219114/115 Programming I

Exercise 3

1. **Coffee machine (coffee_machine.py)**

Write a program that offers to buy one cup of coffee or to fill the supplies or to take its money out. Note that the program is supposed to do one of the mentioned actions at a time. It should also calculate how many ingredients and money have left. Display the number of supplies before and after purchase.

1. First, your program reads one option from the standard input, which can be `"buy"`, `"fill"`, `"take"`. If a user wants to buy some coffee, the input is `"buy"`. If a special worker thinks that it is time to fill out all the supplies for the coffee machine, the input line will be `"fill"`. If another special worker decides that it is time to take out the money from the coffee machine, you'll get the input `"take"`.
2. If the user writes `"buy"` then they must choose one of three types of coffee that the coffee machine can make: espresso, latte, or cappuccino.
   - For one espresso, the coffee machine needs *250 ml* of water and *16 g* of coffee beans. It costs *$4*.
   - For a latte, the coffee machine needs *350 ml* of water, *75 ml* of milk, and *20 g* of coffee beans. It costs *$7*.
   - And for a cappuccino, the coffee machine needs *200 ml* of water, *100 ml* of milk, and *12 g* of coffee. It costs *$6*.
3. If the user writes `"fill"`, the program should ask them how much water, milk, coffee and how many disposable cups they want to add into the coffee machine.
4. If the user writes `"take"` the program should give all the money that it earned from selling coffee.

At the moment, the coffee machine has *$550*, *400 ml* of water, *540 ml* of milk, *120 g* of coffee beans, and *9* disposable cups.

To sum up, your program should print the coffee machine's state, process one query from the user, as well as print the coffee machine's state after that. Try to use functions for implementing every action that the coffee machine can do.

Example 1:

```
The coffee machine has:
400 of water
540 of milk
120 of coffee beans
9 of disposable cups
550 of money

Write action (buy, fill, take):
> buy
What do you want to buy? 1 - espresso, 2 - latte, 3 - cappuccino:
> 3

The coffee machine has:
200 of water
440 of milk
108 of coffee beans
8 of disposable cups
556 of money
```

Example 2:

```
The coffee machine has:
400 of water
540 of milk
120 of coffee beans
9 of disposable cups
550 of money

Write action (buy, fill, take):
> fill
Write how many ml of water do you want to add:
> 2000
Write how many ml of milk do you want to add:
> 500
Write how many grams of coffee beans do you want to add:
> 100
Write how many disposable cups of coffee do you want to add:
> 10

The coffee machine has:
2400 of water
1040 of milk
220 of coffee beans
19 of disposable cups
550 of money
```

Example 3:

```
The coffee machine has:
400 of water
540 of milk
120 of coffee beans
9 of disposable cups
550 of money

Write action (buy, fill, take):
> take
I gave you $550

The coffee machine has:
400 of water
540 of milk
120 of coffee beans
9 of disposable cups
0 of money
```

2. Square root (square_root.py)

Loops are often used in programs that compute numerical results by starting with an approximate answer and iteratively improving it.

For example, one way of computing square roots is Newton's method. Suppose that you want to know the square root of a. If you start with almost any estimate, x, you can compute a better estimate with the following formula:

$$y = \frac{x + a/x}{2}$$

For example, if a is 4 and x is 3:

```
>>> a = 4
>>> x = 3
>>> y = (x + a/x) / 2
>>> y
2.16666666667
```

The result is closer to the correct answer ($\sqrt{4} = 2$). If we repeat the process with the new estimate, it gets even closer:

```
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.00641025641
```

After a few more updates, the estimate is almost exact:

```
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.00001024003
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.00000000003
```

In general we don't know ahead of time how many steps it takes to get to the right answer, but we know when we get there because the estimate stops changing:

```
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.0
>>> x = y
>>> y = (x + a/x) / 2
>>> y
2.0
```

When `y == x`, we can stop. Here is a loop that starts with an initial estimate, `x`, and improves it until it stops changing:

```
while True:
    print(x)
    y = (x + a/x) / 2
    if y == x:
        break
    x = y
```

For most values of `a` this works fine, but in general it is dangerous to test `float` equality. Floating-point values are only approximately right: most rational numbers, like 1/3, and irrational numbers, like $\sqrt{2}$, can't be represented exactly with a `float`.

