

ПРОГРАММА

ПРОФЕССИОНАЛЬНОЙ ПЕРЕПОДГОТОВКИ

**“Python-разработка для БПЛА: паттерны проектирования, API-интерфейсы и библиотеки для оптимизации решений”**

**Задание 5.**

**Работа с Singleton, Adapter, Decorator**

Преподаватель курса

[**Максимов Егор Васильевич**](https://www.odin.study/ru/User/Info/307187)

Выполнил

**Бубнович Антон Владимирович**

**№ потока** [**БПЛА\_256-1**](https://www.odin.study/ru/Cohort/Info/42290)

**Москва 2024 г.**

**Задание 5. Работа с Singleton, Adapter, Decorator**

**Инструкция:**

Шаг 1. Создать класс MySingleton. Добавить в него статическое поле instance и метод get\_instance для получения единственного экземпляра класса.

class MySingleton:

\_instance = None

def get\_instance(self):

pass

Шаг 2. В методе **init** класса MySingleton проверить, существует ли уже экземпляр в instance. Если нет - создать новый и сохранить в instance.

def \_\_init\_\_(self):

if not MySingleton.\_\_instance:

print(" \_\_init\_\_ method called..")

else:

print("Instance already created:", self.get\_instance())

Шаг 3.Вызвать метод get\_instance дважды в коде и убедиться, что он возвращает один и тот же экземпляр класса.

if \_\_name\_\_ == "\_\_main\_\_":

s1 = MySingleton()

s1.get\_instance()

s2 = MySingleton()

s2.get\_instance()

print(f"{s1} is {s2}")

Шаг 4. Создать класс MyClass с методом test. Создать класс MyClassAdapter, в нем вызвать метод test из экземпляра MyClass.

class MyClass:

def test(self):

print("функция из класса Myclass")

class MyClassAdapter(MyClass):

def call\_parent\_def(self):

MyClass().test()

if \_\_name\_\_ == "\_\_main\_\_":

MyClassAdapter().call\_parent\_def()function

Шаг 5. Создать функцию my\_function. Написать декоратор my\_decorator, который будет выводить имя функции перед ее вызовом.

def my\_decorator(func):

print(func.\_\_name\_\_)

Шаг 6. Применить декоратор к функции my\_function и убедиться, что при вызове функции выводится ее имя.

@my\_decorator

def my\_function():

pass