Objectives

- Review solutions for more secure programs
- "Helper" Methods
- Group work: Designing Classes

Handling Exceptions

- Using try/except statements
- Syntax:

```
try:
        <body>
except [<errorType>]:
        <handler>
```

- Example:

```
try:
        age = input("Enter your age: ")
        currentyear = input("Enter the current year: ")
except:
        print "ERROR: Your input was not in the correct form."
        print "Enter integers for your age and the current year"
        return
```

birthyear2.py

`input` as a security hole

- **`input`** is actually **`eval(raw_input(…))`**
- How to exploit?

`input` as a security hole

- **`input`** is actually **`eval(raw_input(…))`**
- How to exploit?
  - Know/guess variable names
  - Use correct Python syntax to be evaluated
- How to fix?
  - Python: in the future, only **`raw_input`** will be allowed
  - Our code: inside a try/except statement, use **`raw_input`** and then cast as an int or float

Designing Bank Classes Summary

- **Nouns** are our classes/objects
- **Verbs** are the methods called on the classes/objects

"Helper" Methods

- Sometimes, you may need helper methods that are part of the class but are not meant to be part of the class's API
  - Make your code cleaner/easier
  - Only call from inside the object
  - Others outside the class shouldn't use
    - Known as "private" methods in other languages
- Convention: method name begins with "_"
- Called as **`self._method(…)`**

Example Helper Methods

"Helper" Method

```
def _isFaceCard(self):
    if self.rank > 10 and self.rank < 14:
        return True
    return False
```

```
def rummyValue(self):
    if self.rank == 14:
        return 15
    elif self._isFaceCard():
        return 10
    else:
        return 5
```

- Only loosely enforced that others can't use
  - dir, help

card4.py

Designing a Music Manager

- Create a music manager that
  - Reads your music library from a file
  - Displays the songs in your music library
  - Stores your music library in a file
  - Allows you to add songs to your library from a file
  - Keeps track of the total length of your music library
  - Allows you to sort the songs in your library
  - Provides user interface to do these things

Designing a Music Manager

- Break down into pieces
- What classes do we need?
  - What data needed to model those classes?
  - What functionality do each of those classes need?
- What does our driver program do?
- How should we implement those classes/program?

Designs

- For each of your classes
  - Data
  - API

Problem: Album Music Files

- Given an album file that has the format
  - <Artist name>
  - <Album name>
  - <number of songs>
  - <Song name 1>
  - <Song length 1>
  - …
  - <Song name n>
  - <Song length n>
- Create Song objects

Length has the format min:seconds

Problem: Library Music Files

- Given a library file that has the format
  - <number of songs>
  - <Song artist 1>
  - <Song album 1>
  - <Song name 1>
  - <Song length 1>
  - …
  - <Song artist n>
  - <Song album n>
  - <Song name n>
  - <Song length n>
- Create a MusicLibrary object

Music Manager Classes/Driver Data

- MusicLibrary
  - Songs
  - Total length
  - Filename
- Song
  - Title
  - Artist name
  - Album name
  - Length
- PlayTime
  - Days, hours,
  - Minutes, seconds
- Driver
  - Music library

What are the data types for each of these?

MM Classes/Driver Functionality

- MusicLibrary
  - Getters
  - String rep
  - Saving library to file
  - Adding albums
  - Sorting
- Song
  - Getters
  - String rep
  - Comparator
  - Writing to a file
- PlayTime
  - Getters, String rep
  - Adding play time
- Driver
  - Getting user input to
    - Read library, album files
    - Store library to file
    - Sort songs
    - View songs
    - Summary: Call appropriate methods on classes to do above

Exam Review

- Added 3 points to all tests, in case delayed test caused problems
  - Mean: 86.3
  - Median: 89.5
- Most difficult part: B (avg - 73%; med - 75%)
  - Understanding OO programming
    - Should see major improvement on final after more practice
  - Understanding control flow

From 10/26 and 10/29

Snippet of Code

- Using our knowledge of Python and the Graphics module's API, we knew what this program does

```
from graphics import *

def main():
    win = GraphWin("My Circle", 100, 100)
    c = Circle(Point(50,50), 10)
    c.draw(win)
    win.getMouse()

main()
```

Constructor

GraphWin object

Also known as an **instance of** the **GraphWin class**

Method called on GraphWin object

From 11/05

Benefits of Classes

- Package/group related data into one object
- Reusing code
  - E.g., Don't need to check if user put in valid time
- Provide interface, can change underlying implementation
  - e.g., Counter's increment -- could implement like in Caesar Ciphers instead

Problem with helper1 and helper2

- Better job with the comments, renaming than last exam
- Problem: flow of control

```
def helper1(word, letter):
    for i in range(len(word)):
        if word[i] == letter:
            return i
    return -1
```

Goes back to whatever called this function.

Returns *position* of *first* occurrence of the letter, -1 if not found.

Creating a Door Class

- Options to represent if door is closed
  - Boolean isClosed: True/False
  - Integer state: 0/1
  - String state: “closed”/”open”
  - Counter isClosed = Counter(0,1)

Creating a Door Class (Example soln)

- def __init__(self)
  - self.isClosed = True
- def __str__(self):
  - if self.isClosed:
    - return “Door is closed”
  - …
- def toggle(self):
  - if self.isClosed:
    - self.isClosed = False
  - else:
    - self.isClosed = True
- def isOpen(self):
  - return not self.isClosed
- Tester function
- def testDoor():
  - door = Door()
  - print door
  - door.toggle()
  - print door.isOpen()

This Week

- Tuesday: Lab
  - MyTunes implementation
- Wednesday
  - Recursion
- Friday
  - Searching
  - Broader Issue: One Laptop Per Child