# 1、函数基础

##### 定义函数：

def function\_name(parameters):  
 # 函数体（包括多个语句）  
 return result

x0;在定义函数的过程中，需要注意以下几点：

* 函数代码块以def关键词开头，一个空格之后接函数标识符名称和圆括号()，再接个冒号。

* 任何传入的参数必须放在圆括号中间。

* 函数的第一行语句后可以选择性的使用文档字符串用于存放函数说明。

文档字符串(Docstring)是一种用于记录和描述函数、类、模块或方法的字符串。文档字符串通常包含有关特定代码块的说明、用法示例以及相关文档信息。文档字符串的主要目的是提供对代码块的文档化和解释，使其容易理解和使用，同时有助于生成文档与自动化测试。

文档字符串通常位于函数、类、模块的开头、紧随定义之后，并使用三重引号（单引号或双引号）括起来。例如：

def add(a, b):  
 """  
 This function takes two numbers as input and returns their sum.  
  
 Parameters:  
 a (int or float): The first number.  
 b (int or float): The second number.  
  
 Returns:  
 int or float: The sum of a and b.  
 """  
 return a + b

文档字符串通常包括以下内容：

1. 描述函数或代码的目的和功能。

1. 参数说明：描述函数的输入参数、包括参数名称、类型及意义。

1. 返回值说明：包括返回值的的类型及含义

1. 使用示例：提供一个或多个示例，以展示如何使用函数

文档字符串是一种非常有用的文档工具，因为它们可以被提取和解析，来生成文档，例如使用工具Sphinx等其他自动生成文档的工具，此外文档字符串还可以通过内置函数“help()”来访问，以获取有关函数、类和模块的信息。

help(add) # 运行查看文档字符串  
  
# todo：help内置函数查看文档字符串输出  
# add(a, b)  
# This function takes two numbers as input and returns their sum.  
#   
# Parameters:  
# a (int or float): The first number.  
# b (int or float): The second number.  
#   
# Returns:  
# int or float: The sum of a and b.

* x0;函数内容以冒号起始，并且缩进。

* 使用return结束函数。默认返回None。

* 若return语句依然在函数体内部，不能回退缩进。直到函数所有的代码写完，才回退缩进，表示函数体结束。

##### 函数调用

函数编写出来就是为了调用。使用函数名后跟圆括号的方式来调用函数。调用函数的同时要根据函数的定义体，提供相应个数和类型的参数，每个参数之间用逗号分隔。Pyhton由于动态语言的特点，在做语法与词法分析检查的时候，并不会对参数类型进行检查，但是在执行的过程中，如果参数类型不符合内部运行机制的话，会弹出相应的错误，如：

# todo：参数类型错误  
sum("add", "uij")  
报错：  
Traceback (most recent call last):  
 File "/Library/Pycode/PyStudy/Base/函数基础.py", line 56, in <module>  
 sum("add", "uij")  
TypeError: sum() can't sum strings [use ''.join(seq) instead]

Pyhton内置函数sum要求必须提供一个如列表、集合、元组这样的可迭代对象。

sum(iterable[, start])  
Parameters:  
# iterable -- 可迭代对象，如：列表、元组、集合。  
# start -- 指定相加的参数，如果没有设置这个值，默认为0。  
"""

##### return 语句：

return语句用于表示函数执行到结束，并且返回return后面的对象。如果函数不需要返回任何值的时候可以没有return语句，它会默认给你返回None，并且不会给任何提示。一旦函数执行的过程中遇到return语句，那么在与return同等缩进代码块中的所有代码都会被忽略，直接跳出函数体。

def func(\*args):  
 pass  
 return  
 # 此时，后面的代码是不会运行的  
 # 但从语法与词法的层面上，这些没有错误  
 print("test")

return可以返回什么：

* 什么都不返回，仅仅return：return

* 数字/字符串/任意数据类型：return 'hello world!'

