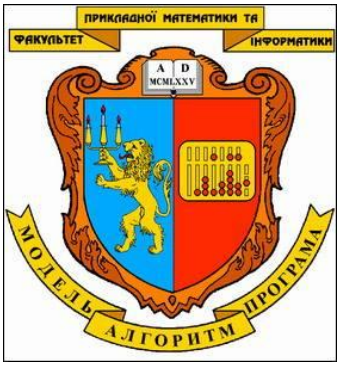
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА
Факультет прикладної математики та інформатики
Кафедра програмування




# ЛАБОРАТОРНА РОБОТА № 2

Виконала:
студентка групи ПМОм-11
Кравець Ольга

Перевірив:
Яцик І. М.

Львів 2024

**Тема:** Розгляд інтерфейсів користувача кожного типу. Визначення характеристик ергономіки до відповідних систем.

**Мета:** Ознайомитись з інтерфейсами користувача кожного типу. Навчитися будувати та порівнювати типи інтерфейсів

**Завдання:** Порівняти чотири типи інтерфейсів на конкретному прикладі. Для прикладу можна розробити користувацький інтерфейс програми побудови графіків або виведення таблиці функцій.

## Хід роботи

**Типи інтерфейсів:**

- примітивний;
- інтерфейс-меню;
- інтерфейс з вільною навігацією(GUI);
- об’єктно-орієнтований інтерфейс.

**Примітивний інтерфейс -** інтерфейс, що організує взаємодію з користувачем у консольному режимі. Користувач взаємодіє з програмою через командний рядок. Введення команди здійснюється вручну через текстовий інтерфейс.

*Переваги:*

- проста реалізація;
- мінімальні вимоги до ресурсів комп’ютера.

*Недоліки:*

- неінтуїтивний інтерфейс для користувача;
- висока ймовірність помилок при введенні команд;
- відсутність візуальних підказок.

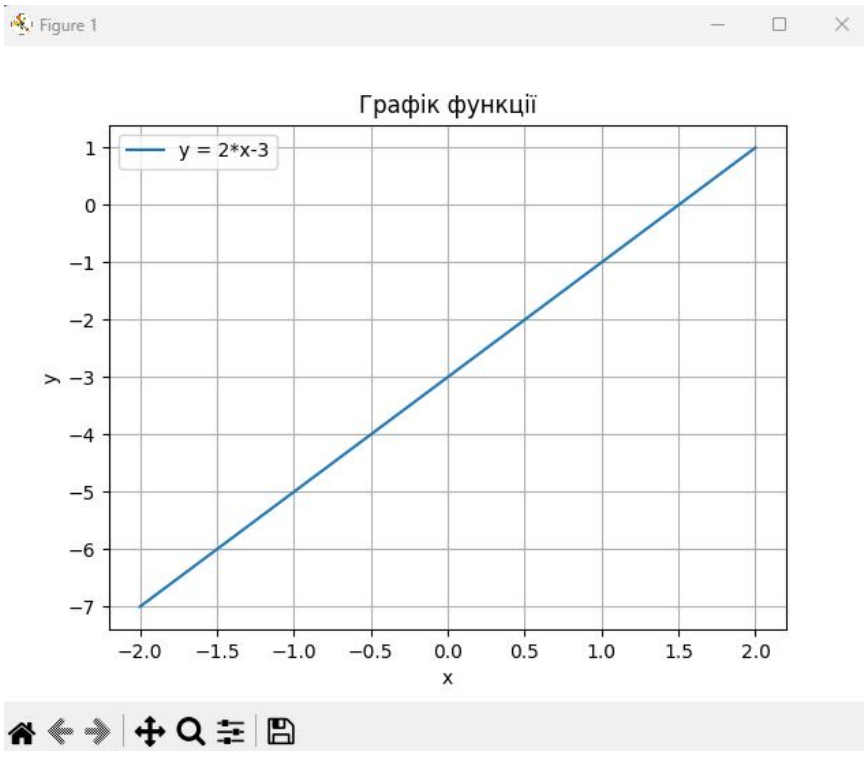
*Приклад*: користувач вручну вводить параметри для побудови графіка функції. На їх основі програма будує його.

