Комп’ютерний практикум 3. Застосунок із GUI на Python для зображення вимірювальної інформації

План роботи

1. Приклади застосунків
2. Індивідуальні завдання

1. Приклади застосунків

1.1. Моделювання вимірювальної інформації

Змоделюємо та зобразимо на графіку змінну у часі значення деякої фізичної величини періодичний сигнал синусоїдальної форми.

Лістинг 1

```
import numpy as np

# sample count
n = 1000

# measurement duration time
t_meas = 2  # s
t_start = 0  # s
dt = 0.001

n = (t_meas - t_start) / dt

# time samples
t = np.arange(t_start, t_meas, dt)

# signal settings
amp = 1.
f = 1.  # Hz
phi = 0.
y = amp * np.sin(2 * np.pi * f * t + phi)

print(f'n = {n}')
print(f't = \n{t}')
print(f'y = \n{y}')
```

Результат виконання скрипту у терміналі показана на рис. 1.

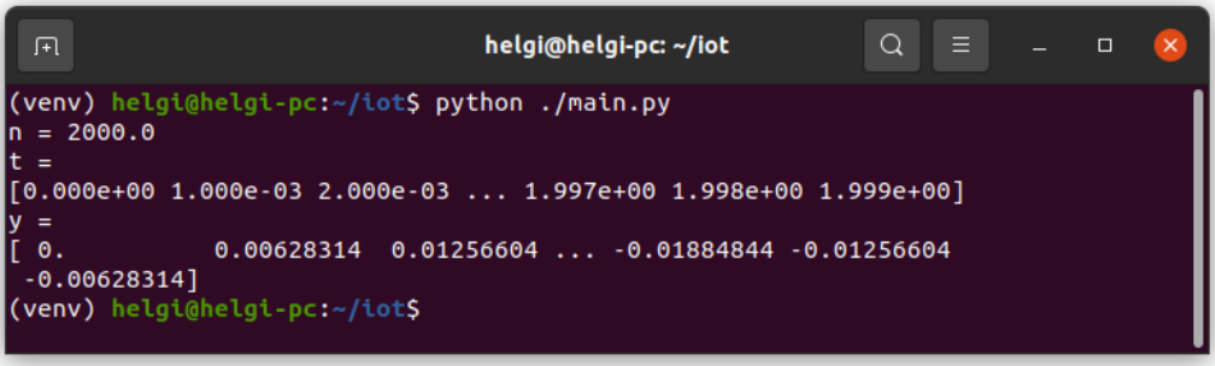

Рис. 1

1.2. Відображення змодельованої інформації

Лістинг 2

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

# samle count
n = 1000

# measurement duration time
t_meas = 2  # s
t_start = 0  # s
dt = 0.001

n = (t_meas - t_start) / dt

# time samples
t = np.arange(t_start, t_meas, dt)

# signal settings
amp = 1.
f = 1.  # Hz
phi = 0.
y = amp * np.sin(2 * np.pi * f * t + phi)

_, ax = plt.subplots()
ax.plot(t, y)
ax.set_xlabel('t, s')
ax.set_ylabel('y')
ax.grid()
plt.show()
```

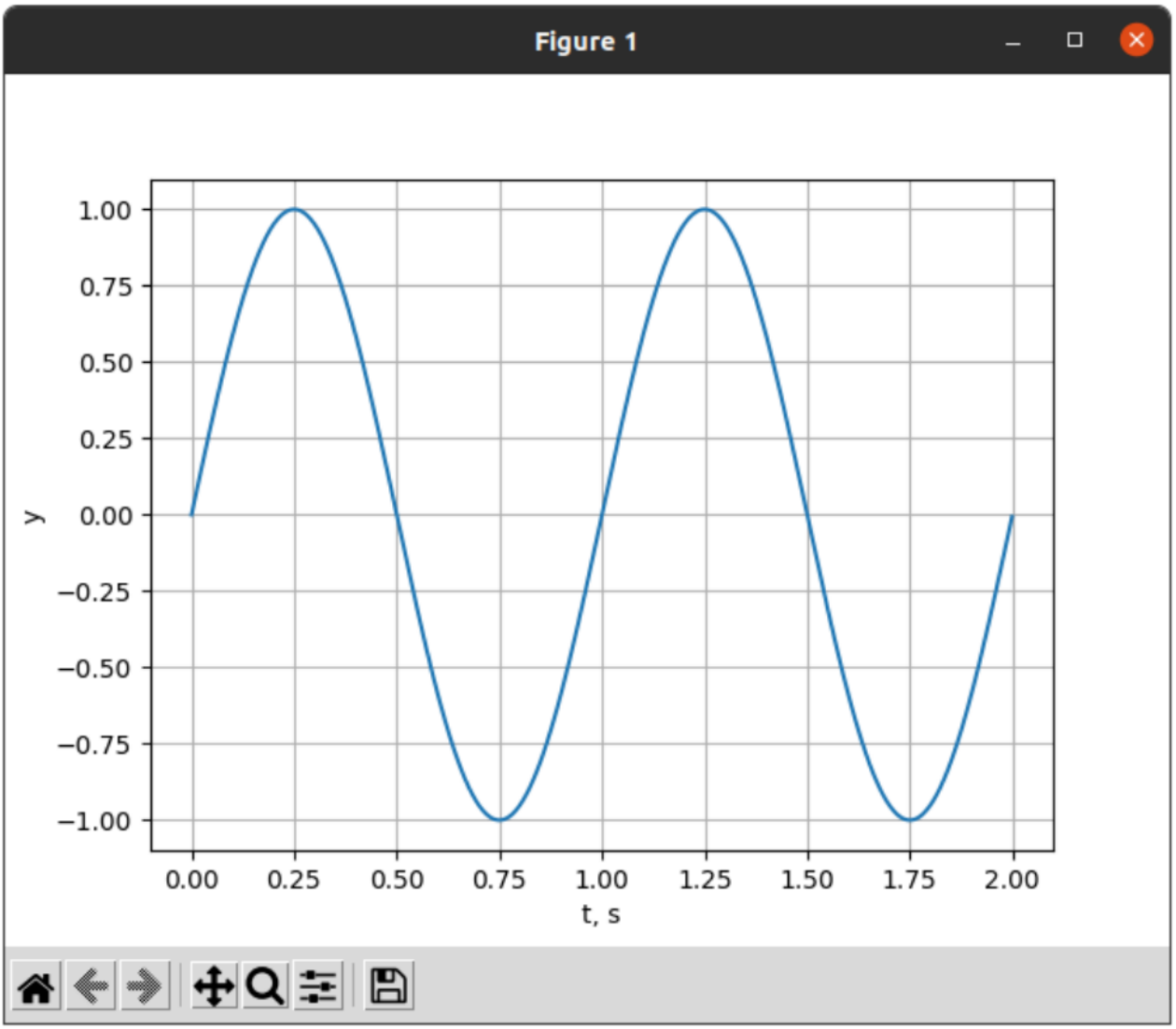

Рис. 2

1.3. Створення графічного інтерфейсу (GUI)

Створення графічного інтерфейсу будемо виконувати на основі Tcl/Tk графічної бібліотеки. Для роботи із цим графічним інструментарієм до складу стандартної бібліотеки *python* входить пакет *tkinter*. Дані бібліотеки можна знайти за наступними посиланнями:

https://docs.python.org/3/library/tkinter.html

http://tkdocs.com/

Основні компоненти пакету *tkinter*. Клас *tkinter.Tk* створює *Tk віджет* найвищого рівня, зазвичай це головне вікно застосунку.

Послідовність дій створення найпростішого застосунку без використання принципі об’єктно-орієнтованого програмування. Розглянемо приклад, наведений у Лістинг л3.

1. Спочатку виконується завантаження необхідних модулів. Завантаження головного модулю *tkinter* із збереженням посилання у змінній *tk* (рядок 1) та завантаження вкладеного модулю, який забезпечує доступ до набору віджетів (рядок 2), бібліотеки *Tk*.