* 一个表达式：return a+b

* 一个判断语句：return a>b

* 一个变量：return a

* 一个函数：return func

* 一个函数的调用：return func()

* 返回它自己：return self

* 多个返回值，用','隔开：return a, a+b, "hello"

简而言之，函数可以return出任意Pyhton对象。

**函数返回值的接受**

将函数返回值保存在变量中：

# todo: 函数返回值的接收  
def func\_02(\*args):  
  
 return sum(\*args)  
  
  
result = func\_02([1, 2, 3])  
print(result)

对于返回多个值的函数，需要对应相应个数的变量来接收，变量之间以逗号分隔。否则会以元组的形式进行接收。

# todo: 多个变量接收返回的多个返回值  
def func\_03(\*args):  
 s = sum(\*args)  
 a = [i\*i for i in [i for i in args][0]]  
  
 return s, a  
  
  
c, d = func\_03([2, 3, 4])  
print(c, d)  
# todo:output  
# 9 [4, 9, 16]  
# (9, [4, 9, 16])单变量接受

##### 参数的传递

函数通常都有参数，用于将外部的实际数据传入函数内部进行处理。但是，在处理不同数据类型的参数时，会有不同的情况发生。主要源于以下两点。

1. Python的函数参数传递的是实际对象的内存地址。

1. Python的数据类型分为可变数据类型与不可变数据类型。

看下面示例：

# todo: 参数的传递  
a = 1  
  
  
def func\_04(b):  
 print("在函数内部修改之前，变量b的内存地址为：{}".format(id(b)))  
 b = 2  
 print("函数内部修改之后，变量b的内存地址为：{}".format(id(b)))  
 print("函数内部的b为: {}".format(b))  
  
  
print("函数调用之前，变量a的内存地址为{}".format(id(a)))  
func\_04(a)  
print("函数外部的a为：{}".format(a))

运行结果：

函数调用之前，变量a的内存地址为4358978936

在函数内部修改之前，变量b的内存地址为：4358978936

函数内部修改之后，变量b的内存地址为：4358978968

函数内部的b为: 2

函数外部的a为：1

为什么在b=2后，函数内的b与函数外的a的内存地址不同了呢？即在此之后a与b是两个不同的对象了。这是因为当作为参数，a被传入函数时，将数字对象1的地址传递给了函数内部的b。执行第一句代码时，此时函数内部的b与函数外部的a其实是一个对象，因此打印出同样的内存地址。而当b=2被执行后，创建一个新的变量b，并赋值2，将数字对象2的内存地址赋值给了变量b。众所周知，赋值语句具有创建新变量的功能，其次，函数由于具有作用域的概念，在其内部的变量不受外部变量的影响，可以独立使用变量，相当于一个新的命名空间。因此此时的内部变量b与外部变量a已经没有任何关系了，是两个不同的变量。同时，由于数字1与2是不可变的数据类型对象，是两个独立的，不同内存地址的对象，因此再次打印内存地址，自然就不一样。若a=2则a与b的内存地址相同，即将存放在a变量中的2的内存地址赋值给变量b，在Pycharm中处于对性能的考虑，对于不可变对象，相同对象时，就直接引用已经存在的对象。

上述变量a为不可变的数据类型对象，如果是可变的数据类型对象，则结果又有不同：

a = [1, 2, 3]  
  
  
def func\_04(b):  
 print("在函数内部修改之前，变量b的内存地址为：{}".format(id(b)))  
 b.append(4)  
 print("函数内部修改之后，变量b的内存地址为：{}".format(id(b)))  
 print("函数内部的b为: {}".format(b))  
  
  
print("函数调用之前，变量a的内存地址为{}".format(id(a)))  
func\_04(a)  
print("函数外部的a为：{}".format(a))

输出：

函数调用之前，变量a的内存地址为4353894656

在函数内部修改之前，变量b的内存地址为：4353894656

函数内部修改之后，变量b的内存地址为：4353894656

函数内部的b为: [1, 2, 3, 4]

函数外部的a为：[1, 2, 3, 4]

调用函数时将列表对象a的内存地址传递给了函数内部的变量b。b.append(4)时，根据传进来的内存地址，找到[1, 2, 3]这个列表对象，并在它的后面添加4。

可以看出，此时a和b实际指向了同一个对象。这是因为b.append(4)这句代码，它不同于"="赋值语句，不会创建新的变量，而列表作为可变类型，具有append方法，这种方法只是对列表的一种调用而已。

**parameters 和arguments 之间的区别是什么?**

许多人交替使用这些术语，但它们是有区别的：

* 1. Parameters 是函数定义中定义的名称

* 1. Arguments是传递给函数的值

# todo: parameters与arguments的区别  
def the\_func(arg1: str, arg2: str):  
 print(f'{arg1} {arg2}!')  
  
  
the\_func('Hello', 'world')

x0;上述代码中的arg1、arg2是parameters，'Hello'、'world'为arguments