Классы

Введение

Всё в Пайтоне **является объектами**. Это означает, что каждый объект в Пайтоне имеет метод и значение по той причине, что все объекты базируются на классе. **Класс** – это проект объекта. Давайте посмотрим на примере, что это значит:

```
x = "Mike"
print(dir(x))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__',
'__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '_formatter_field_name_split',
'_formatter_parser', 'capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha',
'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

В примере мы видим строку, присвоенную переменной x. Это может выглядеть как большой объем, но дело в том, что у этой строки много **методов**. Если вы используете ключевое слово **dir**, вы получите **список всех методов**, которые можно присвоить строке. Мы видим 71 метод! Технически, мы не можем вызвать методы, которые начинаются с подчеркивание, так что это сужает список до 38 методов, но это все еще очень много! *Что это значит?* Это значит что, строка основана на классе, а переменная x – и есть экземпляр этого класса. В Пайтоне мы можем создавать собственные классы. Начнем!

Создание Класса

Создание класса в Пайтоне – это очень просто. Вот простой пример:

```
# Python 2.x syntax

class Vehicle(object):
    """docstring"""

    def __init__(self):
        """Constructor"""
        pass
```

Этот класс не делает ничего конкретного, тем не менее, это очень хороший инструмент для изучения. Например, чтобы создать класс, мы используем ключевое слово **class**, за которым следует наименование класса. В Пайтоне, конвенция указывает на то, что наименование класса должно начинаться с заглавной буквы. Далее нам нужно открыть круглые скобки, за которыми следует слово **object** и закрытые скобки. «**object**» — то, на чем основан класс, или наследуется от него. Это называется базовым классом или родительским классом. Большая часть классов в Пайтоне основаны на объекте. У классов есть особый метод, под названием **__init__**.

Этот метод вызывается всякий раз, когда вы создаете (или создаете экземпляр) объект на основе этого класса. Метод **__init__** вызывается единожды, и не может быть вызван снова внутри программы. Другое определение метода **__init__** — это конструктор, кстати, этот термин редко встречается в Пайтоне. Вы можете подумать, почему я называю

это методом, а не функцией? Функция меняет свое имя на «**method**», когда она находится внутри класса. Обратите внимание на то, что каждый метод *должен иметь как минимум один аргумент*, что в случае с обычной функцией уже не вяжется. В Python 3 нам не нужно прямо указывать, что мы наследуем у объекта. Вместо этого, мы можем написать это следующим образом:

```
# Python 3.x syntax

class Vehicle:
    """docstring"""

    def __init__(self):
        """Constructor"""
        pass
```

Обратите внимание на то, что единственная разница в том, что круглые скобки нам больше не нужны, когда мы **основываем наш класс на объекте**. Давайте немного расширим наше определение класса и дадим ему некоторые атрибуты и методы.

```
class Vehicle(object):
    """docstring"""

    def __init__(self, color, doors, tires):
        """Constructor"""
        self.color = color
        self.doors = doors
        self.tires = tires

    def brake(self):
        """
        Stop the car
        """
        return "Braking"

    def drive(self):
        """
        Drive the car
        """
        return "I'm driving!"
```

В данном примере мы добавили три атрибута и два метода. Эти три атрибута являются:

```
self.color = color
self.doors = doors
self.tires = tires
```

Атрибуты описывают автомобиль. У него есть цвет, определенное количество дверей и колес. Также у него есть два метода. **Метод** описывает, что делает класс. В нашем случае, автомобиль может двигаться и останавливаться. Вы могли заметить, что все методы, включая первый, имеют интересный аргумент, под названием self. Давайте рассмотрим его внимательнее.

Что такое self?

Классам нужен способ, что **ссылаться на самих себя**. Это не из разряда нарциссичного отношения со стороны класса. Это способ сообщения между экземплярами. Слово **self** это способ описания любого объекта, буквально. Давайте взглянем на пример, который мне кажется наиболее полезным, когда я сталкиваюсь с чем-то новым и странным:

Добавьте этот код в конец класса, который вы написали ранее и сохраните:

```
class Vehicle(object):
    """docstring"""

    def __init__(self, color, doors, tires):
        """Constructor"""
        self.color = color
        self.doors = doors
        self.tires = tires

    def brake(self):
        """
        Stop the car
        """
        return "Braking"

    def drive(self):
        """
        Drive the car
        """
        return "I'm driving!"

if __name__ == "__main__":
    car = Vehicle("blue", 5, 4)
    print(car.color)

    truck = Vehicle("red", 3, 6)
    print(truck.color)
```

Условия оператора **if** в данном примере это стандартный способ указать Пайтону на то, что вы хотите запустить код, если он выполняется как автономный файл. Если вы импортировали свой модуль в другой скрипт, то код, расположенный ниже проверки **if** не заработает. В любом случае, если вы запустите этот код, вы создадите два **экземпляра класса** автомобиля (Vehicle): класс легкового и класс грузового. Каждый экземпляр будет иметь свои **собственные атрибуты и методы**. Именно по этому, когда мы выводи цвета каждого экземпляра, они и отличаются друг от друга. Причина в том, что этот класс использует **аргумент self**, чтобы указать самому себе, что есть что. Давайте немного изменим класс, чтобы сделать методы более уникальными:

```
class Vehicle(object):
    """docstring"""

    def __init__(self, color, doors, tires, vtype):
        """Constructor"""
        self.color = color
        self.doors = doors
        self.tires = tires
        self.vtype = vtype

    def brake(self):
        """
        Stop the car
        """
        return "%s braking" % self.vtype

    def drive(self):
        """
        Drive the car
        """
        return "I'm driving a %s %s!" % (self.color, self.vtype)


if __name__ == "__main__":
    car = Vehicle("blue", 5, 4, "car")
```

```
    print(car.brake())
    print(car.drive())

    truck = Vehicle("red", 3, 6, "truck")
    print(truck.drive())
    print(truck.brake())
```

В этом примере мы передаем другой параметр, чтобы сообщить классу, какой тип транспортного средства мы создаем. После этого мы вызываем каждый метод для каждого экземпляра. Если вы запустите данный код, вы получите следующий вывод:

```
car braking
I'm driving a blue car!
I'm driving a red truck!
truck braking
```

Это показывает, как экземпляр отслеживает свой **аргумент self**. Вы также могли заметить, что мы можем переместить переменные атрибутов из метода **__init__** в другие методы. Это возможно потому, что все эти атрибуты связанны с аргументом self. Если бы мы этого не сделали, переменные были бы вне области видимости в конце метода **__init__** .

Подклассы

Настоящая сила классов становится очевидной, когда вопрос касается **подклассов**. Вы, возможно, еще не поняли это, но мы уже создали подкласс, когда создавали класс, основанный на объекте. Другими словами, «**подклассифицировали**» объект. Так как объект – это не очень интересная тема, предыдущие примеры не уделили должного внимания такому сильному инструменту как **подкласс**. Давайте подклассифицируем наш класс Vehicle и узнаем, как все это работает.

```
class Car(Vehicle):
    """
    The Car class
    """

    #--------------------------------------------------------------------
    def brake(self):
        """
        Override brake method
        """
        return "The car class is breaking slowly!"


