Программирование на Python: Часть 6. Классы

Сергей Яковлев
Консультант
независимый специалист

02.06.2017

Основные свойства ООП — полиморфизм, наследование, инкапсуляция. Класс — это пользовательский тип, состоящий из методов и атрибутов. Инстанс класса создается путем вызова имени класса как функции с параметрами. Объект состоит из атрибутов и методов. Атрибут — это переменная, метод — это функция. Отличия метода от функции в том, что у него есть первый параметр — self. Полиморфизм позволяет нам работать с различными типами объектов так, что нам не нужно задумываться о том, к какому типу они принадлежат.

Мы переходим к одной из самых интересных тем цикла — объектно-ориентированному программированию (ООП) в Python. С точки зрения ООП, класс представляет собой коллекцию данных. Использование классов дает нам прежде всего преимущества абстрактного подхода в программировании.

1. **Полиморфизм**: в разных объектах одна и та же операция может выполнять различные функции. Слово «полиморфизм» имеет греческую природу и означает «имеющий многие формы». Простым примером полиморфизма может служить функция `count()`, выполняющая одинаковое действие для различных типов обьектов: `'abc'.count('a')` и `[1, 2, 'a'].count('a')`. Оператор плюс полиморфичен при сложении чисел и при сложении строк.
2. **Инкапсуляция**: можно скрыть ненужные внутренние подробности работы объекта от окружающего мира. Это второй основной принцип абстракции. Он основан на использовании атрибутов внутри класса. Атрибуты могут иметь различные состояния в промежутках между вызовами методов класса, вследствие чего сам объект данного класса также получает различные состояния — state.
3. **Наследование**: можно создавать специализированные классы на основе базовых. Это позволяет нам избегать написания повторного кода.
4. **Композиция**: объект может быть составным и включать в себя другие объекты.

Объектно-ориентированный подход в программировании подразумевает следующий алгоритм действий.

1. Описывается проблема с помощью обычного языка с использованием понятий, действий, прилагательных.
2. На основе понятий формулируются классы.
3. На основе действий проектируются методы.
4. Реализуются методы и атрибуты.

Мы получили скелет — объектную модель. На основе этой модели реализуется наследование. Для проверки модели:

i. пишется так называемый use cases — сценарий возможного поведения модели, где проверяется вся функциональность;
ii. функционал при этом может быть исправлен либо добавлен.

Объектно-ориентированный подход хорош там, где проект подразумевает долгосрочное развитие, состоит из большого количества библиотек и внутренних связей.

Механизм классов языка Python представляет собой смесь механизмов классов C++ и Modula-3. Наиболее важные особенности классов в питоне:

a. множественное наследование;
b. производный класс может переопределить любые методы базовых классов;
c. в любом месте можно вызвать метод с тем же именем базового класса;
d. все атрибуты класса в питоне по умолчанию являются public, т.е. доступны отовсюду; все методы — виртуальные, т.е. перегружают базовые.

Сегодня мы рассмотрим следующие аспекты объектно-ориентированного программирования.

1. Что такое класс.
2. Атрибуты класса.
3. self.
4. Наследование.
5. ООП в действии: пример создания классов.

1. Что такое класс

Класс — это пользовательский тип. Простейшая модель определения класса выглядит следующим образом:

```
class имя_класса:
инструкция 1
.
инструкция №
```

Каждая такая запись генерирует свой объект класса. Отличие от C++ в том, что в плюсах описание класса — это лишь объявление, а в питоне — это создание объекта. Есть также другой тип объекта — инстанс класса, который генерируется при вызове:

```
инстанс_класса = имя_класса()
```

Объект класса и инстанс класса — это два разных объекта. Первый генерируется на этапе объявления класса, второй — при вызове имени класса. Объект класса может быть один, инстансов класса может быть сколько угодно.

Инструкция — это, как правило, определение функции. При определении класса создается новое пространство имен и создается объект-класс, который является оболочкой для всех инструкций.

Объекты классов поддерживают два вида операций:

- доступ к атрибутам;
- создание экземпляра класса.

2. Атрибуты класса

Атрибуты класса бывают двух видов:

- атрибуты данных;
- атрибуты-методы.

Атрибуты данных обычно записываются сверху. Память для атрибутов выделяется в момент их первого присваивания — либо снаружи, либо внутри метода. Методы начинаются со служебного слова def.

Доступ к атрибутам выполняется по схеме obj.attrname.

Пример.

```
class Simple:
 u'Простой класс'
 var = 87
 def f(x):
 return 'Hello world'
```

Здесь Simple.var и Simple.f — пользовательские атрибуты. Есть также стандартные атрибуты:

```
>>> print Simple.__doc__
```

Целое число

```
>>> print Simple.var.__doc__
int(x[, base]) -> integer
...
```

Создание экземпляра класса похоже на то, как будто мы делаем вызов функций:

```
 smpl = Simple()
```

Будет создан пустой объект smpl. Если мы хотим, чтобы при создании выполнялись какие-то действия, нужно определить конструктор, который будет вызываться автоматически:

```
 class Simple:
 def __init__(self):
 self.list = []
```

При создании объекта smpl будет создан пустой список list. Конструктору можно передать аргументы:

```
 class Simple:
 def __init__(self, count, str):
 self.list = []
 self.count = count
 self.str = str

>>> s = Simple(1,'22')
>>> s.count, s.str
1 22
```

Атрибут данных можно сделать приватным (private) — т.е. недоступным снаружи — для этого слева нужно поставить два символа подчеркивания:

```
class Simple:
 u'Простой класс с приватным атрибутом'
 __private_attr = 10
 def __init__(self, count, str):
 self.__private_attr = 20
 print self.__private_attr

s = Simple(1,'22')
print s.__private_attr
```

Последняя строка вызовет исключение — атрибут __private_attr годен только для внутреннего использования.