2. Виклик методу *tk.Tk()* створює та ініціалізує об’єкт класу *Tk* (рядок 4). Після чого, змінна *root* зберігатиме посилання на головне вікно застосунку. У наступному рядку задається заголовок головного вікна.
3. У рядку 7, за допомогою методу *ttk.Frame*, створюється віджет каркас, змінна *frame*, для розміщення графічних віджетів інтерфейсу, таких як мітка (t*tk.Label*) та кнопка (*ttk.Button*). Даний віджет повинен бури розміщений на тлі головного вікна *root*. Перший параметр методу *tk.Frame* приймає посилання на головне вікно застосунку, до якого прив'яжеться віджет.
4. У рядку 8, у тілі рамки *frame* створюється та ініціалізується сітка, яка контролює розміщення графічних віджетів на тілі рамки. У даному випадку це сітка або прямокутна таблиця, яка ділиться на рядки та стовпці. Метод рамки frame.grid() створює порожню сітку у яку пізніше додамо мітку та кнопку.
5. У рядку 9 створюється та ініціалізується мітка за допомогою методу *ttk.Label*. У перший параметр методу передається посилання на рамку, у якій розміщується віджет, другому параметру *text* присвоюється текст, який буде зображатися міткою. Метод *ttk.Label* після створення та ініціалізації об’єкту мітка повертає на нього посилання. Оскільки, у нашому простому застосунку користувач або сам застосунок не буде взаємодіяти із міткою, то ми і не зберігаємо посилання у будь-якій змінній. Проте, через повернуте посилання ми викликаємо метод мітки grid() , який використовуємо для розміщення мітки у батьківському віджеті, рамка. Положення у рамці, використовуючи метод grid(), визначається номером колонки та рядку у сітці рамки за допомогою двох параметрів: column - колонка та row — рядок.
5 Наступний рядок створює віджет кнопка за допомогою методу ttk.Button, вказує батьківський контейнер, текст кнопки та функцію обробника події натискання на кнопку, у даному випадку в параметр command передається метод root.destroy() головного вікна, який виконує знищення вікна та завершення застосунку.
6. Останній рядок запускає нескінчений цикл який обробляє усі події які виникають у застосунку у вигляді черги.

Лістинг 3

