CS5001 Spring 2022 SF and SV Combined
Midterm Exam (100 points)

1. (36 points) Fill in the blanks to match Python errors on the left with a category of error on the right. Categories from the right may be used as many times as needed.

a. Dash in variable name	4. Syntax Error	1. Logical/Semantic Error
b. TypeError	3.Runtime Error	2. PEP8 Style Error
c. No output when run	3.Runtime Error	3. Runtime Error
d. Unexpected indent	2.PEP8 Style Error	4. Syntax Error
e. RecursionError	3.Runtime Error	5. Non-PEP8 Style Error
f. NameError	3.Runtime Error	6. Inefficient Algorithm Error
g. KeyboardInterrupt	3.Runtime Error	
h. Blank line contains whitespace	2. PEP8 Style Error	
i. Expected does not match Actual	1.Logical/Semantic Error	
j. Overly complex construction*	6. Inefficient Algorithm Error	
k. Expected 2 blank lines …	2. PEP8 Style Error	
l. ZeroDivisionError	3.Runtime Error	
m. MemoryError	3.Runtime Error	
n. Missing/unbalanced parentheses	4. Syntax Error	
o. Line too long	2. PEP8 Style Error	
p. Poor choice of variable name	5. Non-PEP8 Style Error	
q. ValueError	3.Runtime Error	
r. IndexError	3.Runtime Error	

* Example: Versus:

```
if bool1 == True:
    return True
else:
    return False
```

```
return bool1
```

2a. (44 points) This problem exercises the use of **while** loops and **lists**. The goal is to create a **list** of ***validated*** user inputs. To get started, download the file called **number_checks.py**. This will facilitate the validation of integer and float values requested from the user (because doing this properly involves try/except, which will be taught *after* the midterm). Save this file in the same folder as your solutions to the midterm. At the top of your solution file for this problem, include this line:

```
from number_checks import *
```

The file provides two functions you may call as part of your solution:

`int_ok(str1)`: returns True iff it is safe to convert str1 to an int

`float_ok(str1)`: returns True iff it is safe to convert str1 to a float

Starting from our standard **template.py**, create a new file called **validated_list.py**. It should contain a function, **validated_list()**, which prompts the user for a ***sequence of numbers***. There are two flavors of legal values for the list:

- a positive **int** less than 100
- a positive **float** less than 100

If the user enters a float which happens to be an exact integer,[1] then **use its *integer* value instead**. Each valid input should be added to the end of a growing list. If the user responds with just the **ENTER/RETURN** key -- returned by input() as the *empty string* -- then they are done. The **list** that has been constructed so far should be returned to the caller. For any other entry, your function should print a polite error message and ***ask for input again***, until a valid input is received and added to the list, or the user signals a desire to quit by hitting the ENTER/RETURN without typing anything first. Sample output from the function is shown below.

Sample input/output from directly calling validated_list() is shown.

```
>>> validated_list()
    Please enter a positive int or float less than 100: 0
    Invalid entry (ignored). Please try again.

    Please enter a positive int or float less than 100: -1
    Invalid entry (ignored). Please try again.

    Please enter a positive int or float less than 100: 1
    OK. You entered 1

    Please enter a positive int or float less than 100: 99.0
    OK. You entered 99.0

    Please enter a positive int or float less than 100: 3.14
    OK. You entered 3.14

    Please enter a positive int or float less than 100: 100
    Invalid entry (ignored). Please try again.

    Please enter a positive int or float less than 100:
    Done.
    [1, 99, 3.14]
```

[1] See https://docs.python.org/3/library/stdtypes.html#additional-methods-on-float.

2b. (20 points) Your **main()** should call **validated_list()** as many times as needed for adequate testing, printing out the result list from each run. Each test should first print ***instructions for a human tester***, prescribing what values to enter; then, at the end of each test, it should print what resulting list was *expected* (as well as the *actual* list returned). Be sure that your tests include an empty list, repeated attempts to enter invalid data, cases where an entered float's value turns out to be an exact int, etc.