学习目标

1. 掌握 try except 的基本语法;
2. 掌握抛出异常 raise 的语法;
3. 掌握 import 导入模块语法;
4. 掌握 from import 导入指定内容的语法;
5. 掌握包的概念;
6. 掌握 import 导入包的语法;

目录

第 1 章 其他------异常

一、 异常的概念

程序在运行时，如果遇到到一个错误，会停止程序的执行，并且提示一些错误信息，这就是异常。

程序停止执行并且提示错误信息这个动作，称之为：抛出(raise)异常。

程序开发时，很难将所有的特殊情况都处理的面面俱到，通过异常捕获 可以针对突发事件做集中的处理，从而保证程序的稳定性和健壮性 。

二、捕获异常

1. 简单的捕获异常语法

在程序开发中，如果对某些代码的执行不能确定是否正确，可以增加 try(尝试) 来捕获异常 。

- 语法格式：

```
try:
    可能出现异常的代码
except:
    出现异常的处理代码
```

- 演练 —— 要求用户输入整数

```
try:
    num1 = int(input("请输入数字：："))
```

```
except:
    print("请输入正确的数字")
```

2. 课堂练习---

try 语法练习：

input 函数接受用户输入，当用户输入不正确的数字后，提示用户输入不正确。

程序继续等待用户输入，用户连续三次输入不正确，程序退出。

3. 错误类型捕获

在程序执行时，可能会遇到不同类型的异常，并且需要针对不同类型的异常，做出不同的响应，这个时候，就需要捕获错误类型了。

- 语法格式：

```
try:
    可能出现异常的代码
except 错误类型 1:
    出现异常的处理代码
except (错误类型 2,错误类型 2):
    出现异常的处理代码
```

- 演练 —— 用户输入两个整数，显示两个整数相除后的结果

```
try:
    num1 = int(input("请输入数字：："))

    num2 = int(input("请输入数字：："))
    print(num1 / num2)
except ValueError:
```

```
    print("请输入正确的数字")
except ZeroDivisionError:
    print("除数不能为 0")
```

4. 课堂练习---

计算器:

定义三个变量, 分别为 num1, op1,num2，其中 num1 和 num2 是用户通过 input 函数输入的任意数字。

op1 为通过 input 函数输入的'+'、'-'、'*'、'/'四个运算符中任意一个。

根据用户输入进行 num1,num2 两个数字的计算。

要求分别能捕捉用户输入不正确数字，以及除数为 0 的错误。

5. 捕获未知错误

在开发时，要预判到所有可能出现的错误，还是有一定难度的 。

如果希望程序无论出现任何错误，都不会因为 程序抛出异常而被终止，可以再增加一个 except 。

- 语法格式

```
try:
    可能出现异常的代码
except Exception as result:
    出现未知异常的处理代码
```

- 演练 —— 两个变量，一个为整数，一个为字符串，显示两个变量相加的结果

```
try:
    a = "aaaa"
    b = 4
    print(a + b)
except Exception as result:
    print(result)
```

6. 没有异常发生才执行的代码

程序运行过程中，如果没有异常发生，需要执行一段代码。这个时候可以使用 else。

- 语法格式

```
try:
    可能出现异常的代码
except Exception as result:
    出现未知异常的处理代码
else:
    没有异常执行的代码
```

- 演练 —— 用户输入两个整数，显示两个整数相除后的结果

```
try:

    num1 = int(input("请输入数字："))

    num2 = int(input("请输入数字："))

    print(num1 / num2)
except Exception as result:

    print("未知错误 %s" % result)

else:

    print("恭喜，没有异常发生")
```

7. 无论是否有异常，都要执行的代码

程序运行过程中，一旦发生异常，代码就会跳转到 except，有时需要不论发生任何异常，都必须要执行一段代码。这个时候可以使用 finnly。

- 语法格式

```
try:
    可能出现异常的代码
except Exception as result:
    出现未知异常的处理代码
finally:
    无论是否有异常都要执行的代码
```

- 演练 —— 用户输入两个整数，显示两个整数相除后的结果

```
try:
    num1 = int(input("请输入数字："))
    num2 = int(input("请输入数字："))
    print(num1 / num2)
except Exception as result:
    print("未知错误 %s" % result)
finally:
    print("无论有无异常，都要执行")
```