```
1   import tkinter as tk
2   from tkinter import ttk
3
4   root = tk.Tk()
5   root.title('Simple GUI app')
6
7   frame = ttk.Frame(root, padding=50)
8   frame.grid()
9   ttk.Label(frame, text='Push').grid(column=0, row=0)
10  ttk.Button(frame, text='Quit', command=root.destroy).grid(column=1, row=0)
11
12  root.mainloop()
```

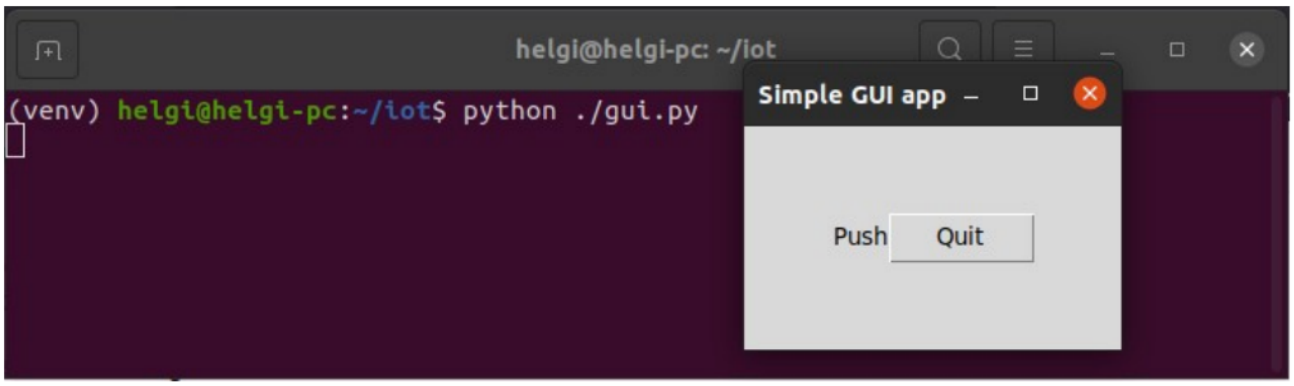

Рис. 3

1.4. Створення графічного інтерфейсу (GUI) із використанням принципів ООП

Лістинг 4

