381
Introduction to Python

Sushil J. Louis

Python - from Monty Python

- Guido Van Rossum started implementing in 1989. BDFL.
- We will be using Python 2.6.4
- Easy to learn, powerful, interpreted language
- Dynamic typing, classes and object orientation support, built in data types, garbage collection, and many modules.
- Very good as a fast prototyping language
- Extensible language - write C code call from python
- Support for modules
- Exceptions
- Everything is easy
- Runs on everything

Python data types

- Numbers
  - 1, 2, 7.8, complex,
- Strings
  - "hello python", 'hello python', '"hello python"'
- Lists
  - [], [1, 2, 3], ['a', 'hello', 342, [9, 7.8, 'by']]
- Tuples
  - (), (5, 6, 7)
  - return (x, x+y, (x+2y))
  - (1, 3, 'the third')
- Dictionaries
  - {}, d = {3:"Sushil", 2: "Chris", 5:"Jane", 'Jett': "Joan"}
  - d[3], d['Jett']

Python programming language constructs

- for i in range(0, 10):
  - print i
- for item in [1, 2, 5, 8, 9, 'jon']:
  - print item
- while b < 1000:
  - b = b + 1
  - print b
- if b < 1000:
  - print 'b is less than 1000'
- elif b == 1000:
  - print 'b is a 1000'
- else:
  - print 'b is greater than 1000'
-

Python functions

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

def doNothing():
    pass

def factorial(n = 10):
    f = 1
    while n > 1:
        f = f * n
    return f
```

Default arguments

```
i = 5
def f(arg=i):
  print arg
i = 6
f()
```

will print?

5

Default Arguments

```
def f(a, L=[]):
  L.append(a)
  return L

print f(1)
print f(2)
print f(3)
```

Will print

```
[1]
[1, 2]
[1, 2, 3]
```

Keyword Arguments

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian
Blue'):
 print "-- This parrot wouldn't", action,
 print "if you put", voltage, "volts through it."
 print "-- Lovely plumage, the", type
 print "-- It's", state, "!"
```

Valid:

```
parrot(1000)
parrot(action="VOOOOM", voltage=100000)
parrot('a thousand', state = 'pushing up the daisies')
parrot('a million', 'bereft of life', 'jump')
```

Keyword Arguments

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
 print "-- This parrot wouldn't", action,
 print "if you put", voltage, "volts through it."
 print "-- Lovely plumage, the", type
 print "-- It's", state, "!"
```

Invalid:

```
parrot() # required argument missing
parrot(voltage=5.0, 'dead') # non-keyword argument following keyword
parrot(110, voltage=220) # duplicate value for argument
parrot(actor='John Cleese') # unknown keyword
```

Coding style

- Use 4-space indentation, and no tabs.
- Use blank lines to separate functions and classes, and larger blocks of code inside functions.
- Use spaces around operators and after commas, but not directly inside bracketing constructs: a = f(1, 2) + g(3, 4).
- Name your classes and functions consistently; the convention is to use CamelCase for classes and startWithLowerCamelCase for functions and methods. Always use self as the name for the first method argument (see *A First Look at Classes* for more on classes and methods).
- Plain ASCII works best in any case

Reading and Writing

```
open(filename, mode)

f = open('/tmp/workfile', 'w')
print f
<open file '/tmp/workfile', mode 'w' at 80a0960>
f.read() # reads the entire file
f.read(size) # reads at most size bytes returns a string

f.readline() # reads one line
f.readlines() # reads all linesreturns a LIST!
```

Reading and Writing

f.write(string) writes the contents of *string* to the file, returning None.

```
value = ('the answer', 42)
s = str(value)
f.write(s)
```

Always close files

```
f.close()
with open('index.html', 'w') as f:
    f.write("My web page was created by python')
```

will close f even if the write crashes!

PICKLING

If you have an object x, and a file object f that's been opened for writing, the simplest way to pickle the object takes only one line of code:

```
pickle.dump(x, f)
```

To unpickle the object again, if f is a file object which has been opened for reading:

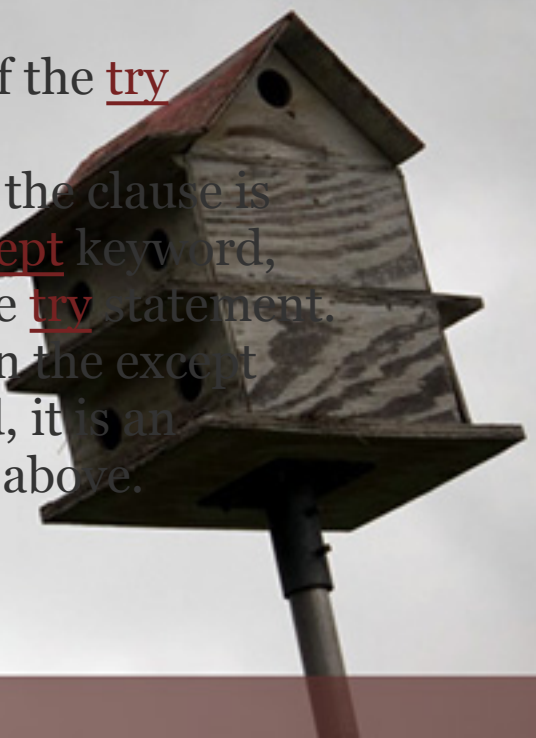
```
x = pickle.load(f)
```

Exceptional Exception handling

