Урок №14

19.09.2025

Правила написання читабельного коду та коментарів до нього

Коментарі в Python

Коментарі — це описи коду, які допомагають усім, хто читає, краще зрозуміти, що і навіщо робить код. Коментарі повністю ігноруються (тобто не виконуються) інтерпретаторами та призначені для колег-програмістів. Наприклад:

```
# Ініціалізуємо дві змінні
num1 = 6
num2 = 9
# Виводимо результат на екран
print('This is output')
```

Тут коментарями є наступні рядки:

`# Ініціалізуємо дві змінні`

`# Виводимо результат на екран`

Типи коментарів у Python

У Python є 2 типи коментарів:

- однорядкові коментарі;
- багаторядкові коментарі.

Однорядкові коментарі в Python

Однорядковий коментар починається та закінчується на одному рядку. Символ `#` використовується для написання однорядкового коментаря. Наприклад:

```
# Створюємо змінну
name = 'Eric Cartman'
# Виводимо значення змінної на екран
print(name)
```

Результат:

```
Eric Cartman
```

Тут у коді два однорядкові коментарі:

```
# Створюємо змінну
# Виводимо значення змінної на екран
```

Також можна використовувати однорядковий коментар у рядку разом із кодом:

```
name = 'Eric Cartman' # значенням змінної name є рядок #
```

Тут код до `#` — виконується, а код після `#` — ігнорується інтерпретатором.

Багаторядкові коментарі в Python

Python не пропонує окремого способу написання багаторядкових коментарів. Проте є варіанти оминути цю заборону.

Рішення №1: Використовувати `#` відразу в декількох рядках, щоб таким чином зробити багаторядковий коментар. Наприклад:

```
# Це довгий коментар
# і він розтягнутий
# на кілька рядків
```

Тут кожен рядок розглядається як однорядковий коментар і всі вони ігноруються інтерпретатором Python.

Рішення №2: Використовувати потрійні лапки — `''' '''` або `""" """`.

Ці потрійні лапки зазвичай використовуються для значень типу **string**. Але якщо ми не присвоюємо наші коментарі будь-якій змінній чи функції, то це альтернативний спосіб написання багаторядкових коментарів. Інтерпретатор ігнорує рядок, який не присвоєно жодній змінній або функції.

Розглянемо на практиці:

```
''' Це також є
чудовим прикладом
багаторядкових коментарів '''
```

Цей рядок не присвоюється жодним змінним, тому він ігнорується інтерпретатором. Попри те, що технічно це не багаторядковий коментар, його можна використовувати як такий.

Користь від коментарів у Python

#1. Роблять код зрозумілішим.

Якщо писати коментарі в коді, то цей код буде легше використовувати в майбутньому. Крім того, іншим розробникам буде легше зрозуміти цей код.

#2. Допомагають під час відлагодження коду.

Якщо ми отримаємо помилку під час роботи програми, то ми можемо закоментувати рядок коду, який викликає помилку, замість того, щоб видаляти його. Наприклад:

```
print('Python')
# print('Error Line )
print('Django')
```

Тут рядок `print('Error Line )` викликає помилку, тому ми зробили його коментарем. Тепер програма працює без помилок.

Це один з наочних прикладів того, як коментарі можуть стати корисним інструментом відлагодження коду.

Примітка: Завжди використовуйте коментарі, щоб пояснити, **чому** ви щось зробили, а **не як** ви це зробили. Коментарі не повинні бути описом погано написаного коду.

Правила написання читабельного коду

Написання читабельного коду є важливою складовою розробки програмного забезпечення. Читабельний код спрощує розуміння функціональності програми, полегшує процес тестування та підтримки, зменшує кількість помилок. У цій статті ми розглянемо деякі правила написання читабельного коду.

1) Використовуйте зрозумілі назви змінних та функцій

Назви змінних та функцій мають бути зрозумілими та відображати їхню функціональність. Назви змінних мають бути короткими та зрозумілими, а назви функцій мають відображати їхню дію. Наприклад, замість назви змінної “**x**” краще використовувати “**кількість_елементів**”, а замість назви функції “**func()**” краще використовувати “**обчислити_суму()**”.

```
# Поганий код
c = 5
d = 12
# Хороший код
city_counter = 5
elapsed_time_in_days = 12
```

2) Використовуйте пробіли та відступи

Використовуйте пробіли та відступи для полегшення читання коду. Використовуйте пробіли між операторами та змінними, наприклад **“a = 5”**, а не **“a=5”.** Використовуйте відступи для відображення залежностей між функціями та блоками коду.

3) Використовуйте коментарі

Використовуйте коментарі для пояснення функціоналу коду та незрозумілих або складних частин коду. Коментарі мають бути зрозумілими та короткими. Не використовуйте коментарі зайво, вони можуть заважати читабельності коду.

4) Використовуйте стандарти оформлення коду

Використовуйте стандарти оформлення коду для полегшення спільної роботи та розуміння коду. Стандарти оформлення коду містять правила відступів, розміру шрифту, використання коментарів, назв змінних та функцій, тощо. Використовуйте стандарти оформлення коду, які використовуються в вашій команді або проекті.

5) Розділяйте код на логічні блоки

При написанні читабельного коду важливо розділяти його на логічні блоки. Це допоможе полегшити розуміння коду і полегшити його зміну в майбутньому. Розділення коду на логічні блоки допомагає уникнути написання довгих функцій або методів, які виконують багато різних дій.

