LAB MODULE 6

The object-oriented approach: classes, methods, objects, and the standard objective features; exception handling, and working with files

Nama	: Muchamad Lutfi Maftuh
NIM	: 19537141023
Prodi	: Teknologi Informasi

LAB 6.1.9.15 (Character frequency histogram)

Objectives

- improving the student's skills in operating with files (reading)
- using data collections for counting numerous data.

Scenario

A text file contains some text (nothing unusual) but we need to know how often (or how rare) each letter appears in the text. Such an analysis may be useful in cryptography, so we want to be able to do that in reference to the Latin alphabet.

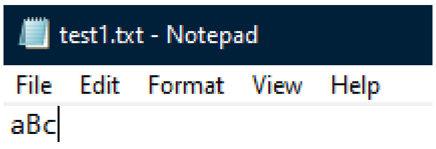
Your task is to write a program which:

- asks the user for the input file's name;
- reads the file (if possible) and counts all the Latin letters (lower- and upper-case letters are treated as equal)
- prints a simple histogram in alphabetical order (only non-zero counts should be presented)

Code Program

```python
filename = input('Enter filename: ')

try:
    file = open(filename, "rt")
except:
    print("Cannot open file.")
    exit()

content = file.read()
data = {}

for ch in content:
    if ch.isalpha():
        chlower = ch.lower()
        if chlower in data.keys():
            data[chlower] += 1
        else:
            data[chlower] = 1

for key in sorted(data.keys()):
    print(key, '->', data[key])
```

Test data



```
aBc
```

```
Enter filename: M:/test1.txt
a -> 1
b -> 1
c -> 1
```

LAB 6.1.9.16 (Sorted character frequency histogram)

Objectives

- improve the student's skills in operating with files (reading/writing)
- using lambdas to change the sort order.

Scenario

The previous code needs to be improved. It's okay, but it has to be better.

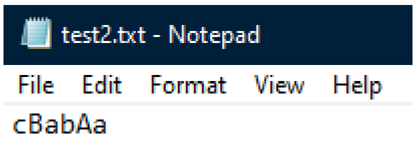
Your task is to make some amendments, which generate the following results:

- the output histogram will be sorted based on the characters' frequency (the bigger counter should be presented first)
- the histogram should be sent to a file with the same name as the input one, but with the suffix '.hist' (it should be concatenated to the original name)

Code Program

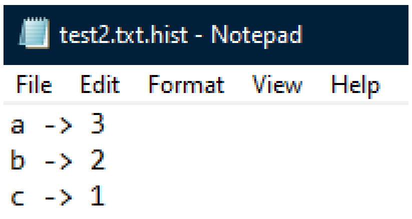
```
filename = input('Enter filename: ')

try:
    file = open(filename, "rt")
except:
    print("Cannot open file.")
    exit()

content = file.read()
data = {}

for ch in content:
    if ch.isalpha():
        chlower = ch.lower()
        if chlower in data.keys():
            data[chlower] += 1
        else:
            data[chlower] = 1

file.close()

try:
    file = open(filename + '.hist', 'w')
except:
    print("Cannot open file.")
    exit()

for key, val in sorted(data.items(), key = lambda k: k[1], reverse = 1):
    print(key, '->', val)
    file.write(key + ' -> ' + str(val) + '\n')

file.close()
```

Test data



```
cBabAa
```

```
Enter filename: M:/test2.txt
a -> 3
b -> 2
c -> 1
```



```
a -> 3
b -> 2
c -> 1
```

LAB 6.1.9.17 (Evaluating students' results)

Objectives

- improve the student's skills in operating with files (reading)
- perfecting the student's abilities in defining and using self-defined exceptions and dictionaries.

Scenario

Prof. Jekyll conducts classes with students and regularly makes notes in a text file. Each line of the file contains 3 elements: the student's first name, the student's last name, and the number of point the student received during certain classes.

The elements are separated with white spaces. Each student may appear more than once inside Prof. Jekyll's file.

Your task is to write a program which:

- asks the user for Prof. Jekyll's file name;
- reads the file contents and counts the sum of the received points for each student;
- prints a simple (but sorted) report, just like this one:

```
Andrew Cox        1.5
Anna Boleyn       15.5
John Smith        7.0
```

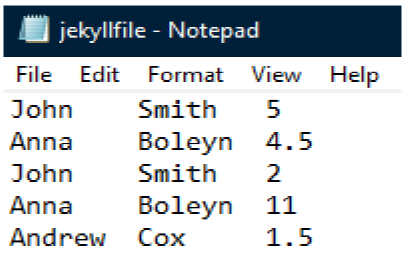
Note:

- your program must be fully protected against all possible failures: the file's non-existence, the file's emptiness, or any input data failures; encountering any data error should cause immediate program termination, and the erroneous should be presented to the user;
- implement and use your own exceptions hierarchy - we've presented it in the editor; the second exception should be raised when a bad line is detect, and the third when the source file exists but is empty.

Code Program

```python
class StudentsDataException(Exception):
    pass

class BadLine(StudentsDataException):
    def __init__(self):
        self.args = "Bad line encountered."

class FileEmpty(StudentsDataException):
    def __init__(self):
        self.args = "File is empty."

filename = input('Enter file name: ')

try:
    f = open(filename, 'r')
    content = f.read()

    data = {}

    if len(content) == 0:
        raise FileEmpty()

    for row in content.split('\n'):
        columns = [column for column in row.split('\t')]

        if len(columns) != 3:
            raise BadLine()

        fullName = columns[0] + ' ' + columns[1]
        points = columns[2]

        if fullName not in data.keys():
            data[fullName] = float(points)
        else:
            data[fullName] += float(points)

    for key, val in data.items():
        print(key, '\t', val)

except FileEmpty as fe:
    print(''.join(fe.args))
except BadLine as bl:
    print(''.join(bl.args))
```

Test data



```
John    Smith   5
Anna    Boleyn  4.5
John    Smith   2
Anna    Boleyn  11
Andrew  Cox     1.5
```

```
Enter file name: M:/jekyllfile.txt
John Smith       7.0
Anna Boleyn      15.5
Andrew Cox       1.5
```