Homework 2

Intro to Programming (Term III/2024–25)

built on 2025/04/30 at 14:25:37 ***due:*** *wed May 7 @ 11:59pm*

This assignment will give you more practice on expressions, pretty printing, and strings. In this assignment, you will solve a number of programming puzzles and hand them in.

Be sure to read this problem set thoroughly, especially the sections related to collaboration and the hand-in procedure.

Overview:

Problem	File Name
1.	`mutation.py`
2.	`posneg.py`
3.	`kshift.py`
4.	`minmax.py`

Problem	File Name
5.	`med4.py`
6.	`kitten.py`
7.	`righttri.py`

Collaboration

We interpret collaboration very liberally. You may work with other students. However, each student ***must*** write up and hand in his or her assignment separately. Let us repeat: You need to write your own code. You must not look at or copy someone else's code. You need to write up answers to written problems individually. The fact that you can recreate the solution from memory will be taken as proof that you actually understood it, and you may actually be interviewed about your answers.

Be sure to indicate who you have worked with (refer to the hand-in instructions).

Logistics

We're using a script to grade your submission before any human being looks at it. Sadly, the script is not as forgiving as we are. *So, make sure you follow the instructions strictly.* It's a bad omen when the course staff has to manually recover your file because the script doesn't like it. Hence:

- Save your work in a file as described in the task description. This will be different for each task. **Do not save your file(s) with names other than specified.**
- Before handing anything in, you should thoroughly test everything you write.
- You will upload each file to our submission site `https://assn.cs.muzoo.io/` before the due date. Please use your SKY credentials to log into the submission site. Note that you can submit multiple times but only the latest version will be graded.
- For some task, you will be able to verify your submission online. Please do so as it checks if your solution is gradable or not. Passing verification does not mean that your solution is correct, but, at least, it passes our preliminary check.
- At the beginning of each of your solution files, write down the number of hours (roughly) you spent on that particular task, and the names of the people you collaborated with as comments. As an example, each of your files should look like this:

```
# Assignment XX, Task YY
# Name: Eye Loveprograming
# Collaborators: John Nonexistent
# Time Spent: 4:00 hrs

... your real program continues here ...
```

- The course staff is here to help. We'll steer you toward solutions. Catch us in real-life or online on Canvas discussion.

Task 1: Mutated Strings (10 points)

For this task, save your work in `mutation.py`

Write a program that reads two strings `s:str` and `t:str` from the console respectively, applies the following mutation steps to the strings, in this specific order:

Step 1: Convert `s` to lowercase and convert `t` to uppercase.
Step 2: In `s`, change each occurrence of `'s'`, `'l'`, and `'a'` into `'m'`.
Step 3: In `t`, change each occurrence of `'P'`,`'O'`, `'I'`, and `'N'` into `'T'`.
Step 4: Swap the first character of `s` and the first character of `t`.
Step 5: Replace the last 1/3 of `s` with the middle 1/3 of `t`.
Step 6: Form `z` by joining `s` with `t`.

Finally, **print out the joined string** `z`.

The first two lines of your program should look like this.

```
s = input()
t = input()
```

Consider the following sample input.

```
HelloWorld!!
ThisIsPython
```

For this example, `s = 'HelloWorld!!'` and `t = 'ThisIsPython'`

1. After applying step 1, `s = 'helloworld!!'` and `t = 'THISISPYTHON'`.
2. After applying steps 2&3, `s = 'hemmowormd!!'` and `t = 'THTSTSTYTHTT'`.
3. After applying step 4, `s = 'Temmowormd!!'` and `t = 'hHTSTSTYTHTT'`.
4. After applying step 5, `s = 'TemmoworTSTY'` and `t = 'hHTSTSTYTHTT'`. Note that prior to the replacement, the last 1/3 of `s` is `md!!` and the middle 1/3 of `t` is `TSTY`.
5. After step 6, the joined string is `z = 'TemmoworTSTYhHTSTSTYTHTT'`

The expected output of your program is the following.

```
TemmoworTSTYhHTSTSTYTHTT
```

For this problem, you may assume that the lengths of `s` and `t` are always multiples of 3.

Task 2: Positive, Negative (10 points)

For this task, save your work in `posneg.py`

Ground rules for this task:

- You can only use Boolean/arithmetic operators and numerical comparisons.
- You **cannot** use any conditional (i.e., **if**) statements.

Write a program that reads three input values from the console.

