КЛАССЫ И НАСЛЕДОВАНИЕ

ВВЕДЕНИЕ

Задачи урока

01 Познакомиться с классами

02 Узнать что такое ООП

03 Понять как можно использовать классы и как они облегчают поддержку и расширение функционала

Классы

Классы являются основной концепцией объектно-ориентированного программирования в Python. Они позволяют определять собственные типы данных и группировать функции, методы и атрибуты, относящиеся к этим типам данных

Определение класса в Python начинается с ключевого слова class, за которым следует имя класса. Имя класса обычно следует стандарту CamelCase, то есть каждое новое слово начинается с большой буквы без использования пробелов или подчеркиваний

Вот пример простого класса в Python

```
class Employee:

    def __init__(self, age, name, salary) -> None:
        self.age = age
        self.name = name
        self.salary = salary

    def set_salary(self, salary):
        self.salary = salary
```

Конструктор класса

это специальный метод, который автоматически вызывается при создании нового объекта класса. В Python конструктор класса называется _init_()

Конструктор выполняет инициализацию объекта и устанавливает начальные значения его атрибутов. Он может принимать параметры, которые передаются при создании объекта класса

Вот пример простого класса Person, имеющего конструктор

```
class Employee:

    def __init__(self, age, name, salary) -> None:
        self.age = age
        self.name = name
        self.salary = salary
```

Для создания объекта класса мы вызываем его конструктор, указывая необходимые аргументы, **как показано ниже**

```
employee = Employee(30, 'Владислав', 100000)
```

Наследование

Наследование

это механизм объектно-ориентированного программирования, позволяющий создавать новые классы на основе уже существующих. Класс, от которого происходит наследование, называется базовым классом или родительским классом, а класс, который наследуется, называется производным классом или дочерним классом

В Python для создания производного класса от базового класса используется следующий синтаксис:

```
class Accountant(Employee):
 …
```

При наследовании дочерний класс получает все атрибуты и методы базового класса, и может добавлять свои собственные атрибуты и методы или переопределять методы базового класса по своему усмотрению

В Python для создания производного класса от базового класса используется следующий синтаксис:

```
class Employee:

    def __init__(self, age, name, salary) -> None:
        self.age = age
        self.name = name
        self.salary = salary

    def set_salary(self, salary):
        self.salary = salary


class Accountant(Employee):

    def __init__(self, age, name, salary) -> None:
        super().__init__(age, name, salary)
        self.audit = False

    def set_audit(self, is_audit):
        self.audit = is_audit

    def conducts_audit(self):
        return self.audit
```

ИТОГИ

- Важно использовать классы, объединенные одной логикой работы. Таким образом, если в логике будут изменения, нам достаточно внести их в логику класса
- Для облегчения задачи разработки и для избежания повторного написания кода, мы используем наследования. В базовом классе делаем основные методы, которые будут во всех наследниках.
  А в наследниках просто используем базовые классы. Это позволяет облегчить поддержку кода, так как при изменениях их нужно внести только в одном месте в методах базового класса