Лекция 10

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

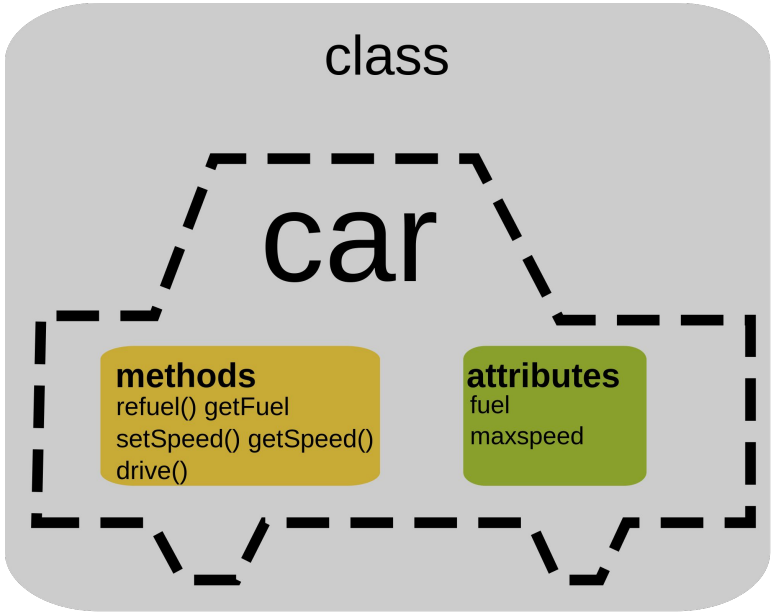
Summary

- Наследование в Python.
  - Единичное,
  - множественное наследование.
- Утиная типизация.
- Полиморфизм в Python.
- Переопределение поведения функции в зависимости от аргументов (*args, **kwargs).
- Статические метода, атрибуты класса.

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

Объект



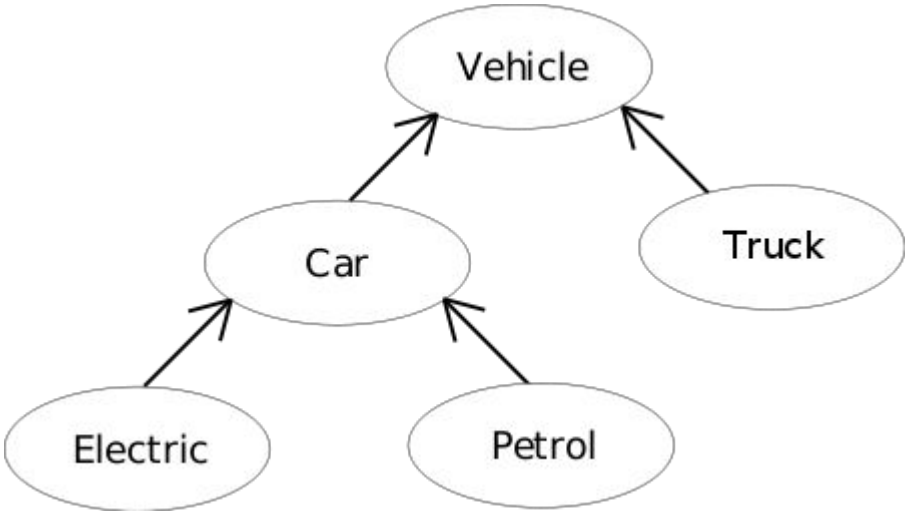
Classname
(Identifier)
Data Member
(Static attributes)
Member Functions
(Dynamic Operations)

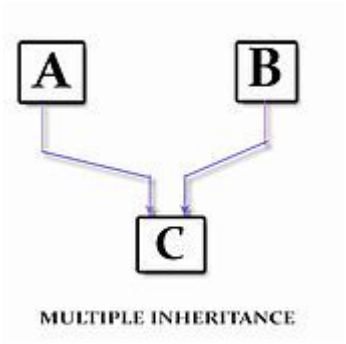
Student	Circle
name grade	radius color
getName() printGrade()	getRadius() getArea()

