A Little Python – Part 3

Introducing Programming with Python

I/O, Files, Object Classes, Exception Handling

Outline

- I/O
  - Files opening
  - File I/O, reading writing
- Python Objects
  - Defining a new object
  - Inheritance
- Exceptions
  - Try clauses

Reminder - Learning ANY Programming Language

- **Syntax**
  - What is a valid statement
  - What statements have meaning
- Variables, data types, **data structure**
- **Control flow, branching, testing, loops, iteration**
- Input/Output, I/O, read/write files
- Procedures, subroutines
- Objects, encapsulation of code + data

Reminder - Learning ANY Programming Language

- **Syntax**
  - What is a valid statement
  - What statements have meaning
- Variables, data types, data structure
- Control flow, branching, testing, loops, iteration
- **Input/Output, I/O, read/write files**
- Procedures, subroutines
- **Objects, encapsulation of code + data**

Files

- Need a way to read input and write output
  - A simple example with print
  - Files are the more general way
- Open

```
open(<filename>,<mode>)
f = open("/Users/dwmc/Development/python/foo.txt","w")
```

- <mode> - a (append), w (write), r (read), b (binary)
  - w overwrites an existing file

File functions

- Result of open() is a File object, with methods

```
f.readline()
f.read()
f.write()
f.close()
```

- Can treat a file with an iterator

```
for line in f:
   statements (block)
```

Sample open and read

```
def readFile(fname=None):
    llist = []
    if fname:
        count = 0
        fin = open(fname, "r")
        text = fin.readline()
        while text != "":
            count += 1
            text = text.rstrip()
            print "%4d:"%(count),text
            llist.append(text)
            text = fin.readline()
        fin.close()
    else:
        print "Must supply a filename"
    return llist
```

Sample open and write

```
def writeFile(fname=None,line_list=[]):
    if fname and line_list!=[]:
        fout = open(fname,"w")
        for line in line_list:
            fout.write(line)
            fout.write("\n")
        fout.flush()
        fout.close()
```

Formatted Output

- Output can be formatted
  - Output string
  - Templates (not covering this approach to output)
- Generic string format
  - Sometimes called "percent" formatting
  - Start a format with % character
  - `%<flag><width>.<precision><length><type>`
- Width and precision are just integer values

Formatted Output

- Formatting
  - `%<flag><width>.<precision><length><type>`
- Output conversion types
  - s - string, using str()
  - r - string, using repr()
  - c - character
  - d - integer
  - f - float
  - e or E - float exponential representation
  - x or X - hexidecimal

Formatted Output

- Formatting
  - `%<flag><width>.<precision><length><type>`
- Flags
  - 0 - zero padded
  - - - (hyphen character) left justify
  - + - signed value
  - ␣ - (space character) right justify
- Length
  - h, l, or L - for “long” integers

Formatting Examples

```
name = "this is a string called 'name'"
number = 253
print "string %s"%(name)
print "string %r"%(name)
print "string % 25s"%(name)
print "number %+d"%(number)
print "number %4ld"%(number)
print "float %4.3e"%(number)
print "hex %x"%(number)
```

System Files, Printing Anywhere

- System input, output, error

```
import sys
…
sys.stdin
sys.stdout
sys.stderr
```

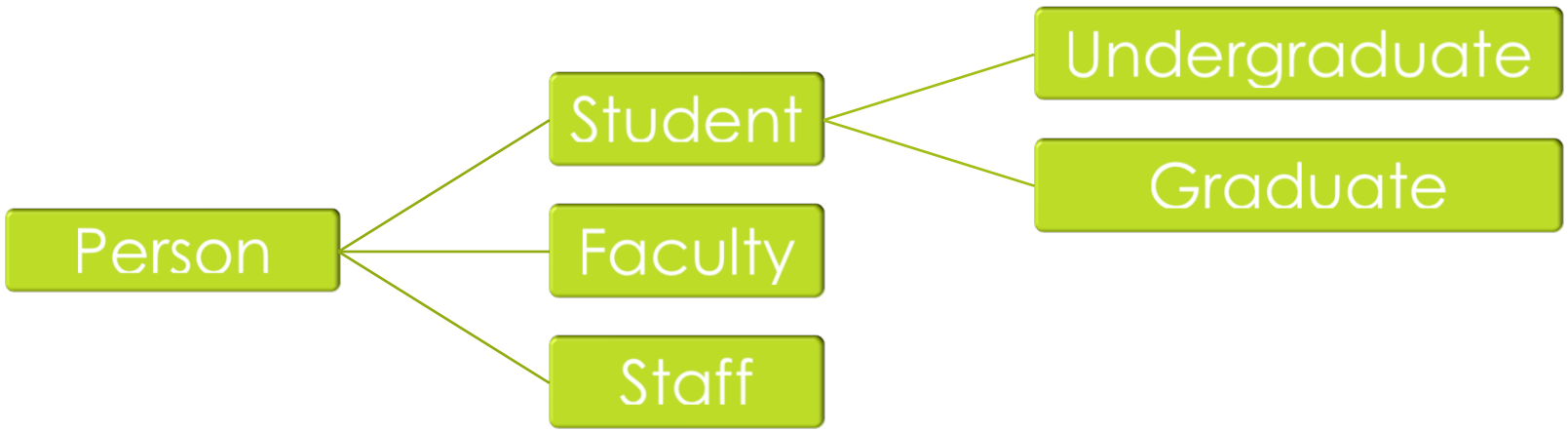
- Printing output into to files

```
foo = open("new-text-file.txt","a”)
print >>foo, "A string going to the file foo"
```

