Практическая работа 13

Конструкторы и деструкторы классов

Время - 2 часа

Класс описывается с помощью ключевого слова *class* по следующей схеме:

```
class <Название класса>[(<Класс1>, …, <Класс№>)]: """ Строка документирования """
<Описание атрибутов и методов>
```

Инструкция создает новый объект и присваивает ссылку на него идентификатору, указанному после ключевого слова class. Это означает, что название класса должно полностью соответствовать правилам именования переменных. После названия класса в круглых скобках можно указать один или несколько базовых классов через запятую. Если же класс не наследует базовые классы, то круглые скобки можно не указывать. Следует заметить, что все выражения внутри инструкции class выполняются при создании класса, а не его экземпляра. Для примера создадим класс, внутри которого просто выводится сообщение:

```
class MyClass:
""" Это строка документирования """ print("Инструкции выполняются сразу")
```

Этот пример содержит лишь определение класса *Myclass* и не создает экземпляр класса. Как только поток выполнения достигнет инструкции *class*, сообщение, указанное в функции *print*(), будет сразу выведено.

С точки зрения пространства имен класс можно представить подобным модулю. Также как в модуле в классе могут быть свои переменные со значениями и функции. Также как в модуле у класса есть собственное пространство имен, доступ к которому возможен через имя класса:

```
class B:
n = 5
def adder(v): return v + B.n
print(B.n) # Вывод: 5 print(B.adder(4)) # Вывод: 9
```

Однако в случае классов используется особая терминология. Имена, определенные в классе, называются атрибутами этого класса. В примере имена *n* и *adder* – это атрибуты класса *B*. Атрибуты-переменные называют полями или свойствами (в других языках понятия "поле" и "свойство" не совсем одно и то же). Полем является *n*. Атрибуты-функции называются методами. Методом в классе *B* является *adder*. Количество свойств и методов в классе может быть любым.

Создание атрибута класса аналогично созданию обычной переменной. Метод внутри класса создается так же, как и обычная функция, – с помощью инструкции *def*. Среди атрибутов можно выделить две группы.

Атрибуты первой группы принадлежат самому классу, а не конкретному экземпляру. Доступ к таким атрибутам вне тела класса осуществляется через точечную нотацию через объект объявления класса.

<Имя класса>.<Имя атрибута>

Если этот атрибут — метод, то по аналогии можно его вызвать.

<Имя класса>.<Имя метода>([<Параметры>])

Во второй группе находятся атрибуты, принадлежащие конкретному экземпляру. В отличии от многих других языков программирования, эти атрибуты создаются уже во время существования объекта. Чтобы создать экземпляр класса используется синтаксис:

<экземпляр класса> = <Название класса>([<Параметры>])

Когда экземпляр класса существует, можно задавать атрибуты уникальные для этогосамого экземпляра:

<экземпляр класса>.<Имя атрибута> = <Значение>

И по аналогии получать доступ к уже существующим атрибутам:

<экземпляр класса>.<Имя атрибута>

При этом если не удается найти атрибут с таким именем у экземпляра, то поиск продолжается в атрибутах, принадлежащих классу. В связи с этим можно считать, что атрибуты самого класса разделяются между всеми его экземплярами.

```
class MyClass:
```

```
    x = 50          # Создаем атрибут объекта класса MyClass

    c1, c2 = MyClass(), MyClass() # Создаем два экземпляра класса
c1.y = 10           # Создаем атрибут экземпляра класса c1 c2.y = 20 #
Создаем атрибут экземпляра класса c2 print(c1.x, ' ', c1.y) # Вывод:
50 10
    print(c2.x, ' ', c2.y) # Вывод: 50 20
```

В этом примере мы определяем класс *MyClass* и создаем атрибут объекта класса: *x*. Этот атрибут будет доступен всем создаваемым экземплярам класса. Затем создаем два экземпляра класса и добавляем одноименные атрибуты: *y*. Значения этих атрибутов будут разными в каждом экземпляре класса. Но если создать новый экземпляр (например, *c3*), то атрибут *y* в нем определен не будет. Таким образом, с помощью классов можно имитировать типы данных, поддерживаемые другими языками программирования (например, тип *struct*, доступный в языке C).

