

PYTHON教程

1. 简介

2. Python历史

3. 安装Python

3.1. Python解释器

4. 第一个Python程序

4.1. 使用文本编辑器

4.2. 输入和输出

5. Python基础

5.1. 数据类型和变量

5.2. 字符串和编码

5.3. 使用list和tuple

5.4. 条件判断

5.5. 模式匹配

5.6. 循环

5.7. 使用dict和set

6. 函数

6.1. 调用函数

6.2. 定义函数

6.3. 函数的参数

6.4. 递归函数

7. 高级特性

7.1. 切片

7.2. 迭代

7.3. 列表生成式

7.4. 生成器

7.5. 迭代器

8. 函数式编程

9. 模块

10. 面向对象编程

11. 面向对象高级编程

12. 错误、调试和测试

13. IO编程

14. 进程和线程

15. 正则表达式

16. 常用内建模块

17. 常用第三方模块

18. 图形界面

19. 网络编程

20. 电子邮件

21. 访问数据库

22. Web开发

23. 异步IO

24. FAQ

25. 期末总结

定义函数



廖雪峰



资深软件开发工程师，业余马拉松选手。

在Python中，定义一个函数要使用 `def` 语句，依次写出函数名、括号、括号中的参数和冒号 `:`，然后，在缩进块中编写函数体，函数的返回值用 `return` 语句返回。

我们以自定义一个求绝对值的 `my_abs` 函数为例：

```
def my_abs(x):
    if x >= 0:
        return x
    else:
        return -x

print(my_abs(-99))
```

请自行测试并调用 `my_abs` 看看返回结果是否正确。

请注意，函数体内部的语句在执行时，一旦执行到 `return` 时，函数就执行完毕，并将结果返回。因此，函数内部通过条件判断和循环可以实现非常复杂的逻辑。

如果没有 `return` 语句，函数执行完毕后也会返回结果，只是结果为 `None`。`return None` 可以简写为 `return`。

在Python交互环境中定义函数时，注意Python会出现 `...` 的提示。函数定义结束后需要按两次回车重新回到 `>>>` 提示符下：

```
Windows PowerShell
>>> def my_abs(x):
...     if x >= 0:
...         return x
...     else:
...         return -x
...
>>> my_abs(-9)
9
>>>
```

如果你已经把 `my_abs()` 的函数定义保存为 `abstest.py` 文件了，那么，可以在该文件的当前目录下启动Python解释器，用 `from abstest import my_abs` 来导入 `my_abs()` 函数，注意 `abstest` 是文件名（不含 `.py` 扩展名）：

```
Windows PowerShell
>>> from abstest import my_abs
>>> my_abs(-9)
9
>>>
```

`import` 的用法在后续模块一节中会详细介绍。

空函数

如果想定义一个什么事也不做的空函数，可以用 `pass` 语句：

```
def nop():
    pass
```

`pass` 语句什么都不做，那有什么用？实际上 `pass` 可以用来作为占位符，比如现在还没想好怎么写函数的代码，就可以先放一个 `pass`，让代码能运行起来。

`pass` 还可以用在其他语句里，比如：

```
if age >= 18:
    pass
```

缺少了 `pass`，代码运行就会有语法错误。

PYTHON教程

- 1. 简介
- 2. Python历史
- 3. 安装Python
- 3.1. Python解释器
- 4. 第一个Python程序
- 4.1. 使用文本编辑器
- 4.2. 输入和输出
- 5. Python基础
- 5.1. 数据类型和变量
- 5.2. 字符串和编码
- 5.3. 使用list和tuple
- 5.4. 条件判断
- 5.5. 模式匹配
- 5.6. 循环
- 5.7. 使用dict和set
- 6. 函数
- 6.1. 调用函数
- 6.2. 定义函数
- 6.3. 函数的参数
- 6.4. 递归函数
- 7. 高级特性
- 7.1. 切片
- 7.2. 迭代
- 7.3. 列表生成式
- 7.4. 生成器
- 7.5. 迭代器
- 8. 函数式编程
- 9. 模块
- 10. 面向对象编程
- 11. 面向对象高级编程
- 12. 错误、调试和测试
- 13. IO编程
- 14. 进程和线程
- 15. 正则表达式
- 16. 常用内建模块
- 17. 常用第三方模块
- 18. 图形界面
- 19. 网络编程
- 20. 电子邮件
- 21. 访问数据库
- 22. Web开发
- 23. 异步IO
- 24. FAQ
- 25. 期末总结

