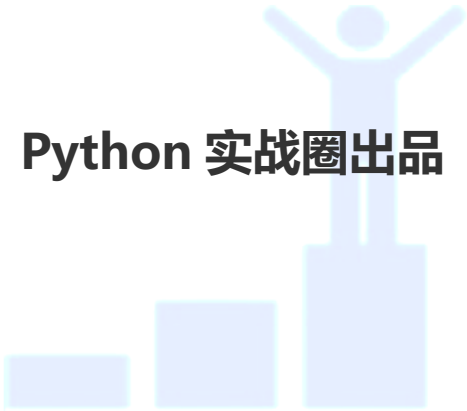
Python 函数，
给你不一样的介绍

Python 实战圈出品



什么是函数

函数是一个独立且封闭完成特定功能的代码块，可以在任何地方被调用。比如第二天内容里面的 print()函数，无论你在程序中的任何地方调用，都是输出（ ）中的内容。这种独立的封闭代码块又称为封装，也可以把函数理解为一个盒子。盒子里面的代码就是封装好的，完成特定的功能，外面的代码不属于函数。

在 python 中，函数分为内建函数和用户自定义函数。用户自定义函数是 python 程序语言已经给我们创建好了，直接可以使用，这一类的函数很多，我们不需要每一个都记住名字，常用的记住即可。官方文档 https://docs.python.org/3/library/functions.html 给出了很多内建函数，如下表。

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

用户自定义函数就是自己根据需要实现的功能设计代码块。本节内容就是介绍如何创

建属于自己的函数，完成特定的功能。

有了封装以后，我们就可以把代码块用一个个函数表示，然后通过一个个的调用就可以把复制的功能拆解，简单化。这个过程就是面向过程的程序设计，与之对应的是面向对象的程序设计（最后一天的内容）。在面向过程的程序设计里面，函数是基本单元。

为什么要用函数

无论任何编程语言，函数的作用都是一样的，概括起来主要有以下几个作用：

第一个：解决代码重复问题

在实际项目中 ，我们经常会遇到代码功能一样，但是参数不同。也就是给定不同的参数，得出不同的结果，比如 print（‘你好’）与 print('吃饭了吗？')，功能都是打印信息。我们不需要为“你好”，“吃饭了吗”编写同样的打印功能代码，也就是实现 print 函数。只需要用同一个函数（print）就可以完成同样的任务。在代码中多次执行同一项任务时，无需反复编写该任务的代码，只需要调用该任务名称的代码块，也就是函数即可完成任务。

第二个：代码结构与思维结构一致

实际生活中，解决问题的方法一般是分步骤，比如考大学，先从小学开始读，然后初中、高中。但是每一个阶段学的内容不同。我们只需要把每一个阶段编写为一个函数，然后按照我们思考的步骤组织在一起。有人认为从高中开始，再到小学；也有认为从小学到高中，按照自然规律。这个就是思维方式的不同。在编程中，这些都是正确的，没有对错之分，能组合在一起完成功能即可。

例子：

```
'''
考上大学之前，我们需要经历的阶段有：
'''
#小学
primary_school()
#初中
junior_middle_school()
#高中
senior_middle_school()
```

第三个：利于分工合作

在实际项目中，要实现的功能往往比较复杂。我们只需要按照功能拆解的方法思

考问题。把一个大问题，拆解为几个小问题，每一个小问题就是一个函数。然后再把所有函数按照思考的过程组织在一起即可。拆解问题的事情一般是公司架构师做的事情，定义好每一个函数的功能以及函数的接口，然后分配给不同的程序员去完成。这个就是多人协作完成一个大的功能。

第四个：代码清晰、易读、易修改

代码被函数组织起来以后，整个程序文件变得有调理、有章法。我们只需要按照函数的结构阅读代码即可，非常清晰明了。

每一个函数完成特定的功能。如果某一个出错了，我们只需要调整该函数即可，方便快捷，尤其是在代码行数比较多的时候能快速定位到错误所在的位置。

如何定义函数

在 Python 语言中，函数的定义语法是：

```
#定义函数语法
def funtion_name(parameter1,parameter2,xxxxx):
   语句 1
   语句 2
   return xxxx
```

- **def：为关键字，告诉 python 这是一个函数**
- **funtion_name：是函数名字。函数是完成特定功能的代码块，给这个代码块取一个名字就是函数名，通常用具有描述性的单词表示，比如 check_events(),检查事件。函数名命名规则与变量的一样。**
- **parameter1、paramter2 、xxxx：等是函数的参数，也就是函数可以加工处理的数据。参数可以有 0 到多个，但是名字不能重名。所有的参数用括号括起来(),即使 0 个参数也需要。**
- **冒号（:）：函数定义需要冒号结尾，此为 python 语法要求**
- **语句 1、语句 2、return:函数的代码块，说明函数具体需要做什么事情，也就是需要实现什么样的功能。**
  - **函数代码块必须缩进，使用 tab 建即可，不是空格键。这是 python 语法要求。具有同样缩进的代码块才是一个完整的代码块**

