CE156 COURSEWORK SPRING 2023

Introduction

You should refer to the Postgraduate Students' Handbook for details of the University policy regarding late submission and plagiarism; the work handed in must be entirely your own.

The assignment comprises five exercises. You should submit to FASER a single zip file containing (i) one .py file for each exercise and (ii) the output file produced by exercise 5.

Exercise 1

Write a program that asks the user to input his/her date of birth in the format mm/dd/yyyy. (Note that this exercise uses the American format, not the European one so 03/25/2023 would be 25th March.)

If the input is not in the correct format or the date is invalid the program should output an appropriate error message. If the input is a valid date the program should calculate and output the user's age. (You will need to check whether the user has had a birthday this year e.g. someone born in January 2000 will be 22 on 25th March 2022, but someone born in April will still be 21.) The program should additionally output the date in European format (e.g. 25/03/2023 for 25th March).

To calculate the age you will need to obtain today's date; to do this you need to use the line

```
from datetime import date
```

and then use `date.today()` to obtain a `date` object containing today's date. You can access the day, month and year of a date `d` using `d.day`, `d.month` and `d.year` (these are all integer values).

Exercise 2

Write a function that returns a list of all non-prime numbers between two positive integers supplied as arguments. Use this is in a program that asks the user to supply two positive integers, checks that the input is valid, then calls the function and outputs the numbers in the returned list, with 10 numbers per output line.

If the user enters negative numbers or supplies non-numeric input the program should output an appropriate error message; the two numbers should be accepted in either order. The range should be inclusive; if the user inputs 100 and 351 (or 351 and 100) these two numbers (which are both non-prime) should be included in the output.

Exercise 3

The three functions for this exercise should be written in a a single .py file. Yhe submitted file should contain these 3 functions together with an auxiliary functions you might use, but should not submit any code that tests the functions. It is of course recommended that you do produce such code to test your functions, but this should not be submitted.

a) Write a function that takes a string as a parameter and returns True if and only if the string is a palindrome (and returns False otherwise).

b) Write a function that takes a string as a parameter, converts the string to upper-case and returns the most frequent letter/digit. (If there are equally frequent letters and digits you may return any one of the most frequent letters/digits.) Characters that are neither letters nor digits should be ignored.

c) Write a function that takes a string as a parameter, counts the number of letters, spaces and digits in the string and returns a `dict` object containing the three counts as values.

Exercise 4

Write a function that takes 3 arguments, a list of tuples and two integers denoting a salaries. Each tuple in the list will contain the name, job title and salary of an employee. The function should output the names and job titles of all employees in the list whose salary is greater than or equal to the smaller of the integer arguments and less than or equal to the larger. The output should be displayed one employee per line, sorted by salary (largest first), in a neatly formatted table. If there are no matches an appropriate message should be output.

Write a program that asks the user to supply a file name and attempts to open the file for reading. (If the file could not be opened the user should be asked to supply another filename; this process should continue until a file has been opened successfully.) Each line of the file should contain the name, job title and salary of a single employee (in that order, separated by commas). A typical line may be

```
Gareth Southgate,manager,2500000
```

The program should create an empty list of tuples and then convert the contents of each line of the file into a tuple with 3 elements, which should be added to the list. (You may assume that the contents of the file are in the correct format so there is no need to perform validation).

After the conversion of the file contents to tuples has been completed the program should print the list of tuples (this is simply to verify that the tuples have been created correctly so no formatting is required) then enter a loop in which the user is asked to supply a salary range. The user input and list of tuples should be passed as arguments in a call to the function described above. After returning from the function the user should be asked if he/she wishes to quit or to supply another salary range.

Hint: the elements of the tuple do not have to be stored in the order described above. A different order may make the sorting easier.

Exercise 5

The aim of this exercise is to process a file containing marks for a university module. The input file will have the format

```
5 30
1991111 65.5 45.5
2031234 75.5 68
```

```
2012345 33 50
2019734 55.5 51.5
2187345 39 47.5
```

The first line of the file contains the number of students and the percentage coursework weighting, e.g. the 30 in this example denotes that the coursework comprises 30% of the overall module mark (with the exam making up the other 70%).

Each subsequent line contains (in order) a registration number, an exam mark and a coursework mark. The number of such lines will match the number of students.

Write a program that will open such a file (whose name should be specified by the user), read the first line and create a 2-dimensional NumPy array where number of rows is the number of students and each row has 4 elements. (Initialise this array using a call to the `array` function with an argument `[[0,0.0,0.0,0.0]]*n`, where `n` is the number of students.)

The program should then read the remaining lines line-by-line and use the input values to store in a row of the array the registration number, the exam mark, the coursework mark and the overall mark (calculated using the exam mark, the coursework mark and the weighting). You may assume that all lines will contain the correct number of marks and the number of students will match the number specified on the first line. The marks in this array are real numbers, hence the use of 0.0 above.

Having generated the array, you should define a named data type (similar to `studType` on slide 46 of part 8 of the lectures slides) whose fields are 4 integers and one string. The program should then create a 1-dimensional array with `n` elements, using this named type as a `dtype` argument in the call to `np.array`. Use a list containing multiple copies of a tuple with default values (e.g. 4 zeroes and an empty string) as the first argument.

From the data in each row of the first array the program should generate a tuple containing the student's registration number, exam mark, coursework mark and overall mark, all rounded to the nearest integer, and a grade string to be calculated using the rules below and store this in the corresponding element in the second array. (You can round a real number x to the nearest integer using `round(x)`.)

The program should then produce a version of the second array sorted by overall mark and output this array to a file (using a single call to the `print` function of the form `print(array2, file=f)`).

It should finally output to the screen/console the number of students who have first, second and third-class marks, the number of students who have failed and the registration numbers of the students who have failed.

You should submit to FASER the program file, and also the output file that your program produced from an input file that will be provided in week 24.

During development and testing it is recommended that you use a file with contents similar to those at the top of the previous page.

Rules for calculating grades

Any student with a rounded mark of less than 30 for either the coursework or the exam has failed, irrespective of the overall module mark. For all other students a rounded overall mark of 70 or more has a first-class grade, a rounded mark between 50 and 69 (inclusive) has a second-class grade, a rounded mark between 40 and 49 has a third-class grade, and a rounded mark below 40 fails.