Методы необязательно определять внутри тела класса:

```
def method_for_simple(self, x, y):
 return x + y

class Simple:
 f = method_for_simple

>>> s = Simple()
>>> print s.f(1,2)
3
```

Пустой класс можно использовать в качестве заготовки для структуры данных:

```
class Customer:
 pass
custom = Customer()
custom.name = 'Вася'
custom.salary = 100000
```

3. self

Обычно первый аргумент в имени метода — self. Как говорит автор языка Гвидо Ван Россум, это не более чем соглашение: имя self не имеет абсолютно никакого специального значения.

self полезен для того, чтобы обращаться к другим атрибутам класса:

```
class Simple:
 def __init__(self):
 self.list = []
 def f1(self):
 self.list.append(123)
 def f2(self):
 self.f1()

>>> s = Simple()
>>> s.f2()
>>> print s.list
[123]
```

Self — это аналог "this" в C++.

4. Наследование

Определение производного класса выглядит следующим образом:

```
class Derived(Base):
```

Если базовый класс определен не в текущем модуле:

```
class Derived(module_name.Base):
```

Разрешение имен атрибутов работает сверху вниз: если атрибут не найден в текущем классе, поиск продолжается в базовом классе, и так далее по рекурсии. Производные классы могут переопределить методы базовых классов — все методы являются в этом смысле виртуальными. Вызвать метод базового класса можно с префиксом:

```
Base.method()
```

В питоне существует ограниченная поддержка множественного наследования:

```
class Derived(Base1,Base2,Base3):
```

Поиск атрибута производится в следующем порядке:

1. в Derived;
2. в Base1, затем рекурсивно в базовых классах Base1;
3. в Base2, затем рекурсивно в базовых классах Base2
4. и т.д.

5. Пример

Создадим два класса: Person — хранит общую информацию о людях — имя, профессия, зарплата; класс Manager — специализированный производный класс. В классе Person мы создадим свою версию для стандартной встроенной функции str, которая есть по умолчанию в любом питоновском классе — для этого она будет иметь префикс с двумя символами

подчеркивания слева и справа. Когда мы попытаемся распечатать инстанс класса, будет вызвана __str__.

```
# -*- coding: utf-8 -*-
class Person:
 def __init__(self, name, job=None, pay=0):
 self.name = name
 self.job = job
 self.pay = pay
 def lastName(self):
 return self.name.split()[-1]
 def giveRaise(self, percent):
 self.pay = int(self.pay * (1 + percent))
 def __str__(self):
 return '[Person: %s, %s]' % (self.name, self.pay)

class Manager(Person):
 def __init__(self, name, pay):
 Person.__init__(self, name, 'mgr', pay)
 def giveRaise(self, percent, bonus=100):
 Person.giveRaise(self, percent + bonus)
```

Создаем первый инстанс класса Person:

```
>>> ivan = Person('Иван Petrov')
```

Создаем второй инстанс класса Person:

```
>>> john = Person('John Sidorov', job='dev', pay=100000)
```

Вызываем перегруженную функцию __str__;

```
>>> print(ivan)
>>> print(john)
```

Выводим фамилию:

```
>>> print(ivan.lastName(), john.lastName())
```

Начисляем премиальные:

```
>>> john.giveRaise(.10)
```

И получаем:

```
>>> print(john)
```

Создаем инстанс класса Manager:

```
>>> tom = Manager('Tom Jones', 50000)
```

Начисляем мегапремиальные:

```
>>> tom.giveRaise(.10)
```

Выводим:

```
print(tom.lastName())
print(tom)
```

Вывод:

```
[Person: Иван Petrov, 0]
[Person: John Sidorov, 100000]
('Petrov', 'Sidorov')
[Person: John Sidorov, 110000]
Jones
[Person: Tom Jones, 5055000]
```

Заключение

Основные свойства ООП — полиморфизм, наследование, инкапсуляция. Класс — это пользовательский тип, состоящий из методов и атрибутов. Инстанс класса создается путем вызова имени класса как функции с параметрами. Объект состоит из атрибутов и методов. Атрибут — это переменная, метод — это функция. Отличия метода от функции в том, что у него есть первый параметр — self. Полиморфизм позволяет нам работать с различными типами объектов так, что нам не нужно задумываться о том, к какому типу они принадлежат. Объекты могут скрывать (инкапсулировать) свое внутреннее состояние. Это достигается за счет того, что доступ к атрибутам осуществляется не напрямую, а через методы. Класс может быть производным от одного или нескольких классов. Производный класс наследует все методы базового класса. Базовых классов может быть несколько. Объектный дизайн должен быть прозрачным, понятным и описан, что называется, в 'терминах человеческого языка'.

В следующей статье мы расскажем о работе со специальными методами и атрибутами классов. Код примеров проверялся на версии питона 2.6.

< Предыдущая статья. Следующая статья >

Об авторе

Сергей Яковлев