МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗВІТ

для лабораторної роботи № 4 з

дисципліни

«Спеціалізовані мови програмування»

Виконав:

студент гр. ІТ-32

Паньків Б. В.

Прийняв:

доц. каф. ІСМ

Щербак С.С.

Львів-2023

Мета. Створення генератора ASCII-арту без використання зовнішніх бібліотек.

Хід виконання:

Завдання 1: Введення користувача
Створіть програму Python, яка отримує введення користувача щодо слова або фрази, яку вони хочуть перетворити в ASCII-арт.
Завдання 2: Набір символів
Визначте набір символів (наприклад, '@', '#', '*', тощо), які будуть використовуватися для створення ASCII-арту. Ці символи будуть відображати різні відтінки.
Завдання 3: Розміри Art-у
Запитайте у користувача розміри (ширина і висота) ASCII-арту, який вони хочуть створити. Переконайтеся, що розміри в межах керованого діапазону
Завдання 4: Функція генерації Art-у
Напишіть функцію, яка генерує ASCII-арт на основі введення користувача, набору символів та розмірів. Використовуйте введення користувача, щоб визначити, які символи використовувати для кожної позиції в Art-у.
Завдання 5: Вирівнювання тексту
Реалізуйте опції вирівнювання тексту (ліво, центр, право), щоб користувачі могли вибирати, як їх ASCII-арт розміщується на екрані.
Завдання 6: Відображення мистецтва
Відобразіть створений ASCII-арт на екрані за допомогою стандартних функцій друку Python.
Завдання 7: Збереження у файл
Додайте можливість зберігати створений ASCII-арт у текстовий файл, щоб користувачі могли легко завантажувати та обмінюватися своїми твореннями.
Завдання 8: Варіанти кольорів
Дозвольте користувачам вибирати опції кольорів (чорно-білий, відтінки сірого) для свого ASCII-арту.
Завдання 9: Функція попереднього перегляду
Реалізуйте функцію попереднього перегляду, яка показує користувачам попередній перегляд їх ASCII-арту перед остаточним збереженням
Завдання 10: Інтерфейс, зрозумілий для користувача
Створіть інтерфейс для користувача у командному рядку, щоб зробити програму легкою та інтуїтивно зрозумілою для використання.

Код:

Functions.py

```
import os
import colorama
from colorama import Fore
import re
from functools import reduce

colorama.init(autoreset=True)
colors = {i: getattr(Fore, color) for i, color in enumerate(sorted(Fore.__dict__.keys()))}


class DataProcessor:

    def __init__(self, file_path):
```

```
    if file_path is None:
        raise ValueError("File path should have a value")
    self.__file_path = file_path

def _read_file(self):
    with open(self.__file_path, "r") as file:
        return file.read()

@staticmethod
def display_colors():
    for i, color in colors.items():
        print(f"{i}. {color}")

@staticmethod
def _parse_metadata(line, meta_data):
    if not re.match(r"^\w+::\d+$", line):
        raise ValueError("Metadata has incorrect format")
    key, value = line.split("::")
    meta_data[key] = int(value)

@staticmethod
def _process_data_annotation(line, meta_data, is_data_annotation_found):
    if line == "@data":
        if "height" not in meta_data:
            raise ValueError("The file has to contain meta information such as height")
        is_data_annotation_found[0] = True

def _get_all_data(self):
    file_data = self._read_file().split("\n")
    is_data_annotation_found = [False]
    representation = ""
    symbol = None

    for line in file_data:
        self._process_data_annotation(line, self.__meta_data, is_data_annotation_found)

        if is_data_annotation_found[0]:
            if re.match("^@symbol::.$", line):
                if symbol is not None:
                    self.__data[symbol] = representation
                symbol = line[9:]
                representation = ""
                length = 0
                counter = 1
            elif re.match(r"^\^.+\$$", line):
                if counter == 1:
                    length = len(line)
                if len(line) != length:
```

```
                raise ValueError("Length of the row has to be equal within a certain character in the file")
                representation += line[1:-1] + ("" if counter == self.__meta_data["height"] else "\n")
                counter += 1
            else:
                raise ValueError("Data information has incorrect format")

        elif not is_data_annotation_found[0]:
            self._parse_metadata(line, self.__meta_data)

    def retrieve(self, text, color_position, width):
        result = {}
        all_needed_symbols = {}
        properties = {}
        row_count = 0
        current_position_in_row = 0

        for i in range(len(text)):
            representation = str(self.__data[text[i]]).split("\n")
            if current_position_in_row + len(representation[0]) > width:
                if len(representation[0]) > width:
                    raise ValueError("Width of the text is too small")
                row_count += 1
                current_position_in_row = 0
                current_position_in_row += len(representation[0])
                properties[row_count] = properties.get(row_count, 0) + 1
            else:
                current_position_in_row += len(representation[0])
                properties[row_count] = properties.get(row_count, 0) + 1
            all_needed_symbols.update({i: reduce(lambda x, y: x + "\n" + y, representation)})

        symbol_count = 0
        for i in properties:
            for j in range(properties[i]):
                representation = all_needed_symbols[symbol_count].split("\n")
                symbol_count += 1
                for k in range(len(representation)):
                    result[k + i * 6] = result.get(k + i * 6, "") + representation[k]

        return colors[color_position] + reduce(lambda x, y: x + "\n" + y, result.values())


class TxtProcessor(DataProcessor):
    __meta_data = {}
    __data = {}

    def __init__(self, file_path):
        if not file_path.endswith(".txt") and os.path.exists(file_path):
```

```
            raise ValueError("File should be .txt file")
        super().__init__(file_path)
        self._get_all_data()
```

