Лекция 9

Язык программирования Python.

Хайрулин Сергей Сергеевич

email: s.khairulin@g.nsu.ru, s.khayrulin@gmail.com

Ссылка на материалы

План

- Лекции/практические занятия
  - Тест
- Дифференцированный зачет в конце семестра
  - Защита задания

Литература

Начальный уровень

- Mark Pilgrim. Dive into Python - http://www.diveintopython.net/
- Марк Лутц. Изучаем Python, 4-е издание // Символ-Плюс 2011.
- ...

Стандарт/Документация

- PEP-8 - https://www.python.org/dev/peps/pep-0008/
- https://www.python.org/
- https://github.com/python/cpython

Экспертный уровень

- Лучано Рамальо: Python. К вершинам мастерства
- Mitchell L. Model. Bioinformatics Programming Using Python // O'Reilly 2010.

Версии Python

- Python 2 вышел 2010 году последняя версия 2.7.16 - исправлялись только баги(ошибки) с января 2020 года поддержка прекращена.
- Python 3 в появился в 2008, является актуальной версией языка. Текущая стабильная версия 3.9, в разработке 3.10
  - Python 3 не гарантирует совместимости кода с Python 2

Summary

- Абстрактные типы данных (АТД)
- ООП
  - Наследование
  - Полиморфизм
  - Инкапсуляция
- ООП в Python
  - Объект object
  - Ключевое слово class
  - метод __init__
  - ключевое слово self
  - свойства/атрибуты класса
  - методы/функции

Абстрактные типы данных (АТД)

АТД - математическая модель с совокупностью операторов, определенных в рамках этой модели (А. Ахо).

Абстрактные типы данных (АТД)

Пример:

- списки - **list**
- словарь - **dict**
- множества - **set**

Не входит в стандартную библиотеку

- графы - graph

Абстрактные типы данных (АТД)

В **Python**:

Базовые типы int, float, str, bool ...

Комплексные типы:

- **list**: append, insert, pop ...
- **dict**: insert, get, extend …

…

ООП

Объе́ктно-ориенти́рованное программи́рование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования (Г. Буч 1998)

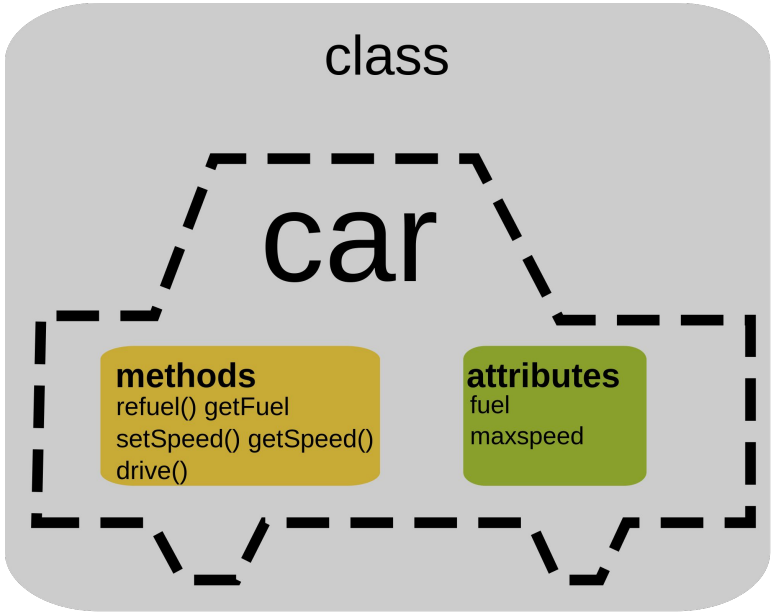
Языки поддерживающие **ООП**: C++, Java, C#, Python, Perl, Go….

ООП

Объектно-ориентированная парадигма имеет несколько принципов:

- Данные структурируются в виде объектов, каждый из которых имеет определенный тип, то есть принадлежит к какому-либо классу.
- Классы – результат формализации решаемой задачи, выделения главных ее аспектов.
- Внутри объекта инкапсулируется логика работы с относящейся к нему информацией.
- Объекты в программе взаимодействуют друг с другом, обмениваются запросами и ответами.
- При этом объекты одного типа сходным образом отвечают на одни и те же запросы.
- Объекты могут организовываться в более сложные структуры, например, включать другие объекты или наследовать от одного или нескольких объектов.

