python_lecture_04_functions

February 13, 2020

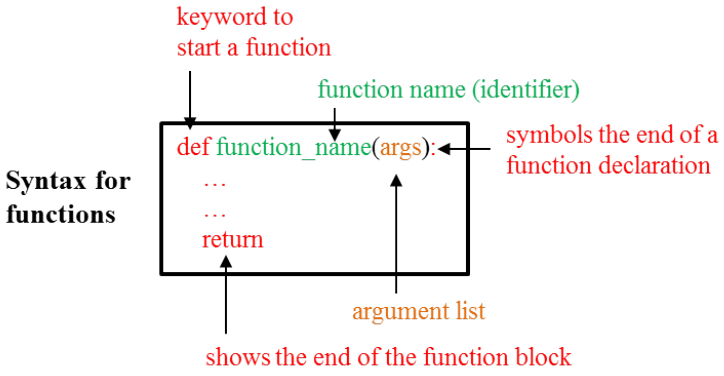
1 Programming with Python

2 4 Functions

Functions are subroutines that perform a sequence of code when they are called. A function is written outside the main program and called on necessity.
The main objective of a function is to keep your main code **clean** and to make code sequences **callabe** from any point of the code. Therefore, repetitive code sequences are avoided.

Creating a function

Every function starts with a keyword **def** followed by a space, a **function name**, **round brackets** and a **colon**. The round brackets **may** contain arguments (strings!) which are values that are passed to the function.
The end of function is expressed by a **return** statement.



syntax for functions

The shortest (and probably most useless) function may look like this:

[1]:
```
def function_name():
    return
```

Pass values

Usually, functions work with values that need to be passed to the function. The passed values are called **arguments**.

[2]:
```
def print_list_of_names(names):
    for name in names:
```

```
        print(name)
    return
```

Return values

Functions can return values for the further processing in the main programm.
The values to be returned are placed after the return statement.

[3]:
```
# add function
def add(x,y):
    result = x + y
    return result
```

Call a function

A function is called by its name followed by round brackets with arguments if needed.

[4]:
```
x = 5
y = 6

z = add(x,y) # The return value is assigned to the variable z

print('the sum of', x, 'and', y, 'is', z)
```

```
the sum of 5 and 6 is 11
```

[5]:
```
# give the sum of a list of numbers
def sum_of_list(list_of_numbers):
    result = 0
    for number in list_of_numbers:
        result = result + number
    return result

some_numbers = [1,5,6,19]
print('the sum of', some_numbers, 'is', sum_of_list(some_numbers))
```

```
the sum of [1, 5, 6, 19] is 31
```

Call functions within function

It is possible to call a function within a function, e.g. to expand existing functionality. This way, old code can be recycled.

[6]:
```
def sum_of_list2(list_of_numbers):
    result = 0
    for number in list_of_numbers:
        result = add(result, number)
    return result

some_numbers = [1,2,6,19]
print('the sum of', some_numbers, 'is', sum_of_list2(some_numbers))
```

```
the sum of [1, 2, 6, 19] is 28
```

2.0.1 Excursion to augmented operators

Some **arithmetic operators**:

Arithmetic operators in Python
Operator
Meaning
Example
+
Add two operands or unary plus
x + y
-
Subtract right operand from the left or unary minus
x - y
*
Multiply two operands
x * y
/
Divide left operand by the right one (always results into float)
x / y
%
Modulus - remainder of the division of left operand by the right
x % y (remainder of x/y)
//
Floor division - division that results into whole number adjusted to the left in the number line
x // y
**
Exponent - left operand raised to the power of right
x**y (x to the power y)

In **boolean operators** or comparison operators:

Comparision operators in Python
Operator
Meaning
Example
>
Greater that - True if left operand is greater than the right
x > y
<
Less that - True if left operand is less than the right
x < y
==
Equal to - True if both operands are equal
x == y
!=
Not equal to - True if operands are not equal
x != y
>=
Greater than or equal to - True if left operand is greater than or equal to the right
x >= y

