A function is a piece of packaged, reusable code that makes our programs more modular and eliminates the need to write a lot of repetitive code.

The function can be saved in advance, given a unique name, and used as long as the name is known. The function can also receive data, perform different actions based on the data, and finally give us the result of processing.

For example, you might define a piece of code in your program that implements a specific function. The question is, if you need to implement the same functionality next time, do you want to copy the code you defined earlier? Doing so would be silly, and would mean copying the previously defined code every time the program needed to implement the function. The right approach is to define the code that implements a particular function as a function that is executed (called) every time the program needs to implement that function.

The definition of Python functions

```
def function_name (parameter_list):
    // Multiple lines of code that implement a specific function
    [return value]
```

The part enclosed by `[]` is optional, which can be used or omitted.
In this format, the meanings of parameters are as follows:
`function_name`: This is an identifier that conforms to Python syntax, but it is not recommended to use simple identifiers like a, b, and c for the function name. It is best to use a function name that reflects the function's function (For example, `string_length` is a good function name).
`parameter list`: Sets how many arguments this function can accept, separated by commas (,).
`[return]` : As an optional parameter to a function, used to set the return value of the function. In other words, a function can have a return value or no return value, depending on the actual situation.

What is return value?

What is a return value?
After a function has finished executing, the result needs to be returned. Like a word problem

on a math paper, when you've done the math, you write, "Answer: The price of the shirt is nine pounds fifteen pence." The function of the return value is to tell the other person that I calculated this amount and you can just take it and use it.

Note: When creating a function, you must keep an empty pair of "()" even if the function does not require parameter, otherwise the Python interpreter will prompt an "invaild syntax" error.

This is an example:

```
def get_y_value(x):
    y = 5*x**3 + x**2 + x-3
    return y
```

Calling functions

Calling a function is executing a function. If you think of the function you create as a tool for some purpose, then calling the function is equivalent to using that tool.

The basic syntax of a function call is as follows:

```
[return_value =] function_name ([parameter_value])
```

The `function_name` refers to the name of the function to be called.
`Parameter_values` are the values of the parameters that were passed in when the function was originally created.
If the function returns a value, we can either accept it through a variable or not.

Note:

1. When calling a function, you must keep an empty pair of "()" even if the function does not require parameter.
2. As many parameters as the function has to be created, the call must pass in as many values as the function was created.

Python function value passing and

reference passing (including the difference between formal and actual arguments)

When using a function, it is common to use formal arguments ("parameters"for short) and actual arguments ("arguments" for short). The difference between them is:
Formal arguments: When defining a function, the arguments in brackets following the function name are formal arguments. For example:

```
# When defining a function, the "obj" is the formal argument
def demo(obj):
    print(obj)
```

Actual arguments: When calling a function, the arguments in parentheses following the function name are called actual arguments, which are the arguments given to the function by the function caller.
For example:

```
# Call the defined demo function, the "a" is the actual argument
a = 10
demo(a)
```

The difference between arguments and parameters is like choosing the lead actor in a play. The characters in the play are the parameters, and the actors who play the roles are the arguments.

In Python, according to the actual argument type, the function argument can be passed in two ways: value passing and reference (address) passing.
Value passing: applies to arguments of immutable type (string, number, tuple);
Passing by reference (address) : applies when the argument type is mutable (list, dictionary);

The difference between value passing and reference passing is that if the value of the parameter changes after the function parameter is passed, the value of the argument will not be affected; As function arguments continue to be passed by reference, the values of arguments change as well as the values of parameters.

For example, define a function called demo that passes a variable of type string (for value passing) and a variable of type list (for reference passing):

```python
def demo(obj) :
    obj += obj
    print("parameter is:",obj)
print("----Value passing-----")
a = "hello world"
print("The value of a is:",a)
demo(a)
print("Argument is:",a)
print("-----Reference passing-----")
a = [1,2,3]
print("The value of a is:",a)
demo(a)
print("Argument is:",a)
```