H2 Computing Practical Worksheet 4

Set 1 – Please attempt the following questions by implementing the solutions using Python 3.

1 Consider the following code:

```
numbers = [x for x in range(1000)]
i = 0
while True:
     print(str(numbers[i]))
     i += 1
```

Clearly, there is a bug in this code. Identify the bug and then use a “try-except” statement to “fix” the code so that it works as intended. (Note that you may not utilise “else” or “finally” in the “try-except” statement.)

2 Consider the following code:

```
From random import randint
name = “”
for i in range(100):
     name = name + str(randint(0, 9))
name = name + “.txt”

f = open(name, “r”)
data = f.read()
f.close()
```

Once again, there is a problem with this code. Use the “try-except” statement to ensure the code does not cause a runtime error. (Note: you may not change the way name is generated or used.)

3 Write code that continually asks for rational number input from the user (until the exit string is input). For each value entered, (if it is not the exit string) utilise exception handling to verify that the inputs are indeed rational numbers before saving them to the file called RATIONALS.TXT. (Note that your code should open, write, and close the file with each entry. Also comment why this may be a more prudent course of action.)

Set 2 – Please attempt the following questions by implementing the solutions using Python 3.

4 The file WORDS.TXT contains a list of single word computing terms used in a textbook.

Each entry has the following format:
<computing term>
<number>

One of the file entries (in both files) is:
program
52

This means that after a complete scan of the textbook the word 'program' was found 52 times.

By utilising the "try-except" statement, write program code to find and output the term with the highest number of occurrences.

5 Typically, an "if-else" statement is utilised in recursive code to differentiate between the general and base case. Write code that recursively generates all possible legal permutations of k brackets by utilising the "try-except" statement instead of the "if-else" statement. (Note that you may not use the "else" or "finally" options in the "try-except" statement that you implement.)

Example, with k = 3, the legal permutations are: ((())), ()(()), ()()(), (())(), (()()).

6 Write your own interpreter for a mathematical calculator language!

Assume that in this language, one may use the following operators, functions, constants and the assignment and print statements.

I.e., each line of the code comprises either an assignment or print statement with nested expressions in the forms:

<expression>: <variable or value or expression> <operator> <variable or value or expression>
OR
<expression>: <function> (<expression>)

+	–	*	/	//	%	**	=
floor(*x*)	ceil(*x*)	abs(*x*)	min(*x*,*y*)	max(*x*,*y*)	pi	e	print(*x*)

The language must also accept the use of variables (which may not contain any of the above operator symbols, or correspond to any of the above function names and constants). The language must of course also accept any rational number and the use of brackets.

Just like a Python .py file, you will read .calc files which will execute the calculator instructions specified within. Implement your interpreter with all exception handling in place to warn the users about errors in their code.

Additional References

- "How to Think Like a Computer Scientist" Appendix A:
  http://www.greenteapress.com/thinkpython/thinkCSpy/html/app01.html

- "Python Programming" Section on Errors and Exceptions:
  https://en.wikibooks.org/wiki/Python_Programming/Errors

- "How to Think Like a Computer Scientist" Chapters 11.5:
  http://www.greenteapress.com/thinkpython/thinkCSpy/html/chap11.html

- "Python Programming" Section on Exceptions:
  https://en.wikibooks.org/wiki/Python_Programming/Exceptions

There are many other exceptional resources available on the internet. If you find any, please share with the class via group chat.