Objectives

- __cmp__ method
- Helper methods
- Command-line arguments
- Group Work: Designing Classes

Comparing Objects of the Same Type

- Special `__cmp__` method
  - 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 Cards?

Comparing Objects of the Same Type

- Example Code:

```
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
```

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 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 == 14:
      return 15
   else:
      return 5
```

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
  - 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 and decrement implementations

COMMAND-LINE ARGUMENTS

Command-line Arguments

- We can run programs from terminal (i.e., the “command-line”) and from IDLE
- Can pass in arguments from the command-line, similar to how we use Unix commands
  - Ex: `cp <source> <dest>`

    $\underbrace{\texttt{<source> <dest>}}_{\text{Command-line arguments}}$
  - Ex: `python maptest.py 3`

    $\overbrace{\texttt{maptest.py 3}}$
- Makes input easier
  - don’t have to retype each time executed

Command-line Arguments

- Using the **sys** module
  - What else did we use from the **sys** module?

$\texttt{python } \underbrace{\texttt{maptest.py 3}}$

$\texttt{python } \underbrace{\texttt{command\_line\_args.py <filename>}}_{\textbf{List}\text{ of arguments, named }\texttt{sys.argv}}$

- How to reference (get value) “<filename>”?

Command-line Arguments

- Using the **sys** module

