

ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

КАФЕДРА АВТОМАТИЗАЦИЯ ПРОИЗВОДСТВЕННЫХ ПРОЦЕССОВ

ЛЕКЦИЯ №6

Объектно-ориентированное
программирование в Python.
Абстрактные классы, интерфейсы и
функции обратного вызова

СОСТАВИТЕЛЬ: КАНД. ТЕХН. НАУК БЫКАДОР В.С.

Абстрактные классы

Абстрактные классы предлагают способ формализации интерфейса у своих наследников, которые в отличии от абстрактных классов называются – конкретными классами.

Нельзя создавать экземпляры абстрактного класса, от абстрактного класса можно только наследовать конкретные классы и создавать экземпляры именно унаследованного конкретного класса.

Для чего вообще нужны абстрактные классы?

1. Для формализации и стандартизации интерфейса однородных классов.

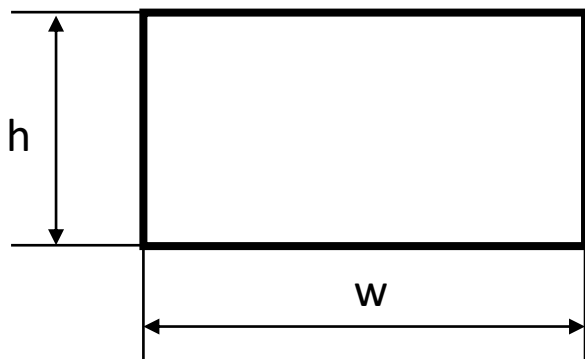
Чтобы каждый класс, который относится к семейству себе подобных, имел такие же методы и свойства (вплоть до названия), что и другие классы семейства.

2. Контроль реализации методов и свойств в каждом конкретном классе – потомке абстрактного класса.

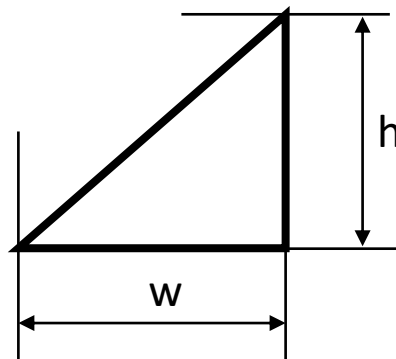
N.B!: Так как абстрактный класс является родительским, то в него можно вынести любые общие для всех потомков свойства и методы.