参数检查

调用函数时，如果参数个数不对，Python解释器会自动检查出来，并抛出 `TypeError`：

```
>>> my_abs(1, 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: my_abs() takes 1 positional argument but 2 were given
```

但是如果参数类型不对，Python解释器就无法帮我们检查。试试 `my_abs` 和内置函数 `abs` 的差别：

```
>>> my_abs('A')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in my_abs
TypeError: unorderable types: str() >= int()
>>> abs('A')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: bad operand type for abs(): 'str'
```

当传入了不恰当的参数时，内置函数 `abs` 会检查出参数错误，而我们定义的 `my_abs` 没有参数检查，会导致 `if` 语句出错，出错信息和 `abs` 不一样。所以，这个函数定义不够完善。

让我们修改一下 `my_abs` 的定义，对参数类型做检查，只允许整数和浮点数类型的参数。数据类型检查可以用内置函数 `isinstance()` 实现：

```
def my_abs(x):
    if not isinstance(x, (int, float)):
        raise TypeError('bad operand type')
    if x >= 0:
        return x
    else:
        return -x
```

添加了参数检查后，如果传入错误的参数类型，函数就可以抛出一个错误：

```
>>> my_abs('A')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in my_abs
TypeError: bad operand type
```

错误和异常处理将在后续讲到。

返回多个值

函数可以返回多个值吗？答案是肯定的。

比如在游戏中经常需要从一个点移动到另一个点，给出坐标、位移和角度，就可以计算出新的坐标：

```
import math

def move(x, y, step, angle=0):
    nx = x + step * math.cos(angle)
    ny = y - step * math.sin(angle)
    return nx, ny
```

`import math` 语句表示导入 `math` 包，并允许后续代码引用 `math` 包里的 `sin`、`cos` 等函数。

然后，我们就可以同时获得返回值：

```
>>> x, y = move(100, 100, 60, math.pi / 6)
>>> print(x, y)
151.96152422706632 70.0
```

但其实这只是一种假象，Python函数返回的仍然是单一值：

PYTHON教程

- 1. 简介
- 2. Python历史
- 3. 安装Python
 - 3.1. Python解释器
- 4. 第一个Python程序
 - 4.1. 使用文本编辑器
 - 4.2. 输入和输出
- 5. Python基础
 - 5.1. 数据类型和变量
 - 5.2. 字符串和编码
 - 5.3. 使用list和tuple
 - 5.4. 条件判断
 - 5.5. 模式匹配
 - 5.6. 循环
 - 5.7. 使用dict和set
- 6. 函数
 - 6.1. 调用函数
 - 6.2. 定义函数
 - 6.3. 函数的参数
 - 6.4. 递归函数
- 7. 高级特性
 - 7.1. 切片
 - 7.2. 迭代
 - 7.3. 列表生成式
 - 7.4. 生成器
 - 7.5. 迭代器
- 8. 函数式编程
- 9. 模块
- 10. 面向对象编程
- 11. 面向对象高级编程
- 12. 错误、调试和测试
- 13. IO编程
- 14. 进程和线程
- 15. 正则表达式
- 16. 常用内建模块
- 17. 常用第三方模块
- 18. 图形界面
- 19. 网络编程
- 20. 电子邮件
- 21. 访问数据库
- 22. Web开发
- 23. 异步IO
- 24. FAQ
- 25. 期末总结

```
>>> r = move(100, 100, 60, math.pi / 6)
>>> print(r)
(151.96152422706632, 70.0)
```

原来返回值是一个tuple! 但是，在语法上，返回一个tuple可以省略括号，而多个变量可以同时接收一个tuple，按位置赋给对应的值，所以，Python的函数返回多值其实就是返回一个tuple，但写起来更方便。

