Lab 4

For Beginners

Project 1

Download the file vabirds.csv from Resources/Data.

Use Pandas to read the file and capture the header information. Convert the years into floats. Find the observations for DownyWoodpecker and plot the information with years as the independent variable.

Project 2

Write a class Bird that contains the species name as a string and a numpy array for the observations. Write a constructor that loads the values into an instance. Create an empty list. Read the first line of the vabirds.csv file. Use the header to create a list of years as in Project 1. Convert the years list to a numpy array. Continue reading the file, creating a Bird instance at each iteration and appending it to the list. Plot the observations for BlueJay.

Project 3

Add a method `stats` to your Bird class that computes the maximum, minimum, mean, and median numbers of birds observed. For the maximum and minimum also obtain the year. (Hint: look up argmax and argmin for numpy). You may wish to add an attribute `years` as well. Print the stats for ChippingSparrow and plot the observations.

For Intermediates

Project 4

In the early 2000's an "urban legend" circulated that one could read text in which all letters except the first and last were scrambled. For example:

Aoccdrnig to rscheearch at an Elingsh uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer is at the rghit pclae.

Write a program to scramble a passage input from a file. Print the result to a file with the same name as the original but a suffix `_scrambled` added (so if the original was Example.txt it will be `Example_scrambled.txt`. Look at the scrambled file first—can you read it?

First write a scramble_word function, then a scramble_line function, and finally a function that writes the lines to the new file. Your main() routine will request from the user the name of the input file, then call the printing function, which will call the scramble_line function that will in turn call the scramble_word function. This is an example of how we divide up work into separate "chunks" or "concerns."

Internal punctuation (apostrophes) can be scrambled, but leave any punctuation such as periods, question marks, etc., at the end of a word in place.

Hint: since you cannot overwrite strings you will need to convert them to a list and back again.

Use Resources/Data/Example.txt as your sample input.

FYI this is an "urban legend" because: firstly, no such research was ever conducted at any university, and secondly it is true only for very practiced readers of English and even then only for familiar words that are easy to recognize in context.

Project 5.

Write a program that reads a file with the following format: each line consists of an actor's name followed by a semicolon, followed by a (partial) list of movies in which the actor has appeared. Movie titles are separated by commas. You can handle this file by reading each line, then splitting on the semicolon so that you carve off the performer's name. Append that to an `actors` list. The rest of the line is a comma-separated string. Don't forget to strip the end-of-line marker. Take the movies string and split on commas to create a list. Append (not extend) this list to a `movies` list. This gives you a two-dimensional list (each element is itself a list).

Use your two lists to print the information in a nicer format. Each line should be printed as

<Actor> has appeared in the following movies: <movies>

You should use your two lists to construct the above string. Use `join` to rejoin the movies list into a string. Use either a format string or concatenation to create the message string.

Read the name of the input file from the command line. Use Resources/Data/movies.txt as your example input file.

Project 6

Modify your code from Project 4 to define a class Actor whose attributes are

1. name
2. filmography

Write a constructor that stores these attributes as members of the instance. The filmography will just be the movie list for this project.

Write a printme method that uses code from Project 1 to print an instance in the format specified in the earlier project. That is, it will use self.name and self.filmography in the formatting.

Keep your Actor class definition in its own file.

Modify your code from Project 1 so that instead of storing actor and movielist separately, you will create instances. Specifically, your actors list will now be a list of instances of your Actor class. As you read the file you will create a new instance using a line like

```
actors.append(Actor(something,something))
```

After creating your list of instances, use your printme method to reproduce the output in the same form as in Project 1.

Project 7

In addition to a constructor and a printme method, your Actor class should contain a method to return the actor name (a "getter") and another to return the filmography as a string. You can use these in your printme method.

Write a separate program with a main() that reads the movies.txt file and constructs the list of Actor instances. Use the list of Actor instances to create a dictionary in which the movie titles are the keys and the value corresponding to each key is a set of the cast member names. You will need to process through your actors list, checking whether each movie title is already in the dictionary. If it is not, first create an empty set, then immediately update the set with the actor name (use the "getter"). If the movie title is already a key, update the set of the castlist.

Write code to request from the user a movie title. Use that movie title to print the castlist of the movie. You can convert a set to a list with list(movie_dict[key]) and then use join to print the castlist neatly.

Be sure to use appropriate functions rather than monolithic code.

For Experts

Project 8

We are going to write the world's stupidest and most boring first-person shooter game.

Design a class Zombie and a class Fighter. Both should inherit from a base class called Character.

Character should have attributes "health" and "ammo", both of which can be initialized to some arbitrary number below a maximum. Character also should have a method hit_enemy, which will use the random module to return damage. I treated damage as a fraction of the instance's current health, so I computed damage by returning a random number between 0 and 1. To do this I used the Python standard random module, and invoke the random.random() built-in.

The amount of ammo expended should be proportional to the damage. If the Character has no ammo it cannot do any damage.

Character should have another method take_hit, which will have as an argument damage (obtained from another Character instance). It will subtract damage from the character's health. I chose to subtract damage*instance.health but there could be other ways to do it.

That's about all that Character does.

Zombie will have an additional attribute of number, which will be assigned after the Character constructor is invoked. Hint: you can do that with Character.__init__(self)

The Zombie should also have a lower maximum health value than the Fighter, say 1 versus 10.

Zombie will have an additional method of die, which will occur after a "hit" when its "health" is 0.25 or less. It returns True if the zombie instances dies and False if it does not.

Fighter will have an additional attribute of name, score, and several more methods. The "Fighter" won't die in this game but if his health level goes below 0.01 the game will end.

He will have a method to "move". When invoked it will return a Zombie with 50% probability. The Fighter should shoot a random Zombie. His chance of killing the

Zombie should be 75%. If the Zombie is not killed, it will shoot at the Fighter. If the Fighter kills a Zombie his score should be incremented by 1.

The Fighter should also have a restore method, which will assign a random (up to max_health) increment of health. There should also be a reload method to add ammunition. In my game, I implemented both by subtracting the instance's health or ammo from its maximum health or ammo, computing a fraction of the gap from 0 to 1 with random.random(), and adding that amount back in.

The Fighter should have a "flee" method which will "kill" all the Zombies, but without incrementing the Fighter's score.

Finally, there should be a "status" method to return the Fighter's health and ammo levels.

The main program will be the gameplay. This will be an entirely text-based game. First ask the user for the Fighter's name. Then enter an infinite loop in which the user is presented with the possible commands (move, shoot, flee, reload, restore, status, quit). Break out of the loop on the quit command and print the Fighter's final score.

I've provided a basic main program. You can write the classes to fit it. If you are so inclined you may change it if you'd rather set up your classes differently.

You can put all three classes into one characters.py module.

If you have time you can make the game more interesting. Right now there is really no way for the player to lose without trying to do so. (You should attempt to do so in order to test your code.) One could try to figure out a way for the zombies to attack before the player has a chance to shoot, or to shoot multiple times, or any other implementation you can imagine.