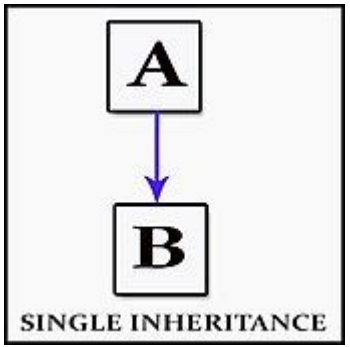
SoccerPlayer	Car
name number xLocation yLocation	plateNumber xLocation yLocation speed
run() jump() kickBall()	move() park() accelerate()

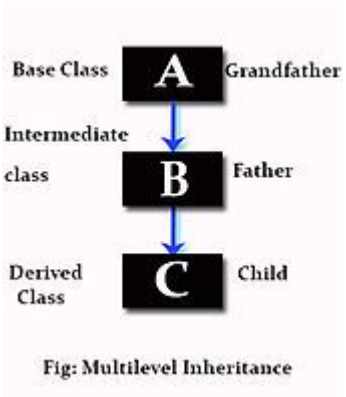
Examples of classes

Наследование



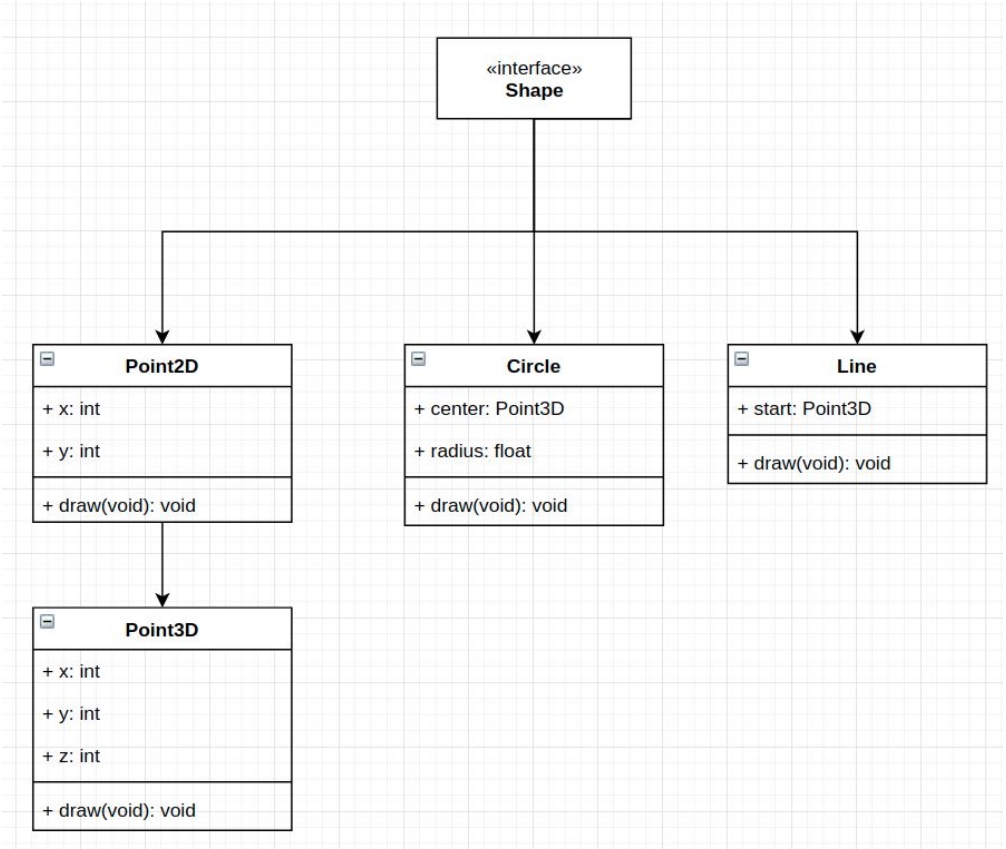






Наследование

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”.