练习

请定义一个函数 `quadratic(a, b, c)`，接收3个参数，返回一元二次方程 $ax^2 + bx + c = 0$ 的两个解。

提示：

一元二次方程的求根公式为：

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

计算平方根可以调用 `math.sqrt()` 函数：

```
>>> import math
>>> math.sqrt(2)
1.4142135623730951
```

```
import math

def quadratic(a, b, c):
    pass

# 测试:
print('quadratic(2, 3, 1) =', quadratic(2, 3, 1))
print('quadratic(1, 3, -4) =', quadratic(1, 3, -4))

if quadratic(2, 3, 1) != (-0.5, -1.0):
    print('测试失败')
elif quadratic(1, 3, -4) != (1.0, -4.0):
    print('测试失败')
else:
    print('测试成功')
```

参考源码

`def_func.py`

小结

定义函数时，需要确定函数名和参数个数；

如果有必要，可以先对参数的数据类型做检查；

函数体内部可以用 `return` 随时返回函数结果；

函数执行完毕也没有 `return` 语句时，自动 `return None`。

函数可以同时返回多个值，但其实就是一个tuple。

« 调用函数

» 函数的参数 »



Comments

PYTHON教程

- 1. 简介
- 2. Python历史
- 3. 安装Python
 - 3.1. Python解释器
- 4. 第一个Python程序
 - 4.1. 使用文本编辑器
 - 4.2. 输入和输出
- 5. Python基础
 - 5.1. 数据类型和变量
 - 5.2. 字符串和编码
 - 5.3. 使用list和tuple
 - 5.4. 条件判断
 - 5.5. 模式匹配
 - 5.6. 循环
 - 5.7. 使用dict和set
- 6. 函数
 - 6.1. 调用函数
 - 6.2. 定义函数
 - 6.3. 函数的参数
 - 6.4. 递归函数
- 7. 高级特性
 - 7.1. 切片
 - 7.2. 迭代
 - 7.3. 列表生成式
 - 7.4. 生成器
 - 7.5. 迭代器
- 8. 函数式编程
- 9. 模块
- 10. 面向对象编程
- 11. 面向对象高级编程
- 12. 错误、调试和测试
- 13. IO编程
- 14. 进程和线程
- 15. 正则表达式
- 16. 常用内建模块
- 17. 常用第三方模块
- 18. 图形界面
- 19. 网络编程
- 20. 电子邮件
- 21. 访问数据库
- 22. Web开发
- 23. 异步IO
- 24. FAQ
- 25. 期末总结

Comments loaded. To post a comment, please [Sign In](#)



仰望星空 @ 2025/12/31 07:54:48

```
import math def quadratic(a,b,c): x1 = (-b+math.sqrt(b**2-4*a*c))/(2*a) x2 = (-b-math.sqrt(b**2-4*a*c))/(2*a) return x1,x2
```

测试:

```
print('quadratic(2, 3, 1) =', quadratic(2, 3, 1)) print('quadratic(1, 3, -4) =', quadratic(1, 3, -4))
```

```
if quadratic(2, 3, 1) != (-0.5, -1.0): print('测试失败') elif quadratic(1, 3, -4) != (1.0, -4.0): print('测试失败') else: print('测试成功')
```



~殤十魂~ @ 2025/12/27 00:42:34

```
import math def eryuanyici(a,b,c): if not all(isinstance(x,(int,float)) for x in (a,b,c)): raise TypeError("the number is not int or float") a = float(a) b = float(b) c = float(c) if a == 0 or b == 0: print("一元二次方程的a值{}和b值{}不能为零。") return None, None delta = (b*b) - 4*a*c if delta == 0: print ("有相同实数根") elif delta < 0: print("无解") if delta > 0: x1 = (-b + math.sqrt(b**2 - 4*a*c))/(2*a) x2 = (-b - math.sqrt(b**2 - 4*a*c))/(2*a) return x1,x2
```



杰 @ 2025/12/15 09:07:31

提供一个内置的测试程序test()。

```
# !/usr/bin/python3
# -*-coding:UTF-8-*-

from math import sqrt

def quadratic(a, b, c):
    """
    功能: 求一元二次方程 ax^2+bx+c=0的解
    参数:
        a, b, c :方程的系数
    返回:
        x1, x2 : 元组。如果没有解, 返回空元组
    """

```

[Read More ▾](#)