Реалізація мовою Python з використанням бібліотеки matplotlib для візуалізації графіка:

example1.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt

def main():
    func_input = input("Введіть функцію у форматі '2*x + 1': ")
    x_range_input = input("Введіть діапазон для x у форматі '-5 5': ")

    x_min, x_max = map(float, x_range_input.split())

    x = np.linspace(x_min, x_max, 400)

    try:
        y = eval(func_input)
    except Exception as e:
        print(f"Помилка у формулі: {e}")
        return

    plt.plot(x, y, label=f'y = {func_input}')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Графік функції')
    plt.legend()

    plt.grid(True)
    plt.show()

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

```
PS D:\Навчання\Магістратура\Ергономіка ПЗ\Практичні\02> & C
іка ПЗ/Практичні/02/example1.py"
Введіть функцію у форматі '2*x + 1': 2*x-3
Введіть діапазон для x у форматі '-5 5': -2 2
```



**Інтерфейс-меню** - дозволяє користувачеві вибирати необхідні операції зі спеціального списку, виведеного йому програмою. Ці інтерфейси припускають реалізацію множини сценаріїв роботи, послідовність дій у які визначається користувачем.

*Переваги:*

- інтуїтивно зрозумілий інтерфейс;
- менший ризик помилок порівняно з примітивним інтерфейсом;
- послідовність дій веде користувача через процес побудови графіка.

*Недоліки:*

- обмеженість варіантів навігації;
- може бути незручним для досвідчених користувачів, які хочуть швидкого доступу до функцій.

*Приклад*:

1. користувач вибирає 1, щоб ввести формулу;
2. вибирає 2, щоб встановити діапазон для x;
3. вибирає 3, щоб побудувати графік на основі введених даних;
4. при необхідності виходить з програми, вибравши 4.

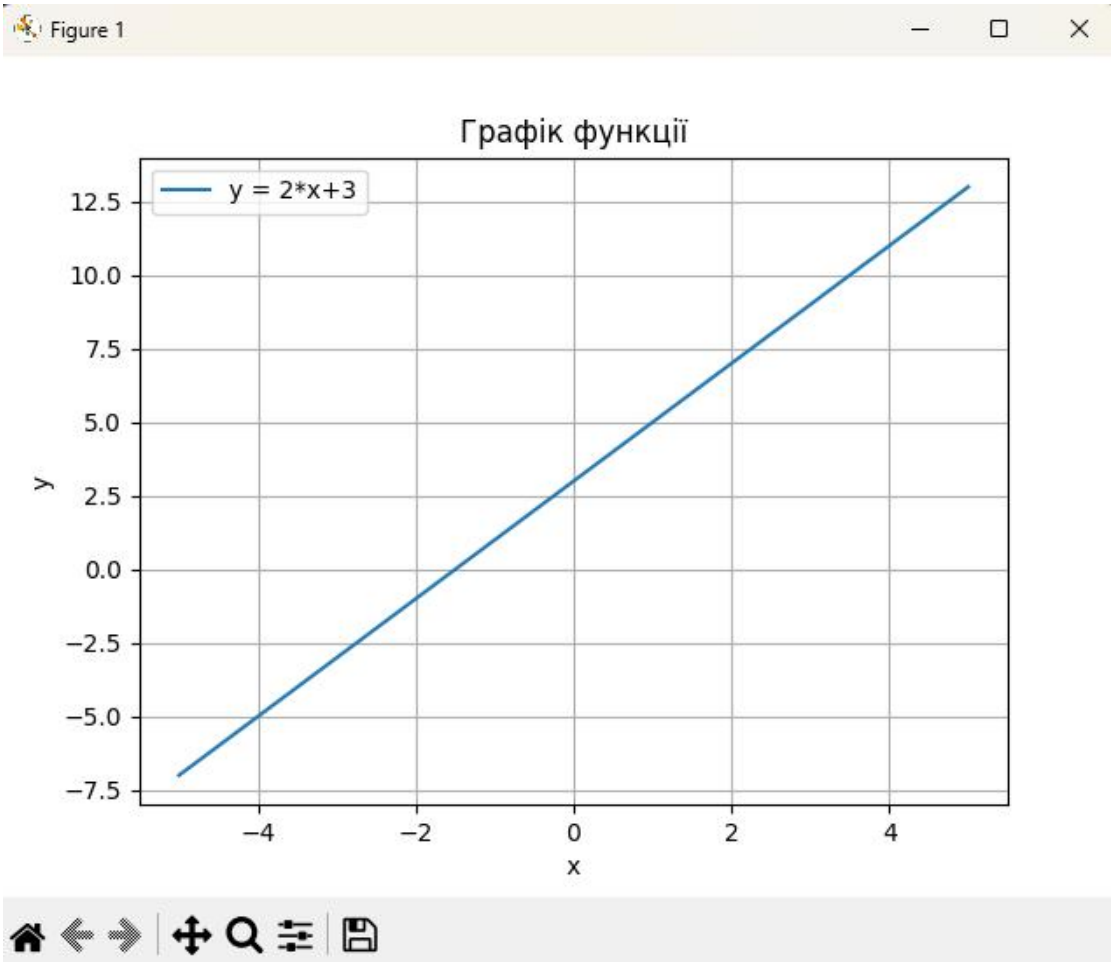
Реалізація мовою Python з використанням бібліотеки matplotlib для візуалізації графіка:

