CSC/CPE 101: Fundamental of Computer Science I

Fall 2019 (LAB-7)

This lab provides an additional exercise on iteration over lists. Additionally it provides an exercise on file I/O, command line arguments, and exceptions (what to do when things go wrong)!

Download the file ‘test0.txt’ from PolyLearn.

Grouping

For this part of the lab you will develop a function in groups.py, define the following function.

groups_of_3

The groups_of_3 function takes a list of values as its only argument. This function will group the elements of the input list into groups of three (i.e., the first three consecutive values form a group; the second three form a group, etc.). This function must return a list of lists where each sub-list (excluding, perhaps, the last) is a group of three values. If there are not enough values to fill the last grouping, then it will contain fewer than three values.

As examples, consider the following.

```
>>> groups_of_3([1,2,3,4,5,6,7,8,9])
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> groups_of_3([1,2,3,4,5,6,7,8])
[[1, 2, 3], [4, 5, 6], [7, 8]]
```

Write at least three tests for your function in groups_tests.py.

File Input, Command Line Arguments, try/except

Create parse_file.py in a directory file.

Command line stuff

Write a program that takes at least one, and optionally a second, command line argument. The required argument should be the name of a file to open for parsing. The optional second argument should be a flag -s. If the user provides the wrong number of arguments or if the optional argument is not '-s', then print the message 'Usage: [-s] file_name' and terminate the program using exit.

File I/O and Exceptions

Next your program should attempt to open the given input file for reading. Opening the file could fail for several reasons (e.g. the given file doesn't exist, the given file doesn't have the needed permissions, etc.) Failure to open the file could result in an exception being raised. This will cause your program to crash!

Exceptions are covered in the next course, so we will use a simple case here. In short, your code will try something. If that something works, then great ... continue on. If that something does not work,

then "catch" the exception and so something else. You can think of this as asking for forgiveness instead of asking for permission. The code outline below gives the simple structure of working with operations that may raise exceptions.

```
try:
   # what you want to attempt to do
except:
   # what to do if the previous code raises an exception
```

Should opening the file raise an exception, print the message 'Unable to open [filename]' and quit your program using exit. Replace [filename] with the name of the actual file. (Alternatively, if you know how, you may print the message in the Exception that was raised.)

Your program should open the file and read each line. The file will consist of integers, floats, and "other" strings that are not ints or floats. You should keep a count of each kind of thing read from the file and print the results when you are finished reading the entire file. For example, given the following input file:

```
abc123 34
2.34 h
3333333
2
```

Your program should output:

```
Ints: 3
Floats: 1
Other: 2
```

So what about that -s option? Should the user request that option, additionally keep track of the sum of all the integers and floats in the file and print the sum after printing the counts of each of the word 'types'.

Sample Runs

Sample run using the file test0.txt that comes with this lab. (Note: The '>' is a sample prompt.)

```
> python3 parse_file.py test0.txt
Ints: 2
Floats: 3
Other: 4
```

Sample run using the file test0.txt that comes with this lab and the -s option.

```
> python3 parse_file.py -s test0.txt
Ints: 2
Floats: 3
Other: 4
Sum: 127.17
```

Note that the -s option should work on either side of the input file.

```
> python3 parse_file.py test0.txt -s
Ints: 2
Floats: 3
Other: 4
Sum: 127.17
```

Failure for incorrect number of arguments.

```
> python3 parse_file.py
Usage: [-s] file_name
```

Failure for incorrect option.

```
> python3 parse_file.py -r test0.txt
Usage: [-s] file_name
```

Failure because the program couldn't open the file.

```
> python3 parse_file.py junk.txt
Unable to open junk.txt
```

Some Hints

How do you figure out if a 'word' in the file is an integer, a float, or something else? Some helpful hints:

- Don't forget you can split each line into words.
- A word that consists of all digits is an integer. Test for that first. Remember the string function isdigit.

For example: s = ['this', 'is', 'a', 'test', '12']
s[4].isdigit() will returns True

- Attempting to use the float function on something that *is* a float will succeed. Attempting to use it on something that isn't a float will fail (i.e. raise an Exception).

Demo and submit your files in the PlyLearn.