Объект



Classname
(Identifier)
Data Member
(Static attributes)
Member Functions
(Dynamic Operations)

Student	Circle
name grade	radius color
getName() printGrade()	getRadius() getArea()

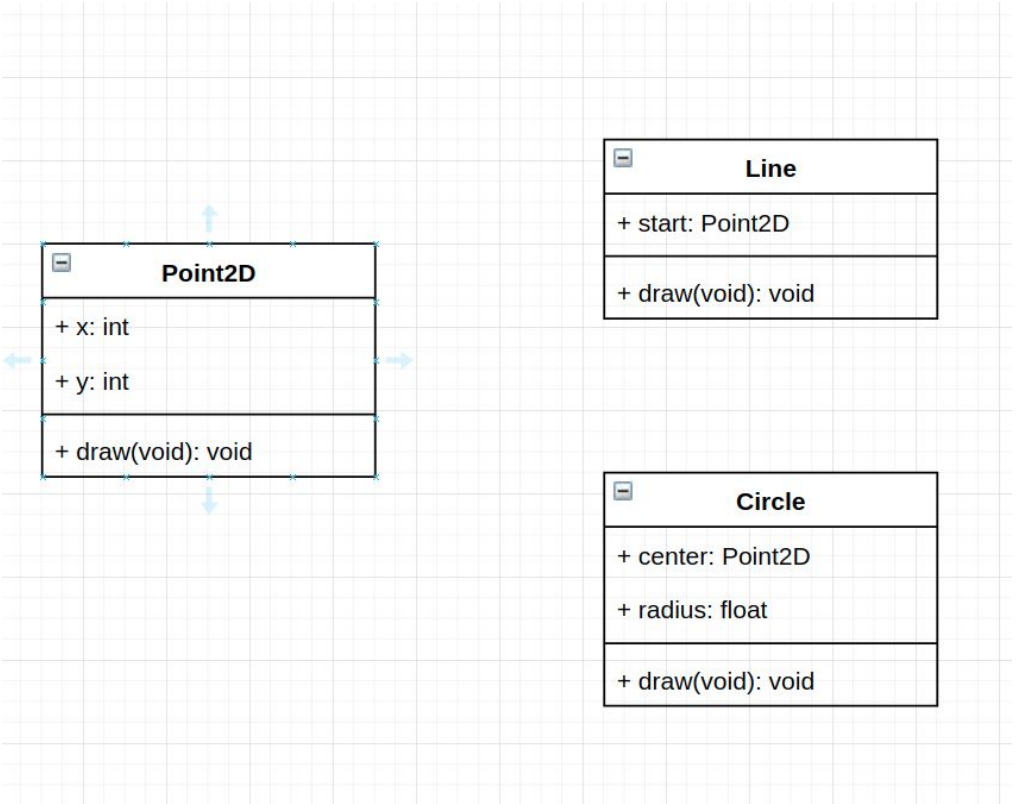
SoccerPlayer	Car
name number xLocation yLocation	plateNumber xLocation yLocation speed
run() jump() kickBall()	move() park() accelerate()

Examples of classes

Объект

Графическая редактирования графики.

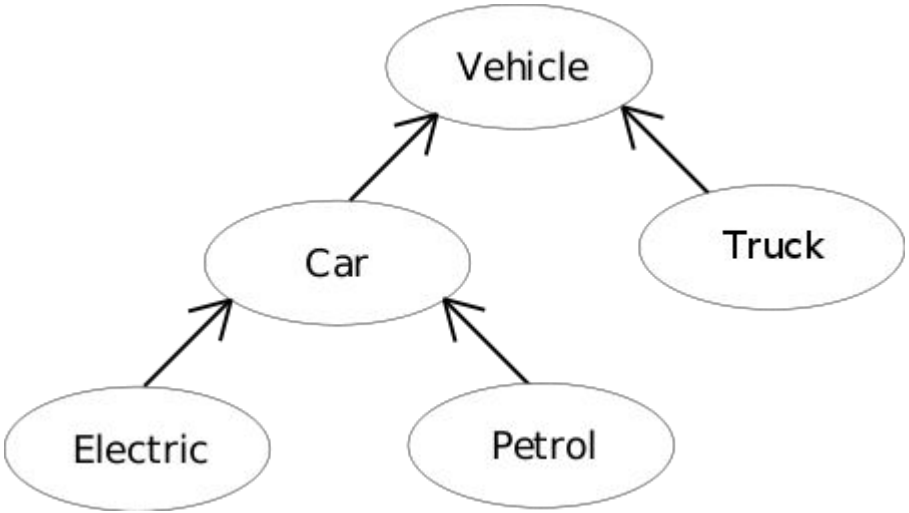
1. Создание сцен из различных графических примитивов
2. Редактирование сцен
3. Сохранение в файл
4. Загрузка из файла
5. … (еще что-нибудь)

