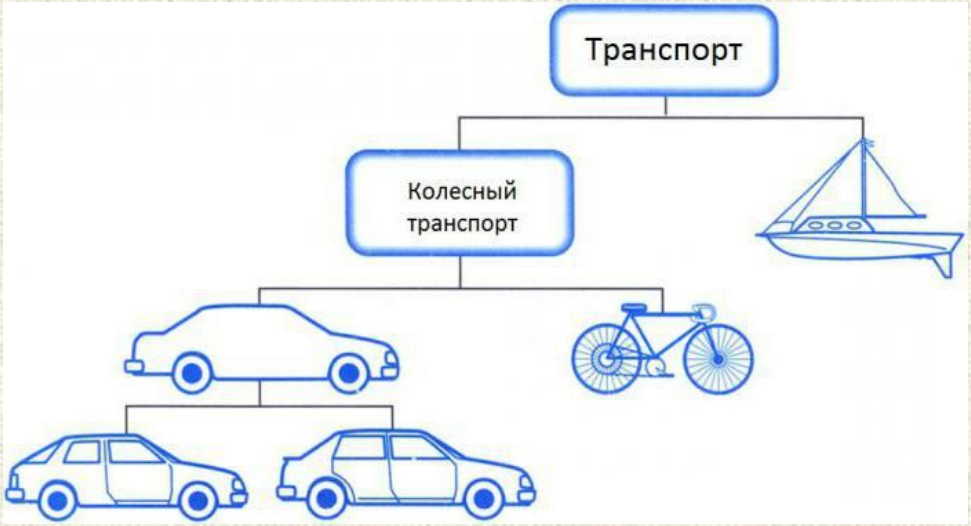
Наследование

Наследование

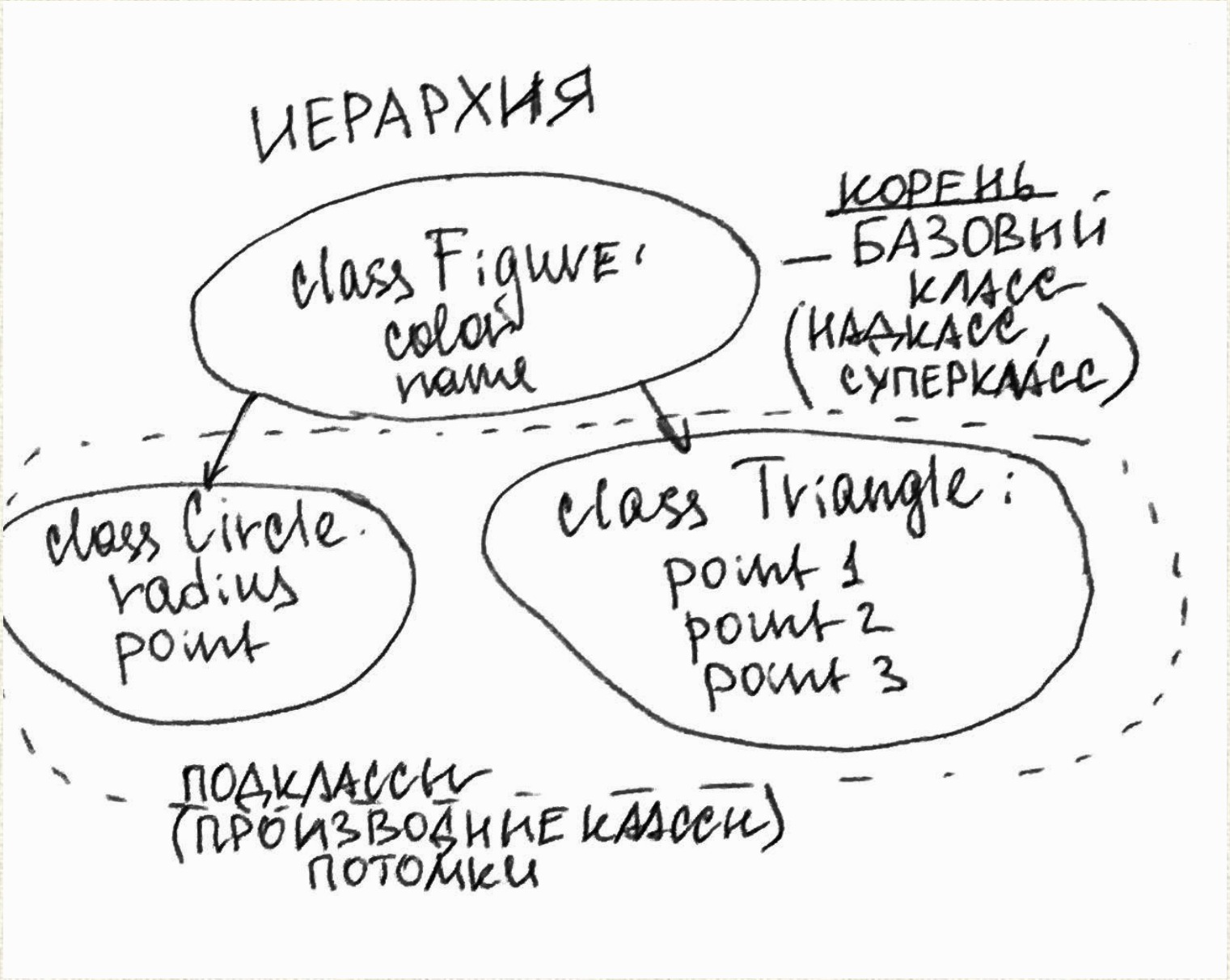
Наследование – возможность передачи атрибутов и методов одного класса (родительского) другим (дочерним).

```
class Родитель:
    Атрибуты
    Методы()

class Дочерний(Родитель):
    ...
```



