Міністерство освіти та науки України
Національний університет «Львівська політехніка»
Кафедра систем автоматизованого проектування

ПОЯСНЮВАЛЬНА ЗАПИСКА

до курсової роботи з навчальної дисципліни
«Розробка кросплатформенних додатків (Python)»
на тему «Додаток для побудови графіків»

Виконав:
ст. гр. ПП-22 Кирилюк Дмитро

Перевірив:
проф. каф. САП Щербовських С. В.

Львів – 2025

Розділ 1. Аналіз проблеми та підготовка до вирішення [20 балів]

Увага!
Результатом курсової роботи є розроблення програмного забезпечення для вирішення проблемної ситуації, яку студент формулює самостійно. Програма має бути написана мовою Python, складатися з одного файлу і працювати на пристроях користувача без необхідності встановлення додаткових програм чи додатків.
Під час виконання курсової роботи обов'язково дотримуйтесь поданої структури.

- Текст та зображення вставляйте тільки у білі поля.
- Текст у зафарбованих полях змінювати або видаляти заборонено.

Якщо ці вимоги не будуть виконані, то робота **не перевірятиметься**.

1.1. Визначення проблемної ситуації [5 балів]

Описати проблемну ситуацію, розкриваючи такі складові:

- **Об'єкт:** Хто або що зазнає впливу проблемної ситуації. Це може бути людина, група осіб, організація, довкілля тощо.
- **Суб'єкт:** Хто або що є причиною виникнення проблемної ситуації. Це може бути конкретна особа, організація, природний процес або технічний фактор.
- **Об'єктивна сторона:** Описати небезпеку або наслідки проблемної ситуації для об'єкта. Як дії суб'єкта спричиняють ці наслідки. Також зазначити, де, коли і за яких обставин виникла проблемна ситуація. Пояснити, як саме відбувається вплив і що є безпосереднім носієм цього впливу (наприклад, дія певного механізму, рішення особи або природне явище).
- **Суб'єктивна сторона:** Пояснити, чому суб'єкт винен у виникненні ситуації. Яка у нього мета, мотиви та емоційний стан (опишіть лише ті характеристики, які можна застосувати до суб'єкта в даній ситуації).

Вимога до обсягу тексту: **не менше 1000 символів.**
Опис має бути детальним з чіткім поясненням всіх елементів проблемної ситуації та їх взаємозв'язку.

У сучасному світі візуалізація даних відіграє важливу роль у прийнятті рішень, аналізі інформації та презентації результатів досліджень. Проте для багатьох користувачів, які не мають програмістських навичок, створення якісних графіків залишається складним завданням. Хоча існують потужні бібліотеки на зразок matplotlib, вони вимагають знання мови програмування Python і спеціалізованих команд, що створює бар'єр для новачків. Відсутність простого графічного інтерфейсу для побудови графіків стає суттєвою перешкодою для ефективної роботи з даними в освітньому, науковому та прикладному середовищі.

Об'єктом впливу в цій ситуації є користувачі — студенти, дослідники-початківці, викладачі та фахівці в різних галузях (економіка, біологія, соціологія тощо), які потребують інструментів для побудови графіків, але не мають програмістських знань або досвіду роботи з бібліотекою matplotlib чи іншими аналогічними інструментами. Ці користувачі часто стикаються з необхідністю візуалізувати дані — наприклад, під час підготовки звітів, презентацій або проведення досліджень.

Суб'єктом, який спричинив проблемну ситуацію, є розробники та спільнота програмного забезпечення, зокрема ті, хто створює інструменти для візуалізації даних, але орієнтується переважно на користувачів із досвідом програмування. Іншою складовою суб'єкта є загальна недооцінка потреб непрофесійних користувачів серед розробників — відсутність програм із простим, інтуїтивно зрозумілим графічним інтерфейсом, що дозволяє будувати графіки без написання коду.

Наслідком цієї ситуації є те, що об'єкт (користувачі) стикається з труднощами у виконанні завдань, пов'язаних із графічною візуалізацією даних. Вони або змушені витрачати значну кількість часу на вивчення бібліотек на кшталт matplotlib, або ж взагалі відмовляються від візуалізації, обмежуючи якість подання своїх результатів. Це негативно впливає як на їхню ефективність, так і на результати роботи. Проблемна ситуація особливо

гостро постає в академічному середовищі, де час і доступ до якісних інструментів обмежений.

Проблема виникає в умовах, коли ринок програмного забезпечення насичений інструментами для професіоналів, але не пропонує доступних рішень для широкого кола користувачів. Безпосереднім носієм впливу є відсутність GUI-програми, яка дозволяє створювати графіки через інтуїтивно зрозумілий інтерфейс, шляхом введення даних у таблицю або імпорту з файлу, вибору типу графіка зі списку, налаштування осей, підписів тощо — без написання жодного рядка коду.

Суб'єкт є винним у виникненні проблемної ситуації, тому що ігнорує потреби непрофесійної частини користувачів, керуючись здебільшого технічними інтересами або з міркувань ефективності розробки (наприклад, створення лише бібліотек, які легко інтегрувати в інші проєкти). Основними мотивами суб'єкта є зосередження на поточних тенденціях програмування, фокусування на професійній спільноті та нехтування зворотним зв'язком від новачків або непрофесійних користувачів. Можливо, суб'єкт вважає, що кожен, хто потребує графіків, має опанувати хоча б базові основи кодування, що формує певну упередженість.

1.2. Потреби та цілі зацікавлених сторін [5 балів]

Визначте **не менше п'яти зацікавлених сторін**. Це можуть бути особи, групи або організації, які мають інтерес у вирішенні проблеми або впливають на її результат.
Для кожної зацікавленої сторони:

- Описати цілі та потреби цієї сторони (що вона хоче досягти в результаті вирішення проблеми).
- Визначити роль цієї сторони у проблемній ситуації (як вона пов'язана із проблемною ситуацією і який вплив має на її вирішення).

Зацікавлені сторони можуть бути активними учасниками проблемної ситуації або тими, хто лише стикається з її наслідками.
Кожна зацікавлена сторона має свої інтереси, цілі та мотивацію, які можуть бути різними.
Усі зацікавлені сторони повинні бути згадані у п. 1.1.

Назва зацікавленої сторони	**Опис цілей та потреб зацікавленої сторони. Роль зацікавленої сторони у проблемній ситуації**
Студенти та учні	Студенти прагнуть швидко та якісно створювати графіки для курсових, лабораторних та наукових робіт без необхідності вивчати програмування або бібліотеки на зразок matplotlib. Їхня потреба — мати простий інструмент для візуалізації даних у навчальному процесі. Вони є основною групою, яка стикається з наслідками проблеми, тобто з браком зручного інструменту. Їхня потреба і скарги можуть стати рушієм для створення рішення.
Викладачі та наукові керівники	Викладачі хочуть, щоб їхні студенти могли ефективно працювати з графіками без необхідності окремого навчання мовам програмування. Їм потрібен інструмент, який дозволяє швидко перевіряти й аналізувати подані графіки. Вони не тільки стикаються з наслідками проблеми (отримують недосконалі або відсутні графіки у звітах), але також можуть впливати на її вирішення через рекомендації щодо інструментів або участь у розробці освітніх рішень.

Розробники програмного забезпечення	Розробники хочуть створювати продукти, що мають попит, та отримувати прибуток або підтримку з боку спільноти. Вони також можуть бути зацікавлені у створенні відкритих проєктів із соціальною значущістю. Це ключова сторона, яка має технічну можливість розробити програму з графічним інтерфейсом для побудови графіків. Водночас, саме відсутність такого рішення з їхнього боку й створила проблему.
Освітні установи	Освітні заклади прагнуть забезпечити студентів інструментами, які сприяють ефективному навчанню, та зменшити технічні бар'єри у навчальному процесі. Вони можуть впроваджувати такі програми у навчальні плани, закуповувати або рекомендувати їх, а також формувати запит на розробку простого ПЗ для візуалізації даних.
Неурядові організації або грантові фонди у сфері освіти і науки	Їхня мета — підтримка доступу до якісної освіти, розвиток цифрової грамотності та зменшення освітньої нерівності. Вони зацікавлені у фінансуванні ініціатив, які покращують навчальний процес. Можуть виступати фінансовими чи організаційними партнерами для розробки й поширення такого програмного забезпечення. Їхній вплив полягає у наданні ресурсів для реалізації рішення.

1.3. Формулювання мети роботи [5 балів]

Сформулювати мету роботи, дотримуючись принципів SMART:

- **S (Specific)**: Мета повинна бути чіткою і зрозумілою. Вона повинна описувати, що саме потрібно досягти, без загальних або нечітких формулювань.
- **M (Measurable)**: Мету слід формулювати так, щоб можна було оцінити її досягнення. Це може бути через кількісні (наприклад, кількість рядків коду) або якісні показники (наприклад, поліпшення ефективності програми).
- **A (Achievable)**: Мета має бути реалістичною. Врахуйте доступні ресурси (час, знання, інструменти) і обмеження, щоб результат був досяжним.
- **R (Relevant)**: Мета повинна мати практичну цінність і важливість для вирішення проблеми. Вона повинна мотивувати до виконання роботи і нести конкретну користь зацікавленим сторонам.
- **T (Time-bound)**: Мета повинна містити чіткі часові обмеження. Це дозволить ефективно планувати виконання роботи і відслідковувати прогрес.

Обґрунтуйте, чому запропонована мета відповідає кожному з цих принципів.

Розробити програму для побудови графіків з графічним інтерфейсом на основі бібліотек tkinter і matplotlib, яка дозволить користувачам без знань програмування будувати п'ять основних типів графіків — лінійний (plot), точковий (scatter), стовпчиковий (bar), гістограму (histogram) та кругову діаграму (pie) — з можливістю введення даних вручну. Графічний інтерфейс програми повинен складатись із 4 частин: меню для вибору типу графіку, кнопка для виведення налаштувань графіку та дозволена кількість графіків; рамка для налаштувань кожного графіку (кожен графік розміщений у комірці); рамка з налаштуваннями остаточної фігури графіку, такими як написи на осях, заголовок, легенда, сітка і кнопка створення графіку; статусне вікно, куди будуть виводитись стани побудови графіків, такі як: "успішно побудовано", "помилка у рамці з номером #", "максимальна кількість рамок досягнута". Програма повинна бути створена за 5 тижнів (3-5 годин на тиждень) із щотижневою перевіркою результатів розробки.