Обычно все методы класса объявляются на уровне класса, а не экземпляра. Вызов такого метода через экземпляр класса с помощью синтаксиса

<экземпляр класса>.<Имя метода>([<Параметры>])

неявно преобразуется к

<Имя класса>.<Имя метода>(<экземпляр класса>, [<параметры>])

Иными словами, методам класса в первом параметре, который необходимо указывать явно, автоматически передается ссылка на экземпляр класса. Общепринято этот параметр называть именем ***self***, хотя это и не обязательно. Доступ к атрибутам и методам класса внутри определяемого метода производится через переменную *self* с помощью точечной нотации. Обратите внимание на то, что при вызове метода не нужно передавать ссылку на экземпляр класса в качестве параметра, как это делается в определении метода внутри класса. Ссылку на экземпляр класса интерпретатор передает автоматически.

Определим класс *MyClass* с атрибутом *x* и методом *print_x()*, выводящим значение этого атрибута, а затем создадим экземпляр класса и вызовем метод:

```
class MyClass:
```

```
    def __init__(self):    # Конструктор
    self.x = 1        # Атрибут экземпляра класса
    def print_x(self):    #     self — это ссылка на экземпляр класса
print(self.x)            # Выводим значение атрибута
    c = MyClass()  # Создание экземпляра класса # Вызываем метод
print_x()
    c.print_x()       # self не указывается при вызове метода print(c.x)
    # К атрибуту можно обратиться непосредственно
```

Все атрибуты класса в языке Python являются открытыми (*public*), т. е. доступными для непосредственного изменения как из самого класса, так и из других классов и из основного кода программы.

Очень важно понимать разницу между атрибутами объекта класса и атрибутами экземпляра класса. Атрибут объекта класса существует в единственном экземпляре, доступен всем экземплярам класса, его изменение можно видеть во всех экземплярах класса. Атрибут экземпляра класса хранит уникальное значение для каждого экземпляра, и изменение его в одном экземпляре класса не затронет значения одноименного атрибута в других экземплярах того же класса. Рассмотрим это на примере, создав класс с атрибутом объекта класса (x) и атрибутом экземпляра класса (y):

```
    class MyClass:
    x = 10      # Атрибут объекта класса общий для всех экземпляровdef
init_(self):
    self.y = 20 # Атрибут экземпляра класса уникальный для каждого
экземпляраc1 = MyClass()# Создаем первый экземпляр класса
    c2 = MyClass()    # Создаем      второй экземпляр класса
    #Выведем значения атрибута x, а затем изменим значение и опять
произведем вывод:print  (c1.x,   c2.x)# Вывод: 10      10
    MyClass.x = 88              # Изменяем атрибут объекта (класса
MyClass) print  (c1.x,          c2.x)    # Вывод: 88      88
    print  (c1.y,   c2.y)    # Вывод: 20 20
    cl.y = 88      # Изменяем атрибут экземпляра класса c1 print  (c1.y,
        c2.y)    # Вывод: 88 20
```

Как видно из примера, изменение атрибута объекта класса x затронуло значение в обоих экземплярах класса. Аналогичная операция с атрибутом y изменяет значение тольков экземпляре c1.

Следует также учитывать, что в одном классе могут одновременно существовать атрибут объекта и атрибут экземпляра с одним именем. Изменение атрибута объекта классамы производили следующим образом:

```
MyClass.x = 88 # Изменяем атрибут объекта класса
```

Если после этой инструкции вставить инструкцию:

```
cl.x = 200# Создаем атрибут экземпляра
```

то будет создан атрибут экземпляра класса, а не изменено значение атрибута объектакласса. Чтобы увидеть разницу, нужно вывести их значения:

```
print (cl.x, MyClass.x)   # 200 88
```

Метод__init__()

При создании экземпляра класса интерпретатор автоматически вызывает метод инициализации*_init_()*. Такой метод принято называть конструктором класса. Формат метода:

```
def init__(self[, <Значение1>[, . .., <ЗначениеN>]]):
<Инструкции>
```

С помощью метода*__init__()* атрибутам класса можно присвоить начальные значения.