`a: int`, `b: int`, and `negative: bool`

Please use the following starter code to read the values.

```
a: int = int(input())
b: int = int(input())
negative:bool = input().lower() == 'true'
```

The input values will be given in this specific order. Here is a sample input.

```
12
-5
True
```

For the above input, `a=12`, `b=-5`, and `negative=True`

As for the output, your program must print out either `True` or `False` according to the following rules. Print `True` if `a` or `b` is negative and the other one is positive. Except if `negative` is True, then print `True` only if both are negative. For the purpose of this problem, the number 0 is neither negative nor positive.

Examples:

- `a=1, b=-1, negative=False`, your program should print `True`.
- `a=-1, b=1, negative=False`, your program should print `True`.
- `a=-4, b=-5, negative=True`, your program should print `True`.
- `a=4, b=-5, negative=True`, your program should print `False`.

Task 3: Shift By k (10 points)

For this task, save your work in `kshift.py`

Ground rules for this task:

- You can only use Boolean/arithmetic operators and string functions/operations (including slicing, i.e., various forms of `[a:b]`).
- You **cannot** use any conditional (i.e., **if**) statements or looping, nor could you write a function.

When a string shifts, the last symbol of the string comes to the front, pushing the other symbols forward by one position. For example, shifting `'hello'` gives `'ohell'` (the `o` at the end has come to the front.)

Moreover, when a string shifts k times, the aforementioned process is applied k times. For instance, after shifting 4 times, the string `'hello'` becomes `'elloh'`. Notice that for a string of length n, shifting n times yields the original string.

Your program reads two input values from the console:

- `st: str` — the starting string
- `k: int`, `k` ≥ 0 — how many times it will be shifted by.

Here is an example input.

```
meogalacitca
3
```

In this example, `st='meogalacitca', k=3`.

Your program must print out the result obtained by shifting `st` a total of `k` times. Keep in mind that `k` may be even larger than the length of `st`, in which case the modulo operator might be useful.

Examples:

- For input `st='meogalacitca', k=3`, your program prints out `'tcameogalaci'`.
- For input `st='meogalacitca', k=7`, your program prints out `'lacitcameoga'`.
- For input `st='meogalacitca', k=12`, your program prints out `'meogalacitca'`.
- For input `st='meogalacitca', k=2020`, your program prints out `'itcameogalac'`.

Task 4: Min/Max (10 points)

For this task, save your work in `minmax.py`

Ground rules for this task:

- You can only use Boolean/arithmetic operators, the `int` function, the `abs` function.
- You **cannot** use any conditional (i.e., **if**) statements, nor the built-in `min`, `max` functions.

Your goal in this task is to compute the minimum and the maximum between two numbers, without using the built-in **min**/**max** or comparisons (how oddly cool!).

Your program reads following two input values from the console.

`a: int` and `b: int`.

Here is an example input.

```
72
43
```

In this example, `a=72, b=43`.

Your program must print out two values on the same line. The first value must be the value of the smaller of `a` and `b`; and the second value must be the larger of `a` and `b`.

The expected output of the sample input is:

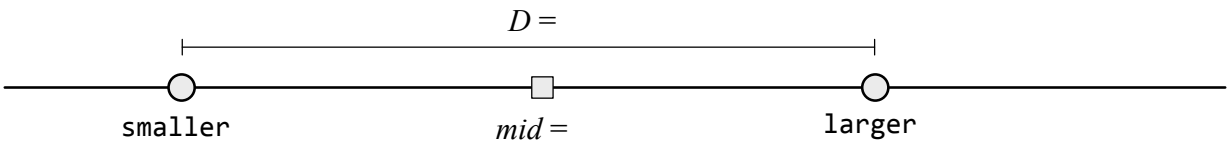
```
43 72
```

It is important to remember that your code has to *strictly adhere* to the ground rules for the problem.

Examples:

- `a = 4, b = 2` gives the output of
  ```
  2 4
  ```
- `a = 2, b = 4` gives the output of
  ```
  2 4
  ```
- `a = 3, b = 8` gives the output of
  ```
  3 8
  ```
- `a = 11, b = 6` gives the output of
  ```
  6 11
  ```

Guidelines: To begin thinking about this problem, consider the diagram and questions below:



- We define D to be the size of the gap between `smaller` and `larger`. Quite curiously, it is possible to write D in terms of `a` and `b`. How can we write D in terms of `a` and `b`?
- Now *mid* is defined to be the midpoint between `smaller` and `larger`. How can we write *mid* in terms of `a` and `b`?
- Do we know how far `smaller` is to *mid*?

The answers to these questions should guide you toward a working program. Finally, the following hint might be useful: (*Hint:* $\frac{a}{2} + \frac{b}{2} = \frac{a+b}{2}$.)

