Лекция 11

Функции. Часть 2

Функции с параметрами по умолчанию

Иногда возникает необходимость в использовании параметра по умолчанию. Т.е. если при вызове такой параметр не указывается, то берется значение по умолчанию. Однако если при вызове его значение указывается, то будет использовано значение, указанное в качестве параметра.

Для этого в Python используется синтаксис вида

```
def  function_name (param_1, param_2, param_3 = Default_volume):
```

Значение по умолчанию

В данном примере для третьего параметра указано значение по умолчанию.

python

Пример написания функции с параметрами по умолчанию

Значение по умолчанию

```
def draw_rectangle(w, h, fill = "X"):
    for i in range(w):
        for j  in range(h):
            print(fill,end="")
        print()
    return

draw_rectangle(7, 5)

draw_rectangle(7,5,"@")
```

Третий параметр не указан, используется значение по умолчанию.

Третий параметр указан, используется его значение.

Использование именованных параметров

При необходимости можно использовать именованные аргументы при вызове функции. Т.е. использовать не позиционное присвоение аргументов, а именованное - использовать имена аргументов при вызове.

При вызове функции используется следующий синтаксис:

function_name (param_1 = value_1, param_2 = value_2)

В таком случае присвоение формальных параметров будет выполнено по именам, а не по позициям.

Пример использования именованных параметров

```
def print_line(w, fill):
    for i in range(w):
        print(fill,end = " ")
    return

print_line(6, "*")

print_line(fill = "+", w = 10)
```

Обычный вызов. Присвоение позиционное.

Именованный вызов. Присвоение по именам.

Использование переменного числа аргументов

В случае, когда нужно написать функцию, которая может принимать заранее неизвестное количество аргументов, можно использовать следующую возможность языка Python:

если при объявлении функции в качестве одного из аргументов указать параметр вида

*args

то все параметры, которые не попадают в именованные или позиционные, будут упакованы в кортеж с именем **args**.

Пример использования переменного числа аргументов

Позиционный параметр.

Остальные параметры будут упакованы

```
def get_class_list(teacher, *students):
    result = {"teacher":teacher,"student_list":[]}
    for student in students:
        result.get("student_list").append(student)
    return result
class_list_1 = get_class_list( "Petr Ivanovich","Nikolay","Olga","Bogdan")

print(class_list_1)
```

Первый параметр «Petr Ivanovich» будет записан в соответствующий позиционный параметр, а все остальные будут записаны в кортеж с именем students.

Пример использования переменного числа аргументов

Все аргументы упакуются в кортеж

```
def get_min(*args):
    min_element = args[0]
    for element in args:
        if min_element > element:
            min_element = element
    return min_element

a = get_min(0, 6, 2, -5)
print(a)
```

Для нахождения минимума среди произвольного количества аргументов используется упаковка этих аргументов в кортеж.

Использование переменного числа именованных аргументов

В случае когда, нужно написать функцию, которая может принимать заранее неизвестное количество именованных аргументов, можно использовать следующую возможность языка Python:

если при объявлении функции в качестве одного из аргументов указать параметр вида

**kwargs

то все параметры, которые не попадают в позиционные и будут вызваны как именованные, будут упакованы в словарь с именем **kwargs**.

Пример использования переменного числа именованных аргументов

```
def print_student(**kwargs):
    for key in kwargs.keys():
        print(key," -> " , kwargs.get(key))
    return

print_student(name = "Alexander", last_name = "Ts", age = 36,  specialty = "Physicist")
```

В этом случае все именованные параметры, которые используются при вызове функции, будут упакованы в словарь. В примере выполняется проход по ключам этого словаря.

Тонкости использования аргументов

При вызове функции аргументы должны указываться в следующем порядке:

1) любые позиционные аргументы (значения)
2) именованные аргументы (name=value)
3) аргументы в форме *sequence
4) аргументы в форме **dict.

При описании функции аргументы должны указываться в следующем порядке:

1) любые обычные аргументы (name)
2) аргументы со значениями по умолчанию (name=value)
3) аргументы в форме *sequence
4) любые имена или пары name=value аргументов, которые передаются только по имени
5) аргументы в форме **dict

Внимание!!! При нарушении такого порядка программа работать не будет.

Некоторые «трюки» с использованием параметров функции

Если нужно, то можно описать функцию так чтобы некоторые параметры могли быть вызваны только по имени и не могли быть вызваны позиционно.

Для этого все параметры, которые должны быть вызваны только по имени, должны быть описаны после параметра вида *arg.

Например:

```
def some_function (*arg, parametr_1):
```

Параметр **parametr_1** может быть вызван только по имени, так как все позиционные аргументы будут собраны в **arg**.

Распаковка кортежа в ряд фактических параметров

При необходимости можно выполнить эту задачу наоборот. Т.е. распаковать кортеж, который используется в качестве фактического параметра, в ряд формальных позиционных параметров.

