МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра ІСМ

******

Звіт

до лабораторної роботи №5

На тему “Розробка ASCII ART генератора для візуалізації 3D-фігур”

З дисципліни “Спеціалізовані мови програмування”

*Виконав:*

*ст. гр. ІТ-31*

*Іштван Комоній*

*Прийняв:*

*Щербак С. С.*

*Львів - 2023*

**Мета роботи:** Створення додатка для малювання 3D-фігур у ASCII-арті на основі об’єктно - орієнтованого підходу та мови Python.

**Хід роботи**

**Програмний код:**

/figure\_art\_interface/figure\_art\_interface

from abc import ABC, abstractmethod

from figures.figure import Figure

from figures.figures\_3d.cube import Cube

from figures.figures\_2d.square import Square

from helpers.helpers import text\_file\_saver

class Command(ABC):

@abstractmethod

def execute(self):

pass

class Generate3DFigureCommand(Command):

def \_\_init\_\_(self, figure\_interface):

self.figure\_interface = figure\_interface

def execute(self):

print(self.figure\_interface.generate\_3d\_figure())

class SetSizeCommand(Command):

def \_\_init\_\_(self, figure\_interface, new\_size):

self.figure\_interface = figure\_interface

self.new\_size = new\_size

def execute(self):

self.figure\_interface.set\_size(self.new\_size)

class SetColorCommand(Command):

def \_\_init\_\_(self, figure\_interface, new\_color):

self.figure\_interface = figure\_interface

self.new\_color = new\_color

def execute(self):

self.figure\_interface.set\_color(self.new\_color)

# Add more command classes for SetType, SetPaddings, Generate2DFigure, SaveToFile3D, and SaveToFile2D

class FigureArtInterface(Figure):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.type = "cube"

self.left\_padding = 5

self.top\_padding = 5

self.bottom\_padding = 5

self.commands = {

1: Generate3DFigureCommand(self),

2: SetSizeCommand(self, 0),

3: SetColorCommand(self, ""),

# Add more command instances here

}

def set\_type(self, type):

self.type = type

def set\_paddings(self, left\_padding, top\_padding, bottom\_padding):

self.left\_padding = left\_padding

self.top\_padding = top\_padding

self.bottom\_padding = bottom\_padding

def set\_primary\_data(self):

size = int(input("figure size: "))

self.set\_size(size)

color = input(

"figure color(blue, green, red, magenta, yellow, white, cyan): ")

self.set\_color(color)

type = input("figure type(skip for default): ")

if len(type):

self.set\_type(type)

def generate\_with\_left\_padding(self, text):

lines = text.split('\n')

padded\_lines = [f"{' ' \* self.left\_padding}{line}" for line in lines]

return '\n'.join(padded\_lines)

def generate\_with\_top\_padding(self, text):

space = self.top\_padding \* '\n'

padded\_lines = space + text

return padded\_lines

def generate\_with\_bottom\_padding(self, text):

space = self.bottom\_padding \* '\n'

padded\_lines = text + space

return padded\_lines

def generate\_with\_paddings(self, text):

return self.generate\_with\_bottom\_padding(self.generate\_with\_top\_padding(self.generate\_with\_left\_padding(text)))

def generate\_3d\_figure(self):

if self.type == 'cube':

cube = Cube()

cube.set\_size(self.size)

cube.set\_color(self.color)

return self.generate\_with\_paddings(cube.generate\_figure())

return super().generate\_figure()

def generate\_2d\_figure(self):

if self.type == 'cube':

square = Square()

square.set\_size(self.size)

square.set\_color(self.color)

return self.generate\_with\_paddings(square.generate\_figure())

return super().generate\_figure()

def save\_to\_file\_2d(self):

filename = input("enter filename before saving: ")

text\_file\_saver(

filename, self.generate\_2d\_figure())

def save\_to\_file\_3d(self):

filename = input("enter filename before saving: ")

text\_file\_saver(

filename, self.generate\_3d\_figure())

@staticmethod

def show\_menu():

print("choose menu option")

print("[ 1 ] - generate 3d figure")

print("[ 2 ] - set size")

print("[ 3 ] - set color(blue, green, red, magenta, yellow, white, cyan)")

print("[ 4 ] - set type(cube)")

print("[ 5 ] - set paddings")

print("[ 6 ] - get 2d version of the figure")

print("[ 7 ] - save to file(3d)")

print("[ 8 ] - save to file(2d)")

print("[ 0 ] - exit")

def loop\_menu(self):

while True:

self.show\_menu()

menu\_choice = int(input("menu key: "))

if (menu\_choice == 1):

print(self.generate\_3d\_figure())

elif (menu\_choice == 2):

new\_size = int(input("enter new size: "))

self.set\_size(new\_size)

elif (menu\_choice == 3):

new\_color = input("enter new color: ")

self.set\_color(new\_color)

elif (menu\_choice == 4):

new\_type = input("enter new type: ")

self.set\_type(new\_type)

elif (menu\_choice == 5):

left\_padding = int(input("enter left padding: "))

top\_padding = int(input("enter top padding: "))

bottom\_padding = int(input("enter bottom padding: "))

self.set\_paddings(left\_padding, top\_padding, bottom\_padding)

elif (menu\_choice == 6):

print(self.generate\_2d\_figure())

elif (menu\_choice == 7):

self.save\_to\_file\_3d()

elif (menu\_choice == 8):

self.save\_to\_file\_2d()

else:

break

def launch(self):

self.set\_primary\_data()

print(self.generate\_3d\_figure())

self.loop\_menu()

