**Декорирование методов класса используя метакласс**

**Условия задачи**

Имеется три класса, использующие фабрику метакласса decorate\_all (функцию, возвращающую метакласс). Их реализация приведена ниже и должна быть использована в исходном коде без изменений:

class Module(metaclass=decorate\_all(call\_tracer, params\_logging)):

NOT\_EVALUATED: float = -1.

def \_\_init\_\_(self, title: str, date: datetime.date):

self.title: str = title

self.date: datetime.date = date

def \_\_hash\_\_(self):

return hash((self.title, self.date.isoformat()))

def \_\_repr\_\_(self):

return f"{self.\_\_class\_\_.\_\_name\_\_}: {self.title}"

class Student(metaclass=decorate\_all(call\_tracer, params\_logging)):

def \_\_init\_\_(self, name: str, teacher: "Teacher", modules: List[Module]):

self.name: str = name

self.teacher: "Teacher" = teacher

self.modules\_table: Dict[Module, float] = {module: Module.NOT\_EVALUATED for module in modules}

def \_\_repr\_\_(self):

return f"{self.\_\_class\_\_.\_\_name\_\_}: {self.name}"

def get\_mark\_by\_module(self, module: Module):

try:

return self.modules\_table[module]

except KeyError:

return None

def set\_mark\_by\_module(self, module: Module, mark: float):

self.modules\_table[module] = mark

def get\_passed\_modules(self):

return list(filter(lambda \_, v: v != Module.NOT\_EVALUATED, self.modules\_table))

class Teacher(metaclass=decorate\_all(call\_tracer, params\_logging)):

def \_\_init\_\_(self, name: str, level: int):

self.name: str = name

self.level: int = level

def \_\_repr\_\_(self):

return f"{self.\_\_class\_\_.\_\_name\_\_}: {self.name}"

def evaluate\_student\_by\_module(self, student: Student, module: Module, mark: float):

if self is student.teacher:

student.set\_mark\_by\_module(module=module, mark=mark)

Данная фабрика принимает на вход набор декораторов, после чего создаваемый метакласс должен применить их ко всем методам целевого класса, кроме магических.

**Исходные код для реализации**

from datetime import datetime

from types import FunctionType

from typing import List, Dict

def call\_tracer(\*args, \*\*kwargs):

pass

def params\_logging(\*args, \*\*kwargs):

pass

def decorate\_all(\*args, \*\*kwargs):

class MetaDecorate(\*margs, \*\*mkwargs):

pass

return MetaDecorate

mark = 5.

decorators\_module = Module(title="Decorators", date=datetime.today().date())

metaclasses\_module = Module(title="Metaclasses", date=datetime.today().date())

teacher = Teacher(name="Arthur", level=2)

rick = Student(name="Rick", teacher=teacher, modules=[decorators\_module])

morty = Student(name="Morty", teacher=teacher, modules=[decorators\_module, metaclasses\_module])

teacher.evaluate\_student\_by\_module(student=morty, module=metaclasses\_module, mark=mark)

assert morty.get\_mark\_by\_module(module=metaclasses\_module) == mark

**Требования к реализации**

\* написать эту фабрику (decorate\_all);

\* написать декоратор 'call\_tracer', выводящий количество вызовов задекорированного метода при его вызове;

\* написать декоратор 'params\_logging', выводящий входные параметры метода при вызове;

\* выполнить пункты выше используя Исходный код для реализации, чтобы получить ожидаемый вывод.

**Ожидаемый вывод**

The on\_call was called with params

args: (Teacher: Arthur,)

kwargs: {'student': Student: Morty, 'module': Module: Metaclasses, 'mark': 5.0}

The evaluate\_student\_by\_module was called 1 times

The on\_call was called with params

args: (Student: Morty,)

kwargs: {'module': Module: Metaclasses, 'mark': 5.0}

The set\_mark\_by\_module was called 1 times

The on\_call was called with params

args: (Student: Morty,)

kwargs: {'module': Module: Metaclasses}

The get\_mark\_by\_module was called 1 times