An Overview of Python with Functional Programming

Adam Getchell

PHY 210

Attributes

- Portable, interpreted, object-oriented, strongly- and dynamically-typed, extensible
  - Can write extensions to Python in C/C++
  - Links to operating system functions using modules
- Open sourced, developed and supported by the Python Organization (http://www.python.org)
- Freely available on most platforms (Windows, Unix, Linux, *BSD, MacOS)
- Supports procedural, object-oriented, and functional programming
- Huge number of add-on modules available
  - Numarray provides capabilities similar to MatLab, IDL, or Octave
    - http://www.stsci.edu/resources/software_hardware/numarray
  - SciPy implements plotting, parallel computation, FFT, ODE solvers, and linear algebra/LAPACK/BLAS routines from http://www.netlib.org using thin python wrappers over C or FORTRAN code (and using WEAVE allows direct inlining of C/C++ code within Python)
    - http://www.scipy.org/
  - mxODBC provides ODBC database connectivity (e.g. Oracle, SQLServer, mySQL, PostgreSQL, DB2, Sybase)
    - http://www.egenix.com/files/python/mxODBC.html
  - ReportLab Open Source PDF Library generates PDF files
    - http://www.reportlab.org/
  - PyXML parses/handles XML with DOM, SAX, XPath/XPointer
    - http://pyxml.sourceforge.net/topics/download.html
  - Jython implements Python on top of Java, gaining access to the Java platform with Python syntax
    - http://www.jython.org
  - wxPython implements a cross-platform GUI library for GUI applications
    - http://www.wxpython.org/
  - MatPlotLib is a cross platform Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive GUI environments
    - http://matplotlib.sourceforge.net
- Runs large-scale software projects
  - Google
  - Hubble Space Telescope data processing pipeline
  - Zope and Plone
  - BitTorrent
  - Mailman

Tokens and Structure

Code Pattern	C	Python
Statements	Unlimited length, terminated with ";" Code blocks delineated by { }	One statement per line, unless explicitly joined by \ or implicitly joined across (), [], or {} Code blocks delineated by indentation level (tabs)
Comment	`/* Comment */`	`# Comment to end of line unless line joined implicitly`
Variables	Explicitly declared (statically typed) `int, long int, double, float`, etc	Objects have identity, type, and value. Identity is memory address, type determines supported methods and whether the object can be changed (mutable). Variables dynamically typed at runtime `Boolean, int` (equivalent to long), `decimal, float`, imaginary, Iterators, strings, Unicode strings, tuples, lists, buffers, xrange, set, frozenset, dictionaries,
Scoping	Global variables declared prior to `main()` Local variables declared in each procedure	Statically scoped block structure (local, global, built-in) with static lexical scoping. Lexical scoping defaults to namespace of enclosing function or class
Functions	Type and arguments declared	Name and parameters must be declared; can take variable list of arguments using `varargs()` Functions are first-class elements of language supporting Functional Programming using List Comprehensions, `lambda, map(), reduce(), filter()` and Generators with `yield()`

Matrices	`A[i][j]`	`Vector = array([1,2,3,4,5]) # from numarray`
		`Matrix = array([1,2],[3,4]) # from numarray`
		`Tuple = ( 1, 2, "three" ) # immutable, ordered, faster than list`
		`List = [ 4, 5, Tuple ] # mutable, ordered, expands dynamically as needed`
		`Dictionary = { "key1": "value1", "key2": "value2" } # mutable, unordered`

Open Files	`Input_file=fopen(filename, "r")`
	`Output_file=fopen(filename, "w")`
		`f = open("/music/_singles/kairo.mp3", "rb") # read binary mode`
		`g = open("/temp/outputfile.txt", "a")  # append mode`
		`h = open("/temp/outputfile.txt", "w")  # overwrite mode`

Input	`scanf(stdin, " %lf ", &var)` `fscanf(input_file, " %lf ", &var)`	`s = raw_input("Who goes there?")  or sys.stdin.readline()` `s = f.read()` or f.readlines()
Output	`printf(stdout, " %lf ", var)` `printf(stdout, " %lf ", var)`	`print s`
Relational Operators	`==, >=, <=, !=, <, >`	Python supports operator overloading `==,>=,<=,!=,<>, <, >, is, is not`
Logical Operators	`&&, ||, !`	`and, or, not`
Bitwise Operators	`&, |, ^, ~, >>, <<`	`&, |, ^, ~, >>, <<`

Tokens and Structure part II

Code Pattern	C	Python

Pointers	Declaration: int *p (p is the address of an integer, it points to an integer)
	
	int *p, value, var;
	var = 10;
	p = &var;
	Value = *p (Value = 10)
		Python does not directly have pointers
		However, everything in Python is an object
		Objects are passed by reference and can be compared using `is` and `is not` operators, which tests for equality by looking at object identity (memory address)
		Python modules and C-extensions can handle the encapsulation of C-pointers in Python objects

If Statements	`if (i<N)`
	`  statements;`
	`else if (i>N)`
	`  statements;`
	`else`
	`  statements;`
		`if i<N:`
		`  statements`
		`elif i>N:`
		`  statements`
		`else:`
		`  statements`

