, 16.0)
```

Numbers

```
>>> (2+3j)*(4+5j)        # Complex Numbers
(-7+22j)

>>> 0b0010               # Binary
2

>>> 0xff                 # Hexadecimal
256

>>> bin (64), oct (64), hex (64)
(‘0b1000000’, ‘0100’, ‘0x40’)

>>> 1/2
0.5
```

Numbers

```
>>> from fractions import Fraction

>>> x = Fraction (2,3)
>>> print x
2/3

>>> a = Fraction (1,3)
>>> b = Fraction (2,3)
>>> c = a-b
>>> print c
(-1/3)

>>> print Fraction ('.25')
1/4
```

String

- Immutable
- Strings are defined using quotes (‘, “ or “““)

```
>>> st= “Hello World”
>>> st= ‘Hello World’
>>> st= “““This is a multi-line
string that uses triple quotes.”””
```

Set

```
>>> x=set('abcde')
>>> y=set('bdxyz')
>>> x
{'a', 'd', 'b', 'c', 'e'}
>>> y
{'x', 'd', 'b', 'z', 'y‘}
>>> x|y                                  #Union
{'d', 'e', 'y', 'c', 'x', 'a', 'b', 'z'}
>>> x&y                                  # Intersection
{'d', 'b'}
>>> x-y                                  # Set Difference
{'a', 'e', 'c'}
>>> x>y, x<y                             #Superset, Subset
(False, False)
>>> a.add(‘hello’)
>>> x                                    # Add member in set
{'a', 'd', 'b', 'c', 'e‘,’hello’}
```

List

- Mutable ordered sequence of items of mixed types
- Lists are defined using square brackets (and commas)

```
>>> li = [“abc”, 34, 4.34, 23]
>>> li[1]
34

>>> list2 = list1              # 2 names refer to 1 ref
                               # Changing one affects both

>>> list2 = list1[:]           # 2 independent copies, 2 refs
```

List

```
>>> li = [1, 11, 3, 4, 5]

>>> li.insert(2, ‘i’)
>>>li
[1, 11, ‘i’, 3, 4, 5]

>>> li.append(‘a’)                 # append takes a singleton as arg
>>> li
[1, 11, ‘i’, 3, 4, 5, ‘a’]

>>> li.extend([9, 8, 7])           # extend takes a list as arg
>>>li
[1, 2, ‘i’, 3, 4, 5, ‘a’, 9, 8, 7]
```

List

```
>>> li = [‘a’, ‘b’, ‘c’, ‘b’]

>>> li.index(‘a’)            # index of first occurrence
1
>>> li.count(‘b’)            # number of occurrences
2
>>> li.remove(‘b’)           # remove first occurrence
>>> li
[‘a’, ’c’, ‘b’]
>>> li.reverse()             # reverse the list
>>> li
[‘b’, ‘c’, ‘a’]
>>> li.sort()                # sort the list
>>> li
[‘a’, ‘b’, ‘c’]
```

Tuple

- A simple immutable ordered sequence of items
- Items can be of mixed types, including collection types
- Tuples are defined using parentheses (and commas)

```
>>> tu = (23, “abc”, 4.56, (2,3), “def”)
>>> tu[1]                # positive index (second item)
‘abc’
>>> t[-3]                # negative index (third last item)
4.56
>>> t[1:4]               # slicing (return copy of subset)
(‘abc’, 4.56, (2,3))
```

Dictionary

- Dictionaries store a mapping between a set of keys and a set of values
  - Keys can be any immutable type
  - Accessed by key, not offset position
  - Values can be any type
  - A single dictionary can store values of different types

- You can define, modify, view, lookup, and delete the key-value pairs in the dictionary.

```
>>> d={'user':'scott','pswd':12345}
>>> d['user']
'scott'
>>> d['pswd']
12345
```

Dictionary

```
>>> d['id']=45                          # insert another key with value
>>> d
{'user': 'tiger', 'pswd': 12345, 'id': 45}

>>> d.keys()                            # keys() method
dict_keys(['user', 'pswd', 'id'])

>>> d.items()                           # items() method
dict_items([('user', 'tiger'), ('pswd', 12345), ('id', 45)])

>>> del(d['user'])                      # del() method
>>> d
{'pswd': 12345, 'id': 45}
```

Input

- The raw_input(string) method returns a line of user input as a string
- The parameter is used as a prompt
- The string can be converted by using the conversion methods int(string), float(string), etc

```
>>> input = raw_input(“Enter your age: ")
>>> age = int(input)
```

Conditional Statements

- If-elif-else:

```
>>> if age <= 3:
            print 'Toddler'
      elif age <= 12:
            print 'Kid'
      elif age <= 19:
            print 'Teenager'
      else:
            print 'Adult'
```

Conditional Statements

- switch-case:

```
>>> choice = ‘three’
>>> print ({‘one’: 1,
       ‘two’: 2,
       ‘three’: 3,
       ‘four’: 4}[choice])
3
```

Iterative Statements

- while-loops:

```
>>> x=1
>>> while(x<10):
        print(x)
        x=x+1
```

Iterative Statements

- For-loops:

```
>>> for i in range(20):
        if (i%3 == 0):
                print (i)
        if (i%5 == 0):
                print( "Bingo!“)
        print ("---“)
```

- range creates a list of numbers in a specified range
- range ([start,] stop [,step])

Loop Control Statements

- break - Jumps out of the closest enclosing loop
- continue - Jumps to the top of the closest enclosing loop
- pass - Does nothing, empty statement placeholder

Functions

- Can be assigned to a variable
- Can be passed as a parameter
- Can be returned from a function (return sends a result back to the caller)
- Functions are treated like any other variable in Python
- The def statement simply assigns a function to a variable
- There is no concept of function overloading in Python (two functions cannot have the same name, even if they have different arguments)

Functions

