ЛАБОРАТОРНАЯ РАБОТА № 4
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА PYTHON

Время выполнения – 4 часа.

Цель работы: познакомиться с принципами объектно-ориентированного программирования на языке программирования python.

Задачи работы

1. Изучить основные конструкции, используемые при построении классов и объектов;
2. Приобрести навыки практического применения ООП.

Перечень обеспечивающих средств

Для выполнения работы необходимо иметь компьютер с установленной операционной системой семейства Windows, установленным python и IDE PyCharm Professional.

Общие теоретические сведения

Объектно-ориентированное программирование или коротко ООП есть парадигма программирования, в которой программы строятся таким образом, что свойства и поведение объединяются в отдельные единые объекты.

Например, объектом может быть человек с такими свойствами, как имя, возраст, адрес и т.д., обладающий таким поведением, как ходьба, разговор, дыхание и бег. Или электронное письмо с такими свойствами, как список получателей, тема, тело и т.д. и таким поведением, как добавление вложений и отправка.

Иными словами, объектно-ориентированное программирование – это подход для моделирования конкретных реальных сущностей, например, автомобиль, а также отношений между такими сущностями, как компании и сотрудники, студенты и преподаватели и т.п. ООП моделируе т реальные объекты программными единицами, которые имеют некоторые данные связаны с ними и могут выполнять определенные действия.

Ключевым выводом является то, что в фокусе парадигмы объектно-ориентированного программирования находятся объекты, в которых данные и методы обработки этих данных объединяются в единое целое и создают общую структуру программы, в отличии от рецептов процедурного программирования, которые являются отдельными функциями или блоками кода.

Поскольку Python является языком программирования с множеством парадигм, вы можете выбрать ту парадигму, которая лучше всего подходит для решения задачи, смешать разные парадигмы в одной программе и/или переключаться с одной парадигмы на другую по мере развития своего решения.

Классы в Python

Первоначально ориентируясь на данные, каждая вещь или объект является экземпляром некоторого класса.

Примитивные структуры данных, доступные в Python, такие как числа, строки и списки, предназначены для представления простых вещей, таких как стоимость чего-либо, название стихотворения и ваши любимые цвета соответственно.

Что если вы хотите представить что-то гораздо более сложное?

Допустим, вы хотите наблюдать за разными животными. Если вы используете список, первый элемент может быть его именем, а второй элемент может представлять возраст.

Как бы вы узнали, какой элемент должен быть? Что делать, если у вас было 100 разных животных? Вы уверены, что у каждого животного есть и имя,

и возраст, и так далее? Что если вы захотите добавить другие свойства этим животным? Это уже требует некоторой организации, и это именно то, что нужно для классов.

Классы используются для создания новых пользовательских структур данных, которые содержат произвольную информацию о чем-либо. В случае с животным мы могли бы создать класс *Animal()* для описания таких свойства, как имя и возраст.

Важно отметить, что класс просто обеспечивает структуру — это образец того, как что-то должно быть определено, но на самом деле это просто шаблон, за которым нет реального контента. Класс *Animal()* может указывать, что имя и возраст необходимы для определения животного, но на самом деле он не содержит ни имени, ни возраста конкретного животного. Это просто описание, классификация.

Класс — это идея, наше представление о характеристиках какой-либо сущности, о том, как эта сущность должна быть определена.

Объекты Python (экземпляры класса)

В то время как класс является шаблоном, экземпляр является реализацией класса с фактическими значениями, буквально объектом, принадлежащим определенному классу. Это больше не идея; это реальное животное, как собака по имени Роджер, которой восемь лет.

Иными словами, класс — это форма или шаблон. Он определяет необходимую информацию. После того, как вы заполните форму, ваша конкретная копия становится экземпляром класса; он содержит актуальную информацию, относящуюся к вам.

Вы можете заполнить несколько копий, чтобы создать много разных экземпляров, но без формы в качестве руководства вы потерялись бы, не зная, какая информация требуется. Таким образом, прежде чем вы сможете создавать отдельные экземпляры объекта, мы должны сначала указать, что нужно, определив класс.

Как описать класс в Python

Вот простое описание класса в Python:

```
class Dog:
    pass
```

Вы начинаете с ключевого слова *class*, которым указываете, что создаете класс, а затем добавляете имя класса *Dog,* начинающееся с заглавной буквы.

