python基础9_异常和模块

一、异常

1. 异常简介

- **异常不是语法错误**，语法错误是程序写错了，异常是指程序已经运行后的非语法错误

```
"""
- 异常：不是语法错误，语法错误，是程序写错了
- 异常：指程序已经运行了(没有语法错误)，突然发生异常，导致程序崩溃
"""
```

```
# 语法错误
# SyntaxError: Missing parentheses in
call to 'print'. Did you mean print(abc)?
# print abc


print('='*20)

# 如果'xxx.txt'文件不存在，只读方法打开
# 解释器检查到异常错误，默认动作程序终止运行（程
序崩溃）
# FileNotFoundError: [Errno 2] No such
file or directory: 'xxx.txt'
f = open("xxx.txt", "r")

f.close()
```

- 常见的异常

```
# FileNotFoundError: [Errno 2] No such
file or directory: 'xxx.txt'
# open("xxx.txt", "r")

# f = open("yyy.txt", "w")
# io.UnsupportedOperation: not readable
```

```
# f.read()

# age = input("输入年龄: ")
# # TypeError: '>' not supported between
instances of 'str' and 'int'
# if age > 18:
#     print("成年了")

# ZeroDivisionError: division by zero
# 10/0
```

2. 异常的基本处理

2.1 处理异常的目的

- 只要解释器检查到异常错误，默认执行的动作是终止程序，为了**防止程序退出**，保证程序正常执行，需人为处理异常

2.2 捕获任意类型的异常

```
"""
try:
    可能发生异常的代码
except:
    # 处理异常的代码
```

```
    1. 如果try里面发生异常
    2. 自动跳转到except里面
"""

try:
    print("="*20)
    f = open("xxx.txt", "r")
    print("="*20)
except:
    print("捕获到异常")
```

2.3 捕获指定异常类型

```
"""
try:
    可能发生异常的代码
except 异常类型:
    # 处理异常的代码
    1. 如果try里面发生异常
    2. 自动跳转到except里面
"""

try:
    f = open("xxx.txt", "r")
```

```
    # print(10/0)

except FileNotFoundError:    # 只是捕获FileNotFoundError异常
    print("捕获到文件找不到异常")
```

2.4 except捕获多个异常

```
"""
try:
    可能发生异常的代码
except (异常类型1, 异常类型2):
    # 处理异常的代码
    1. 如果try里面发生异常
    2. 自动跳转到except里面
"""

try:
    f = open("xxx.txt", "r")

    print(10/0)
except (FileNotFoundError, ZeroDivisionError):  # 捕获多个异常中的一个异常
    print("捕获到文件找不到异常或除数为0异常")
```

2.5 获取异常的信息描述

```
"""
try:
    可能发生异常的代码
except 异常类型 as 异常对象名:
    print(异常对象名) 即可获取异常的信息描述
"""

try:
    f = open("xxx.txt", "r")

except FileNotFoundError as e:
    print("捕获到异常: ", e)
```

2.6 捕获任意类型的异常

```
"""
try:
    可能发生异常的代码
except Exception as 异常对象名:
```

```
    Exception 为异常类的父类
"""

try:
    f = open("xxx.txt", "r")

    # print(10/0)

    # print("18" >= 18)

    # print(age)

except Exception as e:  # 捕获任意类型异常：获取异常信息
    print(type(e), "捕获到异常：", e)
```

2.7 异常中else

```
"""
try:
    可能发生异常的代码
except:
    处理异常的代码
else:
```

```
    没有发生异常，except不满足执行else
"""

try:
    # f = open("xxx.txt", "r")
    pass

except Exception as e:
    print("捕获异常：", type(e), e)

else:
    print("代码执行成功,没有发生异常")
```

2.8 try...finally...

```
"""
try:
    可能发生异常的代码
except:
    处理异常的代码
else:
    没有发生异常，except不满足执行else
finally:
    不管有没有异常，最终都要执行
"""
```

```
"""
try:
    可能发生异常的代码
except:
    处理异常的代码
else:
    没有发生异常，except不满足执行else
finally:
    不管有没有异常，最终都要执行
"""

try:
    print("="*20)
    # num = 666
    print(num)
    print("="*20)

except Exception as e:
    print("捕获异常：", type(e), e)

else:
    print("没有捕获异常,很开心")

finally:
    print("无论有没有发生异常,都会执行这行")
```

- 应用场景

```
# finally：有没有异常，最终都要执行
# 对于文件操作，在文件打开的前提下，后面文件的其它操作，不管有没有发生异常，最终都应该关闭文件


f = open("yyy.txt", "w")

try:
    # data = f.read()
    # print(data)

    f.write("hello")

except Exception as e:
    print("捕获到异常：", type(e), e)

else:
    print("没有发生异常,很开心")

finally:
    print("无论是否发生异常,都需要关闭文件")
    f.close()
```

