Objectives

- Design Discussion
- Unix, Your environment

1

Review && Submission

Review

- What is the difference between comments and doc strings?
- What is the difference between functions and methods?
- What is the difference between instance, class, and local variables?
  - What questions should you ask to determine which you should use?
- What is the difference between *inheriting* and *importing*?

Submission

- In parallel, show Professor Sprenkle your documents for
  - Terminology table
  - Unix commands
  - Know your machine

Purpose of questions?

2

Answers, in Brief

- All doc strings are comments; not all comments are doc strings
  - Doc strings are specifically to describe interface for functions/methods and for classes
- Methods: specific to objects of a certain class
- Instance: one for each object of class
  - Class: one for all objects of class
  - Local: short-lived variable for a specific piece of code
- Inherits: get all properties/methods of parent
  - import: just *uses* code from other

Design Questions

1. `turn` is an *instance* variable of the `Game` class.
   - Is it better design for `turn` to be a local, instance, or class variable? Justify your answer.
2. `user_input` is a *local* variable in the `getInput` method of the `ConnectFour` class.
   - Is it better design for `user_input` to be a local, instance, or class variable? Justify your answer.
3. `RANKS` is a *class* variable of the `Card` class.
   - Is it better design for `RANKS` to be a local, instance, or class variable? Justify your answer.
4. `tokens` is an *instance* variable of the `ConnectFour` class.
   - Is it better design for `tokens` to be a local, instance, or class variable? Justify your answer.
5. `Player` is a class in `war.py`.
   - Is it better design for the `Player` class to be defined in `war.py` or in `game.py`? Justify your answer.
6. `War`'s `step` method takes as a parameter `dummyInput`. What purpose does it serve?

Design Answers, in Brief

1. `turn` should be an *instance* variable of the `Game` class.
   - Each object of the Game class should have its own turn variable.
2. `user_input` should be a *local* variable in the `getInput` method of the `ConnectFour` class.
   - It is not useful to any other method of the class; it is just for that method.
3. RANKS should be a *class* variable of the `Card` class.
   - There should only be one RANKS object for *all* Card objects.
4. `tokens` should be a *class* variable of the `ConnectFour` class.
   - There only needs to be one copy of `tokens` for *all* ConnectFour objects
5. `Player` could be in game.py as an abstract parent *or* as a class in `war.py`.
   - Either answer can be justified.

5

Design Answers, in Brief

`War`'s `step` method takes as a parameter `dummyInput`. What purpose does it serve?

From Game class:

```
def main(self):
   while not self.isGameOver():
      print(self)

      self.step(self.getInput())

   print("\nGame Over!\n")
   print(self)
```

`pass` in War

Conclusion:
- Use of dummyInput is an indication that something should be designed better

6

Unix Review

- What is a synonym for *directory*?
- What is a *path*?
- How do you find out the path of the current directory?
- How do you go into another directory? Give an example.
- How do you view the contents of the directory?
- How do you create a directory? Give an example
- How do you copy a file? Give an example
- How do you rename a file? Give an example
- How do you delete a file? Give an example.

From Lab: Know Your Computer/Programming Environment

- What is the path to your home directory?
- Where are you going to put your files for this class?
- How do you open a terminal?
- Which version of Python are you running?
  - How do you determine that?

What Was That About?

- Comfort with your machine
  - Transition from intro to intermediate
- Other tools are based on Unix
  - If you know Unix, then it makes other things easier
- Instructions for installing software often make use of the command line

Text Editors

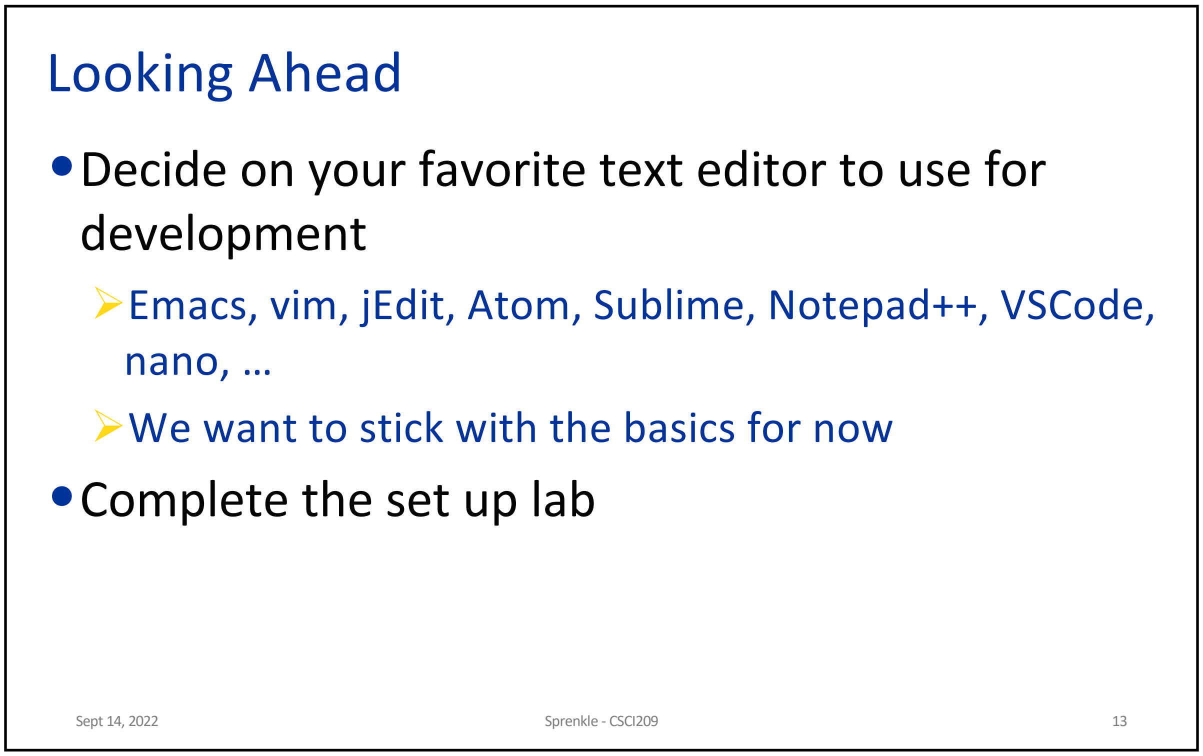
- For editing (plain) text!
  - Only text/characters
    - Example: no font, size changes
  - As opposed to *rich* text
- Examples?
  - Basic: Notepad
  - No frills, all terminal: nano
  - For the nerdier: emacs, vi/vim
  - GUI, in increasing order of fancy: jEdit, gedit, Sublime, Atom

Text Editors: What was that about?

- We'll use text editors to start in a couple of ways (git and programming)
- Want to stick with the basics/fundamental for now
  - Build on them!

Extra Credit Opportunity

- Moderated panel discussion about "Zero Trust"
  - how agencies, organizations, and businesses can implement it to protect themselves against cyber threats.
- Thursday, September 15, 7:30 – 9:00p.m.
- Location: G-14 Science Addition
- Post response on Canvas for up to 10 points EC
  - See discussion forum for more info
- Up to 3 of these in the semester



Looking Ahead

- Decide on your favorite text editor to use for development
  - Emacs, vim, jEdit, Atom, Sublime, Notepad++, VSCode, nano, …
  - We want to stick with the basics for now
- Complete the set up lab

13