**Задание 1**

Создайте родительский класс Animal, у которого есть 3 атрибута:

* color - цвет
* name - кличка
* age - возраст

и абстрактный метод:

* say - издать звук.

Создайте два класса потомка - Cat и Dog, в которых будет переопределен метод say: для класса Cat - Meow!, для Dog - Woof!

from abc import ABC, abstractmethod

class Animal(ABC):

def \_\_init\_\_(self, color, name, age):

self.color = color

self.name = name

self.age = age

@abstractmethod

def say(self):

pass

class Cat(Animal):

def say(self):

return "мяу!"

class Dog(Animal):

def say(self):

return "гаф!"

cat = Cat('white', Маня, 2)

dog = Dog('black', Арчи, 15)

print(cat.say())

print(dog.say())

**Задание 2**

Создайте базовый класса Father, у которого есть два атрибута: name (имя) и surname (фамилия); и дочерний класс Child, у которого будут унаследованы те же атрибуты и дополнительно атрибут patronim (отчество). Создайте один экземпляр класса Child.

class Father:

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

self.name = name

self.surname = surname

class Child(Father):

def \_\_init\_\_(self, name, surname, patronim):

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

self.patronim = patronim

child = Child("Арсений", "Ротов", "Владимирович")

**Задание 3**

А) Реализовать класс Stationery (канцелярия):

* определить в нём атрибут title (название) и абстрактный метод draw (отрисовка);
* создать три дочерних класса Pen (ручка), Pencil (карандаш), Handle (маркер);
* в классу Pen добавьте атрибут color = 'blue';
* в каждом классе реализовать переопределение метода draw. Для каждого класса метод должен выводить уникальное сообщение, например: "Ручка пишет";
* создать экземпляры классов и проверить, что выведет описанный метод для каждого экземпляра.

А)

from abc import ABC, abstractmethod

class Stationery(ABC):

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

self.title = title

@abstractmethod

def draw(self):

pass

class Pen(Stationery):

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

super().\_\_init\_\_(title)

self.color = 'blue'

def draw(self):

print(f"Ручка {self.title} пишет {self.color}")

class Pencil(Stationery):

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

super().\_\_init\_\_(title)

def draw(self):

print(f"Карандаш {self.title} рисует")

class Handle(Stationery):

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

super().\_\_init\_\_(title)

def draw(self):

print(f"Маркер {self.title} выделяет")

pen = Pen('Brauberg')

pen.draw()

pencil = Pencil('Faber-Castel')

pencil.draw()

handle = Handle('Twin Tip')

handle.draw()

Б)

* Добавьте в класс Stationary метод класса set\_color, который присваивает атрибут класса Stationary color;
* Вызовете метод set\_color и установите color='yellow';
* Вызовете атрибуты color у классов Pen, Pencil, Handle. Что вы наблюдаете?

Б)

from abc import ABC, abstractmethod

class Stationery(ABC):

color = None;

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

self.title = title

@abstractmethod

def draw(self):

pass

@classmethod

def set\_color(cls, color):

cls.color = color

Stationery.set\_color('yellow')

class Pen(Stationery):

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

super().\_\_init\_\_(title)

def draw(self):

print(f"Ручка {self.title} пишет {self.color} ")

class Pencil(Stationery):

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

super().\_\_init\_\_(title)

def draw(self):

print(f"Карандаш {self.title} рисует {self.color}")

class Handle(Stationery):

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

super().\_\_init\_\_(title)

def draw(self):

print(f"Маркер {self.title } выделяет {self.color}")

pen = Pen('Brauberg')

pen.draw()

pencil = Pencil('Faber-Castel')

pencil.draw()

handle = Handle('Twin Tip')

handle.draw()

print(pen.color)

print(pencil.color)

print(handle.color)

**Задание 4**

Создайте базовый класс User с атрибутами login, password и методом:

* view - выводит сообщение "Я - User. Могу просматривать содержимое"

Создайте дочерний класс Moderator, у которого так же будут атрибуты login и password и дополнительно - group\_id, а так же два метода:

* view - выводит сообщение "Я - Moderator. Могу просматривать содержимое"
* redact - выводит сообщение "Я - Moderator. Могу редактировать содержимое"

class User:

def \_\_init\_\_(self, login, password):

self.login = login

self.password = password

def view(self):

print("Я - User. Могу просматривать содержимое")

class Moderator(User):

def \_\_init\_\_(self, login, password, group\_id):

super().\_\_init\_\_(login, password)

self.group\_id = group\_id

def view(self):

print("Я - Moderator. Могу просматривать содержимое")

def redact(self):

print("Я - Moderator. Могу редактировать содержимое")

user = User("user\_login", "user\_password")

user.view()

moderator=Moderator("moderator\_login","moderator\_password","moderator\_group\_id")

moderator.view()

moderator.redact()

**Задание 5**

Создайте класс Clock со статичным методом Say\_time, который будет выводить на экран текущее время.

Подсказка: для этого можете воспользоваться стандартной библиотекой time.

import time

class Clock:

@staticmethod

def Say\_time():

print("Текущее время:", time.strftime("%H:%M:%S"))

Clock.Say\_time()

**Задание 6 \* (необязательное)**

Давайте создадим свой класс Time по аналогии с Date\_time выше. Реализуйте в нем классовый метод from\_string(), который будет принимать в себя строку('00:00:00') и будет преобразовывать его в объект Time.

class Time:

def \_\_init\_\_(self, hour, minute, second):

self.hour = hour

self.minute = minute

self.second = second

@classmethod

def from\_string(cls, time\_string):

hour, minute, second = map(int, time\_string.split(':'))

return cls(hour, minute, second)