Лекция 3

Классы и объекты: основы создания и использования

1. Введение в классы и объекты

Классы и объекты являются основными строительными блоками объектно-ориентированного программирования (ООП). Этот подход к разработке программного обеспечения позволяет моделировать реальные сущности с помощью кода, делая его более структурированным, гибким и удобным для поддержки. В основе ООП лежат четыре ключевых принципа — инкапсуляция, наследование, полиморфизм и абстракция. Однако классы и объекты служат основой, на которой построены все остальные концепции.

Класс представляет собой шаблон, определяющий свойства и поведение объектов, которые будут создаваться на его основе. Объект — это конкретный экземпляр класса, который обладает своим состоянием и методами для выполнения действий. Например, класс «Автомобиль» может иметь такие свойства, как цвет, марка и модель, а объекты класса «Автомобиль» будут представлять собой конкретные автомобили с уникальными значениями этих свойств.

Цель этой лекции — объяснить концепцию классов и объектов, их структуру, способы создания и использования, а также их роль в разработке программного обеспечения.

2. Классы: структура и создание

Класс можно рассматривать как шаблон или чертеж, определяющий структуру и поведение объектов. Он состоит из атрибутов (переменных) и методов (функций), которые описывают, что объекты могут делать и какими характеристиками обладают.

2.1 Определение класса

В большинстве объектно-ориентированных языков программирования класс объявляется с помощью ключевого слова class, за которым следует имя класса. Например, определим класс Car на языке Python:

python
Копировать код
```
class Car:
    pass
```

Здесь pass указывает, что класс пуст и не содержит ни атрибутов, ни методов. Это минимальное определение класса, которое можно расширить, добавив переменные и функции.

2.2 Атрибуты класса

Атрибуты — это переменные, которые определяются внутри класса и представляют собой его свойства. Атрибуты могут быть двух типов:

- **Атрибуты класса**: Задаются для всего класса и общие для всех объектов этого класса. Они объявляются непосредственно в теле класса.
- **Атрибуты объекта (экземпляра)**: Создаются для каждого объекта отдельно и могут иметь уникальные значения для каждого объекта.

Пример:

python
Копировать код
```
class Car:
    wheels = 4  # атрибут класса

    def __init__(self, color, model):
        self.color = color  # атрибут объекта
        self.model = model  # атрибут объекта
```

В этом примере wheels — атрибут класса Car, который одинаков для всех объектов, тогда как color и model — атрибуты объекта, которые могут иметь различные значения для каждого экземпляра.

2.3 Методы класса

Методы — это функции, определенные внутри класса, которые описывают поведение объектов. Наиболее важным методом является __init__, который выполняет инициализацию объекта при его создании. Этот метод называется конструктором и позволяет задать начальные значения для атрибутов объекта.

Пример:

python
Копировать код
```
class Car:
    wheels = 4

    def __init__(self, color, model):
        self.color = color
        self.model = model
```

```
    def drive(self):
        return f"The {self.color} {self.model} is driving"
```

В этом примере метод drive определяет поведение объекта Car. При вызове метода на объекте он вернет сообщение, описывающее действие объекта.

3. Объекты: создание и использование

Объект — это конкретный экземпляр класса, который создается с помощью вызова конструктора. Каждый объект имеет свое собственное состояние, определяемое значениями его атрибутов.

3.1 Создание объекта

Создание объекта осуществляется с помощью вызова имени класса, как если бы это была функция, что активирует конструктор __init__:

python
Копировать код
```
my_car = Car("red", "Toyota")
```

В этом случае создается объект my_car класса Car с цветом red и моделью Toyota. Теперь my_car представляет собой уникальный объект, обладающий собственными атрибутами color и model.

3.2 Доступ к атрибутам и методам объекта

Доступ к атрибутам и методам объекта осуществляется с помощью оператора точки (.):

python
Копировать код
```
print(my_car.color)  # выводит "red"
print(my_car.drive())  # выводит "The red Toyota is driving"
```

Здесь my_car.color возвращает значение атрибута color объекта my_car, а my_car.drive() вызывает метод drive для выполнения действия.

