

Основы Python

Классы и ООП

- ООП
 - синтаксис
 - инстанцирование
 - атрибуты
 - методы
-
- Инкапсуляция
 - Наследование
 - Полиморфизм

Про ООП

- любые программы это данные плюс алгоритмы
- ООП способ организации структуры программы, чтобы данные и функционал для обработки этих данных хранились в рамках единой сущности
- существует много терминологии вокруг ООП: абстракция, агрегация, инстанцирование, инкапсуляция, наследование, полиморфизм



- любы
 - ООП
 - чтоб д
 - данны
 - сущ
 - абстра
 - инкап
- ## Определение ООП
- Объектно-ориентированное программирование - это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.*
- 
- Гради Буч



```
class ClassName:  
    ''' docstring '''  
    <body of class>  
s = str()  
isinstance(s, str)  
type(str), type(ClassName)  
instance = ClassName()  
isinstance(instance, ClassName)
```

- Имя принято писать кемелкейсом.
- Описание класса это описание нового типа данных, т.е. этим мы создаем новый кастомный тип

```
class Rectangle:  
    a = 10  
    b = 20  
  
print(Rectangle.a, Rectangle.b)  
rectangle = Rectangle()  
print(rectangle.a, rectangle.b)  
rectangle.c = 30  
print(Rectangle.c)
```

- Создает пространство имен, к которому можно обратиться через точку как из самого объекта класса, так и через инстанс
- У инстанса свое независимое пространство имен, но он имеет доступ к пространству класса

```
class Rectangle:  
    a, b = 10, 20  
    c = [1]
```

```
rectangle = Rectangle()  
rectangle.a = 30  
print(rectangle.a, Rectangle.a)  
rectangle.c.append(2)  
print(rectangle.c, Rectangle.c)
```

- Чем-то напоминает ситуацию с передачей в функцию изменяемых объектов.
- При присвоении значения изменяемому объекту мы создаем локальную переменную, с изменяемым значением мы правим исходный объект

Инстанцирование

```
class Rectangle:  
    def __init__(self):  
        self.a, self.b = 10, 20  
  
rectangle = Rectangle()  
print(rectangle.a, rectangle.b)
```

- При инстанцировании каждый раз вызывается специальный метод `__init__`
- Все функции первым параметром получают `self`, инстанс для которого эта функция вызывается

```
class Rectangle:  
    def __init__(self, a, b):  
        self.a, self.b = a, b  
  
rectangle = Rectangle(10, 20)  
print(rectangle.a, rectangle.b)
```

- Если при инстанцировании класс вызвать с параметрами, то именно метод `__init__` получит эти параметры

ФУНКЦИИ КЛАССА

```
class Rectangle:  
    def __init__(self, a, b):  
        self.a, self.b = a, b  
    def square(self):  
        return self.a * self.b  
  
rectangle = Rectangle(10, 20)  
print(rectangle.square())
```

- В теле класса можно описывать функции
- Все функции первым параметром получают `self`, через данную переменную в функции можно обращаться к атрибутам инстанса и функциям класса

```
class Rectangle:  
    def __init__(self, a, b):  
        self.a, self.b = a, b  
    def square(self):  
        return self.a * self.b  
    def volume(self, c):  
        return self.square() * c
```

```
rectangle = Rectangle(10, 20)  
print(rectangle.volume(5))
```

- Функции ведут себя как обычные за исключением первого параметра, т.е. можно передавать другие параметры, возвращать значения и т.д.

Решение квадратного уравнения

```
class QuadraticEquation:  
    def __init__(self, a, b, c):  
        self.a, self.b, self.c = a, b, c  
    def get_discr(self):  
        d = self.b ** 2 - 4 * self.a * self.c  
        return d  
    ...  
equation = QuadraticEquation(1, 2, 1)  
equation.print_eq_roots()
```

- Описываем атрибутами класса переменные уравнения
- Упаковываем функции внутри класса

Служебные функции класса

```
class Rectangle:  
    def __init__(self, a, b):  
        self.a, self.b = a, b  
    def __len__(self):  
        return 2 * (self.a + self.b)  
  
rectangle = Rectangle(10, 20)  
print(len(rectangle))
```

- имена всяких служебные методов и атрибуты обрамляются двойными подчеркиваниями

```
class Rectangle:  
    def __init__(self, a, b):  
        self.a, self.b = a, b  
        self._c = None  
    def __len__(self):  
        self._c = 2 * (self.a + self.b)  
        return self._c  
  
rectangle = Rectangle(10, 20)  
print(len(rectangle), rectangle._c)
```

- реализовано на уровне соглашения: имена функций и атрибутов начинаются с “_” приватные и обращаться можно к ним только из тела класса



```
class Rectangle:  
    def __init__(self, a, b):  
        self.a, self.b = a, b  
        self.__c = None  
    def set_color(self, c):  
        self.__c = c  
    def get_color(self):  
        return self.__c  
  
rectangle = Rectangle(10, 20)  
rectangle.set_color("#15FF15")  
print(rectangle.get_color(), rectangle._Rectangle__c)
```

- сильно приватные имена функций и атрибутов начинаются с “__”, доступ к ним тоже можно получить, но лучше не надо


```
class Rectangle:  
    def __init__(self, a, b):  
        self.a, self.b = a, b  
    def square(self):  
        return self.a * self.b  
  
class Square(Rectangle):  
    def __init__(self, a):  
        self.a = self.b = a  
  
sq = Square(10)  
print(sq.square())
```

- Указав при определении класса родителя, текущий класс наследует от родителя все атрибуты и функции

Полиморфизм

```
class Rectangle:  
    def __init__(self, a, b):  
        self.a, self.b = a, b  
    def perimeter(self):  
        return 2 * (self.a + self.b)  
  
class Square(Rectangle):  
    def __init__(self, a):  
        self.a = self.b = a  
    def perimeter(self):  
        return self.a * 4  
  
sq = Square(10)  
print(sq.perimeter())
```

- В наследнике можно переопределить функцию с тем же именем в итоге будет вызываться разная функция с одинаковым именем для инстасов разных классов

```
class Rectangle:  
    def __init__(self, a, b):  
        self.a, self.b = a, b  
  
class Square(Rectangle):  
    def __init__(self, a):  
        super().__init__(a, a)  
  
sq = Square(10)  
print(sq.a, sq.b)
```

- С помощью функции `super()` можно обратиться к функциям родительского класса.