```
>>> def max (a, b):
        if a > b:
                print a, ‘ is greater’
        elif b > a:
                print b, ‘ is greater’
        else:
                print ‘both are equal’

>>> x = 5
>>> y = 9
>>> max (x, y)
9 is greater
```

Functions

```
>>> def fact(n):
        if n < 1:
            return 1
        else:
            return n * fact(n - 1)

>>> x = 5
>>> y = fact (x)
>>> print y
120
```

Modules

- Python’s extensive library contains built-in modules that may be used to simplify and reduce development time.

- Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a **module**.

- Definitions from a module can be **imported** into other modules.

- A module can contain executable statements as well as function definitions. These statements are intended to initialize the module. They are executed only the first time the module name is encountered in an import statement.

- **Eg**. import pwd, sys, os, csv, subprocess

Packages

- Collection of modules in directory
- Must have __init__.py file
- May contain subpackages

```
>>> import dir1.dir2.mod
>>> from dir1.dir2.mod import x
```

Class

```
>>> class Stack:
        def __init__(self):                         #constructor
                self.items = []

        def push(self, x):
                self.items.append(x)

        def pop(self):
                x = self.items[-1]
                del self.items[-1]
                return x

        def empty(self):
                return len(self.items) == 0
```

Exception

- An event occurs during the execution of program
- Disrupts normal flow of program’s instruction
- It must be handled

```
>>> def tem_convert(var):
        try:
                x=1/var
                return(x)
        except ZeroDivisionError:
                print("argument dont match")
        return
```

System Administration

Why Python?

- Python is easy to learn
- Allows you to do fairly complex tasks
- Express complex ideas with simple language constructs
- Readability
- Simple support for Object Oriented Programming (OOP)
- Python Standard Library
- Easy access to numerous third-party packages

Example 1

Search for files and show permissions

Steps:

1. Get the search pattern from the user.
2. Perform the search.
3. Print a listing of files found.
4. Using the stat module, get permissions for each file found.
5. Present the results to the user.

Python Code

```
import stat, sys, os, string, commands

#Getting search pattern from user and assigning it to a list
try:
    #run a 'find' command and assign results to a variable
    pattern = raw_input("Enter the file pattern to search for:\n")
    commandString = "find " + pattern
    commandOutput = commands.getoutput(commandString)
    findResults = string.split(commandOutput, "\n")

 #output find results, along with permissions
    print "Files:"
    print commandOutput
    print "================================"
```

Python Code (contd.)

```
for file in findResults:
        mode=stat.S_IMODE(os.lstat(file)[stat.ST_MODE])
        print "\nPermissions for file ", file, ":“
 for level in "USR", "GRP", "OTH":
            for perm in "R", "W", "X":
                if mode & getattr(stat,"S_I"+perm+level):
                    print level, " has ", perm, " permission"
                else:
                    print level, " does NOT have ", perm, "
    permission"
except:
    print "There was a problem - check the message
    above"
```

Example 2

Menu driven operations on a tar archive

Steps:

1. Open the tar file.
2. Present the menu and get the user selection.
3. If you press 1, the program prompts you for the file name in the archive to extract the current directory to and then extracts the file.
4. If you press 2, the program prompts you for the file name and then displays the file information.
5. If you press 3, the program lists all the files in the archive.

Python Code

```
import tarfile, sys

try:
    #open tarfile
    tar = tarfile.open(sys.argv[1], "r:tar")

    #present menu and get selection
    selection = raw_input("Enter\n\
    1 to extract a file\n\
    2 to display information on a file in the archive\n\
    3 to list all the files in the archive\n\n")
```

Python Code (contd.)

```
#perform actions based on selection above
  if selection == "1":
    filename = raw_input("enter the
  filename to extract:  ")
    tar.extract(filename)
  elif selection == "2":
    filename = raw_input("enter the
  filename to inspect:  ")
```

Python Code (contd.)

```
for tarinfo in tar:
        if tarinfo.name == filename:
           print "\n\
           Filename:\t\t", tarinfo.name, "\n\
           Size:\t\t", tarinfo.size, "bytes\n\
  elif selection == "3":
     print tar.list(verbose=True)
except:
  print "There was a problem running the
  program"
```

Example 3

Check for a running process and show information

Steps:

1. Get the name of a process to check and assign it to a variable.
2. Run the ps command and assign the results to a list.
3. Display detailed information about the process with English terms.

Python Code

```
import commands, os, string

program = raw_input("Enter the name of the
    program to check: ")

try:
    #perform a ps command and assign results to a
    list
    output = commands.getoutput("ps -f|grep " +
    program)
    proginfo = string.split(output)
```

Python Code (contd.)

```
 #display results
    print "\n\
    Full path:\t\t", proginfo[5], "\n\
    Owner:\t\t\t", proginfo[0], "\n\
    Process ID:\t\t", proginfo[1], "\n\
    Parent process ID:\t", proginfo[2], "\n\
    Time started:\t\t", proginfo[4]
except:
    print "There was a problem with the
    program."
```

Other Uses of Scripts

- **Managing servers:** Checks patch levels for a particular application across a set of servers and updates them automatically.
- **Logging:** Sends an e-mail automatically if a particular type of error shows up in the syslog.
- **Networking:** Makes a Telnet connection to a server and monitors the status of the connection.
- **Testing Web applications:** Uses freely available tools to emulate a Web browser and verifies Web application functionality and performance.

Thank You