

1 ICS 104 - Introduction to Programming in Python and C

1.1 Files and Exceptions - Lab

2 Lab Objectives

- To read and write text files
- To process collections of data
- To raise and handle exceptions

3 Worked Example

- **Problem Statement** Read two country data files, `Lab10_worldpop.txt` and `Lab10_worldarea.txt` . Both files contain the same countries in the same order. Write a file `Lab10_world_pop_density.txt` that contains country names and population densities (people per square km), with the country names aligned left and the numbers aligned right.



- **Step 1:** Understand the processing task
 - We need to read each file, line by line, and then compute the density (The countries are stored in the same order).

While there are more lines to be read

Read a line from each file.

Extract the country name.

population = number following the country name in the line from the first file

area = number following the country name in the line from the second file

If area != 0

density = population / area

Print country name and density.

- **Step 2:** Determine which files you need to read and write.
 - Input Files: Lab10_worldpop.txt and Lab10_worldarea.txt
 - Output Files: Lab10_world_pop_density.txt

- **Step 3:** Choose a mechanism for obtaining the file names.
 - Hard-coding the filenames.

```
filename = "Lab10_worldpop.txt"
```

- Asking the user for inputting the filename.

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

- Hard-coded file names are included for simplicity.
- **Step 4:** Choose between iterating over the file or reading individual lines.
- We will read individual lines using a `while` loop.
- **Step 5:** Extract the required data.
- **Step 6:** Use functions to factor out common tasks.
- Because both input files have the same format, the name of the country followed by a value, we can use a single function to extract a data record.

In []:

```
1  ##
2  #  This program reads data files of country populations and areas and prints the
3  #  population density for each country.
4  #
5
6  POPULATION_FILE = "Lab10_worldpop.txt"
7  AREA_FILE = "Lab10_worldarea.txt"
8  REPORT_FILE = "Lab10_world_pop_density.txt"
9
10 def main() :
11     # Open the files.
12     popFile = open(POPULATION_FILE, "r")
13     areaFile = open(AREA_FILE, "r")
14     reportFile = open(REPORT_FILE, "w")
15
16     # Read the first population data record.
17     popData = extractDataRecord(popFile)
18     while len(popData) == 2 :
19         # Read the next area data record.
20         areaData = extractDataRecord(areaFile)
21
22         # Extract the data components from the two lists.
23         country = popData[0]
24         population = popData[1]
25         area = areaData[1]
26
27         # Compute and print the population density.
28         density = 0.0
29         if area > 0 :    # Protect against division by zero.
30             density = population / area
31             reportFile.write("%-40s%15.2f\n" % (country, density))
32
33         # Read the next population data record.
34         popData = extractDataRecord(popFile)
35
36     # Close the files.
37     popFile.close()
38     areaFile.close()
39     reportFile.close()
40
41 ## Extracts and returns a record from an input file in which the data is
42 #  organized by line. Each line contains the name of a country (possibly
```

```
43 # containing multiple words) followed by an integer (either population
44 # or area for the given country).
45 # @param infile the input text file containing the line oriented data
46 # @return a list containing the country (string) in the first element
47 # and the population (int) or area (int) in the second element. If the end of
48 # file was reached, an empty list is returned.
49 #
50 def extractDataRecord(infile) :
51     line = infile.readline()
52     if line == "" :
53         return []
54     else :
55         parts = line.rsplit(" ", 1)
56         parts[1] = int(parts[1])
57         return parts
58
59 # Start the program.
60 main()
```

4 Reading the Entire File

- There are two methods for reading an entire file.
 - The call `inputFile.read()` returns a string with all characters in the file.
 - The `readlines` method reads the entire contents of a text file into a list:
 - `inputFile = open("sample.txt", "r")`
 - `listOfLines = inputFile.readlines()`
 - `inputFile.close()`
 - Each element in the list returned by the `readlines` method is a string containing a single line from the file (including the newline character). Only the last line does not contain the newline character.

5 Exercises

5.1 Exercise # 1:

Write a program that reads from an input file and uses the following function to look for palindrome words.

```
def ispalindrome(w):  
    return w == w[::-1]
```

NOTES:

- Palindrome word is a word which reads the same backward as forward
- ispalindrome() function returns True or False depending on passed word. You should be able to use it even if you don't know how it works.
- Consider words of length 3 and above (i.e "A" should not be checked)
- your code output:
 - Output file containing the palindrome words
 - A statement on the screen indicating how many palindrome words were found
- To check your code, copy the following to your input file:
Any good deed is even better in these days
Every motor has a rotor
Using my kayak at noon was a bad idea
- If you use the above three statements in the input file, the output would be:
 - You have 4 palindrome words
They are stored in a file named: palindrome.txt
 - Also an output file will be generated containing:
deed
rotor
kayak
noon

