top

КОМПЬЮТЕРНАЯ АКАДЕМИЯ

Основы Python

Тема «Декораторы»

Декораторы в Python — это удобный инструмент, который позволяет добавлять функциональность к функциям или классам без изменения их исходного кода.

Представим, что мы хотим украсить торт. Мы не меняем сам торт, а добавляем сверху украшения. Так же работают декораторы — они «украшают» наши функции, делая их более гибкими и функциональными.

Сегодня мы:

- **поймем**, что такое декораторы и как они работают.
- **научимся** писать простые и многослойные декораторы.
- **будем использовать** встроенные декораторы Python, такие как @staticmethod, @classmethod и @property.
- **узнаем**, как декораторы применяются в реальных проектах, например, для авторизации пользователей или ведения логов.
- **разберемся**, как декораторы помогают избегать повторения кода и улучшить читаемость программ.

Глоссарий к семнадцатому занятию

Decorator (Декоратор) — инструмент, позволяющий модифицировать поведение функции, метода или класса.

Wrapper (Обёртка) — функция, которая используется внутри декоратора для добавления дополнительной логики.

Function (Функция) — блок кода, выполняющий определенное действие.

Reusable (Переиспользуемый) — что-то, что можно использовать снова и снова.

Code readability (Читаемость кода) — насколько легко читать и понимать ваш код.

Функции

В Python **функции** — это объекты, и их **можно передавать** в другие функции как аргументы.

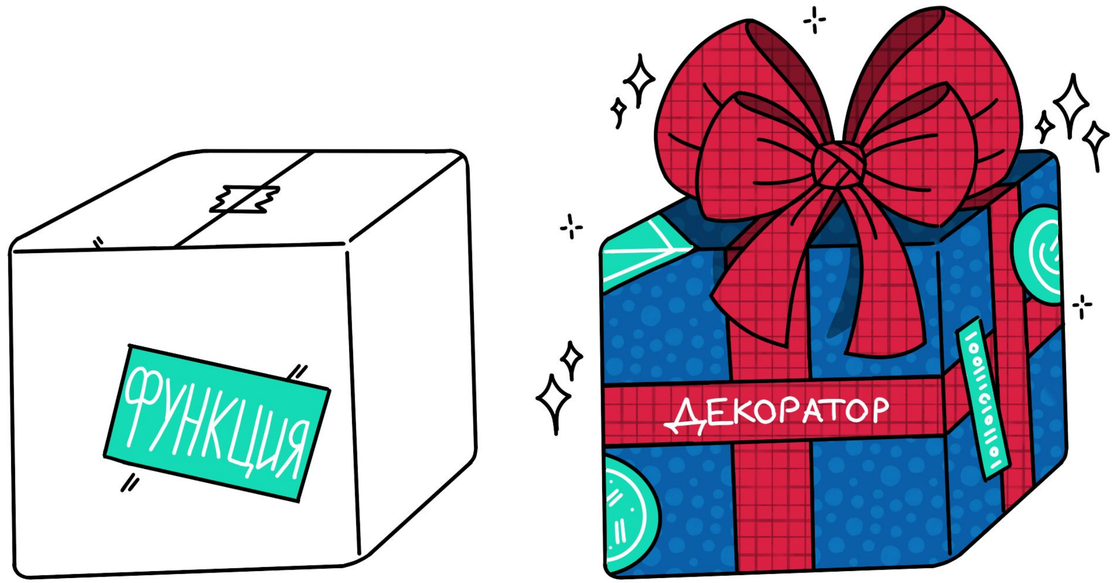
Это позволяет писать код, где одна функция может управлять поведением другой. Эта идея лежит в основе декораторов.

Именно благодаря такой гибкости Python позволяет создавать декораторы, которые оборачивают функции и добавляют им новую функциональность.

Декораторы в Python

Декораторы в Python — это особые функции, которые позволяют оборачивать другие функции или классы, добавляя им дополнительную функциональность.

Они помогают изменить или расширить поведение функции или метода без изменения их исходного кода.



Польза декораторов

Декораторы полезны, когда мы хотим:

- Добавить новую функциональность без изменения существующего кода.
- Избежать повторения кода.
- Сделать ваш код более читаемым и модульным.

Как работают декораторы?

Декоратор — это функция, которая принимает другую функцию как аргумент, выполняет некоторую дополнительную логику и возвращает новую функцию.

Пример 1: Простая функция-декоратор

```
def decorator(func):
    def wrapper():
        print("До вызова функции")
        func()
        print("После вызова функции")
    return wrapper

@decorator
def say_hello():
    print("Привет, мир!")

say_hello()
```

Пример 1: Простая функция-декоратор

Декоратор @decorator оборачивает функцию say_hello внутри функции wrapper. При вызове say_hello() сначала выполняется код в wrapper, а затем вызывается сама функция say_hello.

```
До вызова функции
Привет, мир!
После вызова функции
Как пишутся декораторы?
```

Пример 2: Декоратор с аргументами

```
def decorator(func):
    def wrapper(name):
        print("До вызова функции")
        func(name)
        print("После вызова функции")
    return wrapper

@decorator
def greet(name):
    print(f"Привет, {name}!")

greet("Иван")
```

Пример 2: Декоратор с аргументами

Декораторы могут работать и с функциями, принимающими аргументы.

```
До вызова функции
Привет, Иван!
После вызова функции
```

Встроенные декораторы в Python

Python предоставляет несколько встроенных декораторов:

1. **@staticmethod**

Позволяет создать метод, который не требует доступа к экземпляру класса или его атрибутам.

