**Класс** — это абстрактный тип данных, определяемый программистом, и представляющий собой модель реального объекта в виде данных и функций для работы с ними. Класс – это **абстракция** (шаблон, матрица) некоторой сущности предметной области.

**Объект класса** — это конкретная переменная типа «класс».

**Наследование** – это один из трех основных принципов ООП, предполагающий создание иерархии классов (объектов классов), в которой объекты-потомки наследуют все члены (свойства и поведение) своих классов-предков.

**Полиморфизм** – (от греч. poli – много, morpheo – форма, поведение) – один из трех основных принципов ООП, заключающийся в возможности определения единого по имени действия (поведения), применимого ко всем объектам иерархии наследования (иерархии классов), причем каждый объект может реализовать это действие (поведение) собственным способом (в зависимости от того, к какому именно классу из данной иерархии этот объект относится).

**Абстракция** — это использование только тех характеристик объекта, которые с достаточной точностью представляют его в данной системе.

**Инкапсуляция** – это один из трех основных принципов ООП, предполагающий сокрытие деталей реализации класса от доступа извне данного класса (и, соответственно, извне данного объекта класса) и недопущение изменения внутреннего состояния объекта класса (значений, хранимых в его полях) без ведома самого данного объекта, а также недопущение изменений в состоянии объекта класса, которые приведут к его некорректной работе и/или уничтожению (не путать с деструктором, который является членом экземпляра класса и для того и существует, чтобы очищать оперативную память от данного объекта класса, уничтожая тем самым этот объект).

**Метод** – член класса, представляющий собой функцию, которая находится в данном классе (принадлежит ему и его объектам, может быть вызвана только у экземпляров данного класса). Метод, подобно функции, может принимать ни одного или несколько входных параметров и одно значение возвращать return'ом или ни одного (void). Метод видит содержимое поля своего экземпляра класса (даже закрытые от других экземпляров классов – private, поскольку метод сам находится в этом же (своем) экземпляре класса) и **может изменять** значения **не**константных полей своего класса.

**ООП** — это подход в программировании, при котором мы воспринимаем весь окружающий нас мир как совокупность конкретных объектов, поведение и свойства (характеристики) которых можно отделить от конкретных объектов и создать соответствующие абстрактные шаблонные образы – классы.

**Деструктор** – особый метод класса, который уничтожает объекты своего класса, когда они уже, по мнению программиста, не нужны. Деструктор очищает память от неиспользуемых объектов своего класса, аналогично коду delete. Если вызов конструктора «рождает» объект класса, то вызов для него деструктора – уничтожает этот объект, то есть означает смерть этого объекта.

В объектно-ориентированном программировании **конструктором** класса называют метод, который автоматически вызывается при создании объектов. Его также можно назвать конструктором объектов класса. В Python же роль конструктора играет метод **\_\_init\_\_().**

**class** Person:

**def** \_\_init\_\_(self, n, s):

self.name = n

self.surname = s

p1 = Person("Sam", "Baker")

**print**(p1.name, p1.surname)

\_\_del\_\_ () — это деструктор, метод, который используется для уничтожения экземпляра класса.

del obj

**Функция** – изолированный именованный блок кода, имеющий определенное назначение.

def compute\_surface(radius):

from math import pi

return pi \* radius \* radius

**Декоратор** – это функция, которая принимает другую функцию в качестве аргумента. **Декоратор** модифицирует или улучшает принятую функцию и выдает измененную.

def Decor\_AverageGrade(method\_to\_decor):

def wrapper(self, lie):

lie+=1

return method\_to\_decor(self, lie)

return wrapper

**переопределение методов класса-родителя:**

class Student(Person):

kols=0

def \_\_init\_\_(self, name, surname, age, avg):

super().\_\_init\_\_(name, surname, age)

self.avg = avg

Student.kols+=1

def study(self):

print(f"{self.name} {self.surname} is study")

**urllib** - это стандартная библиотека, которая поставляется с Python и может использоваться напрямую без установки. Предоставляет следующие функции: Запрос страницы, получение ответа, настройки прокси и файлов cookie, обработка исключений, разрешение URL.

**urlparse** - помощью модуля urlparse можно манипулировать URL-адресом. Например, разобрать его на составляющие или получить абсолютный URL-адрес, указав базовый адрес и относительный. URL-адрес состоит из следующих элементов:

**<Протокол>://<Домен>:<Порт>/<Путь>;<Параметры>?<Запрос>#<Якорь>**

Разобрать URL-адрес на составляющие позволяет функция urlparser():

**urlparser([, <Схема>[, <Якорь>]])**

**pyTelegramBotAPI –** подключаемая библиотека, позволяющая пользователю взаимодействовать с командами Telegram для создания телеграмм-ботов на python. Подключение в командной строке: **pip install pyTelegramBotAPI**. Для бота следует получить и подключить в программу класс TeleBot.

**discord.py** - это современная, простая в использовании, многофункциональная и асинхронная готовая API-оболочка для Discord. Установка: **py -3 -m pip install -U discord.py.** Для бота требуется учетная запись, токен.

**NumPy** - это фундаментальный пакет для научных вычислений на Python. Это библиотека Python, которая предоставляет объект многомерного массива, различные производные объекты (такие как замаскированные массивы и матрицы) и набор процедур для быстрых операций с массивами, включая математические, логические, манипуляции с формами, сортировку, выбор, ввод-вывод.

Установка: **python -m pip install --user numpy**

Класс окружность с полем радиус и методом для площади:

from math import sqrt

class Circle:

def \_\_init\_\_(self, R):

self.R = R

def get\_area(self):

return 3.14 \* (self.R^2)

R = Circle(4)

print(R.get\_area())

from abc import ABC, abstractmethod

class Basic(ABC):

@abstractmethod

def hello(self):

print("Hello from Basic class")

class Advanced(Basic):

def hello(self):

super().hello()

print("Enriched functionality")

a = Advanced()

a.hello()

Статические методы декларируются при помощи декоратора staticmethod. Им не нужен определённый первый аргумент (ни self, ни cls).

Их можно воспринимать как методы, которые “не знают, к какому классу относятся”.

Таким образом, статические методы прикреплены к классу лишь для удобства и не могут менять состояние ни класса, ни его экземпляра.

@staticmethod

def is\_adult(age):

return age > 18