1.4. Аналіз існуючих рішень та огляд літератури [5 балів]

Виберіть **не менше п’яти літературних джерел**, які стосуються теми вашої курсової роботи. Це можуть бути наукові статті, книги, дослідження або інші ресурси, які допоможуть розкрити тему.
Для кожного джерела:

- Складіть короткий опис. В цьому описі потрібно проаналізувати які аналоги, прототипи, ідеї чи результати описані в джерелі. Після опису обов’язково додати посилання на джерело.
- Визначте основні недоліки запропонованих рішень або невирішені питання, які стосуються вашої теми.

В результаті огляду потрібно зрозуміти існуючі підходи до вирішення вашої проблеми. Виявити їхні слабкі сторони та обґрунтувати, чому необхідно виконати саме ваше дослідження або розробку.
Важливо! Не використовуйте підручники та технічну документацію з мови програмування Python. Використовуйте лише наукові статті, книги або інші джерела, які **безпосередньо стосуються вашої теми**.

№	Короткий опис аналогів, прототипів, ідей або результатів. Посилання на літературне джерело	Виявлені недоліки
[1]	Tk.DataViz: A Powerful Data Visualization Tool. У цьому дослідженні представлено інструмент Tk.DataViz, розроблений на основі Python з використанням бібліотек Tkinter та Pandas. Інструмент дозволяє користувачам створювати різноманітні графіки, включаючи можливості об’єднання графіків, експорт у PDF та інтерактивні позначки. Основна мета — забезпечити ефективну візуалізацію даних для користувачів без глибоких технічних знань. URL: https://link.springer.com/chapter/10.1007/978-981-19-3311-0_27	- Відсутня підтримка деяких типів графіків, таких як кругові діаграми. - Інтерфейс складний для новачків.
[2]	Learning Vis Tools: Teaching Data Visualization Tutorials. Стаття описує підхід до навчання візуалізації даних, починаючи з інструментів з графічним інтерфейсом і поступово переходячи до програмування з використанням D3.js. Автори підкреслюють важливість доступності інструментів для студентів без попереднього досвіду в програмуванні. URL: [1907.08796] Learning Vis Tools: Teaching Data Visualization Tutorials	- Більшість розглянутих інструментів орієнтовані на веб-розробку та можуть вимагати знань HTML/CSS.
[3]	Tkinter and Data Visualization: Creating Interactive Charts and Graphs. У блозі розглядається використання Tkinter у поєднанні з matplotlib та seaborn для створення інтерактивних графіків. Автор демонструє, як інтегрувати ці бібліотеки для побудови графіків у десктопних додатках. URL: Tkinter and Data Visualization: Creating Interactive Charts and Graphs | by Tom | TomTalksPython | Medium	- Орієнтовано на користувачів з базовими знаннями програмування. - Відсутність повноцінного GUI.
[4]	Tableau — це потужний інструмент для візуалізації даних, який дозволяє користувачам створювати інтерактивні графіки без необхідності програмування. Його drag-and-drop інтерфейс робить процес створення візуалізацій інтуїтивно зрозумілим. Tableau активно використовується в бізнесі, науці та освіті для аналізу та представлення даних.	- Висока вартість ліцензії для повнофункціональної версії. - Обмежена підтримка платформ (наприклад, відсутність версії для

	URL: https://www.wired.com/2010/02/the-quest-to-make-web-data-fun/	macOS на момент публікації).
[5]	Microsoft Excel є одним із найпоширеніших інструментів для створення графіків на основі табличних даних. Користувач може побудувати діаграму, не володіючи знаннями програмування, за допомогою вбудованого інтерфейсу. Excel підтримує різні типи графіків: стовпчикові, лінійні, кругові, гістограми, діаграми розсіювання тощо. Завдяки зручності й широкому розповсюдженню Excel активно використовується в освіті, бізнесі й дослідженнях. URL: Free Online Spreadsheet Software: Excel | Microsoft 365	- Обмежені можливості кастомізації графіків у порівнянні з matplotlib. - Залежність від комерційної ліцензії (не всі користувачі мають доступ до Excel).

Розділ 2. Розробка програмного забезпечення [25 балів]

2.1. Програмний код мовою Python [5 балів]

Подати лістинг програми мовою Python.
Дотримуйтесь стандарту кодування PEP 8, зокрема, використовуйте відступи, правильні назви змінних, форматування і коментарі відповідно до стандарту.
Вимоги до структури програми:

- програма має містити не менше трьох класів,
- сумарна кількість атрибутів і методів у всіх класах повинна бути не менше десяти,
- код має бути зрозумілим і прокоментованим.

