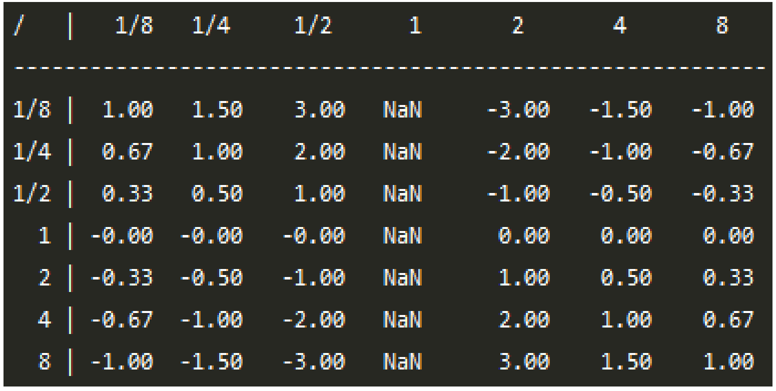
Chapter 6 Lab : Files

Goal: This lab is intended to help you understand file I/O and exceptions. You should hand in at least one *.py* file for each of the below problems. Please name and number the problems appropriately (something like *Lab6_1_YourName.py*). Each code should contain extensive comments in the style discussed in class and output should be clearly displayed and labeled.

1. The file *coordinates.txt* contains multiple sets of numbers corresponding to (x, y) coordinates. Each number is on its own line, with a y coordinate following the corresponding x coordinate. For example:
   1.2
   3.4
   -2.2
   -0.9
   corresponds to two coordinates, the first is $(1.2, 3.4)$ and the second $(-2.2, -0.9)$. Create a code which reads in from the file and outputs, to the terminal,

   – The average coordinates (mean of the x-values, mean of the y-values)
   – The maximum & minimum distances from the origin to the coordinates (remember your distance formula!)
   Note that you should not assume a fixed size for your file (you do not know how many coordinates you are reading in until the file is opened and read). Use a *try-except* block to make sure the file exists. Do not use lists to store any of the x, y values.

2. Print out a division chart for $\frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 0, 2, 4, 8$ (yes, I know this isn't correct mathematically) to a file *divPowerTwo.txt*. Make sure to include a header, a 'lefter' and lines between each row. Make the number on the left the numerator and the number on the right the denominator, as shown below:



```
/   |  1/8   1/4    1/2     1      2      4      8
------------------------------------------------------------
1/8 |  1.00   1.50   3.00   NaN   -3.00  -1.50  -1.00
1/4 |  0.67   1.00   2.00   NaN   -2.00  -1.00  -0.67
1/2 |  0.33   0.50   1.00   NaN   -1.00  -0.50  -0.33
  1 | -0.00  -0.00  -0.00   NaN    0.00   0.00   0.00
  2 | -0.33  -0.50  -1.00   NaN    1.00   0.50   0.33
  4 | -0.67  -1.00  -2.00   NaN    2.00   1.00   0.67
  8 | -1.00  -1.50  -3.00   NaN    3.00   1.50   1.00
```

   Use a *try-except* to catch the divide by zero and print a NaN. Print only two decimal places for each result. Make sure to use *try-except* for file I/O.

3. The file *passwords.txt* contains some of the most common all-lowercase alphabetic passwords. Create a file *newPasswords.txt* which is identical to *passwords.txt*, except:

   – Replace the first character of each password with it's uppercase equivalent, no matter what the character is.
   – For each other character replace with some *l*337 characters:
     ⋆ *a* is replaced with 4
     ⋆ *c* is replaced with (
     ⋆ *e* is replaced with 3
     ⋆ *t* is replaced with 7
     ⋆ *i* is replaced with !
     ⋆ *s* is replaced with $
   For example, the first two entries of your file should be:
     P4$$word
     Qw3r7y

Do *not* use lists (including strings or character lists) or hard-code the number of characters/lines in the source file. You should process the file character by character.

4. Revisit your "Yahtzee" type code from Project 1. You will write and read from a file, *highScore.txt*, whose first line is the highest score that anyone has gotten so far. The second line should be the high scorer's name. Make sure to do all of the following

   - Create the *highScore.txt* file if it does not yet exist (using Python, not manually)
   - On an 'intro screen', before the user starts rolling dice, tell them what the current high score is and whose score it is.
   - Use a try-except to make sure the user does not use an invalid choice for which dice to reroll. If they do (if their answer to which dice they want to re-roll is 'bob') then give them a message, re-display the question, and ask again. Keep doing this until they give a valid answer.
   - If the user receives a score higher than the current high score, ask them for their name and re-write the file accordingly.

   Make sure to test your code for multiple situations and use *try-except* for file I/O.