- **例子：没有合理缩进，引发的常见错误之一**

```
#定义函数语法
def funtion_name(parameter1,parameter):


语句 1
  语句


  return xxxx
#错误提示为：IndentationError: expected an indented block
 File "/Users/yoni.ma/PycharmProjects/seven_days_python/Sixth_day/define_
fun.py", line 18
  语句 1
   ^
IndentationError: expected an indented block
```

- **语句 1、语句 2 等为具体的还是执行内容，可以有 1 到多个语句，不能没有语句。否则代码错误**
- **return：函数返回值，可有可无。根据自己设计的函数功能而决定。**
  - **函数运行完成以后，如果需要返回一个值给调用该函数的地方，则使用 return。否则不需要。**
  - **return 还表示函数的结束，尤其当代码比较多，各种 while, if for 都有的函数。**

如何调用函数

调用函数就是让 python 执行函数的代码，也就是喊出函数的名字。根据程序设计要求，同一个函数可以被调用多次或 1 次。

调用函数也可以用来测试函数功能是否正确，比如每次写完一个函数的功能，然后调用并运行以查看是否出错或期望的结果。如果错误，则调试代码（查找哪里出错）；否则进行下一个函数的编写。运用此方法虽然有一点繁琐，但是可以大大缩减最好的代码调试，因为我们把错误提前解决了。

调用函数语法为：

```
funtion_name()
```

注意：funtion_name()前面没有缩进，否则不能正确调用函数。

例子：

"""

函数的例子：

知识点：

1。 定义函数

2。 函数调用

"""

```
# 定义函数
def welcome_python():
    print('欢迎加入 Python 实战圈！ ')


# 调用函数。为了方便阅读代码，与定义函数最好空两行。
welcome_python()
welcome_python()
```

结果为：

欢迎加入 Python 实战圈！

欢迎加入 Python 实战圈！

无论是定义还是调研函数，常见错误有以下几个情况：

a) 缺少冒号或者是中文冒号

b) 函数体没有缩进

c) 调用函数的时候缩进了，提示函数错误

2. **例子 welcome_python()缩进了，运行函数没有错误，但是运行结果是空。此错误为新手经常遇到的错误之一。**

```
def welcome_python():
    print('欢迎加入 Python 实战圈！ ')

    welcome_python()
```

结果为

```
/usr/local/bin/python3 /Users/yoni.ma/PycharmProjects/seven_days_python/Sixth_day/exa_fun.py

Process finished with exit code 0
```

如何传递参数

函数有时候需要参数，才能更加完美的处理事情。比如前面的例子，如果想具体的欢迎某一个人加入 Python 实战圈，则需要为该函数添加一个参数，因为我们不知道具体欢迎的是谁。否则也可以直接写在 print("欢迎 xxx 加入 Python 实战圈") 函数里面。参数可以替换任何具体的信息，然后在函数体内价格处理，最好输出我们想要的结果。参数的具体值是在函数调用的时候才指定，可以指定多个不同的值。整个过程被称为数据参数传递。

例子：

```
def welcome_python(member_name,hope):
    # 定义带有参数的函数
    print(f'你好，{member_name} ,欢迎加入 Python 实战圈')
    print(f'\t{hope}')


welcome_python('Kim','希望你能坚持下去。')
welcome_python('Grace','希望你可以找到想要的。')
```

结果为

你好，Kim ,欢迎加入 Python 实战圈

希望你能坚持下去。

你好，Grace ,欢迎加入 Python 实战圈

希望你可以找到想要的。

传递实参

在数据参数传递过程中，函数定义中的参数为形式参数，简称为形参。比如 member_name,hope；调用函数中的参数具体值为实际参数，简称为实参.比如'Kim','希望你能坚持下去。'。形参的定义规范与变量定义一样，另外形参定义完成以后，最好不要修改，否则函数内部都需要修改。实参的值可以是任意值。

形参的个数也可以有多个，称为形参列表。那么调用中，实参也必须有多个（实参列表）。在 Python 中，函数把实参列表传递给形参列表的方法有以下几个方法：1. 位置实参、2.关键字实参

