异常处理

什么是异常

运行程序时，可能会发生各种错误，主要有两大类：

- **语法错误 (Syntax Errors):** 在程序执行前，解释器就能检测到的错误，通常是代码不符合 Python 语法规则导致的。

  例如：print(“Hello” (缺少右括号)。这种错误会导致程序无法执行。

- **异常 (Exceptions):** 在程序运行时发生的错误，一般是指**语法可能没有问题**，但由于**某些条件不被满足**而导致程序**非正常中断**的错误类型。当异常发生时，如果程序没有对其进行处理，就会导致程序“崩溃”并停止执行。

  例如：尝试打开一个不存在的文件、除数为零、访问列表越界等。

什么是异常处理

- **异常处理** 是一种编程机制，允许程序在运行时遇到异常时，不立即崩溃或停止，而是能够**优雅地捕获、识别并响应这些错误**，从而使程序能够继续执行或以受控的方式终止。

- **为什么需要异常处理？**
  - **提高程序的健壮性：** 避免程序因意外情况而崩溃。
  - **改善用户体验：** 当发生错误时，向用户提供有意义的提示，而不是一堆看不懂的错误信息。
  - **分离错误处理逻辑与正常业务逻辑：** 异常处理结构使代码结构更清晰，更易于阅读和维护。

异常场景

例如：

```
short_list = [1, 72, 3]

position = 6

print(short_list[position])
```

程序由于访问了不存在的列表元素，而发生下标越界异常。

```
Traceback (most recent call last):
  File "<string>", line 3, in <module>
IndexError: list index out of range
```