```
python command_line_args.py <filename>
```

`sys.argv` ⟶

command_line_args.py	<filename>
0	1

- How to reference (get value) “<filename>”?
  - `sys.argv` is a ***list*** of the arguments
  - `sys.argv[1]` is the filename
  - `sys.argv[0]` is the name of the program

Using Command-line Arguments

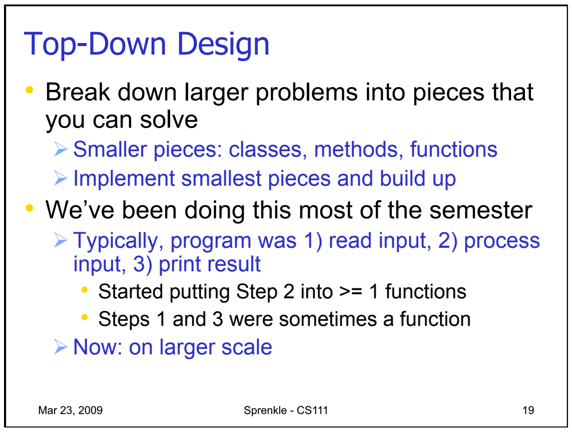
- In general in Python:
  - `sys.argv[0]` is the Python program’s name
- Have to run program from terminal (not from IDLE)
  - Can edit program in IDLE though

➔Useful trick:
  - If can’t figure out bug in IDLE, try running from command-line
    - May get different error message

DESIGNING CLASSES

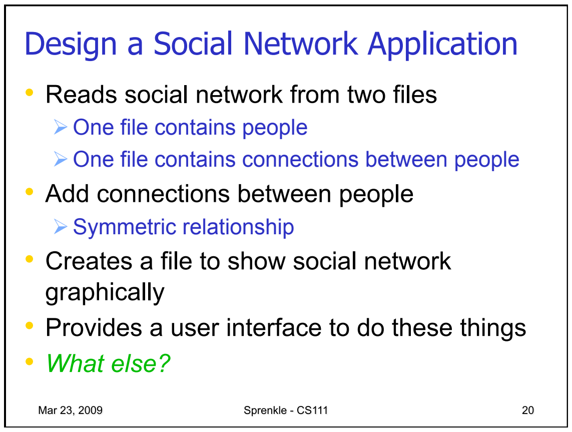
Summary: Designing Classes

- 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)
- General Class Design:
  - **nouns** in a problem are classes/objects
  - **verbs** are methods



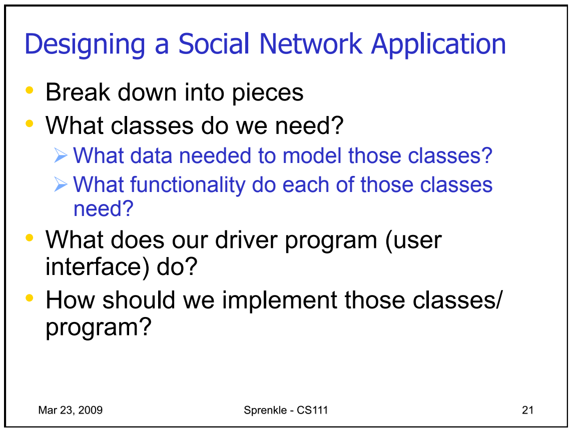
Top-Down Design

- Break down larger problems into pieces that you can solve
  - Smaller pieces: classes, methods, functions
  - Implement smallest pieces and build up
- We've been doing this most of the semester
  - Typically, program was 1) read input, 2) process input, 3) print result
    - Started putting Step 2 into >= 1 functions
    - Steps 1 and 3 were sometimes a function
  - Now: on larger scale



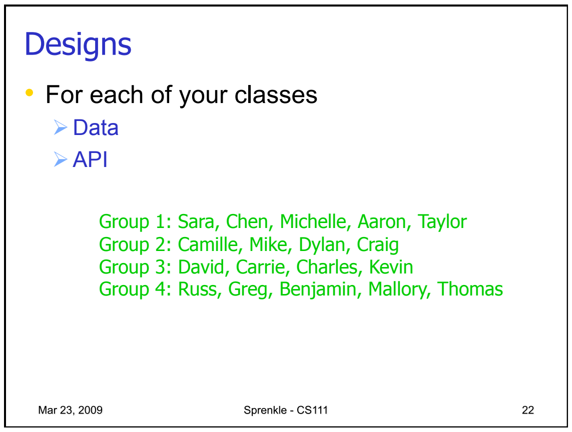
Design a Social Network Application

- Reads social network from two files
  - One file contains people
  - One file contains connections between people
- Add connections between people
  - Symmetric relationship
- Creates a file to show social network graphically
- Provides a user interface to do these things
- *What else?*



Designing a Social Network Application

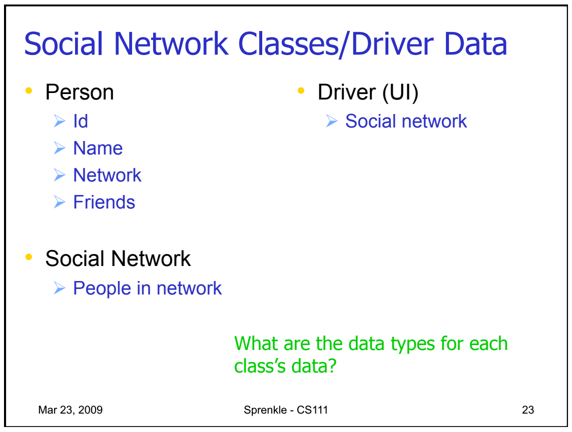
- Break down into pieces
- What classes do we need?
  - What data needed to model those classes?
  - What functionality do each of those classes need?
- What does our driver program (user interface) do?
- How should we implement those classes/ program?



Designs

- For each of your classes
  - Data
  - API

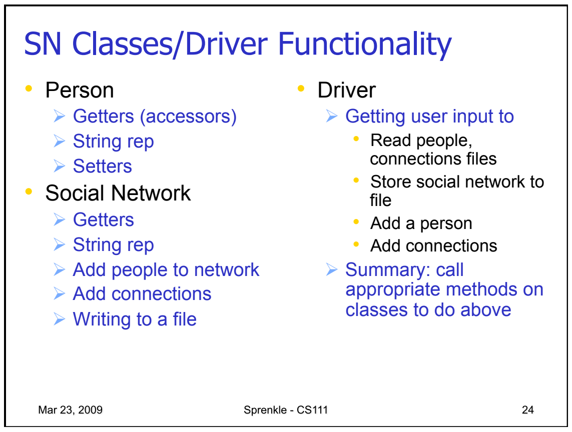
Group 1: Sara, Chen, Michelle, Aaron, Taylor
Group 2: Camille, Mike, Dylan, Craig
Group 3: David, Carrie, Charles, Kevin
Group 4: Russ, Greg, Benjamin, Mallory, Thomas



Social Network Classes/Driver Data

- Person
  - Id
  - Name
  - Network
  - Friends
- Social Network
  - People in network
- Driver (UI)
  - Social network

What are the data types for each class's data?

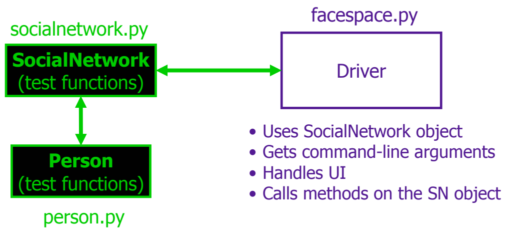


SN Classes/Driver Functionality

- Person
  - Getters (accessors)
  - String rep
  - Setters
- Social Network
  - Getters
  - String rep
  - Add people to network
  - Add connections
  - Writing to a file
- Driver
  - Getting user input to
    - Read people, connections files
    - Store social network to file
    - Add a person
    - Add connections
  - Summary: call appropriate methods on classes to do above

Lab 10 Design

- 3 files: person.py, socialnetwork.py, facespace.py



- Uses SocialNetwork object
- Gets command-line arguments
- Handles UI
- Calls methods on the SN object

Problem: People Files

- Given an people file that has the format
  - <num_users>
  - <user_id>
  - <name>
  - <network>
  - …
  - <user_id_n>
  - <name_n>
  - <network_n>
- Write algorithm to create Person objects to represent each person, add to SocialNetwork object

Problem: Connection Files

- Given a connection file that has the format
  - <user_id> <user_id>
  - <user_id> <user_id>
  - …
  - <user_id> <user_id>
- Each line represents a friend/connection
  - Symmetric relationship
  - Each is a friend of the other
- Update SocialNetwork object

UI Specification

- Checks if user entered command-line argument
  - Default files otherwise
- Read people, connections from files
- Repeatedly gets selected options from the user, until quits
- Repeatedly prompts for new selection if invalid option
- Executes the appropriate code for the selection
- Stops when user quits
- Stores the social network into the file

Write pseudocode

UI Pseudocode

