STSCI Python Introduction

Class 2
Jim Hare

Class URL
www.pst.stsci.edu/~hare

• Each Class Presentation
• Homework suggestions
• Example files to download
• Links to sites by each class and in general
• I will try to have the PowerPoint slides by Friday AM before each Monday’s class.
Today’s Agenda

- Object Oriented Concepts
- Built-in Types
- Operators
- Formatting to Strings
- Control Flow
- Functions
- Passing Arguments to Modules

Object Oriented Concepts

- Every piece of data stored in a program is an object.
- Each object has an identity, a type, and a value.
- When an object is created, the object created is called an instance of that type.
Object Identity

– *Identity* – pointer or reference to the location in memory.

```python
>>> a = 42  # create an instance of an integer a
>>> id(a)  # gets the memory location for object a
13074924
```

Object Type

• The *type* of an object describes the internal representation of the object, the methods and operations that it supports.
• Having the data, methods and operations defined within an object is call *encapsulation*.
• This is what is different from old programming techniques. You don’t have to know how the object is implemented. You just need to know the interface.
Example of type()

```python
>>> nameString = "Zach Hare"
>>> type(nameString)
<type 'str'>
>>> nameList = ['Zach','Hare']
>>> type(nameList)
<type 'list'>
>>> nameDict = {'first':'Zach','last':'Hare'};type(nameDict)
<type 'dict'>
>>> age = 12;type(age)
<type 'int'>
```

Object Methods and Attributes

- A *method* is a function that performs some sort of operation on an object when the method is invoked.
- An *attribute* is a property or value associated with an object.
- Attributes and methods are accessed using the dot (.) operator.
Example of Methods and Attributes

```python
>>> a = 3 + 4j  # Create a complex number
>>> r = a.real  # Get real part (an attribute)
>>> print r
3.0
>>> b = [1,2,3]
>>> b.append(7)  # add element 7 to the list with
              # method append
>>> b
[1, 2, 3, 7]
```

Built-in Types

- None Type
- Numeric Types
- Sequence Types
- Mapping Types
The None Type

- The None type denotes a null object.
- None has no attributes and evaluates to false in Boolean expressions.
- Often used as a default value in optional arguments.

```python
X = None
```

Numeric Types

- Integers - whole numbers from -2147483647 to 2147483647 in 32 bits, 4 bytes.
  Some machines the range is larger 8 bytes.
- Long integers – whole numbers limited only by available memory.
- Floating-point – uses machines double-precision 64-bit, 8 bytes.
- Complex numbers – a pair of floating-point numbers. Z has Z.real and Z.imag.
# Operators and the Order of Evaluation

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>(…), […], […]</td>
<td>Tuple, list, dictionary creation</td>
</tr>
<tr>
<td>‘…’</td>
<td>String conversion</td>
</tr>
<tr>
<td>s[i], s[i:j], s.attr</td>
<td>Indexing and slicing attributes</td>
</tr>
<tr>
<td>f(...)</td>
<td>Function calls</td>
</tr>
<tr>
<td>+x, -x, ~x</td>
<td>Unary operators</td>
</tr>
<tr>
<td>x**y</td>
<td>Power(right associative)</td>
</tr>
<tr>
<td>x * y, x / y, x % y</td>
<td>Multiplication, division, modulo</td>
</tr>
<tr>
<td>x + y, x – y</td>
<td>Addition, subtraction</td>
</tr>
<tr>
<td>x &lt;&lt; y, x &gt;&gt; y</td>
<td>Bit shifting</td>
</tr>
<tr>
<td>x &amp; y</td>
<td>Bitwise and</td>
</tr>
<tr>
<td>x^ y</td>
<td>Bitwise exclusive or</td>
</tr>
</tbody>
</table>

# Operators and the Order of Evaluation

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>x &lt; y, x &lt;= y,</td>
<td>Comparison, identity, and sequence</td>
</tr>
<tr>
<td>x &gt; y, x &gt;= y,</td>
<td>membership tests</td>
</tr>
<tr>
<td>x == y, x != y</td>
<td></td>
</tr>
<tr>
<td>x &lt;&gt; y</td>
<td></td>
</tr>
<tr>
<td>x is y, x is not y</td>
<td></td>
</tr>
<tr>
<td>x in s, x not in s</td>
<td></td>
</tr>
<tr>
<td>not x</td>
<td>Logical negation</td>
</tr>
<tr>
<td>x and y</td>
<td>Logical and</td>
</tr>
<tr>
<td>x or y</td>
<td>Logical or</td>
</tr>
<tr>
<td>lambda args: expr</td>
<td>Anonymous function</td>
</tr>
</tbody>
</table>
Operator Precedence Examples