Object Oriented Python

- Python supports objects
  - Object inheritance
  - Polymorphism based on normal variable usage
- Generally one python object per file
  - But not a requirement

Simple Class Model



Person Object

```
class Person(object):
    def __init__(self, lastname=None, firstname=None):
        self.lastname = lastname
        self.firstname = firstname
        self.lastfirst = True
    def __repr__(self):
        return "<Person: lastname(%s), firstname(%s)>"%(self.lastname,self.firstname)
    def getName(self):
        if self.lastfirst:
           return "%s, %s"%(self.lastname,self.firstname)
        else:
           return "%s %s"%(self.firstname,self.lastname)
    def setLastname(self, lastname=""):
        self.lastname = lastname
    def setFirstname(self, firstname=""):
        self.firstname = firstname
```

Student Object

```
from Person import Person
class Student(Person):
    def __init__(self,  lastname=None,  firstname=None, year=0):
        Person.__init__(self,lastname=lastname,firstname=firstname)
        self.year = year
        self.studentID = 0
    def __repr__(self):
        return "<Student: lastname(%s), firstname(%s), year(%d), ID(%09d)>"%
(self.lastname,self.firstname,self.year,self.studentID)
    def setYear(self, year=0):
        self.year = year
    def setStudentID(self, studentID=0):
        self.studentID = studentID
```

Example Objects

```
>>> import Person
>>> import Student
>>> p1 = Person.Person("Taylor","Sam")
>>> p2 = Student.Student("Smith","Tom",3)
>>> p1
<Person: lastname(Taylor), firstname(Sam)>
>>> p2
<Student: lastname(Smith), firstname(Tom), year(3), ID(000000000)>
>>> p2.setStudentID(234509)
>>> p2
<Student: lastname(Smith), firstname(Tom), year(3), ID(000234509)>
>>> p2.setLastname("Bergman")
>>> p2
<Student: lastname(Bergman), firstname(Tom), year(3), ID(000234509)>
>>> p1
<Person: lastname(Taylor), firstname(Sam)>
>>>
```

Exception Handling

- Things can go wrong

```
>>> k = int("567")
>>> print k
567
>>> k = int("a")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'a'
>>>
```

Exception Handling

- Exceptions can provide a way to detect or recover from errors
- Language

```
try:
   <statements>
except <except_name>, <value>:
   <statements>
except (<except_name>, <except_name>), value:
   <statements>
```

Example – Exception Handling

```
def isLongString(s):
    try:
        long(s)
        return True
    except ValueError:
        return False

def isIntString(s):
    try:
        int(s)
        return True
    except ValueError:
        return False
```

Sample – Exception Handling

```
>>> kint = "123"
>>> kstr = "aat"
>>> print isIntString(kstr)
False
>>> print isIntString(kint)
True
>>>
```

Assignment 3

- Write 2 short programs

1. Write a short program that reads input lines from standard input and writes the line to a file, prepending a three digit line number and a colon. If the user enters two blank lines, stop writing lines, and close the file.

2. Subclass the "Person" object to create a "Faculty" object type. Faculty should have a field called "rank" which can be one of "lecturer", "assistant", "associate" or "full" and a boolean value for whether the faculty member is tenured or not. Modify __repr__ to do the right thing and create set/get methods for Faculty specific fields.