example2.py × example1.py

example2.py > build_plot

```python
import numpy as np
import matplotlib.pyplot as plt

def get_function_input():
    func_input = input("Введіть функцію у форматі '2*x + 1': ")
    return func_input

def get_x_range_input():
    x_range_input = input("Введіть діапазон для x у форматі '-5 5': ")
    x_min, x_max = map(float, x_range_input.split())
    return x_min, x_max

def build_plot(func_input, x_min, x_max):
    x = np.linspace(x_min, x_max, 400)
    try:
        y = eval(func_input)
    except Exception as e:
        print(f"Помилка у формулі: {e}")
        return

    plt.plot(x, y, label=f'y = {func_input}')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Графік функції')
    plt.legend()
    plt.grid(True)
    plt.show()

def menu():
    func_input = None
    x_min = x_max = None

    while True:
        print("\nМеню:")
        print("1. Ввести формулу")
        print("2. Вибрати діапазон для x")
        print("3. Побудувати графік")
        print("4. Вийти")

        choice = input("Зробіть вибір: ")

        if choice == '1':
            func_input = get_function_input()
        elif choice == '2':
            x_min, x_max = get_x_range_input()
        elif choice == '3':
            if func_input and x_min is not None and x_max is not None:
                build_plot(func_input, x_min, x_max)
            else:
                print("Спочатку введіть формулу і діапазон для x.")
        elif choice == '4':
            print("Вихід з програми.")
            break
        else:
            print("Невірний вибір. Спробуйте ще раз.")

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

```
PS D:\Навчання\Магістратура\Ергономіка ПЗ\Практичн
іка ПЗ/Практичні/02/example2.py"

Меню:
1. Ввести формулу
2. Вибрати діапазон для x
3. Побудувати графік
4. Вийти
Зробіть вибір: 1
Введіть функцію у форматі '2*x + 1': 2*x+3

Меню:
1. Ввести формулу
2. Вибрати діапазон для x
3. Побудувати графік
4. Вийти
Зробіть вибір: 2
Введіть діапазон для x у форматі '-5 5': -4 4

