**27.6 Практическая работа**

Цели практической работы

Закрепить понятие функции как объекта высшего порядка, отработать умение использовать функцию в качестве аргумента.

Закрепить понятие декоратора и отработать умение его создавать и применять, а также применять специальный декоратор functools.wraps() для беспроблемного получения информации о декларируемой функции.

Что входит в работу

* Задача 1. Как дела?
* Задача 2. Замедление кода.
* Задача 3. Логирование.
* Задача 4. Счётчик.

Задача 1. Как дела?

Что нужно сделать

Вася совсем заскучал на работе и решил побаловаться с кодом проекта. Он написал надоедливый декоратор, который при вызове декорируемой функции спрашивает у пользователя «Как дела?», вне зависимости от ответа пишет что-то вроде «А у меня не очень!» и только потом запускает саму функцию. Правда, после такой выходки Васю чуть не уволили с работы.

Реализуйте такой же декоратор и проверьте его работу на нескольких функциях.

Пример кода:

@how\_are\_you  
def test():  
    print('<Тут что-то происходит...>')  
  
  
test()

Результат:  
Как дела? Хорошо.  
А у меня не очень! Ладно, держи свою функцию.  
<Тут что-то происходит...>

Что оценивается

* Результат вычислений корректен.
* Формат вывода соответствует примеру.
* Переменные, функции и собственные методы классов имеют значащие имена (не a, b, c, d).
* Классы и методы/функции имеют прописанную документацию.
* Есть аннотация типов для методов/функций и их аргументов (кроме args и kwargs). Если функция/метод ничего не возвращает, то используется None.
* Во всех декораторах используется functools.wraps().

Задача 2. Замедление кода

Что нужно сделать

В программировании иногда возникает ситуация, когда работу функции нужно замедлить. Типичный пример — функция, которая постоянно проверяет, изменились ли данные на веб-странице или её код.

Реализуйте декоратор, который перед выполнением декорируемой функции ждёт несколько секунд.

Что оценивается

* Результат вычислений корректен.
* Сообщения о процессе получения результата осмыслены и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена (не a, b, c, d).
* Классы и методы/функции имеют прописанную документацию.
* Есть аннотация типов для методов/функций и их аргументов (кроме args и kwargs). Если функция/метод ничего не возвращает, то используется None.
* Во всех декораторах используется functools.wraps().

Задача 3. Логирование

Что нужно сделать

Реализуйте декоратор logging, который будет отвечать за логирование функций. На экран выводится название функции и её документация. Если во время выполнения декорируемой функции возникла ошибка, то в файл function\_errors.log записываются название функции и ошибки.

Также постарайтесь сделать так, чтобы программа не завершалась после обнаружения первой же ошибки, а обрабатывала все декорируемые функции и сразу записывала все ошибки в файл.

Дополнительно: запишите дату и время возникновения ошибки, используя модуль datetime.

Что оценивается

* Результат вычислений корректен.
* Сообщения о процессе получения результата осмыслены и понятны для пользователя.
* Переменные, функции и собственные методы классов имеют значащие имена (не a, b, c, d).
* Классы и методы/функции имеют прописанную документацию.
* Есть аннотация типов для методов/функций и их аргументов (кроме args и kwargs). Если функция/метод ничего не возвращает, то используется None.
* Во всех декораторах используется functools.wraps().

Задача 4. Счётчик

Что нужно сделать

Реализуйте декоратор counter, считающий и выводящий количество вызовов декорируемой функции.

Для решения задачи нельзя использовать операторы global и nonlocal (об этом мы ещё расскажем).

Что оценивается

* Результат вычислений корректен.
* Переменные, функции и собственные методы классов имеют значащие имена (не a, b, c, d).
* Классы и методы/функции имеют прописанную документацию.
* Есть аннотация типов для методов/функций и их аргументов (кроме args и kwargs). Если функция/метод ничего не возвращает, то используется None.
* Во всех декораторах используется functools.wraps().

Что оценивается в практической работе

* Практическая работа сдана через GitLab.
* Структура папок и файлов репозитория соответствует репозиторию python\_basic.
* Все задачи выполнены в соответствующих папках и файлах main.py.
* Описания коммитов осмысленные и понятные: 111, done, «я сделалъ» — неверно, added m15 homework, 14.3 fix: variables naming — верно.
* Использованы именованные индексы, не просто i (подробнее об этом в видео 7.2).
* Использованы правильные числа, без дополнительных действий со стороны пользователя, без +1 (подробнее об этом в видео 7.4).
* Правильно оформлен input, без пустого приветствия для ввода (подробнее об этом в видео 2.3).
* Переменные и функции имеют значащие имена, не только a, b, c, d (подробнее об этом в видео 2.3).
* Присутствуют пробелы после запятых и при бинарных операциях.
* Отсутствуют пробелы после имён функций и перед скобками: print (),input () — неверно, print() — верно.
* Правильно оформлены блоки if-elif-else, циклы и функции, отступы одинаковы во всех блоках одного уровня.
* Все входные и выходные файлы называются так, как указано в задачах.
* Работа с файлами осуществляется с помощью контекстного менеджера with.
* Для обработки исключений используются блоки try-except.
* Модели реализованы в стиле ООП, основной функционал описан в методах классов и в отдельных функциях.
* При написании классов соблюдаются основные принципы ООП: инкапсуляция, наследование и полиморфизм.
  + Для получения и установки значений у приватных атрибутов используются сеттеры и геттеры.
  + Для создания нового класса на основе уже существующего используется наследование.
* Если классы вынесены в отдельный модуль, то импортируются определённые классы (запись вида from garden import \* считается плохим тоном).
* Классы и методы/функции имеют прописанную документацию (хотя бы минимальную).
* Есть аннотация типов для методов/функций и их аргументов (кроме args и kwargs). Если функция/метод ничего не возвращает, то используется None.
* Во всех декораторах используется functools.wraps().

Советы и рекомендации

* Арифметические операции [PEP 8](https://docs.python.org/3.7/reference/expressions.html#operator-precedence)остаются в приоритете. Необходимо вводить and, or.
* Руководство по стилю Python [PEP 8](https://www.python.org/dev/peps/pep-0008/) на английском языке.
* Руководство по стилю Python [PEP 8](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html) на русском языке.
* [Список встроенных функций.](https://docs.python.org/3.7/library/functions.html)

Как отправить работу на проверку

Чтобы выполнить практическую работу, обновите репозиторий python\_basic на своём компьютере при помощи IDE PyCharm. Задачи находятся в папке Module27.

Сдайте практические работы этого модуля через систему контроля версий Git сервиса Skillbox GitLab. В форме для сдачи работы напишите «Сделано» и прикрепите ссылку на репозиторий. Ссылки на реплит оставлять не нужно.