При создании экземпляра класса параметры этого метода указываются после имени класса в круглых скобках:

```
<Экземпляр класса> = <Имя класса> ([<Значение1> [, ...,<ЗначениеN>]])
```

Пример использования метода:

```
class MyClass:
def __init__(self, valuel, value2): # Конструктор self.x = valuel
self.y = value2
c = MyClass(100, 300) # Создаем экземпляр класса print(c.x, c.y) # Вывод: 100 300
```

Метод __del__()

Перед уничтожением экземпляра автоматически вызывается метод, называемый деструктором. В языке Python деструктор реализуется в виде предопределенного метода *_del_()* (листинг 13.6). Следует заметить, что метод не будет вызван, если на экземпляр класса существует хотя бы одна ссылка.

Впрочем, поскольку интерпретатор самостоятельно заботится об удалении объектов, использование деструктора в языке Python не имеет особого смысла.

Наследование является, пожалуй, самым главным понятием ООП.

Предположим, у нас есть класс (например, *Class1*). При помощи наследования мы можем создать новый класс (например, *Class2*), в котором будет реализован доступ ко всематрибутам и методам класса *Class1*:

```
class Class1:
# Базовый класс def func1(self):
print("Метод func1() класса Class1") def func2(self) :
print("Метод func2() класса Class1")

class Class2(Classl):
# Класс Class2 наследует класс Classl def func3(self):
print("Метод func3() класса Class2")

c = Class2() # Создаем экземпляр класса Class2
c.func1() # Выведет: Метод func1() класса Classl
c.func2() # Выведет: Метод func2() класса Classl
c.func3() # Выведет: Метод func3() класса Class2
```

Как видно из примера, класс *Classl* указывается внутри круглых скобок в определении класса *Class2*. Таким образом, класс *Class2* наследует все атрибуты и методы класса *Classl*.

Класс *Classl* называется базовым или суперклассом, а класс *Class2* – производным или подклассом.

Если имя метода в классе *Class2* совпадает с именем метода класса *Classl*,

то будет использоваться метод из класса *Class2*. Чтобы вызвать одноименный метод из базового класса, перед методом следует через точку написать название базового класса, а в первом параметре метода – явно указать ссылку на экземпляр класса. Рассмотрим это на примере

```
class Class1:
#       Базовый класс
def __init__( self ): print("Конструктор базового класса")
def func1(self):
print ("Метод func1()      класса Class1")

class Class2(Class1):
#       Класс Class2 наследует класс Classl def __init_( self ):
print("Конструктор производного класса")
Class1.__init__(self)      # Вызываем конструктор базового класса
def func1(self):
print ("Метод func1() класса Class2")
Class1.func1(self)   # Вызываем метод базового класса

c = Class2()    # Создаем экземпляр класса Class2 c.func1()      #
Вызываем метод func1()
```

Вывод:

Конструктор производного класса Конструктор базового класса Метод func1() класса Class2 Метод func1() класса Classl

Обратите внимание, что конструктор базового класса автоматически не вызывается, если он переопределен в производном классе. Поэтому его нужно вызывать явно либо так, как в приведенном примере, либо используя метод *super():*

```
super()._init_()
# Вызываем конструктор базового класса
```

или так:

```
super(Class2, self).__init__()
# Вызываем конструктор базового класса
```

При использовании функции *super()* не нужно явно передавать указатель *self* в вызываемый метод. Кроме того, в первом параметре функции *super()* указывается производный класс, а не базовый.

8. Множественное наследование

В определении класса в круглых скобках можно указать сразу несколько базовых классов через запятую. Рассмотрим пример

```
class Classl:
# Базовый класс для класса Class2 def func1(self):
print ("Метод func1()      класса Classl") class Class2(Classl):
# Класс Class2 наследует класс Classl def func2(self):
print("Метод func2()      класса Class2") class Class3(Classl):
# Класс Class3 наследует класс Classl def func1(self):
print("Метод func1()      класса Class3") def func2(self):
print("Метод func2()      класса Class3") def func3(self):
print("Метод func3()      класса Class3") def func4(self):
print("Метод func4()      класса  Class3") class  Class4(Class2,
                Class3):
# Множественное наследованиеdef func4(self):
print( "Метод func4()      класса Class4")
c = Class4()        # Создаем экземпляр класса Class4 c.func1() #
      Вывод: Метод func1() класса     Class3 c.func2()      #
      Вывод: Метод func2() класса     Class2 c.func3()      #
      Вывод: Метод func3() класса     Class3 c.func4()      #
      Вывод: Метод func4() класса     Class4
```

Метод *func1()* определен в двух классах: *Class1* и *Class3*. Так как вначале просматриваются все базовые классы, непосредственно указанные в определении текущего класса, то метод *func1()* будет найден в классе *Class3* (поскольку он указан в числе базовых классов в определении *Class4*), а не в классе *Classl*.

