**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 (база)**

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 (атрибуты класса)**

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 (ещё атрибуты класса)**

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 класса)**

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 такие методы с помощью декоратора @classmethod. Кстати, если вызвать метод класса от объекта (а так можно), то он подменит сущность класса.  
  И здесь заметно сходство с вызовом от объекта. Там первым аргументом является объект, от которого мы вызываемся, а здесь первым аргументом является ссылка на класс. Таким образом, та сущность, которая вызывает метод (указана слева от точки), всегда первой передаётся в него в качестве аргумента.
* **Статические методы (static methods)**: не изменяют ни состояние класса, ни состояние экземпляра. Определяются с помощью декоратора @staticmethod.

**Слайд 6 (инкапсуляция)**

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)**

Наследование позволяет создавать новый класс на основе существующего. Новый класс (дочерний или производный) наследует атрибуты и методы родительского (базового) класса. Для наследования от базового класса необходимо в скобках от класса написать имя класса, от которого мы наследуемся.

**Слайд 8 (множественное наследование)**

Класс может наследоваться от нескольких классов. Это называется множественным наследованием. Когда используется множественное наследование, Python определяет порядок поиска методов и атрибутов в иерархии классов. Это называется **методом разрешения порядка (MRO, Method Resolution Order)**. Он определяет, в каком порядке будут проверяться классы.

**Слайд 9, 10 (зовём родителей, зовём родителей 2)**

Ключевое слово super() в Python используется для вызова методов родительского класса внутри дочернего класса. Это особенно полезно в контексте наследования, когда нужно расширить или переопределить функциональность родительского класса, но при этом все равно воспользоваться частью его логики.

**Слайд 11 (полиморфизм)**

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 12 (абстрактные методы)**

Абстракция позволяет создавать абстрактные классы и методы, которые должны быть реализованы в дочерних классах. В Python абстракция реализуется с помощью модуля abc.ABC — это такой квазикласс, при наследовании от которого появляется возможность создавать свои абстрактные методы, которые делают класс абстрактным. Как и обычно, если абстрактный метод в дочернем классе не реализован, то у нас вылетает ошибка.

**Слайд 13 (встроенные функции)**

Так как Python язык интерпретируемый и большую часть информации мы получаем из runtime`а, то мы прямо во время исполнения программы можем спросить, является ли объект экземпляром класса или наследуется ли какой-то класс от данного.

**Слайд 14 (пишем документацию)**

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.

**Слайд 15 (заключение)**

В заключение хочу сказать, что это только обзорная информация. Вот список ещё показавшихся мне интересными тем, которые связанны с классами в Python.