Лабораторна робота №4

Жук Дмитро РА-241

18 листопада 2025 р.

Тема: Списки, кортежі, робота з рядками, операції індексації та зрізів.

Мета: Ознайомитися з методами створення, обробки та маніпуляції списками, кортежами й рядками в Python. Навчитися використовувати індексацію, зрізи, функції та методи для роботи з цими типами даних.

Хід роботи

1 Імпорт (Imports)

Імпорт в Python — це спосіб підключити зовнішні модулі або бібліотеки, щоб не писати все з нуля. Використовується ключове слово `import`, або `from module import something` для конкретних частин. Це сприяє повторному використанню: ти береш готові інструменти від спільноти. У цій програмі імпорти розділені на групи — базові, `Tkinter`, `numpy/matplotlib` і `serial` (з `try-except` для обробки помилок імпорту). Це робить код адаптивним: якщо serial не доступний, програма переходить на dummy-режим.

Основний фрагмент імпортів:

```
import struct
import threading
import time
import random
import json
import tkinter as tk
from tkinter import ttk, messagebox, filedialog

```

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

try:
    import serial
    import serial.tools.list_ports
    UART_AVAILABLE = True
except ImportError:
    UART_AVAILABLE = False
```

- `import struct` — для розпаковки байтів у float (використовується в unpack).
- `import threading` — для створення потоків (клас `UARTReader` успадковує `threading.Thread`).
- `import tkinter as tk` і `from tkinter import ...` — імпорт GUI-елементів. `as tk` — аліас для зручності, щоб не писати повне ім'я кожного разу.
- `import numpy as np` — для масивів даних (`self.data = np.zeros(...)`).
- Try-except навколо `import serial` : якщо бібліотека не встановлена, встановлюється флаг `UART_AVAILABLE = False`, і програма не падає, а йде в симуляцію. Це розумний спосіб імпорту для крослатформовості.

2 Функції (Functions)

Функції в Python — це блоки коду, які виконують конкретне завдання, визначаються через `def name(args):`. Вони приймають аргументи, повертають значення (`return`) і можуть мати дефолтні значення. У цій програмі функції — це методи класів (бо код об'єктно-орієнтований), але є й глобальні (як `if name == "main"`). Вони автоматизують повторювані дії, наприклад, читання даних чи оновлення графіка.

Функції (методи):

- **Метод init у класі UARTReader:** Це конструктор, ініціалізує об'єкт. Приймає callback, port тощо, з дефолтними значеннями (baudrate=115200).

```
def __init__(self, callback, port=None, baudrate=115200, dummy=False):
    super().__init__(daemon=True)
    self.callback = callback
    self.dummy = dummy or not UART_AVAILABLE
    self.running = False
```

```
    self.port = port
    self.baudrate = baudrate
    self.ser = None
```

Ця функція встановлює стан потоку. `super()._init_` — викликає конструктор батьківського класу `threading.Thread`. Це приклад спадкування для повторного використання.

- **Метод run у UARTReader:** Перевизначений метод з `Thread`, запускає логіку потоку. Використовує `if` для вибору режиму.

```
def run(self):
    self.running = True
    if self.dummy:
        self._run_dummy()
    else:
        try:
            self.ser = serial.Serial(self.port, self.baudrate, timeout=1)
            self._run_uart()
        except Exception as e:
            print(f"UART open error: {e}")
            self.running = False
```

Функція не повертає нічого (`None`), але запускає приватні методи (`_run_dummy`, `_run_uart`). Приватні методи (з _) — конвенція для ”внутрішнього”використання.

- **Метод `on_new_data` у класі App:** Callback-функція, яка обробляє нові дані.

```
def on_new_data(self, floats):
    self.data = np.roll(self.data, -1, axis=1)
    self.data[:, -1] = floats
```

Приймає список `floats`, маніпулює numpy-масивом textttself.data (зсув і присвоєння). Це проста функція для оновлення даних, викликається з потоку.

- **Метод `update_plot` у App:** Рекурсивно планує себе через `after` для анімації.

