Test 3 – *SOLUTION* to Practice Problems for the Paper-and-Pencil portion

1. Consider the two functions below and the user input shown below that.

```
def sum_numbers1():
    total = 0
    while True:
        number = float(input('Enter a number (0 to quit): '))
        if number == 0:
            break
        total = total + number

    return total


def sum_numbers2():
    total = 0
    while True:
        try:
            number = float(input('Enter a number (0 to quit): '))
            if number == 0:
                break
            total = total + number
        except ValueError:
            print('  You entered a string that is NOT a number.')
            print('  Try again.')

    return total
```

User enters:

30

5

zero

4

0

Questions are on the next page.

a. What happens if a program calls **sum_numbers1** and the user attempts to enter the user-input shown on the previous page?

Answer: After the user enters **zero** the call to the **float** function raises an **Exception** because the **float** function cannot convert the string **zero** to a floating point number. (FWIW, it is a **ValueError exception**.)

Execution leaves the **sum_numbers1** function at that point. The caller then has the opportunity to handle the **Exception**. If it does not do so, the caller's caller has the opportunity to hand the **Exception**. And so forth until either a caller handles the **Exception** or we reach the point at which **main** is called.

If we reach the point at which **main** is called, the program displays (in red, on the Console) the sequence of calls that led to the **Exception** being raised and the printed form of the **Exception**. (FWIW, that sequence of calls is called the ***Traceback*** or ***Stack Trace***).

b. What happens if a program calls **sum_numbers2** and the user attempts to enter the user-input shown on the previous page?

Answer: As in part (a), after the user enters **zero** the call to the **float** function raises an **Exception** because the **float** function cannot convert the string **zero** to a floating point number. (FWIW, it is a **ValueError** exception.)

This time, the **sum_numbers1** function "catches" the **Exception** in the **except** clause of the **try/except** expression. The code in the **except** clause prints a message to the user and the **while** loop continues.

The next user input (**4**) is added to the total. The user input after that causes the **while** loop to finish and the function then returns **39**.

c. Why might it be better for the program to call **sum_numbers2** instead of **sum_numbers1**?

Answer: The **sum_numbers2** function allows the user to make mistakes and recover. Note that the **sum_numbers2** function does not catch ALL user mistakes that are possible. For example, if the user types **30** when she meant

31, or if the user enters **0** before she intended to finish entering numbers, **sum_numbers2** would not catch those errors.

2. Consider the following statements:

```
c1 = rg.Circle(zg.Point(200, 200), 25)
c2 = c1
```

At this point, how many ***rg.Circle*** objects have been constructed? **1** **2**
(circle your choice)

3. Continuing the previous problem, consider an additional statement that follows the preceding two statements:

```
c1.radius = 77
```

After the above statement executes, the variable ***c1*** refers to the same object to which it referred prior to this statement. **True** **False**
(circle your choice)

4. Continuing the previous problems:

- What is the value of ***c1***'s radius after the statement in the previous problem executes? **25** **77** (circle your choice)
- What is the value of ***c2***'s radius after the statement in the previous problem executes? **25** **77** (circle your choice)

5. Which of the following two statements mutates an object? (Circle your choice.)

```
numbers1 = numbers2
```

```
numbers1[0] = numbers2[0]
```

6. Mutable objects are good because:

Answer: They allow for efficient use of space and hence time – passing a mutable object to a function allows the function to change the "insides" of the object without having to take the space and time to make a copy of the object. As such, it is an efficient way to send information back to the caller.

7. Explain briefly why mutable objects are dangerous.

Answer: When the caller sends an object to a function, the caller may not expect the function to modify the object in any way. If the function does an unexpected mutation, that may cause the caller to fail. If the object is immutable, no such danger exists – the caller can be certain that the object is unchanged when the function returns control to the caller.

8. What is the difference between the following two expressions?

```
numbers[3]                    numbers = [3]
```

Answer: **The expression on the left refers to the index 3 item in the sequence called *numbers*. It refers to that item but changes nothing (of itself). The statement on the right sets the variable called *numbers* to a list containing a single item (the number 3).**

9. Consider the two functions below.

```
def f1(list of numbers):
    """
    RETURNs a new list that is the same as the given list of numbers
    except that the last item in the list is doubled.  For example, if
    the given list is [4, 2, 8, 5], this function returns [4, 2, 8, 10].
    """

def f2(list of numbers):
    """
    MUTATEs the given list of numbers by doubling the last item
    in the list. For example, if the given list is [4, 2, 8, 5],
    then this function mutates the list to [4, 2, 8, 10].
    """
```

Suppose that the main program has a list named my_list that contains 10,000 numbers.

Suppose further that the program makes two function calls:

- **f1(my_list)**
- **f2(my_list)**

Which function call will take longer to execute? (Circle your choice.) Why?

Answer: The call to **f1** copies all 10,000 items in the argument (but with the last item doubled). The call to **f2** does no copying. Instead, it accesses the last item in the argument and modifies only that single item. As such, call to **f2** changes only 1 item, while the call to **f1** constructs 10,000 items – the latter takes much more time than the former.

10. In the space below, write an implementation for the function whose specification is shown in the following box. Do NOT use your computer for this (or for any other of these paper-and-pencil problems).

```
def shape(r):
    """
    Prints shapes per the following examples:
    When r = 5:                          When r = 3
        *****5                           ***3
        ****54                           **32
        ***543                           *321
        **5432
        *54321
    Precondition:  r is a non-negative integer.
    For purposes of "lining up", assume r is a single digit.
    """
```

One answer:

```
    for k in range(r):
        for j in range(r - k):
            print('*', end='')
        for j in range(k + 1):
            print(r - j, end='')
        print()
```

11. Consider the code snippet below. It is a contrived example with poor style, but it will run without errors. What does it print when ***main*** runs?

Write your answer in the box to the right.

```
def main():
    for j in range(5):
        for k in range(j):
            print(j, k)
```

Output:

(I have put extra blank lines in this solution to make it more readable.)

```
1 0

2 0
2 1

3 0
3 1
3 2

4 0
4 1
4 2
4 3
```

Output:

(I have put extra blank lines in this solution to make it more readable.)

```
here
there

here
there

here
there
2 2

here
3 1
there
3 2
3 3

here
4 1
4 2
there
4 2
4 3
4 4
```

12. Consider the code snippet below. It is a contrived example with poor style, but it will run without errors. What does it print when ***main*** runs?

Write your answer in the box to the left.

```
def main():
    for j in range(5):
        print('here')
        for k in range(1, j - 1):
            print(j, k)

        print('there')
        for k in range(2, j + 1):
            print(j, k)
```

13. Consider the code snippet in the box below. It is a contrived example with poor style, but it will run without errors. What does it print when ***main*** runs?

    Write your answer in the box shown to the right of the code.

```
def main():
    seq = [('one', 'two', 'three', 'four'),
           ('five', 'six', 'seven'),
           ('eight', 'nine', 'ten'),
           ['is this ok?'],
           (),
           ('123456', '1234')]

    for k in range(len(seq)):
        for j in range(len(seq[k])):
            print(j, k)
            if len(seq[k][j]) > 3:
                print(seq[k][j], len(seq[k][j]))
```

Output:

(I have put extra blank spaces and lines in this solution to make it more readable.)

```
0   0
1   0
2   0
three  5
3   0
four  4

0   1
five  4
1   1
2   1
seven  5

0   2
eight  5
1   2
nine  4
2   2

0   3
is this ok?  11

0   5
123456  6
1   5
1234  4
```