Что такой “абстрактный”

Если вас попросить нарисовать животное, что вы нарисуете? Если вы нарисуете кошку, собаку или корову, это будут кошка, собака или корова. Мы не можем изобразить просто животное, хотя знаем что это такое. Такое понятие называют *абстрактным*, т.е. обобщением, порожденным нашим сознанием. В противовес этому кошка, собака или корова являются *конкретными* воплощениями абстракции “животное”.

Так как кошка, собака или корова *являются* животными, с точки зрения объектно-ориентированного программирования они являются подклассами, а животное – суперклассом. Допустим Мурзик это экземпляр класса Кошка, Шарик это экземпляр класса Собака, а Буренка это экземпляр класса Корова. При этом допустимых экземпляров просто животного мы придумать не можем. По этой причине обычно на уровне языка программирования запрещено создавать экземпляры абстрактных классов.

Абстрактные методы

Кошка говорит “мяу”, собака говорит “гав”, а корова говорит “мууу”. Определять какой звук издает абстрактное животное – не имеет смысла. Таким образом, если реализации какого-либо метода имеют смысл только в конкретных классах, метод в базовом классе можно оставить без реализации, а чтобы компилятор не жаловался, пометить его абстрактным.

С языковой стороны в Python нет возможности объявить абстрактный метод. Вместо этого он полагается на утиную типизацию, позволяя просто не определять метод в предке. В этом случае даже если какой-то программист создаст экземпляр класса, названного нами абстрактным, он будет непригодным к использованию из-за отсутствия нужных методов. А зачастую в Python можно вообще не создавать базовый класс, если единственное для чего он используется, это определение интерфейса для других классов.

Тем не менее, в этом случае имена методов для реализации находятся исключительно в голове разработчиков (и в лучшем случае в документации). Чтобы иметь возможность выявить необходимые для реализации потомками методы, можно их все же объявлять в предке, но при попытке их вызвать напрямую без переопределения, выбрасывать исключительную ситуацию.

```
class Animal:
    def say(self):
        raise NotImplementedError(“Метод абстрактный”)

class Cat(Animal):
    def say(self):
```

```
        print("Meow!")

class Dog(Animal):
    def say(self):
        print("Wow")
```

Однако такой подход не позволяет сделать класс, содержащий абстрактные методы абстрактным. В данном примере все равно можно создавать экземпляры класса Animal.

Так как многие вещи, не являющиеся языковыми конструкциями, в Python можно реализовать самостоятельно, в стандартной библиотеке Python есть модуль abc (сокращение от Abstract Base Classes: абстрактные базовые классы), позволяющий создавать абстрактные методы и абстрактные классы.

Модуль abc определяет метакласс ABCMeta, включащий возможность помечать методы абстрактными. Таким образом, для указания метода абстрактным, необходимо указать классу, что его метаклассом является ABCMeta, а для нужных методов указать декоратор @abstractmethod.

```
from abc import ABCMeta, abstractmethod

class Animal(metaclass=ABCMeta):
    @abstractmethod
    def say(self):
        pass # можно даже определить поведение по-умолчанию

class Cat(Animal):
    def say(self):
        print("Meow!")

class Dog(Animal):
    def say(self):
        print("Wow")
```

Декоратор @abstractmethod можно сочетать с декораторами @propery (и сеттером для него), а также @classmethod, указывая его как самый внутренний декоратор.

```
class Base(metaclass=ABCMeta):
    @property
    @abstractmethod
    def value(self):
        pass
```

```
    @value.setter
    @abstractmethod
    def value(self, x):
        pass

    @classmethod
    @abstractmethod
    def something(cls):
        pass
```

Абстрактные классы

Класс, который содержит хотя бы один абстрактный метод (@abstractmethod), сам является абстрактным и, как следствие, его экземпляры нельзя создавать. Если класс унаследовал абстрактный метод и не переопределил его, в этом случае он также остается абстрактным. Чтобы наследник перестал быть абстрактным, он должен определить все абстрактные методы своих предков.

В Python нет синтаксической возможности определить абстрактный класс, не содержащий абстрактных методов.

Интерфейсы

Благодаря механизму полиморфизма, вызывающему неважно как определены те или иные методы, главное, что у вызываемого есть методы с нужными именами и списками параметров. Хорошей практикой является определять интерфейсы, содержащие заголовки нужных методов и реализовывать их конкретными классами. При этом в качестве параметров собственных методов и функций предпочитать указывать интерфейсы, а не конкретные классы. Этот подход позволит использовать этот метод или функцию, имея любую реализацию нужного интерфейса, а не только одну конкретную.

Если имена классов главным словом имеют существительное, то для интерфейсов распространено указание имен, являющихся прилагательными, часто с суффиксом -able (см. пример далее).

Несмотря на то, что синтаксически в Python нет интерфейсов, их функцию выполняют классы, все методы которых абстрактные. Стоит отметить, что есть сторонние реализации механизмов интерфейсов, которые, впрочем, не стоит сочетать с модулем abc. Наиболее известным является пакет из фреймворка Zope под названием zope.interface. Тем не менее, так как это сторонняя библиотека, она выходит за рамки этого курса и все примеры будут рассмотрены с использованием модуля abc.

```
class Movable(metaclass=ABCMeta): # интерфейс
    @abstractmethod
    def move_to(self, x, y):
        pass

    @abstractmethod
    def move_by(self, delta_x, delta_y):
        pass

class Point(Movable): # реализация
    def move_to(self, x, y):
        # реализация

    def move_by(self, delta_x, delta_y):
        # реализация

    # остальная реализация
```