§6.5-6.10: Writing Library Modules

- *Announcements*

12 Oct 2006
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY
WESTERN
UNIVERSITY

Review of §6.1-6.4

- Working with files: open(), close()
  - File handles / file objects
- Input: read(), readline(), readlines()
- Output: write(), flush()
- The file position pointer: seek(), tell()
- Standard I/O channels: sys.stdin, stdout, stderr
- Python standard math library

Addendum on files and paths

- Specifying file pathnames: use forward slash
  - open('z:/directory/file.txt')
- Changing the current directory:
  - import os
  - os.chdir('z:/directory/')
  - open('file.txt')

Library modules vs. programs

- So far we've been writing Python programs (e.g., helloworld.py)
- Our programs have used library modules (e.g., import math)

- Libraries group related code for reuse (import)
  - Only need to define cos() once
  - Libraries are not intended to be executed (called), unlike programs
- We can create our own libraries for others to use

Designing libraries

- In creating a library, we need to decide what the public interface is: how programs can use it
  - Functions, types, constants, etc. for public use
  - Think about pre-/post-conditions
- We can hide implementation details
  - Certain functions may be for internal use only
- Car: how to use it vs. how it works
  - Owner's manual vs. shop manual
  - A driver doesn't need to understand how the engine works, variable valve timing/lift, etc.

Definition vs. implementation files

- In M2, each library has a definition file and an implementation file:
  - DEF: declares types and procedures
    - Tells programs how to invoke its procedures
    - No bodies to the procedures
  - IMP: implements the procedures
    - Parameter lists must match those in DEF file
- In C/C++, definition files are called header files (.h, .H, .hpp)
- In Python, everything is in one .py file

Example: Fractions ADT

- Often modules are used to define abstract data types: let's make a fraction type: fraction.py
- We can represent a fraction a/b internally as tuple of integers: (a, b)
- Our fractions module will contain the fraction type as well as all the procedures we need to use variables of type fraction
- We want to hide the internal representation as much as possible, so that a program using our library thinks just in terms of the fraction ADT.

Basic fractions functions

- Create a new fraction object:

```
def create(numer, denom):
    """Return a new fraction object.
    Pre: numer and denom are ints; denom != 0.
    """
    return (numer, denom)          # a tuple
```

- Access the internal representation:

```
def get_n(frac):
    """Return the top of the fraction."""
    return frac[0]
def get_d(frac):
    """Return the bottom of the fraction."""
    return frac[1]
```

Accessor (set/get) functions

- Why have get_n() and get_d()?
  Why not just access frac[0] and frac[1] directly?
- Want to hide the fact that our fractions are really just tuples
- Future version could store fractions differently
  - Then just change implementation of get_n() and get_d()
  - Public interface stays the same
- Can also protect against setting a zero denominator

Library functions: invert(), mult()

- Swap numerator and denominator:

```
def invert(frac):
    """Return the reciprocal of the fraction."""
    if get_n(frac) == 0:
        return 1/0                  # raise ZeroDivisionError
    return (get_d(frac), get_n(frac))
```

- Multiply two fractions:

```
def mult(f1, f2):
    """Multiply f1 and f2.  Doesn't cancel common factors."""
    return (get_n(f1) * get_n(f2), get_d(f1) * get_d(f2))
```

- Divide?

Library functions: string()

- Provide a way to pretty-print a fraction:

```
def string(frac):
    """Return a string representation of the fraction."""
    return "%d / %d" % (get_n(frac), get_d(frac))
```

- Library: http://twu.seanho.com/python/fraction.py

Using our library

- Import our library:
  - fraction.py must be in same directory

```
import fraction
```

- Create a couple fractions:

```
f1 = fraction.create(2,3)
f2 = fraction.create(6,7)
```

- Multiply them:

```
f3 = fraction.mult(f1, f2)
```

- Print the result:

```
print fraction.string(f3)
```

Doing this the object-oriented way

- Object-oriented design is organized around the data structure:
  - Build up a suite of functions to use the ADT
- The "real" Python way of writing a fractions ADT is to create a fractions class
  - Classes are user-defined data types
  - Can really hide implementation from user
  - Functions are methods of the class
    - e.g., myFile.read() is a method on file objects
- To see fractions done the OO way: http://twu.seanho.com/python/thinkCS/app_b.html

TODO items

- HW06 due tomorrow: 6.11 #(4, 28)
  - #28: show your Python program
- Lab05 due next week: 6.11 #(33/35)
- Quiz05 (ch6) on Mon
- CMPT140 Final in two weeks: W-Th 25-26Oct