Меню:
1. Ввести формулу
2. Вибрати діапазон для x
3. Побудувати графік
4. Вийти
Зробіть вибір: 3
```



**Інтерфейс з вільною навігацією** також називають **графічним користувальницьким інтерфейсом** (**GUI** - Graphic User Interface) або **інтерфейсом WYSIWYG** (What You See Is What You Get - що бачиш, то та одержиш, тобто, що користувач бачить на екрані, те він і одержить при роздруку). Інтерфейс даного типу орієнтований на використання екрана в графічному режимі з високою розподільною здатністю. Користувач взаємодіє з елементами на екрані, такими як кнопки, поля для введення, випадаючі меню, іконки тощо

*Переваги:*

- інтуїтивно зрозумілий для користувачів різного рівня;
- багато можливостей для взаємодії з програмою;
- швидкий доступ до всіх функцій.

*Недоліки:*

- складність реалізації;
- вимоги до ресурсів комп’ютера.

*Приклад*:

програма містить:

- поле для введення формули;
- повзунки для вибору діапазону x;
- кнопку “Побудувати графік”;
- відображення результату з можливістю зміни кольору та стилю лінії.

Реалізація мовою Python з використанням бібліотеки tkinter для створення GUI та matplotlib для побудови графіка:

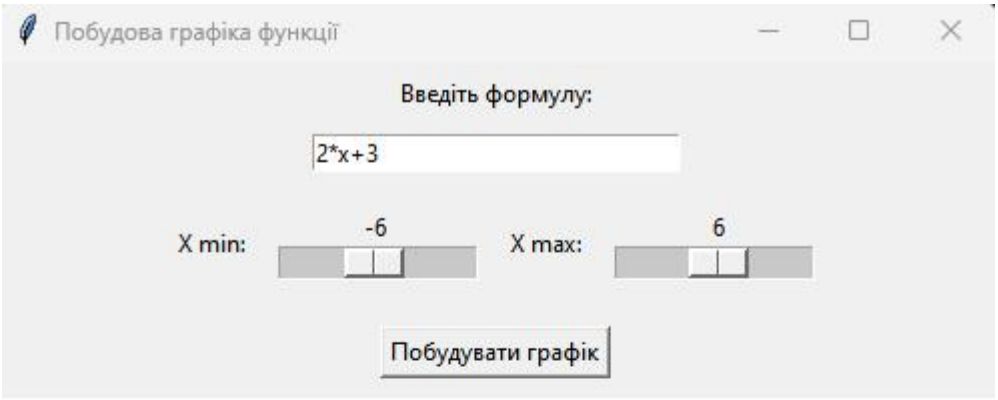
example3.py × example2.py example1.py

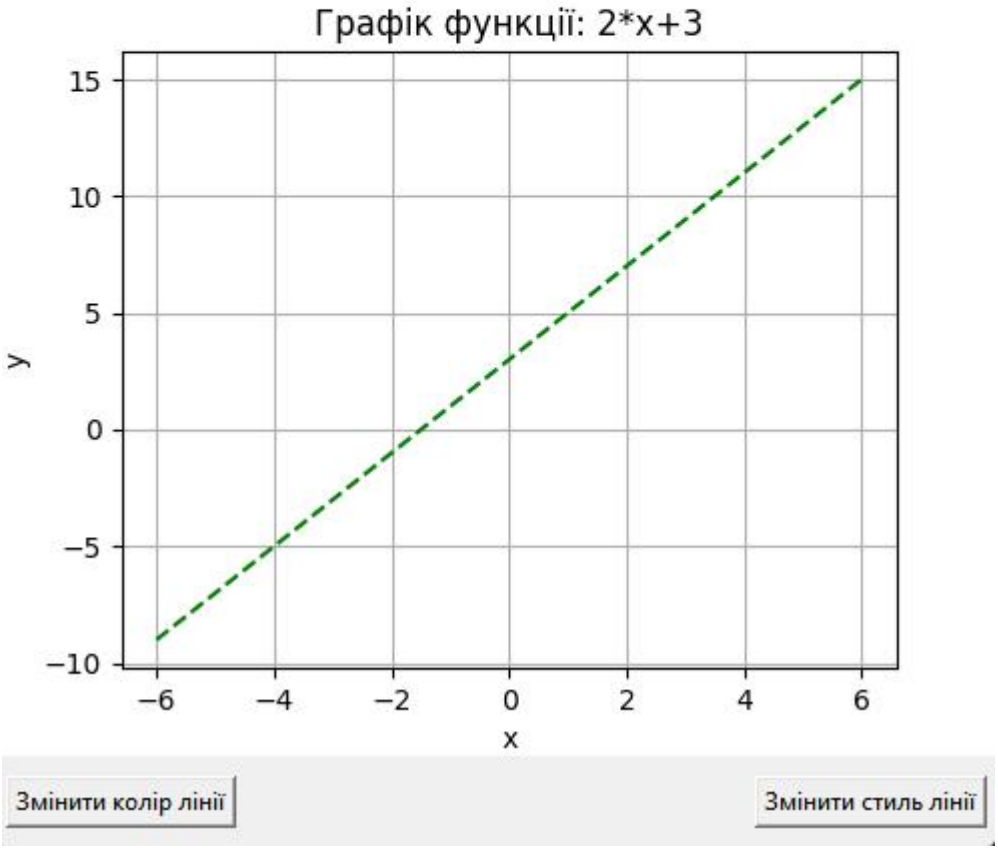
example3.py > GraphApp > change_color

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
from tkinter import ttk

class GraphApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Побудова графіка функції")

        self.formula_label = tk.Label(root, text="Введіть формулу:")
        self.formula_label.pack(pady=5)

        self.formula_entry = tk.Entry(root, width=30)
        self.formula_entry.pack(pady=5)

        self.range_frame = tk.Frame(root)
        self.range_frame.pack(pady=10)

        self.xmin_label = tk.Label(self.range_frame, text="X min:")
        self.xmin_label.grid(row=0, column=0, padx=5)

        self.xmin_slider = tk.Scale(self.range_frame, from_=-100, to=100, orient=tk.HORIZONTAL)
        self.xmin_slider.set(-10)
        self.xmin_slider.grid(row=0, column=1, padx=5)

        self.xmax_label = tk.Label(self.range_frame, text="X max:")
        self.xmax_label.grid(row=0, column=2, padx=5)

        self.xmax_slider = tk.Scale(self.range_frame, from_=-100, to=100, orient=tk.HORIZONTAL)
