Лекция 15

Здравствуйте, уважаемые слушатели!
Тема нашей лекции – Продвинутые темы Python: декораторы, контекстные менеджеры и генераторы

План лекции:
1. Декораторы
2. Контекстные менеджеры
3. Генераторы
4. Заключение

Введение

Python известен своей гибкостью и мощными, но доступными инструментами для разработки. Среди этих инструментов выделяются декораторы, контекстные менеджеры и генераторы, каждый из которых предлагает уникальные возможности для оптимизации кода, управления ресурсами и упрощения сложных задач. Эта лекция посвящена глубокому погружению в эти продвинутые концепции Python, предоставляя примеры их применения и объясняя, как они могут улучшить эффективность и читаемость вашего кода.

1. Декораторы

1.1 Основы декораторов

Декораторы в Python — это функции, которые модифицируют поведение других функций или методов. Они предоставляют простой способ изменения функциональности функций без изменения их кода. Декораторы оборачивают другую функцию, позволяя выполнять код до и после непосредственно вызываемой функции.

1.1.1 Пример декоратора

python
Копировать код
```
def my_decorator(func):
    def wrapper():
        print("Что-то происходит перед вызовом функции.")
        func()
        print("Что-то происходит после вызова функции.")
    return wrapper

@my_decorator
```

```
def say_hello():
    print("Hello!")

say_hello()
```

1.2 Применение декораторов

Декораторы часто используются для логирования, измерения времени выполнения, проверки доступа, кэширования и других аспектов управления поведением функций.

2. Контекстные менеджеры

2.1 Что такое контекстные менеджеры?

Контекстные менеджеры управляют контекстом выполнения блока инструкций через протоколы __enter__ и __exit__. Они часто используются для обеспечения "сетапа" и "теардауна" в операциях, которые требуют освобождения ресурсов, таких как работа с файлами или сетевыми соединениями.

2.1.1 Пример контекстного менеджера

python
Копировать код
```
class ManagedFile:
    def __init__(self, filename):
        self.filename = filename

    def __enter__(self):
        self.file = open(self.filename, 'w')
        return self.file

    def __exit__(self, exc_type, exc_value, traceback):
        if self.file:
            self.file.close()

with ManagedFile('hello.txt') as f:
    f.write('Hello, world!')
```

2.2 Контекстный менеджер и with statement

Python предлагает ключевое слово with, которое упрощает использование контекстных менеджеров, гарантируя, что методы __enter__ и __exit__ будут вызваны в нужные моменты.

3. Генераторы

3.1 Введение в генераторы

Генераторы — это функции, которые возвращают итерируемый объект, по одному значению за раз, используя ключевое слово yield вместо return. Генераторы позволяют программе генерировать большие объемы данных без необходимости держать их все в памяти.

3.1.1 Пример генератора

python
Копировать код

```
def countdown(num):
    print("Начинаем отсчет")
    while num > 0:
        yield num
        num -= 1

for x in countdown(5):
    print(x)
```

3.2 Преимущества генераторов

Генераторы особенно полезны при работе с большими данными или при выполнении операций, которые могут быть описаны как "ленивые вычисления", то есть выполнение наступает только по мере необходимости (при итерации).

4. Заключение

Продвинутые концепции Python, такие как декораторы, контекстные менеджеры и генераторы, представляют собой мощные инструменты для оптимизации кода, управления ресурсами и повышения его эффективности. Их правильное применение может значительно улучшить структуру программы и сделать код более читаемым, безопасным и легко поддерживаемым. Эти возможности делают Python особенно привлекательным для разработчиков, стремящихся создавать чистый, эффективный и масштабируемый код.

Литературы:
1. Программирование на Python для начинающих. Райтман М.А. Москва-ЭМОСКО-2015 стр 112-119
2. Python для «чайников». Джон Полль Мюллер. Диалектика-2022 стр. 101-115