

Лекция 8

Здравствуйте, уважаемые слушатели!

Тема нашей лекции – Объектно-ориентированное программирование: классы и объекты

План лекции:

1. Концепция классов и объектов
2. Наследование
3. Инкапсуляция
4. Полиморфизм и деструкторы
5. Заключение

Введение

Объектно-ориентированное программирование (ООП) является ключевым парадигмом в современной разработке программного обеспечения, предоставляя мощные инструменты для управления сложностью в больших программах. Python поддерживает ООП и предоставляет все необходимые средства для работы с классами и объектами. В данной лекции мы подробно рассмотрим, как Python реализует концепции ООП, включая создание классов, инстанцирование объектов, наследование, инкапсуляцию и полиморфизм.

1. Концепция классов и объектов

1.1 Определение класса

В Python класс определяется с помощью ключевого слова `class`. Класс служит шаблоном для создания объектов (экземпляров), обеспечивая начальное состояние (атрибуты) и поведение (методы) этих объектов.

```
python
```

Копировать код

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def speak(self):
        return f"{self.name} says woof!"
```

В приведенном примере определен класс `Dog` с конструктором `__init__`, который инициализирует новый объект с именем и возрастом. Также класс содержит метод `speak`.

1.2 Создание объектов

Объекты создаются путем вызова класса как функции, передавая аргументы, которые принимает метод `__init__`.

```
python
Копировать код
my_dog = Dog("Rex", 5)
print(my_dog.speak()) # Rex says woof!
```

1.3 Атрибуты и методы

Атрибуты класса — это переменные, которые связаны с классом. Методы класса — это функции, определенные в классе, которые могут изменять объект или взаимодействовать с его атрибутами.

2. Наследование

Наследование — это способность класса наследовать атрибуты и методы другого класса. В Python наследование реализуется путем указания родительского класса в определении класса.

```
python
Копировать код
class Poodle(Dog):
    def speak(self):
        return f"{self.name} says yap!"
```

Здесь класс Poodle наследует от Dog и переопределяет метод speak.

3. Инкапсуляция

Инкапсуляция — это ограничение доступа к составляющим объект компонентам класса. В Python нет ключевых слов для строгой инкапсуляции, но соглашение состоит в использовании подчеркивания перед именем атрибута для указания того, что он должен использоваться только внутри класса.

```
python
Копировать код
class Cat:
    def __init__(self, name, age):
        self._name = name # Инкапсулированный атрибут
        self._age = age

    def speak(self):
```

```
return f"{self._name} says meow!"
```

4. Полиморфизм

Полиморфизм в ООП позволяет методам иметь одинаковые имена, но различное поведение в зависимости от класса. В Python полиморфизм обычно достигается через наследование и переопределение методов.

```
python
```

Копировать код

```
def pet_speak(pet):  
    print(pet.speak())
```

```
pet_speak(my_dog) # Rex says woof!  
pet_speak(Poodle("Molly", 3)) # Molly says yap!
```

5. Деструкторы

В Python деструктор `__del__` вызывается при уничтожении объекта. Это может быть полезно для освобождения ресурсов.

```
python
```

Копировать код

```
class Example:  
    def __del__(self):  
        print("Destructor called, Example deleted.")
```

Заключение

Объектно-ориентированное программирование в Python предоставляет программистам мощные инструменты для создания структурированных и модульных программ. Использование классов, объектов, наследования и полиморфизма позволяет упростить управление большими кодовыми базами и улучшить переиспользование кода. Понимание этих концепций является ключевым для разработки эффективного программного обеспечения на Python и обеспечивает основу для дальнейшего изучения и практического применения ООП.

На этом наш вводный урок заканчивается.

Литературы:

1. Программирование на Python для начинающих. Райтман М.А. Москва-ЭМОСКО-2015 стр 65-69
2. Python для «чайников». Джон Полль Мюллер. Диалектика-2022 стр. 57-63