Увага! Подавайте тільки коректний та протестований код.
Див. PEP 8 «Style Guide for Python Code»: https://www.python.org/dev/peps/pep-0008/

```
from abc import ABC, abstractmethod
from datetime import datetime
import tkinter as tk
from tkinter import ttk

import matplotlib.pyplot as plt

GRAPH_TYPES = ["plot", "scatter", "bar", "histogram", "pie"]

MAX_CELL_NUMBER = 5

COLORS = [
    "red", "blue", "green", "yellow", "black", "white", "purple",
    "orange", "pink", "brown", "gray", "cyan", "magenta", "lime",
    "navy", "gold", "teal", "violet", "indigo", "olive",
]

MARKERS = [
    " ", ".", ",", "o", "v", "^", "<", ">", "1", "2", "3", "4",
    "s", "p", "*", "h", "H", "+", "x", "X", "D", "d", "|", "_",
]

# Function to convert string input to list of floats or strings
def get_list(string_var: tk.StringVar) -> list[float | str]:
    user_input = string_var.get()
    if not user_input:
        return []
    items = user_input.split(",")
    processed_items = []
    for item in items:
        item = item.strip()
        if not item:
            continue
        try:
            processed_items.append(float(item))
        except ValueError:
            processed_items.append(item)
    return processed_items

# Classes for different graph types ----------------------------------
# Base class for all cells
class Cell(ABC):
    def __init__(self, frame):
        self.frame = frame
        self.id = id(self)

        self.label = tk.StringVar()

        self.label_entry_frame = tk.LabelFrame(
            self.frame,
```

```
            text="Label",
            bd=1,
            relief="solid",
        )
        self.label_entry_frame.grid(row=0, column=0, padx=5, pady=5)
        self.label_entry = tk.Entry(
            self.label_entry_frame,
            textvariable=self.label,
        )
        self.label_entry.pack(padx=5, pady=5)

        self.remove_button = tk.Button(
            self.frame,
            text="rm",
            command=lambda: cell_manager.delete_cell(self.id),
        )
        self.remove_button.grid(row=0, column=1, padx=5, pady=5)

    @abstractmethod
    def build(self):
        pass

# Base class for 2D cells
class TwoDimensionalCell(Cell):
    def __init__(self, frame):
        super().__init__(frame)

        self.x = tk.StringVar()
        self.x_entry_frame = tk.LabelFrame(
            self.frame,
            text="x",
            bd=1,
            relief="solid",
        )
        self.x_entry_frame.grid(
            row=1, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.x_entry = tk.Entry(
            self.x_entry_frame,
            textvariable=self.x,
        )
        self.x_entry.pack(padx=5, pady=5, fill="x")

        self.y = tk.StringVar()
        self.y_entry_frame = tk.LabelFrame(
            self.frame,
            text="y",
            bd=1,
            relief="solid",
        )
        self.y_entry_frame.grid(
            row=2, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.y_entry = tk.Entry(
            self.y_entry_frame,
            textvariable=self.y,
        )
        self.y_entry.pack(padx=5, pady=5, fill="x")

        self.color = tk.StringVar(value=COLORS[0])
        self.color_combobox_frame = tk.LabelFrame(
            self.frame,
            text="Color",
            bd=1,
```

```
            relief="solid",
        )
        self.color_combobox_frame.grid(
            row=3, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.color_combobox = ttk.Combobox(
            self.color_combobox_frame,
            textvariable=self.color,
            values=COLORS,
            state="readonly",
        )
        self.color_combobox.pack(padx=5, pady=5, fill="x")

class PlotCell(TwoDimensionalCell):
    def __init__(self, frame):
        super().__init__(frame)

        self.linewidth = tk.StringVar(value="1.5")
        self.linewidth_spinbox_frame = tk.LabelFrame(
            self.frame,
            text="Line width",
            bd=1,
            relief="solid",
        )
        self.linewidth_spinbox_frame.grid(
            row=4, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.linewidth_spinbox = ttk.Spinbox(
            self.linewidth_spinbox_frame,
            from_=0,
            to=30,
            textvariable=self.linewidth,
            increment=0.5,
            format="%.1f",
        )
        self.linewidth_spinbox.pack(padx=5, pady=5, fill="x")

        self.linestyles = ["solid", "dashed", "dashdot", "dotted", "None"]
        self.linestyle = tk.StringVar(value=self.linestyles[0])
        self.linestyle_combobox_frame = tk.LabelFrame(
            self.frame,
            text="Line style",
            bd=1,
            relief="solid",
        )
        self.linestyle_combobox_frame.grid(
            row=5, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.linestyle_combobox = ttk.Combobox(
            self.linestyle_combobox_frame,
            textvariable=self.linestyle,
            values=self.linestyles,
            state="readonly",
        )
        self.linestyle_combobox.pack(padx=5, pady=5, fill="x")

        self.marker = tk.StringVar(value=MARKERS[0])
        self.marker_combobox_frame = tk.LabelFrame(
            self.frame,
            text="Marker",
            bd=1,
            relief="solid",
        )
        self.marker_combobox_frame.grid(
```

```
            row=6, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.marker_combobox = ttk.Combobox(
            self.marker_combobox_frame,
            textvariable=self.marker,
            values=MARKERS,
            state="readonly",
        )
        self.marker_combobox.pack(padx=5, pady=5, fill="x")

    def build(self):
        x = get_list(self.x)
        y = get_list(self.y)

        if not x:
            plt.plot(
                y,
                color=self.color.get(),
                label=self.label.get(),
                linewidth=self.linewidth.get(),
                linestyle=self.linestyle.get(),
                marker=self.marker.get(),
            )
        else:
            plt.plot(
                x,
                y,
                color=self.color.get(),
                label=self.label.get(),
                linewidth=self.linewidth.get(),
                linestyle=self.linestyle.get(),
                marker=self.marker.get(),
                markersize=float(self.linewidth.get()) + 4.5,
            )

class ScatterCell(TwoDimensionalCell):
    def __init__(self, frame):
        super().__init__(frame)

        self.marker = tk.StringVar(value=MARKERS[1])
        self.marker_combobox_frame = tk.LabelFrame(
            self.frame,
            text="Marker",
            bd=1,
            relief="solid",
        )
        self.marker_combobox_frame.grid(
            row=4, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.marker_combobox = ttk.Combobox(
            self.marker_combobox_frame,
            textvariable=self.marker,
            values=MARKERS,
            state="readonly",
        )
        self.marker_combobox.pack(padx=5, pady=5, fill="x")

        self.markersize = tk.StringVar(value="6.0")
        self.markersize_spinbox_frame = tk.LabelFrame(
            self.frame,
            text="Marker size",
            bd=1,
            relief="solid",
        )
```

```
        self.markersize_spinbox_frame.grid(
            row=5, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.markersize_spinbox = ttk.Spinbox(
            self.markersize_spinbox_frame,
            from_=0,
            to=30,
            textvariable=self.markersize,
            increment=0.5,
            format="%.1f",
        )
        self.markersize_spinbox.pack(padx=5, pady=5, fill="x")

    def build(self):
        x = get_list(self.x)
        y = get_list(self.y)

        plt.scatter(
            x,
            y,
            color=self.color.get(),
            label=self.label.get(),
            marker=self.marker.get(),
            s=float(self.markersize.get()) ** 2,
        )

class BarCell(TwoDimensionalCell):
    def build(self):
        x = get_list(self.x)
        y = get_list(self.y)

        plt.bar(x, y, color=self.color.get(), label=self.label.get())

class HistogramCell(Cell):
    def __init__(self, frame):
        super().__init__(frame)

        self.data = tk.StringVar()
        self.data_entry_frame = tk.LabelFrame(
            self.frame,
            text="Data",
            bd=1,
            relief="solid",
        )
        self.data_entry_frame.grid(
            row=1, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.data_entry = tk.Entry(
            self.data_entry_frame,
            textvariable=self.data,
        )
        self.data_entry.pack(padx=5, pady=5, fill="x")

        self.bins = tk.StringVar()
        self.bins_entry_frame = tk.LabelFrame(
            self.frame,
            text="Bins",
            bd=1,
            relief="solid",
        )
        self.bins_entry_frame.grid(
            row=2, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.bins_entry = tk.Entry(
```

```
            self.bins_entry_frame,
            textvariable=self.bins,
        )
        self.bins_entry.pack(padx=5, pady=5, fill="x")

        self.color = tk.StringVar(value=COLORS[0])
        self.color_combobox_frame = tk.LabelFrame(
            self.frame,
            text="Color",
            bd=1,
            relief="solid",
        )
        self.color_combobox_frame.grid(
            row=3, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.color_combobox = ttk.Combobox(
            self.color_combobox_frame,
            textvariable=self.color,
            values=COLORS,
            state="readonly",
        )
        self.color_combobox.pack(padx=5, pady=5, fill="x")

    def build(self):
        data = get_list(self.data)
        bins_str = self.bins.get()
        if not bins_str:
            plt.hist(data, color=self.color.get(), label=self.label.get())
        else:
            plt.hist(
                data,
                bins=int(bins_str),
                color=self.color.get(),
                label=self.label.get(),
            )

class PieCell(Cell):
    def __init__(self, frame):
        super().__init__(frame)

        self.label_entry_frame.config(text="Labels")

        self.data = tk.StringVar()
        self.data_entry_frame = tk.LabelFrame(
            self.frame,
            text="Data",
            bd=1,
            relief="solid",
        )
        self.data_entry_frame.grid(
            row=1, column=0, columnspan=2, padx=5, pady=5, sticky="ew"
        )
        self.data_entry = tk.Entry(
            self.data_entry_frame,
            textvariable=self.data,
        )
        self.data_entry.pack(padx=5, pady=5, fill="x")

    def build(self):
        data = get_list(self.data)
        labels = get_list(self.label)
        if not labels:
            plt.pie(data)
        else:
```

```
            plt.pie(data, labels=labels)

# Class to manipulate cells ------------------------------------------
# This class is responsible for creating, deleting and showing cells
class CellManager:
    def __init__(self):
        # Cell list, where key is cell id and value is Cell object
        self.cells: dict[int, Cell] = {}

    def create_cell(self, graph_type):
        if len(self.cells) == MAX_CELL_NUMBER:
            add_status_text("Max number of cells reached!")
            return

        if (
            graph_option_menu["menu"].entrycget(
                GRAPH_TYPES.index(graph_type), "state"
            )
            == "disabled"
        ):
            return

        frame = tk.LabelFrame(
            cells_list,
            text=graph_type,
            bd=1,
            relief="solid",
        )
        frame.pack(side="left", anchor="n", padx=10, pady=10)

        def disable_pie():
            if not self.cells:
                menu = graph_option_menu["menu"]
                menu.entryconfig(GRAPH_TYPES.index("pie"), state="disabled")

        match graph_type:
            case "plot":
                cell = PlotCell(frame)
                disable_pie()
            case "scatter":
                cell = ScatterCell(frame)
                disable_pie()
            case "bar":
                cell = BarCell(frame)
                disable_pie()
            case "histogram":
                cell = HistogramCell(frame)
                disable_pie()
            case "pie":
                cell = PieCell(frame)
                menu = graph_option_menu["menu"]
                for i in range(menu.index("end") + 1):
                    menu.entryconfig(i, state="disabled")

        self.cells[cell.id] = cell

    def delete_cell(self, cell_id):
        self.cells[cell_id].frame.destroy()
        del self.cells[cell_id]

        if not self.cells:
            menu = graph_option_menu["menu"]
            for i in range(menu.index("end") + 1):
                menu.entryconfig(i, state="normal")
```

```
    def show(self):
        if not self.cells:
            add_status_text("No cells to plot!")
            return

        # Make all frames white
        for cell in self.cells.values():
            cell.frame.config(bg=cells_list.cget("bg"))

        plt.figure()
        for cell in self.cells.values():
            try:
                cell.build()
            except Exception as e:
                add_status_text(f"Error in red cell {cell.id}: {e}")
                cell.frame.config(bg="#FFCCCC")
                plt.close()
                return

        add_status_text("Successfully plotted!")

        if title_var.get():
            plt.title(title_var.get())
        if xlabel_var.get():
            plt.xlabel(xlabel_var.get())
        if ylabel_var.get():
            plt.ylabel(ylabel_var.get())
        if legend_var.get():
            plt.legend()
        if grid_var.get():
            plt.grid()

        plt.show()

# GUI ---------------------------------------------------------------------
root = tk.Tk()
root.title("Plotting App")
root.geometry("530x550")

# Navigation --------------------------------------------------------------
navigation = tk.LabelFrame(
    root,
    text="Choose graph type",
    bd=1,
    relief="solid",
)
navigation.pack(
    anchor="w",
    padx=10,
    pady=5,
    fill="x",
)

graph_type = tk.StringVar(value=GRAPH_TYPES[0])
graph_option_menu = tk.OptionMenu(
    navigation,
    graph_type,
    *GRAPH_TYPES,
)
graph_option_menu.config(width=10)
graph_option_menu.pack(side="left", padx=10, pady=5)

create_cell = tk.Button(
```

```
    navigation,
    text="Create cell",
    command=lambda: cell_manager.create_cell(graph_type.get()),
)
create_cell.pack(side="left", padx=10, pady=5)

number_label = tk.Label(
    navigation,
    text=f"Max number of cells: {MAX_CELL_NUMBER}",
)
number_label.pack(side="right", padx=10, pady=5)

# Cell List ---------------------------------------------------------------
main_frame = tk.Frame(root, relief="solid")
main_frame.pack(pady=5, fill="both", expand=True)

canvas_frame = tk.Frame(main_frame)
canvas_frame.pack(fill="both", expand=True)

canvas = tk.Canvas(canvas_frame)
canvas.pack(side="left", fill="both", expand=True)

y_scrollbar = tk.Scrollbar(
    canvas_frame, orient="vertical", command=canvas.yview
)
y_scrollbar.pack(side="right", fill="y")

x_scrollbar = tk.Scrollbar(
    main_frame, orient="horizontal", command=canvas.xview
)
x_scrollbar.pack(fill="x")

canvas.configure(
    xscrollcommand=x_scrollbar.set, yscrollcommand=y_scrollbar.set
)

cells_list = tk.Frame(canvas, relief="solid")
canvas.create_window((0, 0), window=cells_list, anchor="nw")

def update_scrollregion(event):
    canvas.config(scrollregion=canvas.bbox("all"))

cells_list.bind("<Configure>", update_scrollregion)

def _on_mousewheel(event):
    canvas.yview_scroll(int(-1 * (event.delta / 120)), "units")

def _on_linux_scroll(event):
    if event.num == 4:
        canvas.yview_scroll(-1, "units")
    elif event.num == 5:
        canvas.yview_scroll(1, "units")

canvas.bind("<MouseWheel>", _on_mousewheel)
canvas.bind("<Button-4>", _on_linux_scroll)
canvas.bind("<Button-5>", _on_linux_scroll)

cell_manager = CellManager()

# Plotting ----------------------------------------------------------------
plotting = tk.LabelFrame(
    root,
    text="Plotting",
    bd=1,
```

```
    relief="solid",
)
plotting.pack(
    anchor="w",
    padx=10,
    pady=5,
    fill="x",
)

title_var = tk.StringVar()
title_frame = tk.LabelFrame(
    plotting,
    text="Title",
    bd=1,
    relief="solid",
    width=100,
    height=45,
)
title_frame.pack(side="left", padx=5, pady=5)
title_frame.pack_propagate(False)
title_entry = tk.Entry(
    title_frame,
    textvariable=title_var,
)
title_entry.pack(padx=5, pady=5, fill="x")

xlabel_var = tk.StringVar()
xlabel_frame = tk.LabelFrame(
    plotting,
    text="Xlabel",
    bd=1,
    relief="solid",
    width=100,
    height=45,
)
xlabel_frame.pack(side="left", padx=5, pady=5)
xlabel_frame.pack_propagate(False)
xlabel_entry = tk.Entry(
    xlabel_frame,
    textvariable=xlabel_var,
)
xlabel_entry.pack(padx=5, pady=5, fill="x")

ylabel_var = tk.StringVar()
ylabel_frame = tk.LabelFrame(
    plotting,
    text="Ylabel",
    bd=1,
    relief="solid",
    width=100,
    height=45,
)
ylabel_frame.pack(side="left", padx=5, pady=5)
ylabel_frame.pack_propagate(False)
ylabel_entry = tk.Entry(
    ylabel_frame,
    textvariable=ylabel_var,
)
ylabel_entry.pack(padx=5, pady=5, fill="x")

legend_grid_frame = tk.Frame(plotting)
legend_grid_frame.pack(side="left", padx=5, pady=5)

legend_var = tk.BooleanVar(value=False)
```

```
legend_checkbutton = tk.Checkbutton(
    legend_grid_frame,
    text="Show legend",
    variable=legend_var,
)
legend_checkbutton.pack(anchor="w")

grid_var = tk.BooleanVar(value=False)
grid_checkbutton = tk.Checkbutton(
    legend_grid_frame,
    text="Show grid",
    variable=grid_var,
)
grid_checkbutton.pack(anchor="w")

plot_button = tk.Button(
    plotting,
    text="Plot",
    command=cell_manager.show,
    bg="orange",
    width=10,
    height=2,
)
plot_button.pack(side="left", padx=10, pady=5)

# Status Bar ---------------------------------------------------------------
status_frame = tk.LabelFrame(
    root,
    text="Status",
    bd=1,
    relief="solid",
)
status_frame.pack(padx=10, pady=5, fill="x")

scrollbar = tk.Scrollbar(status_frame)
scrollbar.pack(side="right", fill="y")

status_text = tk.Text(
    status_frame,
    wrap="word",
    height=4,
    width=50,
    bg="white",
    yscrollcommand=scrollbar.set,
    state="disabled",
)
status_text.pack(padx=10, pady=5, fill="x")

scrollbar.config(command=status_text.yview)

# Function to add text to the status bar
def add_status_text(text: str):
    current_time = datetime.now().strftime("[%H:%M:%S]")
    status_text.config(state="normal")
    status_text.insert(tk.END, f"{current_time} {text}\n")
    status_text.config(state="disabled")
    status_text.yview(tk.END)

add_status_text("Welcome to the plotting app!")

if __name__ == "__main__":
    root.mainloop()
```

