CS 115 Lab 11: Loops and Reading Stack Traces

This lab has 2 parts. The first half is focused on *reading a stack trace to debug*. The second half is focused on loops and may be helpful practice for Homework 6.

Reading Stack Traces

In this section, we will practice *reading* exceptions, and using that information to fix mistakes in our code.

We have given you two functions, which when used together, cause an error: There are at least 2 mistakes, one much easier than the other to fix.

1. `compute_total_regeneration_points_in_inventory`: Given an inventory of `Items`, computes sum of all regeneration points for all items in the inventory.
2. `create_healer_inventory`: Creates an inventory for a healer character as a dictionary.

Mistake 1

First, we will fix the bug that is causing the following stack trace on Alex's computer. For this first bug, the mistake is in `compute_total_regeneration_points_in_inventory`.

```
Traceback (most recent call last):
  File "/Users/alagrass/git/cs115/lab11/src/hw5.py", line 83, in <module>
    total_regeneration_points = compute_total_regeneration_points_in_inventory(inventory)
  File "/Users/alagrass/git/cs115/lab11/src/hw5.py", line 72, in
      compute_total_regeneration_points_in_inventory
    total += item_quantity * item.regeneration_points
TypeError: 'int' object is not iterable
```

Report question

Fixing the first bug: Uncomment and run the code at the bottom of the file *on your computer*. Copy and paste the stack trace you get to the report, then answer the following questions about it:

1. What is the *most recent* line of code being run when the error occurred?
2. What function was calling that function?
3. Explain in your own words what that error *means* about what went wrong.
4. Use the error to fix *one line of code causing the bug.*

Mistake 2

After fixing that bug, we still see another error! This one is more difficult to debug. Repeat the same procedure, **focusing on interpreting the error message to narrow down the problem rather than reading all the code**. This time, the bug can be in any of the two functions we gave you.

Report question

Repeat the process as before, but without fixing the code. Start by copying and pasting the stack trace you got to your report.

1. What is the *most recent* line of code being run when the error occurred?
2. What function was calling that function?
3. Explain in your own words what that error *means* about what went wrong.
4. Do not fix the code yet! Leave that for the CA Check-in.

CA Check-In

1. Share your progress or plan to work on Homework 6.
2. Explain the stack trace you got line by line to the CA.
3. Share *any* ideas for locating the error in the code with the CA.
4. Fix the code if you can. Otherwise, the CA will ask you guiding questions to find the error based on the stack trace.

Test that the code you have computes the correct number of regeneration points without errors and submit what you have so far to Gradescope.

Loops

In this part of the lab, we will solve problems with loops. For each problem, think about:

- How you will accumulate results (accumulator variable)?
- What you will be iterating over?
- What loop structures would be most helpful?

Task 1: While loops

In this first task, implement `num_halves_until_one`, which calculates how many times you can divide an integer n by 2 until it reaches 1.

Continue dividing n by 2 (`n // 2`) until `n == 1`. Return the number of divisions it took to reach n == 1. Use integer division: `n // 2` or `int(n/2)`.

Task 2: For Loops

These are practice problems with loops designed to be a warmup for Homework 6.

1. First, implement `accumulate_costs`, which given a list of numbers, returns a **list of the sums of each element with all elements left of it**.

   For example, `accumulate_costs([1,4,9])` should return `[1, 1+4, 1+4+9]`, which is equivalent to `[1,5,14]`.

   ```
   >>> accumulate_costs([1,4,9])
   [1, 5, 14]
   ```

2. Next, implement `accumulate_col_costs`, which given a 2D grid of costs (`grid`) and a `column_index`, returns the list of sums of each element in column `column_index` with all the elements above it.

```
# Example 3 x 2 grid
grid = [[4,  1],
        [7,  3],
        [2,  5]]

>>> accumulate_col_costs(grid, 0)
[4, 11, 13]
>> accumulate_col_costs(grid, 1)
[1, 4, 9]

# Another 3 x 2 grid written differently
another_grid = [[1,  0],[2,  0],[3,  0]]
>>> accumulate_col_costs(another_grid, 0)
6
>> accumulate_col_costs(another_grid, 1)
0
```

3. Finally, write a function that counts the number of **unique** numbers that appear **somewhere** in **both of** two 2D grids of integers. You can consider using nested for loops: one iterating over rows, and the other over columns.

```
gridA = [
    [4,  1],
    [7,  3],
    [2,  5]
]

gridB = [
    [1,  9],
    [4,  3],
    [8,  2]
]
gridC = [
    [1,  1],
    [1,  2]
]
gridD = [
    [1,  1],
    [2,  1]
]

>>> count_shared_values(gridA, gridB) # shared: 1, 4, 3, 2
4


>>> count_shared_values(gridC, gridD) # shared: 1, 2 (unique)
2
```

Submission Instructions

For this lab, you must submit the following files, with exactly the following names:

- `lab11.py`
- `lab11_report.txt`

Remember to put your name and the Stevens Honor Pledge on every file you submit! Failure to put the pledge can result in marks being deducted, with repeat penalties potentially getting stricter between assignments.

Submissions must be handled through Gradescope (accessible through the homework page on Canvas). Grades will be released through Gradescope once the homework is graded.

Rubric

Category	Description	Points
Name & Pledge	For stating your name and writing the full Stevens Honor Pledge on every file for submission.	5
Debugging: Mistake 1	For correctly analysing and correcting the error for Mistake 1.	20
Debugging: Mistake 2	For correctly analysing the error for Mistake 2.	15
Check-In	For performing the CA Check-In during the lab.	10
Task 1: `num_halves_until_one`	For correctly implementing `num_haves_until_one`	10
Task 2: `accumulate_costs`	For correctly implementing `accumulate_costs`	15
Task 2: `accumulate_col_costs`	For correctly implementing `accumulate_col_costs`	10
Task 2: `count_shared_values`	For correctly implementing `count_shared_values`	15
Total		100

Notes and Reminders

- **Remember to include docstrings!** If your submission does not have docstrings, or has docstrings that are not related to your code, you will not get credit for them.
- We have provided a test file for you to test your solutions. By running this test file, you should be able to verify that your files are named properly, and that your code works correctly.
- Test frequently! Try out many different cases to verify that your code runs. Our provided test file may not be comprehensive, and so you should ensure to test your code on many other cases as well.
- Code that doesn't compile (throws a `SyntaxError` on being run) will receive a 0 for the assignment! Make sure you submit code that can be run, even if it does not pass all the test cases.
- **Don't forget to add your name and pledge!**