Lesson 6 Python Function and Module

n this lesson, functions and modules of Python will be explained combining the related routine.

1. Function Introduction

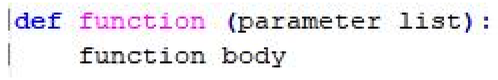
In Python, function is divided into built-in function and user-defined function.

Built-in function that readily comes with Python can be directly called without importing any function libraries. The common built-in functions include print(), input(), range(), etc.

User-defined function will define a piece of regular and reusable code as function to elevate the reuse rate and maintenance of the codes.

1.1 Function Definition

In Python, function generally is composed of function name, parameter list and function body consisting of series of statements. And the format is as follow.



```
def function (parameter list):
    function body
```

1） A block of code starts with keyword “**def**”, and function identifier name and parameter list follows.

2） For the convenience of future maintenance, it is better to reflect the function of the **function** on function identifier name.

3） Parameter list is used to set the parameters that can be received by the function, and the parameters can be separated by “**,**”.

4） Any passed in parameters and variables should be inserted inside the parenthesis.

5） Functions should start with colon “**:**” and are indented strictly.

6） The first line of statement in function can be docstring, i.e. function description.

7） Functions have returned value which is “**None**” by default.

Note: when building function, the parenthesis behind the function name cannot be omitted though there is no parameter in the function.

1.2 Parameter and Argument

When defining function, we should adopt the parameter, while when calling the function, we adopt argument. They both work to pass the data.

1） parameter

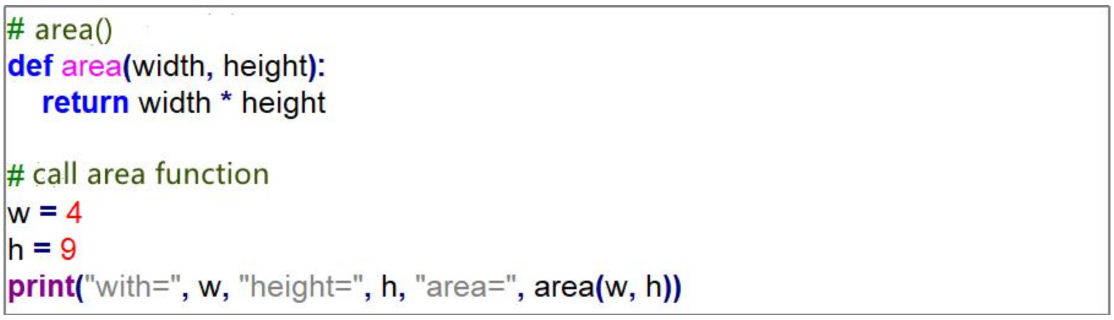
Parameters are not actual variables, so they are also called virtual variable. Parameters are adopted when defining function name and function. They are used to receive the parameters passed in when the function is called, and their scope is generally limited to the inside of the function body.

2） argument

Arguments are the parameters passed to the function when calling the function. They can be constants, variables, expressions and functions, and their scope is based on the actual setting.

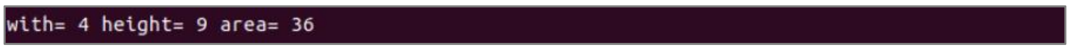
Regardless of the types, all arguments must have certain values when calling the function so that these values can be sent to parameter.

Take the code below for example. “**width**” and “**height**” are parameters, while “**w**” and “**h**” defined outside the function body are arguments.



```python
# area()
def area(width, height):
    return width * height

# call area function
w = 4
h = 9
print("with=", w, "height=", h, "area=", area(w, h))
```



```
with= 4 height= 9 area= 36
```

