Основы объектно-ориентированного программирования - 2

Технологии и языки программирования

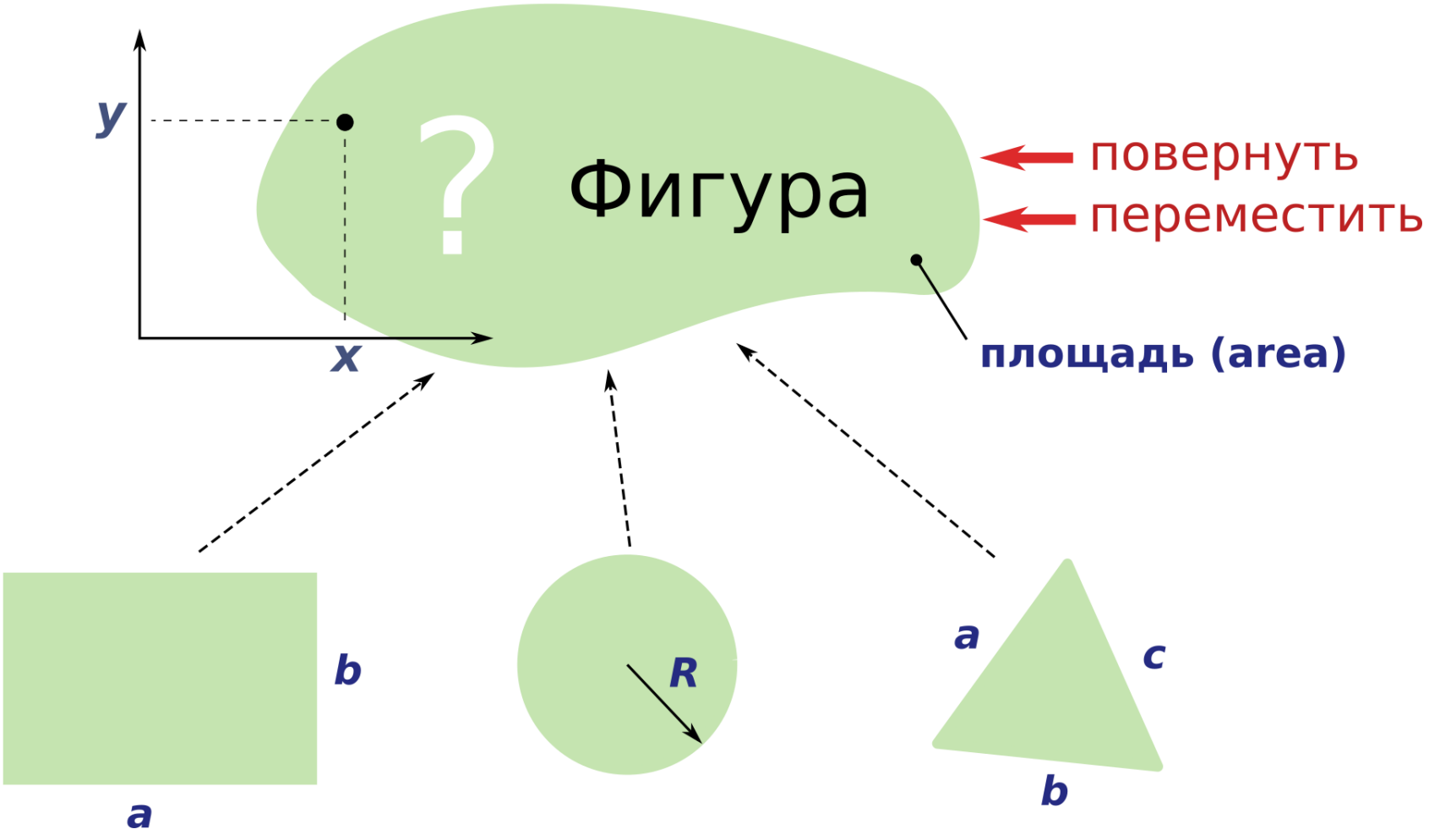
Юдинцев В. В.

Кафедра теоретической механики
Самарский университет

17 марта 2018 г.

Класс = данные + методы

Класс – тип данных, состоящий из набора атрибутов (свойств) и методов – функций для работы с этими атрибутами.



Объявление класса в Python

Точка с координатами *x* и *y* (атрибуты).

```
class Point:
    # Конструктор
    def __init__(self, coordinates):
        self.x = coordinates[0]
        self.y = coordinates[1]
    # Переместить точку
    def move(self, delta):
        self.x = self.x + delta[0]
        self.y = self.y + delta[1]
```

Метод move перемещает точку на заданные расстояния вдоль осей *x* и *y*.

Конструктор

Метод класса, вызываемый при создании объекта – представителя класса.

```
1 class Point:
2     # Конструктор
3     def __init__(self, coordinates):
4         self.x = coordinates[0]
5         self.y = coordinates[1]
```

self – это ссылка на создаваемый в памяти компьютера объект.

```
6 p1 = Point([1, 3])
7 print(p1.x, p1.y)
```

1 3

При создании объекта (строка 6) вызывается конструктор __init__ (строка 3), в который передается ссылка на создаваемый объект (p1) и список [1, 3].

