# Objectives

- Group Work: Designing a Social Network
- Prep for Lab 10

# Review

- Why classes and objects?
- How do we create new data types?

# Review: Card and Deck classes

# 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
- Provides a user interface to access/update a social network

---

# 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?

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

4

# Designs

- For each of your classes
  - Data
  - API

---

# Social Network Classes/Driver Data

- Person
  - Id
  - Name
  - 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
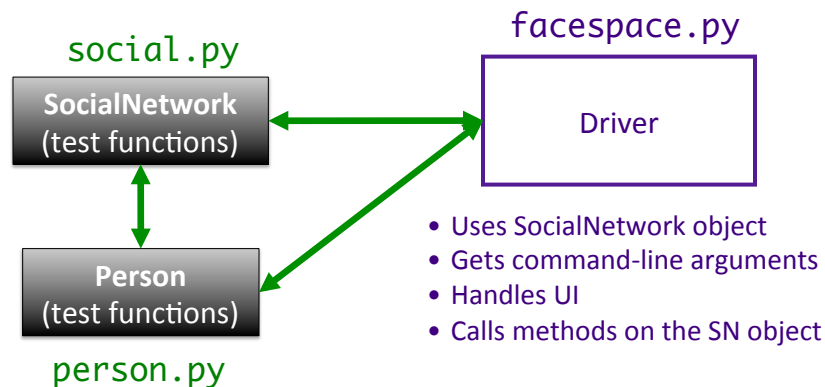
How should we test these?

---

## Lab 10 Design

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

`social.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>
  …
  <user_id_n>
  <name_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 arguments
  - ➢ 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 and deck.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

# Exam 2

| | Section | | | |
|---|---|---|---|---|
| | A | B | C | Total |
| **Average** | 88.64 | 89.45 | 80.21 | 89.63 |
| **Median** | 89.77 | 93.75 | 81.25 | 89.50 |
| **Std Dev** | 3.15 | 4.43 | 4.02 | 10.8 |

# This Week

- Lab 10
  - Use a class that was already defined
  - Define your own classes
- Broader Issue: net neutrality