Классы в Python

Для того чтобы определить свой класс, нужно пользоваться конструкцией

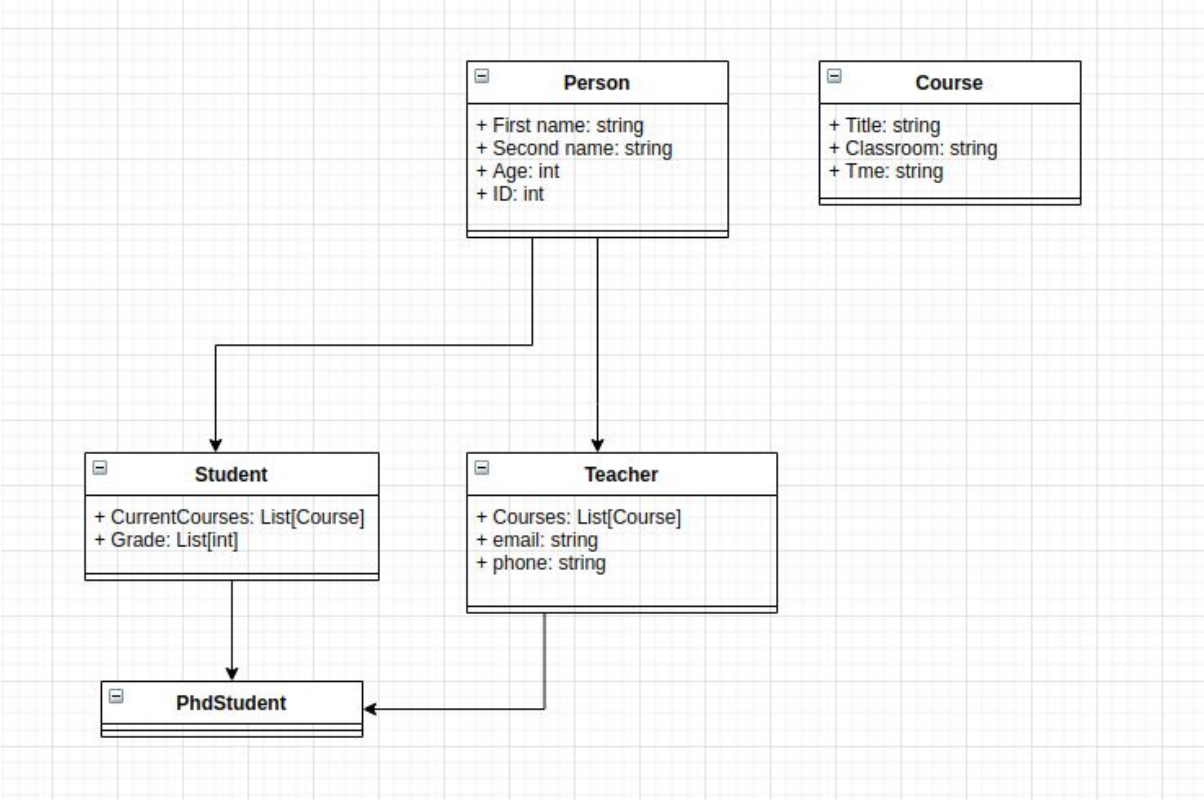
```
class ClassName(ParentClass1, ParentClass2, ...):
    # code here
    …
```

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

Наследование в Python

1. Класс может не иметь предков
    a. Исключение только, что все объекты в Python являются потомками object, то есть **неявно** классы являются наследниками
2. Класс может быть потомком нескольких классов
3. Можно выстраивать сложные иерархические связи в том числе с пересечениями в иерархии

Наследование в Python.



Инициализация полей предков

Наследники некоторого класса получают теже свойства, что и их предок, но также они могут расширять свои характеристики засчет добавления новых полей. Инициализировать поля класса предка можно следующим образом

1. Никак не инициализировать
2. использовать вызов функции **super()**
3. Явно вызывать метод __init__(...) с аргументами, для каждого предка

Инициализация полей предков

Обратиться к атрибутам базового класса не получиться так как их нет...

10 > inheritance > exp.py > ...

```python
class Base:
    def __init__(self, x, y):
        print("Run Paren __init__ method")
        self.x = x
        self.y = y

class Children(Base):
    def __init__(self, z):
        self.z = z

if __name__ == '__main__':
    c = Children(1)
    print(c.x, c.y, c.z)
```

