Домашнее задание
по теме «Метапрограммирование в Python»

Формулировка задания

Решить несколько задач на python с применением знаний о метапрограммировании и метаклассах. Результирующий код должен быть читаемым, с единой системой отступов и адекватными названиями переменных.

Описание плана работы

Решить несколько задач на python с применением знаний о метапрограммировании и метаклассах.

Желательна реализация в файлах py. Сохранить задачи (сделать коммиты для каждой) в локальном git и опубликовать в удаленном репозитории.

Для отчета по работе выполнить задание в файле.py или .ipynb. Сделать снимки экрана корректного выполнения программы в IDE.

Задача 1 Применение метаклассов

Метаклассы позволяют настраивать создание классов в Python. Определив метакласс для класса, вы можете контролировать создание экземпляров класса, его атрибуты и поведение.

Напишите метакласс Python **AttrLoggingMeta**, который регистрирует каждый доступ к атрибуту или его изменение. В метаклассе должен быть переопределен метод **__new__**. В **AttrLoggingMeta** добавить методы по логированию доступа **log_access(name, value),** чтению **log_read(name, value, instance)** и записи **log_write(name, value, instance)** атрибута класса.

Определите класс LoggedClass, используя AttrLoggingMeta в качестве его метакласса. Проверьте правильность реализации методов.

Пример использования в методе __main__:

```
instance = LoggedClass()

print(instance.custom_method)

instance.custom_method = 78

instance.other_custom_method()
```

На выходе программы:

```
Reading attribute custom_method

42

Writing attribute custom_method with value 78

Calling method other_custom_method
```

Задача 2 Динамическое создание класса

Динамическая природа Python позволяет изменять классы и объекты во время выполнения программы. Вы можете добавлять, удалять или изменять атрибуты и методы классов и объектов динамически с помощью встроенных функций, таких как 'setattr', 'getattr' и 'delattr'.

Напишите функцию Python **create_class_with_methods**, которая принимает имя класса, словарь атрибутов и словарь методов и возвращает динамически созданный класс с этими атрибутами и методами.

Для создания класса использовать метод **type.**

Пример использования в методе __main__:

```
attributes = { 'species': 'Human', 'age': 25 }

methods = { 'greet': lambda self: f"Hello, I am a {self.species} and I am {self.age} years old." }

DynamicClass = create_class_with_methods('DynamicClass', attributes, methods)

instance = DynamicClass()

print(instance.greet())
```

На выходе программы:

```
"Hello, I am a Human and I am 25 years old."
```

Задача 3 Генерация кода

Функции Python «exec» и «eval» позволяют выполнять динамически сгенерированный код во время выполнения. Эту функцию можно использовать для создания шаблонов кода, анализа

предметно-ориентированных языков (DSL) или реализации инструментов для генерации кода.

Напишите функцию Python **generate_complex_function**, которая принимает имя функции, список имён параметров и тело функции в виде строк и возвращает динамически сгенерированную функцию.

Пример использования в методе __main__:

```
function_name = 'complex_function'

parameters = ['x', 'y']

function_body = """

if x > y:

    return x - y

else:

    return y - x

"""

complex_func    =    generate_complex_function(function_name, parameters, function_body)

print(complex_func(10, 5))

print(complex_func(5, 10))
```

На выходе программы:

```
5

5
```

Перечень необходимых инструментов

- Python
- venv
- Jupiter Notebook
- IDE VS Code
- GigaIDE

Форма предоставления результата

1. В поле ссылки загрузить ссылку на удаленный репозиторий с доступом для наставника.
2. В поле файла загрузить архив с папкой, в которой разместить отчет со скриншотами по заданиям и решение задач 1-4. Решения должны быть представлены в формате .ipynb или .py.

Шкала оценивания

- 1.0 – отлично
- 0.7–0.9 – хорошо
- 0.5–0.6 – удовлетворительно
- Менее 0.5 – задание не выполнено