Дочерние классы могут дополнять или изменять работу родительских классов.

Наследование



Наследование

```
class Parent:  # родитель
   a = 42

class Child(Parent):  # потомок
   b = 256

parent = Parent()  # создаем экземпляр базового класса
print(parent.a)  # обращаемся к атрибуту экземпляра базового класса

child = Child()  # создаем экземпляр производного класса
print(child.a) # обращаемся к атрибуту экземпляра производного класса
print(child.b) # обращаемся к атрибуту экземпляра производного класса

print(dir(parent)) # ['__class__', ..., '__init__', ..., 'a']
print(dir(child))  # ['__class__', ..., '__init__', ..., 'a', 'b']

# переопределим наследованный атрибут
child.a = "Hello"
print(parent.a)
print(child.a)
```

Наследование

Переопределение методов – возможность дочернего класса создать свою реализацию метода, определенного в родительском классе.

При этом доступ к объектам базового класса можно получить при помощи функции super().

Наследование

При полном переопределении метода достаточно просто создать в дочернем классе метод с таким же названием, что и в родительском классе и записать в него нужные команды.

Для расширения возможностей метода родительского класса нужно вызвать его работу в методе дочернего класса.

Это можно сделать обращением через класс:

```
class Child(Parent):
    …
    def method(self, x):
        Parent.method(self)
        self.x=x
```

Но если изменится имя родительского класса или иерархия наследования, то изменения будут работать некорректно. Поэтому для обращения к предыдущему родителю логичнее использовать функцию super()