Для розділення коду на логічні блоки можна використовувати різні підходи, зокрема, використовувати класи, функції, модулі та інші структури даних. Наприклад, можна розділити код на класи, кожен з яких відповідає за виконання певної функції або реалізацію певного функціоналу.

Наприклад, якщо ви пишете програму, яка обробляє дані користувачів, ви можете розділити код на логічні блоки, які відповідають за обробку даних, валідацію даних, збереження даних у базу даних тощо. Кожен з цих блоків коду можна реалізувати як окрему функцію або метод, що дозволить легко змінювати їх у майбутньому і зробить код більш зрозумілим.

Крім того, розділення коду на логічні блоки допоможе забезпечити його перевикористання в інших проектах або модулях.

6) Використовуйте правильні типи даних

Використовуйте правильні типи даних для змінних та функцій. Наприклад, якщо змінна містить дробове число, використовуйте тип даних **float**, а якщо це ціле число, використовуйте тип **int**. Використовуйте правильні типи даних для полегшення розуміння функціональності коду.

7) Використовуйте модульну структуру коду

Використовуйте модульну структуру коду для полегшення розуміння функціональності та структури коду. Модульна структура дозволяє розділити код на невеликі логічні блоки, які можуть бути перевикористані в інших частинах програми.

8) Використовуйте умовні оператори та цикли з розумінням

Використовуйте умовні оператори та цикли з розумінням їх функціональності та можливих проблем. Уникайте зайвої вкладеності циклів та умовних операторів. Це допоможе зменшити кількість помилок та підвищити читабельність коду.

9) Використовуйте засоби дебагування

Використовуйте засоби дебагування для виявлення та виправлення помилок у коді. Засоби дебагування дозволяють крок за кроком виконувати код та вивчати значення змінних. Це допомагає виявляти помилки та виправляти їх швидше.

10) Використовуйте імена змінних та функцій, які описують їх функціональність

Використовуйте імена змінних та функцій, які описують їх функціональність. Це допоможе полегшити розуміння функціональності та структури коду. Імена змінних та функцій повинні бути зрозумілими та лаконічними.

11) Дотримуйтесь стандарту індексації

Дотримуйтесь стандарту індексації для забезпечення однакового стилю оформлення коду. Стандарт індексації встановлює кількість пробілів для відступів. Використовуйте той самий стиль індексації для всього коду.

12) Документуйте код

Документуйте код для полегшення розуміння функціональності та структури коду. Використовуйте коментарі для опису функціональності та структури коду. Документація повинна бути зрозумілою та лаконічною.

13) Використовуйте тести

Використовуйте тести для виявлення та виправлення помилок у коді. Тести дозволяють автоматично перевіряти функціональність коду та виявляти помилки. Використовуйте тести для кожної нової функції або зміни в коді.

Висновок

Правильне написання читабельного коду є важливим аспектом розробки програмного забезпечення. Читабельний код полегшує спільну роботу та підтримку програмного забезпечення. Дотримуйтесь стандартів оформлення коду та використовуйте засоби дебагування, імена змінних та функцій, стандарти індексації, документуйте код та використовуйте тести, щоб забезпечити ефективну розробку та підтримку програмного забезпечення.

Часті запитання

1. Чому важливо писати читабельний код?

Писати читабельний код важливо, оскільки це полегшує спільну роботу та підтримку програмного забезпечення. Код, який легко зрозуміти, є більш простим у розробці та тестуванні.

2. Які є стандарти оформлення коду?

Стандарти оформлення коду встановлюють правила, які визначають форматування, індексацію, іменування змінних та функцій. Використовування стандартів оформлення коду допомагає підтримувати читабельність та однаковість стилю оформлення коду.

3. Які засоби дебагування можна використовувати?

Деякі засоби дебагування, які можна використовувати, включають інтерактивні середовища розробки (IDE), налагоджувачі та логування.

4. Які є переваги використання тестів для розробки програмного забезпечення?

Використання тестів для розробки програмного забезпечення дозволяє виявляти та виправляти помилки в коді раніше та автоматично перевіряти функціональність програмного забезпечення.

5. Які є недоліки погано написаного коду?

Погано написаний код може призводити до помилок та проблем з підтримкою та розширенням програмного забезпечення. Крім того, він може бути складним у зміні та вивченні, що затримує процес розробки та підтримки програмного забезпечення.

Розглянь приклади і поясни чому один код вважають поганим, а інший хорошим?

№1

```
# Поганий код
# represents the number of active users
au = 55
# Хороший код
active_user_amount = 55
```

№2

```
from datetime import datetime
# Поганий код
genyyyymmddhhmmss = datetime.strptime('04/27/95 07:14:22', '%m/%d/%y %H:%M:%S')
# Хороший код
generation_datetime = datetime.strptime('04/27/95 07:14:22', '%m/%d/%y %H:%M:%S')
```

№3

```
# Поганий код
fna = 'Bob'
cre_tmstp = 1621535852
# Хороший код
first_name = 'Bob'
creation_timestamp = 1621535852
```

№4

```
# Поганий код
def get_name(): pass
def fetch_age(): pass
# Хороший код
def get_name(): pass
def get_age(): pass
```

№5

```python
# Поганий код
class Person:
    def __init__(self, person_first_name, person_last_name, person_age):
        self.person_first_name = person_first_name
        self.person_last_name = person_last_name
        self.person_age = person_age

# Хороший код
class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
```