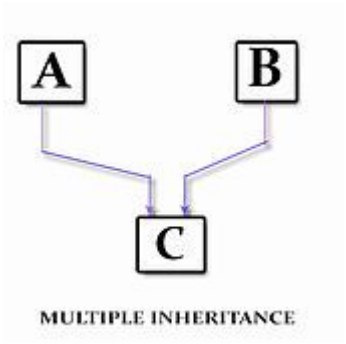


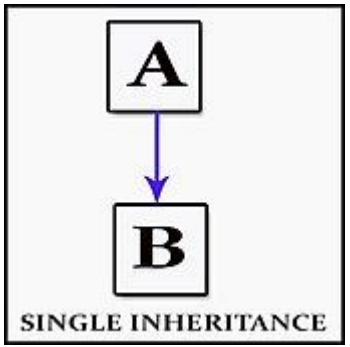
ООП принципы

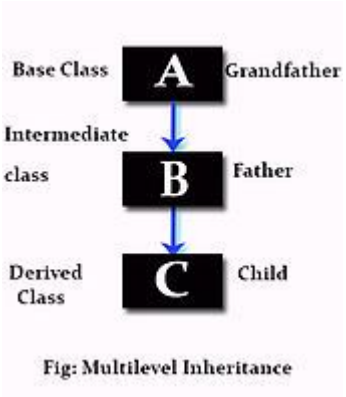
- Наследование
  - Возможность создание новых типов данных базирующихся на других, ранее определенных
- Полиморфизм
  - Возможность переопределения поведения базовых свойств объекта (свойств унаследованных от объектов предков)
- Инкапсуляция
  - Возможность скрывать реализацию тех или иных свойств объекта от конечного пользователя

