


# 图解 Python 编程(24) | 错误与异常处理

🕒 2021-11-09    👁 1111    📄 0    📁 工具教程 python 编程语言

 ShowMeAI用知识加速每一次技术成长

作者：韩信子@ShowMeAI  
教程地址：<https://www.showmeai.tech/tutorials/56>  
本文地址：<https://www.showmeai.tech/article-detail/87>  
声明：版权所有，转载请联系平台与作者并注明出处  
收藏ShowMeAI更多精彩内容

## Python3错误和异常

我们在实际编程过程中，经常会看到一些报错信息，在python中也有专门的方式去处理错误和异常，保证全局流程顺畅。

Python中的语法错误和异常很容易被辨认，我们也可以借助try...except来做相应的处理。



### 语法错误

初学者经常会遇到Python的语法错误或解析错，如下实代码示例

```
1. >>> while True print('Hello ShowMeAI')
2.   File "<stdin>", line 1, in ?
3.     while True print('Hello ShowMeAI')
4.         ^
5. SyntaxError: invalid syntax
```

这个例子中，函数 print() 被检查到有错误，在它前面缺少了一个冒号：。

语法分析器指出了出错的一行，并且在最先找到的错误的位置标记了一个小小的箭头。

### 异常

即便 Python 程序的语法是正确的，在运行它的时候，也有可能发生错误。运行期检测到的错误被称为异常。

大多数的异常都不会被程序处理，都以错误信息的形式展现在这里（下列代码可以在在线python3环境中运行）：

```
1. for i in range(5,-5,-1):
2.     print(5/i)
```

结果如下：

```
1. 1.0
2. 1.25
3. 1.6666666666666667
4. 2.5
5. 5.0
6. Traceback (most recent call last):
7.   File "<string>", line 4, in <module>
8. ZeroDivisionError: division by zero
```

异常以不同的类型出现，这些类型都作为信息的一部分打印出来: 例子中的类型有 ZeroDivisionError，NameError 和 TypeError。

错误信息的前面部分显示了异常发生的上下文，并以调用栈的形式显示具体信息。

### 异常处理

#### try-except

异常捕捉可以使用 try/except 语句。



图解 Python 编程(24) | 错误与异常处理

执行与之对应的处理语句

「try-except」和「try-with-statement」

变量，可通过该变量获得异常的相关信息

搜索 | 微信 ShowMeAI 研究中心

http://www.showmeai.tech/

以下例子中，让用户输入一个合法的整数，但是允许用户中断这个程序（使用 Control-C 或者操作系统提供的方法）。用户中断的信息会引发一个 KeyboardInterrupt 异常。

```
1. while True:
2.     try:
3.         x = int(input("请输入一个数字: "))
4.         break
5.     except ValueError:
6.         print("您输入的不是数字，请再次尝试输入！")
```

try 语句按照如下方式工作：

- 首先，执行 try 子句（在关键字 try 和关键字 except 之间的语句）。
- 如果没有异常发生，忽略 except 子句，try 子句执行后结束。
- 如果在执行 try 子句的过程中发生了异常，那么 try 子句余下的部分将被忽略。如果异常的类型和 except 之后的名称相符，那么对应的 except 子句将被执行。
- 如果一个异常没有与任何的 except 匹配，那么这个异常将会传递给上层的 try 中。

一个 try 语句可能包含多个except子句，分别来处理不同的特定的异常，其中只有一个分支会被执行。

处理程序将只针对对应的 try 子句中的异常进行处理，而不是其他的 try 的处理程序中的异常。

一个except子句可以同时处理多个异常，这些异常将被放在一个括号里成为一个元组，例如：

```
1. except (RuntimeError, TypeError, NameError):
2.     pass
```