Рассмотрим пример

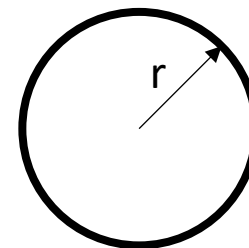
Геометрические фигуры



$$S = w \cdot h$$

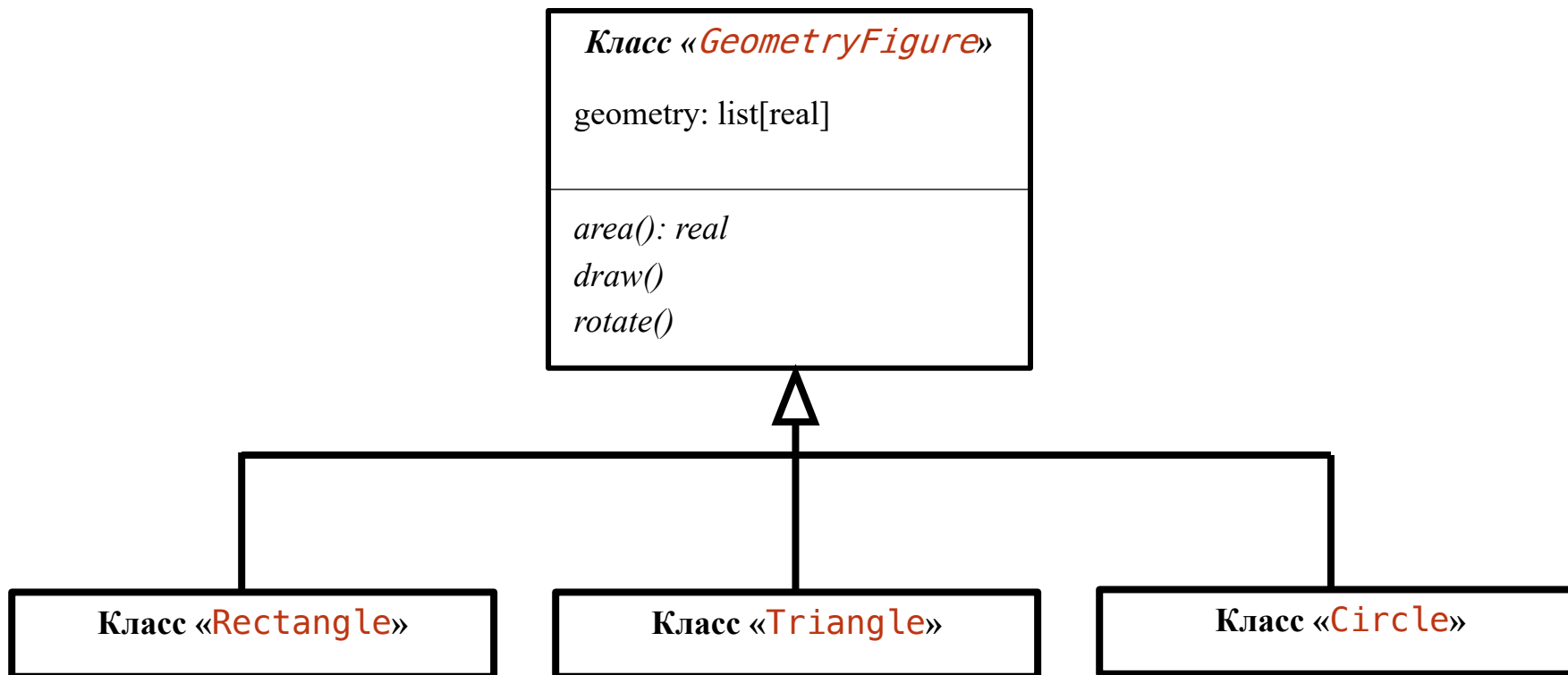


$$S = \frac{1}{2} w \cdot h$$



$$S = \pi r^2$$

Диаграмма классов



Вначале рассмотрим
родительский класс, но
неабстрактный

```
2 class GeometryFigure:
3     def __init__(self, lst_geometry_charact):
4         self._geometry = lst_geometry_charact
5         for item in self._geometry:
6             if item < 0:
7                 raise('Одна из геометрических хар-к меньше нуля!')
8         self._angle = 0
9
10    def area(self):
11        pass
12
13    def draw(self):
14        pass
15
16    def rotate(self):
17        pass
18
```

НЕТ
РЕАЛИЗАЦИИ
МЕТОДОВ

Наследуем потомков от
неабстрактного
родителя

```
20 class Rectangle(GeometryFigure):
21     | pass
22
23 class Triangle(GeometryFigure):
24     | pass
25
26 class Circle(GeometryFigure):
27     | pass
28
```

Создаем экземпляры
родителя, потомков и
вызываем методы...

```
41 rec = Rectangle([4, 8])
42 rec.draw()
43
44 tr = Triangle([2, 4])
45
46 cl = Circle([5])
```

Ошибок нет, но и результата
работы тоже нет :((

```
• (venv) vityalybykador@Air-Vitaly test % /Users/
  CTS/test/for_lectures.py
○ (venv) vityalybykador@Air-Vitaly test %
```

Теперь рассмотрим родительский класс, но **абстрактный!**

```
1 from abc import ABC, abstractmethod
2
3 class GeometryFigure(ABC):
4     def __init__(self, lst_geometry_charact):
5         self._geometry = lst_geometry_charact
6         for item in self._geometry:
7             if item < 0:
8                 raise('Одна из геометрических хар-к меньше нуля!')
9         self._angle = 0
10
11     def _icrememnt_angle(self):
12         self._angle += 90 if self._angle < 360 else 0
13
14     @abstractmethod
15     def area(self):
16         pass
17
18     @abstractmethod
19     def draw(self):
20         pass
21
22     @abstractmethod
23     def rotate(self):
24         pass
25
```

1) Используем специальную библиотеку **abc** для реализации абстрактного класса.

