CPSC 217 Assignment 4

Due: Friday December 7, 2018 at 12:00noon
Weight: 7%
Sample Solution Length: Approximately 130 lines (No comments and no code for the A+ part)

Individual Work:

All assignments in this course are to be completed individually. Students are advised to read the guidelines for avoiding plagiarism located on the course website. Students are also advised that electronic tools may be used to detect plagiarism.

Late Penalty:

Late assignments will not be accepted.

Submission Instructions:

Submit your .py file electronically to the Assignment 4 drop box in D2L. You don't need to submit the data files – we already have them.

Description

In this assignment you will analyze food webs. In order to do this you will need to load a description of predator-prey relationships from a file, store them in a data structure, and identify the relationships between the organisms.

Predator-Prey File Format

The predator prey information is stored in CSV (comma separated value) files. Each line in the file will be of the form `<Predator>,<Prey>,<Prey>,...,<Prey>` where `<Predator>` and `<Prey>` are placeholders for specific animals. Each line will always begin with exactly one predator, followed by 1 or more prey. For example, lines in the file could be `Whelk,Limpets,Mussels` or `Lion,Zebra`. Note the predator and prey names may include a mixture of upper and lowercase letters, and may also include spaces. In order to make the files easier to work with you can assume that there will not be any spaces immediately before or after any of the commas.

Part 1: What do the Predators Eat?

Your first task is to list everything that each predator eats on a single line with nice formatting. For example, if your file contains the line:

```
Lion,Gazelle,Wildebeest,Zebra
```

then you should output a single line for Lion that reads

```
Lion eats Gazelle, Wildebeest and Zebra
```

Notice that commas appear after all items except the last and second last items, and that the word "and" appears between the last and second last items. **You will be graded on correctly following this layout**, but I have provided a module that includes a function that will do the formatting for you – all you need to do is import the function and call it.

In order to complete this and subsequent tasks you must load all of the data from the food web file into a dictionary that describes the eats relationship. The keys in the dictionary will by the names of the predators. The values in the dictionary will be lists, where each element in the list is the name of a prey animal that the predator eats.

The output for Part 1 for AquaticFoodWeb.txt should be:

```
Predators and Prey:
  Bird eats Crab, Limpets, Mussels, Prawn and Whelk
  Crab eats Limpets and Mussels
  Fish eats Prawn
  Limpets eats Seaweed
  Lobster eats Crab, Limpets, Mussels and Whelk
  Mussels eats Phytoplankton and Zooplankton
  Prawn eats Zooplankton
  Whelk eats Limpets and Mussels
  Zooplankton eats Phytoplankton
```

Note: I have displayed my output in sorted order by using Python's sorted function. However, the orders of the predators and prey aren't important. You'll receive full credit as long as all of the predators and their prey are identified successfully.

Part 2: Identify the Apex Predators

We will define an *apex predator* to be any species in the food web that is not eaten by another organism. After displaying the predators and their prey your program should continue by displaying all of the apex predators with an appropriate heading.

Hint: Apex predators are those animals that are keys in the dictionary that do **not** appear in any of the lists of animals eaten.

The output for Part 2 for AquaticFoodWeb.txt should be:

```
Apex Predators: Bird, Fish and Lobster
```

Part 3: Identify the Producers

We will define a *producer* to be any species in the food web that does not eat another species. The output from your program should continue by displaying all of the producers with an appropriate heading.

The output for Part 3 for AquaticFoodWeb.txt should be:

```
Producers: Phytoplankton and Seaweed
```

Part 4: Identify the Most Flexible Eaters

We will define the *most flexible eater* as the organism that eats the greatest number of other organisms in the food web. Identify and display all of the most flexible eaters under an appropriate heading. The most flexible eaters in the aquatic food web are:

```
Most Flexible Eaters: Bird
```

Note: While the aquatic food web only has one most flexible eater, a food web may have several most flexible eaters that all eat the same number of organisms. When that situation occurs your program should display all of the most flexible eaters.

Part 5: The Tastiest Organism

We will define the *tastiest organism* as the member of the food web that is eaten by the most different members of the food chain. Identify and display all of the tastiest organisms under an appropriate heading. The tastiest organisms in the aquatic food web are:

```
Tastiest: Limpets and Mussels
```

Part 6: Determine the Height of Each Organism in the Food Web

We will define the *height* of an organism as the longest path from that organism to a producer. Using this definition, all producers have height 0. Any animal that eats **only** producers has height 1. All other animals in the food web have a height which is one more than the animal they prey on with largest height value.

The description of height above is recursive – it defines the height of a higher animal in terms of the height of a lower animal. As such, you may find yourself wanting to write a recursive function to assist you with determining the height of each animal in the food web. However, we will only cover recursion briefly in this course (if at all), and we will not discuss it until the final week of classes. As a result, if you choose to develop a recursive solution you should expect to do some independent study on recursion (Chapter 12 in the third or fourth edition of Starting Out with Python).