TERMINAL SQL CONSOLE PROBLEMS OUTPUT DEBUG CONSOLE

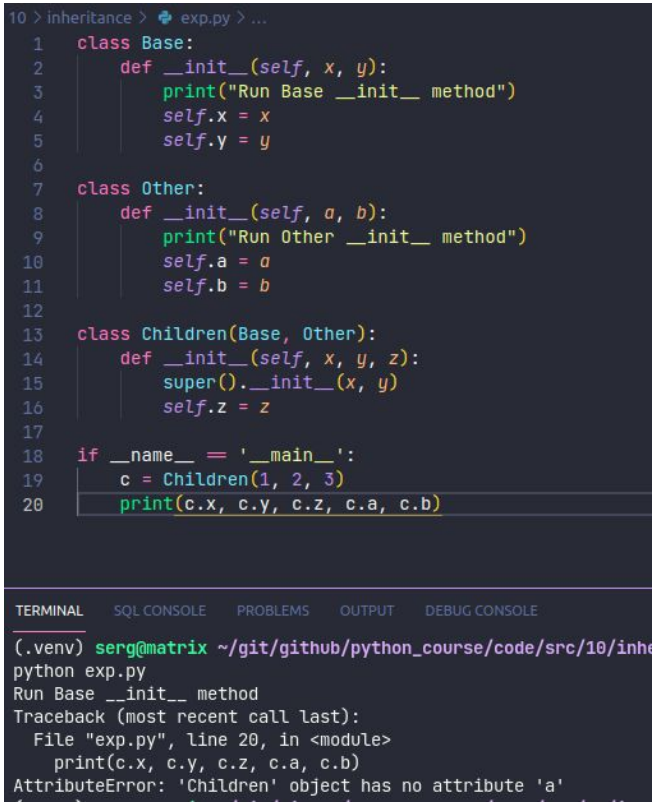
```
(.venv) serg@matrix ~/git/github/python_course/code/src/10/inheritance <master*>
python exp.py
Traceback (most recent call last):
  File "exp.py", line 13, in <module>
    print(c.x, c.y, c.z)
AttributeError: 'Children' object has no attribute 'x'
```

Инициализация полей предков

Функция **super()** - позволяет обращаться к методам базового класса. Не обязательно к методу __init__

Инициализация полей предков

Если базовых классов много, то функция **super()** вернет ссылку только на первого предка, соответственно проинициализировать поля других предков не получится



Инициализация полей предков

Явное указание какой метод и какого класса нужно вызвать это выход. **НО** в отличии от **super()** нужно явно передавать ссылку на экземпляр класса self для которого производиться вызов метода базового класса.

10 > inheritance > exp.py > ...

```python
class Base:
    def __init__(self, x, y):
        print("Run Base __init__ method")
        self.x = x
        self.y = y

class Other:
    def __init__(self, a, b):
        print("Run Other __init__ method")
        self.a = a
        self.b = b

class Children(Base, Other):
    def __init__(self, x, y, z):
        Base.__init__(self, x, y)
        Other.__init__(self, x, y)
        self.z = z

if __name__ == '__main__':
    c = Children(1, 2, 3)
    print(f"{c.x=} {c.y=} {c.z=} {c.a=} {c.b=}")
```

TERMINAL SQL CONSOLE PROBLEMS OUTPUT DEBUG CONSOLE

```
(.venv) serg@matrix ~/git/github/python_course/code/src/10/inhe
python exp.py
Run Base __init__ method
Run Other __init__ method
c.x=1 c.y=2 c.z=3 c.a=1 c.b=2
```

ВНИМАНИЕ!!!

Атрибуты у экземпляра класса могут появляться/исчезать во время жизни объекта,

Наследование в Python.

Наследование в Python.

```python
class Course:
    def __init__(self, title, classroom, time):
        self.title = title
        self.classroom = classroom
        self.time = time

    def get_info(self):
        return f"Course - {self.title}, place - {self.classroom}, time - {self.time}"

class Student(Person):
    def __init__(self, first_name, second_name, age, id, current_courses):
        super().__init__(first_name, second_name, age, id)
        self.current_courses = current_courses

    def get_info(self):
        base_info = super().get_info()
        info = f"Student {base_info} \n"
        info += "============Course List============\n"
        for course in self.current_courses:
            info += course.get_info()
        return info
```

Наследование в Python.

```
35
36  if __name__ == '__main__':
37      s = Student("John", "Dow", 23, 1, [Course("Differential equations", 321, "Wen 11:00 AM")])
38      print(s.get_info())
39
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[22:30:30] serg :: serg-pc  → ~/tmp/oop»
python3 inheritance.py
Student John Dow, age - 23, ID - 1
============Course List============
Course - Differential equations, place - 321, time - Wen 11:00 AM
```