3. 异常的传递

3.1 try嵌套

- 如果异常在内部产生，如果内部不捕获处理，这个异常会向外部传递

```
# try嵌套时，如果内层try没有捕获该异常，就会向外层try进行传递

try:
    f = open("yyy.txt", "w")

    try:
        # 文件是以"w"打开，不能读操作，发生异常
        # 内层的try没有捕获异常，异常会向外层try传递
        ret = f.read()
        print(ret)
    # except Exception as e:
    #     print("内层try捕获到异常：", e)
    finally:
        print("无论是否发生异常，都需要关闭文件")
        f.close()
```

```
except Exception as e:
    print("外层try捕获到异常： ", e)
```

3.2 函数嵌套

- 如果内层函数没有捕获处理该异常，就会向外层函数进行传递

```
# 函数嵌套时，如果内层函数没有捕获处理该异常，就会向外层函数进行传递

# 定义1个函数，函数内部发生了异常 test01()，没有捕获处理
def test01():
    print("开始执行test01================")
    print(num)
    print("结束执行test01================")


# 定义另外一个函数 test02，在函数内部调用test01
def test02():
    print("开始执行test02================")
    test01()
    print("结束执行test02================")
```

```
# 定义一个test03函数，函数内部调用test01，但是对test01做异常处理
def test03():
    print("开始执行test03================")
    try:
        test01()
    except Exception as e:
        print("捕获到异常：", e)
    print("结束执行test03================")


# 调用test02
# NameError: name 'num' is not defined
# test02()

test03()
```

4.【重点】抛出自定义异常

- 抛出自定义的异常语法格式：

```
# 1. 自定义异常类
class 自定义异常类名字(Exception):
    1.1 重新写__init__(self, 形参1,  形参2, ……)
        # 建议调用父类的init，先做父类的初始化工作
        super().__init__()
        咱们自己写的代码
    1.2 重新写__str__(), 返回提示信息

# 2. 抛出异常类
raise 自定义异常类名字(实参1,  实参2, ……)
```

- 案例

```
"""
# 1. 自定义异常类
class 自定义异常类名字(Exception):
    1.1 重新写__init__(self, 形参1,  形参2, ……)
        # 建议调用父类的init，先做父类的初始化工作
        super().__init__()
        咱们自己写的代码

    1.2 重新写__str__(), 返回提示信息
```

```
# 2. 抛出异常类
raise 自定义异常类名字(实参1,  实参2, ……)

# 3. 需求
# 1. 自定义异常类，电话号码长度异常类
    # 1.1 __init__，添加2个属性，用户电话的长度，要求的长度
    # 1.2 __str__ 返回提示描述意思，如：用户电话长度为：xx位，这边要求长度为：11位

# 2. 只要用户输入的手机号码不为11位，抛出自定义异常类
"""

# class 自定义异常类名字(Exception):
#     1.1 重新写__init__(self, 形参1,  形参2, ……)
#         # 建议调用父类的init，先做父类的初始化工作
#         super().__init__()
#         咱们自己写的代码
#
#     1.2 重新写__str__(), 返回提示信息
class NumberError(Exception):
```

```
    def __init__(self, _user_len,
_match_len):
        super().__init__()
        self.user_len = _user_len
        self.match_len = _match_len

    def __str__(self):
        return f"用户输入的手机号码长度:
{self.user_len}, 但是要求是的长度:
{self.match_len}"


try:
    iphone_num = input("请输入手机号码: ")
    if len(iphone_num) != 11:
        raise NumberError(len(iphone_num),
11)
except Exception as e:
    print("捕获到自定义异常: ", type(e), e)
```

二、模块

1. 模块的使用

1.1 模块介绍

- 模块其实就是一个`.py`结尾的文件, 但是文件名遵循标识符命名(字母，数字，下划线组成, 不能以数字开头，不能和关键字同名，建议不和类型同名)
- 作用:
  - 易于代码维护
  - 代码复用
  - 避免名字冲突

1.2 import导入模块

```
"""
模块：就是一个py文件，模块里面有：函数的定义，类的定义，全局变量
导入模块：本质上就是导入一个py文件
"""


"""
导入格式：        import 模块名
使用格式：        模块名.函数    模块名.类名    模块名.变量名
"""
```

```
# 导入模块
import random

# 模块名.函数
num = random.randint(1, 3)
print(num)

# 模块名.类名
# 创建对象
ran = random.Random()
print(type(ran))

# 模块名.变量名
print(random.TWOPI)
```

1.3 from...import导入模块中需要的内容

```
"""
导入格式:        from 模块名 import 需使用的函数、类、变量
使用格式:        函数、类、变量    无需通过模块名引用

# 缺点：可以会出现名字冲突
```

```
"""

from random import randint, Random, TWOPI

# 名字冲突
# TWOPI = 6.28

ret = randint(1, 3)
print(ret)

ran = Random()
print(type(ran))

print(TWOPI)

"""
导入格式:       from 模块名 import 需使用的函数、类、变量
使用格式:       函数、类、变量    无需通过模块名引用
"""

from random import randint, TWOPI

# def randint():
#     print("测试randint")
```

```
ret = randint(1, 3)
print(ret)

print(TWOPI)
```

1.4 from...import导入模块中所有的内容

```
"""
导入格式:       from 模块名 import *
使用格式:       函数、类、变量    无需通过模块名引用
"""

from random import *

ret = randint(1, 3)
print(ret)

# print(TWOPI)

ran = Random()
print(type(ran))
```

