Attach the assignment to your HW/Lab ENGR 102 Section **Lab 7a team**
Date: 10/03/22__ **DUE DATE: 10/12/2022**
You name _____Team # (table)_____

ENGR 102 – Fall 2022
Lab: Topic 7 (team)
50 points

Team submission. one submission per team.

Submit your Py-files together with your word/pdf file with screenshots of your tests outputs. Include any derivations, comments and supplemental notes in your word/pdf files.

No pictures by the phone – it is impossible to read. You will be allowed to resubmit and reupload HW as many times as you want to within the due date/time, only last submission will be graded. For submission you may use this file as template: rename file including your name. Do not forget to put your name inside of this file as well. If it is a team work, include the team number and all team members. For this submission use Team Header, include all team members into the list of participants. Submit 1 assignment per team.

Deliverables:
There are two deliverables for this team assignment. Please submit the following files to Canvas:

- `go_moves_planning.pdf` `[25pts]`
- `go_moves.py` `[25pts]`

Activity #1: Go Moves – team (https://en.wikipedia.org/wiki/Go_(game))
The purpose of this activity is to get you used to using lists of lists, in a 2-D matrix-like format. Your team will create a program that sets up a small Go board and lets users place stones.

First, **BEFORE YOU CODE**, you must **think** about how to structure your program. Create a document named go_moves_planning.pdf that contains an algorithm (in English or pseudocode, **NOT** python) of how you want your program to work. You may want to have several smaller algorithms that do one task each (think functions) instead of one large algorithm.

Here are the details:

- The board is a 9 x 9 board. **Use a list of lists** to store your current board.
- Display the current state of the board before every move. Each empty point should just be a period. Each point with a stone should have that stone's identifier.
- For identifiers, you may use lower-case for the white stones and upper-case for the black stones.
  - Use O/o or distinct round symbols (*, @, ●, etc).
  - For a fun alternative, see the note at the bottom.
- When a stone is placed, it must be placed on an empty point.
- Continue to let users place stones until they enter "stop".
- **Important:** You do **NOT** need to enforce any rules of Go or verify moves, with two exceptions:
  - If a user tries to place a stone where one already exists, print a message to the screen and try again.
  - Alternate turns with black placing the first stone.
  - Other than that, you don't need to worry about placing stones on forbidden points, removing stones, determining a winner, etc.
- This is an open-ended assignment. Feel free to get creative with your output!

You need to specify the system you want to use to identify the locations on the board.

- In your pdf document, include instructions for user input.
- This problem will be graded manually so **your instructions must be clear**.
- In Go a simple coordinate system similar to chess is used to identify points on the board. You do not have to use this system, but you can if you wish.

Your pdf must include an algorithm and instructions for running your program.

Now that your team has planned everything, write a program named go_moves.py that sets up a Go board and lets users place stones. **Remember to write test cases and use the "pyramid" approach to programming!**

As an example, here is what the board would look like in the middle of a game:

```
.........
..o......
.oOo.....
.oOOo....
..oOOo...
...oO....
....o....
.........
.........
```

Remember to discuss this as a team and *think* through exactly what you will do *before* developing it!

Note: You can display empty and filled circles using Unicode characters like the example below.

```
print(chr(9675))    # displays empty circle: ○
print(chr(9679))    # displays filled circle: ●
```

You can also change the color of your output using the `colorama` package like the example below.

```
from colorama import Fore, Style
print(Fore.RED + Style.BRIGHT + 'my text' + Fore.RESET + Style.NORMAL)
# displays 'my text' in bright red, then resets future text back
# to default colors
```

Instead of colorama, you can also use the following strings to do the same thing.

```
begin = '\x1b[1m'
end = '\x1b[39m\x1b[22m'
colors = {'red' : '\x1b[31m', 'blue' : '\x1b[34m'}
print(begin + colors['red'] + 'red text' + end + ' back to normal')
print(begin + colors['blue'] + 'blue text' + end + ' back to normal')
```