2) Наследуем наш класс от Стандартного абстрактного класса **ABC**.

3) Обязательно делаем, то что должно быть реализовано в потомках (в данном случае это методы), тоже абстрактными.

В языке **Python** для этого используется декоратор **@abstractmethod**.

Посмотрим, что нам дало использование абстрактного класса в качестве родительского

```
33  
34 geometry = GeometryFigure([2, 4])  
35
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИНАЛ

JUPYTER

```
⊗ (venv) vitalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /Users/vitalybykador/PROJECTS/test/for_lectures.py  
Traceback (most recent call last):  
  File "/Users/vitalybykador/PROJECTS/test/for_lectures.py", line 34, in <module>  
    geometry = GeometryFigure([2, 4])  
TypeError: Can't instantiate abstract class GeometryFigure with abstract methods area, draw, rotate  
○ (venv) vitalybykador@Air-Vitaly test %
```

Нельзя создать экземпляр абстрактного класса, то того класса, который описывает общий интерфейс и некоторые общие характеристики семейства классов.

Нельзя создать экземпляр конкретного класса, унаследованного от абстрактного класса, пока в конкретном классе не будут реализованы все абстрактные методы и свойства.

```
24 class Rectangle(GeometryFigure):
25     | pass
26
27 class Triangle(GeometryFigure):
28     | pass
29
30 class Circle(GeometryFigure):
31     | pass
32
33
34 rec = Rectangle([4, 8])
35 rec.draw()
36
37 tr = Triangle([2, 4])
38
39 cl = Circle([5])
40
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИНАЛ

JUPYTER

CTS/test/for_lectures.py

Traceback (most recent call last):

File "/Users/vitalybykador/PROJECTS/test/for_lectures.py", line 34, in <module>

rec = Rectangle([4, 8])

TypeError: Can't instantiate abstract class Rectangle with abstract methods area, draw, rotate

(venv) vitalybykador@Air-Vitaly test %

Строка 40, столбец 1 Пробелов: 4 UTF-8 LF { Python

```

27 class Rectangle(GeometryFigure):
28     def area(self):
29         return self._geometry[0] * self._geometry[1]
30
31     def draw(self):
32         print('Отрисовка прямоугольника')
33
34     def rotate(self):
35         self._icrememnt_angle()
36         print(f'Поворот прямоугольника на угол {self._angle}')
37
38 class Triangle(GeometryFigure):
39     pass
40
41 class Circle(GeometryFigure):
42     pass
43
44
45 rec = Rectangle([4, 8])
46 rec.draw()
47 rec.rotate()
48 rec.rotate()
49

```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ JUPYTER

```

(venv) vitalitybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python
s.py
Отрисовка прямоугольника
Поворот прямоугольника на угол 90
Поворот прямоугольника на угол 180
(venv) vitalitybykador@Air-Vitaly test % █