2.2. Архітектури програми [5 балів]

Опишіть архітектуру програми, пояснивши її структуру. Вкажіть, які класи та методи створено, чому було обрано саме таке розбиття, і як це допомагає досягти поставлених цілей. Зверніть увагу на логіку взаємодії між класами та методами, щоб показати, як кожен елемент програми підтримує її загальну функціональність. Розробіть UML-діаграму класів для наочного представлення структури програми (відповідно до вимог п. 2.1). На діаграмі зазначте: назви класів, атрибути (з обов'язковим зазначенням типу, наприклад: `int`, `str`, `list` тощо) та методи кожного класу.
Діаграма має бути зрозумілою, точною та оформленою за стандартами UML. Це дозволить легко сприйняти логіку побудови програми та взаємодію між її компонентами.

Архітектура програми побудована на об'єктно-орієнтованих принципах з використанням абстракції, наслідування та поліморфізму для створення гнучкої системи побудови графіків у графічному інтерфейсі на основі tkinter. Програма містить 8 класів, з яких 2 – абстрактні. Основу складає абстрактний клас Cell, який визначає базову структуру для всіх типів візуалізацій. У ньому реалізовано загальні компоненти інтерфейсу, зокрема поле для мітки (label) і кнопку видалення комірки. Його метод build() є абстрактним і реалізується у дочірніх класах, кожен з яких відповідає окремому типу графіка.

Від класу Cell наслідується клас TwoDimensionalCell, який розширює функціональність для графіків, що потребують осі x та y (наприклад, plot, scatter, bar). У цьому класі реалізовано відповідні поля для введення координат та вибору кольору. Конкретні реалізації PlotCell, ScatterCell і BarCell успадковують TwoDimensionalCell і реалізують специфічну логіку побудови своїх графіків у методі build(). Наприклад, PlotCell містить додаткові параметри, як товщина лінії (linewidth), стиль лінії (linestyle) та маркер (marker), що дозволяє користувачу більш гнучко налаштовувати вигляд графіка. ScatterCell додає параметри для вибору маркера та його розміру, а BarCell використовує тільки базові x, y та колір.

Окремо стоїть HistogramCell, який напряму наслідує Cell, оскільки гістограма не потребує осі x та y. Замість цього вона використовує один набір даних (data) та параметр кількості бінів (bins). Для побудови кругової діаграми (pie chart) створено клас PieCell, який також наслідує Cell і містить поля для введення значень (sizes), підписів (labels) і вибору кольору.

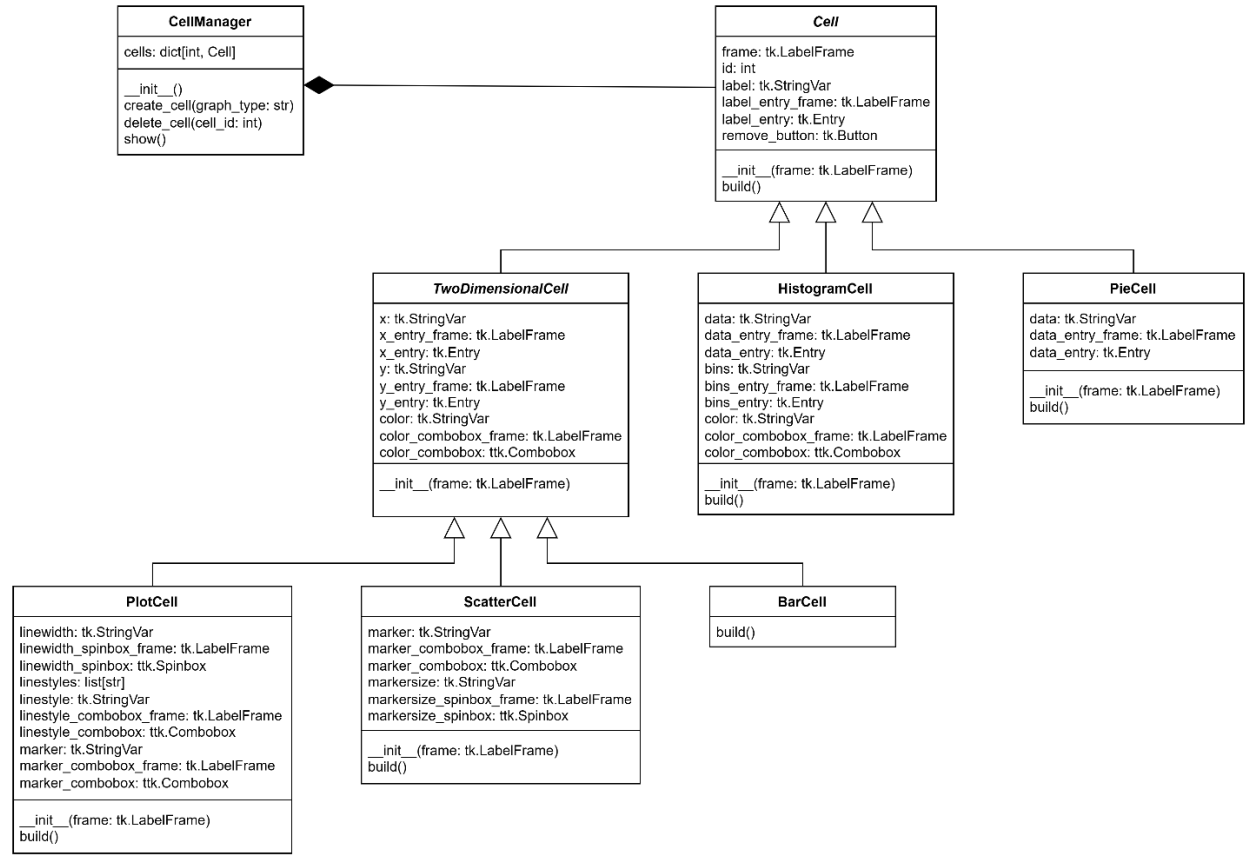
Функція get_list() використовується для обробки введених у поля значень — вона перетворює текст із полів tk.StringVar на список чисел або рядків, що робить можливим динамічне використання користувацького введення. Завдяки чіткому розподілу обов'язків між класами забезпечується модульність і можливість легко розширювати програму, наприклад, додавати нові типи графіків без зміни існуючих класів.

Для контролю правильного створення типів візуалізації є клас CellManager, який містить словник з id Cell і самим класом Cell, методи create_cell(), delete_cell() та show(). Словник потрібен для збереження всіх графіків, які були створені, методи create_cell створює графік із вказаним типом у полі graph_type типу str і блокує графіки, які не можуть бути разом на одній фігурі зі створеним, delete_cell - видаляє графік із вказаним полем cell_id типу int і розблоковує графіки, які можуть поєднуватись із графіками, які залишились, show – будує фігуру, на якій присутні всі графіки або виводить помилку, якщо виникнув вийняток.

Функція add_status_text() дозволяє добавити запис з поля text типу str у статусне вікно і також дату його додавання, що корисно для виведення помилок при побудові графіка черещ метод show().

Такий підхід забезпечує масштабованість, спрощує підтримку коду і дозволяє ефективно реалізувати взаємодію між графічним інтерфейсом та побудовою графіків у matplotlib.

UML-діаграма класів:



2.3. Опис класів, методів та атрибутів [5 балів]

Описати кожен клас, метод та атрибут, які ви подали на діаграмі класів у п. 2.2.
Для кожного класу поясніть його логіку (що саме робить цей клас, яку задачу вирішує).
Для кожного методу опишіть, що саме він виконує, які вхідні параметри та результат роботи.
Для кожного атрибута опишіть, що саме він зберігає та як використовується в класі.
Опис має бути чітким і зрозумілим, щоб можна було легко зрозуміти, як працюють класи і їхні елементи.