Метод *func2()* также определен в двух классах: *Class2* и *Class3*. Так как класс *Class2* стоит первым в списке базовых классов, то метод будет найден именно в нем. Чтобы наследовать метод из класса *Class3*, следует указать это явным образом:

```
class Class4(Class2,      Class3):
# Множественное наследование
func2 = Class3.func2 # Наследуем func2()      из класса Class3,
   а не из класса Class2 def      func4(self):
print("Метод func4()      класса Class4")
```

Задания для самостоятельного выполнения

К*ласс* Vector3D

Экземпляр класса задается тройкой координат в трехмерном пространстве (x,y,z).

Обязательно должны быть реализованы методы:

- приведение вектора к строке с выводом кооржинат (метод __str__),
- сложение векторов оператором `+` (метод __add__),
- вычитание векторов оператором `-` (метод __sub__),
- скалярное произведение оператором `*` (метод __mul__),
- умножение и деление на скаляр операторами `*` и `/` (метод __mul_____и __truediv__),
- векторное произведение оператором `@` (метод __matmul__),
- вычисление длинны вектора методом `norm`.

Пример

v1 = Vector3D(4, 1, 2)print(v1)

v2 = Vector3D()v2.read()

v3 = Vector3D(1, 2, 3)v4 = v1 + v2

print(v4)

a = v4 * v3print(a)

v4 = v1 * 10print(v4)

v4 = v1 @ v3print(v4)

Класс «Прямоугольный треугольник»

Класс содержит два действительных числа – стороны треугольника и включает следующие методы:

– увеличение/уменьшение размера стороны на заданное количество процентов;

– вычисление радиуса описанной окружности,

– вычисление периметра,

– определение значений углов.

Класс «Одномерный масив» TArray

Класс содержит поле для задания количества элементов и поле для хранения элементов массива.

Методы:

– конструктор без параметров, конструктор с параметрами, конструктор копирования,

– ввод и вывод данных,

– поиск максимального и минимального элементов,

– сортировка массива,

– поиск суммы элементов

– перегрузка оператора + (добавление элемента)

– перегрузка оператора * умножение элементов массива на число

Класс «Автобус».

Класс содержит свойства:

– speed (скорость),

– capacity (максимальное количество пассажиров),

– maxSpeed (максимальная скорость),

– passengers (список имен пассажиров),

– hasEmptySeats (наличие свободных мест),

– seats (словарь мест в автобусе);методы:

– посадка и высадка одного или нескольких пассажиров,

– увеличение и уменьшение скорости на заданное значение.

– операции "in", "+=" и "−=" (посадка и высадка пассажира(ов) с заданной фамилией)

Класс «Снежинки» Snow

Класс содержит целое число - количество снежинок.

Класс включает методы перегрузки арифметических операторов сложения, вычитания, умножения и деления. Код этих методов должен выполнять увеличение или уменьшение количества снежинок на число n или в n раз.

Класс также включает метод makeSnow(), который принимает сам объект и числоснежинок в ряду, а возвращает строку вида

*"*****\n*****\n*****...",*

где количество снежинок между '\n' равно переданному аргументу, а количество рядоввычисляется, исходя из общего количества снежинок.

Класс «Снежинка» (SnowFlake)

При инициализации класс принимает целое нечетное число – сторону квадрата, вкоторый вписана снежинка.

Методы:

– thaw() – таять, при этом на каждом шаге пропадают крайние звездочки со всех сторон;параметр показывает, сколько шагов прошло.

– freeze(n) – намораживаться, при этом сторона квадрата, в который вписана снежинка, увеличивается на 2 * n, одновременно добавляются звездочки в нужных местах, чтобыправило соблюдалось.

– thicken() – утолщаться, ко всем линиям звездочек с двух сторон добавляются параллельные (если перед этим снежинка таяла, то теперь звездочки восстанавливаются).