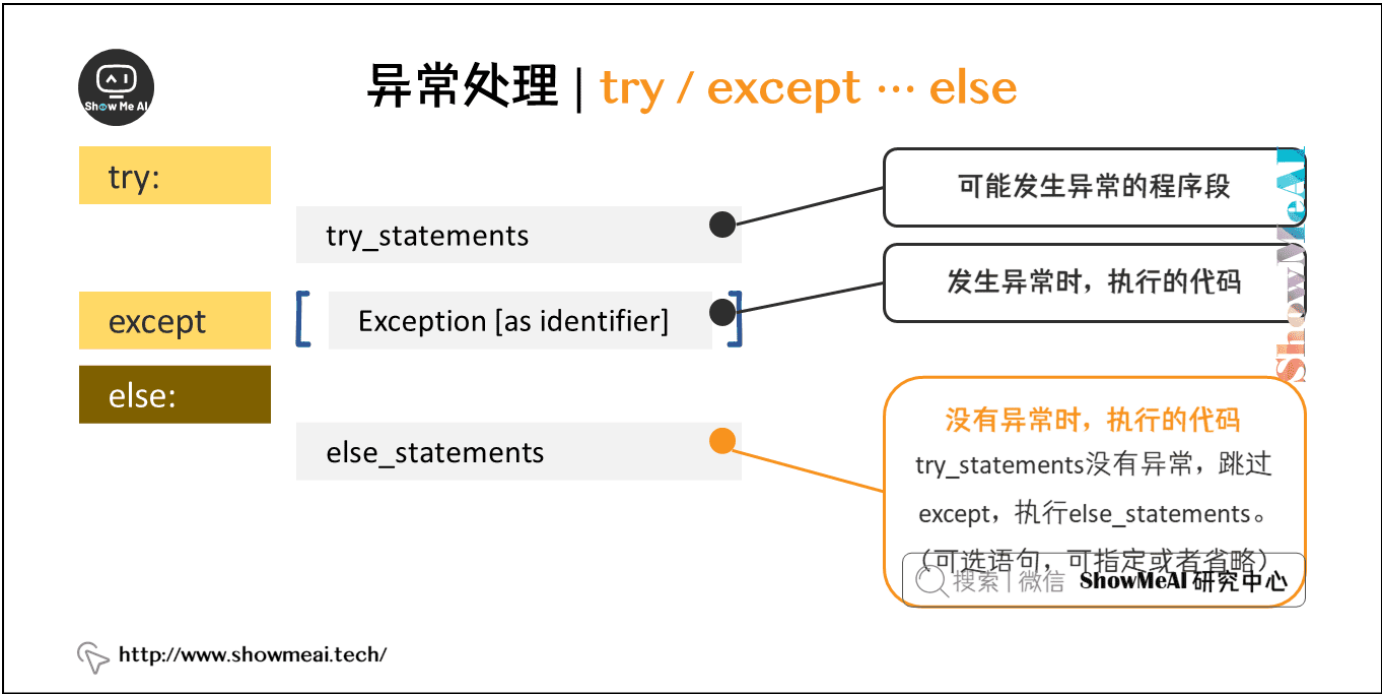
最后一个except子句可以忽略异常的名称，它将被当作通配符使用。你可以使用这种方法打印一个错误信息，然后再再次把异常抛出。

```
1. import sys
2.
3. try:
4.     f = open('ShowMeAI.txt')
5.     s = f.readline()
6.     i = int(s.strip())
7. except OSError as err:
8.     print("系统错误: {}".format(err))
9. except ValueError:
10.    print("无法转换为整型")
11. except:
12.    print("未知错误:", sys.exc_info()[0])
13.    raise
```

try-except-else

try/except 语句还有一个可选的 else 子句，如果使用这个子句，那么必须放在所有的 except 子句之后。

else 子句将在 try 子句没有发生任何异常的时候执行。



以下实例在 try 语句中判断文件是否可以打开，如果打开文件时正常的没有发生异常则执行 else 部分的语句，读取文件内容：

```
1. for arg in sys.argv[1:]:
2.     try:
3.         f = open(arg, 'r')
4.     except IOError:
5.         print('无法打开文件', arg)
6.     else:
7.         print(arg, '有', len(f.readlines()), '行')
8.         f.close()
```

