

Lab 10

- Social Network

Lab 10: Pair Programming

Lizzie	Rachel
Ian	Anna
Lindsey	Chase
Chas	Alison
Ryan	Robert
Joseph	Margaret
Olivia	Parker
Jordan	Kalady
Davis	Mary-Frances
Findley	Andrew

Review Lab 9

- How can you get all the values from a dictionary?
 - How can you turn it into a list?

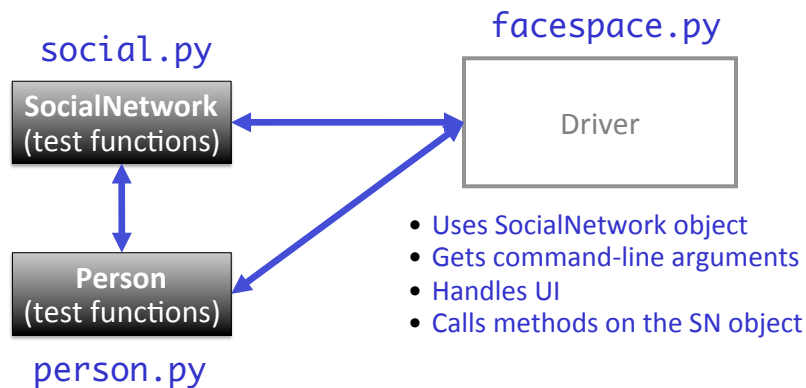
More work with dictionaries in lab10.
Make sure you understand how to use dictionaries.

Review

- Why do we create classes?
- How do we create a class?
 - What are important methods to implement?
 - How do we implement them?

Lab 10 Social Network Design

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



March 27, 2018

Sprenkle - CSC111

5

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?

March 27, 2018

Sprenkle - CSC111

6

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?

Towards a Solution and Hints

- Given “stubs” for each of the class files
- `social.py` is the most filled out
 - Has the methods and docstrings defined
 - **BUT** still refer to the description in the lab on the web for all information
- For whatever variable you’re dealing with, think about its data type and API
 - SocialNetwork API handout
 - Add your Person class’s API to the handout

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: People Files

- Algorithm:

- Open file
- Read the first line in the file
 - that represents the number of users in the file
- Repeat <number of users> times
 - Read the line → that's the userid/username
 - Read the line → that's the name of the user
 - Create a Person object
 - Update the Person's name
 - Add the Person object to the dictionary
- Close the file

```
<num_users>
<user_id>
<name>
...
<user_id_n>
<name_n>
```

`fileobj.readline()` always reads in the *next* line of the file

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

Algorithm: Connection Files

- Given a connection file that has the format

```
<user_id> <user_id>  
<user_id> <user_id>  
...  
<user_id> <user_id>
```

- For each line in the file
 - Split the line into the two ids
 - Look up the two Persons by their ids
 - Make the two Persons friends

UI Specification

- Checks if user entered command-line argument
 - Default files otherwise (see later slides)
- 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

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
```

Why not a GUI?

Implementation Plan

1. Implement **Person** class
 - Test (write test function, 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

Export SocialNetwork to Files

- I provide method to write connections to a file
 - Because only want connection once
- You handle writing to people file
 - Must be in **same format** that you read in
 - Just “undoing” the read
- Good test: if you read in a people file, export it to another file → original and exported file should look similar
 - If you read in that exported file, should see same social network
 - Files themselves may not be exactly the same because of order printed out

Test Data

- SocialNetwork requires: People file, Connections file
- Social Networks:
 - Simple
 - Hollywood
 - Randomly generated files
 - From W&L first and last names, randomly combined, connected
- Can combine multiple files (with unique usernames) to create larger social networks

COMMAND-LINE ARGUMENTS

March 27, 2018

Sprenkle - CSCI111

19

Command-line Arguments

- We can run programs from terminal (i.e., the “command-line”) and from IDLE
- From the command-line, can pass in arguments, similar to how we use Unix commands

➤ Ex: `cp`  `<source> <dest>`
Command-line arguments

➤ Ex: `python3`  `myprog.py 3`

- Makes input easier

➤ Don't have to retype each time executed

March 27, 2018


Sprenkle - CSCI111

20


Command-line Arguments

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


```
python3 myprogram.py 3
```



```
python3 command_line_args.py <filename>
```



List of arguments, named **sys.argv**



- How can we access “<filename>”?
 - Then we can use in our program

March 27, 2018

Sprenkle - CSC111

21

Command-line Arguments

- Using the **sys** module
- ```
python3 command_line_args.py <filename>
```

**sys.argv** → 

|                      |            |
|----------------------|------------|
| command_line_args.py | <filename> |
| 0                    | 1          |

- How can we access “<filename>”?
  - **sys.argv** is a *list* of the arguments
  - **sys.argv[0]** is the name of the program
  - **sys.argv[1]** is the filename

March 27, 2018

Sprenkle - CSC111 `command_line_args.py`

22

## 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