```
while True:
    try:
        x = int(raw_input("Please enter a number: "))
        break
    except ValueError:
        print "Oops! That was no valid number. Try again..."
```

The try statement works as follows.

- First, the *try clause* (the statement(s) between the try and except keywords) is executed.
- If no exception occurs, the *except clause* is skipped and execution of the try statement is finished.
- If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try statement.
- If an exception occurs which does not match the exception named in the except clause, it is passed on to outer try statements; if no handler is found, it is an *unhandled exception* and execution stops with a message as shown above.



Multiple excepts

```
import sys
 try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
 except IOError as (errno, strerror):
    print "I/O error({0}): {1}".format(errno, strerror)
 except ValueError:
    print "Could not convert data to an integer."
 except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
```

Classes

```
class ClassName:
    <statement-1> . . . <statement-N>

class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world'


x = MyClass()
```

CLASSES - CONSTRUCTORS

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart


x = Complex(3.0, -4.5)

> x.r, x.i

(3.0, -4.5)
```

CLASSES - WHAT IS SELF?

```
class Bag:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

Classes - Inheritance

```
class DerivedClassName(BaseClassName):
    <statement-1>
    . . .
    <statement-N>
```

Multiple Inheritance

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    . . .
     <statement-N>
```

Classes - Privacy

“Private” instance variables that cannot be accessed except from inside an object don’t exist in Python. However, there is a convention that is followed by most Python code: a name prefixed with an underscore (e.g. _spam) should be treated as a non-public part of the API (whether it is a function, a method or a data member). It should be considered an implementation detail and subject to change without notice.

CLASSES - DYNAMIC VARIABLES

```
class Employee:
    pass

john = Employee() # Create an empty employee record
# Fill the fields of the record

john.name = 'John Doe'
john.dept = 'computer lab'
john.salary = 1000
```

Iterators Unify Looping

```
for element in [1, 2, 3]:
    print element

for element in (1, 2, 3):
    print element

for key in {'one':1, 'two':2}:
    print key

for char in 123":
    print char

for line in open("myfile.txt"):
    print line
```

Python Standard Library

```
import os os.getcwd() # Return the current working directory
    'C:\\Python26'

os.chdir('/server/accesslogs') #Change current working directory
os.system('mkdir today')# Run the command mkdir in the system shell
0

import shutil
shutil.copyfile('data.db', 'archive.db')
shutil.move('/build/executables', 'installdir')


 import glob
 glob.glob('*.py')


 ['primes.py', 'random.py', 'quote.py']
```

Python Standarad Library

Command line args
% python demo.py one two three

import sys
print sys.argv ['demo.py', 'one', 'two', 'three']

String library

'tea for too'.replace('too', 'two')
'tea for two'

Math

```
import math
math.cos(math.pi / 4.0)
 0.70710678118654757
 math.log(1024, 2)
 10.0

import random
random.choice(['apple', 'pear', 'banana'])
'apple'
random.sample(xrange(100), 10)# sampling without replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
random.random() # random float
0.17970987693706186
random.randrange(6) # random integer chosen from range(6)
4
```

Internet

```
>>> import urllib2
 >>> for line in urllib2.urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl'):
       ... if 'EST' in line or 'EDT' in line: # look for Eastern Time
       ... print line <BR>
Nov. 25, 09:43:32 PM EST

 >>> import smtplib
 >>> server = smtplib.SMTP('localhost')
 >>> server.sendmail('soothsayer@example.org', 'jcaesar@example.org',
          ... """To: jcaesar@example.org
          ... From: soothsayer@example.org
          ...
          ... Beware the Ides of March. ... """)
 >>> server.quit()
```

Everything is so Easy

```
import threading, zipfile
class AsyncZip(threading.Thread):
    def __init__(self, infile, outfile):
        threading.Thread.__init__(self)
        self.infile = infile
        self.outfile = outfile

    def run(self):
        f = zipfile.ZipFile(self.outfile, 'w', zipfile.ZIP_DEFLATED)
        f.write(self.infile)
        f.close()
        print 'Finished background zip of: ', self.infile

background = AsyncZip('mydata.txt', 'myarchive.zip')
background.start()
print 'The main program continues to run in foreground.'
background.join() # Wait for the background task to finish

print 'Main program waited until background was done.'
```