

ООП Python

общие свойства объектов в языке программирования

Class

- данные(свойства)

```
prop = 'value'
```

- методы(действия)

```
def get_prop(self, x, y):  
    self.x = x  
    self.y = y
```

Добавить св-во

- Class.prop = 'value'
- setattr(Class, 'prop', 'value')

Читать св-во

- c = Class.prop - если нет св-ва, возвращает ошибку
- getattr(Class, prop, False) - если нет св-ва, возвращает третий аргумент

Наличие св-ва

- hasattr(Class, prop) - учитывает наследование

Удалить св-во

- del Class.prop - если нет св-ва, возвращает ошибку
- delattr(Class, prop) - если нет св-ва, возвращает ошибку

Пространство имен

- Class.__dict__

Все свойства экз.класса

- dir(obj)

Описание класса, комментарий

- Class.__doc__

Метод класса

@classmethod - работает с атрибутами класса

```
def method_a(cls, arg):  
    return cls.x
```

Статический метод

@staticmethod - независимая, сервисная функция; нет дополнительных параметров, кроме тех, которые указываем; не имеют доступа к атрибутам класса и к атрибутам экз.класса; можем вызывать внутри обычных методов

```
def method_b(x, y):  
    return x*x + y*y
```

Режим доступа (механизм инкапсуляции)

- attr - публичное свойство
- _attr - режим доступа protected
- __attr - режим доступа private

pip install accessify - защитить методы от внешнего доступа

Магические методы - методы, которые вызываются интерпретатором для выполнения различных операций над объектами

Метод	Что делает
<code>__new__(cls) return super().__new__(cls)</code>	Создание объекта cls ссылается на текущий класс должен возвращать адрес нового созданного объекта
<code>__init__(self)</code>	Инициализатор объекта
<code>__del__(self)</code>	Финализатор объекта
<code>__setattr__(self, key, value)</code>	Автоматически вызывается при изменении свойства key класса
<code>__getattr__(self, item)</code>	Автоматически вызывается при получении свойства класса с именем item
<code>__getattr__(self, item)</code>	Автоматически вызывается при получении несуществующего свойства item класса
<code>__delattr__(self, item)</code>	Автоматически вызывается при удалении свойства item (не важно существует оно или нет)
<code>__call__(self, *args, **kwargs)</code>	Позволяет экземплярам класса вести себя так, как будто они функции
<code>__str__(self)</code>	Для отображения информации об объекте класса для пользователей
<code>__repr__(self)</code>	Для отображения информации об объекте класса в режиме отладки
<code>__len__(self)</code>	Позволяет применить функцию len() к экземпляру класса
<code>__abs__(self)</code>	Позволяет применить функцию abs() к

	экземпляру класса
<code>__add__(self)</code>	Для операции сложения
<code>__sub__(self)</code>	Для операции вычитания
<code>__mul__(self)</code>	Для операции умножения
<code>__truediv__(self)</code>	Для операции деления
<code>__eq__(self)</code>	Для равенства ==
<code>__ne__(self)</code>	Для неравенства !=
<code>__lt__(self)</code>	Для оператора <
<code>__le__(self)</code>	Для оператора <=
<code>__gt__(self)</code>	Для оператора >
<code>__ge__(self)</code>	Для оператора >=
<code>__hash__(self)</code>	Вычисляется хэш экземпляра класса
<code>__bool__(self)</code>	Используется в программах, где требуется описать собственные проверки истинности или ложности объектов
<code>__getitem__(self, item)</code>	Получение значения по ключу item
<code>__setitem__(self, key, value)</code>	Запись значения value по ключу key
<code>__delitem__(self, key)</code>	Удаление элемента по ключу key
<code>__iter__(self)</code>	Получение итератора для перебора объекта
<code>__next__(self)</code>	Переход к следующему значению и его считывание
<p>объект-свойство property 1. <code>prop = property(getter, setter)</code> 2. <code>@property</code> <code>def name(self):</code> <code>return self.__name</code> <code>@name.setter</code> <code>def</code> <code>name(self, name):</code> <code>self.__name = name</code> <code>@name.deleter</code> <code>def</code> <code>name(self):</code> <code>del self.__name</code></p>	Для работы с приватными локальными свойствами экземпляров классов
коллекция <code>__slots__</code> <code>__slots__ = ('x', 'y')</code>	Ограничивает допустимый набор имен атрибутов объекта только перечисленными именами - ограничение создаваемых локальных свойств - уменьшение занимаемой памяти, атрибут <code>__dict__</code> удаляется - ускорение работы с локальными свойствами

Дескриптор:

<code>def __get__(self, instance, owner) :</code> <code>return...</code>	Дескриптор не данных только считывают данные имеют приоритет, как и атрибуты класса
<code>def __get__(self, instance, owner) :</code> <code>return...</code> <code>def __set__(self, instance,</code> <code>value) : ...</code> <code>def __del__(self) : ...</code>	Дескриптор данных - это атрибут объекта со связанным поведением, то есть такой атрибут, при доступе к которому его поведение переопределяется методом протокола дескриптора

```

class Integer:
    @classmethod
    def verify_coord(cls, coord):
        if type(coord) != int:
            raise TypeError('Координата должна быть целым числом')

    def __set_name__(self, owner, name):
        self.name = '_' + name

    def __get__(self, owner, instance):
        # return instance.__dict__[self.name]
        return getattr(instance, self.name)

    def __set__(self, instance, value):
        self.verify_coord(value)
        # instance.__dict__[self.name] = value
        return setattr(instance, self.name, value)

class Point3D:
    x = Integer()
    y = Integer()
    z = Integer()

    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

p = Point3D(1, 2, 3)
print(p.__dict__)

```