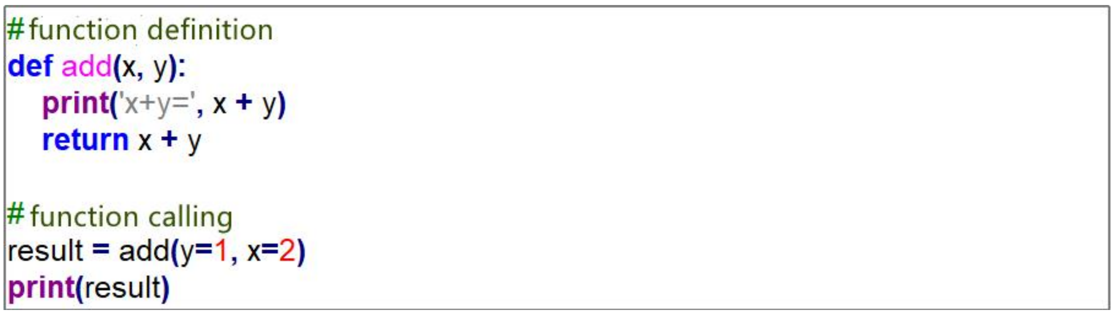
After parameter “**w**” and “**h**” are passed into the function body, “**width**” and “**height**” parameters are assigned to the corresponding values.

1.3 Return Value

After the function is executed, system will feed back some value to external caller, and these values are considered as return values of functions.

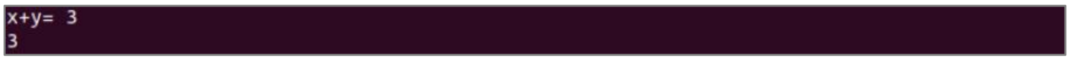
In Python, when function runs to return statement, it means that the function completes running and designated values will be returned. If there is no return statement inside the function, function will return “**None**” by default.

Take the code below for example. After **add()** function is called, this function will return the result of “**x+y**”, and assign variable “**result**” to this value.



```python
#function definition
def add(x, y):
    print('x+y=', x + y)
    return x + y

#function calling
result = add(y=1, x=2)
print(result)
```



```
x+y= 3
3
```

Pay attention, in Python, there can be multiple return values in one function as the picture below shown.

```python
# function definition
def calculate(x, y):
    print('x+y=', x + y)
    print('x*y=', x * y)
    return x + y, x * y

# function calling
a, b = calculate(y=1, x=2)
print(a, b)
```

```
x+y= 3
x*y= 2
3 2
```

There are two return values inside the **calculate()** function, including “**x+y**” and “**x*y**”. After this function is called, “**a**” and “**b**” variables will be assigned to these two return values.

2. Function Passing

There are two types of function parameters, including mutable and immutable, whose calling results are different.

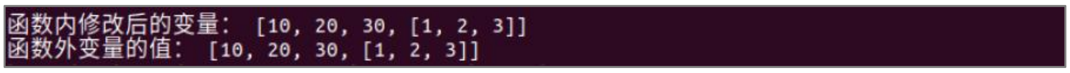
2.1 Mutable Parameter

The calling of mutable parameter is similar to pass-by-reference in C++. If the mutable parameter is passed, such as list and dictionary, modification of the passed in parameters inside the function will affect the external variables.

For example, after the passed in list “**list_01**” is changed inside **change_int()** function, the external variable will also be changed.

```python
def change_int(list_01):
    list_01.append([1, 2, 3])  # change the passed-in list
    print("函数内修改后的变量：", list_01)

list_01 = [10, 20, 30]
change_int(list_01)
print("函数外变量的值：", list_01)
```



```
函数内修改后的变量： [10, 20, 30, [1, 2, 3]]
函数外变量的值： [10, 20, 30, [1, 2, 3]]
```

2.2 Immutable Parameter

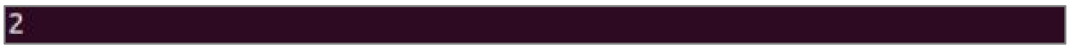
Calling immutable parameters is similar to C++ pass-by-value. If immutable parameters are called, such as integer, string and tuple, the modification of the passed in parameter inside the function will not affect the external variable.

Take code below for example. The variable "**b**" points to the int object "**2**". When passed to the **unchange_int()** function, the variable "**b**" is copied by value, that is, the variable "**a**" and the variable "**b**" both point to the same int object "**2**" .

However, when execute "**a = 10**", variable "**a**" points to newly generated int object "**10**". Therefore, the external variable doesn't change, and the printed value is "**2**".

```
def unchange_int(a):
    a = 10

b = 2
unchange_int(b)
print(b)
```



```
2
```

3.Parameter Type

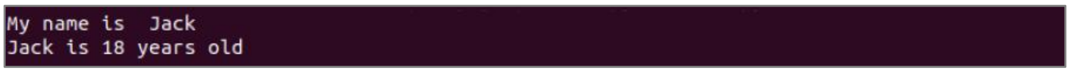
3.1 Positional Parameter

When calling function, each argument is associated with the corresponding parameter in positional order, and this association is called a positional parameter.

