# Python Cheat Sheet-10.异常-Errors and Exceptions

?> PCS-10（    ）：10.1 内置异常（Built-in Exceptions）；10.2 处理异常的方式

参考[Errors and Exceptions](https://docs.python.org/3/tutorial/errors.html)[1]

知识点

描述

代码段

运算结果

备注

**10.1** 内置异常（Built-in Exceptions）

参考[Built-in Exceptions](https://docs.python.org/3/library/exceptions.html)[2]

for i in range(10) print(i)代码缺少了:，为语法/句法错误（syntax error），会引发内置异常SyntaxError错误，并通常会给出错误的详细原因，例如invalid syntax等。反馈的异常信息中，通常会标识行号，并用^等符号标示错误位置，方便快速定位修改。

for i in range(10) print(i)

Input In [14]  
 for i in range(10) print(i)  
 ^  
SyntaxError: invalid syntax

又例如索引值5超出了lst列表索引数，引发IndexError异常。

lst=[1,2,3,4,5]  
element=lst[5]

---------------------------------------------------------------------------  
  
IndexError Traceback (most recent call last)  
  
Input In [16], in <cell line: 2>()  
 1 lst=[1,2,3,4,5]  
----> 2 element=lst[5]  
  
  
IndexError: list index out of range

内置异常的层次结构如下：

BaseException 所有内置异常的基类  
 +-- SystemExit 此异常由 sys.exit() 函数引发  
 +-- KeyboardInterrupt 当用户按下中断键 (通常为 Control-C 或 Delete) 时将被引发  
 +-- GeneratorExit 当一个 generator 或 coroutine 被关闭时将被引发  
 +-- Exception 所有内置的非系统退出类异常都派生自此类  
 +-- StopIteration 由内置函数 next() 和 iterator 的 \_\_next\_\_() 方法所引发，用来表示该迭代器不能产生下一项  
 +-- StopAsyncIteration 必须由一个 asynchronous iterator 对象的 \_\_anext\_\_() 方法来引发以停止迭代操作  
 +-- ArithmeticError 此基类用于派生针对各种算术类错误而引发的内置异常  
 | +-- FloatingPointError 目前未被使用  
 | +-- OverflowError 当算术运算的结果大到无法表示时将被引发  
 | +-- ZeroDivisionError 当除法或取余运算的第二个参数为零时将被引发  
 +-- AssertionError 当 assert 语句失败时将被引发  
 +-- AttributeError 当属性引用或赋值失败时将被引发  
 +-- BufferError 当与 缓冲区(buffer) 相关的操作无法执行时将被引发  
 +-- EOFError 当 input() 函数未读取任何数据即达到文件结束条件 (EOF) 时将被引发  
 +-- ImportError 当 import 语句尝试加载模块遇到麻烦时将被引发  
 | +-- ModuleNotFoundError ImportError 的子类，当一个模块无法被定位时将由 import 引发  
 +-- LookupError 此基类用于派生当映射或序列所使用的键或索引无效时引发的异常  
 | +-- IndexError 当序列抽取超出范围时将被引发  
 | +-- KeyError 当在现有键集合中找不到指定的映射（字典）键时将被引发  
 +-- MemoryError 当一个操作耗尽内存但情况仍可（通过删除一些对象）进行挽救时将被引发  
 +-- NameError 当某个局部或全局名称未找到时将被引发  
 | +-- UnboundLocalError 当在函数或方法中对某个局部变量进行引用，但该变量并未绑定任何值时将被引发  
 +-- OSError 此异常在一个系统函数返回系统相关的错误时将被引发，此类错误包括 I/O 操作失败例如 "文件未找到" 或 "磁盘已满" 等（不包括非法参数类型或其他偶然性错误）  
 | +-- BlockingIOError 当一个操作将会在设置为非阻塞操作的对象（例如套接字）上发生阻塞时将被引发  
 | +-- ChildProcessError 当一个子进程上的操作失败时将被引发  
 | +-- ConnectionError 与连接相关问题的基类  
 | | +-- BrokenPipeError ConnectionError 的子类，当试图写入一个管道而该管道的另一端已关闭，或者试图写入一个套接字而该套接字已关闭写入时将被引发  
 | | +-- ConnectionAbortedError ConnectionError 的子类，当一个连接尝试被对端中止时将被引发  
