Python

11

Объектно-ориентированное программирование(ООП)

1. Объектно-ориентированное программирование подразумевает повторное использование. Компьютерная программа написанная в форме объектов и классов может быть использована снова в других проектах без повторения кода;
2. Использование модульного подхода в объектно-ориентированном программировании позволяет получить читаемый и гибкий код;
3. В объектно-ориентированном программировании каждый класс имеет определенную задачу. Если ошибка возникнет в одной части кода, вы можете исправить ее локально, без необходимости вмешиваться в другие части кода;
4. ООП вносит дополнительный уровень безопасности в разрабатываемую программу;

Класс

```
class ИмяКласса:
    атрибуты_класса
    методы_класса
```

Класс в ООП выступает в качестве чертежа для объекта. В классе определены атрибуты класса и методы. Атрибуты описывают состояние. Методы описывают поведение.

Задание 11.01

Создать пустой класс Dog

Создание объекта

```
class Dog:
    pass
```

Dog - имя класса

```
dog = Dog()
```

dog - Объект(экземпляр) класса Dog

Задание 11.02

Создать два объекта класса Dog. Вывести их на экран

Методы класса

```
class Dog:
    def bark(self):
        print('Woof Woof!')


dog_1 = Dog()
dog_1.bark()
```

self - обязательный атрибут метода. Позволяет обращаться к атрибутам и методам объекта

Обращение к методам происходит через точку

Задание 11.03

Добавить два метода в класс Dog: jump и run. Методы выводят на экран Jump! и Run! соответственно.

Конструктор

```
class Dog:
    def __init__(self, name):
        self.name = name


dog_1 = Dog('Bob')
print(dog_1.name)
```

Конструктор - метод, который вызывается, когда создается объект.

Конструктор инициализирует атрибуты объекта

Задание 11.04

Создать класс Dog. Класс имеет четыре атрибута: height, weight, name, age. Класс имеет три метода: jump, run, bark. Каждый метод выводит сообщение на экран. Создать объект класса Dog, вызвать все методы объекта и вывести на экран все его атрибуты.

Изменение атрибутов

```
class Dog:
    ….
    def change_height(self, height):
        self.height = height
    ….
dog_1 = Dog('Bob', 11, 12, 2015)
print(dog_1.height)
dog_1.change_height(15)
print(dog_1.height)
```

Задание 11.05

Добавить в класс Dog метод change_name. Метод принимает на вход новое имя и меняет атрибут имени у объекта. Создать один объект класса. Вывести имя. Вызвать метод change_name. Вывести имя.

Модификаторы доступа

```
class Dog:
    def __init__(self, name, age, weight):
        self.__name = name  # private
        self._age = age  # protected
        self.weight = weight  # public


dog = Dog('Bob', 8, 2.4)
print(dog.__name)  # ERROR
print(dog._Dog__name)
print(dog._age)
print(dog.weight)
```

Задание 11.06

Добавить в метод инициализации новый приватный атрибут - master. Создать метод get_master() который возвращает значение атрибута master.

getter, setter

```
class Dog:
    ….
    def __init__(self, master):
        self.__master = master
    def get_master(self):
        return self.__master
    def set_master(self, master):
        self.__master = master
    ….
dog = Dog('Alex')
print(dog.get_master())
dog.set_master('Pavel')
print(dog.get_master())
```

Задание 11.07

Добавить новый приватный атрибут адрес(по-умолчанию равен ‘Minsk’). Добавить getter и setter для адреса.

getter, setter через декораторы

```
class Dog:
    def __init__(self, master):
        self.__master = master
    @property
    def master(self):
        return self.__master
    @master.setter
    def master(self, master):
        if len(master) < 5:
            self.__master = master
dog = Dog('Alex')
dog.master = 'Moe'
print(dog.master)
```

Декоратор @property позволяет обращаться к методам без использования скобок

Декоратор @имя_метода.setter позволяет передавать аргумент в функцию через знак присваивания =

Задание 11.08

Сделать все атрибуты класса Dog приватными. Сделать для каждого атрибута getter и setter используя декораторы. Все change методы удалить

Класс как тип данных

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Square:
    def __init__(self, point_a, point_b):
        self.point_a = point_a
        self.point_b = point_b
```

Задание 11.09

Создать три класса: Dog, Cat, Parrot. Атрибуты каждого класса: name, age, master. Каждый класс содержит конструктор и методы: run, jump, birthday(увеличивает age на 1), sleep. Класс Parrot имеет дополнительный метод fly. Cat - meow, Dog - bark.

Наследование

```
class Parent:
    def print_world(self):
        print('world')

class Child(Parent):
    def print_hello(self):
        print('hello')
```

Дочерний класс наследуют все методы родитльского класса.

Задание 11.10

Создать родительский класс Pet, содержащий все общие методы классов Dog, Cat, Parrot. Унаследовать Dog, Cat, Parrot от класса Pet. Удалить в дочерних классах те методы, которые имеются у родительского класса. Создать объект каждого класса и вызвать все его методы.

Перегрузка методов

```
class Dog:
    def set_name(self, name=None):
        if name:
            self.name = name
        else:
            print('Name was not changed')
```

Перегрузка метода позволяет менять логику выполнения метода в зависимости от типа и количества переданных аргументов.

Задание 11.11

Добавить два новых атрибута в родительский класс: weight и height.

Добавить методы change_weight, change_height принимающий один параметр и прибавляющий его к соответствующему аргументу. В случае если параметр не был передан, увеличивать на 0.2.

Изменить метод fly класса Parrot. Если вес больше 0.1 выводить сообщение This parrot cannot fly.