8. 异常捕获完整语法

在实际开发中，为了能够处理复杂的异常情况，完整的异常语法如下：

```
try:
    可能出现异常的代码
except 错误类型 1:
    出现异常的处理代码
```

```
except (错误类型 2,错误类型 2):
    出现异常的处理代码
except Exception as result:
    出现未知异常的处理代码
else:
    没有异常执行的代码
finally:
    无论是否有异常都要执行的代码
```

- 演练 —— 用户输入两个整数，显示两个整数相除后的结果

```
try:
    num1 = int(input("请输入数字：："))

    num2 = int(input("请输入数字：："))
    print(num1 / num2)
except ValueError:
    print("请输入正确的数字")
except ZeroDivisionError:
    print("除数不能为 0")
except Exception as result:
    print("未知错误 %s" % result)
else:
    print("没有异常发生")
finally:
    print("无论有无异常，都要执行")
```

三、抛出异常

在开发中，除了代码执行出错时被动抛出异常之外， 还可以根据应用程序特有的业务需求主动抛出异常。

抛出异常使用 raise 关键字，同时 Python 中提供了一个 Exception 异常类。

- 主动抛出异常语法：

```
raise Exception(“异常描述”)
```

- 演练------提示用户输入密码，如果长度少于 8，抛出异常

```
str1 = input("请输入密码")
if len(str1) < 8:
    raise Exception("密码长度不够")
else:
    print("密码正确")
```

注意

如果通过 raise 抛出异常，但程序中没有使用 try 捕捉异常，程序会终止，所以完整代码如下：

```
try:
    str1 = input("请输入密码")
    if len(str1) < 8:
        raise Exception("密码长度不够")
    else:
        print("密码正确")
except Exception as result:
    print(result)
```

1. 课堂练习---

定义 name 存放姓名，age 存放年龄，通过 input 函数输入这两个变量的值

```
name = input("请输入姓名")
```

```
age = int(input("请输入年龄"))
```

1、当 name 中有数字字符，抛出异常。
2、当 age 小于等于 0， 抛出异常。
3、程序通过 try 语句捕捉上两种情况抛出的异常。

第 2 章 其他------模块

一、模块的概念

- 当项目代码越来越多，不可能把所有代码都放到一个 py 文件中；
- 一个项目往往由多个 py 文件组成；
- 模块是 Python 程序架构的一个核心概念 ；
- 每一个以扩展名 py 结尾的 Python 源代码文件都是一个模块；
- 模块名同样也是一个标识符，需要符合标识符的命名规则；
- 在模块中定义的全局变量 、函数、类 都是提供给外界直接使用的工具；
- 模块就好比是工具包，要想使用这个工具包中的工具，就需要先导入这个模块 。

二、模块的导入方式

1. import 导入

- 语法：

```
import 模块名
```

导入之后通过 **模块名.** 使用模块提供的工具 —— 全局变量、函数、类

- 导入模块演练

1.新建一个文件 module1.py,文件内容如下:

```
# module1.py
# my_max 函数，计算参数最大值
def my_max(*a):
    return max(a)

# my_sum 函数，参数求和
def my_sum(*a):
    sum = 0
    for n in a:
        sum += n
    return sum
```

2.新建一个文件 module2.py,文件内容如下:

```
# module2.py
# 导入 module1 模块
import module1
# 调用 module1 中的 my_max 函数
print(module1.my_max(4,6,2))
```

2. as 指定模块别名

- 语法:

```
import 模块名 as 模块别名
```

注意:

- 如果两个模块，存在同名的函数，那么后导入模块的函数，会覆盖掉先导入的函数 ；
- 开发时 import 代码应该统一写在代码的顶部，更容易及时发现冲突 ；
- 一旦发现冲突，可以使用 as 关键字 给其中一个工具起一个别名 。

- 导入模块演练

module2.py,文件内容更改如下：

```
# module2.py
# 导入 module1 模块
import module1 as m
# 通过别名 m 调用 module1 中的 my_max 函数
print(m.my_max(4,6,2))
```

3. 课堂练习---

新建一个文件 m1.py。

m1.py 中定义两个函数内容如下：

```
def m1_func():
    print("我是 m1 的 func 函数")

def m1_test():
    print("我是 m1 的 test 函数")
```