Клас	**Імена атрибутів та методів**	**Опис атрибутів та методів**
Cell	frame: tk.LabelFrame	Фрейм, у якому будуть розміщуватись всі елементи графіка
	id: int	Унікальний ідентифікатор класу, який потрібен для можливого видалення графіку
	label: tk.StringVar	Змінна прив’язки назви графіку
	label_entry_frame: tk.LabelFrame	Фрейм для поля введення назви графіка
	label_entry: tk.Entry	Поле введення назви графіка
	remove_button: tk.Button	Кнопка для видалення графіка (його фрейму)
	__init__(frame: tk.LabelFrame)	Конструктор, який присвоює значення фрейму в атрибут frame і створює інші атрибути
	build()	Абстрактний метод для побудови графіка
Two Dimensional Cell	x: tk.StringVar	Змінна прив’язки значень осі X графіка
	x_entry_frame: tk.LabelFrame	Фрейм для поля введення значень осі X графіка

	x_entry: tk.Entry	Поле введення значень осі X графіка
	y: tk.StringVar	Змінна прив’язки значень осі У графіка
	y_entry_frame: tk.LabelFrame	Фрейм для поля введення значень осі У графіка
	y_entry: tk.Entry	Поле введення значень осі У графіка
	color: tk.StringVar	Змінна прив’язки кольору графіка
	color_combobox_frame: tk.LabelFrame	Фрейм для поля вибору кольору графіка
	color_combobox: ttk.Combobox	Поле вибору значень кольору графіка
	__init__(frame: tk.LabelFrame)	Конструктор, який створює всі атрибути в межах заданого фрейму
PlotCell	linewidth: tk.StringVar	Змінна прив’язки ширини лінії графіка
	linewidth_spinbox_frame: tk.LabelFrame	Фрейм для поля введення ширини лінії графіка
	linewidth_spinbox: ttk.Spinbox	Поле введення ширини лінії графіка
	linestyles: list[str]	Масив, який містить всі стилі лінії для plot
	linestyle: tk.StringVar	Змінна прив’язки стилю лінії графіка
	linestyle_combobox_frame: tk.LabelFrame	Фрейм для поля вибору стилю лінії графіка
	linestyle_combobox: ttk.Combobox	Поле вибору кольору графіка
	marker: tk.StringVar	Змінна прив’язки маркерів точок графіка
	marker_combobox_frame: tk.LabelFrame	Фрейм для поля вибору маркерів точок графіка
	marker_combobox: ttk.Combobox	Поле вибору маркерів точок графіка
	__init__(frame: tk.LabelFrame)	Конструктор, який створює всі атрибути в межах заданого фрейму
	build()	Метод для побудови графіка plot
ScatterCell	marker: tk.StringVar	Змінна прив’язки маркерів точок графіка
	marker_combobox_frame: tk.LabelFrame	Фрейм для поля вибору маркерів точок графіка
	marker_combobox: ttk.Combobox	Поле вибору маркерів точок графіка
	markersize: tk.StringVar	Змінна прив’язки ширини маркерів точок графіка
	markersize_spinbox_frame: tk.LabelFrame	Фрейм для поля введення ширини маркерів точок графіка
	markersize_spinbox: ttk.Spinbox	Поле введення ширини маркерів точок графіка
	__init__(frame: tk.LabelFrame)	Конструктор, який створює всі атрибути в межах заданого фрейму
	build()	Метод для побудови графіка scatter
BarCell	build()	Метод для побудови графіка bar

Histogram Cell	data: tk.StringVar	Змінна прив’язки даних графіка
	data_entry_frame: tk.LabelFrame	Фрейм для поля введення даних графіка
	data_entry: tk.Entry	Поле введення даних графіка
	bins: tk.StringVar	Змінна прив’язки кількості корзин графіка
	bins_entry_frame: tk.LabelFrame	Фрейм для поля введення кількості корзин графіка
	bins_entry: tk.Entry	Поле введення кількості корзин графіка
	color: tk.StringVar	Змінна прив’язки кольору графіка
	color_combobox_frame: tk.LabelFrame	Фрейм для поля вибору кольору графіка
	color_combobox: ttk.Combobox	Поле вибору значень кольору графіка
	__init__(frame: tk.LabelFrame)	Конструктор, який створює всі атрибути в межах заданого фрейму
	build()	Метод для побудови графіка hist
PieCell	data: tk.StringVar	Змінна прив’язки даних графіка
	data_entry_frame: tk.LabelFrame	Фрейм для поля введення даних графіка
	data_entry: tk.Entry	Поле введення даних графіка
	__init__(frame: tk.LabelFrame)	Конструктор, який створює всі атрибути в межах заданого фрейму
	build()	Метод для побудови графіка pie
Cell Manager	cells: dict[int, Cell]	Словник, який містить id типу графіка і сам клас.
	__init__()	Конструктор, який створює атрибут cells
	create_cell(graph_type: str)	Метод, який створює тип графіку. Він створює фрейм, у якому будуть міститись налаштування графіку і клас, який відповідає зазначеному типу. Також він блокує графіки, які не можуть поєднуватись із цим типом на площині і додає до атрибуту cells.
	delete_cell(cell_id: int)	Метод, який видаляє тип графіку. Він знищує фрейм і клас, які відповідають зазначеному cell_id. Також він розблоковує графіки, які не могли поєднуватись із цим типом на площині і видаляє запис класу з атрибуту cells.
	show()	Будує площину, на якій будуть розміщуватись всі графіки із атрибуту cells, викликаючи метод build кожного класу. Також він задає фігурі додаткові налаштування, які вказані у фреймі plotting, такі як title, x- та ylabel, legend, grid. Якщо виникають помилки, то він очищує фігуру і буфер matplotlib і виводить сповіщення помилки у статусне вікно.

2.4. Перевірка та тестування програми [5 балів]

Перевірте всі методи програми, щоб переконатися, що вони повертають правильні результати за різних умов, включаючи крайні випадки. Оцініть повноту тестування, перевіривши всі методи на різні сценарії, зокрема стандартні, неочікувані та крайні. Проведіть тестування функціональності (чи виконуються завдання програми), адаптивності (чи працює програма з різними вхідними даними), зручності (чи легко користуватися програмою) та відповідності стандарту PEP 8 (чи дотримано правила форматування, таких як назви змінних, відступи та структура коду). Опишіть результати тестів, зазначивши, які методи працюють коректно, а які потребують виправлення, і поясніть зроблені виправлення. На основі тестування дайте оцінку якості програми, визначивши, чи відповідає вона заявленим вимогам і очікуванням.

Під час тестування програми були перевірені всі основні методи, включаючи стандартні, неочікувані та крайні сценарії. Для цього були розроблені юніт-тести з використанням модуля unittest та проведені ручні тестування. Тестування охоплює функціональність, адаптивність, зручність та відповідність стандарту PEP 8.

Функціональність перевірена за допомогою тестів до методів обробки введення (get_list), виведення повідомлень (add_status_text), створення та видалення графіків (create_cell, delete_cell), відображення графіків (show), обробки помилок при некоректному введенні та заборони створення графіків, досягнувши максимальний ліміт. Також було вручну протестовано віджети вікна Plotting та віджет Label у одному з графіків. Усі тести показали очікувану поведінку.

Адаптивність програми перевірялась на введення чисел з плаваючою комою, рядкових значень, порожніх рядків і зайвих ком. Функція get_list обробляє ці варіанти, повертаючи коректні списки, що свідчить про її гнучкість. Усі типи графіків (plot, scatter, bar, histogram, pie) були побудовані з правильними вхідними даними, а при неправильних — повідомлення про помилку з’являлось у статусному вікні. Це демонструє надійну валідацію даних та стійкість до помилок користувача.

З точки зору зручності, програма дозволяє легко створювати графіки через інтерфейс, а блокування пункту меню "pie" після його використання унеможливлює створення кількох кругових діаграм одночасно, що запобігає логічним помилкам. Скидання станів меню після видалення графіка реалізоване коректно, що покращує користувацький досвід.

Щодо дотримання стандарту PEP 8, була проведена перевірка за допомогою ruff із встановленим лімітом довжини рядків у 79 символів, код відповідає вимогам стандарту. Назви змінних, структура коду, форматування та відступи є коректними.

Вміст python-файлу з юніт-тестами:

```
import tkinter as tk
import unittest

import course as crs

# Блокування спливаючих вікон графіків
crs.plt.show = lambda *args, **kwargs: None

# Глобальна функція для перевірки вмісту статусного вікна
def ends_with(text: str) -> bool:
    content = crs.status_text.get("1.0", tk.END).strip()
    return content.endswith(text)

# Тести для функцій
class TestFunctions(unittest.TestCase):
    def test_get_list(self):
        self.assertEqual(crs.get_list(tk.StringVar(value="")), [])
        self.assertEqual(
            crs.get_list(tk.StringVar(value="1, 2, 3")), [1, 2, 3]
        )
        self.assertEqual(
```

```
            crs.get_list(tk.StringVar(value="1, 2, 3, ")), [1, 2, 3]
        )
        self.assertEqual(
            crs.get_list(tk.StringVar(value="1.5, 2.5, 3.5")), [1.5, 2.5, 3.5]
        )
        self.assertEqual(
            crs.get_list(tk.StringVar(value="1.5, 2.5, 3.5, ")),
            [1.5, 2.5, 3.5],
        )
        self.assertEqual(
            crs.get_list(tk.StringVar(value="A, B, C")), ["A", "B", "C"]
        )
        self.assertEqual(
            crs.get_list(tk.StringVar(value="A, B, C, ")), ["A", "B", "C"]
        )

    def test_add_status_text(self):
        text = "Hello"
        crs.add_status_text(text)
        self.assertTrue(ends_with(text))

# Тести створення та видалення графіків
class TestCellsManipulation(unittest.TestCase):
    def test_create_cell(self):
        menu = crs.graph_option_menu["menu"]
        for graph_type in crs.GRAPH_TYPES:
            crs.cell_manager.create_cell(graph_type)
            # Перевірка, що графік створено
            self.assertTrue(len(crs.cell_manager.cells) == 1)
            # Перевірка правильного блокування меню
            if graph_type == "pie":
                for i in range(menu.index("end") + 1):
                    self.assertTrue(menu.entrycget(i, "state") == "disabled")
            else:
                index = crs.GRAPH_TYPES.index("pie")
                self.assertTrue(menu.entrycget(index, "state") == "disabled")

            # Очищення графіків і скидання меню
            for i in range(menu.index("end") + 1):
                menu.entryconfig(i, state="normal")
            crs.cell_manager.cells.clear()

    def test_delete_cell(self):
        menu = crs.graph_option_menu["menu"]
        # Імітація створення графіка
        id = 0
        crs.cell_manager.cells[id] = crs.PlotCell(tk.LabelFrame())
        menu.entryconfig(id + 1, state="disabled")

        crs.cell_manager.delete_cell(id)

        # Перевірка, що графік видалено
        self.assertTrue(not crs.cell_manager.cells)
        self.assertTrue(menu.entrycget(id + 1, "state") == "normal")

        # Очищення графіків і скидання меню
        for i in range(menu.index("end") + 1):
            menu.entryconfig(i, state="normal")
        crs.cell_manager.cells.clear()

# Тести виводу графіків
class TestGraphsOutput(unittest.TestCase):
    def test_output_without_graphs(self):
        crs.cell_manager.show()
```