        self.xmax_slider.set(10)
        self.xmax_slider.grid(row=0, column=3, padx=5)

        self.plot_button = tk.Button(root, text="Побудувати графік", command=self.plot_graph)
        self.plot_button.pack(pady=10)

        self.figure, self.ax = plt.subplots(figsize=(5, 4))
        self.canvas = FigureCanvasTkAgg(self.figure, master=root)
        self.canvas.get_tk_widget().pack()

        self.color_button = tk.Button(root, text="Змінити колір лінії", command=self.change_color)
        self.color_button.pack(side=tk.LEFT, padx=5, pady=10)

        self.style_button = tk.Button(root, text="Змінити стиль лінії", command=self.change_style)
        self.style_button.pack(side=tk.RIGHT, padx=5, pady=10)

        self.line_color = 'blue'
        self.line_style = '-'

    def plot_graph(self):
        self.ax.clear()

        formula = self.formula_entry.get()
        x_min = self.xmin_slider.get()
        x_max = self.xmax_slider.get()
```

```
57
58          x = np.linspace(x_min, x_max, 400)
59          try:
60              y = eval(formula)
61              self.ax.plot(x, y, color=self.line_color, linestyle=self.line_style)
62              self.ax.set_title(f'Графік функції: {formula}')
63              self.ax.set_xlabel('x')
64              self.ax.set_ylabel('y')
65              self.ax.grid(True)
66
67              self.canvas.draw()
68          except Exception as e:
69              tk.messagebox.showerror("Помилка", f"Помилка у формулі: {e}")
70
71      def change_color(self):
72          colors = ['blue', 'red', 'green', 'purple', 'orange']
73          current_color_index = colors.index(self.line_color)
74          new_color_index = (current_color_index + 1) % len(colors)
75          self.line_color = colors[new_color_index]
76          self.plot_graph()
77
78      def change_style(self):
79          styles = ['-', '--', '-.', ':']
80          current_style_index = styles.index(self.line_style)
81          new_style_index = (current_style_index + 1) % len(styles)
82          self.line_style = styles[new_style_index]
83          self.plot_graph()
84
```

```
85  if __name__ == "__main__":
86      root = tk.Tk()
87      app = GraphApp(root)
88      root.mainloop()
89
```