Для этого нужно при вызове функции использовать синтаксис вида:

function_name(*sequence)

где sequence — кортеж. Он будет распакован в последовательность позиционных формальных параметров.

Пример распаковки кортежа в ряд фактических параметров

```
def draw_rectangle(w, h, fill):
    for i in range(w):
        for j  in range(h):
            print(fill,end="")
        print()
    return

crt = (7, 5, "#")

draw_rectangle(*crt)
```

Кортеж который будет распакован в ряд параметров

Распаковка кортежа

Распаковка словаря в ряд фактических параметров

При необходимости можно выполнить распаковку словаря в ряд формальных именованных параметров.

Для этого нужно при вызове функции использовать синтаксис вида:

```
function_name(**kwargs)
```

Где kwargs — словарь. Он будет распакован в последовательность формальных именованных параметров.

Пример распаковки словаря в ряд фактических параметров

```
def draw_rectangle(w, h, fill):
    for i in range(w):
        for j  in range(h):
            print(fill,end="")
        print()
    return

crt = {"w":5, "h":8,"fill": "#"}

draw_rectangle(**crt)
```

Словарь, который будет распакован в ряд параметров

Распаковка словаря

Некоторые вопросы

Эта тема выглядит довольно сложной. Часто ли применяют подобные примеры программирования?

Нет. Подобные приемы программирования применяются не очень часто. Однако их лучше знать, если в дальнейшем будете часто работать с чужим кодом.

Особенности использования функций

На самом деле оператор объявления функции **def**—выполняет ту же функцию, что и объявление переменной. А именно название функции - это имя переменной, а тело функции - это ее значение.

Следовательно, с функциями можно обращаться как с обычными переменными.

Т.е. можно одной переменной присвоить значение функции, и эта переменная также станет функцией.

Можно хранить функции в списке или, например, в словаре.

Пример работы с функциями как с переменными

```
def  func(x):
    return x**2

t = func

print(t(3))
```

Присвоение значения функции значению переменной

Теперь функцию можно вызвать и через переменную t

Пример работы с функциями как с переменными

```
def func (x):
    return 2 * x

def map_function(number_list, func):
    for i in range(len(number_list)):
        number_list[i] = func(number_list[i])
    return

number_list = [1, 2, 3, 4]

map_function(number_list, func)

print (number_list)
```

Функция как параметр для другой функции

Особенности использования функций

```
def func (x):
    return 2 * x

def func(x):
    return x*3

print(func(6))
```

Как и в случае с переменными, если объявить функцию с одним и тем же именем, то старое значение будет заменено на новое. Поэтому вызов функции приведет к выполнению последнего варианта.

В примере будет выполнено умножение на 3.

Список использованной литературы

1) Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011 - 505 — 525 с.
2) Саммерфилд М. Программирование на Python 3. Подробное руководство. - Пер. с англ. - СПб.:Символ-Плюс, 2009. - 210 — 214 с.

Домашнее задание

1) Напишите функцию, которая вернет сумму всех нечетных элементов списка. Например, для списка вида [2,1,4,6,3] ваша программа должна вернуть 4.
2) Напишите функцию, которая будет рисовать на экране треугольник заданной высоты. Символ, которым рисуется треугольник, также должен быть параметром этой функции. Например, если в качестве параметра задать величину высоты равную 5, а символ равным «#», то результат должен быть такой:
   #
   ##
   ###
   ####
   #####
3) Напишите программу, которая попросит ввести пользователя его имя и возраст. После проверки возраста на корректность (чтобы не был меньше 0 и больше 150) выведите имя человека и его возрастную градацию:
   0-10 — детский возраст, 10-25 - юный возраст, от 25 до 44 лет - молодой возраст, 44-60 лет - средний возраст, 60-75 лет - пожилой возраст, 75-90 лет - старческий возраст, а после 90 - долгожитель.

python

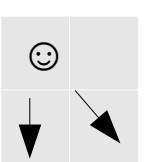
Дополнительное домашнее задание.

1) Существует список [1,2,3,4,5]. Размер списка может быть произвольным, однако заполнен он всегда цифрами от 1 и далее по возрастанию с шагом 1. Напишите программу, которая выведет на экран все возможные комбинации, которые могут быть получены перестановкой элементов этого списка. Внимание! Повторений и пропущенных комбинаций быть не должно.

2)

☺							
7							
5	8						
9	8	2					
1	3	5	6				
6	2	4	4	5			
9	5	3	5	5	7		
7	4	6	4	7	6	8	

На вершине горы стоит золотоискатель (смайлик наверху). Цифра в клетке обозначает количество золотых слитков, которые в ней хранятся. Золотоискатель может за один раз или спуститься на одну клетку вниз, или спуститься на одну клетку вниз и вправо.



Напишите программу, которая определит максимальное количество золотых слитков, которые может найти золотоискатель, двигаясь от вершины горы вниз.