Python ООП Основи.

План

0. Источники.

1. Объектно-ориентированное программирование.

2. Преимущества ООП Python.

3. Недостатки ООП Python.

4. Обзор терминологии ООП.

5. Класс (Code).

6. Объекты(Code).

7. Атрибуты класса(Code)

8. Методы(Code).

9. Статические и классовые методы.

10. Статические методы @staticmethod.

11. Классовые методы @classmethod.

12. Жизненный цикл объекта (Code).

13. Конструкторы (Code).

14. Инкапсуляция (Code).

15. Наследование (Code).

16 Полиморфизм (Code).

0. Источники.

https://python-scripts.com/object-oriented-programming-in-python

https://proglib.io/p/python-oop/

https://python-scripts.com/object-oriented-programming-in-python

https://younglinux.info/oopython/course

https://pythonworld.ru/osnovy/obektno-orientirovannoe-programmirovanie-obshhee-predstavlenie.html

**1. Объектно-ориентированное программирование**

**Объектно-ориентированное программирование** (**ООП**) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

Oбъектно-ориентированное программирование — не зависящая от языка программирования концепция. Это общая концепцию программирования поддерживают большинство современных языков, такие как [Java](https://python-scripts.com/java-vs-python-simple-code-examples), C#, C++ и Python

**Объектно-ориентированный язык работает по следующим принципам:**

* Все данные представляются объектами.
* Программа является набором взаимодействующих объектов, посылающих друг другу сообщения.
* Каждый объект имеет собственную часть памяти и может иметь в составе другие объекты.
* Каждый объект имеет тип.
* Объекты одного типа могут принимать одни и те же сообщения (и выполнять одни и те же действия).

ООП позволяет сократить время на написание исходного кода, однако ООП всегда предполагает большую роль предварительного анализа предметной области, предварительного проектирования. От правильности решений на этом предварительном этапе зависит куда больше чем от непосредственного написания исходного кода.

## **2. Преимущества ООП Python**

* ООП подразумевает **повторное использование**. Компьютерная программа написанная в форме объектов и классов может быть использована снова в других проектах без повторения кода;
* Использование **модулярного подхода** в ООП позволяет получить читаемый и гибкий код;
* В ООП каждый [класс](https://python-scripts.com/data-class) имеет определенную задачу. Если ошибка возникнет в одной части кода, вы можете исправить ее локально, без необходимости вмешиваться в другие части кода;
* **Инкапсуляция данных** вносит дополнительный уровень безопасности в разрабатываемую программу с использованием объектно-ориентированного подхода.

**3. Недостатки ООП Python**

* Для создания объектов необходимо иметь подробное представление о разрабатываемом программном обеспечении;
* Не каждый аспект программного обеспечения является лучшим решением для реализации в качестве объекта. Для новичков может быть тяжело прочертить линию в золотой середине;
* С тем, как вы вносите все новые и новые классы в код, размер и **сложность программы растет** в геометрической прогрессии;

**4. Обзор терминологии ООП**

* **Класс** — определенный пользователем прототип для объекта, который определяет набор атрибутов, которые характеризуют любой объект класса. Атрибутами являются члены данных (переменные класса и переменные экземпляра) и методы, доступ к которым осуществляется через точечную запись.
* **Экземпляр** — индивидуальный объект определенного класса. Например, объект car, принадлежащий классу Car, является экземпляром класса Car.
* **Переменная (атрибут) класса** — переменная, которая используется всеми экземплярами класса. Переменные класса определены внутри класса, но вне любого из методов класса. Переменные класса используются не так часто, как переменные экземпляра.
* **Член данных** — переменная класса или переменная экземпляра, которая содержит данные, связанные с классом и его объектами.
* **Перегрузка функций** — назначение более чем одного поведения определенной функции. Выполняемая операция варьируется в зависимости от типов объектов или аргументов.
* **Переменная экземпляра** — переменная, которая определена внутри метода и принадлежит только текущему экземпляру класса.
* **Наследование** — передача характеристик класса другим классам, которые являются его производными.
* **Instantiation** — создание экземпляра класса.
* **Метод** — особый вид функции, который определен в определении класса.
* **Объект** — уникальный экземпляр структуры данных, который определяется его классом. Объект включает в себя как члены данных (переменные класса и переменные экземпляра), так и методы.
* **Перегрузка оператора** — назначение более чем одной функции определенному оператору.

## **5. Класс (Code)**

**Класс** — тип, описывающий устройство объектов.

Класс в ООП выступает в роли **чертежа для объекта**. Класс можно рассматривать как карту дома. Вы можете понять, как выглядит дом, просто взглянув на его карту.

Cам по себе класс не представляет ничего. К примеру, нельзя сказать что карта является домом, она только объясняет как настоящий дом должен выглядеть.