Наследование



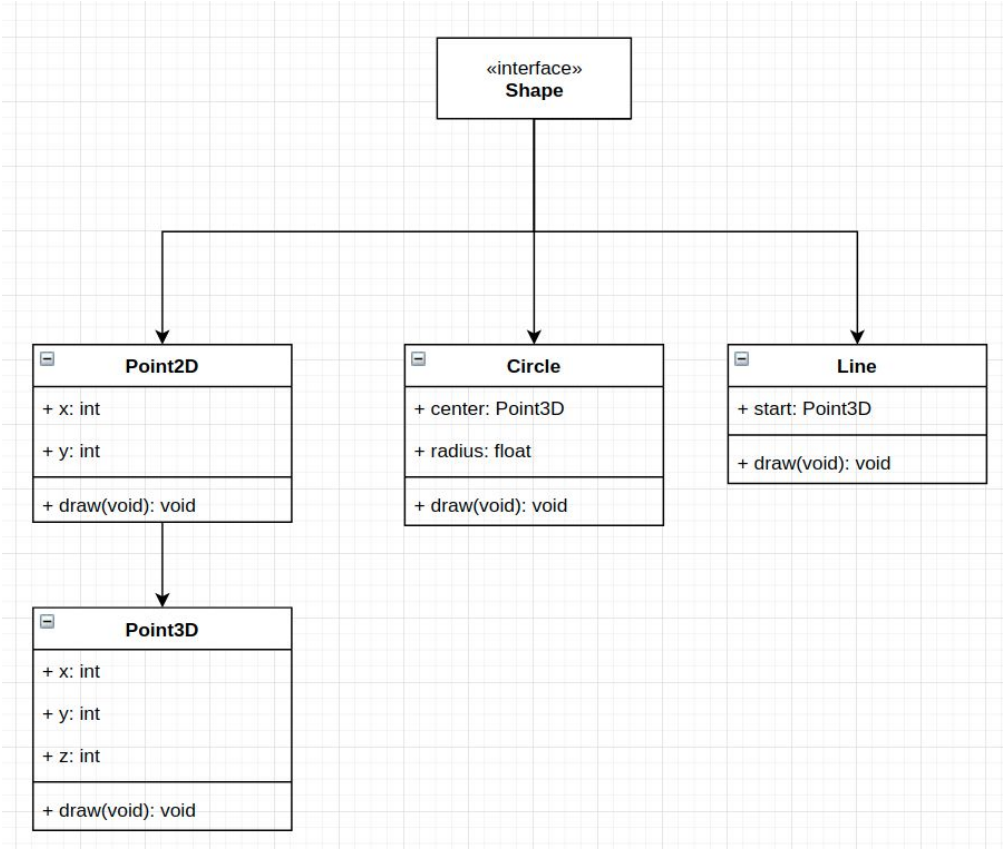




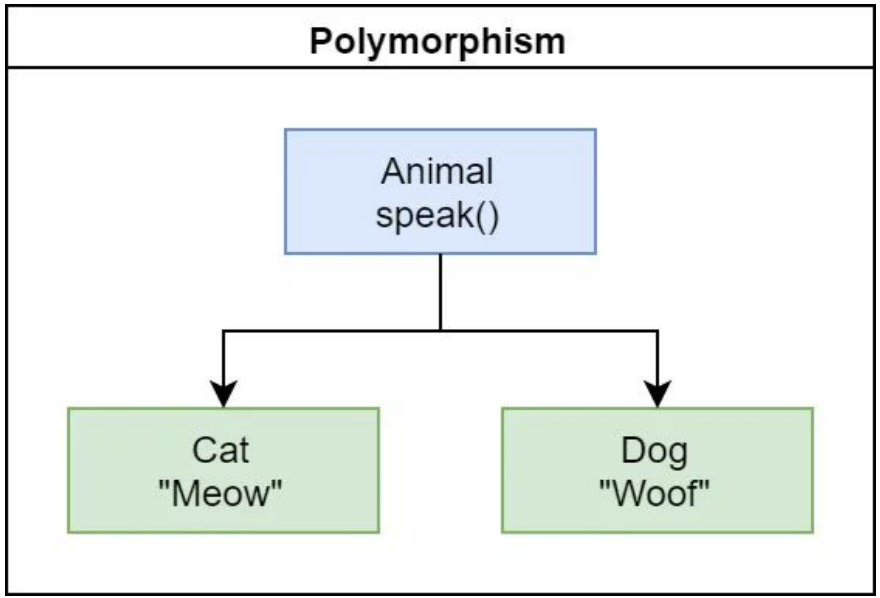


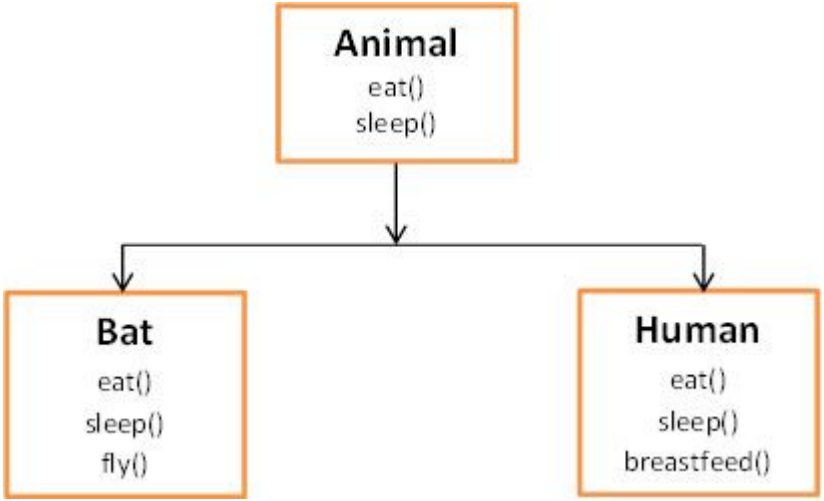
Наследование

В нашей графической библиотеки много сущностей связаны - похожи. Иерархически на это можно посмотреть так, что все графические примитивы обладают как минимум одним общим свойством “их можно рисовать”.