```
1   import tkinter as tk
2   from tkinter import ttk
3
4
5   class GUI(ttk.Frame):
6
7       def __init__(self, master=None, **kwargs):
8           super().__init__(master, **kwargs)
9           self.grid()
10
11          self._create_widgets()
12
13      def _create_widgets(self):
14          ttk.Label(self, text='Push').grid(column=0, row=0)
15          ttk.Button(self, text='Quit',
16                     command=self.master.destroy).grid(column=1, row=0)
17
18
19  root = tk.Tk()
20  root.title('Simple GUI app')
21  app = GUI(root, padding=50)
22  app.mainloop()
```

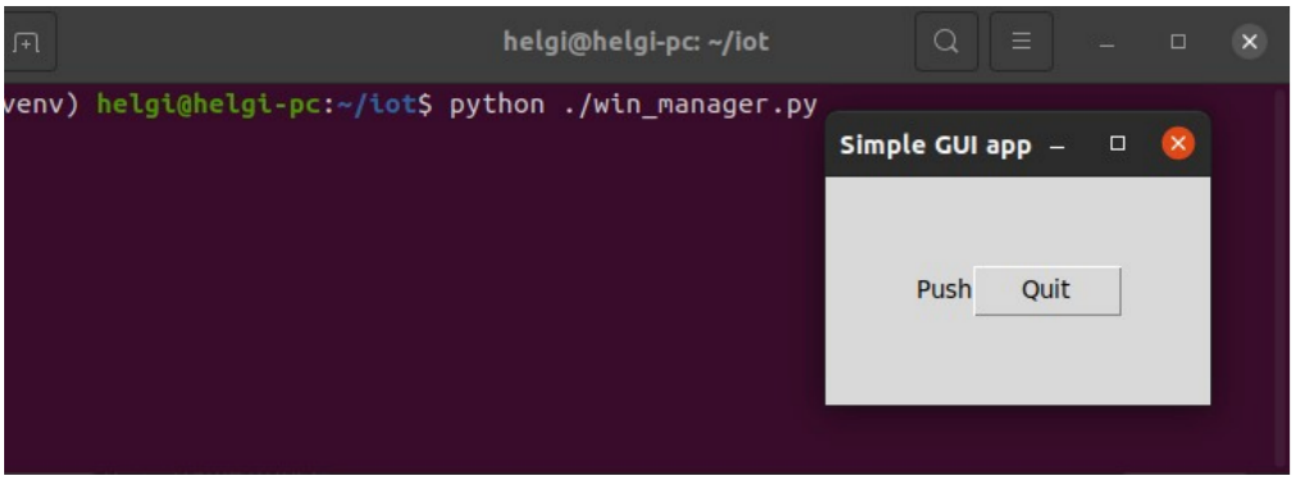

Рис. 4

1.5. Створення графічного інтерфейсу (GUI) для виведення змодельованої інформації

Створимо спочатку просте вікно із кнопкою завершення.

Лістинг 5

```
1   import tkinter as tk
2   from tkinter import ttk
3
4
5   class GUI(ttk.Frame):
6
7       def __init__(self, master=None, **kwargs):
8           super().__init__(master, **kwargs)
9           self.pack(side=tk.TOP, fill=tk.BOTH, expand=tk.YES)
10
11          self._create_widgets()
12
13      def _create_widgets(self):
14          ttk.Button(self, text='Quit',
15                     command=self.master.destroy).pack(side=tk.LEFT)
16
17
18  root = tk.Tk()
19  root.title('Oscilloscope')
20  gui = GUI(root)
21  gui.master.mainloop()
```

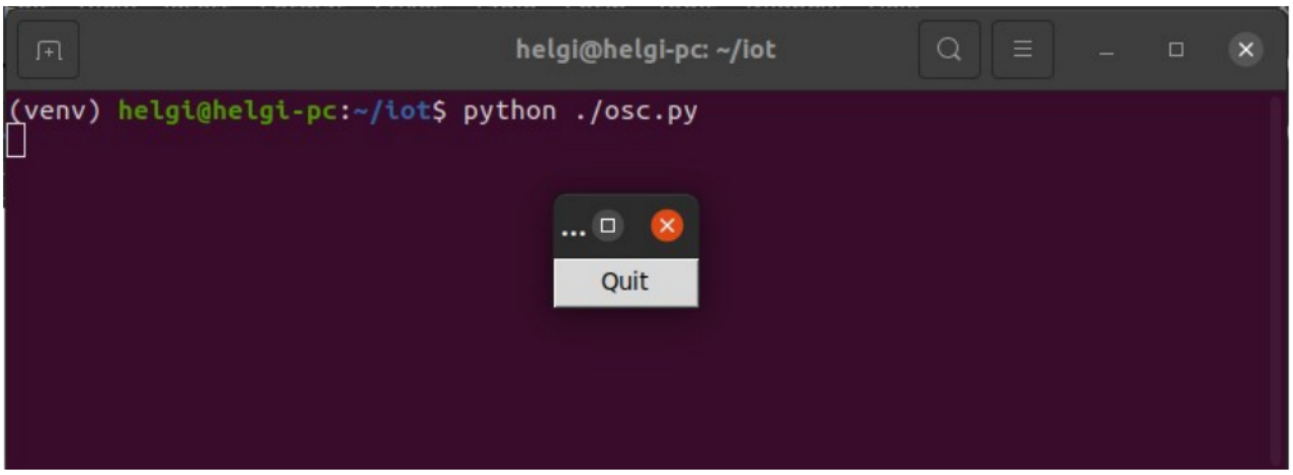

Рис. 5

Тепер додамо до вікна полотно для відображення графіків.

Лістинг 6