Rather than checking whether `x` and `y` are exactly equal, it is safer to use the built-in function `abs` to compute the absolute value, or magnitude, of the difference between them:

```
    if abs(y-x) < epsilon:
        break
```

Where `epsilon` has a value like `0.0000001` that determines how close is close enough.

Create a function **mysqrt** that takes **a** as a parameter, chooses a reasonable value of **x**, and returns an estimate of the square root of a.

Also, create a function **test_square_root** that prints a table like this:

```
a   mysqrt(a)     math.sqrt(a)  diff
-   ---------     ------------  ----
1.0 1.0           1.0           0.0
2.0 1.41421356237 1.41421356237 2.22044604925e-16
3.0 1.73205080757 1.73205080757 0.0
4.0 2.0           2.0           0.0
5.0 2.2360679775  2.2360679775  0.0
6.0 2.44948974278 2.44948974278 0.0
7.0 2.64575131106 2.64575131106 0.0
8.0 2.82842712475 2.82842712475 4.4408920985e-16
9.0 3.0           3.0           0.0
```

The first column is a number, **a**; the second column is the square root of a computed with **mysqrt**; the third column is the square root computed by **math.sqrt**; the fourth column is the absolute value of the difference between the two estimates.

3. Loop exercise (loop_exercise.py)

Complete the body of the functions in loop_exercise.py to produce the outcome shown in loop_exercise_result.txt

4. Polygon art (polygon_art.py)

Study the code in draw_shapes.py and make sure you understand how to create geometrical shapes using for loops. Then, in the file polygon_art.py, create a function:

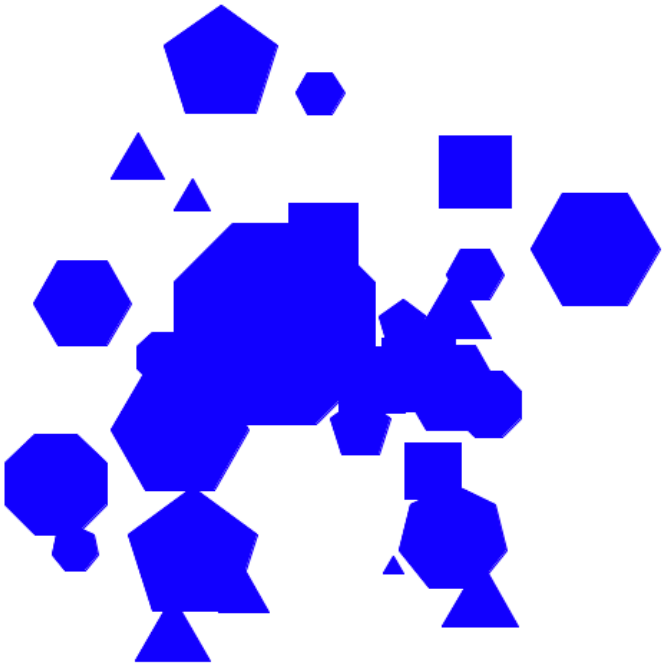
def polygon(x, y, size, n, clr):

That accepts five parameters, where:

- x and y are co-ordinates that start the drawing

- size determines how large a polygon gets drawn
- n indicates the number of sides of a polygon (e.g., when n is 4, the polygon is a square)
- clr represents the color of a polygon

Once you have successfully defined this function, use it to generate a polygon art like the picture shown below:



You might want to import the random module and use the random.randint function to randomize those input parameters (except for the color)

Submission:

- ***Create StudentID_Firstname_ex3 folder, where StudentID is your KU ID and Firstname is your given name***
- ***Put the files to submit, coffee_machine.py, square_root.py, loop_exercise.py, polygon_art.py, into this folder***
- ***Zip the folder and submit the zip file to the course's Google Classroom before the due date***

Grading:

1. Correctness (50%); your code must run and produce correct outcomes; code that does not run because of, for example, syntax errors or name misspelling receives zero credit.
2. Cleanliness (50%): your code must be clean, following PEP 8 style guide; variable names must be meaningful, following PEP 8 convention; comments must be put in for others to be able to read and understand your code, again following PEP 8 convention.