Также мы использовали здесь ключевое слово Python *pass*, которое очень часто используется в качестве заполнителя, заглушки, где в конечном итоге код будет изменён и продолжен. Такая запись позволяет запускать этот код без выдачи сообщения об ошибке.

Атрибуты экземпляра

Все классы создают объекты и все объекты содержат характеристики, называемые атрибутами. Используйте метод *__init__()* инициализации начальных значений атрибутов объекта по умолчанию. Этот метод должен иметь как минимум один аргумент, а также переменную self, которая ссылается на сам объект (например, Dog).

```
class Dog:
    # Атрибуты Инициализатора / Экземпляра
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

В случае нашего класса *Dog()* каждая собака имеет определенное имя и возраст, что, безусловно, важно знать, когда вы начинаете создавать разных собак. Помните: класс предназначен только для определения собаки, а не для создания экземпляров отдельных собак с конкретными именами и возрастами.

Точно так же переменная *self* является экземпляром класса. Поскольку экземпляры класса имеют различные значения, мы можем указать Dog.name = name, а не self.name = name. Но поскольку не все собаки имеют одно и то же имя, мы должны иметь возможность назначать разные значения для разных экземпляров. Отсюда необходимость специальной переменной *self*, которая поможет отслеживать отдельные экземпляры каждого класса.

Атрибуты класса

Хотя атрибуты экземпляра являются специфическими для каждого объекта, атрибуты класса одинаковы для всех экземпляров — в данном случае это все собаки.

```
class Dog:
    # Атрибуты класса
    species = 'mammal'
    # Атрибуты Инициализатора / Экземпляра
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Таким образом, хотя каждая собака имеет уникальное имя и возраст, каждая собака будет млекопитающим.

Создание объектов

Instantiating – необычный термин для создания нового уникального экземпляра класса.

```
class Dog:
    # Атрибуты класса
    species = 'mammal'
    # Атрибуты инициализатора/экземпляра
    def __init__(self, name, age):
        self.name = name
        self.age = age
# Instantiate the Dog object
philo = Dog("Philo", 5)
mikey = Dog("Mikey", 6)
# Access the instance attributes
print("{} is {} and {} is {}.".format(
    philo.name, philo.age, mikey.name, mikey.age))
# Is Philo a mammal?
if philo.species == "mammal":
    print("{0} is a {1}!".format(philo.name, philo.species))
```

В примере выше мы создали новый экземпляр класса *Dog()* и присвоили его переменной *philo*. Затем мы передали ему два аргумента: "Philo" и 5, которые представляют имя и возраст этой собаки соответственно.

Эти атрибуты передаются методу *__init__*, который вызывается каждый раз, когда вы создаете новый экземпляр, прикрепляя имя и возраст к объекту. Вам может быть интересно, почему нам не пришлось передавать аргумент self.

Это магия Питона; когда вы создаете новый экземпляр класса, Python автоматически определяет, что такое *self* (в данном случае это Dog) и передает его методу *__init__*.

Методы экземпляра

Методы экземпляра определены внутри класса и используются для получения содержимого экземпляра. Они также могут быть использованы для выполнения операций с атрибутами наших объектов. Как и метод *__init__*, где первым аргументом всегда является *self*:

```
class Dog:
    # Атрибуты класса
    species = 'mammal'
    # Атрибуты Инициализатора / Экземпляра
    def __init__(self, name, age):
        self.name = name
        self.age = age
    # Метод класса
    def description(self):
        return "{} is {} years old".format(self.name, self.age)
    # Метод класса
    def speak(self, sound):
        return "{} says {}".format(self.name, sound)
# Создаем объект (экземпляр класса) Dog
mikey = Dog("Mikey", 6)
# Вызов методов класса
print(mikey.description())
print(mikey.speak("Gruff Gruff"))
```

В последнем методе, *speak(),* мы определяем поведение. Какие другие виды поведения вы можете назначить собаке? Вернитесь к началу абзаца, чтобы увидеть примеры поведения других объектов

Задание

1. Разработать программный модуль «Учет успеваемости студентов». Программный модуль предназначен для оперативного учета успеваемости студентов в сессию деканом, заместителями декана и сотрудниками деканата. Сведения об успеваемости студентов должны храниться в течение всего срока

их обучения и использоваться при составлении справок о прослушанных курсах и приложений к диплому.

