Problem Set VI: Args & Errors

CPE 101: Fundamentals of Computer Science
Winter 2019 - Cal Poly, San Luis Obispo

Purpose

To learn how to use command line arguments and handle exceptions.

Implementation

Write a program (including a `main` function) that takes at least one - and optionally a second - command line argument. The required argument must be the name of a file to open for reading. The optional second argument must be a flag: `-s`. If the user provides the wrong number of arguments or if the optional argument is not `-s` then print:

```
Usage: [-s] file_name
```

Your program should attempt to open a file for reading. Opening the file could fail for several reasons (e.g. the given file doesn't exist or the given file doesn't have the needed permissions). Failure to open the file could result in an exception being raised, causing your program to crash.

Exceptions are covered in detail in CPE 203, so we will use a simple case. 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 do something else. You can think of this step as asking for forgiveness instead of asking for permission. The code outlined below gives the 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 (replacing `<filename>` with the name of the actual file):

```
Unable to open <filename>
```

Your program should open a file and read each line. The file will consist of integers, floats, and "other" strings that are not integers or floats. You should keep a count of each kind of value 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:

```
int: 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 types.

Sample Runs

Use the following file and command-line arguments for your demo. In each run, the `>` symbol indicates the command run.

File Contents

```
hi 34
56.77
aef56
5.6  7.8  23 blaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaah g
```

No Options

```
> python file_parse.py test0.txt
int: 2
float: 3
other: 4
```

Using -s Option Before File

```
> python file_parse.py -s test0.txt
int: 2
float: 3
other: 4
sum: 127.17
```

Using -s Option After File

```
> python file_parse.py test0.txt -s
int: 2
```

```
float: 3
other: 4
sum: 127.17
```

Incorrect Number of Arguments Failure

```
> python file_parse.py
Usage: [-s] file_name
```

Incorrect Option Failure

```
> python file_parse.py -r test0.txt
Usage: [-s] file_name
```

Invalid File Failure

```
> python file_parse.py junk.txt
Unable to open junk.txt
```

Submission

Zip `lab6.py` and `tests.py` into one zip file and submit it to polylearn.