杰 @ 2025/12/15 09:19:23

需要复数解用“cmath.sqrt”：

```
import cmath
......

if delta < 0:
    # 复数解
    x1 = (-b + cmath.sqrt(delta)) / (2*a)
    x2 = (-b - cmath.sqrt(delta)) / (2*a)
```



hellothankyou @ 2025/12/14 08:06:03

```
import math

def quadratic(a,b,c):

    if a==0:
        raise ValueError('a cannot be zero')
```

PYTHON教程

1. 简介

2. Python历史

3. 安装Python

3.1. Python解释器

4. 第一个Python程序

4.1. 使用文本编辑器

4.2. 输入和输出

5. Python基础

5.1. 数据类型和变量

5.2. 字符串和编码

5.3. 使用list和tuple

5.4. 条件判断

5.5. 模式匹配

5.6. 循环

5.7. 使用dict和set

6. 函数

6.1. 调用函数

6.2. 定义函数

6.3. 函数的参数

6.4. 递归函数

7. 高级特性

7.1. 切片

7.2. 迭代

7.3. 列表生成式

7.4. 生成器

7.5. 迭代器

8. 函数式编程

9. 模块

10. 面向对象编程

11. 面向对象高级编程

12. 错误、调试和测试

13. IO编程

14. 进程和线程

15. 正则表达式

16. 常用内建模块

17. 常用第三方模块

18. 图形界面

19. 网络编程

20. 电子邮件

21. 访问数据库

22. Web开发

23. 异步IO

24. FAQ

25. 期末总结

```
if b*b==4*a*c:  
    return print('无实数解')  
  
x1=(-b+math.sqrt(b*b-4*a*c))/(2*a)  
  
x2=(-b-math.sqrt(b*b-4*a*c))/(2*a)  
  
return x1,x2
```

一开始忘记对函数进行检查了，参考了一下评论区的答案~

Rynhow @ 2025/12/8 07:28:41

```
import math  
  
def quadratic(a, b, c): delta = b ** 2 - 4 * a * c if delta < 0: return '方程无实数根' else: x1 = (-b +  
math.sqrt(b ** 2 - 4 * a * c)) / (2 * a) x2 = (-b - math.sqrt(b ** 2 - 4 * a * c)) / (2 * a) return x1,x2
```

测试：

```
print('quadratic(2, 3, 1) =', quadratic(2, 3, 1)) print('quadratic(1, 3, -4) =', quadratic(1, 3, -4))
```

```
if quadratic(2, 3, 1) != (-0.5, -1.0): print('测试失败') elif quadratic(1, 3, -4) != (1.0, -4.0): print('测试失败')  
else: print('测试成功')
```

某不科学动物 @ 2025/12/7 00:55:32

```
import math  
  
def quadratic(a,b,c):  
    val=b*b - 4*a*c  
    if val < 0 :  
        return '无解'  
    elif val > 0 :  
        x1=(((-b+(math.sqrt(b*b-4*a*c)))/(2*a))  
        x2=(((-b-(math.sqrt(b*b-4*a*c)))/(2*a))  
        return x1,x2  
    else:  
        x1=(((-b+(math.sqrt(b*b-4*a*c)))/(2*a))  
        x2=(((-b-(math.sqrt(b*b-4*a*c)))/(2*a))  
        return 'x1=x2=%s' % x1
```

Read More ▾

啊, 你以为我傻 @ 2025/12/3 03:49:26

```
def quadratic(a,b,c): delta = b**2 - 4*a*c if delta < 0: return "该二次方程无实数解" elif delta == 0: x = (-  
b)/(2*a) return x else: x1= (-b + math.sqrt(b**2-4*a*c))/(2*a) x2= (-b - math.sqrt(b**2-4*a*c))/(2*a)  
return x1,x2 print(quadratic(2,3,1))
```

lv @ 2025/11/28 04:47:27

```
def quadratic(na, b, c):  
    if not isinstance(na,(int,float)) or not isinstance(b,(int,float)) or not isinstance(c,(int,float)): raise  
        TypeError('bad operand type')  
    if b ** 2 - 4 * na * c < 0:  
        raise ValueError('no solution')  
    else:  
        nx = (-b + math.sqrt(b ** 2 - 4 * na * c)) / (2 * na)  
        ny= (-b - math.sqrt(b ** 2 - 4 * na * c)) / (2 * na)  
        return nx,ny
```