1.4 import和from...import...导入模块区别

- import导入模块，把整个模块都加载进来: 使用`模块名.工具`引用模块中的工具.
- from...import可以只导入模块中需要使用的内容, 也可以导入'所有': 引用时不需要模块名

1.5 import...as...能够给导入的模块取别名

```
"""
模块起别名
导入格式：import 模块 as 模块别名
使用格式：模块别名.工具(工具指函数、类、变量)

模块工具起别名
导入格式：from 模块 import 工具 as 工具别名
使用格式：工具别名          无需通过模块名引用
"""
```

```
# 1. 把复杂名字改简单些
# 2. 把已经同名的名字改一个不同名的名字
"""
模块起别名
导入格式：import 模块 as 模块别名
使用格式：模块别名.工具(工具指函数、类、变量)
```

```
模块工具起别名
导入格式：from 模块 import 工具 as 工具别名
使用格式：工具别名  无需通过模块名引用
"""

# 模块起别名
# 导入格式：import 模块 as 模块别名
# 使用格式：模块别名.工具(工具指函数、类、变量)

import random as r

ret = r.randint(1, 3)
print(ret)

print(r.TWOPI)

ran = r.Random()
print(type(ran))

# 模块工具起别名
# 导入格式：from 模块 import 工具 as 工具别名
# 使用格式：工具别名  无需通过模块名引用

from random import randint as ri
```

```
def randint():
    print("测试randint重名")


# ri = 100

ret = ri(1, 3)
print(ret)
```

1.6 模块搜索路径保存在sys.path变量

```
import sys
print(sys.path)
```

```
# 1. 在当前路径找
# 2. 如果当前路径没有，找系统路径

# 如果当前目录下有一个random.py文件,这里import random会直接导入当前目录下的.
# 而不会导入系统的random
# import random
#
# print(random.TWOPI)
```

```
# ret = random.randint(1, 3)
# print(ret)


# 模块搜索路径存储在system模块的sys.path变量中
import sys

#
['C:\\Users\\35477\\Desktop\\python40\\day0
9',
'C:\\Users\\35477\\Desktop\\python40\\day09
', 'C:\\Python\\Python38\\python38.zip',
'C:\\Python\\Python38\\DLLs',
'C:\\Python\\Python38\\lib',
'C:\\Python\\Python38',
'C:\\Python\\Python38\\lib\\site-packages']
print(sys.path)
```

2. 模块制作

2.1 制作自定义模块

- 把写好的代码放在一个py文件中，这个py文件就是模块文件，后续方便别人导入使用

- 测试代码:

```
# 导入模块
import module

# 调用模块中的函数 (模块名.函数)
ret = module.my_add(10, 20)
print(ret)

ret = module.my_sub(100, 50)
print(ret)
```

- 自定义模块:module.py

```
def my_add(a, b):
    """返回2个数相加结果"""
    return a + b


def my_sub(a, b):
    """返回2个数相减结果"""
    return a - b


ret = my_add(2, 2)
print('模块中测试代码: my_add(2, 2) = ', ret)
```

```
ret = my_sub(10, 2)
print('模块中测试代码：my_sub(10, 2) = ',
ret)
```

2.2 模块中`__name__`的作用

- 直接运行此文件，`__name__`的结果为`'__main__'`
- 此文件被当做模块文件导入时，`__name__`的结果不为`__main__`
- 如果不想导入模块时把模块的测试代码也运行，把模块的测试代码放在`if __name__ == '__main__':`条件语句里面
- module.py代码:

```
def my_add(a, b):
    """返回2个数相加结果"""
    return a+b


def my_sub(a, b):
    """返回2个数相减结果"""
    return a-b
```

```
# 把不想导包被执行的代码放入 if __name__ ==
'__main__': 语句里
# if __name__ == '__main__':
if __name__ == '__main__':
    # 下面是模块测试自己模块功能的代码
    ret = my_add(2, 2)
    print('模块中测试代码: my_add(2, 2) = ',
ret)

    ret = my_sub(10, 2)
    print('模块中测试代码: my_sub(10, 2) = ',
ret)
```

2.3 使用from...import*时`__all__`在模块中的作用

- 模块中`__all__`变量，**只对**`from xxx import *`这种导入方式有效
- 模块中`__all__`变量包含的元素，才能会被`from xxx import *`导入

```
"""
```

```
1. 模块中__all__变量，只对from xxx import *这种导入方式有效
2. __all__格式:
    __all__ = ['变量名', '类名', '函数名'……]
3. 模块中__all__变量包含的元素，才会被from xxx import *导入
"""

from module5 import *

ret = my_add(2, 2)
print(ret)

print(num)



# ret = my_sub(10, 2)
# print(ret)

# 注意: __all__ 变量只是针对from 模块 import *这种导入方式使用
# 其他方式导入,不受影响. 见下面实例:

# import module5
# ret = module5.my_sub(10, 2)
# print(ret)
```

```
# from module5 import my_sub
# ret = my_sub(10, 2)
# print(ret)
```