>>> x = 2; y = 3
>>> z = x + y**2 % 6
>>> z
5
>>> z = x + (y**2) % 6

Sequence Types

• Sequences represent ordered sets of objects indexed by non-negative integers and include strings, Unicode strings, lists, tuples, xrange objects, and buffer objects.
• Strings and buffers are sequences of characters.
• xrange objects are sequences of integers.
• Lists and tuples are sequences of arbitrary objects.
List Type Example

```python
>>> xList = [[0,1,2],
            [3,4,5],
            [6,7,8]]
>>> xList [0][0]
0
>>> xList [0][2]
2
>>> xList [2][1]
7
X[row][column]
```

String Splicing

```python
>>> xString = 'I am learning NNNNNN today'
>>> xString[14:20] = 'Python'
Traceback (innermost last): File “
<interactive input>”, line 1, in ?
TypeError: object doesn't support slice assignment
>>> xString = xString[:14] + 'Python' + \
            xString[20:]
>>> xString
'I am learning Python today'
```
Mapping Types

- A mapping object represents an arbitrary collection of objects that are indexed by another collection of nearly arbitrary key values.
- Mapping objects are unordered and keys are immutable. The mappings are mutable.
- Dictionaries are the only built-in mapping type.

Formatting to Strings

- X = “Python”
- Y = “I am learning %s today” % X

- X = 3.146789
- Y = “The real number is %-10.4f” % X
- The real number is 3.1468
String Formatting Conversions

<table>
<thead>
<tr>
<th>Char</th>
<th>Format</th>
<th>Char</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>d,i</td>
<td>Decimal integer of L</td>
<td>u</td>
<td>Unsigned integer or L</td>
</tr>
<tr>
<td>o</td>
<td>Octal integer or L</td>
<td>x</td>
<td>Hexadecimal integer or L</td>
</tr>
<tr>
<td>X</td>
<td>Hexadecimal integer</td>
<td>f</td>
<td>Floating Point [-]m.ddddd</td>
</tr>
<tr>
<td></td>
<td>Uppercase letters.</td>
<td>e</td>
<td>[-]m.dddddE+xx</td>
</tr>
<tr>
<td>E</td>
<td>[-]m.ddddde+xx</td>
<td>G,g</td>
<td>Use %E or %e for exponents less than –4 or greater than the precision; otherwise use %f</td>
</tr>
<tr>
<td>s</td>
<td>String or any object.</td>
<td>r</td>
<td>Same string as repr()</td>
</tr>
<tr>
<td></td>
<td>The formatting code use str() to generate strings.</td>
<td>%</td>
<td>Literal %</td>
</tr>
</tbody>
</table>

String Formatting Examples

```python
a = 42
b = 13.142783
c = “hello”
d = {‘x’:13, ‘y’:1.54321, ‘z’:’world’}
‘a is %d’ % a        #“a is 42”
‘%10d %f’ % (a,b)    # “  42 13.142783”
‘%(x)-10d %(y)0.3g’ %d ”13    1.54”
‘%*.f’ % (5,3,b)     # “13.143”
```
Control Flow

- Supports if, for, while, and try
- Blocks of code executed within a control flow are indented.
- It is a syntax error when indentation is not correct
- Boolean expressions “or”, “and”, and “not" are used for Truth Values
- pass  # do nothing

Control Flow
Conditionals

```python
if expression:
    statements
elif expression:
    statements
elif expression:
    statements
else:
    statements
#endif  Note: You don’t need this but it is wise to keep you organized sometimes.
```
Loops

while

while expression:
  statements
  if expression:
    break
  else:
    statements

Loops

For

for i in range(0,10,2):  # ie.(start,stop,step)
  statements
  if Istring[I] == 'foo':
    break
else:
  statements
#endfor not necessary
range() and xrange()

- range(start, end, stride)
- range constructs a list and populates values according to start, end and stride (step size).
- Remember end is an up to value
- xrange() – value calculated when accessed; it save memory

A = range(5) # A = [0,1,2,3,4]
D = range(8,2,-2) # D = [8,6,4]
b = xrange(1000000) # b = [0,1,2, ..., 999999]