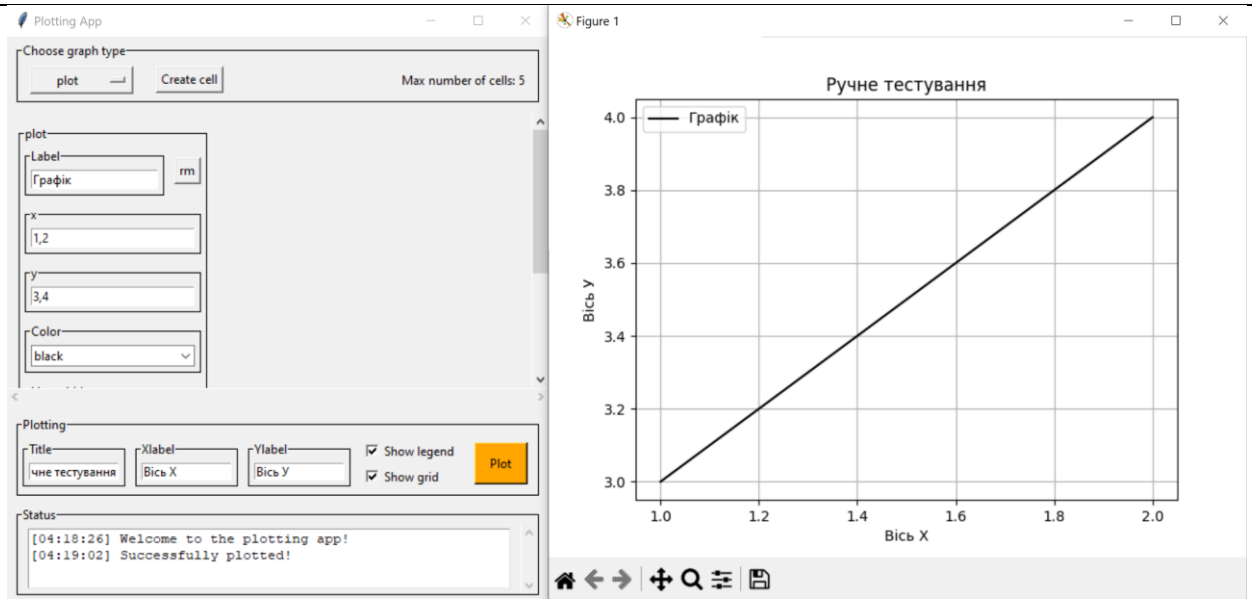
```
        self.assertTrue(ends_with("No cells to plot!"))

    def test_output_plot(self):
        crs.cell_manager.create_cell("plot")
        id, cell = next(iter(crs.cell_manager.cells.items()))
        cell.x.set("1, 2, 3")
        cell.y.set("4, 5, 6")

        crs.cell_manager.show()
        self.assertTrue(ends_with("Successfully plotted!"))
        crs.plt.close()
        crs.cell_manager.delete_cell(id)

    def test_output_scatter(self):
        crs.cell_manager.create_cell("scatter")
        id, cell = next(iter(crs.cell_manager.cells.items()))
        cell.x.set("1, 2, 3")
        cell.y.set("4, 5, 6")

        crs.cell_manager.show()
        self.assertTrue(ends_with("Successfully plotted!"))
        crs.plt.close()
        crs.cell_manager.delete_cell(id)

    def test_output_bar(self):
        crs.cell_manager.create_cell("bar")
        id, cell = next(iter(crs.cell_manager.cells.items()))
        cell.x.set("1, 2, 3")
        cell.y.set("4, 5, 6")

        crs.cell_manager.show()
        self.assertTrue(ends_with("Successfully plotted!"))
        crs.plt.close()
        crs.cell_manager.delete_cell(id)

    def test_output_hist(self):
        crs.cell_manager.create_cell("histogram")
        id, cell = next(iter(crs.cell_manager.cells.items()))
        cell.data.set("1, 1, 2")
        cell.bins.set("2")

        crs.cell_manager.show()
        self.assertTrue(ends_with("Successfully plotted!"))
        crs.plt.close()
        crs.cell_manager.delete_cell(id)

    def test_output_pie(self):
        crs.cell_manager.create_cell("pie")
        id, cell = next(iter(crs.cell_manager.cells.items()))
        cell.data.set("50, 30, 20")

        crs.cell_manager.show()
        self.assertTrue(ends_with("Successfully plotted!"))
        crs.plt.close()
        crs.cell_manager.delete_cell(id)

# Тести при вводі некоректних даних
class TestIncorrectInput(unittest.TestCase):
    def test_incorrect_input(self):
        test_data = [
            ("1, 2, 3, 4, 5, 6", "4, 5, 6"),
            ("1;2;3", "4, 5, 6"),
            ("1 / 2 / 3", "4, 5, 6"),
        ]
```

```
        for x, y in test_data:
            crs.cell_manager.create_cell("plot")
            id, value = next(iter(crs.cell_manager.cells.items()))
            value.x.set(x)
            value.y.set(y)
            crs.cell_manager.show()
            self.assertFalse(ends_with("Successfully plotted!"))
            crs.cell_manager.delete_cell(id)

# Тести фарбування рамок графіків
class TestColoring(unittest.TestCase):
    def test_color(self):
        crs.cell_manager.create_cell("plot")
        id, cell = next(iter(crs.cell_manager.cells.items()))
        cell.x.set("1, 2, 3, 4, 5, 6")
        cell.y.set("4, 5, 6")

        crs.cell_manager.show()
        self.assertTrue(cell.frame.cget("bg") == "#FFCCCC")

        cell.x.set("1, 2, 3")
        crs.cell_manager.show()
        self.assertFalse(cell.frame.cget("bg") == "#FFCCCC")
        crs.plt.close()

        crs.cell_manager.delete_cell(id)

# Тести заборони створення графіків, досягнувши максимальний ліміт
class TestMaxGraphs(unittest.TestCase):
    def test_max_graphs(self):
        for i in range(crs.MAX_CELL_NUMBER):
            crs.cell_manager.create_cell("plot")

        self.assertFalse(ends_with("Max number of cells reached!"))

        crs.cell_manager.create_cell("plot")
        self.assertTrue(ends_with("Max number of cells reached!"))

        crs.cell_manager.cells.clear()

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

Результат юніт-тестів:

```
.............
----------------------------------------------------------------------
Ran 13 tests in 1.779s