```

Интерфейсы

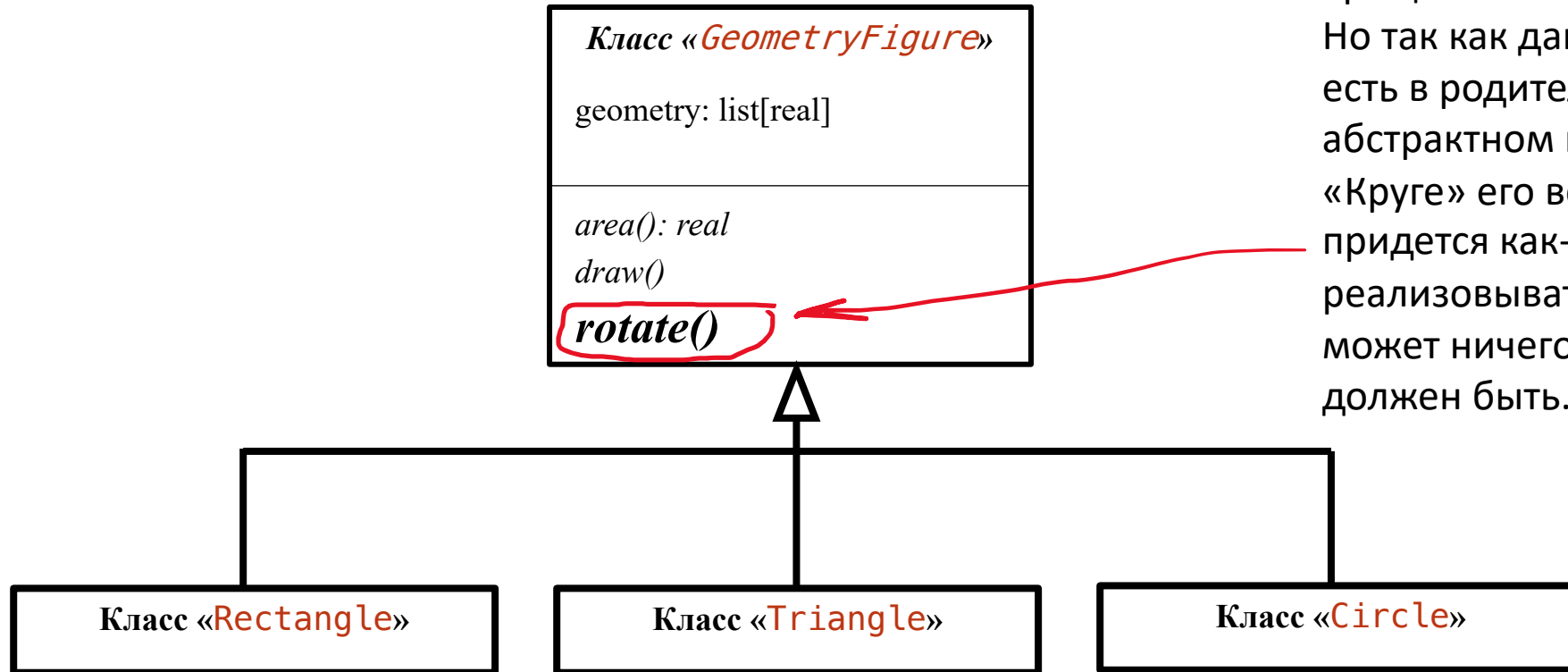
Так как язык программирования **Python** поддерживает множественное наследование и концепцию абстрактных классов, то как таковых интерфейсов в **Python** нет. Интерфейсы используются в тех языках где множественное наследование не поддерживается, например, в языке программирования **C#**.

Тем не менее концепция интерфейсов может быть полезна и её можно применять и при проектировании программ на языке программирования **Python**.

Существует сторонняя библиотека для реализации интерфейсов в **Python**. Это библиотека **zope.interface**. Но мы рассмотрим реализацию интерфейсов стандартными средствами **Python**.

Существует несколько подходов для реализации интерфейсов стандартными средствами **Python**, изучим тот, который кажется наиболее подходящим с точки зрения автора данного курса.