```
import tkinter as tk
from tkinter import ttk
import matplotlib
import numpy as np
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

matplotlib.use('TkAgg')


class GUI(ttk.Frame):

    def __init__(self, master=None, **kwargs):
        super().__init__(master, **kwargs)
        self.pack(side=tk.TOP, fill=tk.BOTH, expand=tk.YES)

        self._create_widgets()

    def _create_widgets(self):
        self._create_plot_widget()

        ttk.Button(self, text='Quit',
                   command=self.master.destroy).pack(side=tk.RIGHT)

    def _create_plot_widget(self):
        self._canvas = FigureCanvasTkAgg(Figure(dpi=100), master=self)
        self._canvas.get_tk_widget().pack(side=tk.TOP,
                                          fill=tk.BOTH,
                                          expand=tk.YES)
        self._ax = self._canvas.figure.add_subplot()
```

```
31            self._ax.set_xlabel('t, s')
32
33
34    if __name__ == '__main__':
35        root = tk.Tk()
36        root.title('Oscilloscope')
37        gui = GUI(root)
38        gui.master.mainloop()
```

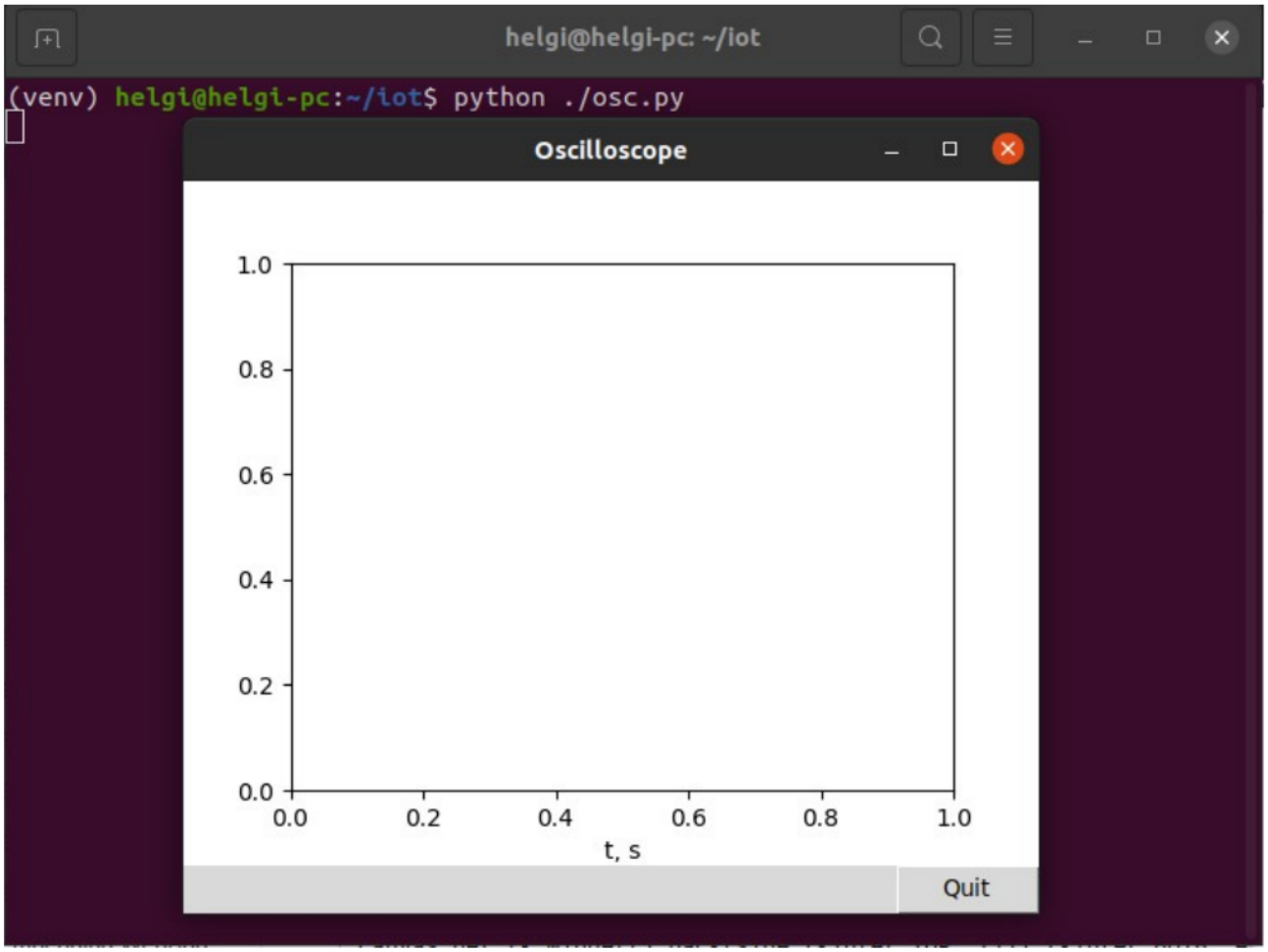

Рис. 6

Наступне, змоделюємо синусоїду та виведемо її на графік.

Лістинг 7