Наследование

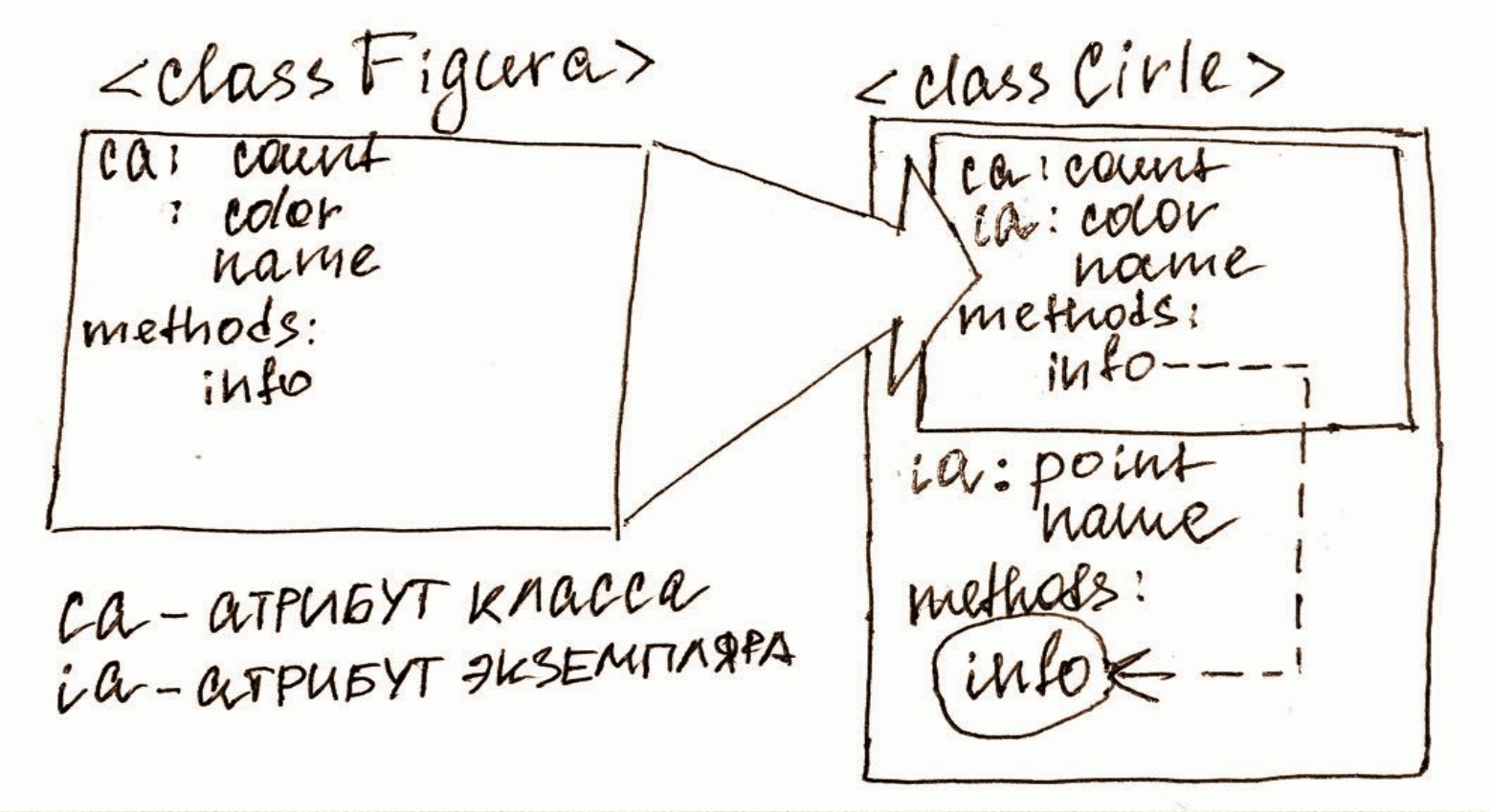
```
class Figure:
    count = 0

    def __init__(self, name, color):
        self.color = color
        self.name = name
        Figure.count += 1

    def print_info(self):
        print('Имя: ', self.name,
              '\nЦвет: ', self.color,
              '\nВсего фигур: ', Figure.count)
```

Наследование

```
class Circle(Figure):
   def __init__(self, name, color, point, radius):
# В дочернем классе необходимо вызвать метод инициализации родителя.
      super().__init__(name, color)
      # инициализируем поля объекта класса потомка
      self.point = point
      self.radius = radius

   def print_info(self):
      # переопределяем метод базового (родительского) класса
      # вызов метода базового класса
      super().print_info()
      print('\nРадиус: ', self.radius,
            '\nВсего фигур: ', Figure.count)
```

Наследование

```
# main
red_figure = Figure('Фигура красная', 'Красный')
red_figure.print_info()
circle = Circle('Круг 1', 'Белый', (10, 12), 5)
circle.print_info()
```