Вызов метода класса

```
class Point:
    # Конструктор
    def __init__(self, coordinates):
        self.x = coordinates[0]
        self.y = coordinates[1]
    # Переместить точку
    def move(self, delta):
        self.x = self.x + delta[0]
        self.y = self.y + delta[1]
```

```
p1 = Point([1, 3])

p1.move([2, 3])
print(p1.x, p1.y)
```

3 6

Инкапсуляция

Ограничение доступа к полям класса

Разрешение прямого доступа к свойствам объекта может нарушать его целостность

```
class Circle:
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r
        self.area = math.pi*r*r
```

```
circle = Circle(0, 0, 1)
print(circle.area)
```

```
3.14159265358979
```

```
circle.r = 10
print(circle.area)
```

```
3.14159265358979
```

Скрытые свойства класса

Атрибуты, объявленные с одним подчёркиванием не предназначены для использования вне класса

```
class Circle:
    def __init__(self, radius):
        self._radius = radius
        self._area = math.pi*radius**2
```

Это просто соглашение. Атрибут доступен вне класса:

```
c1 = Circle(10.0)
print(c1._radius)
```

10.0

```
c1._radius = 5
```

Скрытые свойства класса

Свойства, объявленные с двойным подчёркиванием перед именем недоступны вне класса:

```
1 class Circle:
2     def __init__(self, radius):
3         self.__radius = radius
4         self.__area = math.pi*radius**2
```

```
c1 = Circle(10.0)
print(c1.__radius)
```

AttributeError: 'Circle' object has no attribute '__radius'

Скрытые свойства класса

Свойства, объявленные с двойным подчёркиванием перед именем недоступны вне класса:

```
1 class Circle:
2     def __init__(self, radius):
3         self.__radius = radius
4         self.__area = math.pi*radius**2
```

```
c1 = Circle(10.0)
print(c1.__radius)
```

AttributeError: 'Circle' object has no attribute '__radius'

Если очень хочется:

```
print(c1._Circle__radius)
```

Доступ при помощи методов

```
class Circle:
  def __init__(self, radius):
    self.__radius = radius
    self.__area = math.pi*radius**2
  # Поучить значение радиуса
  def get_radius(self):
    return self.__radius
  # Установить значение радиуса и площади
  def set_radius(self, radius):
    self.__radius = radius
    self.__area   = math.pi*radius**2
  # Получить значение площади
  def get_area(self):
    return self.__area
```

Доступ при помощи методов

```
c = Circle(10.0)

print(c.get_area())
```

314.1592653589793

```
c.set_radius(5)

print(c.get_area())
```

78.53981633974483

Слишком много скобок...

Упрощаем: синтаксис property

```
1  class Circle:
2    def __init__(self, radius):
3      self.__radius = radius
4      self.__area = math.pi*radius**2
5
6    def get_radius(self):
7      return self.__radius
8    def set_radius(self, radius):
9      self.__radius = radius
10     self.__area   = math.pi*radius**2
11
12   def get_area(self):
13     return self.__area
14
15   radius = property(get_radius, set_radius)
16   area   = property(get_area)
```

Синтаксис property

```
c = Circle(10.0)
```

Вызывается get_radius

```
print(c.area)
```

314.1592653589793

Вызывается set_radius и get_radius

```
c.radius = 5
print(c.area)
```

78.53981633974483

```
c.area = 0
```

AttributeError: can’t set attribute

Синтаксис @property

```
class Circle:
    def __init__(self, radius):
        self.__radius = radius
        self.__area = math.pi*radius**2
    @property
    def area(self):
        return self.__area
    @property
    def radius(self):
        return self.__radius
    @radius.setter
    def radius(self, radius):
        self.__radius = radius
        self.__area   = math.pi*radius**2
```

Синтаксис @property

```
c = Circle(10.0)
```

Вызывается radius(self)

```
print(c.area)
```

314.1592653589793

Вызывается radius(self, radius) и radius(self)

```
c.radius = 5
print(c.area)
```

78.53981633974483

```
c.area = 0
```

AttributeError: can’t set attribute

Вычисляемые атрибуты (свойства)

```
1  class Circle:
2      def __init__(self, radius):
3          self.__radius = radius
4
5      @property
6      def area(self):
7          return math.pi*self.__radius**2
8
9      @property
10     def radius(self):
11         return self.__radius
12
13     @radius.setter
14     def radius(self, radius):
15         self.__radius = radius
```