PYTHON教程

- 1. 简介
- 2. Python历史
- 3. 安装Python
 - 3.1. Python解释器
- 4. 第一个Python程序
 - 4.1. 使用文本编辑器
 - 4.2. 输入和输出
- 5. Python基础
 - 5.1. 数据类型和变量
 - 5.2. 字符串和编码
 - 5.3. 使用list和tuple
 - 5.4. 条件判断
 - 5.5. 模式匹配
 - 5.6. 循环
 - 5.7. 使用dict和set
- 6. 函数
 - 6.1. 调用函数
 - 6.2. 定义函数**
 - 6.3. 函数的参数
 - 6.4. 递归函数
- 7. 高级特性
 - 7.1. 切片
 - 7.2. 迭代
 - 7.3. 列表生成式
 - 7.4. 生成器
 - 7.5. 迭代器
- 8. 函数式编程
- 9. 模块
- 10. 面向对象编程
- 11. 面向对象高级编程
- 12. 错误、调试和测试
- 13. IO编程
- 14. 进程和线程
- 15. 正则表达式
- 16. 常用内建模块
- 17. 常用第三方模块
- 18. 图形界面
- 19. 网络编程
- 20. 电子邮件
- 21. 访问数据库
- 22. Web开发
- 23. 异步IO
- 24. FAQ
- 25. 期末总结

Meow @ 2025/11/13 02:15:22



```
import math

def quadratic(a, b, c):
    delta = b*b - 4*a*c
    if delta < 0:
        return None
    elif delta == 0:
        x = -b / (2*a)
        return x
    else:
        x1 = (-b + math.sqrt(delta)) / (2*a)
        x2 = (-b - math.sqrt(delta)) / (2*a)
        return x1, x2

# 测试:
print('quadratic(1, 2, 5) =', quadratic(1, 2, 5))
```

Read More ▾

待绝笔墨痕干 @ 2025/11/2 21:22:26



```
import math
def quadratic(a,b,c):
    i = b**2-4*a*c
    if i<0:
        print('此方程无解! ')
    else:
        i=math.sqrt(i)
        x1=(-b+i)/(2*a)
        x2=(-b-i)/(2*a)
        if x1==x2:
            print('此方程有唯一解, x=%s'%x1)
            return x1
        else:
            print('此方程有两个解, x1=%s,x2=%s'%(x1,x2))
            return x1,x2
print('quadratic(2, 3, 1) =', quadratic(2, 3, 1))
```

Read More ▾

FW贝利亚 @ 2025/11/6 17:50:47



输出: 此方程有两个解, x1=-0.5,x2=-1.0 quadratic(2, 3, 1) = (-0.5, -1.0) 此方程有两个解, x1=1.0,x2=-4.0 quadratic(1, 3, -4) = (1.0, -4.0) 此方程有两个解, x1=-0.5,x2=-1.0 此方程有两个解, x1=1.0,x2=-4.0 测试成功

print() 不会立刻打印整句, 它会先执行括号里的所有表达式。 所以函数里的输出会比外层的输出先出现。

Moon @ 2025/11/11 21:26:48



Good, 老A

Moon @ 2025/10/30 03:18:08



参考评论区加入了delta, 粘贴到代码区这回代码终于不乱了

```
import math
def quadratic(a,b,c):
    delta=b**2-4*a*c
    if delta<0:
        raise ValueError('no solution')
    elif delta==0:
        nx=-b/(2*a)
        return nx
    elif delta>0:
        nx1=(-b+math.sqrt(b**2-4*a*c))/2*a
        nx2=(-b-math.sqrt(b**2-4*a*c))/2*a
        return nx1, nx2
```