3.3 Изменение состояния объекта

Атрибуты объекта можно изменять после его создания, изменяя его состояние:

python
Копировать код
```
my_car.color = "blue"
print(my_car.drive())  # выводит "The blue Toyota is driving"
```

Это позволяет динамически обновлять состояние объекта в зависимости от условий программы.

4. Конструкторы и деструкторы

Конструктор __init__ и деструктор __del__ — это специальные методы, которые управляют жизненным циклом объекта, обеспечивая его корректное создание и удаление.

4.1 Конструктор __init__

Метод __init__ — это специальный метод, который вызывается автоматически при создании нового объекта. Он позволяет инициализировать атрибуты объекта и выполнять другие необходимые действия.

Пример:

python
Копировать код
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Здесь метод __init__ инициализирует атрибуты name и age при создании объекта Person.

4.2 Деструктор __del__

Метод __del__ — это специальный метод, который вызывается перед уничтожением объекта. Он используется для выполнения действий, необходимых перед освобождением ресурсов, таких как закрытие файлов или завершение соединений.

Пример:

python
Копировать код
```
class FileHandler:
    def __init__(self, file_name):
        self.file = open(file_name, 'w')

    def __del__(self):
        self.file.close()
        print("File closed")
```

В этом примере метод __del__ закрывает файл при уничтожении объекта FileHandler, что гарантирует освобождение ресурсов.

5. Инкапсуляция и скрытие данных

Инкапсуляция — это один из основных принципов ООП, который обеспечивает контроль над доступом к данным. Она позволяет защитить данные от некорректного использования и ошибок. Для достижения инкапсуляции используются приватные атрибуты и методы, доступ к которым возможен только изнутри класса.

5.1 Приватные атрибуты и методы

Приватные атрибуты и методы обозначаются префиксом _ или __ (двойное подчеркивание). Они недоступны напрямую за пределами класса и защищают внутреннее состояние объекта.

Пример:

python
Копировать код

```
class Account:
    def __init__(self, owner, balance):
        self.owner = owner
        self.__balance = balance  # приватный атрибут

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def get_balance(self):
        return self.__balance
```

В этом примере __balance является приватным атрибутом, доступ к которому осуществляется только через методы deposit и get_balance.

5.2 Геттеры и сеттеры

Геттеры и сеттеры — это методы, которые позволяют контролировать доступ к приватным атрибутам, предоставляя возможность чтения и изменения их значений при необходимости.

Пример:

python
Копировать код

```
class Temperature:
    def __init__(self, celsius):
        self.__celsius = celsius

    def get_celsius(self):
        return self.__celsius

    def set_celsius(self, value):
        if value >= -273.15:  # проверка на физически допустимое значение
            self.__celsius = value
```

Здесь методы get_celsius и set_celsius предоставляют безопасный доступ к атрибуту __celsius.

6. Полезные функции и методы для работы с классами

Для работы с классами и объектами в Python предусмотрены специальные методы и функции:

- **__str__** и **__repr__**: Эти методы позволяют определить, как объект будет отображаться при выводе. __str__ используется для пользовательского отображения, а __repr__ — для внутреннего представления.
- **isinstance**: Проверяет, является ли объект экземпляром определенного класса.
- **issubclass**: Проверяет, является ли класс подклассом другого класса.

Пример:

python
Копировать код

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

    def __str__(self):
        return f"{self.name}, {self.age} years old"

john = Person("John", 30)
print(john)  # выводит "John, 30 years old"
```

7. Заключение

Классы и объекты являются основными строительными блоками объектно-ориентированного программирования, предоставляя способ моделировать реальные сущности и их поведение в программном коде. Классы служат

шаблонами, которые описывают общие свойства и функции, а объекты представляют собой конкретные экземпляры этих классов с собственными значениями атрибутов. Использование классов и объектов позволяет создавать гибкие, масштабируемые и структурированные программы, в которых логика взаимодействия между сущностями выражена на высоком уровне абстракции. Эти принципы делают ООП мощным инструментом для разработки сложного программного обеспечения, поддерживая его простоту и понятность.