

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра «Програмна інженерія»

ЗВІТ

до практичного заняття №1 з дисципліни

«Аналіз та рефакторинг коду»

На тему: «Правила оформлення програмного коду»

Виконав:

ст. гр. ПЗП-22-4

Попов Богдан Сергійович

Прийняв:

ст. викладач кафедри ПІ

Сокорчук Ігор Петрович

Харків 2024

1 МЕТА

Навчитися рефакторингу програмного коду, закріпити основні правила оформлення коду.

2 ЗАВДАННЯ

Обрати мову програмування для прикладів коду. Створити презентацію на тему «Правила оформлення програмного коду».

3 ХІД РОБОТИ

Було обрано мову програмування Python. У презентації (Додаток А) наведено основні рекомендації щодо оформлення програмного коду з описами, а також приклад коду до і після застосування цих рекомендацій.

ВИСНОВКИ

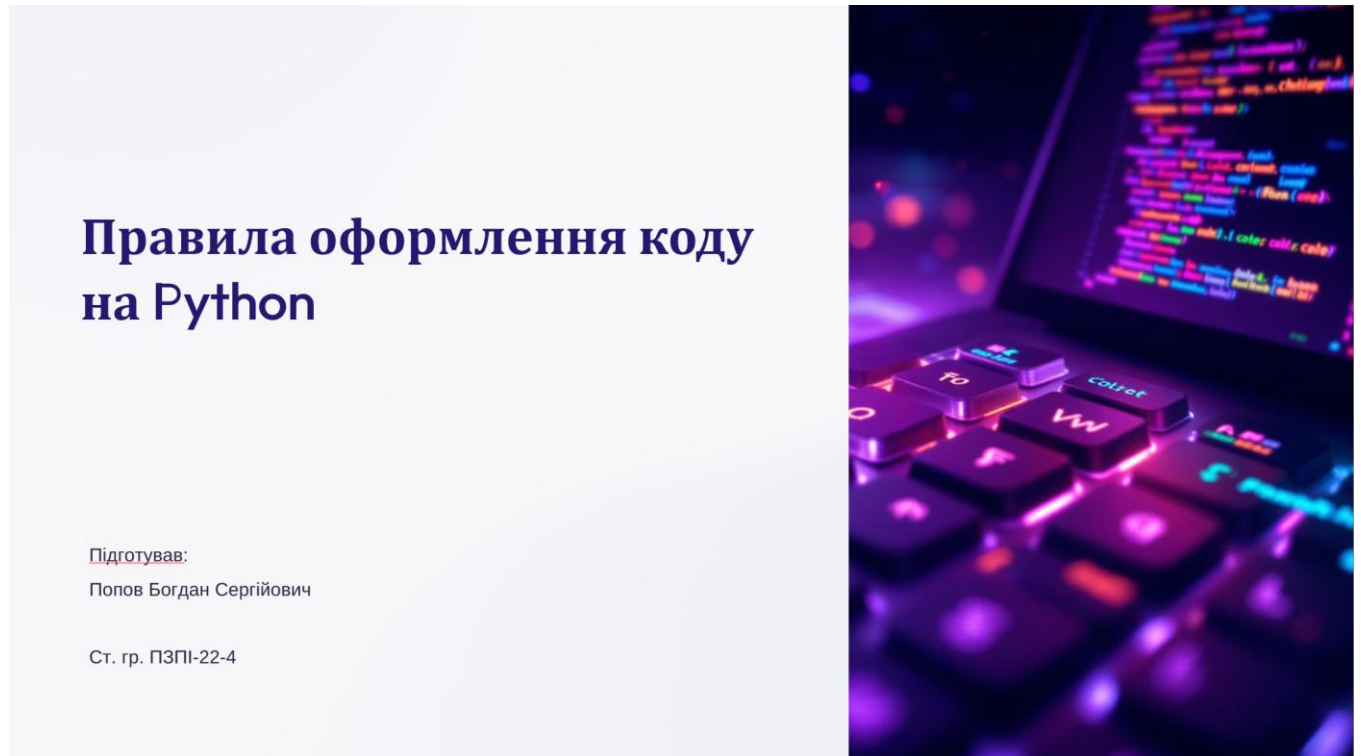
Набуто навичок рефакторингу програмного коду, детально розглянуто основні правила оформлення коду.