runner.py

```
from functions import *


def write_into_file(file_path, text):
    with open(file_path, "w") as file:
        file.write(text)


def read_from_file(file_path):
    with open(file_path, "r") as file:
        return file.read()


def main():
    try:
        initial_text = input("Enter text in order to display: ")
        DataProcessor.display_colors()
        color_position = int(input("Enter position of color you would like to use: "))
        width = int(input("Enter width of text you would like to display: "))
        file_path = input("Enter path to file containing alphabet: ")

        txt_processor = TxtProcessor(file_path)
        text = txt_processor.retrieve(initial_text, color_position, width)

        print(text)
        write_into_file("output.txt", text)

    except KeyError as e:
        print(f"Key error! {e}")
    except ValueError as e:
        print(e)
    except FileNotFoundError:
        print("File not found! Please, check the path to the file")


if __name__ == "__main__":
    main()
```

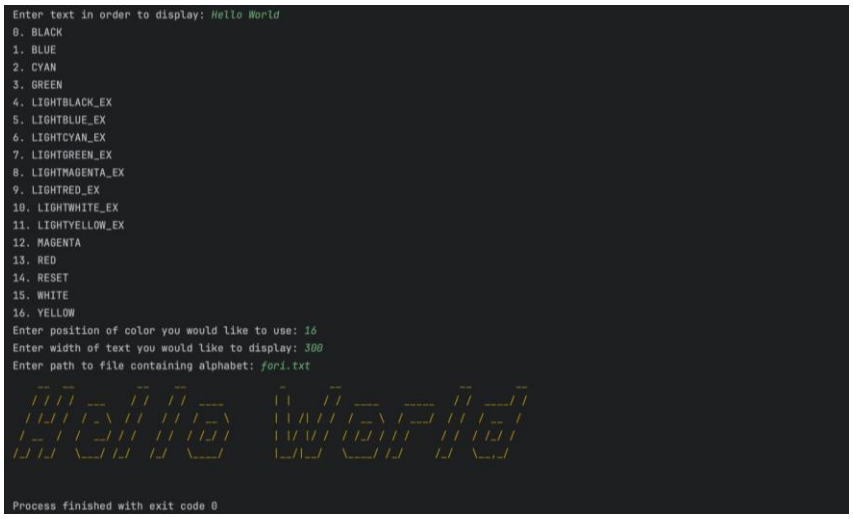

Рис. 1 Виконання програми

Посилання на GitHub-репозиторій із кодом: https://github.com/BOHDAN1329/SMP

Висновки: Виконуючи ці завдання, я створив генератор ASCII-арту з нуля, та надав можливість налаштовувати символи, розміри, вирівнювання та кольори, що дозволило мені глибше розібратися як створюється ASCII-арт.