/figures/figure

class Figure:

def \_\_init\_\_(self):

self.size = 1

self.color = 'green'

def set\_size(self, size):

self.size = size

def set\_color(self, color):

self.color = color

def generate\_figure(self):

return ""

/figures/figures\_2d/square

from figures.figure import Figure

from helpers.helpers import paint\_text

class Square(Figure):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def generate\_square(self):

n = self.size

line = ''

if n > 1:

line += n \* 2 \* '\_'

else:

line += '\_'

top\_line = ' ' + line

bottom\_line = '|' + line + '|' + '\n'

inner\_spaces = n \* 2 \* ' '

inner\_fill = '|' + inner\_spaces + '|'

content = ''

i = 1

while (i < n):

content += inner\_fill + '\n'

i += 1

square = top\_line + '\n' + content + bottom\_line

return square

def generate\_figure(self):

generated = self.generate\_square()

colored = paint\_text(generated, self.color)

return colored

/figures/figures\_3d/cube

from figures.figure import Figure

from helpers.helpers import paint\_text

class Cube(Figure):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def generate\_cube(self):

n = self.size

line = ''

if n > 1:

line += n \* 2 \* '\_'

else:

line += '\_'

top\_line = n \* ' ' + line

mid\_line\_space = 2 \* n - 2

mid\_line = '/' + line + '/' + mid\_line\_space \* ' ' + '\\'

bottom\_line = (n - 1) \* ' ' + '\\' + line + '\/'

inner\_spaces = n \* 2 \* ' '

inner\_top = '/' + inner\_spaces + '/'

inner\_bottom = '\\' + inner\_spaces + '\\'

top\_content = ''

bottom\_content = ''

for i in range(1, n):

top\_content += (n - i) \* ' ' + inner\_top + \

(i - 1) \* 2 \* ' ' + '\\' + '\n'

bottom\_content += (i - 1) \* ' ' + inner\_bottom + \

(n - i) \* 2 \* ' ' + '/' + '\n'

cube = top\_line + '\n' + top\_content + mid\_line + \

'\n' + bottom\_content + bottom\_line + '\n'

return cube

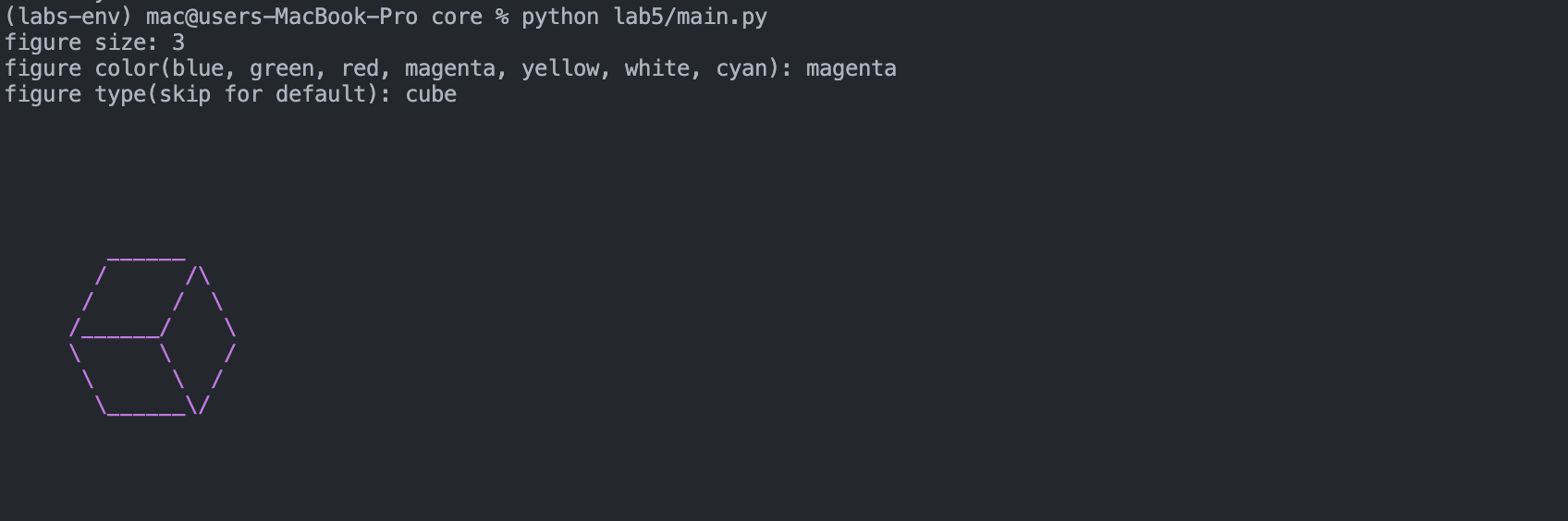
def generate\_figure(self):

generated = self.generate\_cube()

colored = paint\_text(generated, self.color)

return colored

Результат виконання програми:



**Висновок:** Під час виконання даної лабораторної роботи було створено додаток для малювання 3D-фігур у ASCII-арті на основі об’єктно - орієнтованого підходу та мови Python.