C200 Programming Exam Fall 2023

Dr. M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

October 12, 2023

Introduction

This is the programming part of the exam and the deadline is October 13, Friday, 11 PM. There is **no Autograder** for this so you will push the files to github.iu.edu. You are to complete this by yourself using only,

- The official Python documentation https://www.python.org/.
- Lecture slides from any section.
- Personal and class notes.

Instructions

- Pull the latest changes to your repo, you will see that we have created a folder named midterm in your repository.
- The folder contains the starter code in **midterm.py**.
- Read the instructions for problems (starting next page) and implement the programs in midterm.py file.
- Remember to commit and push your work to the GitHub repository by the deadline.
- If your answer includes any elements that we have not covered in lectures/labs, it will be considered cheating–period; there are, however, many solutions. We will check only for correctness. Thus, there isn't any partial credit for the programming.

Note: If you don't see the midterm folder or midterm.py in your repo then you can either contact us via InScribe. Alternatively, you can create a folder named 'Midterm' in your repo, create a new file named 'midterm.py' inside the Midterm folder and paste the starter code (given at the end of this document) in midterm.py and start working in that file.

Program

Please read the problem carefully. We are not including unit testing–you should make some test cases yourself using print. Lastly, write all the print functions below main.

Problem 1: Plasma Energy Concentration

You're working as a consultant developing software for a health center ALL RIGHT. The most immediate problem is to determe the concentration of a drug after some time elapses, because another injection might be required. The drug concentration after t hours is given by:

$$C(t) = C_0 e^{-rt} \quad (1)$$

where C_0 is the initial concentration and some drug-related r value. Your task is to complete two programs–the second program requires the first. The first function uses Eq. (1) to determine the time that the drug concentration is reduced by some percent. For example, given a percent decrease percent_decrease and r-value r, what is the time (in hours) for this drop to occur. If a patient requires 50% reduction and has an r-value of 0.2, then the time 3.466 hours will elapse before this occurs. We'll round to three decimal places. Because fractions of hours are hard for people to use, ALL RIGHT wants this value to be converted into hours and minutes, obviously using the decimal portion for minutes. This function convert_HM(hours) takes hours as a float and returns a tuple (x,y) where x are the hours and y the minutes where y is an integer. For example, 3.466 is converted into (3,27). While this isn't exact, we can use the math module to determine if .1 tolerance is sufficient:

```
>>> import math
>>> math.isclose(3.466,3 + 27/60, abs_tol = .1)
True
```

Note that $27/60$ is approximately 0.45, therefore 3 + 0.45 is 3.45 which is approximately equal to 3.466, if our tolerance is 0.1. You already have the hours (before the decimal) and need to find the minutes by converting the numbers after the decimal.

The following code:

```
if __name__ == "__main__":

    #Problem 1
    data = [[50, 0.2],[35, 0.44],[20,.01]] #percent decrease, r-value

    for d in data:
        hours = next_injection(*d)
        h_, m_ = convert_HM(hours)
        print(hours, h_,m_ , math.isclose(hours,h_ + m_/60,abs_tol = ↩
            .1))
```

produces, when correct,

```
3.466 3 27 True
2.386 2 23 True
160.944 160 56 True
```

Requirements

1. Round the float hours to three places.
2. You will have to type cast the minutes–it is up to you how to do this correctly to match the output
3. You cannot use any module other than math

Problem 2: Clambake Technique for Lists

In this problem the function m(x,lst) takes a number and a possibly empty list of numbers and returns the maximal portion of the list (starting from the 0 position) that sums to less than or equal to x. The function returns a three-tuple (x,y,z) where x is the number sent to the function, y is the sum of the maximal part of the list that is less than x, and y is that list. For example,

```
  data = [[0,[0,0,1]],[1,[0,0,0]],[2,[1,0,1,0,2]],
                [1,[2,1,0]],[4,[1,0,2,0,.5,.1,0,5]],
                [5,[5,5]], [2,[-1,2,0,1,-1]],[1,[]]]
    for d in data:
        print(m(*d))
```

produces:

```
(0, 0, [0, 0])
(1, 0, [0, 0, 0])
(2, 2, [1, 0, 1, 0])
(1, 0, [])
(4, 3.6, [1, 0, 2, 0, 0.5, 0.1, 0])
(5, 5, [5])
(2, 1, [-1, 2, 0, 1, -1])
(1, 0, [])
```

For example m([2,[1,0,1,0,2]]) returns (2,2,[1,0,1,0]) since that's the largest list starting from index 0 that sums to a value less than or equal to 2. For m([1, [2,1,0]]) returns (1, 0, []) since we can't get a list with sum less than or equal to 1 hence the returned list is empty. For m([1,[0,0,0]]) the tuple is (1,0,[0,0,0]). **Requirements**

1. No looping other than what we've discussed in lecture is allowed.
2. You can use sum() or create your own additional functions that must be locally defined.
3. You cannot use any module other than math

The entire starter code is shown here for convenience:

```python
import math

# Problem 1
#input percent_decrease as positive integer, r as float
#output hours as float
def next_injection(percent_decrease,r):
    pass

#input hours as float
#output tuple (x,y) where x hours, y minutes
def convert_HM(hours):
    pass

# Problem 2
#input x number and possibly empty list of numbers
#output number, sum of list, list where sum list is <= x
def m(x,lst):
    pass


if __name__ == "__main__":

    ## Problem 1
    data = [[50, 0.2],[35, 0.44],[20,.01]]

    for d in data:
        hours = next_injection(*d)
        h_, m_ = convert_HM(hours)
        print(hours, h_,m_ , math.isclose(hours,h_ + m_/60,abs_tol = ↩
            .1))

    ## Problem 2
    data = [[0,[0,0,1]],[1,[0,0,0]],[2,[1,0,1,0,2]],
                  [1,[2,1,0]],[4,[1,0,2,0,.5,.1,0,5]],
                  [5,[5,5]],[2,[-1,2,0,1,-1]],[1,[]]]

    for d in data:
        print(m(*d))
```