AI Assignment 5

Name: Yash Oswal
Div: B Roll no: 38
SRN: 201901226

---

1. Map Colouring

Output:

```
                The Graph Is
A ->  ['B', 'C', 'D']
B ->  ['A', 'C']
C ->  ['A', 'B', 'D', 'E']
D ->  ['A', 'C', 'F', 'E']
E ->  ['F', 'C', 'D']
F ->  ['E', 'D', 'G']
G ->  ['F']
```

```
                The Solution Is -
A ->  Red
B ->  Green
C ->  Yellow
D ->  Green
E ->  Red
F ->  Yellow
G ->  Red
```

2. Cryptharithmatic

Output:

```
Enter First Word - BASE
Enter Second Word - BALL
Enter Result - GAMES
Solution Is -
        BASE
+       BALL
-------------
        GAMES


Result 1 = 7483 + 7455 = 14938
```

3. Crossword Puzzle:
   Output

```
======= The Initial Crossword is =======
 # # # # # # # # #
 #               #
 #   #   # # # # #
 #   #
 #   #   # # # # #
 #   #   # # # # #
 #   # # # # # # #
The Across Words Are :  HYBRID, EARTH
The Down Words Are :  BREAD, HELMET



======== The Final Crossword is ========
 # # # # # # # # #
 # H Y B R I D #
 # E # R # # # #
 # L # E A R T H
 # M # A # # # #
 # E # D # # # #
 # T # # # # # #
```

Code:

1. Map Colouring

```
from colorama import Fore, Back, Style, init
init(strip=False)
init(autoreset=True)
class map_coloring():
   # Colors Used
   colors = [Fore.RED+'Red', Fore.GREEN+'Green', Fore.YELLOW+'Yellow',
   Fore.MAGENTA+'Violet']
   # Map
   states = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
   neighbors = {}
   neighbors['A'] = ['B', 'C', 'D']
   neighbors['B'] = ['A', 'C']
   neighbors['C'] = ['A', 'B', 'D', 'E']
   neighbors['D'] = ['A', 'C', 'F', 'E']
   neighbors['E'] = ['F', 'C', 'D']
   neighbors['F'] = ['E', 'D', 'G']
   neighbors['G'] = ['F']
   # Output
   colors_of_states = {}
   def print_graph(self):
      for key in self.neighbors:
         print(Fore.CYAN+ key + Fore.WHITE + ' -> ', self.neighbors[key])
   def promising(self, state, color):
      for neighbor in self.neighbors.get(state):
         color_of_neighbor = self.colors_of_states.get(neighbor)
         if color_of_neighbor == color:
            return False
      return True
   def get_color_for_state(self, state):
      for color in self.colors:
         if self.promising(state, color):
            return color
   def start(self):
      print(Fore.BLUE+"\n\n\t\tThe Graph Is ")
      self.print_graph()
```

```
      print("\n\n")
      for state in self.states:
         self.colors_of_states[state] = self.get_color_for_state(state)
         print(f"Color Used For State {state} is {self.colors_of_states[state]}")
         print(Fore.BLUE+"\n\n\t\tThe Solution Is - ")
         for key in self.colors_of_states:
            print(Fore.BLUE+key + Fore.WHITE+' -> ', self.colors_of_states[key])
temp = map_coloring()
temp.start()
```

2. Cryptharithmetic

```
from colorama import Fore, Back, Style, init
init(strip=False)
init(autoreset=True)
class cryptarithmetic():
   solved = False
   count = 0
   def start(self):
      word1 = input("Enter First Word - ").upper()
      word2 = input("Enter Second Word - ").upper()
      result = input("Enter Result - ").upper()
      values = []
      visited = [False for x in range(10)]
      equation = [word1, word2, result]
      # Get Unique Words
      set = []
      for c in word1:
         if c not in set:
            set.append(c)
      for c in word2:
         if c not in set:
            set.append(c)
      for c in result:
         if c not in set:
            set.append(c)
      if len(set) > 10:
         print("\nNo Solution (as values will repeat)\n")
```

```
            exit()
         print("Solution Is - ")
         print(f" \t{word1}\n+\t{word2}\n-------------\n\t{result}\n\n")
         self.solve(set, values, visited, equation)
      def solve(self, letters, values, visited, equation):
         if len(letters) == len(values):
            map = {}
            for letter, val in zip(letters, values):
               map[letter] = val
            if map[equation[0][0]] == 0 or map[equation[1][0]] == 0 or map[equation[2][0]] == 0:
               return
            word1, word2, res = "", "", ""
            for c in equation[0]:
               word1 += str(map[c])
            for c in equation[1]:
               word2 += str(map[c])
            for c in equation[2]:
               res += str(map[c])
            if int(word1) + int(word2) == int(res):
               self.count += 1
               print(Fore.GREEN+f"Result {self.count} = {word1} + {word2} = {res}\n")
               solved = True
            return
         for i in range(10):
            if not visited[i]:
               visited[i] = True
               values.append(i)
               self.solve(letters, values, visited, equation)
               values.pop()
               visited[i] = False
temp = cryptarithmetic()
temp.start()
```