Наследование в Python.

```
36  class Teacher(Person):
37      def __init__(self, first_name, second_name, age, id, courses, email, phone):
38          super().__init__(first_name, second_name, age, id)
39          self.courses = courses
40          self.email = email
41          self.phone = phone
42
43      def get_info(self):
44          base_info = super().get_info()
45          info = f'''Teacher {base_info}, contacts:
46             mail - {self.email}
47             phone - {self.phone}
48  '''
49          info += "============Courses============\n"
50          for course in self.courses:
51              info += course.get_info() + "\n"
52          return info
53
```

Наследование в Python.

```
54
55  if __name__ == '__main__':
56      t = Teacher(
57          "John", "Dow", 47, 2, [
58              Course("Differential equations", 321, "Wen 11:00 AM"),
59              Course("Topology", "412a", "Mon 9:00 AM"),
60          ],
61          "john@dow.com",
62          "123545678"
63      )
64      print(t.get_info())
65
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[22:41:23] serg :: serg-pc  ➜  ~/tmp/oop»
python3 inheritance.py
Teacher John Dow, age - 47, ID - 2, contacts:
           mail - john@dow.com
           phone - 123545678
============Courses============
Course - Differential equations, place - 321, time - Wen 11:00 AM
Course - Topology, place - 412a, time - Mon 9:00 AM
```

Наследование в Python.

Python поддерживает множественное наследование, множественное наследование позволяет объединять в одном объекте свойства разных объектов.

Наследование в Python.

```python
class Worker:
    def __init__(self, sallary, exp="inf"):
        self.sallary = sallary
        self.exp = "inf"

    def is_expired_contract(self):
        if self.exp == "inf":
            return False
        elif datetime.now() > self.exp:
            return True
        return False


class Teacher(Person, Worker):
    def __init__(self, first_name, second_name, age, id, courses, email, phone, sallary):
        Person.__init__(self, first_name, second_name, age, id)
        Worker.__init__(self, sallary)
        self.courses = courses
        self.email = email
        self.phone = phone

    def get_info(self):
        base_info = super().get_info()
        info = f'''Teacher {base_info}, contacts:
           mail - {self.email}
           phone - {self.phone}
'''
        info += "============Courses============\n"
        for course in self.courses:
            info += course.get_info() + "\n"
        return info

```

Наследование в Python.

```
71
72  if __name__ == '__main__':
73      phd = Teacher(
74          "John", "Dow", 47, 2, [
75              Course("Differential equations", 321, "Wen 11:00 AM"),
76              Course("Topology", "412a", "Mon 9:00 AM"),
77          ],
78          "john@dow.com",
79          "123545678",
80          1000
81      )
82      print(phd.get_info())
83      print(phd.is_expired_contract())
84
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 1

```
[0:29:40] serg :: serg-pc  ➜  ~/tmp/oop»
python3 inheritance.py
Teacher John Dow, age - 47, ID - 2, contacts:
           mail - john@dow.com
           phone - 123545678
============Courses============
Course - Differential equations, place - 321, time - Wen 11:00 AM
Course - Topology, place - 412a, time - Mon 9:00 AM

False
```

Интерфейс

В программирование **интерфейсом** называется конструкция, которая задает поведение. Интерфейс не обладает полями/атрибутами и не реализует поведение. Он лишь указывает на то, какими те или иные классы должны обладать методами, чтобы имплементировать интерфейс. В Python нет возможности явно задавать интерфейсы, но неявно делать это возможно.

Интерфейс

```python
class IWorker:
    def work(self):
        raise NotImplementedError("This interface so it doesn't do any work")

    def get_break(self):
        raise NotImplementedError("This interface so it doesn't do any work")


class CEO(IWorker):
    def __init__(self, name):
        self.name = name

    def work(self):
        print(f"CEO {self.name} is doing some realy important work.")

    def get_break(self):
        print(f"CEO {self.name} tired so get rest.")

class Coder(IWorker):
    def __init__(self, name):
        self.name = name

    def work(self):
        print(f"Coder {self.name} is writing a lot of code.")