<=
Less than or equal to - True if left operand is less than or equal to the right
x <= y
Some **assignment operators** operators:
Assignment operators in Python
Operator
Example
Equivatent to
=
x = 5
x = 5
+=
x += 5
x = x + 5
-=
x -= 5
x = x - 5
*=
x *= 5
x = x * 5
/=
x /= 5
x = x / 5
%=
x %= 5
x = x % 5
//=
x //= 5
x = x // 5
**=
x **= 5
x = x ** 5
Some **logical operators**:
Logical operators in Python
Operator
Meaning
Example
and
True if both the operands are true
x and y
or
True if either of the operands is true
x or y
not
True if operand is false (complements the operand)
not x

2.0.2 Back to functions

[7]:
```
# get the arithmetic mean of a list of numbers
def arithmetic_mean(list_of_numbers):
    result = 0
    for number in list_of_numbers:
        result += number # += is a abbreviation for result = result + number
    result /= len(list_of_numbers) # the abbreviation works for division--> /=,␣
 ↪plus --> +=, minus --> -=, divide --> /=
    return result

some_numbers = [1,2,6,19, 5.0, 17.4]
print('the arithmetic mean of', some_numbers, 'is',␣
 ↪arithmetic_mean(some_numbers))
```

```
the arithmetic mean of [1, 2, 6, 19, 5.0, 17.4] is 8.4
```

[8]:
```
# get the median of a list of numbers
def median(list_of_numbers):
    sorted_list = sorted(list_of_numbers)
    if len(sorted_list)%2 == 0: # even number of list elements
        i = int(len(sorted_list)/2)
        j = int(len(sorted_list)/2+1)
        median = (sorted_list[i]+sorted_list[j])/2
    else: # odd number of list elements
        i = int(len(sorted_list)/2 + 0.5)
        median = sorted_list[i]
    return median

some_numbers = [1,2,6,19]
some_numbers2 = [1,2,6,19,35]

print('the meadian of', some_numbers, 'is', median(some_numbers))
print('the meadian of', some_numbers2, 'is', median(some_numbers2))
```

```
the meadian of [1, 2, 6, 19] is 12.5
the meadian of [1, 2, 6, 19, 35] is 19
```

[9]:
```
# get the crossfoot of a number
def crossfoot(number):
    """The function give the sum of the single digits of an integer number"""
    list_of_char = list(str(number)) # list(str) converts a string into a list␣
 ↪of characters containing one character per list element
    result = 0
    for element in list_of_char:
        result += int(element)
    return result
```

```
some_number = 12883
crfoot = crossfoot(some_number)
print('the crossfoot of', some_number, 'is', crfoot)
```

```
the crossfoot of 12883 is 22
```

[10]:
```
# crossfoot as one-liner using list comprehension
some_number = 1288

crfoot = sum([int(s) for s in list(str(some_number))])

print('the crossfoot of', some_number, 'is', crfoot)
```

```
the crossfoot of 1288 is 19
```

[11]:
```
# you can assign the return value of a function to variable
some_number = crossfoot(12883)
print(some_number)
```

```
22
```

Return multiple values

Seperate the return values with a comma for **multiple return values**

[12]:
```
# get the crossfoot and the length of a number
def crossfoot_and_digits(number):
    """The function give the sum of the single digits as well as
    the number digits"""
    crfoot = 0
    digits = 0
    crfoot = crossfoot(number)  # the crossfoot function is recycled here
    digits = len(str(number))
    return crfoot, digits


some_number = 12546845
crfoot, digits = crossfoot_and_digits(some_number)
print('the crossfoot of', some_number, 'is', crfoot, 'and has', digits,␣
 ↪'digits')
```

```
the crossfoot of 12546845 is 35 and has 8 digits
```

[13]:
```
x = 23656225234
a, b = crossfoot_and_digits(x)
print('the crossfoot of', some_number, 'is', a, 'and has', b, 'digits')
```

```
the crossfoot of 12546845 is 40 and has 11 digits
```