新建一个文件 m2.py，

m2.py 中定义一个函数 m2_func,函数内容如下：

```
def m2_func():
    print("我是 m2 的 func 函数")
```

新建一个文件 m.py， 在 m.py 中通过 import 导入 m1.py 和 m2.py，并且调用 m1_func、m1_test 和 m2_func 函数。

4. from...import 导入

import 模块名 是一次性把模块中所有内容全部导入。

如果希望从某一个模块中导入部分内容，就可以使用 from ... import 的方式 。

- 语法一：

```
from 模块名 import 工具名
```

- 语法二：

```
from 模块名 import *
```

语法二的作用是从指定模块导入所有工具。

导入之后不需要通过 **模块名.** 可以直接使用模块提供的工具 —— 全局变量、函数、类 。

- 导入模块演练

module2.py,文件内容更改如下：

```
# module2.py
# 导入 module1 模块
from module1 import my_max, my_sum
# 调用 module1 中的 my_max 函数
print(my_max(4,6,2))
# 调用 module1 中的 my_sum 函数
print(my_sum(4,6,2))
```

或者使用 import *

```
# module2.py
# 导入 module1 模块
from module1 import *
# 调用 module1 中的 my_max 函数
print(my_max(4,6,2))
# 调用 module1 中的 my_sum 函数
print(my_sum(4,6,2))
```

5. 课堂练习---

m1.py 中定义两个函数内容如下：

```
def m1_func():
    print("我是 m1 的 func 函数")

def m1_test():
    print("我是 m1 的 test 函数")
```

只导入 m1.py 的 m1_test 函数，同时调用 m1_test 函数

6. __name__属性

如果是被其他文件导入的模块， __name__ 就是模块名 ；

如果是当前执行的程序 __name__ 是“__main__”；

在很多 Python 文件中都会看到以下格式的代码：

```
# 导入模块
# 定义全局变量
# 定义类
# 定义函数
```

```python
# 在代码的最下方
def main():
    # ...
    pass


# 根据 __name__ 判断是否执行下方代码
if __name__ == "__main__":
    main()
```

第 3 章 其他------包

一、 包(package)的概念

- 包是一个包含多个模块的特殊目录;
- 目录下有一个特殊的文件 __init__.py ；
- 使用包的作用是可以通过 **import 包名** 的方式, 一次性把一个目录下所有的模块都导入。

二、案例演示

1. 在项目中新建一个 目录 my_pack ；
2. 在 my_pack目录下，新建两个文件 a1.py, a2.py ；
3. 在 a1.py文件中定义一个 my_func1 函数 ；

```python
# a1.py
def my_func1():
```

```
    print("my_func1")
```

4. 在 a1.py文件中定义一个 my_func2 函数

```
# a2.py
def my_func2():
    print("my_func1")
```

5. 在 my_pack目录下新建文件 __init__.py，文件内容如下

```
# __init__.py
# 从当前目录导入 a1 模块
from . import a1
# 从当前目录导入 a2 模块
from . import a2
```

1. import 包名 导入

```
# 导入 my_pack 包
import my_pack
# 调用 my_pack 包里 a1 模块的 my_func1 函数
my_pack.a1.my_func1()
# 调用 my_pack 包里 a2 模块的 my_func2 函数
my_pack.a2.my_func2()
```

2. from 包名.模块名 import 工具名 导入

```
# 从包 my_pack 的 a1 中导入 my_func1 函数
from my_pack.a1 import my_func1
# 从包 my_pack 的 a2 中导入 my_func2 函数
from my_pack.a2 import my_func2
# 调用时直接写函数名即可
my_func1()
my_func2()
```

3. 课堂练习---

新建一个目录 my_pack1,在 my_pack1 目录下新建文件 m1.py。

m1.py 中定义两个函数内容如下：

```
def m1_func():
    print("我是 m1 的 func 函数")

def m1_test():
    print("我是 m1 的 test 函数")
```

在 my_pack1 目录下新建一个文件 m2.py;

m2.py 中定义一个函数 m2_func,函数内容如下：

```
def m2_func():
    print("我是 m2 的 func 函数")
```

在 my_pack1 目录下建立__init__.py 文件。

新建一个 py 文件， 通过 import my_pack1 导入 m1.py 和 m2.py，并且调用 m1_func、m1_test 和 m2_func 函数。