3. Crossword

```
from typing import List
def check_right(i, j, grid) -> tuple[int, int, int]:
   counter = 0
   while (counter + j) < len(grid[i]):
      if grid[i][j + counter] == ' ':
         counter += 1
      else:
         break
   if counter < 2:
      return None
   else:
      return (i, j, counter)

def check_down(i, j, grid) -> tuple[int, int, int]:
   counter = 0
   while (counter + i) < len(grid):
      if grid[i + counter][j] == ' ':
         counter += 1
      else:
         break
   if counter < 2:
      return None
   else:
      return (i, j, counter)

def get_across_slots(grid: list[str]):
   accross_slots = []
   i = 0
   while i < len(grid):
      j = 0
      while j < len(grid[i]):
         if grid[i][j] == ' ':
            if slot := check_right(i, j, grid):
               accross_slots.append(slot)
               j += slot[2]
         j += 1
      i += 1
```

```
   return accross_slots

def get_down_slots(grid: list[str]):
   t_grid = []
   # Get transpose of grid
   for i in range(len(grid)):
      string = ''.join([row[i] for row in grid])
      t_grid.append(string)
   down_slots = get_across_slots(t_grid)
   # The down slots are for the transposed grid,
   # so we need to convert them to our original grid's coordinates
   down_slots = [(slot[1], slot[0], slot[2]) for slot in down_slots]
   return down_slots
def start(across_words: list[str], down_words: list[str], grid: list[str]) -> list[str]:
   across_slots = get_across_slots(grid)
   down_slots = get_down_slots(grid)
   # We need a mutable grid, so we use list[list[str]]
   mut_grid = []
   for i in range(len(grid)):
      arr = []
      for j in range(len(grid[i])):
         arr.append([grid[i][j]])
      mut_grid.append(arr)
# Start filling the across words
   i = 0
   while len(across_words):
      used = False
      if used:
         across_slots.pop(i)
      else:
         i = (i + 1) % len(down_slots)
      if len(across_words[0]) == across_slots[i][2]:
         x, y, _ = across_slots[i]
         for counter, letter in enumerate(across_words[0]):
            mut_grid[x][y + counter] = [letter]
         else:
            used = True
         across_words.pop(0)
```

```
# Start filling the down words
   i = 0
   while len(down_words):
      used = False
      if used:
         down_slots.pop(i)
      else:
         i = (i + 1) % len(down_slots)
      if len(down_words[0]) == down_slots[i][2]:
         x, y, _ = down_slots[i]
         for counter, letter in enumerate(down_words[0]):
            mut_grid[x + counter][y] = [letter]
         else:
            used = True
         down_words.pop(0)
# Convert list[list[str]] to list[str]
   grid = []
   for i in range(len(mut_grid)):
      string = ""
      for j in range(len(mut_grid[i])):
         for k in range(len(mut_grid[i][j])):
            string += mut_grid[i][j][k][0]
      grid.append(string)
   return grid

def display_grid(grid: list[str]) -> None:
   for row in grid:
      for col in row:
         print(f"{col:>2}", end='')
      print()

def main():
   ACROSS = ['HYBRID', 'EARTH']
   DOWN = ['BREAD', 'HELMET']
   grid = [
   "########",
   "#     #",
   "# # ####",
```

```
    "# #    ",
    "# # ####",
    "# # ####",
    "# ######",
    ]

    print(" The Initial Crossword is ".center(40, '='))
    display_grid(grid)
    print("The Across Words Are : ", ', '.join(ACROSS))
    print("The Down Words Are : ", ', '.join(DOWN))
    result = start(ACROSS, DOWN, grid)
    print('\n\n')
    print(" The Final Crossword is ".center(40, '='))
    display_grid(result)
main()
```