 | | +-- ConnectionRefusedError ConnectionError 的子类，当一个连接尝试被对端拒绝时将被引发  
 | | +-- ConnectionResetError ConnectionError 的子类，当一个连接尝试被对端重置时将被引发  
 | +-- FileExistsError 当试图创建一个已存在的文件或目录时将被引发  
 | +-- FileNotFoundError 将所请求的文件或目录不存在时将被引发  
 | +-- InterruptedError 当系统调用被输入信号中断时将被引发  
 | +-- IsADirectoryError 当请求对一个目录执行文件操作 (例如 os.remove()) 时将被引发  
 | +-- NotADirectoryError 当请求对一个非目录执行目录操作 (例如 os.listdir()) 时将被引发  
 | +-- PermissionError 当在没有足够访问权限的情况下试图执行某个操作时将被引发 —— 例如文件系统权限  
 | +-- ProcessLookupError 当给定的进程不存在时将被引发  
 | +-- TimeoutError 当一个系统函数在系统层级发生超时的情况下将被引发  
 +-- ReferenceError 此异常将在使用 weakref.proxy() 函数所创建的弱引用来访问该引用的某个已被作为垃圾回收的属性时被引发  
 +-- RuntimeError 当检测到一个不归属于任何其他类别的错误时将被引发  
 | +-- NotImplementedError 此异常派生自 RuntimeError。在用户自定义的基类中，抽象方法应当在其要求所派生类重载该方法，或是在其要求所开发的类提示具体实现尚待添加时引发此异常  
 | +-- RecursionError 此异常派生自 RuntimeError。 它会在解释器检测发现超过最大递归深度时被引发  
 +-- SyntaxError 当解析器遇到语法错误时引发  
 | +-- IndentationError 与不正确的缩进相关的语法错误的基类  
 | +-- TabError 当缩进包含对制表符和空格符不一致的使用时将被引发  
 +-- SystemError 当解释器发现内部错误，但情况看起来尚未严重到要放弃所有希望时将被引发  
 +-- TypeError 当一个操作或函数被应用于类型不适当的对象时将被引发  
 +-- ValueError 当操作或函数接收到具有正确类型但值不适合的参数，并且情况不能用更精确的异常例如 IndexError 来描述时将被引发  
 | +-- UnicodeError 当发生与 Unicode 相关的编码或解码错误时将被引  
 | +-- UnicodeDecodeError 当在解码过程中发生与 Unicode 相关的错误时将被引发  
 | +-- UnicodeEncodeError 当在编码过程中发生与 Unicode 相关的错误时将被引发  
 | +-- UnicodeTranslateError 在转写过程中发生与 Unicode 相关的错误时将被引发  
 +-- Warning 警告类别的基类  
 +-- DeprecationWarning 如果所发出的警告是针对其他 Python 开发者的，则以此作为与已弃用特性相关警告的基类  
 +-- PendingDeprecationWarning 对于已过时并预计在未来弃用，但目前尚未弃用的特性相关警告的基类  
 +-- RuntimeWarning 与模糊的运行时行为相关的警告的基类  
 +-- SyntaxWarning 与模糊的语法相关的警告的基类  
 +-- UserWarning 用户代码所产生警告的基类  
 +-- FutureWarning 如果所发出的警告是针对以 Python 所编写应用的最终用户的，则以此作为与已弃用特性相关警告的基类  
 +-- ImportWarning 与在模块导入中可能的错误相关的警告的基类  
 +-- UnicodeWarning 与 Unicode 相关的警告的基类  
 +-- BytesWarning 与 bytes 和 bytearray 相关的警告的基类  
 +-- EncodingWarning 与编码格式相关的警告的基类  
 +-- ResourceWarning 资源使用相关警告的基类

**10.2** 处理异常的方式

**1.**

try:  
 statements  
except [built-in exception/(exceptions)]:  
 statements

首先执行try代码块；如果没有触发异常，则跳过except代码块，执行完try代码块。如果在执行try代码块时发生了异常，则跳过代码块中剩余的部分。如果触发的异常与except关键字后指定的异常相匹配，则会执行except代码块，然后跳到try/except代码块之后执行；如果触发的异常与except语句中指定的异常不匹配，则它会被传递到外部的try语句中。如果没有找到处理程序，则是一个未处理异常且终止程序。

while True:  
 try:  
 x=int(input('Please enter a number:'))  
 break  
 except ValueError:  
 print('Oops! That was no valid number. Try again...')

Please enter a number: d  
  
  
Oops! That was no valid number. Try again...  
  
