函数

> 版权：智泊AI
>
> 作者：Jeff

1.函数概述

1.1为什么使用函数

问题：代码重复
　　后期维护成本太高
　　代码可读性不高

解决问题：函数
　在一个完整的项目中，某些功能会被反复使用，那么将这部分功能对应的代码提取出来，当需要使用功能的时候直接使用

本质：对一些特殊功能的封装

优点：
　a.简化代码结构，提高应用的效率
　b.提高代码复用性
　c.提高代码的可读性和可维护性

建议：但凡涉及到功能，都尽量使用函数实现

1.2认识函数

> 需求: 求圆的面积
>
> $s = \pi r^2$
>
> 代码演示：

```
import math
def s(r):
    return math.pi * r * r
```

1.3定义函数

> 语法:
>
> def 函数名(参数1，参数2，参数3....):
>
> 　　函数体
>
> 　　返回值(可以有也可以没有,根据具体的场景)
>
> 说明：

a.函数由两部分组成：声明部分和实现部分

b. def,关键字，是define的缩写，表示定义的意思

c. 函数名：类似于变量名，遵循标识符的命名规则，尽量做到见名知意

d.（）：表示的参数列表的开始和结束

e. 参数1，参数2，参数3.... ：参数列表【形式参数，简称为形参】，其实本质上就是一个变量名，参数列表可以为空

f.函数体：封装的功能的代码

g.返回值：一般用于结束函数，可有可无，如果有返回值，则表示将相关的信息携带出去，携带给调用者，如果没有返回值，则相当于返回None

```
# test()   name 'test' is not defined   # 函数调用不能在函数定义的上面
def test():
  print("平安!")
```

1.4函数调用

```
test()   # 函数调用
test()
test()

'''
函数调用的格式：  函数名()
1.函数是条狗,哪里需要哪里吼.     函数定义以后不会自动执行. 若想执行函数中的函数体,需要调用函数.
2.函数必须先定义,后调用.调用函数必须在定义函数的下边.
3.函数可以多次调用.
4.在同一个文件中,出现了重名的函数,后面的函数会把前面的函数覆盖.调用的时候执行后面的函数.
'''
```

2.使用函数

函数的分类:

```
1.按照函数是否是自己定义分为:
  内置函数(python已经定义的,可以直接使用的): 比如:print()  min()  sum()  max()等
  自定义函数:根据自己的需求定义的函数

2.根据函数中有没有参数分为:
  有参数的函数: 在定义函数时传入的参数叫做形式参数(形参)    在调用函数时传入的参数叫做实际参数(实参)
  没有参数的函数:

3.根据定义的函数有没有函数名分为:
  有名字的函数: def 函数名():
          函数体
  匿名函数: 通过 lambda 定义
```

4.根据函数中是否有返回值分为：
  有返回值的函数： 使用return 返回数据
  没有返回值的函数：

2.1 无参数的函数

> 代码演示：
>
> ```
> def test():
>   print("年少不知软饭香!")
> ```

2.2有参数的函数

代码演示:

```
# 有参数的函数：
def my_sum(a,b):    # a,b就是形参
    print(a + b)

my_sum(12,23)  # 12,23就是实参
```

2.3没有返回值的函数

```
# 没有返回值的函数
def say():
    print("我是没有返回值的函数")

say()
```

2.4有返回值的函数

```
# 有返回值的函数：  通过return关键字返回数据
def demo():
    return "我是有返回值的函数"

str = demo()
print(str)

'''
1.return 后面可以返回一个或者多个数据,返回多个数据,以元组的形式展示出来
2.return 下面的代码不会执行
'''
def demo1():
    print("哈!")
    return 12,34,578
    print("haha")   # 不会执行
tuple1 = demo1()
print(tuple1)  # (12, 34, 578)
```

```
# 函数中没有return或者return后面没有数据返回,则默认返回的是None
def demo2():
    return
print(demo2())  # None

# 注意:
#     1.有返回值的函数,使用return关键字将数据返回,若想使用返回的数据,
#        需要在调用函数的位置定义一个变量接收返回的数据.
#     2.return 返回的数据可以是一个或者多个,返回的多个数据是以元组的形式展示.
#     3.return关键词下面的代码不会执行.
#     4.在函数中若不写return 或者 return后面没有返回具体的数据,则返回的是None
#
```

