Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

Звіт
про виконання лабораторної роботи №4
на тему:

«Метапрограмування в Python: створення динамічних класів і атрибутів»

Виконала:
Студентка групи ФеС-32
Філь Дарина

Львів - 2025

Мета:

Ознайомитись з основами метапрограмування та його застосуванням у сучасних мовах програмування, зокрема в Python. Вміти створювати більш гнучкі та динамічні програми, автоматизувати рутинні завдання та модифікувати поведінку програм під час виконання.

Теоретичні відомості:

Метапрограмування — це техніка, яка дозволяє програмам працювати з власним кодом, змінювати його або генерувати новий код під час виконання. Python підтримує метапрограмування завдяки своїй динамічній природі.

Основні випадки використання метапрограмування:

1. Декоратори – зміна поведінки функцій або класів без зміни їхнього коду.
2. Модифікація класів (метакласи) – вплив на створення класів.
3. Динамічне створення класів – генерування класів у реальному часі.
4. Рефлексія та introspection – аналіз коду під час виконання.

Де це використовується?

✅ Фреймворки (Django, Flask) – використовують метакласи для ORM.

✅ Автоматична генерація коду – у DSL (спеціалізованих мовах).

✅ Оптимізація коду – для зменшення дублювання.

✅ Тестування та логування – автоматична інспекція коду.

Метапрограмування має ряд недоліків, про які варто знати:

1. Складність для розуміння і підтримки коду: Оскільки метапрограмування дозволяє динамічно змінювати структуру програми (створювати нові методи, класи, атрибути на льоту), це може зробити код менш прозорим і важким для розуміння. Іншим розробникам буде складно зрозуміти, де і як саме відбуваються ці зміни.

2. Зниження продуктивності: Метапрограмування може додати додаткові витрати на виконання, оскільки під час роботи програми виконуються додаткові операції, такі як перевірки, рефлексія або створення нових методів і класів динамічно. Це може вплинути на швидкість виконання, особливо в критичних частинах програми.

3. Проблеми з дебагінгом: Оскільки код змінюється динамічно, часто стає складніше відстежувати помилки. Наприклад, помилки, які виникають через метапрограмування, можуть бути неочевидними і важко відслідковувати в процесі виконання програми. Це ускладнює процес налагодження.

4. Зниження безпеки коду: Метапрограмування може призвести до створення неочевидних чи небезпечних конструкцій у програмі. Наприклад, динамічне створення методів може призвести до конфліктів імен або виклику методів, яких не існує, що збільшує ризик помилок.

5. Ускладнення тестування: Тести, написані для програм, що активно використовують метапрограмування, можуть бути складнішими, оскільки структура програми змінюється динамічно. Це може зробити написання юніт-тестів складнішим і менш ефективним.

6. Залежність від специфічних мовних можливостей:

Метапрограмування вимагає використання специфічних можливостей мови програмування (наприклад, рефлексії, інспекції класів, створення коду на льоту), що може зробити програму менш сумісної з іншими платформами, які не підтримують ці функції.

7. Погіршення читабельності коду: Створення складних і динамічних конструкцій може зробити код менш читабельним і важким для розуміння іншими програмістами, особливо якщо вони не знайомі з метапрограмуванням.

Завдання до роботи:

1. Реалізувати програму, яка обчислює площу фігур динамічно. Дано масив об'єктів класів фігур: shapes = [Rectangle(4, 5),Triangle(3, 4, 5), Circle(7)]. Створити динамічно класи для кожної фігури та методи для обчислення площі периметра чи довжини кола.

2. Створити відповідний метод який викликає ці класи та відповідні методи

3. Використовуючи мета-клас, вивести прізвище студента, який виконав роботу.

Хід роботи

1. Реалізувати програму, яка обчислює площу фігур динамічно. Дано масив об'єктів класів фігур: shapes = [Rectangle(4, 5),Triangle(3, 4, 5), Circle(7)]. Створити динамічно класи для кожної фігури та методи для обчислення площі периметра чи довжини кола.

```
Figure = type(
    'Figure',
    (object,),
    {
        '__init__': lambda self, a, b=None: setattr(
            self,
            '__dict__',
            {'a': a, 'b': b, 'name': self.__class__.__name__}
        ),
        '__str__': lambda self: f'{self.name} Дарина\n{self.name} Периметр/Довжина: {self.perimetr()}\n{self.name} Площа: {self.area()}',
    },
)
```

```
Circle = type(
    'Circle',
    (Figure,),
    {
        'area': lambda self: 3.14 * self.a**2,
        'perimetr': lambda self: 2 * 3.14 * self.a,
    },
)

Triangle = type(
    'Triangle',
    (Figure,),
    {
        'area': lambda self: (sqrt(3) / 4) * self.a**2,
        'perimetr': lambda self: self.a * 3,
    },
)

Square = type(
    'Square',
    (Figure,),
    {
        'area': lambda self: self.a * self.b,
        'perimetr': lambda self: 2 * (self.a + self.b),
    },
)
```

2. Створити відповідний метод який викликає ці класи та відповідні методи.

```
def func():  1 usage
    obja = Circle(2)
    objb = Square( *args: 3, 5)
    objc = Triangle(5)

    for obj in (obja, objb, objc):
        print(obj)
```

3. Використовуючи мета-клас, вивести прізвище студента, який виконав роботу.

```
PS E:\СПП\Lab4> python lab4.py
Circle Дарина
Circle Периметр/Довжина: 12.56
Circle Площа: 12.56
Square Дарина
Square Периметр/Довжина: 16
Square Площа: 15
Triangle Дарина
Triangle Периметр/Довжина: 15
Triangle Площа: 10.825317547305483
```

Висновок:

У цій лабораторній роботі, я ознайомилась з основами метапрограмування та його застосуванням у сучасних мовах програмування, зокрема в Python. Вмію створювати більш гнучкі та динамічні програми, автоматизувати рутинні завдання та модифікувати поведінку програм під час виконання.

Додаток

Код програми:

```
from math import sqrt

Figure = type(
    'Figure',
    (object,),
```

```
    {
        '__init__': lambda self, a, b=None: setattr(
            self,
            '__dict__',
            {'a': a, 'b': b, 'name': self.__class__.__name__}
        ),
        '__str__': lambda self: f'{self.name} Дарина\n{self.name}
Периметр/Довжина: {self.perimetr()}\n{self.name} Площа: {self.area()}',
    },
)

Circle = type(
    'Circle',
    (Figure,),
    {
        'area': lambda self: 3.14 * self.a**2,
        'perimetr': lambda self: 2 * 3.14 * self.a,
    },
)

Triangle = type(
    'Triangle',
    (Figure,),
    {
        'area': lambda self: (sqrt(3) / 4) * self.a**2,
        'perimetr': lambda self: self.a * 3,
    },
)

Square = type(
    'Square',
    (Figure,),
    {
        'area': lambda self: self.a * self.b,
        'perimetr': lambda self: 2 * (self.a + self.b),
    },
)

def func():
    obja = Circle(2)
```

```
    objb = Square(3, 5)
    objc = Triangle(5)

    for obj in (obja, objb, objc):
        print(obj)

if __name__ == '__main__':
    func()
```