使用 else 子句比比把所有的语句都放在 try 子句里面要好，这样可以避免一些意想不到，而 except 又无法捕获的异常。

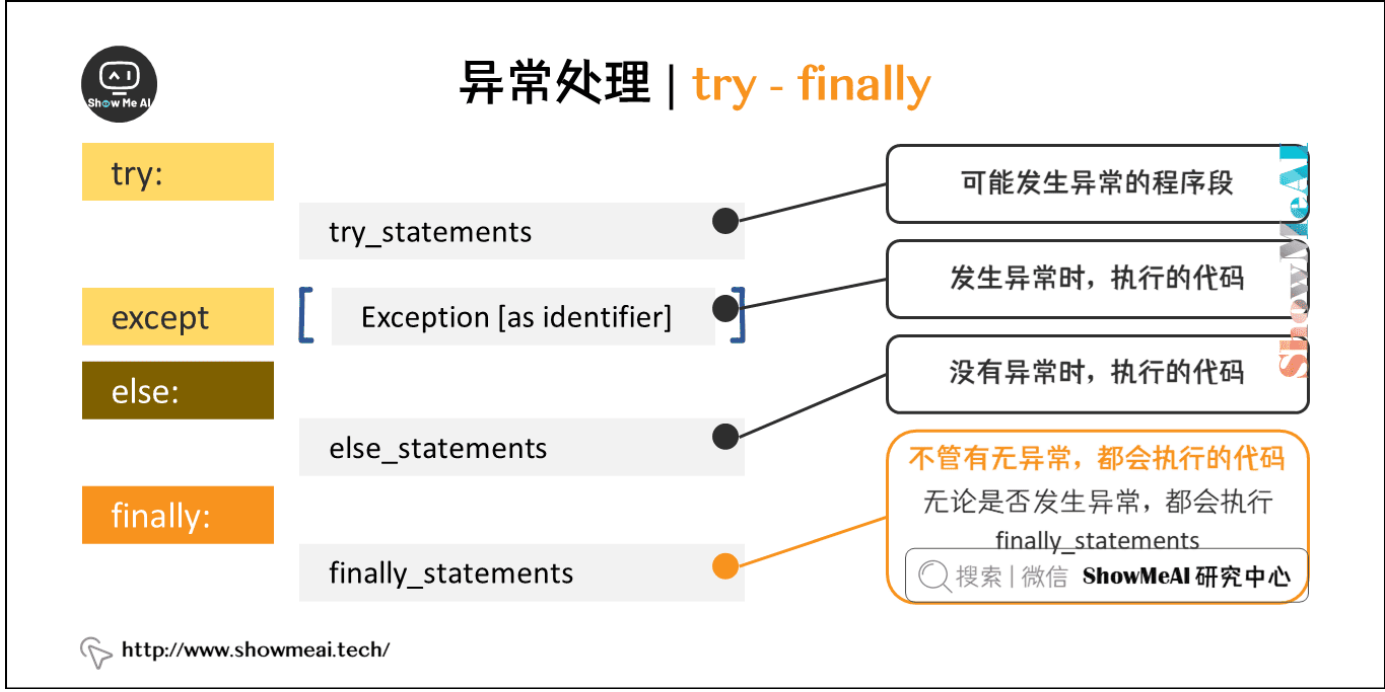
异常处理并不仅仅处理那些直接发生在 try 子句中的异常，而且还能处理子句中调用的函数（甚至间接调用的函数）里抛出的异常。例如：

```
1. >>> def this_fails():
2.     x = 1/0
3.
4. >>> try:
5.     this_fails()
```

```
5.     raise ZeroDivisionError
6.     except ZeroDivisionError as err:
7.         print('出现错误:', err)
8.
9. #出现错误: int division or modulo by zero
```