位置实参

位置实参是基于参数的位置，实参与形参的顺序必须相同。每一个实参都关联到函数定义中的一个形参。比如上面的例子。如何位置顺序不对，则结果可能大不一样。位置实参的位置很重要。如果实参的位置和形参的相反，虽然代码可以运行，但是意思肯定错误了。也就是语法正确，语义有问题。该传递方式也是最常用的方式。

例子：

```
# 颠倒实参顺序
print('颠倒实参顺序,结果完全不同 ')
welcome_python('希望你能坚持下去。','Kim')
welcome_python('希望你可以找到想要的。','Grace')
```

结果为

颠倒实参顺序,结果完全不同

你好，希望你能坚持下去。 ,欢迎加入 Python 实战圈

Kim

你好，希望你可以找到想要的。 ,欢迎加入 Python 实战圈

Grace

关键字实参

每一个实参都有变量名+值组成，并且不用考虑函数形参的顺序。调用形式：形参名字=“值”。

优点：无需考虑实参的顺序，易错点：形参的名字一定要正确。

例子：

```
# 关键字实参
welcome_python(hope= '希望你能坚持下去。',member_name='Kim')
welcome_python(member_name='Grace',hope='希望你可以找到想要的。')
```

结果为

```
你好，Kim ,欢迎加入 Python 实战圈
  希望你能坚持下去。
你好，Grace ,欢迎加入 Python 实战圈
  希望你可以找到想要的。
```

默认值

如果形参的值，固定不变，则可以在指定默认值。比如希望每一个加入 Python 实战圈的人都可以坚持下去，则形参 hope 指定为默认值即可。

例子：

```
def welcome_python(member_name,hope='希望你能坚持下去'):
  # 定义带有默认值参数的函数
  print(f'你好，{member_name} ,欢迎加入 Python 实战圈')
  print(f'\t{hope}')


welcome_python('Kim','希望你能坚持下去。')
welcome_python('Kim')
welcome_python('Grace','希望你可以找到想要的。')
```

```
welcome_python('None')
```

结果为

```
你好，Kim ,欢迎加入 Python 实战圈
  希望你能坚持下去。
你好，Kim ,欢迎加入 Python 实战圈
  希望你能坚持下去
你好，Grace ,欢迎加入 Python 实战圈
  希望你可以找到想要的。
你好，None ,欢迎加入 Python 实战圈
  希望你能坚持下去
```

任意数量的实参

有时候，我们不知道需要传递多少个实参。函数调用时候，无需考虑形参的个数。可以任意指定实参的数量。PYTHON 通过使用*，让程序创建一个空元组，可以接受任意数量的参数值。

例子：

```
def welcome_python(*member_names,hope='希望你能坚持下去'):
  # 任意多个参数的函数
  for member_name in member_names:
    print(f'你好，{member_name}. 欢迎加入 Python 实战圈')
    print(f'\t{hope}')


welcome_python('Kim','Grace','Nelson Lam','None')
```

结果为

```
你好，Kim. 欢迎加入 Python 实战圈
  希望你能坚持下去
你好，Grace. 欢迎加入 Python 实战圈
  希望你能坚持下去
```

```
你好，Nelson Lam. 欢迎加入 Python 实战圈
  希望你能坚持下去
你好，None. 欢迎加入 Python 实战圈
  希望你能坚持下去
```

返回值

函数运行完成以后，如果需要返回一个值给调用该函数的地方，则使用 return 返回。函数返回值的类型可以是基本数据类型，也可以是字典、列表等。

返回值能让主程序简单，把大部分处理工作交给函数。比如 加入 Python 实战圈的人有外国人，名字分为 last name 与 first name. 则欢迎加入 python 实战圈的语句中，名字的处理可以使用函数解决。

例子：

```
def full_name (first_name,last_name):
  # 带有返回值的函数
  person = first_name + ' ' + last_name
  return person.title()


name = full_name('Nelson','lam')
welcome_python(name)
```

结果为

```
你好，Nelson Lam ,欢迎加入 Python 实战圈
  希望你能坚持下去
```

传递数据结构

传递列表

列表也可以作为函数的参数。函数就可以直接访问列表中的元素

例子：

```
def welcome_python(member_names,hope='希望你能坚持下去'):
  # 列表作为参数的函数
```

```
    for member_name in member_names:
        print(f'你好，{member_name}. 欢迎加入 Python 实战圈')
        print(f'\t{hope}')


list_name =['Kim','Grace','Nelson Lam','None']
welcome_python(list_name)
```

结果为

```
你好，Kim. 欢迎加入 Python 实战圈
    希望你能坚持下去
你好，Grace. 欢迎加入 Python 实战圈
    希望你能坚持下去
你好，Nelson Lam. 欢迎加入 Python 实战圈
    希望你能坚持下去
你好，None. 欢迎加入 Python 实战圈
    希望你能坚持下去
```