PYTHON教程

- 1. 简介
- 2. Python历史
- 3. 安装Python
 - 3.1. Python解释器
- 4. 第一个Python程序
 - 4.1. 使用文本编辑器
 - 4.2. 输入和输出
- 5. Python基础
 - 5.1. 数据类型和变量
 - 5.2. 字符串和编码
 - 5.3. 使用list和tuple
 - 5.4. 条件判断
 - 5.5. 模式匹配
 - 5.6. 循环
 - 5.7. 使用dict和set
- 6. 函数
 - 6.1. 调用函数
 - 6.2. 定义函数**
 - 6.3. 函数的参数
 - 6.4. 递归函数
- 7. 高级特性
 - 7.1. 切片
 - 7.2. 迭代
 - 7.3. 列表生成式
 - 7.4. 生成器
 - 7.5. 迭代器
- 8. 函数式编程
- 9. 模块
- 10. 面向对象编程
- 11. 面向对象高级编程
- 12. 错误、调试和测试
- 13. IO编程
- 14. 进程和线程
- 15. 正则表达式
- 16. 常用内建模块
- 17. 常用第三方模块
- 18. 图形界面
- 19. 网络编程
- 20. 电子邮件
- 21. 访问数据库
- 22. Web开发
- 23. 异步IO
- 24. FAQ
- 25. 期末总结

```
nx2=(-b-math.sqrt(b**2-4*a*c))/2*a
return nx1,nx2
# 测试:
print('quadratic(2, 3, 1) =', quadratic(2, 3, 1))
```

Read More ▾

 Jsurobfjakxbsiakbd @ 2025/10/31 04:16:41
/2*a 是不是少个括号

 Moon @ 2025/11/11 21:14:27
还真是，谢谢纠正

 Y² @ 2025/11/9 03:53:03
import math def quadratic(a,b,c): s1=(-b+math.sqrt(b2-(4*a*c)))/(2*a) s2=(-b-math.sqrt(b2-(4*a*c)))/(2*a) return s1,s2

 夜卜 @ 2025/10/28 07:15:08
import math def quadratic(a, b, c):

```
mid = b**2 - 4*a*c
if mid > 0:
    # 两个不相等的实根
    x1 = (-b + math.sqrt(mid)) / (2 * a)
    x2 = (-b - math.sqrt(mid)) / (2 * a)
    return x1,x2
elif mid == 0:
    # 两个相等的实根
    x = -b / (2 * a)
    return x, x
else:
    return "不考虑复数"
```

行简 @ 2025/10/28 05:33:59
def quadratic(a, b, c): if not isinstance(a,(int,float)) or not isinstance(b,(int,float)) or not isinstance(c,(int,float)): raise TypeError('Bad operand type') if a == 0 : print('输入参数不合理，无法给出合理输出') return None else : f = b ** 2 - 4 * a * c

```
if f > 0 :
    x1 = (-b + math.sqrt(f)) / (2 * a)
    x2 = (-b - math.sqrt(f)) / (2 * a)
    print(f'该函数与坐标轴相交, 有两个根, 分别是: {x1:.1f},{x2:.1f}')
    return x1,x2
elif f == 0 :
    x = -b / (2 * a)
    print(f'该函数与坐标轴相切, 有一个根, 是: {x}')
    return x
else :
    print(f'方程在实数范围内无定义, 我们不教')
    return None
```

Read More ▾

PYTHON教程

1. 简介

2. Python历史

3. 安装Python

3.1. Python解释器

4. 第一个Python程序

4.1. 使用文本编辑器

4.2. 输入和输出

5. Python基础

5.1. 数据类型和变量

5.2. 字符串和编码

5.3. 使用list和tuple

5.4. 条件判断

5.5. 模式匹配

5.6. 循环

5.7. 使用dict和set

6. 函数

6.1. 调用函数

6.2. 定义函数

6.3. 函数的参数

6.4. 递归函数

7. 高级特性

7.1. 切片

7.2. 迭代

7.3. 列表生成式

7.4. 生成器

7.5. 迭代器

8. 函数式编程

9. 模块

10. 面向对象编程

11. 面向对象高级编程

12. 错误、调试和测试

13. IO编程

14. 进程和线程

15. 正则表达式

16. 常用内建模块

17. 常用第三方模块

18. 图形界面

19. 网络编程

20. 电子邮件

21. 访问数据库

22. Web开发

23. 异步IO

24. FAQ

25. 期末总结



圆圈 @ 2025/10/28 05:17:07

```
import math
x = float(input('请输入: '))
y = float(input('请输入: '))
z = float(input('请输入: '))
def quadratic(a, b, c):
    if a == 0:
        print('无实数解')
    elif b * b - 4 * a * c <= 0:
        print('无解')
    else:
        x1 = (-b + math.sqrt(b * b - 4 * a * c)) / (2 * a)
        x2 = (-b - math.sqrt(b * b - 4 * a * c)) / (2 * a)
        return x1, x2
print(quadratic(x, y, z))
```