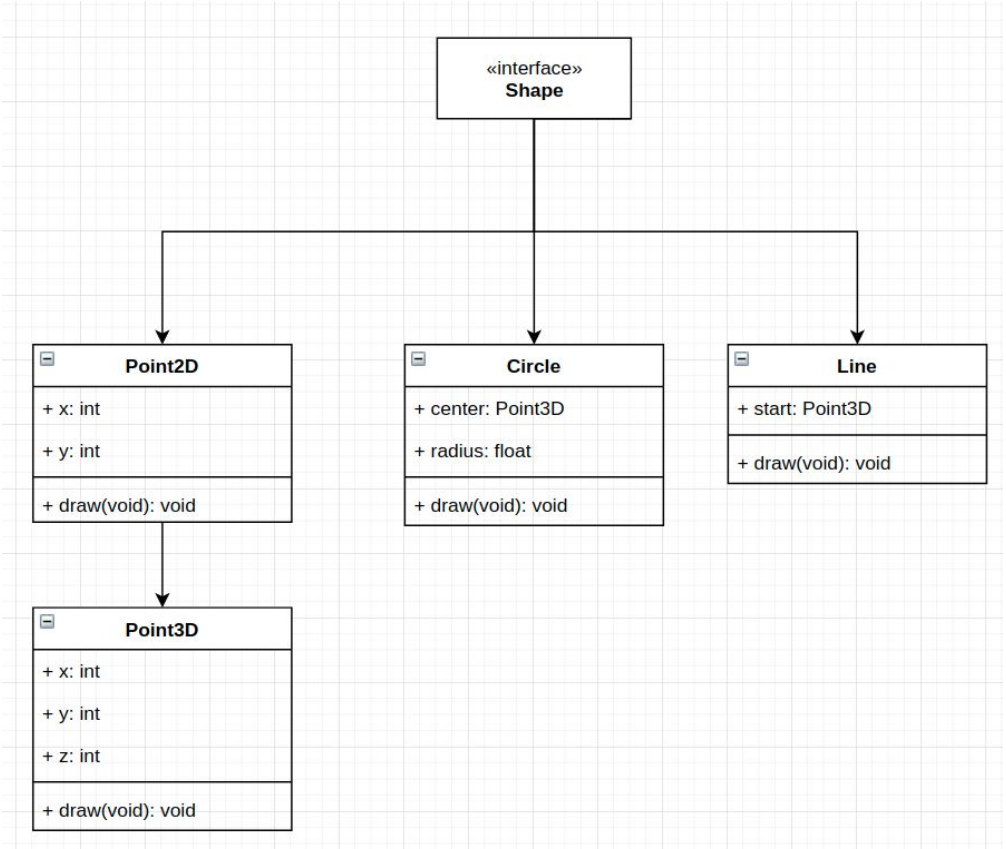
Полиморфизм



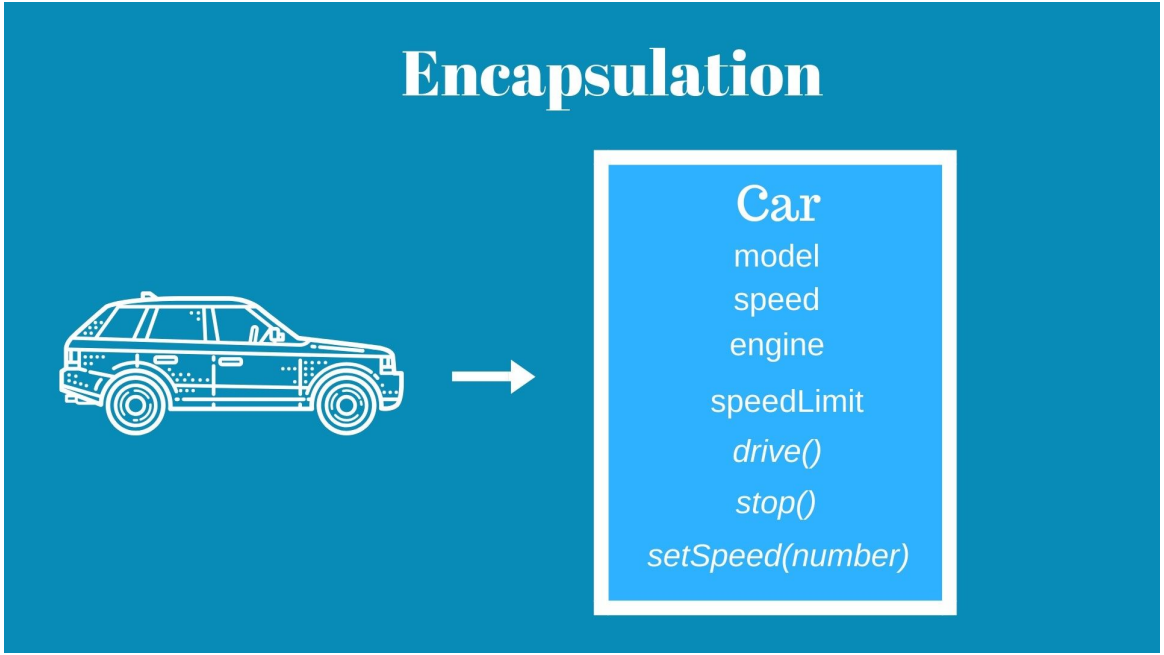


Полиморфизм

Итак мы определились с тем, что все примитивы можно рисовать. **НО,** очевидно, что рисуются все фигуры по разному.

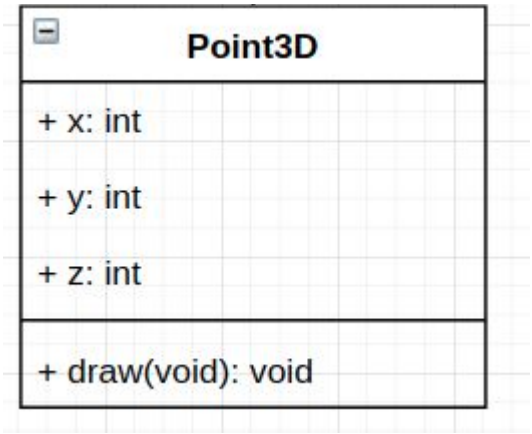


Инкапсуляция



Инкапсуляция

Инкапсуляция - это совокупность всех атрибутов и методов класса, которые характеризуют сам класс.



ООП в Python

Python - поддерживает принципы ООП. То есть язык является объектно-ориентированным, **НО** в отличие от полностью объектно-ориентированных языков, таких как JAVA, C#, **Python** - только лишь поддерживает эту парадигму программирования, также про **Python** можно сказать, что он структурный и процедурный...

Объект object

Return a new featureless object. `object` is a base for all classes. It has the methods that are common to all instances of Python classes. This function does not accept any arguments (https://docs.python.org/3/library/functions.html?highlight=object#object)

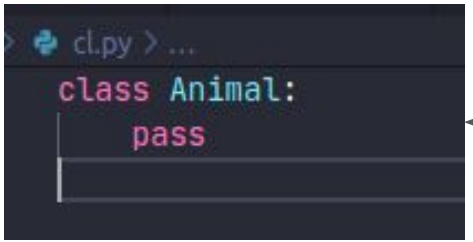
Классы в Python

Для того чтобы определить свой класс, нужно пользоваться конструкцией

```
class ClassName(ParentClass1, ParentClass2, ...):
    # code here
    …
```

Классы могут не иметь предков и соответственно классы предки не указываются в скобках, но явно каждый созданный класс, является наследником класса **object**