OK
```

Скріншоти ручного тестування:



Отже, усі функціональні вимоги до програми виконано. Жодні методи не потребують суттєвого виправлення. Якість програми висока: вона відповідає заявленим вимогам, очікуванням користувача, має логічну структуру, зручний інтерфейс, а також надійно обробляє виняткові ситуації.

2.5. Інструкція користувача [5 балів]

Розробіть інструкцію користувача, дотримуючись такої структури:

1. Вступ

1.1. Короткий опис програми та її мети.

Опишіть, що робить програма та для чого вона створена. Наприклад: «Програма дозволяє розраховувати статистичні показники з набору даних»..

1.2. Цільова аудиторія.

Вкажіть, хто буде використовувати програму: студенти, професіонали, звичайні користувачі або інші категорії.

2. Основні функції

2.1. Опис основних функцій програми.

Перерахуйте та поясніть основні можливості програми. Наприклад: "Розрахунок середнього арифметичного, побудова графіків».

2.2. Покрокові інструкції з використання кожної функції.

Надайте зрозумілі інструкції, як використовувати кожну функцію програми, крок за кроком.

2.3. Приклади використання програми.

Наведіть реальні приклади застосування програми. Наприклад: «Як порахувати середнє арифметичне або створити графік із вхідних даних».

3. Налаштування

3.1. Огляд доступних налаштувань програми.

Опишіть усі доступні налаштування програми, наприклад, зміну мови, налаштування кольорів інтерфейсу або шрифтів.

3.2. Як налаштувати програму.

Дайте покрокову інструкцію з налаштування програми відповідно до потреб користувача.

Інструкція повинна бути написана простою мовою, включати чіткі пояснення та, за потреби, ілюстрації або скріншоти для кращого розуміння.

1. Вступ

1.1. Короткий опис програми та її мети.

Програма дозволяє будувати графіки п’яти типів— лінійного (plot), точкового (scatter), стовпчикового (bar), гістограми (histogram) та кругової діаграми (pie) — з набору даних введених вручну. Вона не потребує знань програмування і дозволяє користувачам швидко візуалізувати числові дані у зрозумілому форматі.

1.2. Цільова аудиторія.

Програму будуть використовувати студенти та учні для виконання лабораторних робіт, досліджень та презентацій результатів; викладачі та наукові керівники для демонстрації статистичних даних, побудови графіків під час занять чи консультацій; розробники програмного забезпечення як приклад інтеграції tkinter і matplotlib або як інструмент для швидкої візуалізації даних; освітні установи для впровадження у навчальний процес інструментів візуалізації даних; неурядові організації та грантові фонди у сфері освіти і науки для створення навчальних курсів, тренінгів та аналітичних звітів з використанням візуалізації.

2. Основні функції

2.1. Опис основних функцій програми.

Побудова графіків п’яти типів: plot, scatter, bar, histogram, pie; введення даних вручну через текстові поля; додавання кількох графіків одночасно; налаштування параметрів: назви осей, заголовок, сітка, легенда; виведення стану побудови у статусне вікно; обмеження на кількість графіків для уникнення перевантаження.

2.2. Покрокові інструкції з використання кожної функції.

Загальні дії:

1. Запустіть програму.

2. Оберіть тип графіку з випадаючого меню у верхній частині вікна, у рамці “Choose graph type”.

3. Натисніть кнопку «Create cell» — знизу з'явиться рамка з налаштуваннями для вибраного графіку.

4. Заповнення полів у рамці (для кожного типу графіка):

Тип графіка	Параметр	Опис	Формат
Plot	Label	Назва серії на графіку (легенда)	Текст
	X	Значення по осі X	Числа через кому: 1, 2, 3
	Y	Значення по осі Y	Числа через кому: 4, 5, 6
	Color	Колір лінії	Вибір з меню (наприклад, red, blue)
	Line width	Товщина лінії	Десяткове число: 1.5
	Line style	Тип лінії	solid, dashed, dashdot, dotted, None
	Marker	Символ точок на лінії	Наприклад: o, x, ^, None
Scatter	Label	Назва серії	Текст
	X	Координати X точок	Числа через кому: 1, 2, 3
	Y	Координати Y точок	Числа через кому: 4, 5, 6
	Color	Колір точок	Вибір з меню
	Marker	Форма точок	Наприклад: o, s, x, ^
	Marker size	Розмір точок (у пікселях)	Десяткове число: 6.0
Bar	Label	Назва серії	Текст

	X	Категорії/мітки	Числа або текст через кому: A, B, C
	Y	Висоти стовпчиків	Числа через кому: 5, 7, 3
	Color	Колір стовпців	Вибір з меню
Histogram	Label	Назва серії	Текст
	Data	Значення для побудови гістограми	Числа через кому: 4, 5, 6, 4, 2
	Bins	Кількість інтервалів (кошиків)	Ціле число: 10 (опціонально)
	Color	Колір стовпчиків	Вибір з меню
Pie	Labels	Мітки секторів	Текст через кому: Apples, Bananas, Pears
	Data	Значення секторів (сума > 0)	Числа через кому: 30, 20, 50

5. Налаштування остаточного вигляду графіку.

Перейдіть до нижньої частини інтерфейсу, рамки “Plotting”. Там розміщено поля:

- Заголовок графіку (“Title” - тип Текст)
- Назви осей X і Y (“Xlabel”, “Ylabel” - тип Текст)
- Опції: відображення легенди (“Show legend”), сітки (“Show grid”)
- Кнопка для побудови графіку (“Plot”)

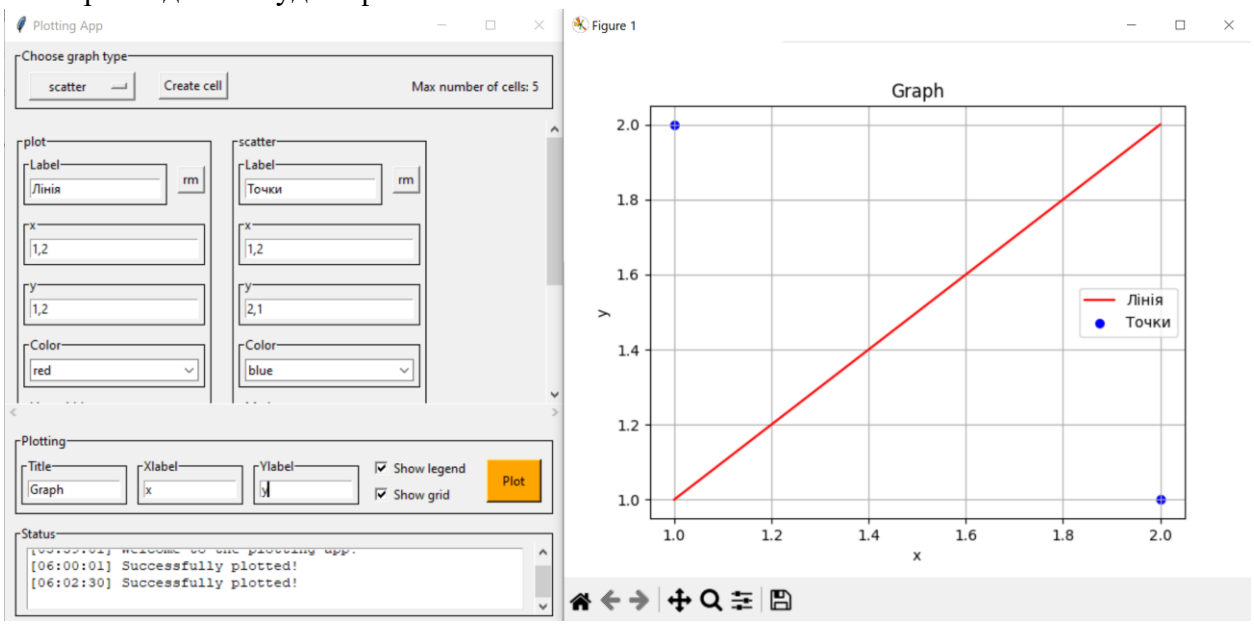
Налаштуйте остаточний вигляд графіку (якщо потрібно) та нажміть кнопку “Plot”.

6. Перевірка результату.

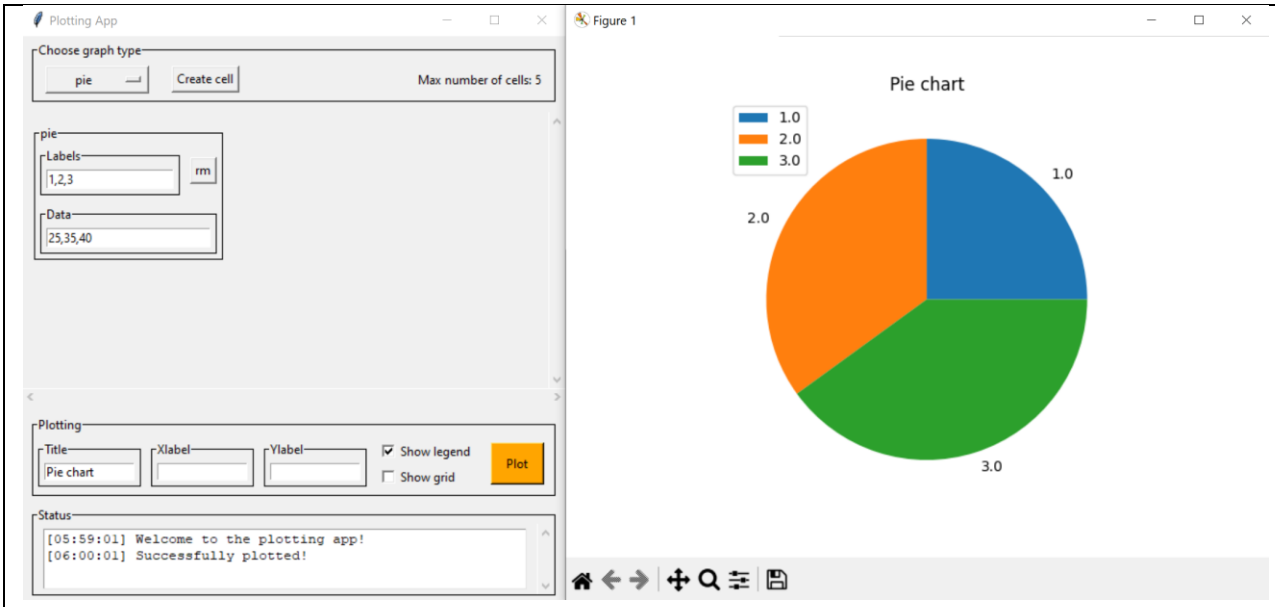
- Якщо все введено коректно, у статусному вікні з’явиться повідомлення “Successfully plotted!”.
- Якщо є помилки, наприклад, некоректні дані у певній рамці, з’явиться “Error in red cell ###: {текстовт повідомлення помилки}”
- Якщо відсутні рамки з налаштуваннями графіків - "No cells to plot!"
- Якщо ви додали максимальну кількість графіків - "Max number of cells reached!"

2.3. Приклади використання програми.

Приклад 1: Побудова plot і scatter



Приклад 2: Побудова кругової діаграми



3. Налаштування

3.1. Огляд доступних налаштувань програми.

У програмі встановлено обмеження на кількість одночасно створених графіків. Ви можете додати не більше 5 графіків одночасно. Якщо спробуєте додати більше, виводиться повідомлення в статусне вікно, що максимальна кількість досягнута, і новий графік не створиться.

3.2. Як налаштувати програму.

Змінити кількість дозволених графіків через графічний інтерфейс не можна. Щоб отримати дозвіл на безмежну кількість графіків, потрібно придбати преміум версію додатку.

Розділ 3. Аналіз результатів роботи [15 балів]

3.1. Демонстрація виконання програми [5 балів]

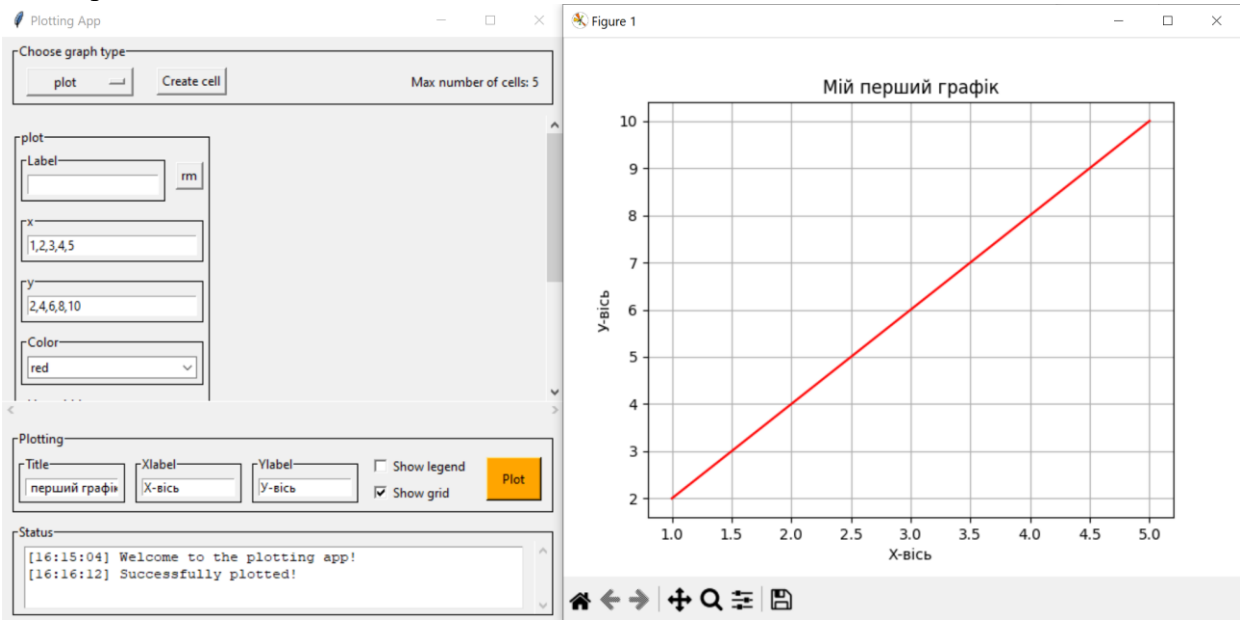
Показати приклади реального використання вашої програми:

- подайте скріншоти екрана, де видно, як програма працює,
- опишіть, які операції ви виконували під час демонстрації та які результати отримали,
- поясніть, як саме програма виконувала кожен крок і що отримали в результаті.

Скріншоти мають бути чіткими, з видимим інтерфейсом програми та результатами її роботи.
Опис має бути зрозумілим і детальним, щоб показати, як програма працює на практиці.

Приклад 1: Побудова лінійного графіка

Скріншот 1:



Опис дій:

- У головному меню обрано тип графіка plot (лінійний).
- Натиснуто кнопку «Додати графік» — з’явилась нова рамка для введення даних.
- У відповідні поля введено X-значення: 1, 2, 3, 4, 5 та Y-значення: 2, 4, 6, 8, 10.
- Задано заголовок графіка: "Мій перший графік", назви осей: "X-вісь" і "Y-вісь", та увімкнено відображення сітки.
- Натиснуто кнопку "Побудувати графік".

Результат:

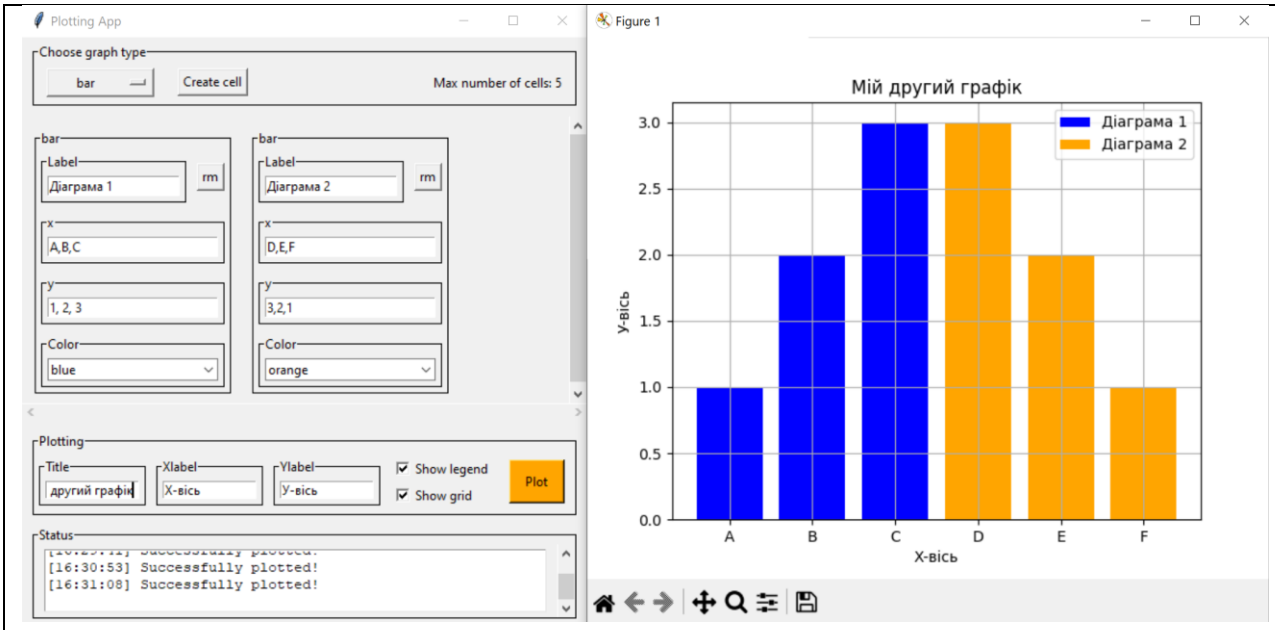
У вікні з’явився лінійний графік з правильно відображеними даними, підписаними осями та заголовком. У статусному вікні з’явилося повідомлення: "Успішно побудовано".

Виконання програми:

При натисканні кнопки “Додати графік” програма передавала вибраний тип графіку класу, який призначений для створення, видалення і будування остаточного графіку, CellManager, і він створив рамку відповідного до графіку класу із налаштуваннями. Далі після налаштування графіку, після натисканну кнопки “Побудувати графік”, цей самий клас почав будувати графік, вивів повідомлення “Успішно побудовано” у статусне вікно і сам графік.