Bacium @ 2025/10/24 23:04:21

```
# 请定义一个函数quadratic(a, b, c), 接收3个参数, 返回一元二次方程
import math

def quadratic(a, b, c):
    if a == 0:
        raise ValueError('a cannot be 0')
    if b * b - 4 * a * c:
        return print("无实数解")
    x1 = (-b + math.sqrt(b * b - 4 * a * c)) / (2 * a)
    x2 = (-b - math.sqrt(b * b - 4 * a * c)) / (2 * a)
    print(x1, x2)
    return x1, x2

quadratic(1, 2, 1)
```



Sfz @ 2025/10/24 04:25:21

```
import math

def quadratic(a, b, c):
    n = b**2 - 4*a*c
    if n < 0:
        print('输入值有误')
        return None
    else:
        x1 = (-b + math.sqrt(n)) / (2*a)
```

[Read More ▾](#)



z. @ 2025/10/22 23:54:33

```
import math

def quadratic(a, b, c):
    delta = b**2 - 4*a*c
    if delta < 0:
        return 'None'
    x1 = (-b + math.sqrt(delta)) / (2*a)
    #math.sqrt(delta)计算平方根
    x2 = (-b - math.sqrt(delta)) / (2*a)
    return x1, x2
```

测试:

```
print('quadratic(2, 3, 1) =', quadratic(2, 3, 1))
print('quadratic(1, 3, -4) =', quadratic(1, 3, -4))

if quadratic(2, 3, 1) != (-0.5, -1.0):
    print('测试失败')
elif quadratic(1, 3, -4) != (1.0, -4.0):
    print('测试失败')
else:
    print('测试成功')
```



可燃乌龙茶 @ 2025/10/22 04:30:11

测试的时候 set(quadratic(2, 3, 1)) != set((-0.5, -1.0)) 使用set可以避免顺序问题



是季不是鸡 @ 2025/10/18 21:56:35

依旧改动

PYTHON教程

- 1. 简介
- 2. Python历史
- 3. 安装Python
- 3.1. Python解释器
- 4. 第一个Python程序
- 4.1. 使用文本编辑器
- 4.2. 输入和输出
- 5. Python基础
- 5.1. 数据类型和变量
- 5.2. 字符串和编码
- 5.3. 使用list和tuple
- 5.4. 条件判断
- 5.5. 模式匹配
- 5.6. 循环
- 5.7. 使用dict和set
- 6. 函数
 - 6.1. 调用函数
 - 6.2. 定义函数
 - 6.3. 函数的参数
 - 6.4. 递归函数
- 7. 高级特性
 - 7.1. 切片
 - 7.2. 迭代
 - 7.3. 列表生成式
 - 7.4. 生成器
 - 7.5. 迭代器
- 8. 函数式编程
- 9. 模块
- 10. 面向对象编程
- 11. 面向对象高级编程
- 12. 错误、调试和测试
- 13. IO编程
- 14. 进程和线程
- 15. 正则表达式
- 16. 常用内建模块
- 17. 常用第三方模块
- 18. 图形界面
- 19. 网络编程
- 20. 电子邮件
- 21. 访问数据库
- 22. Web开发
- 23. 异步IO
- 24. FAQ
- 25. 期末总结

```
def power(m : float, n : float = 2.0) :  
    """  
    计算 m 的 n 次方  
    m: 底数, n: 指数 (支持正数、负数和小数)  
    """  
    # 处理指数为0的情况  
    if n == 0.0 :  
        return 1.0  
  
    # 处理底数为0且指数为负数的情况  
    if m == 0.0 and n < 0 :  
        raise ValueError("0的负指数次方未定义")  
  
    # 如果指数是整数  
    if n == int(n) :  
        result : float = 1.0  
        abs_n = abs(int(n))  
        for _ in range(abs_n) :  
            result = result * m  
        # 如果指数为负数, 取倒数  
        if n < 0 :  
            return 1.0 / result  
        else :  
            return result  
    else :  
        # 如果指数是小数, 使用牛顿迭代法计算  
        # 使用公式: m^n = e^(n * ln(m)), 但不使用math库  
  
        # 特殊情况: 计算平方根 (n=0.5)  
        if n == 0.5 :  
            return newton_sqrt(m)  
        else :  
            # 对于其他小数指数, 使用近似方法  
            # 这里使用二分法来近似计算  
            return approximate_power(m, n)  
  
def newton_sqrt(x : float, precision : float = 1e-10) -> float:  
    """  
    使用牛顿迭代法计算平方根  
    """  
    if x < 0:  
        raise ValueError("不能计算负数的平方根")  
    if x == 0:  
        return 0.0  
  
    # 初始猜测值  
    guess = x / 2.0  
  
    # 迭代直到精度满足要求  
    while True:  
        new_guess = (guess + x / guess) / 2.0  
        if abs(new_guess - guess) < precision:  
            return new_guess  
        guess = new_guess  
  
def approximate_power(base : float, exponent : float, precision : float = 1e-6) -> float:  
    """  
    使用二分法近似计算任意次方  
    这是一个简化的实现, 主要用于学习目的  
    """  
    # 对于正指数, 使用二分法近似  
    if exponent > 0:  
        # 找到上下界  
        low = 0.0  
        high = base if base > 1 else 1.0  
  
        # 二分查找  
        while high - low > precision:
```

