МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №3

з курсу

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

Виконала студентка

групи ІТ-21сп

**Романишин.Д.О.**

Прийняв

**Щербак С.С.**

Львів - 2023

**Мета:** створення додатка Генератора ASCII-арту.

**План роботи**

**Завдання 1: Введення користувача.**

Створити Python-програму, яка приймає введення користувача для слова або фрази, яку треба перетворити в ASCII-арт.

**Завдання 2: Бібліотека ASCII-арту.**

Інтегрувати бібліотеку ASCII-арту (наприклад, pyfiglet або art) у програму для генерації ASCII-арту з введення користувача.

**Завдання 3: Вибір шрифту.**

Дозволити користувачам вибирати різні стилі шрифтів для свого ASCII-арту. Надати список доступних шрифтів та дозвольте їм вибрати один.

**Завдання 4: Колір тексту.**

Реалізувати опцію вибору користувачем кольору тексту для їхнього ASCII-арту. Підтримати основний вибір кольорів (наприклад, червоний, синій, зелений).

**Завдання 5: Форматування виводу.**

Переконатися, що створений ASCII-арт правильно відформатований та вирівнюється на екрані для зручності читання.

**Завдання 6: Збереження у файл.**

Додати функціональність для збереження створеного ASCII-арту у текстовому файлі, щоб користувачі могли легко завантажувати та обмінюватися своїми творіннями.

**Завдання 7: Розмір ARTу.**

Дозволити користувачам вказувати розмір (ширина і висота) ASCII-арту, який вони хочуть створити. Масштабувати текст відповідно.

**Завдання 8: Вибір символів.**

Дозволити користувачам вибирати символи, які вони хочуть використовувати для створення ASCII-арту (наприклад, '@', '#', '\*', тощо).

**Завдання 9: Функція попереднього перегляду.**

Реалізувати функцію попереднього перегляду, яка показує користувачам попередній перегляд їхнього ASCII-арту перед остаточним збереженням.

**Завдання 10: Інтерфейс, зрозумілий для користувача.**

Створити зручний для користувача інтерфейс командного рядку для додатка, щоб зробити його інтуїтивно зрозумілим та легким у використанні.

Код виконаних завдань представлено нижче.

**Main.py**

from ArtConsole import ArtConsole

art = ArtConsole()

art.run()

**ArtGenerator.py**

import pyfiglet

from colorama import Fore

class ArtGenerator:

\_message = ''

\_ascii\_art = None

\_font = 'standard'

\_color = Fore.RESET

\_width = 80

\_direction = 'auto'

\_justify = 'auto'

\_user\_height = None

\_user\_width = None

\_FONTS = {

'1': "standard",

'2': "slant",

'3': "big",

'4': "block",

'5': "bubble",

'6': "digital",

'7': "isometric1",

'8': "isometric2",

'9': "letters",

'10': "script",

'11': "shadow",

'12': "starwars",

}

\_COLORS = {

'BLACK': Fore.BLACK,

'RED': Fore.RED,

'GREEN': Fore.GREEN,

'YELLOW': Fore.YELLOW,

'BLUE': Fore.BLUE,

'MAGENTA': Fore.MAGENTA,

'CYAN': Fore.CYAN,

'WHITE': Fore.WHITE,

'RESET': Fore.RESET,

}

\_DIRECTIONS = {

'LTR': 'left-to-right',

'RTl': 'right-to-left',

}

\_JUSTIFIES = {

'CENTER': 'center',

'RIGHT': 'right',

'LEFT': 'left',

}

# GET/SET MESSAGE

@property

def message(self):

return self.\_message

@message.setter

def message(self, value):

self.\_message = value

# GET/SET FONT

@property

def font(self):

return self.\_font

@font.setter

def font(self, value):

self.\_font = value

# GET/SET COLOR

@property

def color(self):

return self.\_color

@color.setter

def color(self, value):

self.\_color = value

# GET/SET WIDTH

@property

def width(self):

return self.\_width

@width.setter

def width(self, value):

self.\_width = value

# GET/SET DIRECTION

@property

def direction(self):

return self.\_direction

@direction.setter

def direction(self, value):

self.\_direction = value

# GET/SET JUSTIFY

@property

def justify(self):

return self.\_justify

@justify.setter

def justify(self, value):

self.\_justify = value

# GET/SET User height

@property

def user\_height(self):

return self.\_user\_height

@user\_height.setter

def user\_height(self, value):

self.\_user\_height = value

# GET/SET User height

@property

def user\_width(self):

return self.\_user\_width

@user\_width.setter

def user\_width(self, value):

self.\_user\_width = value

def \_\_init\_\_(self, message=''):

self.\_message = message

self.\_ascii\_art = pyfiglet.figlet\_format(self.\_message)

def set\_custom\_font(self, symbol):

self.\_font = 'banner3'

self.create()

