Занятие №4

I T E A

IT Education Academy

WWW.ITEA.UA

I T E A

www.itea.ua

Поляков Антон

Фото инструктора

- Python backend developer

КОНТАКТНЫЕ ДАННЫЕ

Telegram: @polyakov1

Метаклассы

Метаклассы — это классы которые управляют созданием других классов.
Создать класс без литерала class, можно следующим образом:

```
my_class = type(«MyClass», (), {«var»:True})
```

1 — Название класса
2 — Классы от которых наследуемся
3 — Аттрибуты класса

Метаклассы

```
class MyMetaClass(type):

    def __new__(cls, name, base, attrs):
        print(cls, name, base, attrs)

        return super().__new__(cls, name, base, attrs)



class ClassName(metaclass=MyMetaClass):
    """docstring for ClassName"""
    def __init__(self, arg):
        self.arg = arg
```

Абстрактные классы

```
from abc import ABC, abstractmethod

class ChessPiece(ABC):
    # общий метод, который будут использовать все наследники этого класса
    def draw(self):
        print("Drew a chess piece")

    # абстрактный метод, который будет необходимо переопределять для каждого подкласса
    @abstractmethod
    def move(self):
        Pass
```

Таким образом, используя концепцию абстрактных классов, мы можем улучшить качество архитектуры приложения, уменьшить объем работы и при этом, обеспечить легкость дальнейшей поддержки кода.

Property

```
class C(object):
    def __init__(self):
        self._x = None

    @property
    def x(self):
        """I'm the 'x' property."""
        return self._x

    @x.setter
    def x(self, value):
        self._x = value

    @x.deleter
    def x(self):
        del self._x
```

Как это работает с property()

```
class C(object):
    def __init__(self):
        self._x = None

    def _x_get(self):
        return self._x

    def _x_set(self, value):
        self._x = value

    def _x_del(self):
        del self._x

    x = property(_x_get, _x_set, _x_del,
                 "I'm the 'x' property.")
```

Classmethod staticmethod

```
class A(object):
    def foo(self, x):
        print "executing foo(%s, %s)" % (self, x)

    @classmethod
    def class_foo(cls, x):
        print "executing class_foo(%s, %s)" % (cls, x)

    @staticmethod
    def static_foo(x):
        print "executing static_foo(%s)" % x
```

Метод __call__

Метод __call__ вызывается при обращении к экземпляру как к функции. Это не повторяющееся определение — если метод __call__ присутствует, интерпретатор будет вызвать его, когда экземпляр вызывается как функция, передавая ему любые позиционные и именованные аргументы:

```
class C:
    def __call__(self, a, b, x=50, y=60):
```

```
class C:
    def __call__(self, *args, **kwargs): # Произвольные аргументы
        pass
```

```
class C:
    def __call__(self, *args, x = 100, **kwargs): # Аргументы которые могут
        pass                                      # передаваться только по имени
```

__call__

```
class dec1(object):
    def __init__(self, f):
        self.f = f
    def __call__(self):
        print "Decorating", self.f.__name__
        self.f()

@dec1
def func1():
    print "inside func1()"
```

List comprehension

squares = [x ** 2 for x in range(10)] — compr
Аналог:
squares = []
for x in range(10):
  squares.append(x ** 2)

odds = [x for x in range(10) if x % 2 != 0] — условие
[x ** 2 if x % 2 == 0 else x ** 3 for x in range(10)] — if + else

{key: value for key, value in zip([1, 2, 3], ['a', 'b', 'c'])} — compr в словарях

Задача

Создайте класс ПЕРСОНА с абстрактными методами, позволяющими вывести на экран информацию о персоне, а также определить ее возраст (в текущем году). Создайте дочерние классы: АБИТУРИЕНТ (фамилия, дата рождения, факультет), СТУДЕНТ (фамилия, дата рождения, факультет, курс), ПРЕПОДАВАТЕЛЬ (фамилия, дата рождения, факультет, должность, стаж), со своими методами вывода информации на экран и определения возраста. Создайте список из n персон, выведите полную информацию из базы на экран, а также организуйте поиск персон, чей возраст попадает в заданный диапазон.

Задача

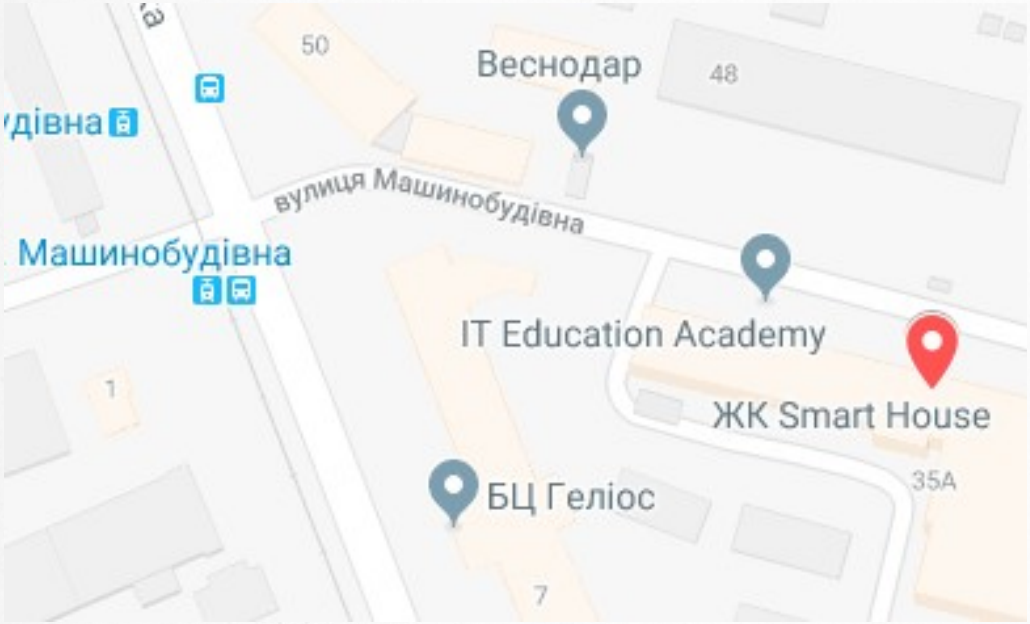
Создать подобие социальной сети. Описать классы, которые должны выполнять соответствующие функции (Предлагаю насследовать класс авторизации от класса регистрации). Добавить проверку на валидность пароля (содержание символов и цифр), проверка на уникальность логина пользователя. Человек заходит, и имеет возможность зарегистрироваться (ввод логин, пароль, потдверждение пароля), далее входит в свою учетную запись. Добавить возможность выхода из учетной записи, и вход в новый аккаунт. Создать класс User, котоырй должен разделять роли обычного пользователя и администратора. При входе под обычным пользователем мы можем добавить новый пост, с определённым содержимим, так же пост должен содержать дату публикации. Под учётной записью администратора мы можем увидеть всех пользователей нашей системы, дату их регистрации, и их посты.

Реализация паттерна Singletone

```
class Singletone:

    _instance = None  # Keep instance reference

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super().__new__(cls)
        return cls._instance
```



КОНТАКТНЫЕ ДАННЫЕ

ITEA

ул. Машиностроительная, 41, ЖК «Smart House», Киев

ул. Срибнокильская, 1, офис 269, Киев

пр. Академика Глушкова, 1, корп.17, Киев

+38 (044) 599-01-79
facebook.com/Itea
info@itea.ua
itea.ua