Objectives

- Defining our own classes
- Using our defined classes

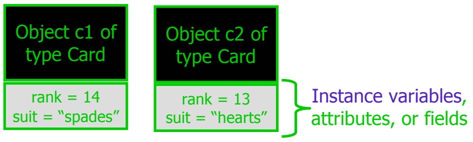
Classes and Objects

- We're all of type *homo sapien*
- Each of us has these attributes:
  - Height
  - Weight
  - Hair color
  - Hair type
  - Skin color

We all have these attributes, different values for the attributes

Classes and Objects

- c1 = Card(14, "spades")
- c2 = Card(13, "hearts")



Creating a Deck Class (Partial)

- List of Card objects

```
from card import *

class Deck:
    def __init__(self):
        self.cards = [ ]
        for suit in ["clubs","hearts","diamonds","spades"]:
            for rank in range(2,15):
                self.cards.append(Card(rank, suit))

    def __str__(self):
        result = ""
        for card in self.cards:
            result += str(card) + "\n"
        return result
```

Constructor initializes instance variable: self.cards

Creates and returns a string

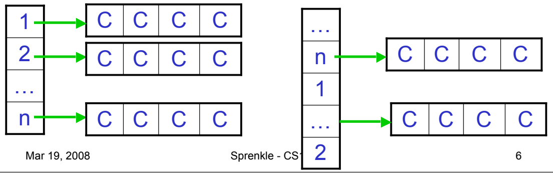
Displays cards on separate lines

Deck API

- What methods should our Deck class provide?
  - shuffle()
    - Shuffles the cards
  - draw()
    - Draws one card from the Deck object
  - deal(numplayers, numcards)
    - Deals numcards to each of the numplayers players

Deal Discussion

- Return proposals, given that a hand is a list of cards
  - Return a *dictionary* of hands
  - →Preferred: Return a *list* of hands
    - Dictionaries take up a lot of space, much more than a list that's as long as the number of players



Adding Deck Functionality

- Functionality:
  - shuffle()
  - draw()
  - numCardsRemaining()
- What do the method headers look like?
- What should they return?
- How do we implement them?

Deck API

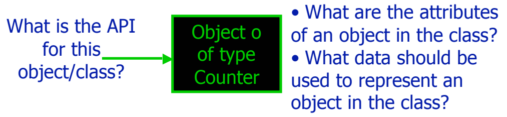
- Deck() ← Constructor
- shuffle()
- draw()
- numRemaining()
- __str__()

Extra Credit Opportunity

- Write additional code for Deck and Card classes
  - Leading to a game…
- Possible functionality
  - Dealing a hand to a player
  - Alternative comparisons of cards for your game
- Adding a Player class for a particular game
- Due next Tuesday before lab

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 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

- Instance variables
  - High, Low, Current Value
- API (methods)
  - Counter(low, high)
  - increment()
  - setValue(value)
  - getValue()
  - getLow()
  - getHigh()

counter.py

Creating a Counter Class

- Data: Instance variables that represent
  - High, Low, Current Value
- Methods (API)
  - Counter(low, high)
  - increment([amount]) ← Defaults to 1,
  - decrement([amount]) ← -1
  - setValue(value)
  - getValue()
  - getLow()
  - getHigh()

counter.py

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?

caesar2.py

Exam 2

- Mean: 85%, Median: 88%
- Overall: better performance than first exam
- Change in performance from first exam
- Last 3 pages: best indicators of understanding
- Effect on final course grade
  - Combined exams worth 30%
  - Weighted toward higher grade (e.g., 16/14)

Exam 2 Problem

- Unexpected trick question
  - list2=[`‘tutu’, ‘same’, ’foot’, ‘yellow’`]
  - list1=[`7, -2, 3, 4`]
  - list2.remove(3)
    - Error because 3 isn’t in the list (no change to list)
- Everyone gets that one right
  - Other questions based on what you said for that list

Data Types of Loop Variables

For Loop Header	Data Type of x
for x **in** <file>:	string (line in a file)
for x **in** xrange(...):	int
for x **in** <string>:	string (character)
for x **in** <list>:	Depends on elements in list

Averaging Grades

- Similar to averaging temperatures problem in class and reading case values from file in lab

```
gradeFile = file(“grades.txt”, “r”)
total = 0
numGrades = 0
for grade in gradeFile:
    grade = grade.strip()
    total += grade
    numGrades += 1
gradeFile.close()
print “The average grade is”, \
    float(total)/numGrades
```

`grade` needs to be converted to a *number* to add to `total`

Tracing Through Functions

- Only *one thing* gets returned from function
- Trace through call to

  helper1("e", "general")

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

Tracing Through Functions

- Only *one thing* gets returned from function
- helper1("e", "general")

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

Name: **firstPos**
Comment: returns the position of the first occurrence of the given letter in the word; returns -1 if the letter does not occur in the word

Writing Code

- Write a function named `getBrand` that, given as a parameter a CandyBar object, returns a string representing the brand name for the candy bar

Writing Code

- Write a function named `getBrand` that, given as a parameter a CandyBar object, returns a string representing the brand name for the candy bar

```
def getBrand( candybar ):
        if candybar.hasPeanuts():
                return "Snickers"
        if candybar.hasCaramel():
                return "Milky Way"
        return "Three Muskateers"
```

Broader Issue

- Facebook knows what you did last summer