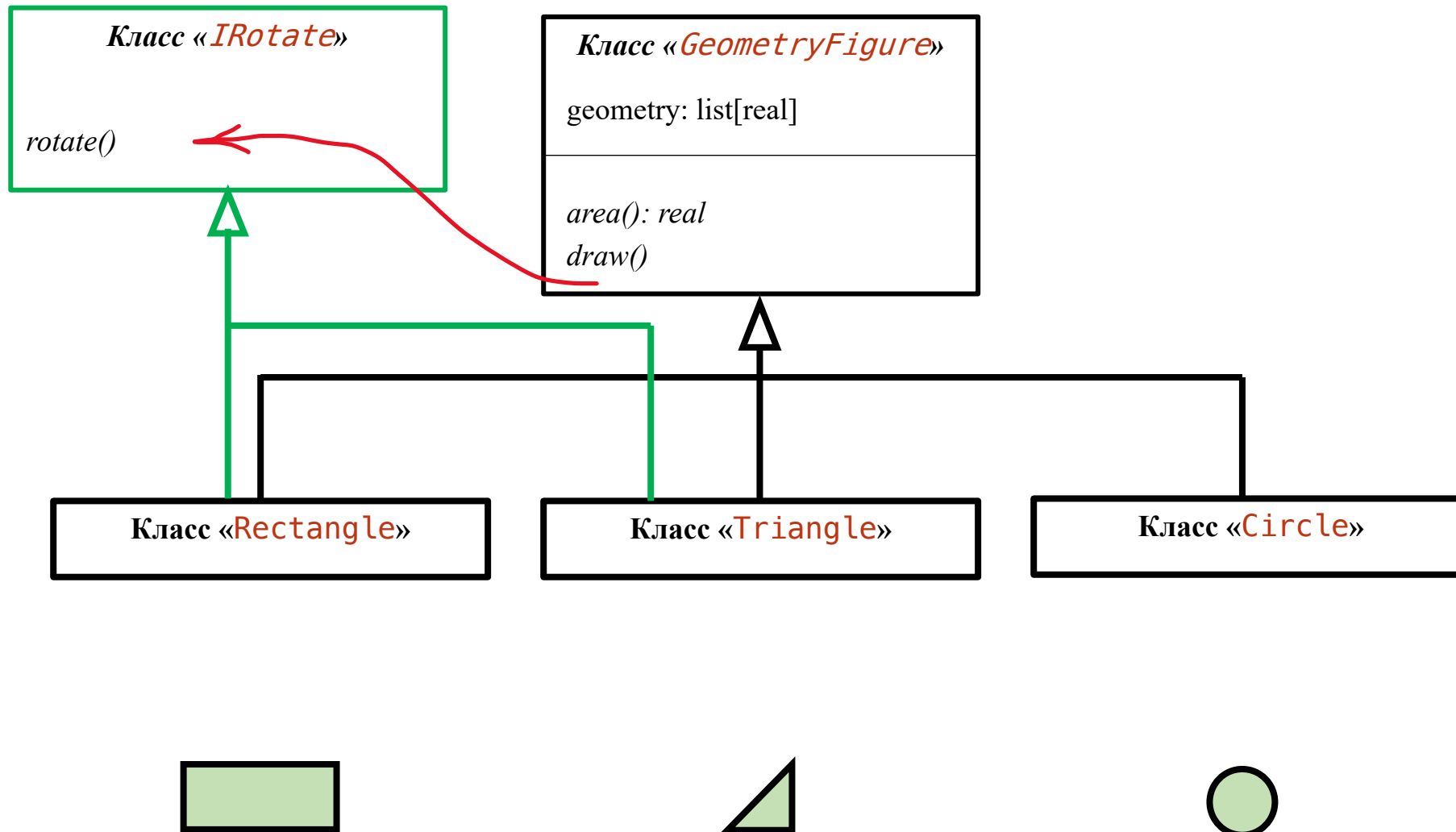
Рассмотрим проблему на примере



Для класса «*Circle*» вращение не нужно. Но так как данный метод есть в родительском абстрактном классе, то в «Круге» его всё равно придется как-то реализовывать, он даже может ничего не делать, но должен быть.



Решим проблемы через интерфейсы



Реализация в коде

```
1  from abc import ABC, abstractmethod
2
3  class GeometryFigure(ABC):
4      def __init__(self, lst_geometry_charact):
5          self._geometry = lst_geometry_charact
6          for item in self._geometry:
7              if item < 0:
8                  raise('Одна из геометрических хар-к меньше нуля!')
9          self._angle = 0
10
11     def _icrememnt_angle(self):
12         self._angle += 90 if self._angle < 360 else 0
13
14     @abstractmethod
15     def area(self):
16         pass
17
18     @abstractmethod
19     def draw(self):
20         pass
21
22     class IRotate(ABC):
23         @abstractmethod
24         def rotate(self):
25             pass
```