2. Разработать программный модуль «Личные дела студентов». Программный модуль предназначен для получения сведений о студентах сотрудниками деканата, профкома и отдела кадров. Сведения должны храниться в течение всего срока обучения студентов и использоваться при составлении справок и отчетов.

3. Разработать программный модуль «Решение комбинаторно-оптимизационных задач». Модуль должен содержать алгоритмы поиска цикла минимальной длины (задача коммивояжера), поиска кратчайшего пути и поиска минимального связывающего дерева.

4. Разработать приложение «Органайзер». Приложение предназначено для записи, хранения и поиска адресов и телефонов физических лиц и организаций, а также расписания, встреч и др. Приложение предназначено для любых пользователей компьютера.

5. Разработать приложение «Калькулятор». Приложение предназначено для любых пользователей и должно содержать все арифметические операции (с соблюдением приоритетов) и желательно (но не обязательно) несколько математических функций.

6. Разработать программный модуль «Кафедра», содержащий сведения о сотрудниках кафедры (ФИО, должность, ученая степень, дисциплины, нагрузка, общественная работа, совместительство и др.). Модуль предназначен для использования сотрудниками отдела кадров и деканата.

7. Разработать программный модуль «Лаборатория», содержащий сведения о сотрудниках лаборатории (ФИО, пол, возраст, семейное положение, наличие детей, должность, ученая степень). Модуль предназначен для использования сотрудниками профкома и отдела кадров.

8. Разработать программный модуль «Автосервис». При записи на обслуживание заполняется заявка, в которой указываются ФИО владельца,

марка автомобиля, вид работы, дата приема заказа и стоимость ремонта. После выполнения работ распечатывается квитанция.

9. Разработать программный модуль «Учет нарушений правил дорожного движения». Для каждой автомашины (и ее владельца) в базе хранится список нарушений. Для каждого нарушения фиксируется дата, время, вид нарушения и размер штрафа. При оплате всех штрафов машина удаляется из базы.

10. Разработать программный модуль «Картотека агентства недвижимости», предназначенный для использования работниками агентства. В базе содержатся сведения о квартирах (количество комнат, этаж, метраж и др.). При поступлении заявки на обмен (куплю, продажу) производится поиск подходящего варианта. Если такого нет, клиент заносится в клиентскую базу и оповещается, когда вариант появляется.

11. Разработать программный модуль «Картотека абонентов АТС». Картотека содержит сведения о телефонах и их владельцах. Фиксирует задолженности по оплате (абонентской и повременной). Считается, что повременная оплата местных телефонных разговоров уже введена.

12. Разработать программный модуль «Авиакасса», содержащий сведения о наличии свободных мест на авиамаршруты. В базе должны содержаться сведения о номере рейса, экипаже, типе самолета, дате и времени вылета, а также стоимости авиабилетов (разного класса). При поступлении заявки на билеты программа производит поиск подходящего рейса.

13. Разработать программный модуль «Книжный магазин», содержащий сведения о книгах (автор, название, издательство, год издания, цена). Покупатель оформляет заявку на нужные ему книги, если таковых нет, он заносится в базу и оповещается, когда нужные книги поступают в магазин.

14. Разработать программный модуль «Автостоянка». В программе содержится информация о марке автомобиля, его владельце, дате и времени въезда, стоимости стоянки, скидках, задолженности по оплате и др.

15. Разработать программный модуль «Кадровое агентство», содержащий сведения о вакансиях и резюме. Программный модуль предназначен как для

поиска сотрудника, отвечающего требованиям руководителей фирмы, так и для поиска подходящей работы.

При разработке программы не ограничиваться функциями, приведенными в варианте, добавить несколько своих функций. Обязательно использование объектно-ориентированного подхода.

Вопросы и задания для защиты работы

1. Что такое класс в Python? Каковы его основные характеристики?
2. Опишите базовые принципы ООП.
3. Что такое экземпляр класса? Каким образом можно осуществить его создание?
4. Дайте определение атрибута класса и опишите его основные особенности.
5. Что такое методы класса? Каковы особенности создания и вызова метода?
6. Каковы отличия закрытых методов от обычных?
7. В чем заключается преимущество использования конструктора __init__() при создании класса?
8. Как осуществляется перегрузка специальных методов класса в Python?
9. Как реализуется принцип наследования в Python? Приведите примеры.
10. В чем смысл использования абстрактного метода в Python?