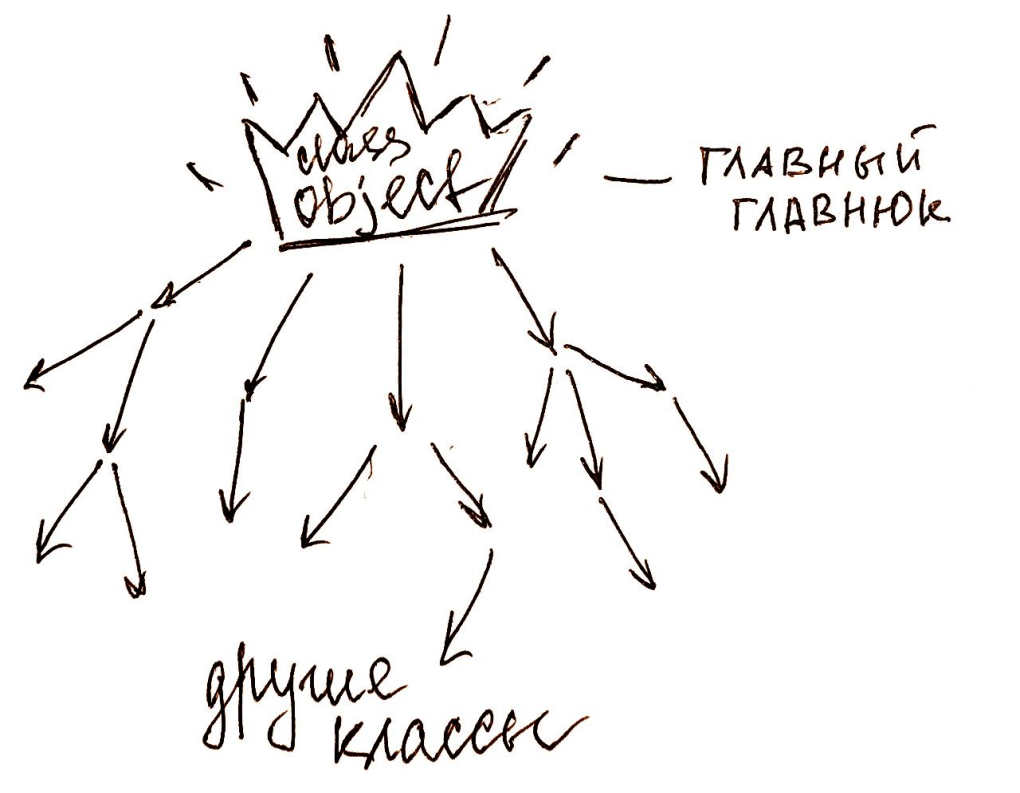


Наследование

В языке Python во главе иерархии классов стоит класс object.

Функция **issubclass**(X, Y) проверяет, является ли класс X производным (то есть потомком) от базового класса Y.

Принадлежность объекта a классу b можно выяснить с помощью функции **isinstance**(a, b).



Наследование

```
class Parent(object):  # object можно не указывать
   pass

class Child(Parent):
   pass

issubclass(Parent, object)  # True
issubclass(Child, Parent)  # True
issubclass(Child, object)  # True
issubclass(Child, str)  # False
issubclass(Parent, Parent)  # класс является подклассом самого себя

parent = Parent()
isinstance(parent, Parent)  # True
```

Абстрактные классы

Абстрактный класс - это класс, предназначенный только для наследования. Его экземпляры обычно не имеют смысла.

Абстрактный класс необходим как декларация функциональности некоторого типа, реализация которой делегируется классу потомку.

```
class Figure:
    count = 0

    def __init__(self, name, color):
        self.color = color
        self.name = name
        Figure.count += 1

    def print_info(self):
        pass

    def get_square(self):
        pass
```

