**Конструктор класса – метод \_\_init\_\_()**

В объектно-ориентированном программировании конструктором класса называют метод, который автоматически вызывается при создании объектов. Его также можно назвать конструктором объектов класса. Имя такого метода обычно регламентируется синтаксисом конкретного языка программирования. Так в Java имя конструктора класса совпадает с именем самого класса.

В Python же роль конструктора играет метод \_\_init\_\_().

В Python наличие пар знаков подчеркивания спереди и сзади в имени метода говорит о том, что он принадлежит к группе методов перегрузки операторов. Если подобные методы определены в классе, то объекты могут участвовать в таких операциях как сложение, вычитание, вызываться как функции и др.

При этом методы перегрузки операторов не надо вызывать по имени. Вызовом для них является сам факт участия объекта в определенной операции. В случае конструктора класса – это операция создания объекта. Так как объект создается в момент вызова класса по имени, то в этот момент вызывается метод \_\_init\_\_(), если он определен в классе.

Необходимость конструкторов связана с тем, что нередко объекты должны иметь собственные свойства сразу. Пусть имеется класс Person, объекты которого обязательно должны иметь имя и фамилию. Если класс будет описан подобным образом

class Person:

def setName(self, n, s):

self.name = n

self.surname = s

то создание объекта возможно без полей. Для установки имени и фамилии метод setName() нужно вызывать отдельно:

>>> from test import Person

>>> p1 = Person()

>>> p1.setName("Bill", "Ross")

>>> p1.name, p1.surname

('Bill', 'Ross')

В свою очередь, конструктор класса не позволит создать объект без обязательных полей:

class Person:

def \_\_init\_\_(self, n, s):

self.name = n

self.surname = s

p1 = Person("Sam", "Baker")

print(p1.name, p1.surname)

Здесь при вызове класса в круглых скобках передаются значения, которые будут присвоены параметрам метода \_\_init\_\_(). Первый его параметр – self – ссылка на сам только что созданный объект.

Теперь, если мы попытаемся создать объект, не передав ничего в конструктор, то будет возбуждено исключение, и объект не будет создан:

>>> p1 = Person()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: \_\_init\_\_() missing 2 required

positional arguments: 'n' and 's'

Однако бывает, что надо допустить создание объекта, даже если никакие данные в конструктор не передаются. В таком случае параметрам конструктора класса задаются значения по умолчанию:

class Rectangle:

def \_\_init\_\_(self, w = 0.5, h = 1):

self.width = w

self.height = h

def square(self):

return self.width \* self.height

rec1 = Rectangle(5, 2)

rec2 = Rectangle()

rec3 = Rectangle(3)

rec4 = Rectangle(h = 4)

print(rec1.square())

print(rec2.square())

print(rec3.square())

print(rec4.square())

Вывод:

10

0.5

3

2.0

Если класс вызывается без значений в скобках, то для параметров будут использованы их значения по умолчанию. Однако поля width и height будут у всех объектов.

Кроме того, конструктору вовсе не обязательно принимать какие-либо параметры, не считая self. Значения полям могут назначаться как угодно. Также не обязательно, чтобы в конструкторе происходила установка атрибутов объекта. Там может быть, например, код, который порождает создание объектов других классов.

В других языка программирования, например в Java, классы могут содержать несколько конструкторов, которые между собой отличаются количеством параметром, а также, возможно, их типом. При создании объекта срабатывает тот конструктор, количество и типы параметров которого совпали с количеством и типами переданных в конструктор аргументов.

В Python создать несколько методов \_\_init\_\_() в классе можно, однако "рабочим" останется только последний. Он переопределит ранее определенные. Поэтому в Python в классах используется только один конструктор, а изменчивость количества передаваемых аргументов настраивается через назначение значений по-умолчанию.

Помимо конструктора объектов в языках программирования есть обратный ему метод – деструктор. Он вызывается, когда объект не создается, а уничтожается.

В языке программирования Python объект уничтожается, когда исчезают все связанные с ним переменные или им присваивается другое значение, в результате чего связь со старым объектом теряется. Удалить переменную можно с помощью команды языка del.

В классах Python функцию деструктора выполняет метод \_\_del\_\_().