CSCI 150: Exam 2—Redo Problems

Directions: If you received less than a *Complete* assessment on Exam #2, this contains extra problems that you might need to work. Please check *your* assessment rubric form from the exam. Follow the directions for which problems you need to do.

- If you have any tracing problems to do, those should be done on paper
- If you have to rework either #4 or #5 from the in-class, your solutions should be written out by hand – though you are welcome to test your code on a machine. Please pay attention to indentation!
- Any functions you redo from either the Take-Home or the document below should be together in a single .py or .ipynb file.
- For the redo, you can ask me any questions, as well as your classmates, and the CSCI tutors. You may not look up anything on the internet, except for the resources on the class Teams page, the class webpage, or the official Python 3 documentation.

Tracing

Tracing Problem #1: Trace the following code:

```
def f1(n: int) -> str:
    a = 'exam 2'
    i = 0
    while i < len(a):
        if n >= len(a):
            return a
        else:
            if a[i] > a[n]:
                return a[0:i]
        i += 1
    print('Hello')
    return a

def f2(n: int) -> int:
    if n < 6:
         return len(f1(n))

    print(n)
    return len(f1(n // 2))

def main1():
    print(f2(3))
    print(f2(10))
    print(f1(7))

main1()
```

Tracing Problem #2: Trace the following code:

```
def main2():
    i = 0
    s = 'computer'
    t = 'a'
    j = 0
    while i < len(s):
        if s[i] > t:
            t = s[i]
        else:
            j += i
        i += 1

    print(t)
    print(j)

main2()
```

Additional Coding

1. Write a function `list_diff` which takes in a list of intergers `lst`, which will always have at least two entries. Your function should return the difference between adjacent entries as a new list.

   For example:

   - `list_diff([7, 2, 2, 4, 10])` should return `[5, 0, -2, -6]`. (Notice that the answer does pay attention to the order in which the entries appear!)
   - `list_diff([3, 3, 3, 3])` should return `[0, 0, 0]`
   - `list_diff([4, 3, 2, 1, 0, -1])` should return `[1, 1, 1, 1, 1]`
   - `list_diff([6, 4])` should return `[2]`
   - `list_diff([12, 16, 0, -3, 11, 19, 1])` should return `[-4, 16, 3, -14, -8, 18]`

2. Write a function `str_count` which takes no parameters, but asks the user to input a string `s` (always assumed to be lower case letters) and an integer `n`, and returns an integer. If n is either longer than or equal to the length of `s` or negative, the function should return `0`. Otherwise, it should return a count of the number of characters in `s` closer to the front of the alphabet than `s[n]`.

   For example, consider the following transcript:

```
Please enter a string of lower case letters: hello
Please enter an integer: 3
```

This should return `2`, since the 'h' and 'e' are each smaller than `s[3] = 'l'`.

```
Please enter a string of lower case letters: hello
Please enter an integer: 2
```

This should return `2`, since the 'h' and 'e' are each smaller than `s[2] = 'l'`.

```
Please enter a string of lower case letters: hendrix
Please enter an integer: 3
```

This should return `0`, since no letter of 'hendrix' is smaller than 'd'.

```
Please enter a string of lower case letters: hendrix
Please enter an integer: 7
```

This should return `0`, since `7` is at least as long as 'hendrix'

```
Please enter a string of lower case letters: william
Please enter an integer: 6
```

This should return `5`.