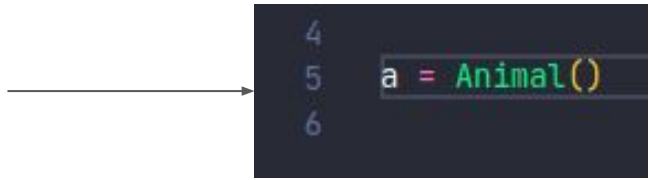
Ключевое слово class



Определение класса

Ключевое слово class

Создание экземпляра класса



Ключевое слово class

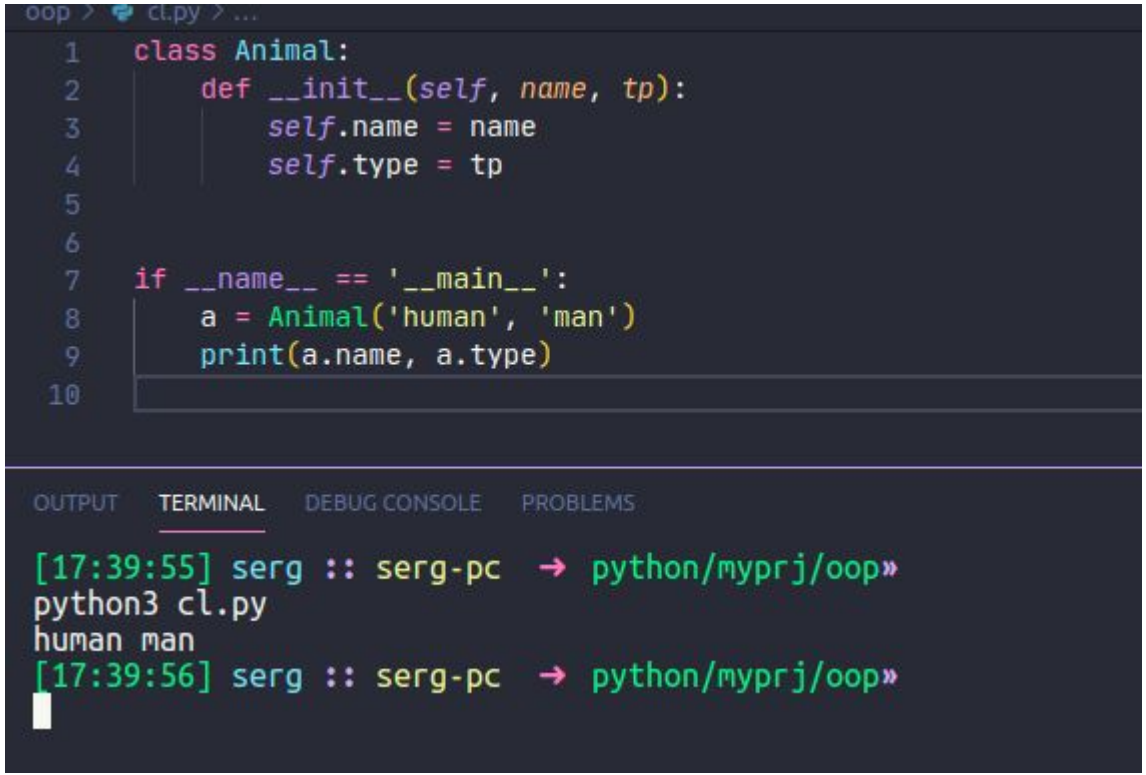

```python
from .shape import Shape

class Point2D(Shape):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def draw(self):
        pass

class Point3D(Point2D):
    def __init__(self, x, y, z):
        super().__init__(x, y)
        self.z = z

    def draw(self):
        pass

p2D = Point2D(1,2)

p3D = Point3D(1,2,3)
```

Метод конструктора __init__

Обычно для того чтобы определить экземпляр класса, обычно определятся специализированный метод который называется конструктором. Конструктор - это функция, которая может принимать (а может и не принимать) некоторое количество аргументов, с помощью которых инициализируется атрибуты этого класса.

В Python - таким методом называется специально именованная функция __init__(...)

Метод конструктора __init__



Ключевое слово self

oop > cl.py > ...

```python
class Animal:
    def __init__(self, name, tp):
        self.name = name
        self.type = tp


if __name__ == '__main__':
    a = Animal('human', 'man')
    print(a.name, a.type)