Нет такой вещи, как просто машина. Машина — это абстрактная концепция, которую реализуют в Toyota, Honda, Ferrari, и других компаниях.

Ключевое слово class используется для [создания класса](https://python-scripts.com/python-class) в Python.

## **6. Объекты(Code)**

**Kласс предоставляет чертеж объекта**. Однако, чтобы на самом деле использовать объекты и методы класса, вам нужно **создать объект из этого класса**.

**Объект** — это экземпляр класса.

Объект также называется **экземпляром**. Тем не менее, процесс создания объекта класса называется **инициализация**. В Python, чтобы создать объект класса, нам просто нужно вписать название класса, с последующими открывающимися и закрывающимися скобками.

## **7. Атрибуты класса(Code)**

Атрибуты могут быть наглядно отнесены к двум типам:

* атрибуты класса
* атрибуты экземпляров

**Атрибуты класса** делятся среди всех объектов класса, в то время как атрибуты экземпляров являются собственностью экземпляра.

**Атрибуты экземпляра** объявляются внутри любого метода, в то время как атрибуты класса объявляются вне любого метода.

Атрибуты класса также называются переменными. В зависимости от [области видимости](https://python-scripts.com/scope), переменные также могут относиться к двум типам:

* [локальные переменные](https://python-scripts.com/scope" \l "local-variables)
* [глобальные переменные](https://python-scripts.com/scope" \l "global-variables).

Локальная переменная в классе (Сode) — это переменная, доступ к которой возможен только внутри блока кода, в котором она определена. Например, если **определил переменную внутри метода**, к нему не удастся получить доступ откуда-либо вне метода.

Глобальная переменная (Сode) определяется вне любого блока, то есть метода, операторов-if, и тому подобное. Доступ к глобальной переменной может быть получен где угодно в классе.

Атрибуты экземпляра и класса отличаются способом получения доступа к ним. Другими словами, речь идет об использовании названия класса и использовании названия экземпляра. С другой стороны, **глобальные** и **локальные переменные** отличаются своими областями видимости, другими словами, местами, где к ним может быть получен доступ.

## **8. Методы(Code)**

В ООП методы используются для реализации функционалов объекта.

### **9. Статические и классовые методы**

Статические и классовые методы доступны без инстанцирования.

### **10. Статические методы** @staticmethod

У них нет обязательных параметров-ссылок вроде self. Доступ к таким методам можно получить как из экземпляра класса, так и из самого класса (Code).

Cтатические методы могут иметь доступ только к атрибутам класса в Python, вы не сможете обратиться к методам через self.

### **11. Классовые методы** @classmethod

Они аналогичны методам экземпляров, но выполняются не в контексте объекта, а в контексте самого класса (классы – это тоже объекты). Такие методы требуют обязательную ссылку на класс (cls).

### **12. Жизненный цикл объекта (Code)**

Метод \_\_new\_\_ -непосредственно создает новый экземпляр класса. Первым параметром он принимает ссылку на сам класс: Метод \_\_new\_\_ может быть очень полезен для решения ряда задач, например, создания иммутабельных объектов или реализации [паттерна Синглтон](https://ru.wikipedia.org/wiki/Одиночка_(шаблон_проектирования))

Mетод \_\_init\_\_ инициализация инстанса одного класса с заранее заданными параметрами с помощью инициализатора (специальный

Метод-деструктор \_\_del\_\_ для удаления инстанса. На практике деструктор используется редко, в основном для тех ресурсов, которые требуют явного освобождения памяти при удалении объекта. Не следует совершать в нем сложные вычисления.

## **13. Конструкторы (Code)**

Конструктор — это специальный метод, который вызывается по умолчанию когда вы **создаете объект класса (**экземпляра класса, Инстанц класса**)**.

Для создания конструктора вам нужно создать метод с ключевым словом \_\_init\_\_

За исключением названия, конструктор может использоваться как обычный метод. Вы можете передавать и получать значения из конструктора. Он обычно используется таким образом, когда вам нужно **инициализировать** значения атрибута при создании экземпляра класса.

## **Принципы ООП на Python**

## https://devpractice.ru/python-lesson-14-classes-and-objects/

## **14. Инкапсуляция (Code)**

Инкапсуляция — ограничение доступа к составляющим объект компонентам (методам и переменным). Инкапсуляция делает некоторые из компонент доступными только внутри класса.

Инкапсуляция в Python работает лишь на уровне соглашения между программистами о том, какие атрибуты являются общедоступными, а какие — внутренними.

Есть три типа модификаторов доступов в Python ООП:

1. публичный — **public** доступа открыт из любой точки вне класса;
2. приватный — **private** доступа открыт **только внутри класса**;
3. защищенный — **protected** доступ открыт только внутри того же пакета.

**15. Наследование (Code)**

Абстрактние класси

Множественное наследование

**16 Полиморфизм (Code)**