```
1     import tkinter as tk
2     from tkinter import ttk
3     import matplotlib
4     import numpy as np
5     from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
6     from matplotlib.figure import Figure
7
8     matplotlib.use('TkAgg')
9
```

```
10
11      class GUI(ttk.Frame):
12
13          def __init__(self, master=None, **kwargs):
14              super().__init__(master, **kwargs)
15              self.pack(side=tk.TOP, fill=tk.BOTH, expand=tk.YES)
16
17              self._create_widgets()
18
19              self._update_canvas()
20
21          def _create_widgets(self):
22              self._create_plot_widget()
23
24              ttk.Button(self, text='Quit',
25                         command=self.master.destroy).pack(side=tk.RIGHT)
26
27          def _create_plot_widget(self):
28              self._canvas = FigureCanvasTkAgg(Figure(dpi=100), master=self)
29              self._canvas.get_tk_widget().pack(side=tk.TOP,
30                                                fill=tk.BOTH,
31                                                expand=tk.YES)
32              self._ax = self._canvas.figure.add_subplot()
33              self._ax.set_xlabel('t, s')
34              self._ax.set_ylabel('y(t)')
35              self._ax.grid()
36
37          def _update_canvas(self):
38              # measurement duration time
39              t_meas = 2  # s
40              t_start = 0  # s
41              dt = 0.001
42              n = (t_meas - t_start) / dt
43              # time samples
44              t = np.arange(t_start, t_meas, dt)
45              # signal settings
46              amp = 1.
47              f = 1.  # Hz
48              phi = 0.
49              y = amp * np.sin(2 * np.pi * f * t + phi)
50
51              self._ax.plot(t, y)
52              self._ax.figure.canvas.draw()
53
54
```