    def get_break(self):
        print(f"Coder {self.name} tired so get rest.")

```

Интерфейс/Абстрактный класс

Если говорить про нашу графическую библиотеку, то почти интерфейсом там будет класс Shape. Такой класс называется абстрактным, так как все же имеет некоторое общее для всех его наследников свойство или поведение

Утиная типизация

Принцип звучит сл. образом - если объект ходит, как утка, плавает, как утка, и крякает, как утка, значит, он утка - это возможно благодаря динамической типизации типов.

Утиная типизация

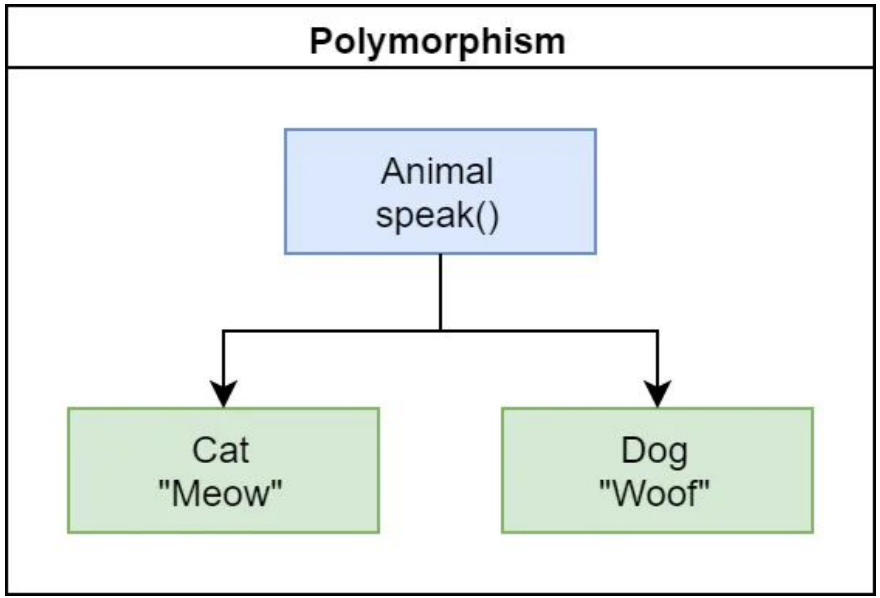
duck.py > ...

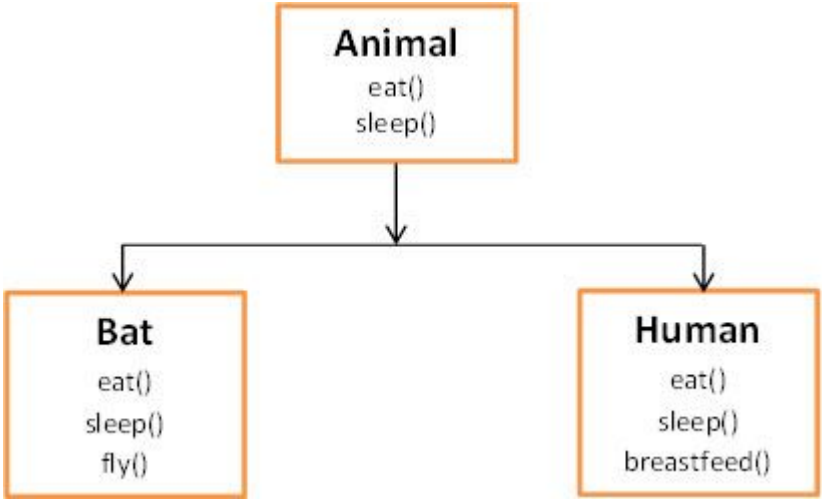
```python
class A:
    def work(self):
        print("Do some work by class A")


class B:
    def work(self):
        print("Do some work by class B")


def run_worker(worker):
    worker.work()


if __name__ == '__main__':
    a = A()
    b = B()
    run_worker(a)
    run_worker(b)

```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 1

```
[12:15:58] serg :: serg-pc  ➜  ~/tmp/oop»
python3 gp.py
25
[12:15:59] serg :: serg-pc  ➜  ~/tmp/oop»
python3 duck.py
Do some work by class A
Do some work by class B
```

Полиморфизм





Полиморфизм Python

gp.py > Shape > area

```python
class Shape:
    def area(self):
        raise NotImplementedError("Too abstract figure dont't know how to calculate area for it.")