# self.\_ascii\_art = self.\_ascii\_art(font='')

for original\_char, user\_char in zip("#/", symbol):

self.\_ascii\_art = self.\_ascii\_art.replace(original\_char, user\_char)

def save(self, filename):

with open(filename, 'w') as file:

self.create()

file.write(self.\_ascii\_art)

def \_create(self, \*\*kwargs):

if 'font' in kwargs:

font = kwargs['font']

else:

font = self.\_font

return pyfiglet.Figlet(font=font, direction=self.\_direction, justify=self.\_justify, width=self.\_width)

def create(self):

try:

art = self.\_create()

self.\_ascii\_art = art.renderText(self.\_message)

except NameError as e:

raise NameError(f"Error: {e}")

def prev\_view(self):

art = self.\_create()

art = (self.\_color + art.renderText(self.\_message))

return art

def \_art\_zoom(self):

art = self.\_create(font='banner3')

art = art.renderText(self.\_message)

return art

def zoom(self):

line = ''

lines = []

len\_line = 0

len\_lines = 0

mx = 0

art = self.\_art\_zoom()

for i in art:

if i != '\n':

line += i

mx += 1

else:

if len\_line < mx:

len\_line = mx

mx = 0

lines += [line]

line = ''

len\_lines = lines.\_\_len\_\_()

square\_base = len\_lines \* len\_line

square\_update = self.\_user\_height \* self.\_user\_width

if square\_update >= square\_base:

pw = round(square\_update / square\_base)

else:

pw = round(square\_base / square\_update)

new\_lines = []

for line in lines:

new\_line = ''

for l in line:

new\_line += l

new\_line += l

new\_lines += [new\_line] \* pw

laq = ''

for line in new\_lines:

laq += line + '\n'

self.\_ascii\_art = laq

def \_\_str\_\_(self):

return self.\_color + self.\_ascii\_art

**ArtConsole.py**

from ArtGenerator import ArtGenerator

class ArtConsole(ArtGenerator):

@classmethod

def set\_parameters(cls, dct, input\_message):

for key, value in dct.items():

print(f"{key}: {value}")

value = input(f"Select {input\_message}: ")

current\_value = dct.get(value, None)

if current\_value:

return current\_value

else:

return None

def input\_function(self, dct, input\_message, default):

check = input(f"do you want to set {input\_message}? (1/0): ")

font = None

if check == '1':

font = self.set\_parameters(dct, 'font')

if font:

return font

else:

return default

def configuration(self):

message = input("Input message: ")

self.message = message

self.font = self.input\_function(self.\_FONTS, 'font', self.font)

self.color = self.input\_function(self.\_COLORS, 'color', self.color)

self.justify = self.input\_function(self.\_JUSTIFIES, 'justify', self.justify)

self.direction = self.input\_function(self.\_DIRECTIONS, 'direction', self.direction)

change\_font = input('do you want to create own font? (1/0): ')

if change\_font == '1':

custom\_symbol = input("Input custom symbol: ")

self.set\_custom\_font(custom\_symbol)

change\_zoom = input('do you want to create zoom? (1/0): ')

if change\_zoom == '1':

height = int(input("Input custom height: "))

width = int(input("Input custom width: "))

self.user\_height = height

self.user\_width = width

self.zoom()

watch = input("do you want to watch art before to save? (1/0):")

if watch == '1':

if self.\_ascii\_art:

print(self.\_ascii\_art)

else:

print(self.prev\_view())

check = input("do you want to save? (1/0): ")

if check == '1':

save = input("Input filename: ")

self.save(save)

def run(self):

while input("input 'f' if you want to exit: ") != 'f':

try:

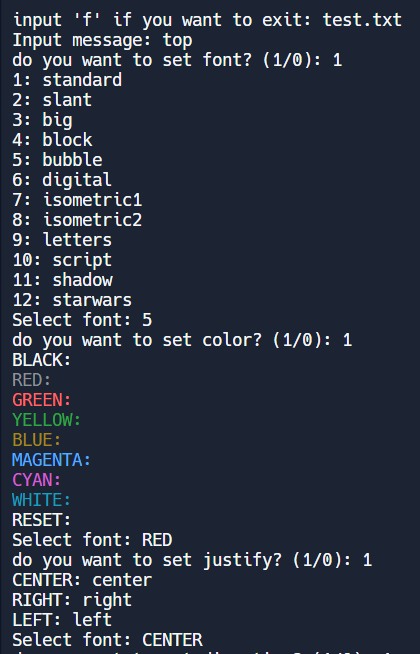
self.configuration()

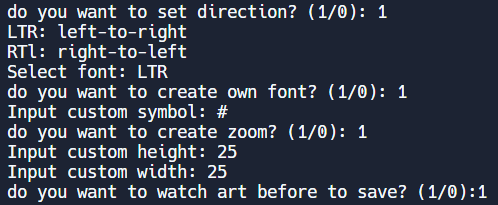
self.create()

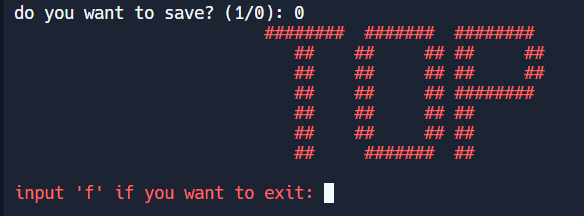
print(self)

except Exception as e:

print(e)

**Результат виконання програми**





**Висновок:** під час виконання лабораторної роботи я навчився розробляти ASCII ART генератор для візуалізації текстових даних.