  
Please enter a number: 3

while True:  
 try:  
 x=9/int(input('Please enter a number:'))  
 break  
 except ValueError:  
 print('Oops! That was no valid number. Try again...')

Please enter a number: 0  
  
  
  
---------------------------------------------------------------------------  
  
ZeroDivisionError Traceback (most recent call last)  
  
Input In [18], in <cell line: 2>()  
 1 while True:  
 2 try:  
----> 3 x=9/int(input('Please enter a number:'))  
 4 break  
 5 except ValueError:  
  
  
ZeroDivisionError: division by zero

except后可以用()追加多个异常，只要满足其中一个，就执行except代码块。

while True:  
 try:  
 x=9/int(input('Please enter a number:'))  
 break  
 except (ValueError, ZeroDivisionError): #或者使用ArithmeticError  
 print('Oops! That was no valid number or 0. Try again...')

Please enter a number: 0  
  
  
Oops! That was no valid number or 0. Try again...  
  
  
Please enter a number: 7

如果except后不指定异常，则触发任何存在的异常。

while True:  
 try:  
 x=9/int(input('Please enter a number:'))  
 break  
 except:   
 print('Oops! That was no valid number or 0. Try again...')

Please enter a number: d  
  
  
Oops! That was no valid number or 0. Try again...  
  
  
Please enter a number: 0  
  
  
Oops! That was no valid number or 0. Try again...  
  
  
Please enter a number: 3

**2.**

try:  
 statements  
except exception as alias:  
 statements  
except exception(s):  
 statements  
except:  
 statements  
...

可以有多个except，根据异常不同执行不同的代码块。可以用except exception as alias:方式为异常定义别名(变量)，该变量绑定到一个异常实例并将参数存储在instance.args中。为了能够直接调入存储的参数而不必引用.args，该异常实例定义了\_\_str\_\_()，从而可以直接用定义的变量读取参数值。

import sys  
  
try:  
 f = open('myfile.txt')  
 s = f.readline()  
 i = int(s.strip())  
except OSError as err:  
 print("OS error: {0}".format(err))  
except ValueError:  
 print("Could not convert data to an integer.")  
except BaseException as err:  
 print(f"Unexpected {err=}, {type(err)=}")  
 raise

1.5  
3

**3.**

try:  
 statements  
except:  
 statements  
else:  
 statements

try...except后可以跟随else， 当try代码块没有引发异常，但又必须执行的代码可以放置在else代码块中。如果将必须执行的代码块放置于try中，则可能会意外捕捉到try...except语句保护的代码触发的异常。

def fetcher(obj,index):  
 print(obj[index])  
  
try:  
 obj=[9/i for i in range(1,10)]  
 index=5  
 fetcher(obj,index)   
except IndexError:  
 print('ndexError:{}'.format(IndexError))  
else:  
 fetcher(list(range(10)),3)

1.5  
3

**4.触发异常（Raising Exceptions）**

raise可以强制触发指定的异常。raise唯一的参数就是触发的异常。

raise NameError('HiThere')

---------------------------------------------------------------------------  
  
NameError Traceback (most recent call last)  
  
Input In [44], in <cell line: 1>()  
----> 1 raise NameError('HiThere')  
  
  
NameError: HiThere

通过raise触发异常，并在except下打印该异常实例参数。

try:  
 raise Exception('spam', 'eggs')  
except Exception as inst:  
 print(type(inst)) # the exception instance  
 print(inst.args) # arguments stored in .args  
 print(inst) # \_\_str\_\_ allows args to be printed directly,  
 # but may be overridden in exception subclasses  
 x, y = inst.args # unpack args  
 print('x =', x)  
 print('y =', y)

<class ‘Exception’> (‘spam’, ‘eggs’) (‘spam’, ‘eggs’) x = spam y = eggs

如果只想判断是否触发了异常，但并不打算处理该异常，则可以使用更简单的 raise 语句重新触发异常。

try:  
 raise NameError('HiThere')  
except NameError:  
 print('An exception flew by!')  
 raise

An exception flew by!  
  
  
  
---------------------------------------------------------------------------  
  
NameError Traceback (most recent call last)  
  
Input In [45], in <cell line: 1>()  
 1 try:  
----> 2 raise NameError('HiThere')  
 3 except NameError:  
 4 print('An exception flew by!')  
  