Приклад 2: Побудова гістограми

Скріншот 2:



Опис дій:

- У головному меню обрано тип графіка bar (стовпчаста діаграма).

- Двічі натиснуто кнопку «Додати графік» — з’явились 2 нові рамки для введення даних.

- У відповідні поля першої рамки введено X-значення: A, B, C та Y-значення: 1, 2, 3, а у відповідні поля другої рамки – D, E, F та 3, 2, 1. Також у кожній рамці у поле Label було записано відповідно “Діаграма 1” та “Діаграма 2”

- Задано заголовок графіка: "Мій другий графік", назви осей: "X-вісь" і "Y-вісь", та увімкнено відображення легенди та сітки.

- Натиснуто кнопку "Побудувати графік".

Результат:

У вікні з’явилось графік з правильно відображеними даними, підписаними осями та заголовком. У статусному вікні з’явилося повідомлення: "Успішно побудовано".

Виконання програми:

При подвійному натисканні кнопки “Додати графік” програма двічі передавала вибраний тип графіку класу, який призначений для створення, видалення і будування остаточного графіку, CellManager, і він створив дві рамки відповідного до графіку класу із налаштуваннями. Далі після налаштування графіку, після натисканну кнопки “Побудувати графік”, цей самий клас почав будувати графік, викликавши метод build() з кожного класу графіків, вивів повідомлення “Успішно побудовано” у статусне вікно і сам графік.

3.2. Оцінка досягнення поставлених цілей [5 балів]

Пояснити, як результати вашої роботи відповідають цілям та потребам зацікавлених сторін, визначених у п. 1.2.
Подати якісну оцінку ступеня задоволення зацікавлених сторін. Це означає, що потрібно описати, чи були досягнуті поставлені цілі і чи задоволені вимоги, які ставили ці сторони. Оцінка має бути обґрунтованою, з конкретними прикладами того, як ваші результати відповідають вимогам.
Запропонуйте напрямки покращення програми.

Назва зацікавленої сторони	**Опис та оцінка досягнення цілей та потреб зацікавленої сторони**
Студенти та учні	Програма повністю задовольняє потребу студентів у простому інструменті для побудови графіків. Вона не вимагає знання

	програмування, має інтуїтивний інтерфейс і підтримує декілька типів графіків, що підходить для курсових і лабораторних робіт. Приклад: студент може побудувати гістограму за власними даними в декілька кліків. Ціль досягнута повністю.
Викладачі та наукові керівники	Викладачі отримують інструмент, що дозволяє студентам самостійно й без помилок будувати графіки. Це зменшує навантаження на викладача та покращує якість звітів. Програма також дозволяє швидко перевірити коректність побудови графіків. Однак наразі немає функції автоматичної перевірки типу графіка чи підписів осей. Потреба задоволена на високому рівні.
Розробники програмного забезпечення	Потенціал реалізовано як приклад пропрієтарного рішення, тому для доступу до коду потрібен дозвіл. Можна розширити функціонал, інтегрувати з веб або додати підтримку Excel/CSV. Наразі є лише графічний інтерфейс на tkinter, але є можливість переробки класів. Ціль задоволена частково.
Освітні установи	Програма відповідає вимогам освітніх установ щодо простоти, доступності та практичної користі. Її можна інтегрувати в навчальні курси без потреби змінювати існуючі програми. Документація доступна, однак інтерфейс програми наразі англійською мовою, тому бажаним покращенням є впровадження багатомовної підтримки. Потреба частково задоволена.
Неурядові організації або грантові фонди	Програма демонструє потенціал для підвищення цифрової грамотності та доступності інструментів для навчання. Вона може бути використана у соціальних проєктах та освітніх програмах. Проте наразі не має офіційної підтримки або ліцензії, що ускладнює її широке впровадження. Потреба частково задоволена.

3.3. SWOT-аналіз результатів розробки [5 балів]

Виконайте SWOT-аналіз ваших результатів. Такий аналіз повинен оцінити:
S (Strengths) – сильні сторони (переваги вашої роботи).
W (Weaknesses) – слабкі сторони (недоліки або обмеження вашої роботи).
O (Opportunities) – можливості для покращення (як можна вдосконалити роботу або розширити її можливості).
T (Threats) – загрози (ризики або проблеми, які можуть виникнути в майбутньому).
Під час аналізу використати оцінку ступеня задоволеності зацікавлених сторін, яку ви описали в п. 3.2.
Сильні та слабкі сторони повинні бути конкретними і базуватися на реальних результатах.
Можливості та загрози мають вказувати на те, як можна покращити роботу чи які проблеми можуть виникнути в майбутньому. Оцінка повинна бути обґрунтованою та логічною.

Зацікавлені сторони, які задіяні у проблемній ситуації БЕЗПОСЕРЕДНЬО	**Зацікавлені сторони, які задіяні у проблемній ситуації ОПОСЕРЕДКОВАНО**
Сильні сторони	**Можливості**
Програма вирізняється простотою та доступністю, адже не вимагає знань програмування, що повністю відповідає очікуванням як студентів, так і викладачів. Інтуїтивний інтерфейс дозволяє швидко будувати графіки, що особливо цінно під час підготовки звітів. Рішення має чіткий освітній фокус і добре інтегрується в	Подальший розвиток програми може включати інтеграцію з Excel або CSV-файлами, що значно підвищить зручність та розширить цільову аудиторію, зокрема серед дослідників і аналітиків. Локалізація інтерфейсу на кілька мов дозволить залучити користувачів з інших країн. Розширення функціоналу – використання

навчальний процес, відповідаючи цілям і стандартам освітніх установ. Крім того, позитивні відгуки основних користувачів свідчать про високе задоволення потреб студентів і викладачів.	всієї функціональності графіків, покращення обробника помилок, додавання автоматичного підбору типу графіка, шаблонів тощо – зробить програму ще більш корисною. Водночас існує можливість укладання партнерств з освітніми закладами чи грантодавцями, що відкриває шлях до фінансування або офіційного впровадження.
Слабкі сторони	**Загрози**
Програма має певні обмеження у функціональності: не до кінця використовується вся функціональність деяких графіків; коли виводяться помилки в статусне вікно, то не завжди вказується поле рамки, у якому була помилка; відсутня підтримка імпорту даних з форматів Excel/CSV. Також наразі інтерфейс доступний лише однією мовою, що обмежує ширше застосування. Ще одним недоліком є закритий код – відсутність open-source статусу знижує потенціал для участі сторонніх розробників і спільноти.	Серйозною загрозою є конкуренція з боку потужніших продуктів, таких як Google Charts, Desmos, Microsoft Excel або інші онлайн-конструктори, які можуть витіснити програму у разі недостатнього розвитку. Низька залученість розробників, особливо без підтримки open-source, може призвести до зупинки подальшого вдосконалення. Технічні бар’єри в деяких освітніх установах, що не дозволяють встановлювати стороннє програмне забезпечення, також можуть заважати впровадженню. Крім того, відсутність регулярних оновлень або технічної підтримки знижує актуальність продукту з часом.

Додаток А. Самоаналіз оцінювання роботи [10 балів]

Структурний елемент пояснювальної записки	Максимальна оцінка	Самооцінка
1.1. Визначення проблемної ситуації	5	5
1.2. Потреби та цілі зацікавлених сторін	5	5
1.3. Формулювання мети роботи	5	5
1.4. Аналіз існуючих рішень та огляд літератури	5	5
2.1. Програмний код мовою Python	5	5
2.2. Архітектура програми	5	5
2.3. Опис класів, методів та атрибутів	5	5
2.4. Перевірка та тестування програми	5	4
2.5. Інструкція користувача	5	4
3.1. Демонстрація виконання програми	5	5
3.2. Оцінка досягнення поставлених цілей	5	5
3.3. SWOT-аналіз результатів розробки	5	5
Додаток А. Самоаналіз оцінювання роботи	10	9
Всього:	**70**	**67**