class Circle(Shape):
    def __init__(self, r, x=0, y=0):
        self.center_x = x
        self.center_y = y
        self.radius = r

    def area(self):
        return math.pi * self.radius ** 2


if __name__ == '__main__':
    c = Circle(5)
    print(c.area())

```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 1

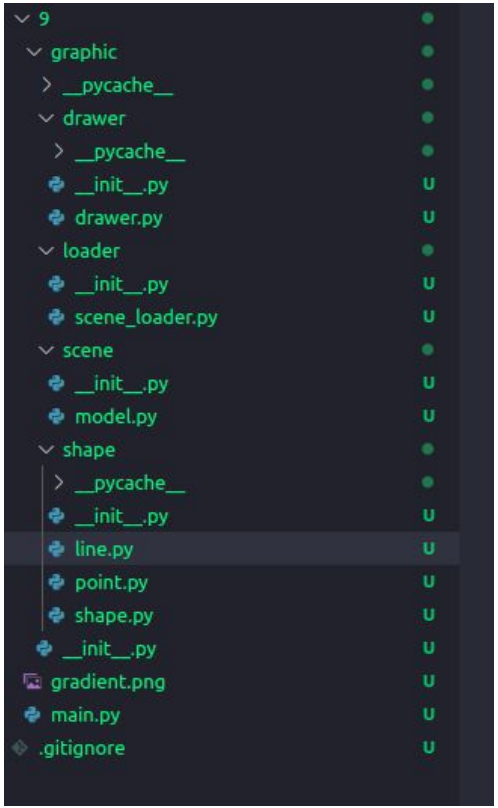
```
[11:49:56] serg :: serg-pc  ➜  ~/tmp/oop»
python3 gp.py
78.53981633974483
```

Полиморфизм

```
17
18
19  class Rectangle(Shape):
20      def __init__(self, h, w, x=0, y=0):
21          self.center_x = x
22          self.center_y = y
23          self.h = h
24          self.w = w
25
26      def area(self):
27          return self.h * self.w
28
29
30  if __name__ == '__main__':
31      r = Rectangle(5, 5)
32      print(r.area())
33
```

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS 1

[11:56:15] serg :: serg-pc  → ~/tmp/oop»
python3 gp.py
25
```

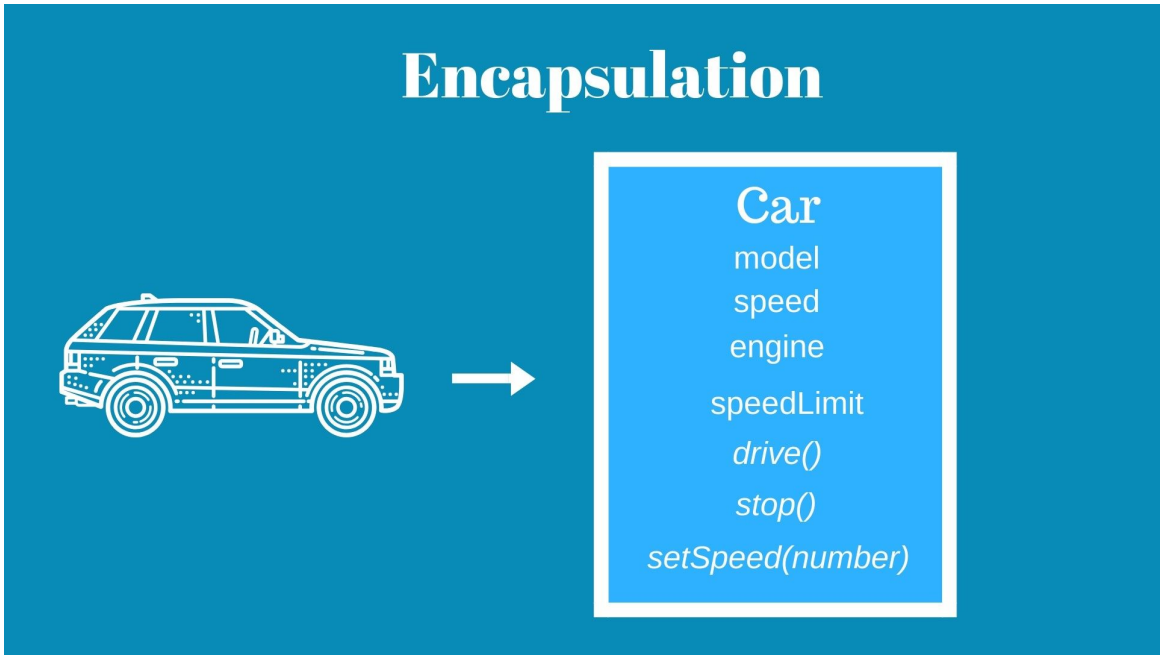
Полиморфизм Python




```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
```

Инкапсуляция

Свойства языка, позволяющее скрыть реализацию или данные от конечного пользователя.