For Statements	`for (i=0; i < N; i++)` `  statements;`	`for i in range(N):` `  statements`
While Statements	`while (i<N)` `  statements;`	`while i < N:` `  statements`
Including files	`#include <stdio.h>`	`import sys`

Exception Handling	None	`for arg in sys.argv[1:]:`
		`  try:`
		`  f = open(arg, 'r')`
		`  except IOError:`
		`  print 'cannot open', arg`
		`  else:`
		`  print arg, 'has', len(f.readlines()), 'lines'`
		`  f.close()`

Classes	None	`class MyClass:`
		`  "A simple example class"`
		`  i = 12345`
		`  def f(self):`
		`  return 'hello world'`

Tokens and Structure part III

Code Pattern	C	Python

Compound Data Type Operations	None	`>>> a = ['spam', 'eggs', 100, 1234]`
		`>>> a ['spam', 'eggs', 100, 1234]`
		`>>> a[0]`
		`'spam'`
		`>>> a[3]`
		`1234`
		`>>> a[-2]`
		`100`
		`>>> a[1:-1]`
		`['eggs', 100]`
		`>>> a[:2] + ['bacon', 2*2]`
		`['spam', 'eggs', 'bacon', 4]`
		`>>> 3*a[:3] + ['Boe!']`
		`['spam', 'eggs', 100, 'spam', 'eggs', 100, 'spam', 'eggs', 100, 'Boe!']`

Multiple Inheritance	None	`class DerivedClassName(Base1, Base2, Base3):` `  statements`

Structs	`typedef struct`
	`{`
	`  int x;`
	`  int array[100];`
	`} Foo; /* note semi-colon here */`
		`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	None	`for element in [1, 2, 3]:`
		`  print element`
		The for loop calls `iter()` on element

Generators	None	`def reverse(data):`
		`  for index in range(len(data)-1, -1, -1):`
		`  yield data[index]`
		`>>> for char in reverse('golf'):`
		`... print char`
		`...`
		`f`
		`l`
		`o`
		`g`

Script

```
# Need this to run console commands
import os
# Need this to send email
import smtplib
# Need this to process email
from email.MIMEText import MIMEText
# Need this for time
from time import strftime

# First, run the webchecker script and write results to file
os.system("webchecker.py -q http://169.237.48.10 > results.txt")

# Get local time
runtime = strftime('%a %d %b %Y %H:%M:%S')
# Open the file
fp = open("results.txt")
# Create message
msg = MIMEText(fp.read())
fp.close()

me = "Webnanny"
you = "AdamG@hrrm.ucdavis.edu"
msg['Subject'] = 'www.hr.ucdavis.edu link check %s' % runtime
msg['From'] = me
msg['To']= you

server = smtplib.SMTP('hrrm.ucdavis.edu')
server.set_debuglevel(1)
#server.connect()
server.sendmail(me, you, msg.as_string())
server.quit()
```

Procedural Program

```
"""Delete files older than AGING constant using os.path.walk"""
import sys, os, time
AGING = 172800  # Default 2 days
def lister(dummy, dirName, filesInDir):
    print '[' + dirName + ']'
    for fname in filesInDir:
        path = os.path.join(dirName, fname)
        if not os.path.isdir(path):
            print path, time.ctime(os.stat(path).st_mtime), fileage(path),
    prune(path)
            purge(path)
def fileage(file):
    curtime = time.time()
    modtime = os.stat(file).st_mtime
    age = curtime - modtime
    return age
def prune(file):
    if fileage(file) > AGING:
        return ("T")
    else:
        return ("File should not be deleted")
def purge(file):
    if prune(file) == "T":
        os.remove(file)
if __name__ == '__main__':
    os.path.walk(sys.argv[1], lister, None)
```

Object Oriented Program

```
class Employee:
    def __init__(self, name, salary = 0):
        self.name = name
        self.salary = salary
    def giveRaise(self, percent):
        self.salary += self.salary * percent
    def work(self):
        print self.name, "does stuff"
    def __repr__(self):
        return "<Employee: name=%s, salary=%s>" % (self.name, self.salary)

class Chef(Employee):
    def __init__(self, name):
        Employee.__init__(self, name, 50000)
    def work(self):
        print self.name, "makes food"

class Server(Employee):
    def __init__(self, name):
        Employee.__init__(self, name, 40000)
    def work(self):
        print self.name, "interfaces with customers"

class PizzaRobot(Chef):
    def __init__(self, name):
        Chef.__init__(self,name)
    def work(self):
        print self.name, "makes pizza"

if __name__ == "__main__":
    bob = PizzaRobot('bob')
    print bob
    bob.giveRaise(0.2)
    print bob; print

    for klass in Employee, Chef, Server, PizzaRobot:
        obj = klass(klass.__name__)
        obj.work()
```

