"""
封装
学习目标：
1、理解封装的概念
2、掌握私有成员的使用

面向对象编程的思想简单理解：基于模板（类）取创建实体（对象），使用对象完成功能开发。
面向对象的三大特性：
1、封装
2、继承
3、多态
封装，表示的是将现实中的事物：
（1）属性
（2）行为
封装到类中，描述为：
（1）成员变量
（2）成员方法
从而完成程序对现实事物的描述。

苹果手机
安卓

设计一个手机：
开放给用户
属性：序列号、品牌、型号、价格
行为：上网、打游戏、通话

不开放给用户
属性：驱动信息、运行电压
行为：程度调度、内存管理

私有成员变量
私有成员方法
"""

```
# 定义一个手机类
class Phone:
    # 开放给用户 属性
```

```
    ID = None  #序列号
    brand = None # 品牌
    model = None # 型号
    price = None # 价格

    # 不开放给用户的属性
    # 运行电压
    __current_voltage = None

    # 开放给用户 行为
    def call_by_5G(self):

        print("手机正在使用5G上网")
    # 开放给用户 行为-打游戏
    def play_game(self):
        self.__current_voltage = 44
        print(self.__current_voltage)
        print("手机正在使用游戏")


    # 不开放给用户的行为
    # 内存管理
    def __memory_management(self):
        print("手机正在使用内存管理")

# 基于类创建对象
phone = Phone()
phone.play_game()
# 使用成员变量
# phone.ID = "123456789"
# phone.brand = "华为"
# phone.model = "P30"
# phone.price = 5000
# 使用私有方法
# phone.
phone.__current_voltage = 33  # 通过我们这个phone对象重新初始化一个成员变量#     有误导
# 创建的新的公有成员(仅限于该对象，并不会影响到类或者其他对象)，而不是访问类中真正的私有变量
# # phone.__current_voltage = 33  # 不报错但无效
print(phone.__current_voltage) # 报错，无法使用
phone.play_game()
"""
注意：私有变量无法赋值也无法获取值
总结：
```

```
    1、封装的概念
    将现实事物在类中描述为属性和行为，即为封装
    2、什么是私有成员，为什么需要私有成员？
    现实事物有部分属性和行为是不公开对使用者开发的。同样在类中描述属性和行为的时候也需要达到这个要求，
    3、如何定义私有成员？
    成员变量和成员方法命名 以 __(2个下划线)开头即可
    4、私有成员的访问限制？
    类对象无法访问私有成员的
    类中的其他成员可以访问私有成员
"""

# 思考？
# 私有成员定义了解，但是他有什么实际意义呢？
# 在类中提供进供内部使用的属性和方法，而不对外部开发（类对象无法使用）


"""
练习：设计带有私有成员的手机

设计一个手机类，内部包含：
私有成员变量：__is_5g_enable,bool类型，True表示开启5g，False表示关闭5g
私有成员方法：__check_5g() 会判断私有成员变量__is_5g_enable的值
    如果为True，打印开启5g
    如果为False，关闭5g
共开的成员变量：手机序列号、手机价格
共开的成员方法：call_by_5g()、play_game()
    call_by_5g调用私有成员方法__check_5g()，判断5g的状态
    打印输出结果：正在通话中
运行结果：
    5g关闭，使用4g网络
    正在通话中

"""
class Phone:
    # 成员变量
    ID= None  # 手机序列号
    price =  None  # 价格
    # 私有成员变量
    __is_5g_enable = False

    # 成员方法
    def call_by_5g(self):
        self.__check_5g()
```

```
        print("正在通话中")

    def play_game(self):
        print("正在打游戏")

    # 私有成员方法
    def __check_5g(self):
        if self.__is_5g_enable:
            print("开启5g")
        else:
            print("关闭5g,使用4g")

# 创建对象
iphone = Phone()
iphone.ID = "00000011110"
iphone.price = 7000
iphone.call_by_5g()

"""
继承
继承的基础语法：
学习目标：
    1、理解继承的概念
    2、掌握继承的使用方式
    3、掌握pass关键字
    4、扩展：掌握return yield关键字
继承分类：单继承和多继承
单继承
语法：
class 类名（父类名）：
    类的内容体
继承表示：将从父类那里继承（复制）来成员变量和成员方法（不含私有成员）

多继承
python的类之间也支持多继承，即一个类，可以继承多个父类。
语法：
class 类名(父类名1,父类名2,父类名3...父类名N):
    类的内容体
"""
from distutils.ccompiler import gen_lib_options


# **********************************单继承**************************************
```

```
#
# class Phone:
#     ID = None  # 序列号
#     producer = None  # 手机厂商
#
#     def call_by_5g(self):
#         print("手机通过5G拨打电话")
#
#
#
# class Phone2025:
#     ID = None
#     producer = None
#
#     # 透明屏
#     screen = None
#
#     def call_by_5g(self):
#         print("手机通过5G拨打电话")
#
#     # 折叠
#     def fold(self):
#         print("折叠手机")


# 父类手机
# class Phone:
#     ID = None  # 序列号
#     producer = None  # 手机厂商
#
#     def call_by_5g(self):
#         print("手机通过5G拨打电话")

#
# # 子类手机
# class Phone2025(Phone):
#     # 透明屏
#     screen = None
#     # 折叠
#     def fold(self):
#         print("折叠手机")

# **********************************多继承**************************************
```

```python
# 原始手机
class Phone:
    ID = None  # 序列号
    producer = None  # 手机厂商

    def call_by_5g(self):
        print("手机通过5G拨打电话")

# 要有红外遥控功能（开空调）
class Remote_Control:
    rc_type= "红外遥控"
    def control(self):
        print("红外遥控开启！")


# 刷卡功能NFC
class NFCReader:
    nfc_type= "2025代AI级别NFC"
    producer = "HM"

    # 读卡功能
    def read_card(self):
        print("读卡功能")
    # 写卡功能
    def write_card(self):
        print("写卡功能")




class My_Phone(Phone,Remote_Control,NFCReader):
    pass


# 创建对象
my_phone  = My_Phone()
print(my_phone.producer)  # 结果是None而非HM

"""
多继承的注意事项：
多个父类中，如果有同名成员，那么默认以继承顺序（从左到右）为优先级。
即：先继承的保留，后继承的被覆盖。
```

