CS 171
Week 9 Lab

Overview

This lab consists of a collection of problems that you may solve in any order. To earn full credit, you must submit solutions to all problems at one time.

Important:

- You must be present in the lab session to earn credit
    - If you are recorded as absent but still complete the work, you will not earn credit. If you arrived after the CA/TA took attendance, be sure to check in so you are recorded as present/late.
    - If you miss the lab meeting, contact your instructor via email right away to request a make-up session.
- You may collaborate with at most one lab partner; Feel free to switch partners next week if you wish
- Each student must submit their own work to Gradescope for credit, even if you worked with a partner
- At most 100 points are possible. To get a combined score, submit all of your solutions at once.
- Please only use features of the language that have been covered in reading and lectures
    - Solutions which make use of language features not yet covered will not receive credit.
    - Students whose submissions repeatedly make use of additional skills not covered in the course so far may be suspected of violating academic integrity.
    - *Even if you know the language already, please work within the subset of the language covered up to this point in the course.*
- You may not use artificial intelligence tools to solve these problems.
- Your lab work is due at the end of the lab period. Late work will be accepted with a 30-point penalty if it is submitted within 12 hours of the end of your lab.

Helpful to Know:

- The last score earned will be recorded for your lab score
- To get a combined lab score, you must submit multiple files at the same time
- Lab assistants are here for your support. Ask them questions if you are stuck!

Materials to Use:

- Lecture slides for Weeks 1 – 9
- Think Python Chapters 1 – 10 and 12 – 13
- Prof. Boady's Chapters on Linear Search & Binary Search

Problem A

Submission File: `lab09a.py`

A `GameShowHost` is offering to give you a fantastic prize as long as you can guess the value of the prize before the host gets bored and walks away. You can ask the `GameShowHost` to tell you the lowest possible value and the highest possible value of the prize, but they won't tell you the actual value -- you have to guess. All secret prices are positive integer values.

In Week 4, you learned about string methods, which are commands that can be run on an instance of a string, such as `someString.isdigit()` or `someString.startswith(...)`. In this problem instead of using a string, we will be interacting with a `GameShowHost` object and its commands. The `GameShowHost` code is pre-loaded into Gradescope, and you will not have access to a copy of for your local development purposes.

Here is some sample code that demonstrates interaction with a `GameShowHost`:

```
drewCarey = GameShowHost(1, 10)    # A new host named drewCarey
low = drewCarey.getMinValue()      # Returns 1
high = drewCarey.getMaxValue()     # Returns 10
aGuess = 5
result = drewCarey.guess(aGuess)   # Returns "Higher", "Lower", "Correct", or
                                   # "Out of Guesses"
print(result)                      # Prints "Lower"
```

You address the `GameShowHost` by name (in the case above, `drewCarey`) followed by a dot and then what you'd like to ask. You can ask to `getMinValue()`, `getMaxValue()`, or make a `guess(someNumber)`. The `GameShowHost` will respond to your guess by returning a String, as described in the comments above.

```
def playGame(host):
    low = host.getMinValue()
    high = host.getMaxValue()
    for guess in range(low, high + 1):
        if host.guess(guess) == "Correct":
            return guess
    return -1
```

This starter code receives a pre-configured `GameShowHost` object and attempts to find the secret price by guessing each number in the range from low to high. Copy the playGame function above into an appropriately-named file and submit it to Gradescope to confirm that the naïve strategy works for some games.

You should modify this code to use a more sophisticated guessing strategy.
The `playGame()` function should return the price you discover (as an integer), or a negative value if you cannot guess the price before you run out of guesses.

Problem B

Submission File: `lab09b.py`

We know that all printable characters have corresponding numeric values defined by the Unicode standard, which are easily found by using the built-in `ord()` function.

Let's define a score for a string to be the sum of the numeric values of its characters.

```
The score for "Hello CS 171" is 867:
The value of 'H' is 72.
The value of 'e' is 101.
The value of 'l' is 108.
The value of 'l' is 108.
The value of 'o' is 111.
The value of ' ' is 32.
The value of 'C' is 67.
The value of 'S' is 83.
The value of ' ' is 32.
The value of '1' is 49.
The value of '7' is 55.
The value of '1' is 49.
The sum of these is 867
```

First, write the function `stringScore(string)` which returns the score for any given string. Next, write the function `stringHighScore(string, token)` which breaks apart the given string using the provided token. This function determines the score for each part of the string (excluding the token character) using the `stringScore(…)` function and returns the substring with the highest score.

For example, `stringHighScore("Everything passes. Everything changes. Just do what you think you should do.", " ")` returns "Everything" because it is the substring with the highest score, as shown below.

```
The value of 'Everything' is 1061.
The value of 'passes.' is 701.
The value of 'Everything' is 1061.
The value of 'changes.' is 775.
The value of 'Just' is 422.
The value of 'do' is 211.
The value of 'what' is 436.
The value of 'you' is 349.
The value of 'think' is 542.
The value of 'you' is 349.
The value of 'should' is 655.
The value of 'do.' is 257.
```

Breaking this string on periods instead of spaces would result in `stringHighScore("Everything passes. Everything changes. Just do what you think you should do.", ".")` returning `" Just do what you think you should do"` as shown below:

The value of 'Everything passes' is 1748.
The value of ' Everything changes' is 1854.
The value of ' Just do what you think you should do' is 3431.
The value of '' is 0.

Revise stringHighScore(...) so it will operate with or without a token. If no token is provided, assume the default token of a space (' '). `stringHighScore("Hello CS 171")` should return "Hello".

This exercise relies on the linear search algorithm to find the largest value.

Problem C

Submission File: `lab09c.py`

Everything you eat as food consists of three macronutrients: proteins, carbohydrates, and fats. The energy that food contains is measured in Calories. A gram of fat contains 9 Calories. A gram of protein contains 4 Calories. A gram of carbohydrate also contains 4 Calories.

While it's not relevant to this problem, 1 Calorie contains 1,000 calories.

For this activity, write a function, `totalCalories(proteins, carbohydrates, fats)` which receives the number of grams of proteins, carbohydrates, and fats and returns the number of Calories contained within the food represented. If the value of any parameter is negative or non-numeric, that value should be treated as 0.

The default order of the parameters should be proteins, carbohydrates, and fats. Calling `totalCalories(1, 10, 100)` should return 944, consisting of 1 gram of proteins (4 Calories), 10 grams of carbohydrates (40 Calories) and 100 grams of fats (900 Calories).

The default value of each of the parameters should be 0. Therefore, calling `totalCalories(5)` should return the correct Calorie count for a food consisting of 5 grams of protein. Calling `totalCalories(8, 6)` should return the correct Calorie count for a food consisting of 8 grams of protein and 6 grams of carbohydrate.

Finally, your function should recognize parameter names proteins, carbohydrates, and fats such that values can be passed in by name. For example, `totalCalories(fats=4)` should return 45 Calories and `totalCalories(proteins=8, fats=6)` should return 86.