**Объектно-ориентированное программирование в языке Python**

**Слайд 1**

В Python классы — это ключевой элемент объектно-ориентированного программирования (ООП). Они позволяют объединять данные и функции, работающие с этими данными, в единую структуру. Все мы знаем, что такое класс, поэтому здесь долго останавливаться не будем, а сразу перейдём к примерам.

**Слайд 2**

Для создания класса используется ключевое слово class.

Объект создаётся вызовом класса, как функции

Также у класса, как и в других языках, для создания объекта используется конструктор. В него неявно передаётся параметр self, обозначающий текущий объект. Далее мы также можем реализовать уже какую-то логику в конструкторе.

**Слайд 3**

Далее затронем атрибуты класса.

Атрибуты — это переменные, которые принадлежат классу или его экземплярам.

* **Атрибуты экземпляра (instance attributes)**: создаются и управляются в методе \_\_init\_\_.
* **Атрибуты класса (class attributes)**: объявляются непосредственно в теле класса и доступны всем экземплярам.

**Слайд 4**

Здесь вариант, когда мы меняем атрибут класса внутри конструктора. И тогда этот атрибут становится не полем класса, а полем экземпляра (о чём я и написал на слайде: он приватизируется). Это не термин, но легко себе представить, как объект забирает свойство, являющееся общим для всех сущностей класса, и задаёт ему своё кастомной значение.

**Слайд 5**

Методы — это функции, определённые внутри класса, которые работают с его атрибутами и могут изменять состояние объекта.

* **Методы экземпляра (instance methods)**: работают с конкретным экземпляром класса. Первым аргументом всегда передаётся self, который ссылается на текущий экземпляр.
* **Методы класса (class methods)**: работают с классом в целом. Первый аргумент — cls (ссылка на сам класс). Определяются с помощью декоратора @classmethod.
* **Статические методы (static methods)**: не изменяют ни состояние класса, ни состояние экземпляра. Определяются с помощью декоратора @staticmethod.

**Слайд 6**

Инкапсуляция скрывает внутреннее состояние объекта и позволяет контролировать доступ к данным через методы. В Python нет строгой инкапсуляции, но существуют соглашения:

* **Публичные атрибуты**: доступны отовсюду, например, self.name.
* **Защищённые атрибуты**: начинаются с одного подчёркивания (\_self.attribute). Предполагается, что они не должны использоваться вне класса, но Python не посчитает ошибкой обращение к такому атрибуту вне класса.
* **Приватные атрибуты**: начинаются с двух подчёркиваний (\_\_self.attribute). Имена таких атрибутов подвергаются манглингу (перемешиванию), чтобы их было трудно случайно перезаписать.

**Слайд 7**

Наследование позволяет создавать новый класс на основе существующего. Новый класс (дочерний или производный) наследует атрибуты и методы родительского (базового) класса.

**Слайд 8**

**Слайд 9**

Полиморфизм позволяет использовать один интерфейс для разных типов объектов. Это означает, что методы, имеющие одно и то же имя, могут вести себя по-разному в зависимости от объекта, который их вызывает.

**Слайд 10**

**Слайд 11**

**Слайд 11**

**Слайд 12**

**Слайд 13**

В Python описание метода — это текстовое объяснение того, что делает данный метод, его аргументы и возможный результат. Обычно это описание оформляется в виде **докстринга** (docstring) и размещается непосредственно под определением метода в тройных кавычках. Докстринг позволяет легко понять, что делает метод.