

Лабораторна робота №5

Жук Дмитро РА-241

Тема: Робота з файлами, зчитування/запис, обробка винятків для стабільності програм.

Мета: Ознайомитися з методами роботи з файлами в Python, включаючи відкривання, зчитування, запис та коректне закриття файлів у текстовому й бінарному режимах. Навчитися використовувати менеджер контексту `with`, обробляти винятки `try-except-finally`, `try-except-else` під час файлових операцій, а також застосовувати механізми обробки помилок для забезпечення стабільності та відмовостійкості програм при роботі з зовнішніми ресурсами.

Хід роботи

Розглянемо детально всі місця, де програма взаємодіє з файлами, зчитує/записує дані та захищає себе за допомогою винятків.

1 Зчитування конфігураційного JSON-файлу

Програма дозволяє користувачеві завантажити конфігурацію графіків із JSON-файлу.

```
1 def load_config(self):
2     path = filedialog.askopenfilename(filetypes=[("JSON Files",
3         "*.json")])
4     if not path:
5         return
6     with open(path, 'r') as f:
7         self.config_data = json.load(f)
8     self.setup_plots()
```

- `filedialog.askopenfilename(...)` — відкриває стандартне системне вікно вибору файлу, користувач сам обирає шлях.
- `if not path: return` — якщо користувач натиснув «Скасувати», функція тихо завершується, нічого не ламається.
- `with open(path, 'r') as f:` — безпечне відкриття файлу. Контекстний менеджер `with` гарантує, що файл завжди буде закритий, навіть якщо станеться помилка.
- `json.load(f)` — зчитує весь файл і перетворює його в Python-об'єкт (словник зі списками).
- Після успішного зчитування викликається `self.setup_plots()` для перебудови графіків.

Тут немає явного блоку `try-except`, тому що:

- `filedialog` — вже захищений від неправильних шляхів.
- `with open` + `json.load` — у реальних умовах може викинути `FileNotFoundError`, `PermissionError` або `json.JSONDecodeError`, але в цій програмі автор свідомо покладається на те, що користувач обирає існуючий валідний файл. Це прийнятно для десктопної утиліти.

2 Обробка винятків при імпорті бібліотеки `serial`

Приклад захисного програмування:

```

1 try:
2     import serial
3     import serial.tools.list_ports
4     UART_AVAILABLE = True
5 except ImportError:
6     UART_AVAILABLE = False

```

- Якщо бібліотека `pyserial` не встановлена — програма не впаде при запуску.
- Замість цього встановлюється глобальний флаг `UART_AVAILABLE = False`.
- Далі цей флаг використовується в кількох місцях наприклад, у `refresh_ports`, і програма автоматично переходить у dummy-режим. Користувач навіть не помітить відсутності бібліотеки — просто працюватиме симуляція.

3 Обробка винятків при відкритті СОМ-порту

```
1 def run(self):
2     self.running = True
3     if self.dummy:
4         self._run_dummy()
5     else:
6         try:
7             self.ser = serial.Serial(self.port, self.baudrate,
8 timeout=1)
9             self._run_uart()
10        except Exception as e:
11            print(f"UART open error: {e}")
12            self.running = False
```

- Якщо порт зайнятий, не існує, немає прав доступу — виняток ловиться.
- Виводиться зрозуміле повідомлення в консоль.
- Потік коректно завершується `self.running = False`, GUI не зависає.

4 Перетворення baud rate — захист від некоректного вводу користувача

```
1 def start_reader(self):
2     if self.reader and self.reader.running:
3         return
4     try:
5         baud = int(self.baud_var.get())
6     except ValueError:
7         messagebox.showerror("Error", "Invalid baud rate")
8     return
```

- Користувач може ввести літери замість цифр.
- `int()` викине `ValueError` → ловимо, показуємо вікно з помилкою і не запускаємо потік.
- Програма залишається повністю працездатною.

5 Безпечне закриття програми

Метод `on_close`:

```
1 def on_close(self):
2     self._closing = True                                # 1
3     if getattr(self, "_after_id", None):                # 2
4         try:
5             self.root.after_cancel(self._after_id)
6         except Exception:
7             pass
8
9     if self.reader:                                     # 3
10        try:
11            self.reader.stop()
12            self.reader.join(timeout=2)
13        except Exception:
14            pass
15
16        try:                                             # 4
17            self.root.quit()
18        except Exception:
19            pass
20        try:
21            self.root.destroy()
22        except Exception:
23            pass
```

- Встановлюється флаг `_closing = True` — метод `update_plot` відразу припиняє планувати нові оновлення.
- Скасовується запланований `after(100, self.update_plot)` — запобігає виклику після знищення вікна.
- Зупиняється і чекається завершення потоку читання UART — без цього може бути «зависле» вікно.
- `quit()` і `destroy()` обгорнуті в окремі `try-except`, бо в деяких ситуаціях (наприклад, коли Tkinter вже частково знищений) вони можуть кидати винятки.

6 Захист у циклі оновлення графіку

```
1 def update_plot(self):
2     if getattr(self, "_closing", False):
3         return
4
5     # ... updating schedules ...
6
7     try:
8         self.canvas.draw()
9     except tk.TclError:
10        return
11
12    try:
13        self._after_id = self.root.after(100, self.update_plot)
14    except tk.TclError:
15        self._after_id = None
```

- Якщо користувач закрив вікно — `canvas.draw()` або `root.after` викличуть `TclError`.
- Ловимо їх і просто виходимо — без крашу всієї програми.

7 Додаткові дрібні, але важливі захисти

```
1 if ports:
2     self.port_var.set(ports[0])
```

— тільки якщо список портів не порожній.

```
1 n_subplots = len(subplots_cfg) or 1
```

— гарантія, що завжди буде хоча б один графік.

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

— захист від некоректних індексів у JSON-конфігурації.

Висновок

У ході виконання лабораторної роботи, оформленої за допомогою системи L^AT_EX, я закріпив навички роботи з файлами в Python, зокрема відкривання, зчитування та запис у різних режимах. Також було опрацьовано використання конструкцій `with`, `try-except` та інших механізмів обробки винятків, що забезпечують стабільність програм при роботі з зовнішніми ресурсами. Аналіз програми показав, як коректно обробляти помилки під час завантаження файлів, взаємодії з COM-портами та завершення роботи застосунку.