Short-Answer Questions

(a) Which values does this code print?

```
for q in range(3, 10, 2):
  print(q)
```

(b) You have a reference to an unsorted list of strings in the variable `a`. Give one Python statement to put a reverse-sorted version of the list in variable `b`, *without* modifying the original list. Note: reverse-*sorted*, not just reversed. That is, sorted from greatest to least.

(c) The below line of code causes an error when the key doesn't exist in the dictionary, but you'd still like it to run when the key *does* exist in the dictionary. Augment the code below (add any other lines before or after the given line) to avoid the error when the key doesn't exist but still run the code when it does exist.

```
    print(x["offset"])
```

(d) What is printed by this code?

```
print("cccoriander"[1:9:2])
```

Unmapped First
There is a global variable `d` containing a reference to a dictionary. We wish to sort a list `q` based on whether its elements appears in the dictionary: elements of the list that do *not* appear as keys in the dictionary should come first, and elements of the list that *do* appear as keys in the dictionary should come last. Write a sorting key function that will achieve this task, *and* sort the list `q` (in place) using that key function. *You must use a key function for this problem. If you do not remember key functions or how they work, make a guess.*

Code Documentation

This code will be used for the questions on the following page.

```python
import csv
import typing

def second(x):
    return x[1]

def movieFilter(

    filename: ________________________________________,

    minRating: _______________________________________,

    chooseGenre: _____________________________________

) -> _________________________________________________:
    """








    """
    movies = []
    with open(filename) as file:
        reader = csv.DictReader(file)
        for row in reader:
            rating = int(row["rating"])
            genres = row["genre"].split(";")
            if rating >= minRating and chooseGenre in genres:
                movies.append((row["title"], rating))
    movies.sort(key=second, reverse=True)
    return movies
```

(a) Fill in the missing type annotations (each blank space given with an underline) on the previous page. Your annotations must be as precise as possible (e.g., if a list, a list of *what*). You do not need to fill in the type annotations for `second`.

(b) Fill in the missing docstring for `movieFilter` on the previous page. You do not need to fill in a docstring for `second`.

(c) Give the content of a short CSV file (three rows of data plus a header, so four lines) that would work with this code. You do not need to remember or use real movies; feel free to use titles like "A", "B", "C".

(d) Assume that your above CSV has been written to `movies.csv`. Write *three* tests for `movieFilter` using your CSV, testing three *distinct* aspects of the behavior of `movieFilter`. That is, don't simply test one returned value with three different inputs; test three different things.

Leaderboard
The file `scores.txt` contains scores for a game. Each line has a player name, then a space, then the score. Multiple rounds of the game are played, so a player may appear multiple times. For example, `scores.txt` could contain the following:

```
Alice 34
Carlos 86
Bob 94
Bob 68
Carlos 66
Bob 0
Alice 48
```

Each player's cumulative score is simply the sum of their score for each game, even though players may have played different numbers of games.

Write a Python program that reads scores from `scores.txt` in the above format and prints each player's name and cumulative score, in order by the cumulative score, from greatest to least. For example, with the above `scores.txt`, your program should print

```
Bob 162
Carlos 152
Alice 82
```

Repeater
This partial code will be used for the questions on the following page.

```
import typing

def addOne(x: int) -> int:
    return x + 1

def addOneButText(x: str) -> str:
    return x + "1"

def repeater(


    f: ______________________________________________________________,


    x: ______________________________________________________________,


    n: ______________________________________________________________


) -> _______________________________________________________________:
    """
    Return the result of applying f to x, n-many times. For instance,
    if n is 2, then return f(f(x)). If n is 0, return x.
    """
    # ...
    # Function body to be filled in later
    # ...



assert (
    repeater(addOne, 42, 2) == addOne(addOne(42))
), "Apply addOne twice"

assert repeater(addOne, 42, 0) == 42, "Apply addOne zero times"

assert (
    repeater(addOneButText, "0", 3) == "0111"
), "Apply addOneButText three times"
```

(a) Fill in the missing type annotations (each blank space given with an underline) on the previous page. Your annotations must be as precise *as possible*; do not cause type errors in the tests or other reasonable uses of `repeater` due to over-precise type annotations, but be as precise as possible otherwise (this will require at least one imprecise type for this code).

(b) Write a function body for `repeater` that will do what the docstring describes. `repeater` with your body must work with the tests given on the previous page. You must assert that `n` is non-negative. (Do not write the body in the code on the previous page. Write it here.)

(c) Create a function `repeaterBreaker`. This function must

- Work if passed to `repeater` as `f` with `n` set to `0`.
- Work if passed to `repeater` as `f` with `n` set to `1`, with at least some value of `x`. It does not need to work for *all* possible values of `x`, only some given value.
- Crash (that is, report an error, not merely give a surprising result) when passed to `repeater` as `f` if `n` has any value greater than `1`, for the same `x` that it works with if `n` is `1`.

Style: You do not need to write a docstring or tests for this function, but must write correct type annotations.

(d) Give an example value of `x` that works with `n=1` but crashes with `n=2` with your above `repeaterBreaker`.

```
x = ______________________________________________________________
repeater(repeaterBreaker, x, 1) # No crash
repeater(repeaterBreaker, x, 2) # Crash
```