Assignment 7

Arrays

Date Due: Wednesday, October 21, 2020, 11:59pm **Total Marks: 20**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form **aNqM**, meaning Assignment N, Question M. Put your name and student number at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you. Do not submit folders, zip documents, even if you think it will help.
- Programs must be written in **Python 3**, and the file format must be text-only, with the file extension `.py`.
- Documents submitted for discussion questions should make use of common file formats, such as plain text (`.txt`), Rich Text (`.rtf`), and PDF (`.pdf`). We permit only these formats to ensure that our markers can open your files conveniently.
- **Assignments must be submitted electronically to Canvas.** There is a link on the course webpage that shows you how to do this.
- **Canvas will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Questions are annotated using descriptors like "easy" "moderate" and "tricky". All students should be able to obtain perfect grades on "easy" problems. Most students should obtain perfect grades on "moderate" problems. The problems marked "tricky" may require significantly more time, and only the top students should expect to get perfect grades on these. We use these annotations to help students be aware of the differences, and also to help students allocate their time wisely. Partial credit will be given as appropriate, so hand in anything you've done, even if it's not perfect.

Question 1 (5 points):

Purpose: To practice working with 2D arrays.

Degree of Difficulty: Easy.

Officer Jenny is in charge of training new police recruits. She just gave her students a multipe-choice quiz. She wants to produce a summary of student performance for all questions where **fewer than 60% of the recruits got the question right**. Write a program to accomplish this task.

Using Numpy Arrays

Although in principle this question could be solved using lists only, our goal is to gain practice working with arrays. Therefore, you must convert the quiz data to an array and perform all calculations using array operations. Before you start anything, make sure you have imported the `numpy` module. This module is NOT standard, so if you are working on your own machine, you may need to install it first.

Quiz Data

A starter file is provided for you that contains the quiz data as a list-of-lists. The data is organized such that each sublist represents the quiz result for a single student. Each sublist is the same length and contains multiple integers, indicating the student's score on each question on the quiz. A value of 1 means the student got the question right and 0 means the student got the question wrong.

The program you hand in should compute the result for the `quizResults` list, but two other very small lists are provided to help you perform simple testing. For example, `test1` represents a quiz with just 2 questions that was written by 3 students, where all 3 students got the first question wrong.

Program Behaviour

Your program should:

(a) Create a 2D array from the list of lists for a quiz. Your program should work no matter how many students there are, or how many questions in the quiz.

(b) For each question, calculate the percentage of students who answered incorrectly
  - If fewer than 60% answered correctly, print the information (performance and question number) for that question to the console

Take advantage of the power of arrays to do this! In particular, you can slice a 2D array across multiple dimensions; think about how this might be useful.

Sample Run

```
Poorly done questions:
Only 45 percent of students solved question 1
Only 55 percent of students solved question 3
Only 50 percent of students solved question 8
```

What to Hand In

(a) A document entitled `a7q1.py` containing your finished program, as described above.

Evaluation

- 1 mark for creating a 2D array
- 3 marks for calculating the student performance for each question
- 1 mark for correct console output
- -1 mark if the student did not include their name, NSID, student number and instructor's name at the top of the submitted file.

Question 2 (8 points):

Purpose: To practice simple file I/O and numpy array operations

Degree of Difficulty: Moderate.

For provincial election purposes, the province of Saskatchewan is divided into constituencies. Each constituency elects a representative to send to the provincial parliament. Nearly always, each representative belongs to a registered political party.

In 2016, the Saskatchewan Party won the election with a total of 51 out of 61 constituencies (you can see the full results here: `https://results.elections.sk.ca/ge28/`). For this question, you will use real election data to determine how many of those wins were a **majority victory**, i.e. where the Saskatchewan Party won more than half of the votes cast in the constituency.

Using Numpy Arrays

The main purpose of this question is to practice data manipulation using numpy arrays.

You will need all three of: array arithmetic, array relations, and logical array indexing. You should be able to use all three of these techniques in a useful way while solving this problem.

Before you start anything, make sure you have imported the `numpy` module. This module is NOT standard, so if you are working on your own machine, you may need to install it first.

Task Breakdown

The steps below will help you break down this process into manageable pieces.

Load the Data into Arrays

Download the election results data file from the course website. It looks like this:

```
constituency,green,ndp,pc,liberal,saskparty,wip
ARM RIVER,241,1457,338,207,6187,0
ATHABASCA,53,1756,0,262,644,0
...
```

The first line of the file is a header that lists all the political parties. The other lines list the name of a constituency and the number of votes won by each party (in the order shown in the header) in that constituency.

Write code to open this file and read the election data into **two separate lists**: one for the constituency names and one for the vote tallies. Then, convert each list to a **numpy array**. Make sure to maintain the order so that the arrays line up by position (i.e. `ARM RIVER` is first in the array of constituencies, and its matching votes are first in the array of votes)

Test these arrays before you move on! From this point on, you must perform all calculations using arrays rather than lists.

Total Votes by Constituency

Next, create an array that contains the total number of votes cast in each constituency, again making sure to maintain the order of the data. Take good advantage of array operations such as slicing and array arithmetic here.

Finding the Majority Wins

Finally, use the arrays you have created so far to find all of the constituencies where the Sask Party won more than half of the total votes cast in that constituency. Make good use of **array relations** and **logical indexing** here. In particular, you can use **array relations** on two arrays so long as they are the same size; this might be useful to do using the array of total votes that you created in the previous step.