PYTHON教程

- 1. 简介
- 2. Python历史
- 3. 安装Python
 - 3.1. Python解释器
- 4. 第一个Python程序
 - 4.1. 使用文本编辑器
 - 4.2. 输入和输出
- 5. Python基础
 - 5.1. 数据类型和变量
 - 5.2. 字符串和编码
 - 5.3. 使用list和tuple
 - 5.4. 条件判断
 - 5.5. 模式匹配
 - 5.6. 循环
 - 5.7. 使用dict和set
- 6. 函数
 - 6.1. 调用函数
 - 6.2. 定义函数**
 - 6.3. 函数的参数
 - 6.4. 递归函数
- 7. 高级特性
 - 7.1. 切片
 - 7.2. 迭代
 - 7.3. 列表生成式
 - 7.4. 生成器
 - 7.5. 迭代器
- 8. 函数式编程 >
- 9. 模块 >
- 10. 面向对象编程 >
- 11. 面向对象高级编程 >
- 12. 错误、调试和测试 >
- 13. IO编程 >
- 14. 进程和线程 >
- 15. 正则表达式 >
- 16. 常用内建模块 >
- 17. 常用第三方模块 >
- 18. 图形界面 >
- 19. 网络编程 >
- 20. 电子邮件 >
- 21. 访问数据库 >
- 22. Web开发 >
- 23. 异步IO >
- 24. FAQ
- 25. 期末总结

```
mid = (low + high) / 2.0
# 这里使用一个简化的近似，实际应该使用更复杂的算法
# 为了简化，我们使用线性插值
if mid ** exponent < base:
    low = mid
else:
    high = mid
return (low + high) / 2.0
else:
    # 对于负指数，先计算正指数再取倒数
    return 1.0 / approximate_power(base, -exponent, precision)

####

def quadratic(a : float, b : float, c : float = 0.0) :
    # 处理a等于零
    if a == 0.0 :
        result : float = -(c / b)
        return result, None
    else :
        # 计算判别式
        discriminant = power(b, 2) - 4 * a * c
        # 处理无实数根情况
        if discriminant < 0 :
            return None, None
        else :
            # 计算平方根（现在可以使用我们自己的 power 函数了！）
            sqrt_discriminant = power(discriminant, 0.5)
            x_1 = (-b + sqrt_discriminant) / (2 * a)
            x_2 = (-b - sqrt_discriminant) / (2 * a)
            return x_1, x_2

# 测试 power 函数
print('power(25, 0.5) =', power(25, 0.5))
print('power(9, 0.5) =', power(9, 0.5))

print('quadratic(2, 3, 1) =', quadratic(2, 3, 1))
print('quadratic(1, 3, -4) =', quadratic(1, 3, -4))

if quadratic(2, 3, 1) != (-0.5, -1.0):
    print('测试失败')
elif quadratic(1, 3, -4) != (1.0, -4.0):
    print('测试失败')
else:
    print('测试成功')
```

[Collapse ▲](#)