### try-finally语句

try-finally 语句无论是否发生异常都将执行最后的代码。



以下实例中 finally 语句无论异常是否发生都会执行：

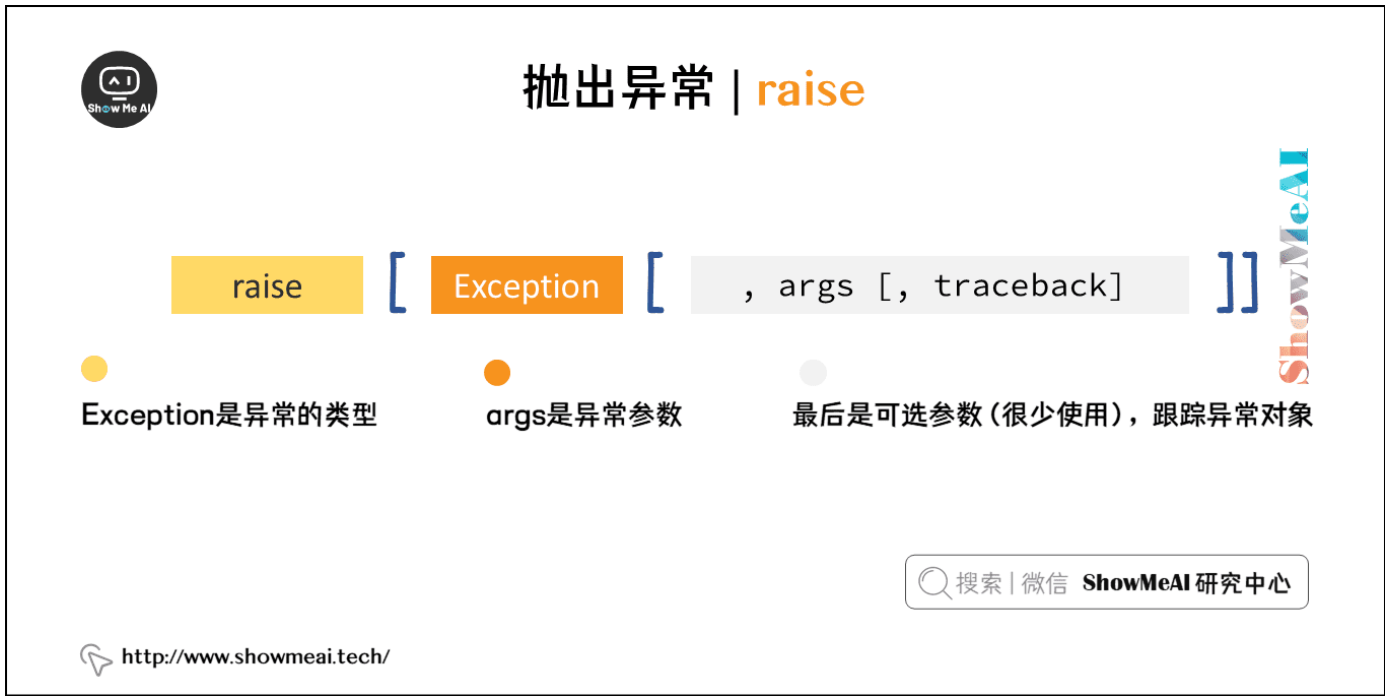
```
1. try:
2.     runoob()
3. except AssertionError as error:
4.     print(error)
5. else:
6.     try:
7.         with open('ShowMeAI.log') as file:
8.             read_data = file.read()
9.         except FileNotFoundError as fnf_error:
10.            print(fnf_error)
11. finally:
12.    print('无论异常是否发生，都会执行本句话。')
```

### 抛出异常

Python 使用 raise 语句抛出一个指定的异常。

raise语法格式如下：

```
1. raise [Exception [, args [, traceback]]]
```



```
7.     raise ValueError, msg % m
10. NameError: NameError
```

## 用户自定义异常

你可以通过创建一个新的异常类来拥有自己的异常。异常类继承自 `Exception` 类，可以直接继承，或者间接继承，例如：

```
1. >>> class NewError(Exception):
2.     def __init__(self, value):
3.         self.value = value
4.     def __str__(self):
5.         return repr(self.value)
6.
7. >>> try:
8.     raise NewError(2*2)
9. except NewError as e:
10.    print('New exception occurred, value:', e.value)
11.
12. My exception occurred, value: 4
13. >>> raise NewError('oops!')
14. Traceback (most recent call last):
15.   File "<stdin>", line 1, in ?
16. __main__.NewError: 'oops!'
```

在这个例子中，类 `Exception` 默认的 `init()` 被覆盖。

当创建一个模块有可能抛出多种不同的异常时，一种通常的做法是为这个包建立一个基础异常类，然后基于这个基础类为不同的错误情况创建不同的子类：

```
1. class Error(Exception):
2.     """Base class for exceptions in this module."""
3.     pass
4.
5. class InputError(Error):
6.     """Exception raised for errors in the input.
7.
8.     Attributes:
9.         expression -- input expression in which the error occurred
10.        message -- explanation of the error
11.    """
12.
13.    def __init__(self, expression, message):
14.        self.expression = expression
15.        self.message = message
16.
17. class TransitionError(Error):
18.     """Raised when an operation attempts a state transition that's not
19.     allowed.
20.
21.     Attributes:
22.         previous -- state at beginning of transition
23.         next -- attempted new state
24.         message -- explanation of why the specific transition is not allowed
25.    """
26.
27.    def __init__(self, previous, next, message):
28.        self.previous = previous
29.        self.next = next
30.        self.message = message
```

大多数的异常的名字都以“Error”结尾，就跟标准的异常命名一样。

## 定义清理行为

`try` 语句还有另外一个可选的子句，它定义了无论在任何情况下都会执行的清理行为。例如：

```
1. >>> try:
2. ...     raise KeyboardInterrupt
3. ... finally:
4. ...     print('Goodbye, ShowMeAI!')
5. ...
6. Goodbye, ShowMeAI!
7. Traceback (most recent call last):
8.   File "<stdin>", line 2, in <module>
9. KeyboardInterrupt
```