```
def update_plot(self):
    if getattr(self, "_closing", False):
        return
    if self.config_data and self.lines:
        for item in self.lines:
            line, cfg, ch_idx = item
            scaled = self.data[ch_idx] * cfg.get("scale", 1.0)
```

```
            line.set_ydata(scaled)
        for ax in self.subplots:
            ax.relim()
            ax.autoscale_view(scaley=True)
        try:
            self.canvas.draw()
        except tk.TclError:
            return
    try:
        self._after_id = self.root.after(100, self.update_plot)
    except tk.TclError:
        self._after_id = None
```

Це ключова функція для візуалізації: оновлює лінії, масштабує, перемальовує. Використовує `try-except` для обробки помилок `Tkinter`.

3 Модульність коду (Modularity)

Модульність — це розбиття коду на незалежні модулі (класи, функції, файли), щоб кожна частина робила свою справу і легко замінювалася. У Python це досягається класами, методами і імпортами. Ця програма модульна: логіка читання даних в окремому класі (`UARTReader`), `GUI` і візуалізація — в `App`.

- **Клас `UARTReader`:** Окремий модуль для читання даних, успадковує Thread. Містить методи run, `_run_uart`, `_run_dummy`, `stop`. Це ізолює багатопотоковість від `GUI`.

```
class UARTReader(threading.Thread):
    # ... (all methods inside)
```

Клас — як модуль: створюєш об'єкт, запускаєш, зупиняєш. Модульність: dummy режим відділений від реального `UART`.

- **Клас `App`:** Головний модуль для `GUI`. Містить `init` для налаштування інтерфейсу, методи для завантаження конфігу, старту/стопу, оновлення.

```
class App:
    def __init__(self, root):
        # ... menu settings, controls, placeholder graphics
    # other methods: load_config, setup_plots, refresh_ports, etc.
```

`init` будує інтерфейс (меню, кнопки, canvas). Інші методи — як підмодулі: `setup_plots` очищує і налаштовує графіки, `on_close` чисто закриває програму.

- **Глобальний блок if `name == "main"`:** Це модульність для запуску як скрипту, але дозволяє імпортувати клас App в інший файл без запуску.

```
if __name__ == "__main__":
    root = tk.Tk()
    app = App(root)
    root.protocol("WM_DELETE_WINDOW", app.on_close)
    root.mainloop()
```

Це стандартний патерн: програма запускається тільки якщо файл виконується безпосередньо.

4 Повторне використання коду (Code Reuse)

Повторне використання — це уникнення дублювання через функції, класи, імпорти і спадкування. У Python це ООП (спадкування), імпорти бібліотек і виклики методів. Тут багато прикладів: готові бібліотеки `(serial, matplotlib)`, спадкування `(Thread)`, виклики методів у циклах.

- **Спадкування від `threading.Thread`:** `UARTReader` повторно використовує логіку потоків з стандартної бібліотеки.

```
class UARTReader(threading.Thread):
    def __init__(self, ...):
        super().__init__(daemon=True)
    def run(self):  # redefinition
        ...
```

Замість писати свій потік, беремо готовий і додаємо свою логіку.

- **Виклик `callback` у циклі:** У `_run_uart` і `_run_dummy` повторно використовується `self.callback` для обробки даних.

```
# in _run_uart
if frame[-2:] == TERMINATOR:
    floats = struct.unpack('<16f', frame[:64])
    self.callback(floats)
```

```
# in _run_dummy
dummy_floats = [random.uniform(0, 10) for _ in range(16)]
```

```
self.callback(dummy_floats)
```

Один і той же `callback (on_new_data)` використовується в обох режимах — немає дублювання обробки.

- **Цикли для налаштування в `setup_plots`:** Повторне використання коду в `for` для `subplot` і каналів.

```
for i in range(n_subplots):
    subplot_cfg = subplots_cfg[i] if i < len(subplots_cfg) else {}
    ax = self.fig.add_subplot(n_subplots, 1, i+1)
    # ... settings
    self.subplots.append(ax)
```

```
for ch_idx, ch_name in enumerate(self.channel_keys):
    # ...
    self.lines.append((line, ch, ch_idx))
```

Замість писати код для кожного `subplot`/каналу окремо, цикл повторно використовує логіку для всіх.

- **Імпорти для повторного використання:** `Numpy` для масивів (`np.roll`, `np.zeros`), `Matplotlib` для графіків (`ax.plot`, `fig.tight_layout`) — все готове, не пишемо з нуля.

Висновок

У лабораторній роботі №4 розглянуто організацію Python-програми: імпорт модулів з обробкою винятків, використання функцій і методів класів, принципи модульності (окремі класи UARTReader та App) та повторне використання коду (спадкування, callback, цикли, готові бібліотеки). Програма демонструє стійку багатопоточну візуалізацію даних з UART або в симуляційному режимі.

Звіт оформлено в LaTeX (`extarticle 14 pt, XeLaTeX, listings, tcolorbox`), що забезпечило професійний вигляд документу.