if __name__ == "__main__":
    car = Car("yellow", 2, 4, "car")
    car.brake()
    'The car class is breaking slowly!'
    car.drive()
    "I'm driving a yellow car!"
```

В этом примере, мы подклассифицировали **класс Vehicle**. Вы могли заметить, что мы не использовали методы **__init__** и drive. Причина в том, что когда мы хотим сделать из класса подкласс, мы уже имеем все атрибуты и методы, только если мы не переопределяем их. Таким образом, вы могли заметить, что мы переопределяем **метод brake** и указываем ему делать кое-что другое. Другие методы остаются такими же, какими они и были до этого. Так что, когда вы указываете автомобилю тормозить, он использует оригинальный метод, и мы узнали, что мы водим желтый автомобиль. Когда мы используем значения родительского класса по умолчанию – мы называем это **наследование**.

Это достаточно большой раздел в **объектно-ориентированном программировании**. Это также простой пример **полиморфизма**. **Полиморфические**

классы имеют одинаковый интерфейс (методы, атрибуты), но они не контактируют друг с другом. Касаемо полиморфизма в Пайтоне, не очень сложно выяснить, что интерфейсы являются идентичными. С этого момента мы знакомимся с понятием утиная типизация. Суть утиной типизации заключается в том, что если это ходит как утка, и крякает как утка – значит, это должна быть утка.

Задание

1. Создать два класса согласно варианту. Один класс будет основным, и он содержит массив второго(дополнительного) класса.
2. Реализовать конструктор в дополнительном классе для инициализации
3. Реализовать метод добавления в массив основного класса
4. Реализовать метод удаления в массиве основного класса
5. Реализовать метод редактирования в массиве основного класса
6. Реализовать метод поиска в массиве основного класса
7. Реализовать метод сортировки в массиве основного класса
8. Реализовать вывод данных в основном и дополнительном классе

Вариант 1
Основной класс: Журнал содержит номер, имя
Дополнительный класс: Статья содержит дату, тему, отправителя

Вариант 2
Основной класс: Журнал содержит карта, имя
Дополнительный класс: Платеж содержит дату, сумму, описание

Вариант 3
Основной класс: Энциклопедия содержит название, год издания
Дополнительный класс: Личность содержит имя, род деятельности, описание

Вариант 4
Основной класс: Кинотеатр содержит название, адрес
Дополнительный класс: Сеанс содержит дату, время, название

Вариант 5
Основной класс: Календарь содержит название помещения, адрес
Дополнительный класс: Мероприятие содержит дату, название, описание

Вариант 6
Основной класс: Каталог содержит название, год создания
Дополнительный класс: Фильм содержит название, жанр, описание

Вариант 7
Основной класс: Библиотека содержит название, адрес
Дополнительный класс: Книга содержит название, жанр, автор

Вариант 8
Основной класс: Отдел содержит название, телефон
Дополнительный класс: Работник содержит имя, должность, дату приема

Вариант 9
Основной класс: Приложение содержит название, требуем объем
Дополнительный класс: Пользователь содержит логин, пароль, роль

Вариант 10
Основной класс: Расписание содержит номер учебного заведения, адрес
Дополнительный класс: Занятие содержит название, день недели, время

Вариант 11
Основной класс: Журнал содержит название предмета, имя учителя
Дополнительный класс: Урок содержит дату, тему, номер класса

Вариант 12
Основной класс: Журнал содержит название организации, телефон
Дополнительный класс: Клиент содержит имя, адрес, номер договора

Вариант 13
Основной класс: Тест содержит название предмета, сложность
Дополнительный класс: Вопрос содержит текст вопроса, ответ, количество баллов

Вариант 14
Основной класс: Журнал содержит организацию, контактный телефон
Дополнительный класс: Экскурсия содержит дату, экскурсовод, описание

Вариант 15
Основной класс: Справочник содержит название, имя владельца
Дополнительный класс: Контакт содержит имя, телефон, адрес

Вариант 16
Основной класс: Журнал содержит название организации, имя водителя
Дополнительный класс: Рейс содержит дату, время, цена

Вариант 17
Основной класс: Мастерская содержит название, телефон
Дополнительный класс: Мастер содержит имя, возраст, стиль

Вариант 18
Основной класс: СТО содержит адрес, телефон
Дополнительный класс: Машина содержит модель, причина поломки, год выпуска

Вариант 19
Основной класс: Магазин содержит название, имя директора
Дополнительный класс: Товар содержит название, тип, цена

Вариант 20
Основной класс: Журнал содержит год поступления, специальность
Дополнительный класс: Студент содержит имя, возраст, средний бал