```
Use default files if only one command-line argument
Read people, connections from files
while True:
    display menu options
    prompt for selection
    while invalid option
        print error message
        prompt for selection
    break if selected quit
    otherwise, do selected option
Store library to designated file
```

Implementation Plan

- Implement `Person` class
  - Test (write test functions, e.g., `testPerson()`)
- Implement `SocialNetwork` class
  - Example runs in lab write up
  - Note: in general, methods for classes will not prompt for input (Use input parameters)
  - Test
- Implement driver program

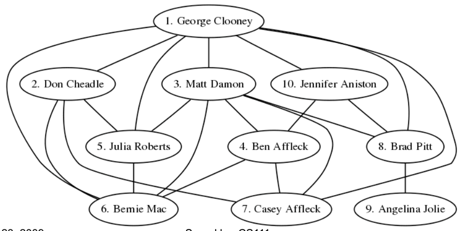
Plan for Implementing a Class

- Write the constructor and string representation/print methods first
- Write function to test them
- While more methods to implement …
  - Write method
  - Test

- See counter.py and card.py for example test functions

Writing Data To File

- I will provide method that prints your social network to a file in a particular format (dot)
- Can display network graphically using `dot` program, e.g.,



Broader Issue

- One of Social Network articles
  - News feed
  - Privacy/security