Objectives

- Command-line arguments
- Group Work: Designing Classes

Review

- What method do we implement to compare two objects of the same type?
- What does the method header look like?
- What does it return?
- How can we use it?

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>`

    Command-line arguments
  - Ex: `python command_line_args.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?

`python command_line_args.py <filename>`

List of arguments, named **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 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

Amy
Andrew
Logan
Hank
Jeni

Dalena
Phil
Collier
Kelly Mae
Dave

Nick
George
Will
James
Ben

Sirocco
Taylor
Harrison
Shannon
Luke

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?

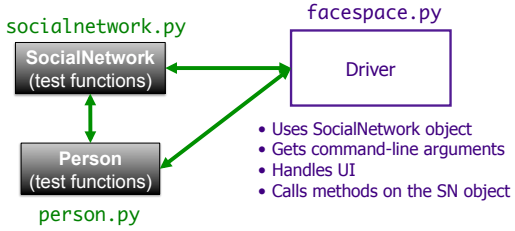
SN Classes/Driver Functionality

- 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`



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

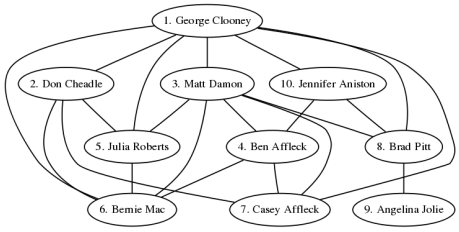
1. Implement `Person` class
   - Test (write test functions, e.g., `testPerson()`)
2. Implement `SocialNetwork` class
   - Example runs in lab write up
   - Note: Methods for classes will **not** prompt for input; Use input parameters
   - Test
3. Implement driver program

Plan for Implementing a Class

- Write the constructor and string representation/print methods first
- Write function to test them
  - See counter.py and 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 two graphs that look something like this and put them on a new web page for Lab 10



This Week

- Lab 10
- Broader Issue: One of Social Network articles
  - News feed
  - Privacy/security