# 函数def  
  
  
#函数的上和下必须各空两行  
def my\_abs(x):  
 if x >= 0:  
 return x  
 else:  
 return -x  
  
  
print(my\_abs(-50))  
# 输出： 50

# 利用isinstance验证，如果不是数字，则raise一个错误  
def my\_abs(x):  
 if not isinstance(x, (int, float)):  
 raise TypeError('bad operand type')  
 if x >= 0:  
 return x  
 else:  
 return -x  
  
  
print(my\_abs('中'))  
# 输出：  
# Traceback (most recent call last):  
# File "D:/python-learning/python-tests/test1/t1.py", line 25, in <module>  
# print(my\_abs('中'))  
# File "D:/python-learning/python-tests/test1/t1.py", line 18, in my\_abs  
# raise TypeError('bad operand type')  
# TypeError: bad operand type

返回多个值的函数

# 函数可以返回多个值，其返回的是一个元组tuple  
def r2():  
 a = 2  
 b = 3  
 return a, b  
  
print(r2())  
# 输出：(2, 3)  
  
print(type(r2()))  
# 输出：<class 'tuple'>  
  
# 也可以这样取值  
x, y = r2()  
print(x)  
# 输出：2  
print(y)  
# 输出：3

■递归函数

# 在函数内部，可以调用其他函数。如果一个函数在内部调用自身本身，这个函数就是递归函数。  
# 举个例子，我们来计算阶乘n! = 1 x 2 x 3 x ... x n，用函数fact(n)表示，可以看出：  
# fact(n) = n! = 1 x 2 x 3 x ... x (n-1) x n = (n-1)! x n = fact(n-1) x n  
# 所以，fact(n)可以表示为n x fact(n-1)，只有n=1时需要特殊处理。  
# 于是，fact(n)用递归的方式写出来就是：  
  
  
def fact(n):  
 if n == 1:  
 return 1  
 return n \* fact(n - 1)  
  
  
print(fact(1))  
# 输出：1  
print(fact(5))  
# 输出：120

●递归函数 分型树实例

import turtle  
  
  
def draw\_branch(branch\_length):  
 if branch\_length > 5:  
 # 右侧树枝  
 turtle.forward(branch\_length)  
 # print('向前', branch\_length)  
 turtle.right(20)  
 # print('右转', 20)  
 draw\_branch(branch\_length - 15)  
  
 # 左侧树枝  
 turtle.left(40)  
 # print('左转', 40)  
 draw\_branch(branch\_length - 15)  
  
 # 返回之前树枝  
 turtle.right(20)  
 # print('右转', 20)  
 turtle.backward(branch\_length)  
 # print('倒退', branch\_length)  
  
  
  
def main():  
 turtle.left(90)  
 turtle.penup()  
 turtle.backward(150)  
 turtle.pendown()  
 turtle.color('brown')  
 draw\_branch(40)  
 turtle.exitonclick()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

■函数的参数

●位置参数

# 计算乘方的函数  
def power(x, n):  
 s = 1  
 while n > 0:  
 n = n - 1  
 s = s \* x  
 return s  
  
  
print(power(2, 3))  
# 输出： 8

●默认参数

# 一是必选参数在前，默认参数在后，否则Python的解释器会报错（思考一下为什么默认参数不能放在必选参数前面）；  
# 二是如何设置默认参数。  
# 当函数有多个参数时，把变化大的参数放前面，变化小的参数放后面。变化小的参数就可以作为默认参数。  
# 使用默认参数有什么好处？最大的好处是能降低调用函数的难度。  
# 默认参数n=2  
def power(x, n=2):  
 s = 1  
 while n > 0:  
 n = n - 1  
 s = s \* x  
 return s  
  
# 只输入一个参数时，默认第二个参数等于2  
print(power(2))  
# 输出： 4  
print(power(2, 3))  
# 输出： 8

●可变参数

# 计算若干个参数的平方相加 ，例如1的平方加2的平方加3的平方  
def calc(numbers):  
 sum = 0  
 for n in numbers:  
 sum = sum + n \* n  
 return sum  
  
# 这样传入的参数必须是一个数组或元组  
print(calc([1, 2, 3]))  
# 输出： 14  
  
# 使用可变参数\* 就可以随意传入参数了  
def calcs(\*numbers):  
 sum = 0  
 for n in numbers:  
 sum = sum + n \* n  
 return sum  
  
  
print(calcs(1, 2, 3))  
# 输出： 14

●关键字参数

# 关键字参数有什么用？它可以扩展函数的功能。  
# 比如，在person函数里，我们保证能接收到name和age这两个参数，  
# 但是，如果调用者愿意提供更多的参数，我们也能收到。  
# 试想你正在做一个用户注册的功能，除了用户名和年龄是必填项外，其他都是可选项，  
# 利用关键字参数来定义这个函数就能满足注册的需求。  
def person(name, age, \*\*kw):  
 print('name:', name, 'age:', age, 'other:', kw)  
  
# 可以只输入必填参数  
person('Tom', 22)  
# 输出： name: Tom age: 22 other: {}  
  
# 而关键字参数允许传入0个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个dict。  
person('Alice', 21, city='北京')  
# 输出： name: Alice age: 21 other: {'city': '北京'}  
  
# 可以把所有关键字参数先存到一个dict里，然后再把dict传入person，注意前面要加\*\*  
extra = {'city': 'Beijing', 'job': 'Engineer'}  
person('Jack', 24, \*\*extra)  
# 输出： name: Jack age: 24 other: {'city': 'Beijing', 'job': 'Engineer'}

●命名关键字参数

# 如果要限制关键字参数的名字，就可以用命名关键字参数，例如，只接收city和job作为关键字参数。  
# 和关键字参数\*\*kw不同，命名关键字参数需要一个特殊分隔符\*，\*后面的参数被视为命名关键字参数。  
def person(name, age, \*, city, job):  
 print(name, age, city, job)  
  
  
person('Tom', 22, city='北京', job='程序员')  
# 输出：Tom 22 北京 程序员