2. **@classmethod**

Создает метод, который получает доступ к самому классу, а не к его экземпляру.

3. **@property**

Используется для создания методов, которые работают как атрибуты.

Как работают декораторы с несколькими слоями?

Мы можем применять несколько декораторов к одной функции. Они выполняются сверху вниз.

Декораторы и реальная жизнь

1. **Логирование (Logging)**

   Декораторы часто используются для автоматического ведения журнала работы функций

2. **Авторизация (Authorization)**

   Проверяют, имеет ли пользователь доступ к выполнению определенных операций.

Преимущества декораторов:

- Упрощают добавление новой функциональности.
- Повышают читаемость и модульность кода.
- Помогают избежать дублирования кода.

top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

Практикум

Задача 1: Логирование действий пользователя

Ситуация: мы разрабатываем приложение, которое отслеживает действия пользователей, такие как вход в систему, обновление профиля или отправка сообщения. Для каждого действия нужно сохранять лог с именем пользователя и названием выполненной функции. Лог-файлы позволяют анализировать действия пользователей и выявлять ошибки в работе системы.

Задача: создать декоратор log_action, который:

1. Логирует имя пользователя и выполняемое действие.
2. Сохраняет эту информацию в текстовый файл actions.log.
3. Работает с любыми функциями, которые принимают username как первый аргумент.

Задача 2: Авторизация доступа к секретным данным

Ситуация: мы разрабатываем систему для управления секретными данными, доступ к которым должен быть только у пользователей с правами администратора. Необходимо автоматически проверять права доступа перед выполнением функции.

Задача: создать декоратор authorize_admin, который:

1. Проверяет, является ли пользователь администратором.
2. Если пользователь администратор, выполняет функцию.
3. Если пользователь не администратор, выводит сообщение "Доступ запрещен".

Итоги занятия

- **изучили**, как работают декораторы и их назначение.
- **создавали** собственные декораторы для изменения поведения функций.
- **применяли** встроенные декораторы @staticmethod, @classmethod и @property.
- **писали** многослойные декораторы и применяли их на практике.
- **реализовывали** декораторы для задач: логирования, авторизации и измерения времени выполнения.

top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

Домашняя работа

Описание задачи: создайте программу для управления списком задач с использованием декораторов.

Пользователь должен иметь возможность:

1. Добавлять задачи в список.
2. Просматривать все задачи.
3. Выполнять задачи (удалять их из списка).
4. Отменять последние изменения в списке (восстанавливать удалённые задачи).

Указания:

1. Программа должна поддерживать следующие команды:

- add: добавляет новую задачу.
- show: показывает список всех задач.
- complete: помечает задачу как выполненную (удаляет её из списка).
- undo: отменяет последнее изменение (восстанавливает последнюю удалённую задачу).
- exit: завершает программу.

Указания:

2. Используйте декоратор для логирования действий пользователя:

- Логируйте каждое добавление, выполнение или восстановление задачи.
- Выводите информацию о времени выполнения действия.

3. Реализуйте отмену последнего действия с использованием стека для хранения удалённых задач.

4. Добавьте обработку ошибок, например, ввод неверной команды или попытка выполнения задачи из пустого списка.

Ожидаемый результат:

1. Добавление задачи: при вводе команды add, программа запрашивает текст задачи у пользователя. Задача добавляется в список, и выводится сообщение: "Задача добавлена."
2. Просмотр задач: при вводе команды show, программа отображает все задачи с их номерами. Если список пуст, выводится сообщение: "Список задач пуст."
3. Выполнение задачи: при вводе команды complete, программа запрашивает номер задачи для выполнения. Задача удаляется из списка, и выводится сообщение: "Задача выполнена." Удаленная задача сохраняется в стеке для возможности восстановления.

Ожидаемый результат:

4. Отмена действия: при вводе команды undo, программа восстанавливает последнюю удалённую задачу. Если стек пуст, выводится сообщение: "Нет действий для отмены."

5. Завершение работы: при вводе команды exit, программа завершает работу с сообщением: "Программа завершена."

Дополнение:

Для соблюдения принципов ООП добавьте метод run в класс TaskManager. Этот метод будет отвечать за основной цикл программы, обрабатывая команды add, show, complete, undo, exit. Это сделает код более структурированным.

Критерии оценивания:

4 балла — Корректность: Программа правильно выполняет добавление, удаление, восстановление и отображение задач.

4 балла — Использование декораторов: Логирование действий реализовано через декоратор, выводится время выполнения действий.

4 балла — Удобство использования: Интерфейс программы интуитивно понятен, сообщения об ошибках и действиях информативны.

Максимальный балл за задание — 12 баллов.