传递字典

函数中传递字典。在 Python 中，使用作为形参接受实参传递的键值对。注意区别：在形参列表中，*表示传递空元组、**表示传递字典。**

例子：

```python
# -*- coding: utf-8 -*-
def employee(first_name,last_name,**employee_infor):
    """
    创建字典，存储所有的员工信息
    """
    employee = {}
    employee["first_name"] = first_name
    employee["last_name"] = last_name

    #使用for循环存储所有的其他键值对信息
    for key,value in employee_infor.items():
        employee[key]=value #注意key不用有引号，否则只能显示组后一个实参

    return employee

my_employee = employee("Yoni","ma",location="北京",dep="大数据")
print("Print my employee information:\n")
for key, value in my_employee.items():
    print(key+":"+value+"\n")
```

结果为：

```
Print my employee information:
first_name:Yoni
last_name:ma
location:北京
dep:大数据
```

盒子的秘密

函数可以被看作一个盒子，分为盒子外和盒子内。盒子内就是函数的内容，也被称为是私有的，内部的资源只能被自己使用。盒子外就是函数之前的内容，也被称为是外部内容，函数不能直接使用，只能通过参数传递的方法进行。盒子内与外的变量名字可以相同或者不同，不冲突。函数的 return 可以看作是和外部沟通的桥梁。

盒子内部的变量为局部变量，也就是定义在函数内部的变量并且作用域为整个函数；盒子外的变量为全局变量，作用域为整个文件。

例子：

```
#这是全局变量
add_sum = 3
def sum_add (para, para2):
  # 求 2 个参数的和"
```

```
    add_sum = para + para2  # total 在这里是局部变量.
    print("函数内部变量 add_sum =  ",add_sum)
    return add_sum


# 调用 sum 函数
ad_sum = sum_add(18,31)
print('18 + 31 = ',ad_sum)
print("函数外面是全局变量 : add_sum = ",add_sum)
```

结果为：

```
函数内部变量 add_sum =  49
18 + 31 =  49
函数外面是全局变量 : add_sum =  3
```

将函数存储在模块中

有时候，我们可以将函数存在单独的文件中，隐藏其逻辑。这样就可以把主要精力用在主体程序。

比如把员工信息的函数存放在文件 employee_model.py 中，则主函数直接调用该文件即可。

例子：

```
"""
    调用函数
"""

import employee_model

my_employee = employee_model.employee( "Yoni", "Ma", location='Beijing', dep ="Big data")
print("Print my employee information:\n")
for key, value in my_employee.items():
    print(key+":"+value+"\n")
```

结果为：

```
Print my employee information

first_name Yoni

last_name Ma

location Beijin

dep Bi  data


Process finished with exit code
```

使用下面的 4 种方法把模块中函数导入到主体文件中

```
#导入整个模块文件的函数，必须使用module_name.function_name()调用
import employee_model
my_employee = employee_model.employee("Yoni","ma",location="北京",dep="大数据")
#导入模块中特定的函数，调用时候，可以不用模块名字
from employee_model import employee
my_employee = employee("Yoni","ma",location="北京",dep="大数据")

#使用as制定函数的别名
from employee_model import employee as em
my_employee = em("Yoni","ma",location="北京",dep="大数据")

#使用* 导入模块中的所有函数,但是加载会很慢
from employee_model import *
my_employee = employee("Yoni","ma",location="北京",dep="大数据")

```

如何设计函数

面对问题时，如何把代码构建为函数。第一步，先把构思问题的构成部分；第二步，每一个部分写一个函数。然后考虑每个函数都应该只负责一件事情。如何函数处理的事情比较多，则要考虑拆解为两个函数，然后采用调用的方法实现。这样做的目的是把复杂任务分成不同的步骤来完成。

每一个函数的编码规范如下：

- **函数名字尽量使用小写字母+下划线**

- **给函数注释其功能**
- **形参的默认值，等号两边不能有空格**
- **函数中的关键字实参，也不能有空格**
- **注意区分函数的参数调用，位置实参、关键字实参等**
- **程序中有两个函数时候，空 2 行表示区分**
- **所有 import 语句放在文件开头**

项目实战：如何利用 python 删除一个指定的文件

要求：

必须使用函数

提示与问题拆解

a) **如何找到指定路径下的所有文件**
b) **如何找到指定的文件名在指定的路径下**
c) **如何删除找到的文件为新的名字**
d) **使用 os 模块下的 remove()方法**

欢迎扫码加入 Python 实战圈