Посилання на відео-презентацію

Посилання на GitHub репозиторій с кодом

ДОДАТОК А

Презентація на тему «Правила оформлення програмного коду».



Правила оформлення коду на Python

Підготував:
Попов Богдан Сергійович

Ст. гр. ПЗПІ-22-4

Рис. 1 – Перший слайд з презентації



Читабельність коду

Ключовий фактор

Зрозумілий код - це запорука успіху будь-якого проекту. Легкість читання коду спрощує його розуміння та налагодження, а також робить його доступним для інших розробників.

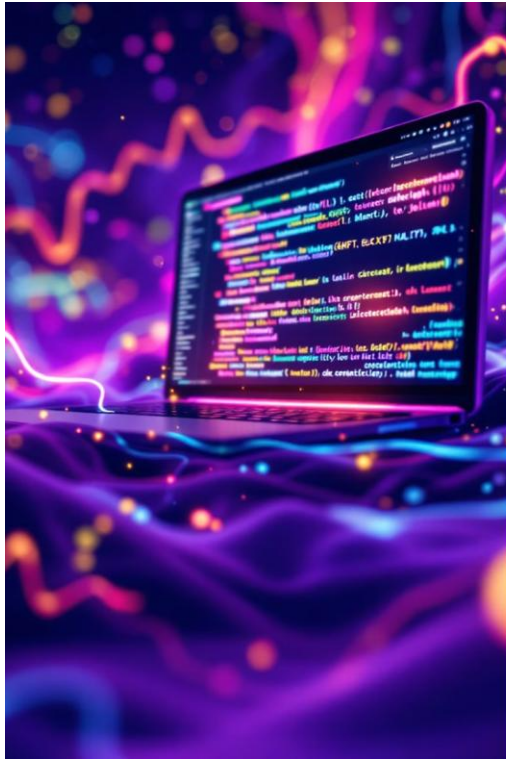
Код, написаний за правилами чистого програмування, зменшує кількість помилок, сприяє ефективній роботі в команді та забезпечує легкість внесення змін у майбутньому.

Дотримання єдиних стандартів оформлення підвищує продуктивність команди, адже дозволяє розробникам швидко розібратися в чужому коді та зосередитися на вирішенні завдань, а не на його розумінні.

Зверніть увагу

Зрозумілі назви змінних, функцій та класів дозволяють легко зрозуміти їх призначення без необхідності додаткового пояснення. Коментарі повинні пояснювати, що робить код, а не очевидні речі. Замість надмірних або складних коментарів краще прагнути до того, щоб сам код був самодокументованим. Коментарі важливі, коли потрібно пояснити складну логіку або нетривіальні рішення. Форматування коду, таке як правильні відступи, вирівнювання, дотримання стандартної довжини рядка (80 або 120 символів), робить код структурованим і легко читабельним. Використання стандартів форматування (наприклад, PEP 8 для Python) забезпечує однаковість у проєкті. Дотримання цих принципів підвищує якість коду, полегшує його підтримку та сприяє командній роботі.

Рис. 2 – Другий слайд з презентації



Назви змінних та функцій

Описові

Використовуйте назви, які описують призначення змінної чи функції.

Читабельні

Уникайте скорочень, що зменшують читабельність коду.

Зберігайте послідовність

Дотримуйтесь єдиного стилю для написання назв.

Рис. 3 – Третій слайд з презентації

Форматування коду

Відступи

Використовуйте 4 пробіли для відступів, а не табуляцію.

```
def example_function():  
    for i in range(10):  
        print(1)
```

Розбиття на рядки

Обмежуйте довжину рядка до 79 символів.

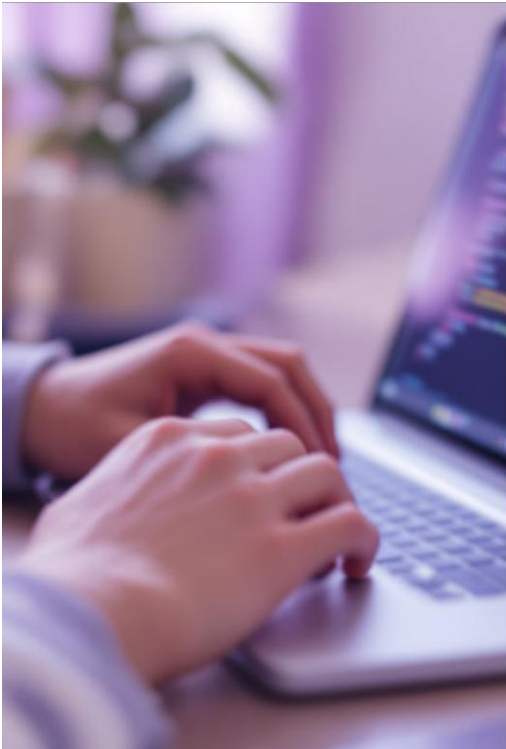
```
result = some_function(argument1, argument2,  
                        argument3, argument4)
```