Task 5: Median of Four (10 points)

For this task, save your work in `med4.py`

Ground rules for the problem:

- The only things you can use are the built-in **min** and **max**, as well as arithmetic/math operators.
- You are **not** permitted to use the built-in sorting functions, nor write a loop or a function, nor a list and its operations.

In statistics, the *median* is the middle number in a sorted list of numbers. For four numbers, the median is taken to be the average of two numbers in the middle of the sorted list. For example, the data set $3, 5, 7, 12$ (which is already sorted) has a median value of $\frac{5+7}{2} = 6$.

In this problem, you will compute the median of 4 numbers, without being able to sort. Your program will read the following 4 input values from the console.

```
p: int = int(input())
q: int = int(input())
r: int = int(input())
s: int = int(input())
```

Here is an example input.

```
28
13
4
10
```

In this example, `p=28, q=13, r=4, s=10`.

Your task is to write a program that prints out sets the median value of these 4 numbers while obeying the ground rules.

Examples:

- For input numbers $7, 5, 3, 12$, the answer is:

  ```
  6.0
  ```

- For input numbers $2, 1, 5, 11$, the answer is:

  ```
  3.5
  ```

- For input numbers $99, 8, 0, 24$, the answer is:

  ```
  16.0
  ```

(*Hint:* Subtract, i.e., `-`, to remove unwanted numbers from a sum.)

Task 6: In Trouble? (10 points)

For this task, save your work in `kitten.py`

Ground rules for this task:

- You can only use Boolean operators and numerical comparisons.
- You **cannot** use any conditional (i.e., **if**) statements.

As the story has it, Dr. Sunsern has a super loud meow-ing kitten. He often gets in trouble because of her. Your program will read three values from the console.

- `hour: int` — the current hour in the range between 0 and 23 (inclusive).
- `minute: int` – the current minute in the range between 0 and 59 (inclusive).
- `meowing: bool` — whether the kitten is meowing at this time.

Please use the following starter code to read the values.

```
hour: int = int(input())
minute: int = int(input())
meowing:bool = input().lower() == 'true'
```

He is in trouble if the kitten meows before 6:30 or after 21:00 (meowing at exactly 6:30 and 21:00 are fine). Your program is to print out `True` if he is in trouble because of the kitten or `False` otherwise.

Examples:

- With `hour = 11, minute = 19, meowing = True`, the output should be:
  ```
  False
  ```
  Meow-ing at 11:19 is outside the trouble-causing range.
- With `hour = 23, minute = 7, meowing = True`, the output should be:
  ```
  True
  ```
  Meow-ing at 23:07 is after 21, so this will cause trouble.
- With `hour = 4, minute = 55, meowing = False`, the output should be:
  ```
  False
  ```
  Although 4:55 is in the trouble-causing range, the kitten doesn't meow at this point, so no trouble is caused.
- With `hour = 6, minute = 15, meowing = True`, the output should be:
  ```
  True
  ```
  Meow-ing at 6:15 is before 6:30, so this will cause trouble.

Task 7: Right Triangle? (10 points)

For this task, save your work in `righttri.py`

Ground rules for this task:

- You can only use Boolean operators, numerical comparisons, and the `abs` function.
- You **cannot** use any conditional (i.e., **if**) statements.

A *right triangle* is a triangle in which one angle measures 90 degrees. There is a simple test to figure out whether a triangle is a right triangle:

1. Let c be the length of the longest side, and a and b be the lengths of the other two sides.
2. If $c^2 = a^2 + b^2$, we know that the triangle is a right triangle; otherwise, it is not.

For this problem, three parameters—`x: float, y: float, z: float`—have been set by the grader before your program begins. They are the lengths of the three sides of a triangle, in no particular order.

Please use the following starter code to read the values.

```
x: float = float(input())
y: float = float(input())
z: float = float(input())
```

Your task is to write a program that implements the test as described above so that when the program is finished, it must print out **`True`** or **`False`** to correctly indicates whether the side lengths, as given, make up a right triangle.

Floating-Point Warning: Because of how computers represent floating-point numbers, it is impossible to test whether two floating numbers have the exact same numerical value. Hence, when `r` and `s` are floating-point numbers, writing `s == r` to test if they have the same numerical value is generally meaningless. A much better way to test for equality is to use the test

```
abs(s - r) < T
```

where `T` is a "threshold" number, below which we deem two numbers identical. For this problem, use $T = 10^{-7}$. In Python, we can write 10^{-7} simply as `1e-7`, hence writing `abs(s - r) < 1e-7`.

Examples:

- `x=4.0, y=5.0, z=3.0` gives

```
True
```

- `x=11.5, y=5.0, z=3.0` gives

```
False
```