Абстрактные классы

```
class Circle(Figure):
    def __init__(self, name, color, point, radius):
        super().__init__(name, color)
        self.point = point
        self.radius = radius

    def print_info(self):
        print('Имя: ', self.name,
              '\nЦвет: ', self.color,
              '\nВсего фигур: ', Figure.count,
              '\nРадиус: ', self.radius,
              '\nВсего фигур: ', Figure.count)

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

Множественное наследование

Множественное наследование - производный класс может иметь более, чем один базовый класс.

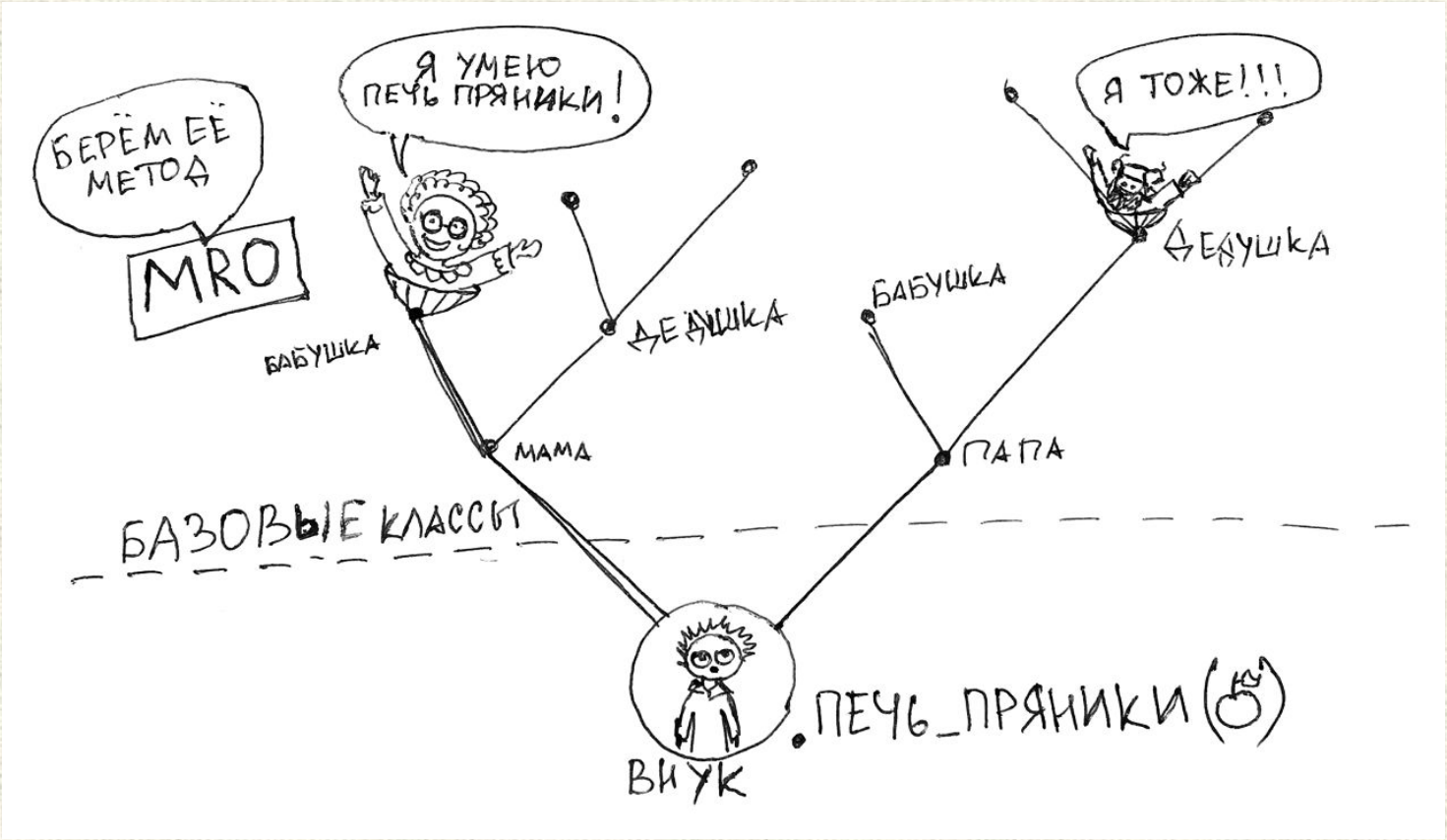
```
class A:
  def a(self): return 'a'

class B:
  def b(self): return 'b'

class C:
  def c(self): return 'c'

class AB(A, B):
  pass

class BC(B, C):
  pass

class ABC(A, B, C):
  pass
```

Множественное наследование

Стрекодил (крокоза)

Разрешение методов

Порядок разрешения методов (Method resolution order) - это алгоритм, который определяет, какой метод вызвать в дочернем классе, в случае, когда базовые классы имеют методы с одинаковыми именами.



Разрешение методов

В Python поиск метода происходит сначала в дереве классов первого родителя, потом второго и т.д.