Побудова графіка функції
Введіть формулу:
2*x+3
X min:
-6
X max:
6
Побудувати графік


Графік функції: 2*x+3
15
10
5
0
−5
−10
y
−6
−4
−2
0
2
4
6
x


Змінити колір лінії
Змінити стиль лінії

**Об’єктно-орієнтовані інтерфейси -** представлені інтерфейсом прямого маніпулювання. Користувач взаємодіє з програмою шляхом маніпуляції об'єктами на екрані. Наприклад, переміщення елементів, зміна їхніх параметрів за допомогою миші.

*Переваги:*

- інтуїтивний і зрозумілий інтерфейс;
- дозволяє виконувати складні операції простими діями;
- зручний для роботи з великою кількістю об'єктів та даних.

*Недоліки:*

- найскладніший в реалізації;
- вимоги до високої потужності комп’ютера;
- може бути перевантажений для користувачів, які не знайомі з концепцією прямого маніпулювання об'єктами

*Приклад*:

- користувач може перетягувати осі графіка для зміни масштабу;
- користувач може редагувати функцію безпосередньо на графіку;
- можливість додавання кількох графіків для порівняння шляхом перетягування об'єктів.

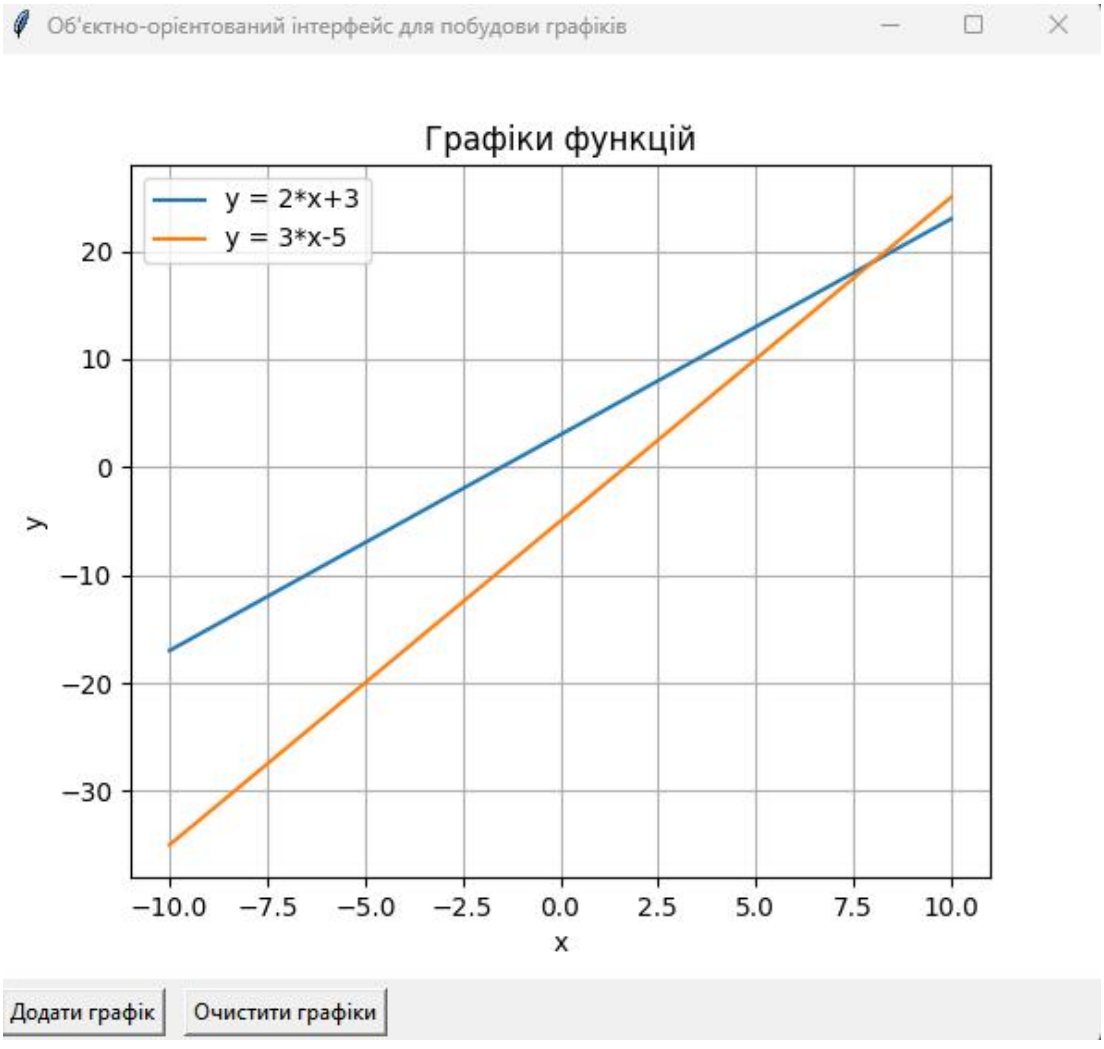
Реалізація мовою Python з використанням бібліотеки tkinter для створення GUI та matplotlib для побудови графіка:

example4.py

example4.py > GraphApp > on_drag

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
from tkinter import simpledialog, messagebox

class GraphApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Об'єктно-орієнтований інтерфейс для побудови графіків")

        self.figure, self.ax = plt.subplots(figsize=(6, 5))
        self.canvas = FigureCanvasTkAgg(self.figure, master=root)
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=1)

        self.graphs = []
        self.dragging = False

        self.add_button = tk.Button(root, text="Додати графік", command=self.add_graph)
        self.add_button.pack(side=tk.LEFT, padx=5, pady=5)

        self.clear_button = tk.Button(root, text="Очистити графіки", command=self.clear_graphs)
        self.clear_button.pack(side=tk.LEFT, padx=5, pady=5)

        self.canvas.mpl_connect('button_press_event', self.on_click)
        self.canvas.mpl_connect('motion_notify_event', self.on_drag)

    def add_graph(self):
        formula = simpledialog.askstring("Введіть функцію", "Введіть формулу у форматі '2*x + 3':")
        if formula:
            self.graphs.append(formula)
            self.plot_graphs()

    def plot_graphs(self):
        self.ax.clear()
        x = np.linspace(-10, 10, 400)
        for formula in self.graphs:
            try:
                y = eval(formula)
                self.ax.plot(x, y, label=f'y = {formula}')
            except Exception as e:
                messagebox.showerror("Помилка", f"Помилка у формулі: {e}")

        self.ax.set_title('Графіки функцій')
        self.ax.set_xlabel('x')
        self.ax.set_ylabel('y')
        self.ax.legend()
        self.ax.grid(True)
        self.canvas.draw()

    def clear_graphs(self):
        self.graphs = []
        self.ax.clear()
        self.canvas.draw()
```

```
56      def on_click(self, event):
57          if event.inaxes:
58              self.start_x, self.start_y = event.xdata, event.ydata
59              self.dragging = True
60
61      def on_drag(self, event):
62          if self.dragging and event.inaxes:
63              dx = event.xdata - self.start_x
64              dy = event.ydata - self.start_y
65              self.start_x, self.start_y = event.xdata, event.ydata
66
67              xlim = self.ax.get_xlim()
68              ylim = self.ax.get_ylim()
69              self.ax.set_xlim(xlim[0] - dx, xlim[1] - dx)
70              self.ax.set_ylim(ylim[0] - dy, ylim[1] - dy)
71              self.canvas.draw()
72
73          if event.button == 1 and not event.inaxes:
74              self.dragging = False
75
76  if __name__ == "__main__":
77      root = tk.Tk()
78      app = GraphApp(root)
79      root.mainloop()
```



| Тип інтерфейсу | Зручність використання | Складність реалізації | Вимоги до ресурсів |
|---|---|---|---|
| примітивний | 1 | 1 | 1 |
| меню | 1 | 1 | 1 |
| з вільною навігацією(GUI) | 3 | 2 | 3 |
| об’єктно-орієнтований | 4 | 4 | 4 |

*Заувага:*

1 - низький рівень

2 - середній рівень

3 - високий рівень

4 - дуже високий рівень

**Висновки:** на цій лабораторної роботи я порівнювала чотири типи інтерфейсів на конкретному прикладі. Для прикладу розробила користувацький інтерфейс програми побудови графіків на Python з використанням допоміжних бібліотек. Для простих програм, де важливі швидкість та мінімальні вимоги до ресурсів, підходить примітивний або інтерфейс-меню. Однак, для більш складних і візуально орієнтованих задач, таких як побудова графіків, найкраще підходить графічний інтерфейс з вільною навігацією (GUI) або об’єктно-орієнтований інтерфейс, особливо якщо потрібно забезпечити максимальну інтерактивність та зручність для користувача.

Ставлю вподобайку цій лабораторній.

**Список використаних джерел:**


1. Лекція 4. Види інтерфейсів користувача. - Харківський національний економічний університет імені Семена Кузнеця.
2. Навчальний посібник “Інтерфейс користувач-комп’ютер” - Вінницький державний технічний університет.
3. Python GUI's With TKinter - Codemy.com.