If you don't want to do some independent study on recursion then you can determine the heights of all the organisms in the food web using the following algorithm:

```
set the heights of all organisms, including the producers, to 0

set changed to true
while something has changed
  set changed to false
  for each animal, a, in the food web
    for each animal, p, that a preys on

      if the height of a is less than or equal to the height of p
        set the height of a to the height of p + 1
        set changed to true
```

The output for Part 4 for AquaticFoodWeb.txt should be:

```
Heights:
  Bird: 4
  Crab: 3
  Fish: 3
  Limpets: 1
  Lobster: 4
  Mussels: 2
  Phytoplankton: 0
  Prawn: 2
  Seaweed: 0
  Whelk: 3
  Zooplankton: 1
```

Requirements

- Your program must read the data file name as a command line argument. The food web file will be the first and only command line argument, appearing in sys.argv[1]. If no command line argument is provided then your program should read the name of the file from the user. If more than 1 command line argument is provided then your program should display an appropriate error message and quit.
- Your program must perform basic error checking, such as ensuring that the file specified by the user exists. If an error is encountered while opening a file then your program should display an appropriate error message and exit.
- You must store the predator-prey relationships in a dictionary where the keys are the predator names and the values are lists of prey eaten by each predator.
- You must only read each file once. It is **not** acceptable to submit a solution that requires the file to be read several times to complete the analysis.
- Close every file that you open.
- Your program must make appropriate use of functions. Specifically, you should write one function for each part of the assignment for parts 2 through 6. (You can also write one or more functions for part 1 if you want to). Your solution will **not** receive full credit unless your code is divided into appropriate functions that make appropriate use of parameters and return values. **A significant deduction will be made if your program is all in one function, even if it generates beautiful results.**
- The only lines of code in your program that should be outside of a function definition are constants (if any), import statements (if any) and the call to the main function (which is normally the last line in the file).
- Do **not** define one function inside another function.
- Your program must **not** use global variables (except for constant values that are never changed, and in this assignment you may not even find that you want any global constants). If you load some data inside a function then you must return it as a result so that it can be used in subsequent functions.

- You may assume that the data in the files you are working with is correct. Once you open the file successfully you don’t need to worry about problems such as reading a number where a word is expected, a missing comma, a blank line, etc.
- Your program must use good programming style. This includes things like appropriate variable names, good comments, minimizing the use of magic numbers, etc. Your program should begin with a comment that includes your name, student number, and a brief description of the program. **Each function should begin with a comment that describes its purpose, parameters (if any) and return values (if any).**
- Break and continue are generally considered bad form. As a result, you are **NOT** allowed to use them when creating your solution to this assignment. In general, their use can be avoided by using a combination of if statements and writing better conditions on your while loops.

Dealing with Command Line Parameter inside IDLE:

If you are working in IDLE instead of directly from the command prompt, you might find yourself wondering how to provide command line parameters when running inside IDLE. Unfortunately IDLE doesn't provide a good option for specifying command line parameters. Instead, the best strategy is probably to 'fake it' by adding the following line right after import sys:

```
sys.argv = ["Assignment4.py", "AquaticFoodWeb.txt"]
```

This will overwrite whatever is in sys.argv with a list that has the name of your .py file as the first element and the name of the file that you want to process as your second element. You’ll need to take this out before you submit the assignment because your TA will run your submission from the command prompt, but you should be able to test all of the cases required for the assignment by changing the values in this list:

- You can test different files (including files that don't exist) by changing the second element in the list.
- You can test the case where the user provides too many command line parameters by adding another element to the list.
- You can test the case where the user doesn’t provide a command line parameter by removing the second element from the list.

Hints:

Develop an algorithm for Part 1 before trying to write the code. The algorithm should have “For each line in the file” as an outer loop, followed by a description of the steps that you are going to follow to process the line and get the predator and prey into the dictionary correctly. Then you’ll have a separate loop which will displays the relationships that you stored in the dictionary.

Begin with Part 1 – all of the other parts rely on it. Don’t move on to the other parts until you have correct lists in Part 1.

Parts 2 through 6 can be performed in any order. Part 4 is probably the easiest to complete. Part 6 is probably the most difficult.

Having trouble formatting the output in Part 1? There's a module on the course website that you can import to do the formatting for you.

Looking for an A+? Identify the Herbivores, Omnivores and Carnivores:

A herbivore is an animal that only eats plants. In this assignment, we will assume that all producers are plants and that all plants are producers. Display the herbivores under an appropriate heading. If there aren't any herbivores then you should display the heading, followed by "(None)".

An omnivore is an animal that eats both plants and animals. Display the omnivores under an appropriate heading. If there aren't any omnivores then you should display the heading, followed by "(None)".

A carnivore is an animal that only eats other animals. Display the carnivores under an appropriate heading. If there aren't any carnivores then you should display the heading, followed by "(None)".

The herbivores, omnivores and carnivores for AquaticFoodWeb.txt are shown below:

```
For an A+:

  Herbivores: Limpets and Zooplankton

  Omnivores: Mussels

  Carnivores: Bird, Crab, Fish, Lobster, Prawn and Whelk
```

Grading:

This assignment will be graded on a combination of functionality and style. A base grade will be determined from the general level of functionality of the program (Does it output the prey for each predator? Is it formatted correctly? Does it identify the apex predators and producers? Does it identify the most flexible eaters and the tastiest organisms?). The base grade will be recorded as a mark out of 12.

Style will be graded on a subtractive scale from 0 to -3. For example, an assignment which receives a base grade of 12 (A), but has several stylistic problems resulting in a -2 adjustment will receive an overall grade of 10 (B+). Fractional grades will be rounded to the closest integer.

Total Score (Out of 12)	Letter Grade
12	A
11	A-
10	B+
9	B
8	B-
7	C+
6	C
5	C-
4	D+
3	D
0-2	F