Пусті рядки


Використовуйте пусті рядки для візуального розділення код.

```
def initialize():  
    # Ініціалізація  
    setup_environment()  
  
def process_data():  
    # Обробка даних  
    data = load_data()  
    analyze_data(data)
```

Рис. 4 – Четвертий слайд з презентації




Використання пробілів та відступів




Пробіли

Використовуйте пробіли навколо операторів (наприклад, +, -, *, /).



Відступи

Застосовуйте відступи для виділення блоку коду.



Послідовність

Дотримуйтесь єдиного стилю для використання пробілів та відступів.

Рис. 5 – П'ятий слайд з презентації

Модульність та структурованість коду

- 1 Розбиття на модулі**
Розбийте код на окремі модулі для кращої організації.
- 2 Функції**
Використовуйте функції для розбиття коду на окремі логічні блоки.
- 3 Класи**
Використовуйте класи для створення об'єктів та їхніх методів.


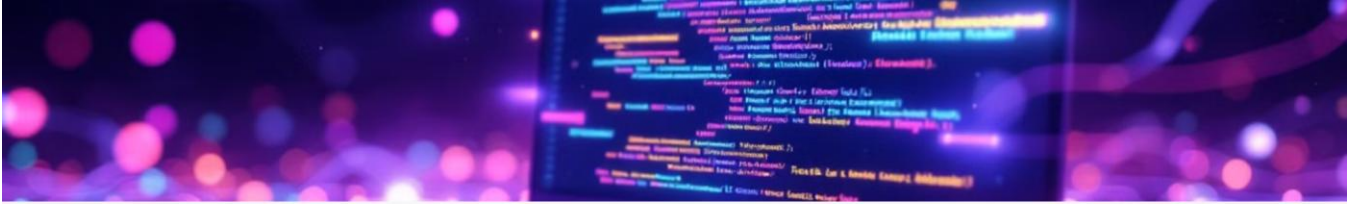


Рис. 6 – Шостий слайд з презентації



Обробка помилок та винятків

- 1** **Виявлення помилок**
Використовуйте блоки try/except для обробки можливих помилок.
- 2** **Обробка винятків**
Обробляйте винятки, щоб запобігти зупинці програми.
- 3** **Повідомлення про помилки**
Надавайте чіткі та зрозумілі повідомлення про помилки.

Рис. 7 – Сьомий слайд з презентації

Висновки та найкращі практики



1	Читабельність Зрозумілий код - це ключ до успіху.
2	Стандарти Дотримуйтесь стандартів PEP 8.
3	Тестування Регулярно тестуйте код.
4	Постійне вдосконалення Постійно вдосконалюйте свої навички.

Рис. 8 – Восьмий слайд з презентації

До рефакторингу

Код має складні вкладенні умови, які ускладнюють читабельність

```
def process_order(order):
    if order:
        if order['status'] == 'pending':
            if 'items' in order and len(order['items']) > 0:
                print("Order is valid and ready to process.")
            else:
                print("Order has no items.")
        else:
            print("Order is not pending.")
    else:
        print("No order provided.")
```

Після рефакторингу

Код спрощено за допомогою охоронних виразів:

```
def process_order(order):
    if not order:
        print("No order provided.")
        return

    if order['status'] != 'pending':
        print("Order is not pending.")
        return

    if 'items' not in order or len(order['items']) == 0:
        print("Order has no items.")
        return

    print("Order is valid and ready to process.")
```

Результат рефакторингу

- Читабельність: Логіка спрощена, кожна перевірка зрозуміла і не вкладена.
- Підтримуваність: Легко додати нові перевірки, оскільки кожен блок обробляє окремий випадок.
- Продуктивність: Менше перевірок у вкладених умовах, кожна умова виконується окремо

Рис. 9 – Дев'ятий слайд з презентації