```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[17:39:55] serg :: serg-pc  → python/myprj/oop»
python3 cl.py
human man
[17:39:56] serg :: serg-pc  → python/myprj/oop»
```

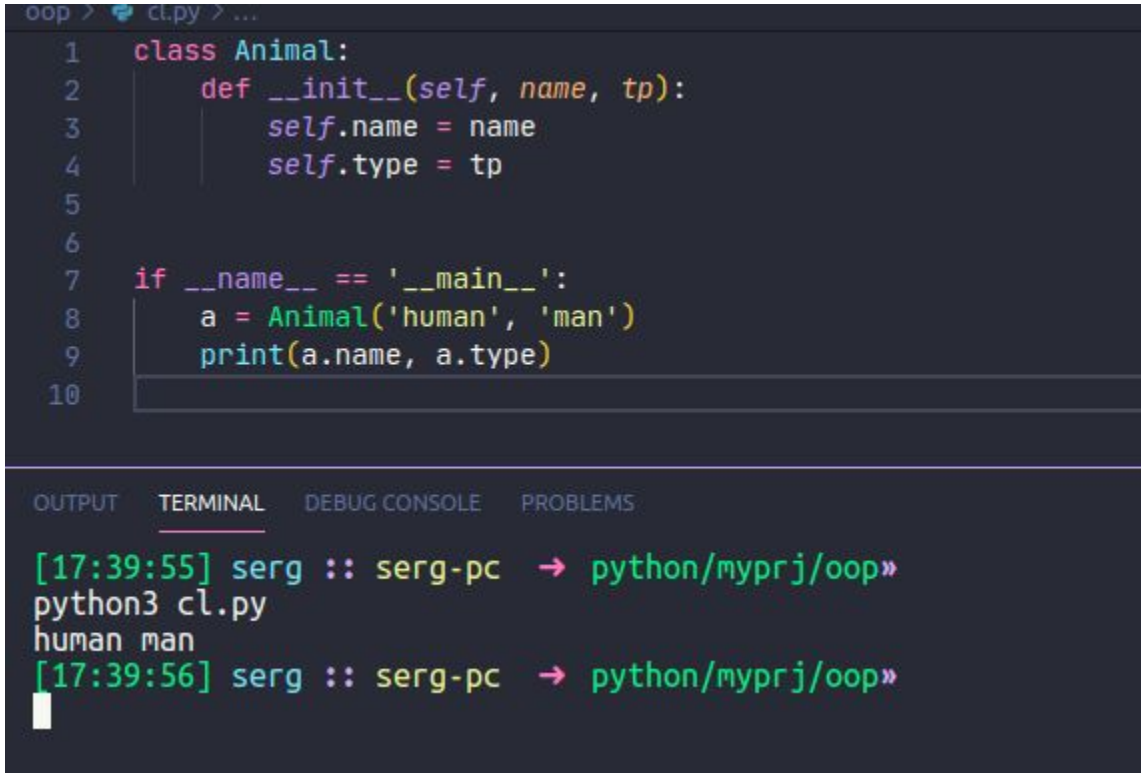
a = A()
self == a
a.someMeth(arg1) ⇔ A.someMeth(a, arg1)
self - это указатель на текущий экземпляр класса, для которого осуществляется вызов.

Свойства/атрибуты класса

Атрибуты класса - это характеристики, которые описывают свойства класса. Например для класса Человек - такими характеристиками могут быть рост, вес, имя, возраст и так далее.

Для того чтобы задать атрибут обычно используется специализированный атрибут *self,* который является первым аргументов любого метода класса

Свойства/атрибуты класса



Статические свойства/атрибуты класса

Статические поля/аргументы - доступны всем экземплярам класса экземпляры также могут их менять. Обычно если атрибут пишется с большой буквы, то подразумевается, что такой атрибут не доступен для изменений.

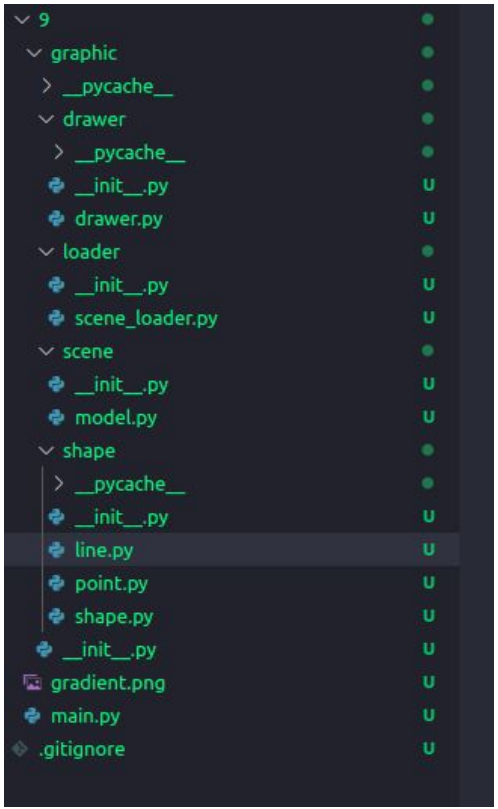
```python
from .shape import Shape

class Point2D(Shape):
    description = "This is definition of 2D point"