Take the code below for example. When calling **describe_student()** function, name and age parameters should be offered in sequence. “**Jack**” and “**18**” are respectively stored in “**person_name**” and “**student_age**”.

```
def describe_student(person_name, student_age):
    print("My name is ", person_name)
    print(person_name + " is " + student_age + " years old")

describe_student('Jack', '18')
```



```
My name is  Jack
Jack is 18 years old
```

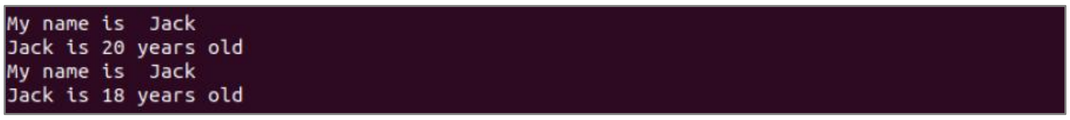
3.2 Default Parameter

When defining functions, we can specify the default value of each parameter. When calling function, if arguments are provided to parameters, adopt the designated argument. Otherwise, the default value of parameter should be adopted.

For example, set the default value of “**student_age**” parameter as “**18**”. When argument is offered to “**student_age**” parameter during calling describe_student() function, adopt the designated argument. And the designated argument of this example is “**20**”.

```
def describe_student(person_name, student_age='18'):
    print("My name is ", person_name)
    print(person_name + " is " + student_age + " years old")

describe_student('Jack', '20')
describe_student('Jack')
```

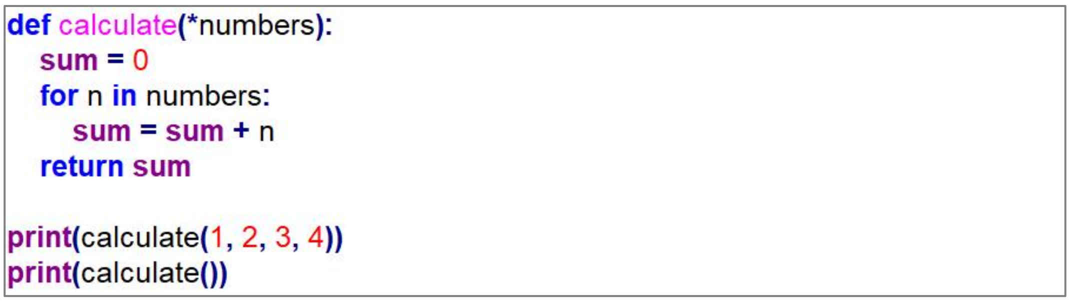


```
My name is  Jack
Jack is 20 years old
My name is  Jack
Jack is 18 years old
```

3.3 Variable-length Parameter

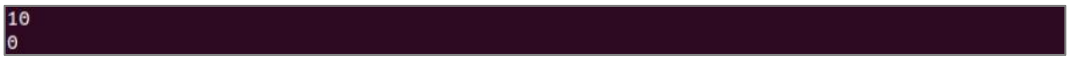
In Python, function can also be defined as variable-length parameter which is also called mutable parameter. By adding “*” in front of the identifier, the corresponding parameter can be defined as variable-length parameter.

Take the codes below for example. After parameter “**number**” is defined as variable-length parameter, the called **calculate()** function can be directly used even though the passed in parameters are neither list nor tuple.



```python
def calculate(*numbers):
    sum = 0
    for n in numbers:
        sum = sum + n
    return sum

print(calculate(1, 2, 3, 4))
print(calculate())
```