– show() – показывать (рисуется снежинка в виде квадратной

матрицы со звездочками идефисами в пустых местах). Класс «Robot»

Класс инициализируется начальными координатами – положением Робота наплоскости, обе координаты заключены в пределах от 0 до 100.

Робот может передвигаться на одну клетку вверх (N), вниз (S), вправо (E), влево (W).

Выйти за границы плоскости Робот не может.

Метод move() принимает строку – последовательность команд перемещения робота, каждая буква строки соответствует перемещению на единичный интервал в направлении, указанном буквой. Метод возвращает список координат – конечное положение Робота после перемещения.

Метод path() вызывается без аргументов и возвращает список координат точек, покоторым перемещался Робот при последнем вызове метода move. Если метод не вызывался, возвращает список с начальным положением Робота.

Класс «Темы» (Themes)

Экземпляру класса при инициализации передается аргумент – список тем дляразговора.

Класс реализует методы:

– add_theme(value) – добавить тему в конец;

– shift_one() – сдвинуть темы на одну вправо (последняя становится первой, остальныесдвигаются);

– reverse_order() – поменять порядок тем на обратный;

– get_themes() – возвращает список тем;

– get_first() – возвращает первую тему.

Пример 1

Ввод:

tl = Themes(['weather', 'rain']) tl.add_theme('warm') print(tl.get_themes()) tl.shift_one() print(tl.get_first())

Вывод:

('weather', 'rain', 'warm')warm

Пример 2

Ввод:

tl = Themes(['sun', 'feeding']) tl.add_theme('cool') tl.shift_one() print(tl.get_first()) tl.reverse_order() print(tl.get_themes())

Вывод:

cool

('feeding', 'sun', 'cool')

Класс «ПчёлоСлон» (BeeElephant)

Экземпляр класса инициализируется двумя целыми числами: первое относится кпчеле, второе – к слону.

Класс реализует следующие методы:

– fly() – может летать – возвращает True, если часть пчелы не меньше части слона, иначе

False;

– trumpet() – трубить – если часть слона не меньше части пчелы, возвращает строку:“tu-tu-doo-doo!”, иначе “wzzzzz”.

– eat(meal, value) – есть – может есть только нектар (nectar) или траву (grass). Если съедает нектар, то из части слона вычитается количество съеденного, пчеле добавляется, иначе наоборот: у пчелы вычитается, а слону добавляется. Не может увеличиться выше 100 и уменьшиться меньше 0;

– get_parts() – возвращает список из значений: [часть пчелы, часть слона].Пример 1

Ввод:

be = BeeElephant(3, 2) print(be.fly()) print(be.trumpet()) be.eat('grass', 4) print(be.get_parts()) Вывод:

True wzzzzz(0, 6)

Пример 2

Ввод:

be = BeeElephant(13, 87)print(be.fly()) print(be.trumpet()) be.eat('nectar', 90)

print(be.trumpet()) print(be.get_parts()) Вывод:

False

tu-tu-doo-doo! wzzzzz

(100, 0)

Класс «Разговор» (Talking)

Экземпляр класса инициализируется с аргументом name – именем котенка. Классреализует методы:

– to_answer() – ответить: котенок через один раз отвечает да или нет, начинает с да. Метод возвращает “moore-moore”, если да, “meow-meow”, если нет. Одновременно увеличивается количество соответствующих ответов;

– number_yes() – количество ответов да;

– number_no() – количество ответов нет.Пример 1

Ввод:

tk = Talking('Pussy') print(tk.to_answer()) print(tk.to_answer()) print(tk.to_answer())

print(f'{tk.name} says "yes" {tk.number_yes()} times, "no" {tk.number_no()} times')

Вывод:

moore-moore meow-meow moore-moore

Pussy says "yes" 2 times, "no" 1 times

Пример 2

Ввод:

tk = Talking('Pussy') tk1 = Talking('Barsik') print(tk.to_answer()) print(tk1.to_answer()) print(tk1.to_answer()) print(tk1.to_answer())

print(f'{tk.name} says "yes" {tk.number_yes()} times, "no" {tk.number_no()} times')

print(f'{tk1.name} says "yes" {tk1.number_yes()} times, "no"

{tk1.number_no()} times')

Вывод:

moore-moore moore-moore meow-meow moore-moore

Pussy says "yes" 1 times, "no" 0 times Barsik says "yes" 2 times, "no" 1 times