Реализация в коде

```
27 class Rectangle(GeometryFigure, IRotate):
28     pass
29
30 class Circle(GeometryFigure):
31     pass
32
33 rec = Rectangle([4, 8])
34 cl = Circle([5])
35
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИНАЛ

JUPYTER

```
⊗ (venv) vityalybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/python /User
2.py
Traceback (most recent call last):
  File "/Users/vitalybykador/PROJECTS/test/for_lectures2.py", line 33, in <module>
    rec = Rectangle([4, 8])
TypeError: Can't instantiate abstract class Rectangle with abstract methods area, draw, rotate
○ (venv) vityalybykador@Air-Vitaly test %
```


Реализация в коде

```
27 class Rectangle(GeometryFigure, IRotate):
28     def area(self):
29         pass
30
31     def draw(self):
32         pass
33
34     def rotate(self):
35         pass
36
37 class Circle(GeometryFigure):
38     pass
39
40 rec = Rectangle([4, 8])
41 cl = Circle([5])
42
```

ПРОБЛЕМЫ

ВЫХОДНЫЕ ДАННЫЕ

КОНСОЛЬ ОТЛАДКИ

ТЕРМИНАЛ

JUPYTER


```
⊗ (venv) vitalitybykador@Air-Vitaly test % /Users/vitalybykador/PROJECTS/test/venv/bin/pyt
2.py
Traceback (most recent call last):
  File "/Users/vitalybykador/PROJECTS/test/for_lectures2.py", line 41, in <module>
    cl = Circle([5])
TypeError: Can't instantiate abstract class Circle with abstract methods area, draw
○ (venv) vitalitybykador@Air-Vitaly test %
```

Функции обратного вызова (callback-функции)

При реализации программ в рамках парадигмы ООП часто используют событийную модель обмена сообщениями между объектами программы. Для реализации событийной модели используются так называемые функции обратного вызова или callback-функции.

По своей сути функция обратного вызова это функция, которая переданная в другую функцию в качестве аргумента (или метод класса), затем функция обратного вызова вызывается там куда она была передана.

Функции обратного вызова (callback-функции)



Функция обратного вызова

Определяет, что будет выполнено, когда её вызовут.

Функция которая вызывает callback-функцию

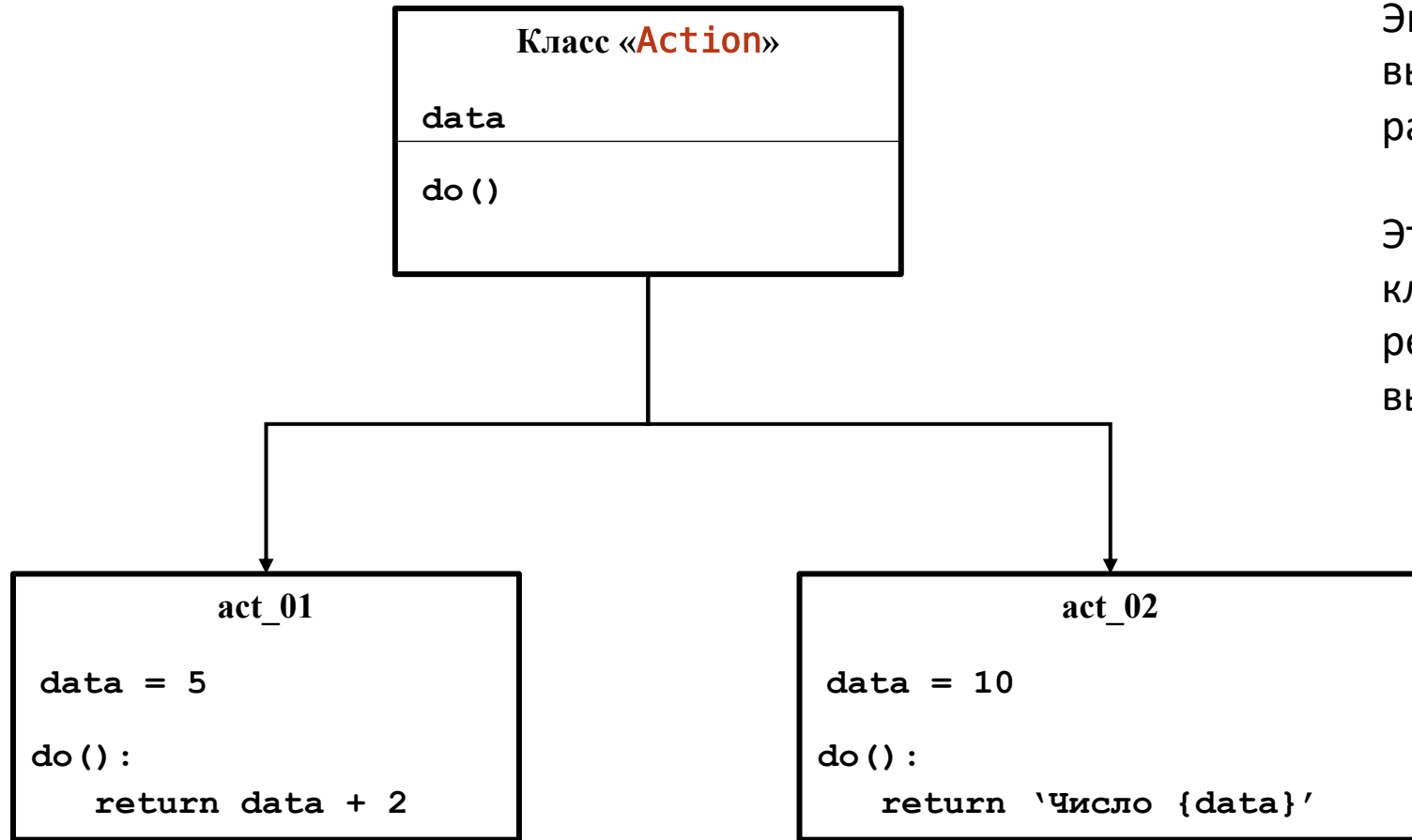