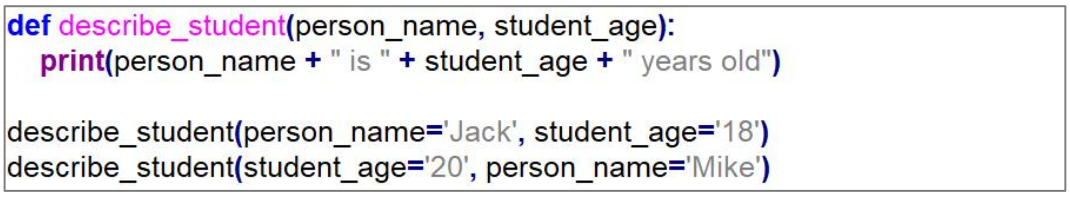


```
10
0
```

3.4 Keyword Parameter

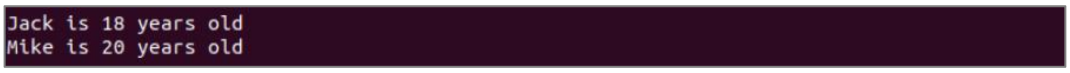
Keyword parameter will be passed by “**parameter name-value**” pair. In this way, when designating the argument of function, the position of argument and parameter can be different and we just need to ensure the parameter name is correct.

Check the code below. When passing the parameter, the function can output normally though the parameter order is adjusted.



```python
def describe_student(person_name, student_age):
    print(person_name + " is " + student_age + " years old")

describe_student(person_name='Jack', student_age='18')
describe_student(student_age='20', person_name='Mike')
```

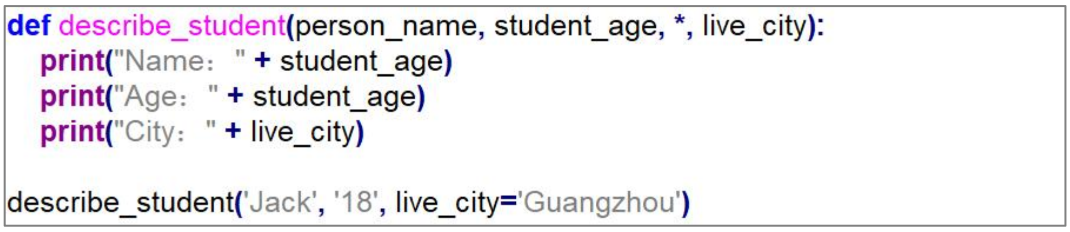


```
Jack is 18 years old
Mike is 20 years old
```

3.5 Named Keyword Argument

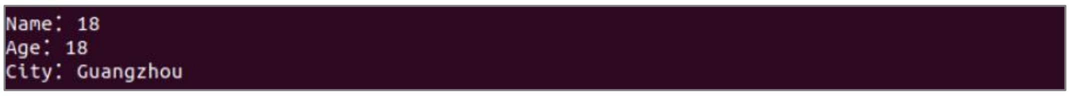
Named keyword arguments can be used when it is necessary to restrict parameters to be passed only by keyword. In user-defined function, parameters are separated by "*" and the parameters following "*" are named keyword parameter.

Take the codes below for example. "**live_city" following** "*" is named keyword parameter. Therefore, the parameters must be passed by keyword, otherwise the program will throw error.



```
def describe_student(person_name, student_age, *, live_city):
    print("Name：  " + student_age)
    print("Age：  " + student_age)
    print("City：  " + live_city)

describe_student('Jack', '18', live_city='Guangzhou')
```



```
Name： 18
Age： 18
City： Guangzhou
```

Note: In Python, the parameters should be defined in order that is positional parameter, default parameter, mutable parameter, named keyword parameter and keyword parameter.

4. Module Introduction

In general, module is a file suffixed by "**.py**". In addition, there is other file type of module, such as "**.pyo**", "**.pyc**", "**.pyd**", "**.so**" and "**.dll**". If you are Python novice, you can skip these types.

Package the code that can realize specific function as an independent module, which is convenient for other programs and scripts to be imported and used, and avoid the function name from overlapping the variable name.

Modules comes from three channels.

1） Python built-in modules (standard library)

2） the third-party module

3） User-defined module

When you need to call the function of a module, import this module through import statement before calling the function. There are two ways to import the module. (content inside the square parenthesis can be omitted)

1） import **module name** [as alias]

All members of the designated module will be imported when we use the import statement in this syntax. When you need to use the members of the module, this module name should be used as the prefix of the member name.

2） from **module name** import **member name**[as alias]

When the import statement in this syntax is adopted, only the designated member of the module will be imported but not all members. At the same time, when this member is used in the program, there is no need to add any prefix to the member name.

Note: “from module name import *” can also be used to import all the members of the designated members, but it is not recommended to use this.
