"""
异常传递
学习目标：
    1、了解异常具有传递性的
当函数func01中发生异常，并且没有捕获处理这个异常时候，异常会传递到函数func02，当func02也没有捕获处理
main()函数会捕获这个异常，这就是异常的传递性。
注意：当所有函数都没有捕获异常的时候，程序会报错。
"""

```
def func01():
    print("这是func01开始")
    num = 1 / 0
    print("这是func01结束")

def func02():
    print("这是func02开始")
    func01()
    print("这是func02结束")



def main():
    # try:
    #     func02()
    # except Exception as e:
    #     print("捕获到异常：", e)
    func02()

main()
```

"""
利用异常具有传递性的特点，当我们想要保证程序不会因为异常而崩溃的时候，就可以在main函数中设置捕获异常
由于无论在整个程序哪里发生异常，最终都会传递到mian函数中，这样就可以确保所有的异常都会被捕获。
"""
"""
python的模块
1、模块的导入
学习目标：
    1、了解什么是模块
    2、掌握导入python内置的模块
模块：模块（Module）是一个python文件，以.py结尾，模块能定义函数、类和变量，模块里也能包含可执行的代码
作用：python中有很多各种不同的模块，每一个模块都可以帮助我们快速实现一些功能，比如实现和时间相关的功能
我们就可以认为一个模块就是一个工具包，每一个工具包中都含有不同的工具共我们使用进而实现各种不同的功能。

2、模块的导入方式

模块在使用前需要导入，语法格式：

[from 模块名] import [模块 | 类 | 变量 | 函数 | *] [as 别名]

常用的组合形式：

(1)import 模块名

基本语法：

```
import 模块名
import 模块名1,模块名2
模块名.功能名()
```

(2)from 模块名 import 类、变量、方法等

(3)from 模块名 import *

基本语法：

```
from 模块名 import *
功能名()
```

(4)import 模块名 as 别名

基本语法：

```
import 模块名 as 别名
别名.功能名()
```

(5)from 模块名 import 功能名 as 别名

基本语法：

```
from 模块名 import 功能名 as 别名
别名()
```

3、自定义模块

学习目标：

1、了解如何自定义模块并使用

2、了解__main__变量的作用

测试模块

总结：

如何自定义模块并导入？

在python代码文件中正常写代码即可。通过import、from关键字和导入python内置模块一样导入即可。

__main__变量的功能是？

只有当程序是直接执行的时候才会进入 if 内部，如果是被导入的，则无法进入if

注意事项：

1、不同模块，同名功能，如果都被导入，那么后导入的会覆盖先导入的。

2、__all__变量可以控制import * 的时候哪些功能可以被导入。

```
"""
# (1)import 模块名
# import time
# print("开始")
```

```
# # 让程序睡眠5秒钟（阻塞）
# time.sleep(5)
# print("结束")


# (5)from 模块名 import 功能名 as 别名
# from time import sleep as sl
# print("开始")
# sl(5)
# print("结束")



#(4)import 模块名 as 别名
import time as tt
tt.sleep(5)



# (3)from 模块名 import *
from time import *
print("开始")
sleep(5)
print(process_time())
print("结束")

# def test(a,b):
#     print(a + b)
#
#
#
# # test(1,1)
# """
# 问题：
#     此时，无论是当前文件，还是其他已经导入该模块的文件，在运行的时候哦，都会执行test函数的调用。
# """
#
# # 解决方案
# # 只在当前文件中调用该函数，其他导入的文件不符合该条件，则不执行test函数的调用
# if __name__ == '__main__':
#     test(1,1)

__all__ = ['test_A']
```

```python
def test_A(a,b):
    print("test_A")
    print(a + b)


def test_B(a,b,c):
    print("test_B")
    print(a - b)
# def test(a,b):
#     print(a + b)
#
#
#
# # test(1,1)
# """
# 问题：
#     此时，无论是当前文件，还是其他已经导入该模块的文件，在运行的时候哦，都会执行test函数的调用。
# """
#
# # 解决方案
# # 只在当前文件中调用该函数，其他导入的文件不符合该条件，则不执行test函数的调用
# if __name__ == '__main__':
#     test(1,1)

__all__ = ['test_A']

def test_A(a,b):
    print("test_A")
    print(a + b)


def test_B(a,b,c):
    print("test_B")
    print(a - b)
# import my_module1
# my_module1.test(10,20)
# """
# 注意：每一个python文件都可以作为一个模块，模块的名字就是文件名字，也就是说自定义模块名必须要复合标
# """

# from my_module1 import test
# from my_module2 import test
```

```
# test(1,1)
# """
# 注意：当导入多个模块的时候，且模块内有同名功能，当调用这个同名功能的时候，调用到的是后面导入的模块的
#
# """

"""

__all__
如果一个模块文件中有all变量，当使用from xxxxx import *导入时，只能导入这个列表中的元素

"""



from my_module1 import *

test_A(1,2,2)
# import my_package.my_module1
# import my_package.my_module2
from my_package import *

# 包中的my_module1的info_print1()
# my_module1.info_print1()

"""
my_module1报红，证明不可用。
注意：
__all__变量针对的时from xxxx import * 这种方式,对import xxxxx 方式无效
"""
# 包中的my_module2的info_print2()
my_module2.info_print2()
```