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def draw(self):
        pass

class Point3D(Point2D):
    def __init__(self, x, y, z):
        super().__init__(x, y)
        self.z = z

    def draw(self):
        pass
```

Методы/функции

Методы класса - это характеристики, которые определяют поведение класса, то есть функции которые меняют состояние экземпляра класса.

Для того чтобы задать метод нужно определять функцию с первым аргументов == *self, self* - дает доступ методу к конкретному экземпляру класса, для которого производиться вызов конкретного метода.

Методы/функции



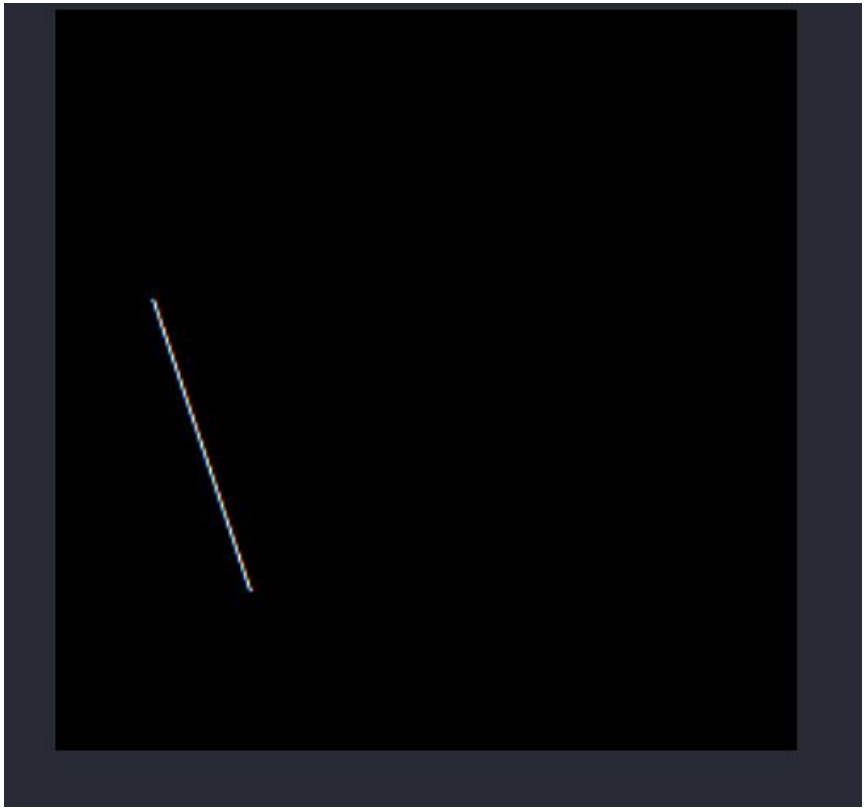
9 > graphic > shape > line.py > ...

```python
from .shape import Shape
from ..drawer import Color

class Line(Shape):
    def __init__(self, start, end, drawer):
        Shape.__init__(self, drawer)
        self.start = start
        self.end = end

    def draw(self):
        """ Realization of Bresenham's algorithm for drawing lines
        """
        dx = self.end.x - self.start.x
        dy = self.end.y - self.start.y
        yi = 1
        if dy < 0:
            yi = -1
            dy = -dy
        D = 2*dy - dx
        y = self.start.y
        c = Color(255,255,255)
        for x in range(self.start.x, self.end.x + 1):
            self.drawer.put_pixel(x, y, c)
            if D > 0:
                y = y + yi
                D = D + (2 * (dy - dx))
            else:
                D = D + 2 * dy

```

Методы/функции

```
9 > main.py > main
1   from graphic.drawer import Drawer, Color
2   from graphic.shape import Point2D, Line
3
4   def main():
5       drawer = Drawer(256, 256)
6       p_start = Point2D(100, 100, drawer=drawer)
7       p_end = Point2D(200, 200, drawer=drawer)
8       l = Line(p_start, p_end, drawer)
9       l.draw()
10      drawer.save()
11
12  if __name__ == '__main__':
13      main()
```

Методы/функции



Статические методы класса

Статические методы - обладают похожим с обычными методами поведением, но при этом эти методы не имеют доступ к полям экземпляра класса, то есть у такого метода нет аргумента self.

Статические методы класса

```python
from .shape import Shape
from ..drawer import Color

class Line(Shape):
    line_counter = 0
    def __init__(self, start, end, drawer):
        Shape.__init__(self, drawer)
        self.start = start
        self.end = end

    @staticmethod
    def stat():
        Line.line_counter += 1
        print(f"{Line.line_counter} was created")
```

Задачи

1. Подумайте, какими свойствами и методами могли бы обладать следующие объекты: Ученик, Учитель, Школа, Экзамен, Турнир.
2. Опишите несколько из объектов перечисленных выше на Python.
3. Напишите структуры классов для некоторой древовидной структуры данных: дерево, корень, потомки.
4. Реализуйте метод прохода по такой структуре (в глубину, в ширину)

Лабораторная работа № 3

Необходимо дополнить библиотеку для рисования.

1. Научить ее рисовать круги
2. Научить ее рисовать прямоугольники
3. Сохранять/загружать сцены из текстовых файлов формат

```
figure 2

circle
10 10        // center
10           // radius
155 155 155 // color

squre
100 100      // x0 y0
10, 10       // w, h
155 155 155 // color

```