Инкапсуляция

В Python инкапсуляция реализуется на уровне договоренности именования атрибутов/методов класса. Обычно выделяют три вида доступа:

1. public - доступен везде для всех
2. protected - доступен только для наследников класса
3. private - доступен только внутри класса

public


10 > inheritance > incapsulation.py > ...
1 class A:
2 def __init__(self, a):
3 self.public_atribute = a
4
5 if __name__ == '__main__':
6 a = A(1)
7 print(a.public_atribute)
8
TERMINAL
SQL CONSOLE
PROBLEMS
OUTPUT
DEBUG CONSOLE
(.venv) serg@matrix ~/git/github/python_course/code/src/10/inherita
python incapsulation.py
1

private/protected

10 > inheritance > incapsulation.py > ...

```python
class A:
    def __init__(self, a, b, c):
        self.public_atribute = a
        self._protected_field = b
        self.__private_field = c

if __name__ == '__main__':
    a = A(1, 2, 3)
    print(f"{a.public_atribute=} {a._protected_field=} {a.__private_field=}")
```

TERMINAL SQL CONSOLE PROBLEMS OUTPUT DEBUG CONSOLE

```
(.venv) serg@matrix ~/git/github/python_course/code/src/10/inheritance <master*>
python incapsulation.py
Traceback (most recent call last):
  File "incapsulation.py", line 9, in <module>
    print(f"{a.public_atribute=} {a._protected_field=} {a.__private_field=}")
AttributeError: 'A' object has no attribute '__private_field'
```

Инкапсуляция

incapsultion.py > ...

```python
import math


class Circle:
    def __init__(self, radius, center_x=0, center_y=0):
        self.radius = radius
        self.center_x = center_x
        self.center_y = center_y

    def area(self):
        return math.pi * self.radius ** 2


if __name__ == '__main__':
    c = Circle(10)
    print(c.area())

```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[13:40:40] serg :: serg-pc  → ~/tmp/oop»
python3 incapsultion.py
314.1592653589793
[13:43:54] serg :: serg-pc  → ~/tmp/oop»
```

Инкапсуляция

```
...
# Где-то в программе поменяли радиус изначальной окружность
# при этом это сильно влияет на результаты работы
# так как мы не предполагаем, что это свойстро должно меняться.
c.radius = 20
# Кроме того, безконтрольный доступ к полям, может привести к
# аварийным ситуациям.
c.radius = "incorrect radius" # После этого метод area() перестанет работать
c.radius = -1 # После этого метод area() будет возвращать невероятные результаты
...
```

Инкапсуляция

Реализации в Python.
Во многих языках программирования.
Ограничение доступа к полям и методам, организовывается с помощью ключевых слов (public/private). В Python для этого - существует соглашение об именовании.

incapsultion.py > ...

```python
import math


class Circle:
    def __init__(self, radius, center_x=0, center_y=0):
        self.__radius = radius
        self.__center_x = center_x
        self.__center_y = center_y

    def area(self):
        return math.pi * self.__radius ** 2


if __name__ == '__main__':
    c = Circle(10)
    print(c.area())
    print(c.__radius)

```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[13:54:22] serg :: serg-pc  ➜  ~/tmp/oop»
python3 incapsultion.py
314.1592653589793
Traceback (most recent call last):
  File "incapsultion.py", line 17, in <module>
    print(c.__radius)
AttributeError: 'Circle' object has no attribute '__radius'
```

Инкапсуляция

```python
class Circle:
    def __init__(self, radius, center_x=0, center_y=0):
        self.__radius = radius
        self.__center_x = center_x
        self.__center_y = center_y

    def area(self):
        return math.pi * self.__radius ** 2

    def on_circle(self, fig):
        for vertx_x, vertex_y in fig:
            if self.__on(vertx_x, vertex_y) is False:
                return False
        return True

    def __on(self, point_x, point_y):
        if self.__dist(point_x, point_y) == self.__radius:
            return True
        return False

    def __dist(self, point_x, point_y):
        return math.sqrt((self.__center_x - point_x)**2 + (self.__center_y - point_y)**2)
```

На сегодня все!