Определяет когда будет вызвана функция обратного вызова.

Тот кто пишет класс **Button** (кнопка), определяет отрисовку, расположение в окне программы, пишет код нажатия на кнопку.

Но, программист пишущий код класса **Button** не знает как его кнопка будет использоваться в конкретной программе. В одной программе при нажатии на кнопку **Ok** будет закрываться окно, в другой – запускаться расчеты, в третьей – выводится сообщение пользователю...

Поэтому программист, который пишет класс **Button** отдает реализацию действия по нажатию кнопки тому программисту, который будет этот класс использовать в конкретной программе. И взаимодействие второго программиста с классом **Button** реализуется через функцию обратного вызова.

Рассмотрим пример



Экземпляры класса «**Action**», которые при вызове метода **do()** должны делать разные действия.

Эти действия на этапе проектирования класса неизвестны, поэтому будут реализованы через функции обратного вызова в конкретной программе.

В программном коде

Функции обратного
вызова.

```
class Action:
    def __init__(self, data):
        self.__data = data

    def do(self, callback):
        return callback(self.__data)
```

Класс «Action».

```
def fnc_callback_01(data):
    return data + 2

def fnc_callback_02(data):
    return f'Число {data}'
```

```
act_01 = Action(5)
res = act_01.do(fnc_callback_01)
print(res)

print('')

act_02 = Action(10)
res = act_02.do(fnc_callback_02)
print(res)
```

Передача функций обратного
вызова в метод do() экземпляров
класса «Action».

```
1 class Action:
2     def __init__(self, data):
3         self.__data = data
4
5     def do(self, callback):
6         return callback(self.__data)
7
8
9
10 def fnc_callback_01(data):
11     return data + 2
12
13 def fnc_callback_02(data):
14     return f'Число {data}'
15
16
17 act_01 = Action(5)
18 res = act_01.do(fnc_callback_01)
19 print(res)
20
21 print('')
22
23 act_02 = Action(10)
24 res = act_02.do(fnc_callback_02)
25 print(res)
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

7

Число 10

○ (venv) vityalybykador@Air-Vitaly test %