Лабораторна робота №2

Жук Дмитро РА-241

17 листопада 2025 р.

Тема: Умовні оператори (`if/else`) та цикли (`for, while`) для автоматизації обчислень.

Мета: Ознайомитися з принципами використання умовних операторів `if, elif, else` та циклів `for, while` для створення автоматизованих алгоритмів обчислення та прийняття рішень у програмах на Python.

Хід роботи

Принципи використання умовних операторів (`if, elif, else`) і циклів (`for, while`). У цій програмі вони допомагають автоматизувати процеси: приймати рішення на основі умов (наприклад, чи доступний `UART`, чи завантажена конфігурація), обробляти дані в циклі (наприклад, безперервно читати з порту) і обчислювати значення (наприклад, масштабувати дані для графіків). Це робить програму гнучкою — вона реагує на різні сценарії без ручного втручання.

1 Умовні оператори: if, elif, else

Програма перевіряє умови і виконує різні гілки коду залежно від них. Наприклад, if перевіряє, чи істина умова, `else` — що робити, якщо ні. `elif` тут не використовується (в коді його немає), але `if-else` часто комбінуються з `try-except` для обробки помилок.

Приклади:

- **Перевірка на dummy-режим у класі UARTReader (метод run):** Зберігає стан потоку (чи він повинен працювати). Тут `if-else` вирішує, чи запускати реаль-

не читання з UART, чи симуляцію.

```
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
```

- **Обробка фреймів даних у _run_uart:** Тут `if` перевіряє, чи є дані і чи закінчується фрейм термінатором. Це рішення автоматизує розбір потоку байтів — програма сама відфільтровує валідні дані для обчислення.

```
if data:
    buffer += data
    while len(buffer) >= FRAME_SIZE:
        frame, buffer = buffer[:FRAME_SIZE], buffer[FRAME_SIZE:]
        if frame[-2:] == TERMINATOR:
            floats = struct.unpack('<16f', frame[:64])
            self.callback(floats)
```

Якщо дані прийшли (`if data`), додаємо в буфер. Потім, якщо фрейм повний і має правильний кінець (`if frame[-2:] == TERMINATOR`), розпаковуємо в `float` і викликаємо `callback`.

- **Налаштування subplot у setup_plots:** Тут `if-else` в тернарному виразі вирішує, яку конфігурацію взяти для `subplot`. Це автоматизує створення графіків — якщо конфіг не вказаний, використовується дефолтний.

```
subplot_cfg = subplots_cfg[i] if i < len(subplots_cfg) else {}
```

Якщо індекс i в межах конфігу, беремо його; інакше — порожній словник. Потім:

```
if subplot_idx < 0:
    subplot_idx = 0
if subplot_idx >= len(self.subplots):
    subplot_idx = len(self.subplots) - 1
```

Це ”захищає”індекс, щоб уникнути помилок — програма сама коригує значення для коректного обчислення позиції графіка.

- **Перевірка портів у refresh_ports:** `If-else` перевіряє доступність `UART` і заповнює список портів. Автоматизує інтерфейс: якщо `UART` немає, список порожній, і користувач не побачить помилок.

```
if UART_AVAILABLE:
    ports = [p.device for p in serial.tools.list_ports.comports()]
else:
    ports = []
```

- **Обробка помилок у start_reader:** `Try-except` (яке схоже на `if-else` для помилок) перевіряє `baud rate`. Якщо не число — показує помилку, автоматизуючи валідацію вводу.

```
try:
    baud = int(self.baud_var.get())
except ValueError:
    messagebox.showerror("Error", "Invalid baud rate")
    return
```

У цілому, умовні оператори роблять програму ”розумною”— вона приймає рішення про режими роботи, обробляє помилки і фільтрує дані автоматично, без потреби в ручних перевірках.

2 Цикли: for і while

Цикли використовуються для повторюваних обчислень і обробки колекцій. `While` — для безкінечних процесів (поки умова True), `for` — для ітерації по списках чи діапазонах.

- **While у _run_uart і _run_dummy:** Це основний цикл для читання даних. `While` автоматизує безперервне опитування порту або генерацію симуляції, обчислюючи нові значення в реальному часі.

```
while self.running:
    data = self.ser.read(FRAME_SIZE)
    if data:
        buffer += data
        while len(buffer) >= FRAME_SIZE:
            frame, buffer = buffer[:FRAME_SIZE], buffer[FRAME_SIZE:]
            if frame[-2:] == TERMINATOR:
                floats = struct.unpack('<16f', frame[:64])
                self.callback(floats)
```

Зовнішній `while` триває, поки `self.running` `True` (тобто потік активний). Вкладений `while` обробляє весь буфер, якщо накопичилося багато даних. У dummy-режимі:

```
while self.running:
    dummy_floats = [random.uniform(0, 10) for _ in range(16)]
    self.callback(dummy_floats)
    time.sleep(0.1)
```

Це генерує випадкові дані кожні 0.1 секунди — автоматизований алгоритм симуляції для тестування.

- **For у setup_plots:** `For` ітерує по діапазону для створення `subplot` і по каналам для налаштування ліній. Це автоматизує обчислення позицій і масштабів для графіків.

```
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)
    ax.set_title(subplot_cfg.get("title", f"Plot {i+1}"))
    ax.set_ylabel(subplot_cfg.get("y_label", ""))
    self.subplots.append(ax)
```

Цикл створює стільки `subplot`, скільки потрібно. Потім:

```
for ch_idx, ch_name in enumerate(self.channel_keys):
    ch = channels_cfg[ch_name]
    subplot_idx = ch.get("subplot", 0)
    # ... (index checks)
    ax = self.subplots[subplot_idx]
    line, = ax.plot(np.zeros(self.history_length), label=f"{ch.get('name',
    ch_name)} [{ch.get('unit','')}]")
    self.lines.append((line, ch, ch_idx))
```

```
    ax.legend(fontsize=6, ncol=4)
```

Enumerate дає індекс і ім'я каналу, for обходить всі канали і додає лінії на графіки — автоматично формує візуалізацію на основі конфігу.

- **For у update_plot:** `For` цикл оновлює дані на графіках. Автоматизує обчислення масштабування і перемальовування кожні 100 мс.

```
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)
```

Перший for масштабує дані для кожної лінії (обчислення scaled), другий — налаштовує осі. Це робить графіки динамічними без ручного втручання.

Висновок

Виконана лабораторна робота дозволила на практиці ознайомитися з використанням умовних операторів та циклів у програмуванні на Python. Застосування `if/else` дало можливість автоматизувати прийняття рішень у програмі, забезпечити обробку помилок та контроль за коректністю даних без ручного втручання. Використання циклів `for` та `while` показало ефективність автоматизованої обробки повторюваних завдань та безперервного опитування даних, що дозволяє створювати динамічні алгоритми обчислень і відображення інформації на графіках.

Під час виконання роботи я також опанував LaTeX, зокрема освоїв використання середовищ `lstlisting` для форматованого вставлення коду та `itemize` для структурованого оформлення списків.