Functional Programming

- Functions are first class objects. Everything done with "data" can be done with functions (e.g. passing a function to another function).
- Recursion is used as the primary control structure. This allows the compiler to determine the layout of program statements rather than the programmer (e.g., loops don't have to be "unrolled" for efficiency)
- There is a focus on LIST Processing (for example, the name Lisp). Lists are often used with recursion on sub-lists as a substitute for loops.
- "Pure" functional languages eschew side-effects. This excludes the almost ubiquitous pattern in imperative languages of assigning first one, then another value to the same variable to track the program state.
- FP either discourages or outright disallows *statements*, and instead works with the evaluation of expressions (in other words, functions plus arguments). In the pure case, one program is one expression (plus supporting definitions).
- FP worries about *what* is to be computed rather than *how* it is to be computed.
- FP uses "higher order" functions (functions that operate on functions that operate on functions) [1]
- Note: "Currying" allows functions of more than one variable to be converted to composite functions of one variable, e.g. $F(x,y,z) = g(x) \circ h(y) \circ i(z)$

Functional Code

Code Pattern	Procedural	Functional

If Statements	`if i=1:`
	`  namenum(i)='one'`
	`elif i=2:`
	`  namenum(i)='two'`
	`else:`
	`  namenum(i)='other'`
		`>>> pr = lambda s:s`
		`>>> namenum = lambda x: (x==1 and pr("one")) \`
		`.... or (x==2 and pr("two")) \`
		`.... or (pr("other"))`
		`>>> namenum(1)`
		`'one'`
		`>>> namenum(2)`
		`'two'`
		`>>> namenum(3)`
		`'other'`

For Statements	`for e in list: func(e)`	`map(func, list)`

While Statements	`while <cond>:`
	`  <pre-suite>`
	`if <break_condition>:`
	`  break`
	`else:`
	`  <suite>`
		`def while_block():`
		`  <pre-suite>`
		`  if <break_condition>:`
		`  return 1`
		`  else:`
		`  <suite>`
		`  return 0`
		`while_FP = lambda: (<cond> and while_block()) or while_FP()`

Example Program	`xs = (1,2,3,4)`
	`ys = (10,15,3,22)`
	`bigmuls = []`
	`# ...more stuff...`
	`for x in xs:`
	`  for y in ys:`
	`  # ...more stuff...`
	`  if x*y > 25:`
	`  bigmuls.append((x,y))`
	`  # ...more stuff...`
	`  # ...more stuff...`
	`print bigmuls`
		`# No side-effects possible`
		`bigmuls = lambda xs,ys: filter(lambda (x,y):x*y > 25, combine(xs,ys))`
		`combine = lambda xs,ys: map(None, xs*len(ys), dupelms(ys,len(xs))) dupelms = lambda lst,n: reduce(lambda s,t:s+t, map(lambda l,n=n: [l]*n, lst))`
		`print bigmuls((1,2,3,4),(10,15,3,22)) #replace with list comprehension`

List Comprehensions	None	`print [(x,y) for x in (1,2,3,4) for y in (10,15,3,22) if x*y > 25]`

Functional Code part II

Code Pattern	C	Python

QuickSort	`void q_sort( input_type a[], int left, int right )`
	`{`
	`  int i, j;`
	`  input_type pivot;`
	`  if (left + CUTOFF <= right) {`
	`  /* CUTOFF should be 20 but example is 2 */`
	`  pivot = median3( a, left, right );`
	`  i = left; j = right-1; /* why? */`
	`  for (;;) {`
	`  while (a[++i] < pivot);`
	`  while (a[--j] > pivot);`
	`  if (i < j)`
	`  swap( &a[i], &a[j] );`
	`  else`
	`  break;`
	`  }`
	`  swap( &a[i], &a[right-1] ); /* why? */`
	`  q_sort( a, left, i-1 );`
	`  q_sort( a, i+1, right);`
	`  }`
	`}`
	
	`void quick_sort( input_type a[], int n ) {`
	`  q_sort( a, 1, n);`
	`  insertion_sort( a, n );`
	`/* n<=20: insertion is better, use CUTOFF */ }`
		`def qsort(L):`
		`  if L == []: return []`
		`  return qsort([x for x in L[1:] if x< L[0] ]) + L[0:1] + \`
		`  qsort([x for x in L[1:] if x>=L[0] ])`

References

1. ActivePython from ActiveState
   - http://www.activestate.com/Products/ActivePython/
2. Charming Python: Functional programming in Python, Part 1
   - http://www-106.ibm.com/developerworks/library/l-prog.html
3. Currying: From Wikipedia
   - http://en.wikipedia.org/wiki/Currying
4. Enhanced Python Distribution from EnThought
   - http://www.enthought.com/python/
5. MatPlotLib: SourceForge
   - http://matplotlib.sourceforge.net/
6. Numarray: Space Telescope Science Institute
   - http://www.stsci.edu/resources/software_hardware/numarray
7. Python - Object/Type Structures
   - http://wiki.cs.uiuc.edu/cs427/Python+-+Object%2FType+Structures
8. Python Software Foundation
   - http://www.python.org/
9. Python Programming Language: From Wikipedia
   - http://en.wikipedia.org/wiki/Python_programming_language