Классы в Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

Python полностью объектно-ориентирован, то есть вы можете определять свои собственные классы, наследовать новые классы от своих или встроенных классов, и создавать экземпляры классов, которые уже определили.

Определить класс в Python просто. Также как и в случае с функциями, раздельное объявление интерфейса не требуется. Вы просто определяете класс и начинаете программировать. Определение класса в Python начинается с зарезервированного слова class, за которым следует имя (идентификатор) класса. Формально, это все, что необходимо, в случае, когда класс не должен быть унаследован от другого класса.

class PapayaWhip:
    pass

1. Определенный выше класс имеет имя PapayaWhip и не наследует никакой другой класс. В именах классов каждое слово обычно пишется с большой буквы, но это не требование, а лишь соглашение.
2. Вы наверное уже догадались, что каждая строка в определении класса имеет отступ, также как и в случае с функциями, оператором условного перехода if, циклом for или любым другим блоком кода. Первая строка без отступа находится вне блока class.

Класс PapayaWhip не содержит определений методов или атрибутов, но с точки зрения синтаксиса, тело класса не может оставаться пустым. В таких случаях используется оператор pass. В языке Python pass — зарезервированное слово, которое говорит интерпретатору: «идем дальше, здесь ничего нет». Это оператор не делающий ровным счетом ничего, но тем не менее являющийся удобным решением, когда вам нужно сделать заглушку для функции или класса.

Выражение pass в языке Python аналог пустого множества или фигурных скобок в языках Java или C++.

Многие классы наследуются от других классов, но не этот. Многие классы определяют свои методы, но не этот. Класс в Python не обязан иметь ничего, кроме имени. В частности, людям знакомым с C++ может показаться странным, что у класса в Python отсутствуют в явном виде конструктор и деструктор. Несмотря на то, что это не является обязательным, класс в Python может иметь нечто, похожее на конструктор: метод *init*().

МЕТОД *INIT*

В следующем примере демонстрируется инициализация класса Fib, с помощью метода *init*().

class Fib:
'''iterator that yields numbers
   in the Fibonacci sequence'''
    def \_\_init\_\_(self, max):

        …

1. Классы, по аналогии с модулями и функциями могут (и должны) иметь строки документации (docstrings).
2. Метод *init*() вызывается сразу же после создания экземпляра класса. Было бы заманчиво, но формально неверно, считать его «конструктором» класса. Заманчиво, потому что он напоминает конструктор класса в языке C++: внешне (общепринято, что метод *init*() должен быть первым методом, определенным для класса), и в действии (это первый блок кода, исполняемый в контексте только что созданного экземпляра класса). Неверно, потому что на момент вызова *init*() объект уже фактически является созданным, и вы можете оперировать корректной ссылкой на него (self)

Первым аргументов любого метода класса, включая метод *init*(), всегда является ссылка на текущий экземпляр класса. Принято называть этот аргумент self. Этот аргумент выполняет роль зарезервированного слова this в C++ или Java, но, тем не менее, в Python self не является зарезервированным. Несмотря на то, что это всего лишь соглашение, пожалуйста не называйте этот аргумент как либо еще.

В случае метода *init*(), self ссылается на только что созданный объект; в остальных методах — на экземпляр, метод которого был вызван. И, хотя вам необходимо явно указывать self при определении метода, при вызове этого не требуется; Python добавит его для вас автоматически.

СОЗДАНИЕ ЭКЗЕМПЛЯРОВ

Для создания нового экземпляра класса в Python нужно вызвать класс, как если бы он был функцией, передав необходимые аргументы для метода *init*(). В качестве возвращаемого значения мы получим только что созданный объект.

>>> import fibonacci2
>>> fib = fibonacci2.Fib(100)
>>> fib
<fibonacci2.Fib object at 0x00DB8810>
>>> fib.\_\_class\_\_
<class 'fibonacci2.Fib'>
>>> fib.\_\_doc\_\_
'iterator that yields numbers in the Fibonacci sequence'

1. Вы создаете новый экземпляр класса Fib (определенный в модуле fibonacci2) и присваиваете только что созданный объект переменной fib. Единственный переданный аргумент, 100, соответствует именованному аргументу max, в методе *init*() класса Fib.
2. fib теперь является экземпляром класса Fib
3. Каждый экземпляр класса имеет встроенный атрибут *class*, который указывает на класс объекта. Java программисты могут быть знакомы с классом Class, который содержит методы getName() и getSuperclass(), используемые для получения информации об объекте. В Python, метаданные такого рода доступны через соответствующие атрибуты, но используемая идея та же самая.
4. Вы можете получить строку документации (docstring) класса, по аналогии с функцией и модулем. Все экземпляры класса имеют одну и ту же строку документации.

Для создания нового экземпляра класса в Python, просто вызовите класс, как если бы он был функцией, явные операторы, как например new в С++ или Java, в языке Python отсутствуют.

ПЕРЕМЕННЫЕ ЭКЗЕМПЛЯРА

Перейдем к следующей строчке:

class Fib:
    def \_\_init\_\_(self, max):
        self.max = max

1. Что такое self.max? Это переменная экземпляра. Она не имеет ничего общего с переменной max, которую мы передали в метод *init*() в качестве аргумента. self.max является «глобальной» для всего экземпляра. Это значит, что вы можете обратиться к ней из других методов.

class Fib:
    def \_\_init\_\_(self, max):
        self.max = max

    def \_\_next\_\_(self):
        …
        if fib > self.max:
            …

1. self.max определена в методе \_\_*init\_\_*…
2. …и использована в методе \_\_next\_\_.

Переменные экземпляра связаны только с одним экземпляром класса. Например, если вы создадите два экземпляра класса Fib с разными максимальными значениями, каждый из них будет помнить только свое собственное значение.

>>> import fibonacci2
>>> fib1 = fibonacci2.Fib(100)
>>> fib2 = fibonacci2.Fib(200)
>>> fib1.max
100
>>> fib2.max
200