In [1]:

```

1  # YOUR CODE HERE
2
3  TEXT_FILE = "input.txt"
4  PALINDROME_FILE = "palindrome.txt"
5  def main():
6      counter = 0
7      textFile = open(TEXT_FILE, "r")
8      palindromeFile = open(PALINDROME_FILE, "w")
9      for line in textFile:
10         wordList = line.split()
11         for word in wordList:
12             if ispalindrome(word):
13                 if len(word) >= 3:
14                     palindromeFile.write("%s \n" %word)
15                     counter += 1
16         print("You have",counter,"palindrome words")
17         print("They are stored in a file named: palindrome.txt")
18     textFile.close()
19     palindromeFile.close()
20
21 def ispalindrome(w):
22     return w == w[::-1]
23
24 main()

```

You have 4 palindrome words

They are stored in a file named: palindrome.txt

5.2 Exercise # 2:

Write a program that evaluates and prints the result of $\frac{\sqrt{a}}{b}$

. Where a should be less than 100. Your code should report error messages using *try except* block.

NOTES:

- You should use *try* and *except* in your code
- The *ZeroDivisionError* exception will be generated automatically when trying to divide by 0
- The *ValueError* exception will take place in two cases:
 - Automatically when trying to find the square root of a negative number

- manually when $a > 100$ (using *raise* with if statement)
- The *if statement* in the previous point should be the only *if statement* in your code
- Sample runs are shown below:

Sample Run #1

```
enter the value of a 85
enter the value of b 3
The result is 3.07
```

Sample Run #2

```
enter the value of a -9
enter the value of b 9
The value of a should be non_negative and below 100
```

Sample Run #3

```
enter the value of a 104
enter the value of b 88.7
The value of a should be non_negative and below 100
```

Sample Run #4

```
enter the value of a 55.5
enter the value of b 0
The value of b can not be 0 |
```

```

In [3]: 1 # YOUR CODE HERE
        2
        3 try:
        4     from math import sqrt
        5     a = float(input("enter the value of a "))
        6     b = float(input("enter the value of b "))
        7     c = sqrt(a)/b
        8
        9     if a > 100:
       10         raise ValueError
       11     print("The result is %.2f " %c)
       12
       13 except ZeroDivisionError:
       14     print("The value of b can not be 0")
       15
       16 except ValueError:
       17     print("The value of a should be non_negative and below 100")

```

```

enter the value of a 55.5
enter the value of b 0
The value of b can not be 0

```

5.3 Exercise # 3:

Each line of a text-file scores.txt contains the ID of a student and his grades in 4 exams. Write a python program that reads grades.txt and writes to an output file output.txt the ID , the worst exam grade, the best exam grade and the average grade for each student. You have to use a dictionary with ids as key.

Also display the dictionary on the screen as shown below.

```
{'201100010': [82.5, 75.0, 95.0], '201100020': [43.75, 30.0, 50.0], '201100030': [93.0, 90.0, 97.0], '201100040': [72.5, 60.0, 80.0], '201100050': [63.0, 60.0, 70.0]}
```

```

1 contents of output file
2     ID      AVG      MIN      MAX
3  201100010   82.50   75.00   95.00
4  201100020   43.75   30.00   50.00
5  201100030   93.00   90.00   97.00
6  201100040   72.50   60.00   80.00
7  201100050   63.00   60.00   70.00

```

In [4]:

```

1  # YOUR CODE HERE
2
3  textFile = open("scores.txt", "r")
4  scores = {}
5  for line in textFile:
6      wordList = line.split()
7      counter = 0
8      for words in wordList:
9          if counter == 0:
10             wordList.pop(0)
11             scores[words] = wordList
12             counter += 1
13  outputFile = open("sortedscores.txt", "w")
14  outputFile.write("      ID      AVG      MIN      MAX\n")
15  for key in scores:
16      ID = key
17      for i in range(len(scores[key])):
18          i = float((scores[key])[0])
19          scores[key].pop(0)
20          scores[key].append(i)
21      average = sum(scores[key])/len(scores[key])
22      minimum = min(scores[key])
23      maximum = max(scores[key])
24      for i in range(len(scores[key])):
25          scores[key].pop(0)
26      scores[key].append(average)
27      scores[key].append(minimum)
28      scores[key].append(maximum)
29      average = "{:.2f}".format(average)
30      minimum = "{:.2f}".format(minimum)
31      maximum = "{:.2f}".format(maximum)
32      outputFile.write(" {0}    {1}    {2}    {3} \n".format(ID,average,minimum,maximum))
33  print(scores)
34  textFile.close()
35  outputFile.close()

```

```

{'201100010': [82.5, 75.0, 95.0], '201100020': [43.75, 30.0, 50.0], '201100030': [93.0, 90.0, 97.0], '201100040': [72.5, 60.0, 80.0], '201100050': [63.0, 60.0, 70.0]}

```