  
NameError: HiThere

**5.异常链（Exception Chaining）**

raise 语句支持可选的from 子句，该子句用于启用链式异常

def func():  
 raise ConnectionError  
  
try:  
 func()  
except ConnectionError as exc:  
 raise RuntimeError('Failed to open database') from exc

---------------------------------------------------------------------------  
  
ConnectionError Traceback (most recent call last)  
  
Input In [46], in <cell line: 4>()  
 4 try:  
----> 5 func()  
 6 except ConnectionError as exc:  
  
  
Input In [46], in func()  
 1 def func():  
----> 2 raise ConnectionError  
  
  
ConnectionError:   
  
  
The above exception was the direct cause of the following exception:  
  
  
RuntimeError Traceback (most recent call last)  
  
Input In [46], in <cell line: 4>()  
 5 func()  
 6 except ConnectionError as exc:  
----> 7 raise RuntimeError('Failed to open database') from exc  
  
  
RuntimeError: Failed to open database

**6.自定义异常（User-defined Exceptions）**

通过定义内置异常类，通常为Exception的子类来自定义异常。通常异常命名以Error结尾，类似标准异常的命名。同时，可以定义\_\_str\_\_()类，附加状态信息或者方法。

class AlreadyGotOneError(Exception):   
 def \_\_str\_\_(self):return 'So you got an exception...'  
 pass #自定义异常  
  
def grail():  
 raise AlreadyGotOneError #引发自定义异常  
  
try:  
 grail()  
except AlreadyGotOneError as ago\_e:  
 print(ago\_e)  
 print('got exception!')

So you got an exception...  
got exception!

**7.定义清理操作(Defining Clean-up Actions)**

try:  
 statements  
finally:  
 statements

如果存在 finally 子句，则 finally 子句是 try 语句结束前执行的最后一项任务。不论 try 语句是否触发异常，都会执行 finally 子句。以下内容介绍了几种比较复杂的触发异常情景：

* 如果执行 try 子句期间触发了某个异常，则某个 except 子句应处理该异常。如果该异常没有 except 子句处理，在 finally 子句执行后会被重新触发;
* except 或 else 子句执行期间也会触发异常。 同样，该异常会在 finally 子句执行之后被重新触发;
* finally 子句中包含 break、continue 或 return 等语句，异常将不会被重新引发;
* 如果执行 try 语句时遇到 break,、continue 或 return 语句，则 finally 子句在执行 break、continue 或 return 语句之前执行;
* 如果 finally 子句中包含 return 语句，则返回值来自 finally 子句的某个 return 语句的返回值，而不是来自 try 子句的 return 语句的返回值。

try:  
 raise KeyboardInterrupt  
finally:  
 print('Goodbye, world!')

Goodbye, world!  
  
  
  
---------------------------------------------------------------------------  
  
KeyboardInterrupt Traceback (most recent call last)  
  
Input In [58], in <cell line: 1>()  
 1 try:  
----> 2 raise KeyboardInterrupt  
 3 finally:  
 4 print('Goodbye, world!')  
  
  
KeyboardInterrupt:

def bool\_return():  
 try:  
 return True  
 finally:  
 return False  
  
bool\_return()

False

从下述案例可以看出，不管是不是触发了异常，finally代码块都会执行。

def divide(x, y):  
 try:  
 result = x / y  
 except ZeroDivisionError:  
 print("division by zero!")  
 else:  
 print("result is", result)  
 finally:  
 print("executing finally clause")  
  
divide(2, 1)

result is 2.0  
executing finally clause

divide(2, 0)

division by zero!  
executing finally clause

divide("2", "1")

executing finally clause  
  
  
  
---------------------------------------------------------------------------  
  
TypeError Traceback (most recent call last)  
  
Input In [62], in <cell line: 1>()  
----> 1 divide("2", "1")  
  
  
Input In [60], in divide(x, y)  
 1 def divide(x, y):  
 2 try:  
----> 3 result = x / y  
 4 except ZeroDivisionError:  
 5 print("division by zero!")  
  
  
TypeError: unsupported operand type(s) for /: 'str' and 'str'

**8.预定义的清理操作(Predefined Clean-up Actions)**

某些对象定义了不需要该对象时要执行的标准清理操作。无论使用该对象的操作是否成功，都会执行清理操作。

例如下述案例，在语句执行完毕后，即使在处理时遇到问题，也都会关闭文件f。

with open("myfile.txt") as f:  
 for line in f:  
 print(line, end="")

参考文献（References）:

[1] Errors and Exceptions, <https://docs.python.org/3/tutorial/errors.html>.

[2] Built-in Exceptions, <https://docs.python.org/3/library/exceptions.html>.

PC10-ipynb download