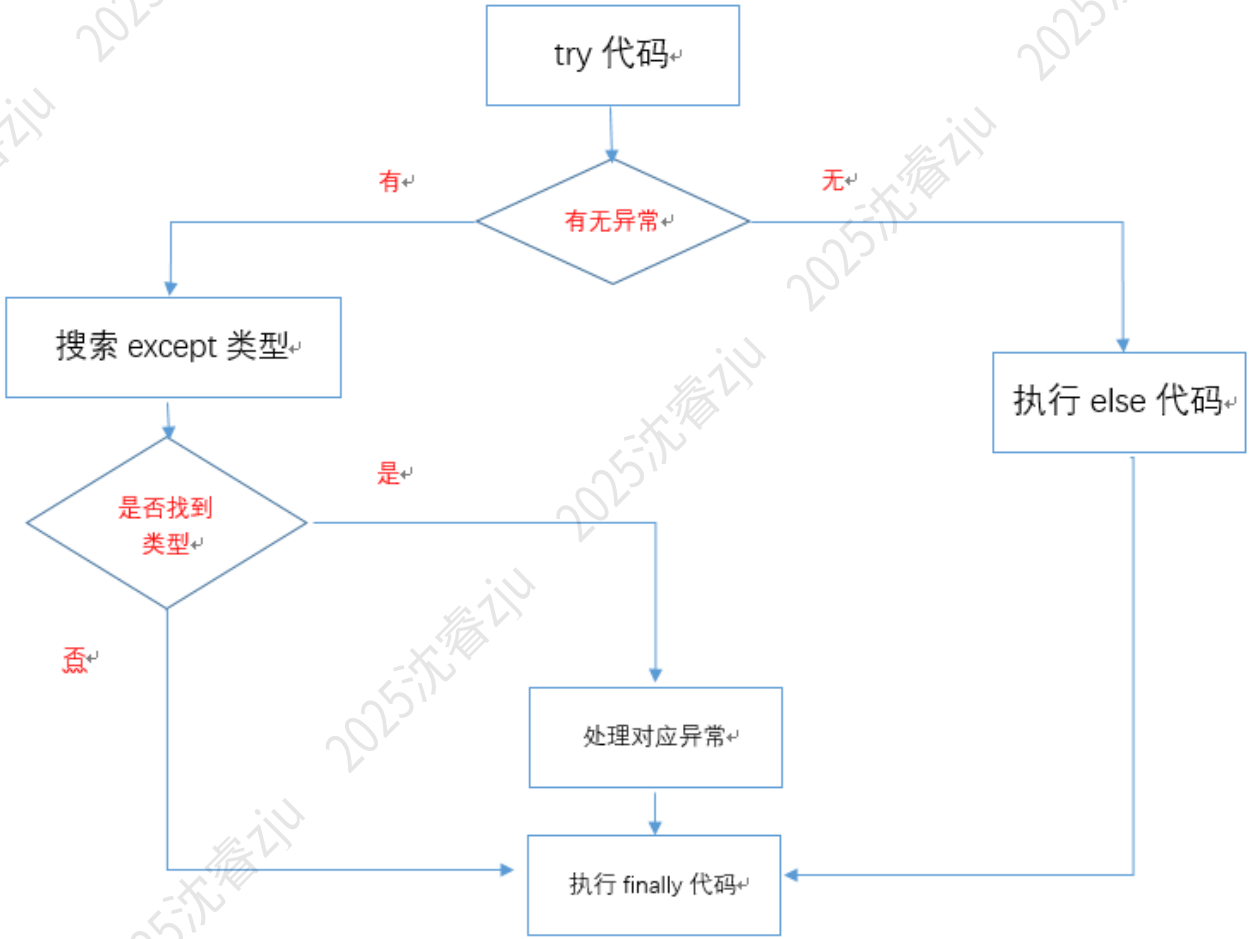

异常处理

- 异常处理程序可以处理完程序异常之后继续正常执行，不至于因异常导致退出或崩溃。
- **try-except** 语句实现异常处理。

```
short_list = [1, 72, 3]
position = 6
try:
    print(short_list[position])
except:
    print(f'索引应该在0到{len(short_list)-1}之间')
```


语法格式

```
try:
    语句块1
except 异常类型1:
    语句块2
except 异常类型2:
    语句块3
…
except 异常类型N:
    语句块N+1
except:
    语句块N+2
else:
    语句块N+3
finally:
    语句块N+4
```



说明

- 正常程序在语句块1中执行。
- 如果程序执行中发生异常，中止程序运行，跳转到所对应的异常处理块中执行。
- 在“except 异常类型 ”语句中找对应的异常类型，如果找到的话，执行后面的语句块。
- 如果找不到，则执行“except”后面的语句块N+2。
- 如果程序正常执行没有发生异常，则继续执行else后的语句块N+3。
- 无论异常是否发生，最后都执行finally后面语句块N+4。

```
short_list = [1, 72, 3]
position = int(input())
try:
    k = short_list[position]
except:
    print(f'索引应该在0到{len(short_list)-1}之间')
else:
    print(k)
finally:
    print("代码执行完毕，无论是否发生异常。")
```

```
5
索引应该在0到2之间
代码执行完毕，无论是否发生异常。
```

```
0
1
代码执行完毕，无论是否发生异常。
```


处理多种异常

```
try:
    num_str = input("请输入一个整数: ")
    num = int(num_str)
    result = 10 / num
except ValueError:
    print("错误! 请输入有效的整数。")
except ZeroDivisionError:
    print("错误! 除数不能为零。")
else:
    print(f"结果是: {result}")
```

```
请输入一个整数: a
错误! 请输入有效的整数。
```

```
请输入一个整数: 0
错误! 除数不能为零。
```

```
请输入一个整数: 10
结果是: 1.0
```


获取所有异常

- 当想要捕获所有类型的异常时，可以使用： except Exception as e:

```
while True:
    try:
        a = int(input())
        b = int(input())
        value = a / b
        print(value)
    except Exception as e:
        print("invalid input:",e)
    else:
        break
```

说明：

Exception 是 Python 中所有内建异常的基类。因此，捕获 Exception 会捕获所有类型的异常。

as e 的作用是将捕获到的异常对象赋值给变量 e，这样就可以访问异常的详细信息，例如错误消息。

虽然捕获所有异常很方便，但在实际编程中通常**不推荐**这样做。

10

常见内建异常类型

异常名称	描述
TypeError	对类型无效的操作
ZeroDivisionError	除(或取模)零 (所有数据类型)
NameError	名称错误：当尝试访问未定义的变量时发生。
ImportError	导入模块/对象失败
IndexError	序列中超出有效索引(index)
KeyError	键错误：当尝试访问字典中不存在的键时发生。
ValueError	传入无效的参数
FileNotFoundError	文件未找到错误：当尝试打开不存在的文件时发生。

手动设置异常

某些情况下，我们还会**主动地**在特定条件下引发异常。这通常是为了**让程序更明确、更健壮、更易于维护。**

raise 表达式　　(表达式描述引发的异常)

```
def area(r):
    if r>=0:
        s=3.14159*r*r
        return s
    else:
        raise ValueError("参数错误，半径小于0")

r=float(input())
try:
    print(area(r))
except ValueError as msg:
    print(msg)
```

示例：手动设置多种异常

场景： 验证输入的用户年龄是否合法，要求年龄必须是大于 0 的整数。

```
def process_age(age):
    if not isinstance(age, int):
        raise TypeError("年龄必须是整数")
    if age <= 0:
        raise ValueError("年龄必须是大于 0 的整数")
    print(f"合法年龄: {age}")
```