```
55    if __name__ == '__main__':
56        root = tk.Tk()
57        root.title('Oscilloscope')
58        gui = GUI(root)
59        gui.master.mainloop()
```

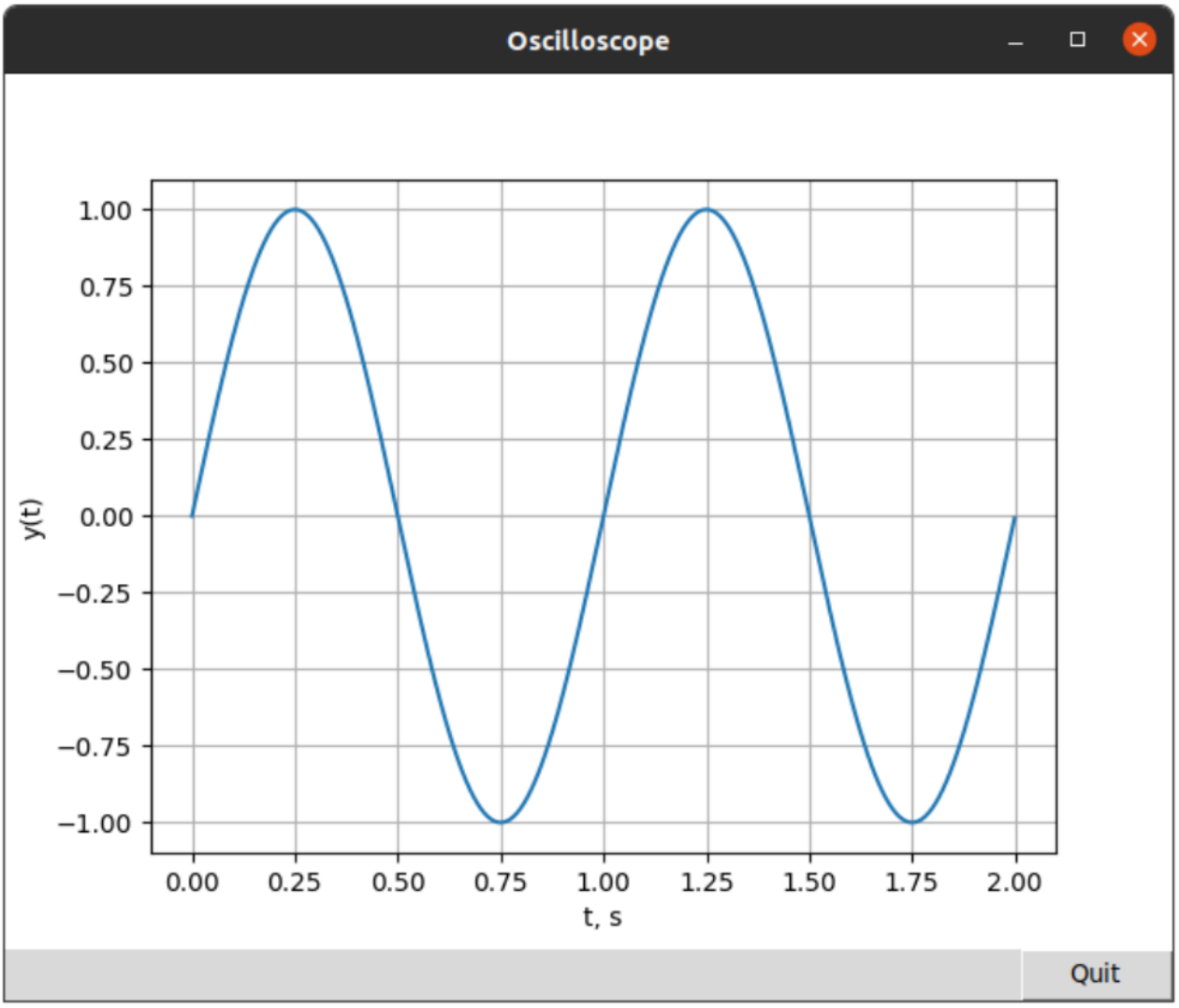

Рис. 7

2. Індивідуальні завдання

2.1. Створити застосунок для відображення змодельованого сигналу за прикладом лістингу 7.

2.2. Використати застосунок із пункту 2.1 та відобразити на графіку сигнал із шумом. Тип сигналу вибрати із таблиці 1.

Таблиця 1

№. п/п	Індивідуальне завдання
1.	Пилоподібний сигнал.

2.	Трикутний сигнал.
3.	Міандр.
4.	Прямокутний сигнал.
5.	Суму двох синусоїд із різними амплітудами та частотами.
6.	Добуток двох косинусних сигналів.
7.	Суму двох косинусних сигналів.