Koç University COMP125-01/02/03
Programming with Python: Midterm 5
Instructors: Mehmet Emre Gürsoy, Mehmet Sayar, Barış Akgün
Date: December 27, 2020

Start: 6:30pm
Deadline to submit on Blackboard: 7:45pm

Instructions

Make sure you read and understand every part of this document

Download & Extract the zip to your Desktop:

Blackboard -> Assessments -> Midterm 5 -> MT5.zip

Open in Spyder:

ExceptionLog.py FuncArguments.py DailyTemperatures.py

- Do not change the names of the files. Do not change the names of the functions that are given to you.
- Do not use external libraries or modules other than math and random.

When you are finished:

Submit a single compressed archive (zip file) containing:

ExceptionLog.py FuncArguments.py DailyTemperatures.py

Multiple attempts are allowed, so upload ASAP, clean up later.
We will grade only the LAST version submitted to Blackboard.

By submitting this midterm, you certify that you have already completed the Honor Pledge essay given on Blackboard. If you have not done so, we reserve the right to dismiss your midterm.

"You certify that you have completed this exam on your own without any help from anyone else. You understand that the only sources of authorized information in this exam are (i) the course textbook, (ii) the material that is posted at Blackboard for this class, and (iii) any study notes handwritten by yourself. You have not used, accessed or received any information from any other unauthorized source in taking this exam. The effort in the exam belongs completely to you."

Communication before & during exam:

Zoom session (Meeting ID: 922 5796 1536 Passcode: 129757)

Starts at 18:15 pm
Students are not allowed to speak or chat via this channel once the exam starts.
Violations may be punished via disciplinary action !!!

For clarification questions:

Raise your hand in Zoom.
We will transfer you to a breakout room.
Only clarification questions regarding the midterm are allowed.
Installation related or my_code_is_not_working questions will not be answered.

Q1: Exception Log - 35 pts

In software systems, it is common to maintain a “log file” that records events, errors or exceptions that occur while the software is running. The log file is periodically reviewed by system administrators to check for abnormal behavior, unusual events, cyberattacks, etc.

In this question, you will implement code to maintain an “exception log file” to record exceptions that occur in a simple Python function.

Open ExceptionLog.py and find the function divide_elementwise(a, b). This function:

- Takes as input two matrices a and b (in list of lists representation)
- Performs element-wise division: divide a[i][j] by b[i][j]
- A caveat: a and b originally contain string elements, therefore the strings are converted to floats by divide_elementwise before division is performed.

You may assume a and b are square matrices (number of rows = number of columns).

There are multiple exceptions that may occur in divide_elementwise. You should modify the divide_elementwise function that is given to you so that these exceptions are handled. Furthermore, you should ensure that the appropriate exception log messages are appended to exceptionlog.txt file.

Here are the potential exceptions and how you should handle them:

- A string element cannot be converted to float since it contains text (not a number).
    - Determine what exception/error will be raised.
    - Catch the exception corresponding to this particular problem.
    - When the exception is caught, you should append a one-line message to exceptionlog.txt: “Exception! Cannot convert string to float.”
- A division-by-zero occurs.
    - Determine what exception/error will be raised.
    - Catch the exception corresponding to this particular problem.
    - When the exception is caught, you should append a one-line message to exceptionlog.txt: “Exception! Cannot divide by zero.”
- The sizes of a and b are different, e.g., a is NxN matrix but b is MxM where **M ≠ N**:
    - Explicitly check for this condition at the beginning of the function (before division is performed) and raise an exception.
    - Catch the exception corresponding to this problem.
    - When the exception is caught, you should append a one-line message to exceptionlog.txt: “Exception! Matrix sizes are different.”

If divide_elementwise runs without any exceptions, it should append a one-line message to exceptionlog.txt: “divide_elementwise() ran without any exceptions.”

After every execution of divide_elementwise, regardless of whether an exception occurred or not, it should always append a one-line message to exceptionlog.txt: “Exiting divide_elementwise() now.”

Rewrite divide_elementwise by keeping the original functionality but using a try-except structure to achieve the above exception handling behavior.

IMPORTANT: You must handle all exceptions within divide_elementwise. Do not handle exceptions within the main() function.

IMPORTANT: In all cases, you must APPEND to exceptionlog.txt. Never overwrite existing information in exceptionlog.txt.

Q2: Functions and Arguments - 15 pts

Open FuncArguments.py and implement the following functions. You may need to edit both the function header and the function body.

func1

Implement a function called func1 such that:

- func1 has 3 inputs: a, b, c (assume all of them are integers)
- func1 has 2 outputs:
  - First output is a+b+c
  - Second output is a-b-c
- You must implement func1 in 4 lines of code or less (including function definition line)

Sample executions of func1:

```
res1, res2 = func1(10,2,5)
print(res1)
print(res2)
res3 = func1(7, 2, -1)
print(res3)
```

-->

```
17
3
(8, 6)
```

func2

Implement a function called func2 such that:

- func2 has 2 inputs: name, message
  - When calling func2, name is mandatory
  - When calling func2, message is optional. If no message is provided in the function call, func2 assumes message = “Hello”
- func2 has 1 output: a greeting message constructed as in the following examples.

```
res = func2("John")
print(res)
res = func2("Joe", "Goodbye")
print(res)
res = func2("Alice", "How are you")
print(res)
```

-->

```
Hello, John
Goodbye, Joe
How are you, Alice
```

func3

Implement a function called func3 such that:

- func3 has 2 inputs: a list and a threshold
    - Both are mandatory when calling func3
- func3 has 1 output: it returns the average of the elements in the list whose values are greater than the threshold

For example, when the list is = [0, 10, 20, 30, 40] and threshold is 15, func3 returns the average of 20, 30, 40; which is 30. When the list is = [100, 0, 55, 70, 30] and threshold is 60, func3 returns the average of 100 and 70; which is 85.

```
lst = [0, 10, 20, 30, 40]
th = 15
res = func3(inp_list = lst, threshold = th)
print(res)
lst = [100, 0, 55, 70, 30]
th = 60
res = func3(threshold = th, inp_list = lst)
print(res)
```

-->

```
30.0
85.0
```

IMPORTANT: In all three functions, the outputs should be **RETURNED**. They should not be printed to the console. In the above examples, printing is performed in the main() function; see FuncArguments.py.

IMPORTANT: In this question (all three functions), you do not need to do any error checking or exception handling.

Q3: Daily Temperatures - 50 pts

You are given two data files: ISTANBUL.txt and ANKARA.txt, which contain daily temperature readings from the two cities. The data contains 4 columns: first column is month, second column is day, third column is year, fourth column is average daily temperature (in Fahrenheit). Note that there are some entries with value “-99” in the temperature column; this is because the temperature reading for that day is missing.

1	1	1995	55.0
1	2	1995	57.1
1	3	1995	44.4
1	4	1995	47.6
1	5	1995	44.1
1	6	1995	45.2
1	7	1995	49.5
1	8	1995	42.5
1	9	1995	43.6
1	10	1995	41.0

Open DailyTemperatures.py and solve the following parts.

Part A: Write a function called read_temperatures(filename) to read the data from the given text file into a matrix (stored as a list of lists).

- filename is the name of the file that should be read, e.g., “ISTANBUL.txt”
- The return value should be a matrix in list of lists representation such that:
  - Each row in the matrix is one row of the data file
  - First column is month, second column is day, third column is year, fourth column is average daily temperature (same column order as the file)
- Convert day, month, year to int. Convert temperature to float.

Partial output when the matrix is printed to console (full matrix is quite large):

```
[5, 4, 2020, 53.1], [5, 5, 2020, 52.1], [5, 6, 2020, 55.2], [5, 7,
2020, 50.1], [5, 8, 2020, 49.0], [5, 9, 2020, 50.9], [5, 10, 2020,
54.5], [5, 11, 2020, 59.6], [5, 12, 2020, 67.1], [5, 13, 2020,
53.6]]
```

Part B: Write a function called delete_year(data, year) to remove data from a given year.

- data is the matrix from Part A
- The return value should contain all rows from data minus those rows that have year equal to the given year

For example, if year = 2020, then the return value of delete_year contains all rows other than those rows with year equal to 2020.

Part C: Write a function called monthly_averages(data, year) to calculate month-by-month average temperatures for the given year.

- data is the matrix from Part B (or Part A)
- year is an integer, e.g.: year = 2004 or year = 2019
- The return value should be a list of length 12:
  [avg_of_January, avg_of_February, avg_of_March, …, avg_of_December]
- When calculating the monthly averages, only calculate the monthly averages for the given year. Do not calculate a multi-year average.
- Make sure you account for missing data (“-99”). If 2 days out of 30 are missing for a certain month, then calculate the average across the remaining 28 days.
  - Do not take “-99”s into consideration. Do not treat “-99” as 0 Fahrenheit.

Sample output when the return value is printed to console:

```
[25.22258064516129, 33.49285714285714, 43.719354838709684,
48.89999999999999, 57.222580645161294, 65.33, 74.31935483870967,
73.63548387096773, 69.21, 51.164516129032265, 41.01666666666665,
36.98064516129033]
```

Part D: Write a function called yearly_average(data) to calculate the year-by-year average temperatures for each year between 1995-2019.

- data is the matrix from Part B (or Part A)
- Assume years are between 1995-2019 (you can write your loop accordingly)
- Do not forget to handle the missing values (“-99”) correctly!
- There is no return value. The function should write to a file called “out.txt”:
  - year,avg_temp_in_year
  - Ensure there are 2 digits after the decimal point when writing temperatures

```
1995,50.13
1996,50.58
1997,48.54
1998,50.38
1999,51.14
2000,49.05
2001,52.77
2002,48.34
2003,51.42
2004,50.25
2005,50.87
```

Submission

Solve each question in its own file. **Do not change the names of the files. Do not change the names of the functions (you may have to change function parameters in Q2).**

Do not use external libraries or modules other than math and random.

When you are finished, compress your MT5 folder containing your answers:
ExceptionLog.py FuncArguments.py DailyTemperatures.py

The result should be a SINGLE compressed file (file extension: .zip, .rar, .tar, .tar.gz, or .7z). Upload this compressed file to Blackboard.

You may receive 0 if one or more of the following is true:

- You do not follow the submission instructions, i.e., you submit something other than a single compressed file.
- Your compressed file does not contain your Python files or the file names are wrong.
- Your compressed file is corrupted.
    - **After you submit, you should download your submission from Blackboard to make sure it is not corrupted and it has the latest version of your code.**
- Your code contains syntax errors.
- You use external libraries or modules other than math and random.
- Your code does not run without errors the way that it is submitted.
    - **We should not have to open your file and comment/uncomment parts of it in order to run your code.**
- Your code does not terminate, e.g.: it contains infinite loops.