Note that you **cannot assume** what the index of the Sask Party's vote column will be. Use the header info from the file to find it!

Your program should print to the console the **number** of constituencies where the Sask Party won a majority victory, as well as the names of those constituencies. Nice formatting is not required: simply printing all of the constituency names in an array is fine.

Sample Run

The output of your program might look something like this:

```
The Saskatchewan Party won a vote majority in 45 constituencies.
---------
['ARM RIVER' 'BATOCHE' 'BIGGAR-SASK VALLEY' 'CANNINGTON' 'CANORA-PELLY'
...
...
 'WEYBURN-BIG MUDDY' 'WOOD RIVER' 'YORKTON']
```

What to Hand In

(a) A document entitled `a7q2.py` containing your finished program, as described above.

Evaluation

- 2 marks for loading the data into numpy arrays
- 2 marks for making good use of array slicing and arithmetic to find the total votes
- 3 marks for making good use of array relations and logical indexing to find the majority wins
- 1 mark for correct console output
- -1 mark if the student did not include their name, NSID, student number and instructor's name at the top of the submitted file.

Question 3 (7 points):

Purpose: To practice slicing with arrays.

Degree of Difficulty: Moderate.

Background

The evil Ice King has sent his army of penguins (all named Gunther) to attack the Candy Kingdom.[1] The Candy Kingdom's wise ruler, Princess Bubblegum, has devised a defence to ward off the invading penguin army. Princess Bubblegum's "Fish-a-pult" will lob barrels of fish onto the battlefield. This will cover an area of the battlefield with fish within which the penguins will stop to feed, and then, too full to fight, will retreat for an afternoon nap.

Suppose we have a 2D integer array representing the battlefield. Each array value represents the number of penguins currently in a 1 square meter area of the battlefield. Thus, a 100 by 100 array would represent a 100m by 100m battlefield, and each value in the array is the number of penguins in the corresponding one square meter.

The Fish-a-pult can fire barrels of fish of different weights. The weight of the fish barrel determines the size of the **square** area[2] that will be covered by fish when the barrel explodes on impact. If a barrel weighs n pounds, then it covers a square area $2n + 1$ meters on each side. Thus, a 1-pound barrel of fish will cover a 3m square area (corresponding to a 3 by 3 slice of the array!).

Your Task

Your job is to write a part of the targeting software for the Fish-a-pult.

Write a function called `find_target_location` which has the following parameters:

- `battlefield`: a 2D integer array representing the current state of the battlefield (containing the number of penguins at each location, as above)
- `fish_weight`: an integer representing the weight of the barrel of fish to be fired. The weight also determines the size of the area covered by fish, as above.

Your function should return the optimal target location (as described below) as a tuple, i.e, the pair of array offsets corresponding to the location. If there is more than one location tied for the most number of penguins, it does not matter which one you return.

Your task is to determine the optimal target location for a given quantity of fish. The optimal location is the battlefield target location which, if hit with a barrel weighing `fish_weight` pounds, will distract the most penguins. In other words, if the `fish_weight` is n pounds, then find the center of the $2n + 1$ by $2n + 1$ square on the battlefield containing the most penguins. This can be done fairly easily using loops and slicing. You will also find the `numpy.sum()` function useful.

You need not consider locations where the fish would land outside of the battlefield – these cannot be optimal locations. A target location where the entire detonation area lies within the battlefield grid will always be as good as or better than a target location where the detonation area falls off the grid because we know there are zero penguins outside the grid.

If `fish_weight` equals or exceeds either half of the battlefield width (not rounded), or half of the battlefield height (not rounded), then this means that the detonation area is larger than the entire battlefield. Thus, there are no target locations where the detonation area falls entirely within the battlefield grid. In such a case, return `None`. Firing more fish than needed to cover the entire battlefield is a waste of tasty fish!

[1] This question was inspired by Adventure Time

[2] The perfectly square area of detonation is a miracle of Candy technology.

How do I know if it's working?

Two data files are provided, as well as starter code. The starter code reads the data files which contain the battlefield data and stores the data in 2D arrays. Use these arrays and the information below to determine if your code is working.

Battlefield	Fish Barrel Weight	Expected Result
field1.csv	1	(2,2)
field1.csv	2	(2,2)
field1.csv	3	None
field2.csv	1	(1,2)
field2.csv	2	(2,3) or (4,3)
field2.csv	3	(3,4)
field2.csv	4	(4,4)
field2.csv	5	None

Files Provided

- `a6q3-starter.py`: Starter file for this assignment. This includes a function that can read CSV files containing battlefield data.
- `field1.csv` and `field2.csv`: these are files that can be read by the starter code and provide data for your program.

Evaluation

- 1 mark: Correctly returns None if `fish_weight` exceeds half of the battlefield width or height.
- 2 marks: Checks each battlefield location that would not cause the detonation area to fall outside the battlefield.
- 1 mark: Does **not** check battlefield locations that would cause the detonation area to fall outside the battlefield.
- 2 marks: Correctly calculates the number of penguins affected for the detonation area at battlefield locations.
- 1 mark: Correctly returns the center of the detonation area that will cover the most penguins.
- -1 mark if the student did not include their name, NSID, student number and instructor's name at the top of the submitted file.