Try and Except Control

- Try and Except blocks are a way to override python stopping the execution of a program due to a system error.
- The except block provides a way to execute statements on a specific error or set of errors
- Except blocks without a specified type of error execute the statements for any error.
Try and Except Block

try:
    statements
except:
    statements to handle error #could just use pass

Except: Error Types

try:
    ...
except IOError, e:  # e is optional args of
    raise
    ...
except TypeError:
    • See Appendix A, “The Python Library” for details
    of exception errors
Passing Arguments to Modules

- Using the python module sys
  - import sys
  - sys.argv – a list containing the module name in `sys.argv[0]` followed by arguments passed
  - sys.path – path used to find modules, uses PYTHONPATH environment variable

Passing Arguments to Modules

os module

- os.environ – is a dictionary of the current environment variables
  - print os.environ['DSQUERY']
    nomad
  - os.environ['DSQUERY'] = “R2D2”
  
**Note:** This will only change the environment variable for your current program run

- Example /ess5/psdpython/dbtools.py
os.system

• os.system(‘pwd’)  
• os.system(‘echo $DSQUERY’)  
• The shell is the basic c shell not tcsh.

Mail Example

import os
toList = [{'name': ’Paul’, ’email’: ’plee@stsci.edu’},  
           {'name': ’Erin’, ’email’: ’roye@stsci.edu’}]
for toDict in toList:
    mailString = “Hi %s,
The Python class is today at 11:30.
Jim” % toDict[’name’]
    os.system(“printf ‘%s’ | mailx –s ‘Class Today’ “  
               “–r hare@stsci.edu %s “  
               “%(mailString,toDict[’email’])”)
Functions

- Function template
- Consists of def statement, parameters and return
  
  ```python
  def test1(x,y,j='testcase1')
  j='testcase1' is the default
  Example
  /ess5/psdpython/pythonclass/class2/update_default_db.py
  ```

Passing Arguments –

```python
if __name__== '__main__':
    if __name__ == '__main__':
        if len(sys.argv) != 3:
            print "Error in arguments try again."
            sys.exit()
        database = string.lower(sys.argv[1])
        account = sys.argv[2]
        run(database,account)
```
Passing Arguments - continued

• __main__ is the name of the originating or first module run in the current execution of python

```python
def run(database, account='sogsspss'):
    os.system('printf "sp_defaultdb %s,%s\ngo" | isql' % (account, database))
    os.system('printf "select db_name()\ngo" | isql')
    return None
```

Functions Returning Values

def splitonperiod(inString):
    import string
    stringList = string.split(inString,'.')
    ...
    test = 1
    return test,stringList

aString = "md.summer.2002.STSCI.EDU"
status,varList = splitonperiod(aString)
Raw Input Command

```python
first = raw_input('Type your first name: 
mid = raw_input('Type your middle name: 
last = raw_input('Type your last name: 
print "Your full name is %s %s %s"
    % (first,mid,last)
• Example: myName.py
```

Files

• myFile = open('build_id.data', 'r')
• File modes are: ‘r’, ‘w’, or “a”. ‘b’
  following means binary ie. ‘rb’
• Updates ‘r+’ or ‘w+’, can perform both
  input and output, as long as output
  operations flush their data before
  subsequent input operations ie. f.flush()
File Methods - Read

- F.read - Reads at most n bytes
- F.readline() - Reads a single line of input
- F.readlines() - Reads all the lines and returns a list.

I.e. myList = F.readlines()

File Methods - Write

- F.write(S) - Writes string S
- F.writelines(L) - Writes all strings in List L
File Methods
• F.close() - closes the file F
• F.tell() - returns the file pointer
• F.seek(offset [,where]) – seek to new file position
• F.isatty() - 1 if F is interactive terminal
• F.flush() - Flushes output buffers
• F.truncate([size]) - truncates to size bytes
• F.fileno() - integer descriptor

File Attributes
• F.closed - Boolean 0 = open, 1=closed
• F.mode - I/O mode of file
• F.name - name of file or source
• F.softspace - indicator of whether a space character needs to be printed before another value when using the print statement.
Standard Input, Output, and Error

• import sys - to get access to Standard I/O files
• sys.stdin - Standard Input
• sys.stdout - Standard Output
• sys.stderr - Standard Error

Examples and Homework

• Look at:
  – osEnv.py – a module to create environment variable setup scripts for regression tests.
  – osReg.py – a module to run a test in two different environments and compare results.
  – Download these to your working directory.
  – From python import osEnv
  – osEnv.makeEnvFile(‘test1’)
  – Get out of python and look at envtest1.py that you just generated