Objectives

- Defining classes
- Using our classes
- __cmp__ method
- Helper methods
- Broader Issue: Environmental Monitoring

Review

```
from graphics import *

win = GraphWin("Picture")
win.setBackground("black")
```

```
from card import *

c = Card(7, "diamonds")
print c.getRank()
```

- Same programming as before
- Just defining our own classes

Review

- Where do we define the data that is needed to represent every object of a class?
  - How do we access that data?
- How do we create a new method?

Creating a Counter Class

- Has a fixed range
- Starts at some low value, increments by 1, loops back around to low value if gets beyond some maximum value
- Example application of the counter: Caesar cipher for letters 'a' to 'z'

What is the API for this object/ class?

Object o of type Counter

- What are the attributes of an object in the class?
- What data should be used to represent an object in the class?

Creating a Counter Class

- Data: Instance variables that represent
  - Start, Stop, Current Value
- Methods (API)
  - Counter(start, stop)
  - increment([incValue]) ← Defaults to 1
  - setValue(value)
  - getValue()
  - getLow()
  - getHigh()

Applying the Counter Class

- To the Caesar Cipher program
- Plug in the Counter object and call its methods as appropriate…

Applying the Counter Class

- To the Caesar Cipher program
- Compare implementations, with and without using the counter
- Any drawbacks from using Counter class?

caesar_with_counter.py

Mar 26, 2010 Sprenkle - CSCI111 7

__CMP__ METHOD

Mar 26, 2010 Sprenkle - CSCI111 8

__cmp__: Compare Objects of Same Type

- Header: `def __cmp__(self, other)`
  - `other` is another object of the *same type*
- Returns
  - Negative integer if self < other
  - 0 if self==other
  - Positive integer if self > other
- Similar to implementing `Comparable` interface in Java
- Can now use objects in comparison expressions
  - <,>,==, `sort`

How would you compare 2 Card objects?

Mar 26, 2010 Sprenkle - CSCI111 9

Comparing Objects of the Same Type

```
def __cmp__(self, other):
    """ Compares Card objects by their ranks """

    if self.rank < other.getRank():
        return -1
    elif self.rank > other.getRank():
        return 1
    else:
        return 0

# Could compare by black jack or rummy value
```

Mar 26, 2010 Sprenkle - CSCI111 card3.py 10

Frequency Object

```
def __cmp__(self, other):
   """Compares this object with another object.
      Used in a sort method."""
   if self.count == other.count:
           return cmp(self.key, other.key)
   return  cmp(self.count, other.count)
```

Mar 26, 2010 Sprenkle - CSCI111 11

HELPER METHODS

Mar 26, 2010 Sprenkle - CSCI111 12

Helper Methods

- Part of the class
- Not part of the API
- Make your code easier but others outside the class shouldn't use
- Convention: method name begins with "_"

Example Helper Methods

- Only *loosely* enforces that other can't use
  - Doesn't show up in `help`
  - Does show up in `dir`

Helper Method:

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

In use:

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

card4.py

Summary: Designing Classes

- What does the object/class represent?
- How to model/represent the class's *data*?
  - Instance variable
  - Data type
- What *functionality* should objects of the class have?
  - How will others want to use the class?
  - Put into methods for others to call (API)

Benefits of Classes

- Package/group related data into one object
  - `Deck` can have list of `Card` objects rather than a list of ranks and a list of suits
- Reusing code
  - E.g., Don't need to check if user put in valid key
- Provide interface, can change underlying implementation without affecting calling code

Considerations for Using Classes

Only use class if you're using most of its functionality/information

- Don't use `Counter` for validating if a number is within the valid range
  - Because not using the wrapping/current value

Since don't know implementation, may inadvertently duplicate code

- Redo something done by class
- Could have efficiency penalties
- **But** time saved reusing code is usually worth it

Changing Implementations

- Same API, different implementations

```
def __init__(self, rank, suit):
    self.rank = rank
    self.suit = suit

def getRank(self):
    return self.rank

def getSuit(self):
    return self.suit
```

```
def __init__(self, rank, suit):
    self.cardid=rank
    if suit == “clubs”:
        self.cardid += 13
    elif suit == “hearts”:
        self.cardid += 26
    elif suit == “diamonds”:
        self.cardid += 39

def getRank(self):
    return (self.cardid-2) % 13 + 2

def getSuit(self):
    suits = ["spades", "clubs", "hearts", "diamonds"]
    whichsuit = (self.cardid-2)/13
    return suits[whichsuit]
```

Tradeoff: Saving information (memory); Computing information

card_byid.py

Two Counter Implementations

- Compare counter.py and counter2.py's increment implementations

Extra Credit Functionality Ideas

- Return the card's color (Red/Black), using a constant defined at the top for each color
  - What game is this useful for?
- Boolean methods: isBlack(), isRed()
- Boolean method: isOppositeColor(card)
- Boolean method: isSameSuit(card)
- Create a Hand class (very similar to Deck class)
  - Methods that check if all same suit, all same rank
- Player class for various games …
- Test/Demonstrate your methods

Due Tuesday before lab

BROADER ISSUE

Broader Issues: Environmental Monitoring

- Interdisciplinary projects involving sensor networks
  - Important new-ish CS research area
- Disclaimer:
  - Not a seismologist or a biologist
- Groups

Zebra:	Zebra:	Zebra:	Volcano:	Volcano:
Sirocco	George	Will	Andrew	Ben
Harrison	Kelly Mae	Logan	Nick	Dave
James	Dalena	Collier	Shannon	Luke
Amy	Phil	Jeni	Hank	Taylor

Discussion

- What are the CS challenges to the projects?
  - Any challenges only applicable to one project?
- How does the environment impact the CS research problems/solutions?
- How did the researchers address these challenges?
  - How would ***you*** address the challenges?

Overview of Challenges: Efficiency

- Some programmers thought that efficiency didn't matter anymore
  - GB of memory, terabytes of storage on machines
- Now: small and embedded devices
  - Need to be efficient!
- Energy in battery powered nodes
- Amount of data stored (when to delete?)
- When, amount of data transferred

Overview of Challenges: Reliability

- Data delivery
  - Missing data
  - Connectivity (good signal?)
  - Duplicate data (different sources?)
  - Dead sensor nodes
  - Calibration of data (time synchronization)
- Nodes
  - Withstand extreme weather, conditions
  - Battery life
- Robustness: recover from software failure/ malfunction or bad data

Overview of Challenges

- Testing
  - Accurately simulate conditions (which will vary widely over long periods of time)
- Different goals from domain scientists
  - CS: push boundaries of sensor networks
    - Example: Improve reliability of data to 95%
    - Seismologists: need 100% reliable data