2.2函数嵌套函数

> 函数的调用顺序：
>
> 函数之间可以进行相互嵌套调用
>
> ```
> def test():
>   test1()
>   print(11111)
>
> def test1():
>   test2()
>   print(22222)
>
> def test2():
>   test3()
>   print(333333)
>
> def test3():
>   print(44444)
>
> test()
> ```

2.3函数中的参数

> 参数列表：如果函数所实现的功能涉及到未知项参与运算，此时就可以将未知项设置为参数
>
> 格式：参数1,参数2.....
>
> 分类：
>
> 形式参数：在函数的声明部分，本质就是一个变量，用于接收实际参数的值 【形参】
>
> 实际参数：在函数调用部分，实际参与运算的值，用于给形式参数赋值 【实参】
>
> 传参：实际参数给形式参数赋值的过程，形式参数 = 实际参数
>
> 代码演示：

```
def get_max(num1,num2): # num1,num2是形参
  if num1 > num2:
     return num1
  else:
     return num2

max_value = get_max(12,345)  # 12,345是实参
print(max_value)
```

2.4参数的类型【掌握】

a.必需参数/位置参数

调用函数的时候必须以正确的顺序传参，传参的时候参数的数量和形参必须保持一致

代码演示：

```
# 必需参数
def student(name,age):
   print("我的姓名是%s,年龄是%d"%(name,age))

student("jeff",18)
# student(18,"jeff")  # %d format: a number is required, not str
```

b.关键字参数

使用关键字参数允许函数调用的时候实参的顺序和形参的顺序可以不一致，可以使用关键字进行自动的匹配

代码演示：

```
def student(name,age):
  print("我的姓名是%s,年龄是%d"%(name,age))

student(age = 22,name = "lucy")
```

c. 默认参数(默认参数是在函数定义的时候,直接给形参赋的值)

调用函数的时候，如果没有传递参数，则会使用默认参数,如果传输了参数,则使用传递的参数.

代码演示：

```python
def get_sum(num1,num2 = 12):
  print(num1+num2)

get_sum(23)
get_sum(23,87)

# 注意:
#    如果函数中有多个形参,给该函数设置默认参数时,一般把这个默认参数放在形参列表的最后面.
```

d.不定长参数(可变参数)

可以处理比当初声明时候更多的参数 *(元组) **(字典)

*args: 用来接收多个位置参数 argments 得到的形式是 元组

**kwargs: 用来接收多个关键字参数 keyword arguments 得到的形式是字典

代码演示:

```python
# 不定长参数: *args   **kwargs
def fn(*args):
   print(args)

fn(34,533,4,3)

# 注意:
'''
在自定义函数时,若函数中有多个参数,某一个参数是不定长参数,一般把不定长参数放在参数列表的最后面.
'''
def fn1(name,*args):
   print(name,args)
fn1("赵杰",12,345,6,"hehe",True)

# **kwargs 接收的是关键字参数,格式是 key=value这种形式
def fn2(**kwargs):
   print(kwargs)
fn2(x = 10,y = 12,z = 88)
```

3.匿名函数【掌握】

不再使用def这种的形式定义函数，使用lambda来创建匿名函数

特点:

a.lambda只是一个表达式，比普通函数简单

b.lambda一般情况下只会书写一行，包含参数，实现体，返回值

语法:lambda 参数列表： 实现部分

代码演示：

```
# 匿名函数：lambda
def fn(n):
  return n**2
print(fn(3))

# 匿名函数
f1 = lambda n: n**2
print(f1(3))

# 匿名函数
f2 = lambda x,y:x*y
print(f2(12,3))
```