## Objectives

- Wrap up defining classes
- Command-line arguments
- Group Work: Designing Classes

## Review

- How do we define a class?
- What do methods look like in Python?
- What parameter does every method take?
- What is the difference between calling methods and functions?

## Review

- What does the `__init__` method do?
  - How should you implement it?
  - When does it get called?
- What does the `__str__` method do?
  - How should you implement it?
  - When does it get called?

## __CMP__ METHOD

## `__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?

## 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
```

`card3.py`

## 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)
```

## HELPER METHODS

## 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

## 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 $\underbrace{\texttt{<source> <dest>}}_{\text{Command-line arguments}}$
  - Ex: python $\overbrace{\texttt{command\_line\_args.py file.txt}}$
- Motivation: 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?

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 $\longrightarrow$

| "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

command_line_args.py

## 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 still edit program in IDLE

➔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

## Requirements for 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 as a graph
- 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

Jean Paul, Lida, Yates, Colin

Nick, Anh, Minh, Callie

Will, Meng, Ola

## 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?

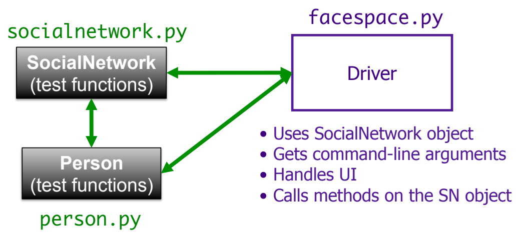
## SN Classes/Driver Functionality

- 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

How should we test these?

## Lab 10 Design

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


socialnetwork.py
SocialNetwork (test functions)
facespace.py
Driver
Person (test functions)
person.py
• Uses SocialNetwork object
• Gets command-line arguments
• Handles UI
• Calls methods on the SN object

## Problem: People Files

- Given a 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 user 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 social network to designated file
```

## Implementation Plan

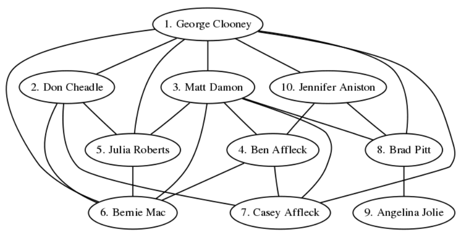
1. Implement `Person` class
   - Test (write test functions, e.g., `testPerson()`)
2. Implement `SocialNetwork` class
   - Example runs in lab write up
   - Note: Methods for classes will **not** prompt for input; Use input parameters
   - Test
3. Implement driver program

## Plan for Implementing a Class

- Write the constructor and string representation/print methods first
- Write function to test them
  - See `card.py` for example test functions
- While more methods to implement …
  - Write method
  - Test
  - REMINDER: methods should **not** be using `input` function but getting the input as parameters to the method

## Goal Output

- You will create graphs that look something like this and put them on a new web page for Lab 10



## This Week

- Lab 10
- Broader Issue: An article about social networking
  - News feed
  - Privacy/security