错误分类

错误一般可分为两种：

- 语法错误：又称解析错误（语法分析器检查到的错误）
- 异常：在运行时检测到的错误（程序的语法是正确的）

内置异常

- 查看内置异常

```
import builtins



print(dir(builtins))
```

处理异常

try ... except ...

- 如果在执行 try 子句时没有异常发生，则不会执行 except 子句
- 如果 try 子句发生了异常，则跳过该子句中剩下的部分，执行 except 子句
- 如果 except 子句没有指定异常类型，则可以处理 try 中的所有异常类型
- 如果 except 子句指定了异常类型，则只能处理对应的异常类型（指定多个异常类型时，可以用元组来表示）
- 如果一个异常没有与任何的 except 匹配，则报错

```
def div(a, b):
    try:
        c = a / b
        print(f"{a} / {b} = {c}")

    except:
        print('try中发生异常')


div(2, 1)
div(2, 0)
div('2', 2)
```

```
def div(a, b):
    try:
        c = a / b
        print(f"{a} / {b} = {c}")

    except ZeroDivisionError:
        print('try中发生了除数为0的异常')

    except TypeError:
        print('try中发生了类型异常')
```

```
div(2, 1)
div(2, 0)
div('2', 2)
```

```
def div(a, b):
    try:
        c = a / b
        print(f"{a} / {b} = {c}")

    except (ZeroDivisionError, TypeError):
        print('try中发生了除数为0的异常或者类型异常')


div(2, 1)
div(2, 0)
div('2', 2)
```

try ... except ... 嵌套

- 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

```
def div(a, b):
    try:
        try:
            c = a / b
            print(f"{a} / {b} = {c}")

        except ZeroDivisionError:
            print('try中发生了除数为0的异常')
    except:
        print('发生了除0以外的异常')


div('2', 2)
```

try ... except ... else ...

- else 子句必须放在所有的 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

```
def div(a, b):
    try:
        c = a / b
        print(f"{a} / {b} = {c}")

    except ZeroDivisionError:
        print('try中发生了除数为0的异常')

    except:
        print('发生了除0以外的异常')

    else:
        print('try中没有异常')
```

```
div(2, 1)
div(2, 0)
div('2', 2)
```

try ... except ... as ...

- as 后面为异常实例对象的名称

```
def div(a, b):
    try:
        c = a / b
        print(f"{a} / {b} = {c}")

    # e: ZeroDivisionError('division by zero')
    except ZeroDivisionError as e:
        print(isinstance(e, ZeroDivisionError))
        print(e)
        print(ZeroDivisionError('division by zero'))


div(2, 0)
```

try ... finally ...

- finally 子句将作为 try 语句结束前的最后一项任务被执行
- 不论 try 语句是否产生了异常都会被执行

- 如果 finally 子句中包含一个 return 语句，则返回值将来自 finally 子句中的 return 语句的返回值，而非来自 try 子句中的 return 语句的返回值

```
def div(a, b):
    try:
        c = a / b
        print(f"{a} / {b} = {c}")

    finally:
        print("执行finally子句")


div(2, 0)
```

```
def div(a, b):
    try:
        c = a / b
        print(f"{a} / {b} = {c}")

    except:
        print('try中发生异常')

    finally:
        print("执行finally子句")


div(2, 0)
```

```python
def return_num():
    try:
        return 1

    finally:
        return 2


print(return_num())
```

```python
def div(a, b):
    try:
        c = a / b
        print(f"{a} / {b} = {c}")

    except:
        print('except在发生异常时执行')

    else:
        print('else在没有异常时执行')

    finally:
        print('finally在任何情况下都会被执行')


div(2, 1)
div(2, 0)
```

抛出异常

- raise 语句可以主动的抛出异常
- raise 后面可以是 异常实例 / 异常类 / 没有内容

```
def div(a, b):
    if b == 0:
        raise ZeroDivisionError('除数为0')

    c = a / b
    print(f"{a} / {b} = {c}")


div(2, 1)
div(2, 0)
```

```
def div(a, b):
    if b == 0:
        raise ZeroDivisionError

    c = a / b
    print(f"{a} / {b} = {c}")


div(2, 1)
div(2, 0)
```

```
def div(a, b):
    if b == 0:
        raise

    c = a / b
    print(f"{a} / {b} = {c}")


div(2, 1)
div(2, 0)
```

自定义异常

- 自定义的异常类通常需要直接或间接的继承 Exception 类

```
class MyError:
    def __init__(self, message=''):
        self.message = message

    def __str__(self):
        return str(self.message)


class MyError2(Exception):
    pass


print(MyError("发生了一个异常"))
print(MyError2("发生了一个异常"))
```

assert 断言

- assert 用于判断一个表达式，在表达式为 False 的时候触发 AssertionError 异常
- assert expression 等价于
  if not expression: raise AssertionError
- assert expression [, arguments]
  等价于：if not expression: raise AssertionError(arguments)

```
num = int(input("请输入一个整数: "))
assert num != 1
print("断言条件为True, 用户没有输入1")
```

```
num = int(input("请输入一个整数: "))
assert num != 1, "用户不能输入1"
print("断言条件为True, 用户没有输入1")
```