CSE 160 Spring 2025 - Practice Quiz 3 Key

1. As we have seen in Homework 3, we can store 2D black and white images in a doubly nested list (rows and columns) with 0 = white and 255 = black, with our values being integers between (and including) 0 and 255. An example of a 4 x 4 pixel image is as follows:

```
4x4_image = [
             [1, 25, 90, 10],
             [10, 95, 60, 30],
             [20, 70, 85, 67],
             [40, 45, 68, 56]
             ]
```

So let's think now about black and white video data! We can think of video data as being a 3D pixel grid. Specifically, the black and white pixels in the 3D grids are stored in a triply nested list (row, columns, time marker). Let's take a look at an example of a 4x4 pixel video:

```
4x4_video = [
             [
              [1, 25, 90, 10],
              [10, 95, 60, 30],
              [20, 70, 85, 67],
              [40, 45, 68, 56]
             ],
             [
              [1, 25, 90, 15],
              [10, 95, 60, 30],
              [18, 71, 85, 67],
              [40, 45, 68, 56]
             ],
             [
              [1, 25, 90, 20],
              [10, 95, 60, 30],
              [16, 72, 85, 67],
              [40, 45, 68, 56]
             ],
             [
              [1, 25, 90, 25],
              [10, 95, 60, 30],
              [14, 73, 85, 67],
              [40, 45, 68, 56]
             ]
            ]
```

Your task is to write a function (`2_last_image`) that takes in black and white video data (in the form of the 3D pixel grids called `video_data`) and returns the image from the second to last frame. Hint: pay attention to the example grids and think about list indexing and range!

```
def 2_last_image(video_data):
    2_last = video_data[-2]
    return 2_last
```

2. You are given a CSV file tracking the days it has rained and the growth of several plants in the following format:

```
rain, mint, basil, cypress, vine
y, 2, 5, 1, 3
n, 1, 5, 0, 4
y, 4, 3, 1, 2
```

Write a function that reads in a file in the format shown above, **prints the number of days it rained**, and **returns a dictionary** with the total growth of each plant in the file. The output for the example above should be:

```
print(read_plants(file))
>>> "Rained 2 days"
>>> {"mint":7,
     "basil":13,
     "cypress": 2,
     "vine":9}
```

```
def read_plants(file):
    plant_file = open(file)  # Open the file

    days_rained = 0
    plant_growth = {}

    for line in plant_file:  # Read the file line by line
        # Split the line into columns
        columns = line.split(",")

        # Get the header with the plant names
        if columns[0] == "rain":
            plant_names = columns
        # Get the data for each day
        else:
            if columns[0] == "y":  # Count days rained
                days_rained += 1
            # Add plant growth data to dictionary
            for i in range(1, len(columns)):
                if plant_names[i] not in plant_growth:
                    plant_growth[plant_names[i]] = 0
                plant_growth[plant_names[i]] += int(columns[i])

    plant_file.close()
    print(f"Rained {days_rained} days")
    return plant_growth
```

3. You are given a dictionary as follows:

```
gdp = {2023: 27.72, 2020: 21.35, 2021: 23.68, 2022: 26.01}
years = [2020, 2021, 2022, 2023]
```

a. Give the print output of the following line of code

```
print(gdp[years[1]])
```

Output:
```
23.68
```

b. Why would the following lines of code *not* work for printing out the values in the dictionary ordered by year?

```
for year in gdp:
      print(gdp[year])
```

Answer:
Dictionaries are unordered, so values will not be printed in order of ascending year.

Also okay: The values will be printed in the order they were added, not order of ascending year.

c. Without changing the body of the for loop, edit the first line above so that it would correctly print out the values ordered by year:

```
for year in years:
      print(gdp[year])
```