Exercise 4

1. **Coffee machine (coffee_machine_2.py)**

Let's improve the program coffee_machine.py from exercise 3 so it can do multiple actions, one after another. It should repeatedly ask a user what they want to do. If the user types "buy", "fill" or "take", then the program should do exactly the same thing it did in the previous step. However, if the user wants to switch off the coffee machine, they should type "exit". The program should terminate on this command. Also, when the user types "remaining", the program should output all the resources that the coffee machine has.

Write a program that will work endlessly to make coffee for all interested persons until the shutdown signal is given. Introduce two new options: "remaining" and "exit".

Do not forget that you can be out of resources for making coffee. If the coffee machine doesn't have enough resources to make coffee, the program should output a message that says it can't make a cup of coffee.

And the last improvement to the program at this step — if the user types "buy" to buy a cup of coffee and then changes his mind, they should be able to type "back" to return into the main cycle.

Your coffee machine should have the the same initial resources as in the example (400 ml of water, 540 ml of milk, 120 g of coffee beans, 9 disposable cups, $550 in cash.

The greater-than symbol followed by space (>) represents the user input. Notice that it's not the part of the input.

Example 1:

```
Write action (buy, fill, take, remaining, exit):
> remaining

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, remaining, exit):
> buy

What do you want to buy? 1 - espresso, 2 - latte, 3 - cappuccino, back - to main menu:
> 2
I have enough resources, making you a coffee!

Write action (buy, fill, take, remaining, exit):
> remaining

The coffee machine has:
50 of water
465 of milk
100 of coffee beans
8 of disposable cups
$557 of money

Write action (buy, fill, take, remaining, exit):
> buy

What do you want to buy? 1 - espresso, 2 - latte, 3 - cappuccino, back - to main menu:
> 2
```

```
Sorry, not enough water!

Write action (buy, fill, take, remaining, exit):
> fill

Write how many ml of water do you want to add:
> 1000
Write how many ml of milk do you want to add:
> 0
Write how many grams of coffee beans do you want to add:
> 0
Write how many disposable cups of coffee do you want to add:
> 0

Write action (buy, fill, take, remaining, exit):
> remaining

The coffee machine has:
1050 of water
465 of milk
100 of coffee beans
8 of disposable cups
$557 of money

Write action (buy, fill, take, remaining, exit):
> buy

What do you want to buy? 1 - espresso, 2 - latte, 3 - cappuccino, back - to main menu:
> 2
I have enough resources, making you a coffee!

Write action (buy, fill, take, remaining, exit):
> remaining

The coffee machine has:
700 of water
390 of milk
80 of coffee beans
7 of disposable cups
$564 of money

Write action (buy, fill, take, remaining, exit):
> take

I gave you $564

Write action (buy, fill, take, remaining, exit):
> remaining

The coffee machine has:
700 of water
390 of milk
80 of coffee beans
7 of disposable cups
0 of money

Write action (buy, fill, take, remaining, exit):
> exit
```

2. Hangman (hangman.py)

You will play Hangman with the computer who will be responsible for coming up with a word and providing feedback to the user whether a character entered by him/her is in the chosen word.

Familiarize yourself with the classic Hangman in the following short video:

https://www.youtube.com/watch?v=IeW9ZotUVYo

Create the following word list: 'python', 'java', 'kotlin', 'javascript'.

Program the game to choose a random word from it.

Start the game by printing out the word “HANGMAN” and, in the following line, print a number of - mark equal to the length of the computer chosen word.

If the player guesses a letter contained in the chosen word, it should be uncovered in the word.

The player starts the game with 8 "lives", that is our player can input the wrong letter 8 times.

Print ‘No such letter in the word’ and reduce the attempts count if the word guessed by the program doesn't contain this letter.

Print ‘No improvements’ and reduce the attempts count if the guessed word contains this letter but the user tried this letter before.

The attempts count should be decreased only if there are no letters to uncover.

Example 1:

```
H A N G M A N

------
Input a letter: > t

--t---
Input a letter: > z
No such letter in the word

--t---
Input a letter: > t
No improvements

--t---
Input a letter: > t
No improvements

--t---
Input a letter: > y

-yt---
Input a letter: > x
No such letter in the word

-yt---
Input a letter: > y
No improvements

-yt---
Input a letter: > p

pyt---
Input a letter: > p
No improvements

pyt---
Input a letter: > q
No such letter in the word
```

```
pyt---
Input a letter: > p
No improvements
You are hanged!
```

Example 2:

```
H A N G M A N

----
Input a letter: > j

j---
Input a letter: > i
No such letter in the word

j---
Input a letter: > g
No such letter in the word

j---
Input a letter: > g
No such letter in the word

j---
Input a letter: > g
No such letter in the word

j---
Input a letter: > g
No such letter in the word

j---
Input a letter: > a

ja-a
Input a letter: > v

java
You guessed the word!
You survived!
```

3. Pinwheel art (pinwheel.py)

Revisit the Pinwheel project in the “Beauty and Joy of Computing” (BJC) curriculum.

https://bjc.edc.org/bjc-r/cur/programming/1-introduction/3-drawing/4-modify-your-pinwheel.html?topic=nyc_bjc%2F1-intro-loops.topic&course=bjc4nyc.html&novideo&noassignment

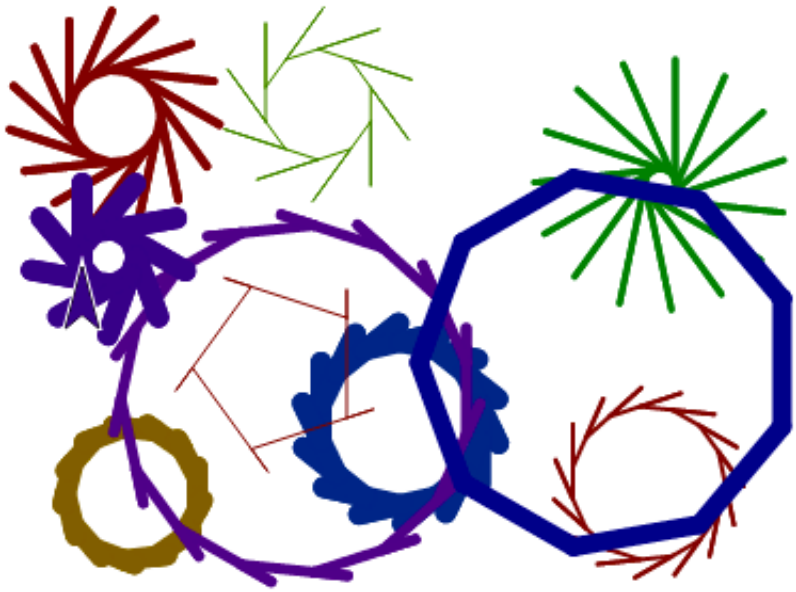
Create a function:

def pinwheel(num_branch, size, backup):

That accepts five parameters, where:

- num_branch, size, and backup are defined in the same way as in the BJC Pinwheel project.

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



You might want to import the random module and use the random.randint function to randomize locations, pen sizes, colors, and those input arguments to the pinwheel function.

You may find it useful to define a list of color as:
clr_list = ['red', 'blue', 'gold', 'brown', 'violet', ‘pink', 'orange', ‘yellow']

Then, random.randint(0, 7) will randomly give an index value to one of the colors in the above list.

Submission:

- ***Create StudentID_Firstname_ex4 folder, where StudentID is your KU ID and Firstname is your given name***
- ***Put the files to submit, coffee_machine_2.py, hangman.py, and pinwheel.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.