Синтаксис property

```
c = Circle(10.0)
```

Вызывается area(self)

```
print(c.area)
```

314.1592653589793

Вызывается radius(self, radius) и area(self)

```
c.radius = 5
print(c.area)
```

78.53981633974483

Переопределение атрибутов

```
1  import math
2
3  class Shape:
4
5      def __init__(self, x, y):
6          self.x = x
7          self.y = y
8
9      def move(self, dx, dy):
10         self.x +=  dx
11         self.y +=  dy
12
13     @property
14     def area(self):
15         raise NotImplementedError()
```

Rectangle

```
class Rectangle(Shape):

    def __init__(self, x, y, a, b):
        super().__init__(x, y)
        self.a = a
        self.b = b

    @property
    def area(self):
        return self.a*self.b
```

Circle

```
class Circle(Shape):

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

    @property
    def area(self):
        return math.pi*self.r**2
```

Фигуры

```
figures = []

figures.append( Rectangle(0,0,2,4) )
figures.append( Rectangle(1,15,4,2) )

figures.append( Circle(5,10,5) )
figures.append( Circle(6,7,3) )

sum(fig.area for fig in figures)
```

122.81415022205297

Полиморфизм

Перегрузка операторов

Перегрузка операторов — один из способов реализации полиморфизма, когда различные операторы (+, -, /, ...) имеют различный смысл в зависимости от типов аргументов.

- 1 + 3 = 4
  результат сложения двух целых чисел – их арифметическая сумма
- “1” + “3” = “13”
  результат сложения двух строк – конкатенация (склейка) строк

Python позволяет определять правила выполнения операций для своих типов (классов).

Перегрузка операторов

Класс list2 создаётся на основе класса list, переопределяя операцию сложения:

```
class list2(list):

  def __add__(self, other):
    return list2([i[0] + i[1] for i in \
                        zip(self,other)])
```

```
a = list2([1, 2, 3])
b = list ([7, 4, 1])
c = a + b
print(c)
```

[8, 6, 4]

Перегрузка операторов

Класс list2 создаётся на основе класса list, переопределяя операцию сложения и вычитания:

```
class list2(list):
  def __add__(self, other):
    return list2([i[0] + i[1] for i in \
                  zip(self,other)])
  def __sub__(self, other):
    return list2([i[0] - i[1] for i in \
                  zip(self,other)])
```

```
a = list2([1, 2, 3])
b = list ([7, 4, 1])
c = a - b
print(c)
```

[-6, -2, 2]

Перегрузка операторов

Перегрузка оператора сравнения (равенство):

```
class Circle:
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r
    def __eq__(self, other):
        return self.r == other.r
```

Две окружности равны, если равны их радиусы

```
o1 = Circle(0, 0, 2)
o2 = Circle(1, 0, 2)
o1 == o2
```

True

Перегрузка операторов

Если операция сравнения для объектов типа Circle не определена:

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

Окружности не равны:

```
o1 = Circle(1, 0, 2)
o2 = Circle(1, 0, 2)
o1 == o2
```

False

Перегрузка преобразования в текст

```
class Circle:
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r
    def __str__(self):
        return "Окружность радиуса "+str(self.r)
```

```
o1 = Circle(0, 0, 2)
print(o1)
```

Окружность радиуса 2

Перегрузка операторов сравнения

```
<       __lt__(self, other)
<=      __le__(self, other)
==      __eq__(self, other)
!=      __ne__(self, other)
>=      __ge__(self, other)
>       __gt__(self, other)
```

Перегрузка арифметических операторов

```
+       __add__(self, other)
-       __sub__(self, other)
*       __mul__(self, other)
//      __floordiv__(self, other)
/       __truediv__(self, other)
%       __mod__(self, other)
**      __pow__(self, other)
```

Классы, модули, структуры?

Советы от Гвидо ван Россума:

- Избегайте усложнения структур данных.
- Кортежи лучше объектов (можно воспользоваться именованными кортежами).
- Предпочитайте простые поля функциям, геттерам и сеттерам.
- Используйте больше чисел, строк, кортежей, списков, множеств, словарей.
- Взгляните также на библиотеку collections, особенно на класс deque.

Именованные кортежи

Если необходима структура данных со свойствами (атрибуами, полями) без сложного поведения (вычисляемые поля), необходимости наследования, то лучше использовать более простые типы, например namedtuple:

```
1 import collections
2
3 Book = collections.namedtuple('Book',
4         ['name', 'cost', 'pages'])
5
6 book = Book('Python cookbook', 25.0, 500)
```

```
print(book.pages)
```

```
500
```

http://valsorym.org/articles/python-3-namedtuple/

Структура

Эмуляция структуры в стиле языка Си:

```
class DataStructure:
    pass
```

```
a = DataStructure()

a.x = 1
a.y = 2

print(a.x)
```

```
1
```

Структура

Создание структуры с функцией инициализации:

```
class Struct:
  def __init__(self, **kwargs):
    for k, v in kwargs.items():
      setattr(self, k, v)

class MyStruct(Struct):
  pass
```

```
ms = MyStruct(foo = 10, bar = "abc")
print(ms.foo)
```

10

https://utcc.utoronto.ca/~cks/space/blog/python/EmulatingStructsInPython