以上例子不管 `try` 子句里面有没有发生异常，`finally` 子句都会执行。

如果一个异常在 `try` 子句里（或者在 `except` 和 `else` 子句里）被抛出，而又没有任何的 `except` 把它截住，那么这个异常会在 `finally` 子句执行后被抛出。

下面是一个更加复杂的例子（在同一个 `try` 语句里包含 `except` 和 `finally` 子句）：

```
1. >>> def divide(x, y):
2.     try:
3.         result = x / y
4.     except ZeroDivisionError:
5.         print("除数为0!")
6.     else:
7.         print("结果为", result)
8.     finally:
9.         print("除法计算结束")
10.
11. >>> divide(2, 1)
12. 结果为 2
13. 除法计算结束
```



```
12. 结束/结束
13. 除法计算结束
14. >>> divide(2, 0)
15. 除数为0!
16. 除法计算结束
17. >>> divide("2", "1")
18. 除法计算结束
19. Traceback (most recent call last):
20.   File "<stdin>", line 1, in ?
21.   File "<stdin>", line 3, in divide
22. TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

## 预定义的清理行为

一些对象定义了标准的清理行为，无论系统是否成功的使用了它，一旦不需要它了，那么这个标准的清理行为就会执行。

这面这个例子展示了尝试打开一个文件，然后把内容打印到屏幕上：

```
1. for line in open("ShowMeAI.txt"):
2.     print(line, end="")
```

以上这段代码的问题是，当执行完毕后，文件会保持打开状态，并没有被关闭。

关键词 `with` 语句就可以保证诸如文件之类的对象在使用完之后一定会正确的执行他的清理方法：

