"""
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()

"""
面向对象
1、初始对象
学习目标：
    1、理解使用对象完成数据组织的思路

总结：
    1、生活中或是程序中，我们都可以使用设计表格、生产表格、填写表格的形式组织数据。
```

2、进行对比，在程序中：

(1)设计表格 ------>设计类（class）

(2)打印表格-------->创建对象

(3)填写表格-------->对象属性赋值

成员方法

学习目标：

1、掌握类的定义和使用语法

2、掌握成员方法的使用

3、掌握self关键字的作用

1、类的定义和使用

类的定义语法格式：

class 类名称:

类的属性

类的行为

class：是关键字，表示要定义类

类的属性：即定义在类中的变量（成员变量）

类的行为：即定义在类中的函数（成员方法）

创建类对象的语法格式：

对象 = 类名称()

成员变量和成员方法

在类中定义的属性(变量),称之为成员变量。

在类中定义的行为(函数),称之为成员方法。

成员方法的定义语法：

在类中定义成员方法和定义函数基本一致，但仍有细微区别：

def 方法名(self,形参1,形参2,形参3,....形参N):

方法体

（注意：self关键字在成员方法定义的时候，必须填写。）

self关键字：

1、它用来表示类对象自身的意思

2、当我们使用类对象调用方法时，self会自动被python传入

3、在方法内部，想要访问类的成员变量，必须使用self

注意事项：

self关键字，尽管在参数类表中，但是传参的时候可以忽略它。

"""

```
# 1、组织数据
# (1)在程序中设计一个表格，称为设计类（class）
```

```python
class Student:
    # 类的属性
    name = None # 学生的名称
    age = None # 学生的年龄

    # 类的行为
    def say_hi(self,msg):
        print(f"hello，大家好好,{msg}")


# (2)在程序中打印生成表格，我们称之为：创建对象
stu1 = Student()

# (3)在程序中填写表格，我们称之为对象属性赋值
stu1.name = "张三"
stu1.age = 18
stu1.say_hi("很高兴认识大家")


"""
类和对象
学习目标：
    1、掌握使用类描述现实世界事务的思想
    2、掌握类和对象的关系
    3、理解什么是面向对象

类和对象
基于类创建对象的语法：  对象名 = 类名称()
为什么要创建对象？
类只是一种程序内的设计（设计图纸），需要基于设计（设计图纸）生成实体（对象），才能正常工作。
这种套路就称之为：面向对象编程。

使用类和对象描述现实事务

在现实中：
生成一个闹钟
设计闹钟图纸-----> 生产实体

在程序中：
类（设计闹钟图纸）：
    属性（材料、价格、形状）
    行为（闹铃）
```

基于类创建对象

```
"""

# 定义一个闹钟类（闹钟设计图纸）
class Clock:
    # 闹钟的属性
    material = None # 材料
    price = None  # 价格
    shape = None # 形状
    # 闹钟的行为
    def ring(self):
        import winsound
        winsound.Beep(1000,3000)
        print("ring ring ring")

# 创建对象
# 基于类创建对象
clock1 = Clock()
# 使用对象给闹钟属性赋值
clock1.material = "铁"
clock1.price = 19.99
clock1.shape = "圆形"
print(f"闹钟的材料{clock1.material},价格是：{clock1.price},形状是：{clock1.shape}")
# 调用闹钟的行为
clock1.ring()


"""
```

构造方法

学习目标：

1、掌握使用构造方法向成员变量赋值

python类可以使用__init__()方法，称之为构造方法

可以实现：

1、在创建类对象（构造类）的时候，会自动执行。

2、在创建类对象（构造类）的时候，将传入的参数自动传递给__init__()方法使用。

总结：

1、构造方法的名称

__init__

2、构造方法的作用

```
        (1)构建类对象的时候会自动运行
        (2)构建类对象的传参会传递给构造方法，借此特性可以给成员变量赋值
"""

class Student:
    # name = None
    # age = None      可以省略
    # id = None
    def __init__(self,name,age,id):
        self.name = name
        self.age = age
        self.id = id
        print("构造方法执行")


stu = Student("张三",18,"2500001")

"""
构造方法的注意事项：
    1、构造方法的名称：__init__(init前后有两个下划线)
    2、构造方法也是成员放啊，不要忘记在参数列表中添加self


"""
"""
练习案例：学生信息的录入
开班了，有一批学生信息需要录入系统，请设计一个类，记录学生的
姓名、年龄、地址、电话等信息
请实现：
    通过for循环配合input()输入语句，并使用构造方法，完成学生信息的键盘录入功能
    （输入完成后，使用print语句，完成信息的输出）
    当前录入第1位学生信息，总共需要录入10为学生信息。
    请输入学生的姓名：张三
    请输入学生的年龄：18
    请输入学生的地址：合肥
    学生1信息录入完成，信息为：【学生姓名：张三，年龄：18，地址：合肥】
    当录入第2为学生信息，总共需要录入10为学生信息。

"""

class Student:
    #使用构造方法时，成员变量前期还是写上
    name = None
```

```
    age = None
    address = None
    # 使用构造方法
    def __init__(self,name,age,address):
        self.name = name
        self.age = age
        self.address = address
        print(f"学生姓名：{self.name}，年龄：{self.age}，地址：{self.address}")
    # 使用魔术方法
    def __str__(self):
        return f"学生姓名：{self.name}，年龄：{self.age}，地址：{self.address}"
        # print(f"学生姓名：{self.name}，年龄：{self.age}，地址：{self.address}")

def main():
    student = []
    total_student = 2
    # 使用for循环遍历
    for i in range(total_student):
        print(f"当前录入第{i + 1}位学生信息，总共需要录入{total_student}为学生信息。")
        name = input("请输入学生的姓名：")
        age = input("请输入学生的年龄：")
        address = input("请输入学生的地址：")
        # 基于类创建对象
        stu = Student(name,age,address)
        print(stu)
        # # 向列表中添加元素
        student.append(stu)
        print(f"学生{i + 1}信息录入完成，信息为：【{stu}】")
    print("\n所有学生信息已录入完成，信息如下：")
    for i in student:
        print(i)

if __name__ == '__main__':
    main()

"""
其他内置方法
学习目标：
    1、掌握几种常用的类的内置方法
魔术方法
    __init__()构造方法，也称为魔术方法，是python类内置的方法之一。
    魔术房方法：内置的类方法，各自有特殊的功能，这些内置方法称之为魔术方法。
常见的魔术方法：
```

```
    1、__init__()   构造方法
    2、__str__() 字符串方法
    3、__lt__ 小于、大于符号比较
    4、__le__ 小于等于、大于等于符号比较
    5、__eq__  == 号比较       中  == 中

"""

class Student:
    # name = None
    # age = None
    def __init__(self,name,age):
        self.name = name
        self.age = age

    # __str__() 字符串方法
    # def __str__(self):
    #     return f"Student对象类，name:{self.name},age:{self.age}"

    # __lt__ 小于、大于符号比较
    def __lt__(self,other):
        return self.age < other.age

    # __le__ 小于等于、大于等于符号比较
    def __le__(self,other):
        return self.age <= other.age

    # __eq__  == 号比较
    def __eq__(self,other):
        return self.age == other.age



stu1 = Student("张三",18)
stu2 = Student("李四",20)
print(stu1)  # <__main__.Student object at 0x000002556F29E560>
print(stu2)  # <__main__.Student object at 0x000001771607FDC0>
# print(str(stu2))
print(stu1 >= stu2)
print(stu1 <= stu2)
print(stu1 == stu2)
```