File Input-Output II and Exceptions

Model 1 File names and paths

Here are some useful commands when working with files:

Python code	Description
`import os`	`A useful module for dealing with filenames and paths`
`os.getcwd()`	`Returns the path of the current working directory as a string`
`os.path.abspath('out.txt')`	`Returns the absolute (exact/longer) path of the out.txt file as a string`
`os.path.exists('out.txt')`	`Returns True/ False based on whether a file with that name exists`
`os.path.isdir('out.txt')`	`Returns True/ False based on whether the path provided is a directory`
`os.listdir('target/path/')`	`Returns a list with the names of the files and directories residing at the target path`
`os.path.join('target/path','file.txt')`	`Takes a directory name and a file name and joins them into a complete path`

Model 2 `with` statement

To avoid forgetting to close a file, you may consider using a `with` statement for working with files, that looks like this:

```
1 with open('out.txt', 'r') as out_file:
2     for line in out_file:
3         print(line, end = '')
```

1. Write the alternate version of this code that opens and closes the file as we learned in last class:

2. Using the `with` statement write code that creates a file named `lines.txt` and writes 100 lines like this:

```
Line #1
Line #2
Line #3
...
```

Model 3 Exceptions and how to catch them

When the file you are looking for in your code to read from does not exist or something goes wrong when reading or writing from a file, the file system (which is part of the operating system on your computer - be it Windows or MAC-OS) communicates it back to your program by triggering an exception. ***Exceptions are what make our programs crash.***
Recall that when our programs crashed in the past we interpreted it as a sign of an error (***syntactic and runtime errors***). The difference between the errors that we studied prior in the class and the exceptions triggered when working with files is that often times the later are beyond the programmer's control. ***For example, the file that the programmer is attempting to read may be corrupted***. For these cases (and these cases ONLY!), you may consider using a `try-except` structure like this:

```
try:
    out_file = open('out.txt', 'r')
    for line in out_file:
        print(line, end = '')
    out_file.close()
except FileNotFoundError:
    print('There is no file with this name')
except PermissionError:
    print('You are not allowed to access this file')
except:
    print('Something else went wrong, not sure what')
```

3. When an exception is triggered inside the `try` block, what does the program do?

The following sample file called `studentdata.txt` contains one line for each student in an imaginary class. The student's name is the first thing on each line, followed by some exam scores. The number of scores might be different for each student.

```
joe 10 15 20 30 40
bill 23 16 19 22
sue 8 22 17 14 32 17 24 21 2 9 11 17
grace 12 28 21 45 26 10
john 14 32 25 16 89
```

4. Write a function called `get_six_plus` that takes in the name of the file and returns a list of the names of students with more than six quiz scores.

5. Write a function called `print_averages` that takes in the name of the file and prints the name of the student and their average score on each line.

6. Write a function called `save_averages` that saves students' averages to a file called `averages.txt` instead of printing them.

7. Write a function called `append_stats` that saves students' minimum, maximum and average score at the end of the file, for example:

```
joe 10 15 20 30 40
bill 23 16 19 22
sue 8 22 17 14 32 17 24 21 2 9 11 17
grace 12 28 21 45 26 10
john 14 32 25 16 89

Stats:
joe [10, 40, 23.0]
bill [16, 23, 20.0]
sue [2, 32, 16.166666666666668]
grace [10, 45, 23.666666666666668]
john [14, 89, 35.2]
```

Hint: You may want to use a dictionary to save the stats for each student and then append them to the file.