```
1. with open("ShowMeAI.txt") as f:
2.     for line in f:
3.         print(line, end="")
```

以上这段代码执行完毕后，就算在处理过程中出问题了，文件 `f` 总是会关闭。

## 视频教程

也可以点击 [这里](#) 到B站查看有【中英字幕】的版本

【双语字幕+资料下载】Python 3全系列...

去bilibili观看

分享

扫一扫 手机看



进入bilibili,一起发弹幕吐槽!

去吐槽

00:00 / 23:16



360P



## 一键运行所有代码

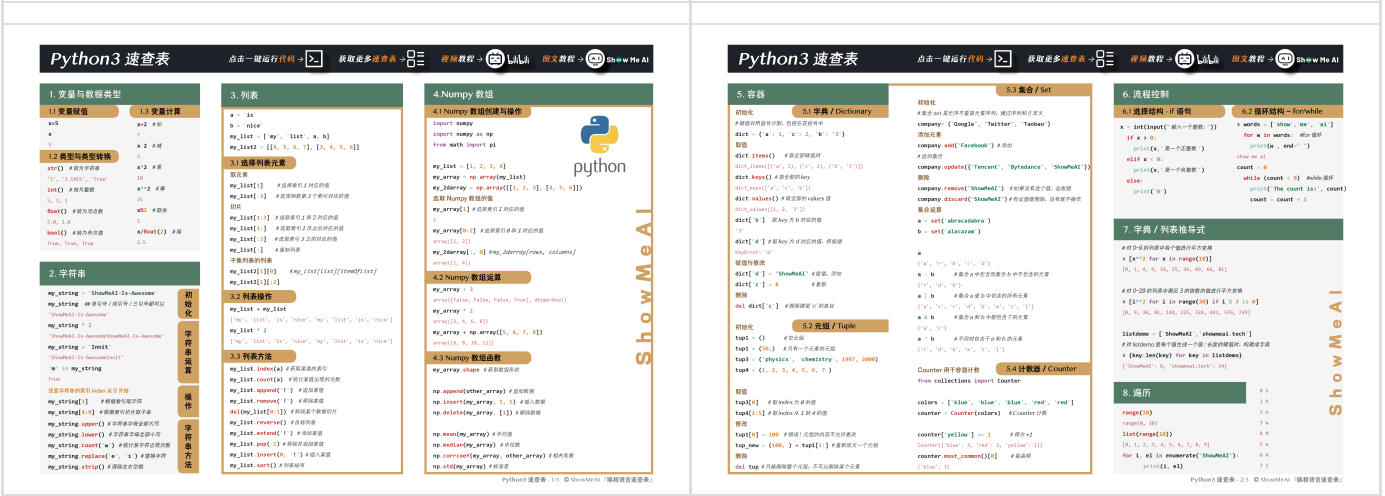
图解Python编程系列 配套的所有代码，可前往ShowMeAI 官方 **GitHub**，下载后即可可在本地 Python 环境中运行。能访问 Google 的宝宝也可以直接借助 Google Colab一键运行与交互学习！

## 下载Python要点速查表

Awesome cheatsheets | **ShowMeAI速查表大全** 系列包含『编程语言』『AI技能知识』『数据科学工具库』『AI垂直领域工具库』四个板块，追平到工具库当前最新版本，并跑通了所有代码。点击 [官网](#) 或 [GitHub](#) 获取~

ShowMeAI速查表大全

Python 速查表（部分）



## 拓展参考资料

- Python教程 - Python3文档
- Python教程 - 廖雪峰的官方网站

## ShowMeAI图解Python编程系列推荐（要点速查版）

- [ShowMeAI 图解 Python 编程\(1\) | 介绍](#)
- [ShowMeAI 图解 Python 编程\(2\) | 安装与环境配置](#)
- [ShowMeAI 图解 Python 编程\(3\) | 基础语法](#)
- [ShowMeAI 图解 Python 编程\(4\) | 基础数据类型](#)
- [ShowMeAI 图解 Python 编程\(5\) | 运算符](#)
- [ShowMeAI 图解 Python 编程\(6\) | 条件控制与if语句](#)
- [ShowMeAI 图解 Python 编程\(7\) | 循环语句](#)
- [ShowMeAI 图解 Python 编程\(8\) | while循环](#)
- [ShowMeAI 图解 Python 编程\(9\) | for循环](#)
- [ShowMeAI 图解 Python 编程\(10\) | break语句](#)
- [ShowMeAI 图解 Python 编程\(11\) | continue语句](#)
- [ShowMeAI 图解 Python 编程\(12\) | pass语句](#)
- [ShowMeAI 图解 Python 编程\(13\) | 字符串及操作](#)
- [ShowMeAI 图解 Python 编程\(14\) | 列表](#)
- [ShowMeAI 图解 Python 编程\(15\) | 元组](#)
- [ShowMeAI 图解 Python 编程\(16\) | 字典](#)
- [ShowMeAI 图解 Python 编程\(17\) | 集合](#)
- [ShowMeAI 图解 Python 编程\(18\) | 函数](#)
- [ShowMeAI 图解 Python 编程\(19\) | 迭代器与生成器](#)
- [ShowMeAI 图解 Python 编程\(20\) | 数据结构](#)
- [ShowMeAI 图解 Python 编程\(21\) | 模块](#)
- [ShowMeAI 图解 Python 编程\(22\) | 文件读写](#)
- [ShowMeAI 图解 Python 编程\(23\) | 文件与目录操作](#)
- [ShowMeAI 图解 Python 编程\(24\) | 错误与异常处理](#)
- [ShowMeAI 图解 Python 编程\(25\) | 面向对象编程](#)
- [ShowMeAI 图解 Python 编程\(26\) | 命名空间与作用域](#)
- [ShowMeAI 图解 Python 编程\(27\) | 时间和日期](#)

## ShowMeAI系列教程精选推荐

- [大厂技术实现：推荐与广告计算解决方案](#)
- [大厂技术实现：计算机视觉解决方案](#)
- [大厂技术实现：自然语言处理行业解决方案](#)
- [图解Python编程：从入门到精通系列教程](#)
- [图解数据分析：从入门到精通系列教程](#)
- [图解AI数学基础：从入门到精通系列教程](#)
- [图解大数据技术：从入门到精通系列教程](#)
- [图解机器学习算法：从入门到精通系列教程](#)
- [机器学习实战：手把手教你玩转机器学习系列](#)
- [深度学习教程：吴恩达专项课程·全套笔记解读](#)
- [自然语言处理教程：斯坦福CS224n课程·课程带学与全套笔记解读](#)
- [深度学习与计算机视觉教程：斯坦福CS231n·全套笔记解读](#)

## 图解 Python 编程(24) | 错误与异常处理

[< 上一篇](#)

[图解 Python 编程\(23\) | 文件与目录操作](#)

[下一篇 >](#)

[图解 Python 编程\(25\) | 面向对象编程](#)