Homework 2

Intro to Programming (Term I/2021–22)

built on 2021/09/22 at 14:35:15 ***due:*** *fri sep 29 @ 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 given two strings `s:str` and `t:str`, 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`.

For example, assume `s = 'HelloWorld!!'` and `t = 'ThisIsPython'`. Let's apply the mutations.

1. After step 1, `s = 'helloworld!!'` and `t = 'THISISPYTHON'`.
2. After steps 2&3, `s = 'hemmowormd!!'` and `t = 'THTSTSTYTHTT'`.
3. After step 4, `s = 'Temmowormd!!'` and `t = 'hHTSTSTYTHTT'`.
4. After 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'`

For this problem, assume that the lengths of `s` and `t` are multiples of 3. The grader will define `s` and `t` for you before your program runs. During development, **you may define your own `s` and `t`. However, remove them before submission**.

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.

Before your program begins, the grader has set the following variables:

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

Your program will set a variable `pos_neg` according to the following rules: Set it to True if one is negative and one is positive. Except if `negative` is True, then set `pos_neg` to `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` gives `pos_neg = True`.
- `a=-1, b=1, negative=False` gives `pos_neg = True`.
- `a=-4, b=-5, negative=True` gives `pos_neg = True`.
- `a=4, b=-5, negative=True` gives `pos_neg = 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.

Before your program begins, the grader has set the following variables:

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

Your program will set a variable `shifted_st` to 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`, we have `'tcameogalaci'`.
- For input `st='meogalacitca', k=7`, we have `'lacitcameoga'`.
- For input `st='meogalacitca', k=12`, we have `'meogalacitca'`.
- For input `st='meogalacitca', k=2020`, we have `'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 and the `abs` function.
- You **cannot** use any conditional (i.e., **if**) statements, nor the built-in `min`, `max`, or other 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!).

Before your program begins, the grader has set the following variables:

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

Your program will set two variables as follows:

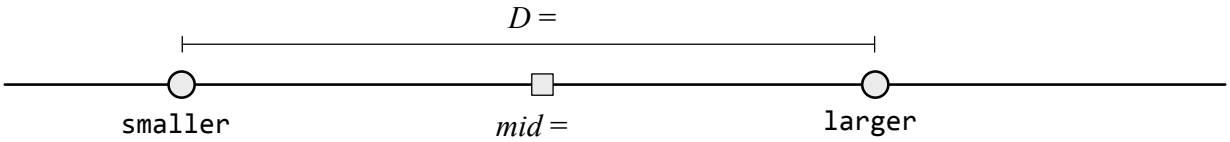
- `smaller: int` will be equal to the smaller of `a` and `b`; and
- `larger: int` will be equal to the larger of `a` and `b`.

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

Examples:

- `a = 4, b = 2` results in `smaller: int = 2, larger: int = 4`.
- `a = 2, b = 4` results in `smaller: int = 2, larger: int = 4`.
- `a = 3, b = 8` results in `smaller: int = 3, larger: int = 8`.
- `a = 11, b = 6` results in `smaller: int = 6, larger: int = 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.

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. Before your program begins, the grader has set the following variables:

`p: int`, `q: int`, `r: int`, and `s: int`

which are input numbers given in arbitrary order (not necessarily sorted).

Your task is to write a program that sets the variable `median: float` to the median 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. Three parameters are set by the grader before your program begins:

- `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.

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 set the variable

```
in_trouble: bool
```

to indicate whether Dr. Sunsern is in trouble because of the kitten.

Examples:

- With `hour = 11, minute = 19, meowing = True`, we have `in_trouble = False`. Meowing at 11:19 is outside the trouble-causing range.
- With `hour = 23, minute = 7, meowing = True`, we have `in_trouble = True`. Meow-ing at 23:07 is after 21, so this will cause trouble.
- With `hour = 4, minute = 55, meowing = False`, we have `in_trouble = 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`, we have `in_trouble = 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.

Your task is to write a program that implements the test as described above so that when the program is finished, the variable `right_triangle: bool` 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 `right_triangle=True`.
- `x=11.5, y=5.0, z=3.0` gives `right_triangle=False`.