"""
"""
总结：
1、什么是继承？
继承就是一个类，继承另外一个类的成员变量和成员方法。
语法：
class 类（父类,[父类1,父类2,父类3,...,父类N]）:
类的内容体
2、单继承和多继承
单继承：一个类继承另一个类
多继承：一个类继承多个类，按照顺序从左向右一次继承。
多继承中，如果有同名方法或属性，先继承的优先级高于后继承

3、pass关键字
pass是占位语句，用来保证函数（方法）或类定义的完整性，表示无内容，空的意思。

"""
#扩展：掌握return yield关键字
"""
return关键字
概念：是python函数中用于返回值的关键字，将函数的运行结果返回到函数调用的位置，
使调用者能够获取函数处理后的结果。一旦执行return关键字，函数执行结束。

原理：函数执行到return时，就会停止函数体内后续代码的执行，并把return后面的值（可以是变量、表达式等）
回调用该函数的地方。若return后无表达式，默认返回None。

yield关键字
概念：yield关键字一般在函数里，包含yield的函数会变成 生成器函数 。生成器函数和普通函数不同，调用生成
不会马上执行函数体里的代码，而是返回一个生成器对象。生成器对象是一种迭代器，能通过next()函数或者for循

原理：
1、调用生成器函数：当调用生成器函数时，函数不会立即执行，而是返回一个生成器对象。
2、执行生成器对象：当生成器对象调用next()函数时，函数会开始执行，直到遇到yield语句。
此时，函数会暂停执行，并将yield后面的值返回给调用者。
3、恢复执行：再次调用next()函数时，函数会从上次暂停的地方继续执行，知道再次遇到yield语句或者函数
如果函数结束，会抛出异常。

"""

```python
# def add(a,b):
#     return a + b
#
# sum_value = add(3,5)
# print(sum_value)



# 生成一个从0到n-1的整数
def my_generator(n):
    i = 0
    while i < n:
        yield i
        i += 1
# 创建生成器对象
gen = my_generator(5)

# 使用next()函数逐个获取值
# print(next(gen))
# print(next(gen))
# print(next(gen))

# 使用for循环遍历生成器对象
for num in gen:
    print(num)


"""
生成器的优势：在于它是惰性求值的，也就是，它只会在需要的时候生成值，这样节省内存，特别是处理大规模数据
"""

"""
两者对比：
执行过程：return直接结束函数并返回值；yield暂停函数执行，保存当前状态，下一次继续从暂停处执行。
返回结果：return可返回热议类型的单个值或多个值（以元组等形式），函数执行完后返回；yield逐个返回值，可
内存使用：return若返回大量的数据，需要一次性准备好数据占用内存；yield按需生成值，处理大数据集或无限序
应用场景：return用于一般函数获取最终计算结果；yield用于需要逐步生成、迭代数据的场景，如处理大文件、生

"""

"""
```

继承

复写和使用父类成员

学习目标：

1、掌握复写父类成员的方法

2、掌握如何在子类中调用父类成员

复写：子类继承父类的成员属性和成员方法后，如果对其"不满意"，那么就可以进行复写。

即：在子类中重新定义同名的属性或方法即可。

调用父类同名成员

一旦复写父类成员，那么类对象调用成员的时候，就会调用复写后的新成员，如果需要使用被复写的父类成员，需要特殊的调用方式。

方式1：

调用父类成员：

使用成员变量：父类名.成员变量

使用成员方法：父类名.成员方法（self）

方式2：

使用super()调用父类成员：

使用成员变量：super().成员变量

使用成员方法：super().成员方法()

```
# 注意：只能在子类内部调用父类的同名成员，子类的类对象直接调用子类复写的成员。
"""
# **********子类复写父类的成员**********
# class Phone:
#     ID = None   # 序列号
#     producer = None  # 厂商
#
#
#     def call_by_5g(self):
#         print("使用5G拨打电话****")
#
# # 定义子类重写父类成员
# class MyPhone(Phone):
#     producer = "数加"  # 复写父类的成员属性
#
#     def call_by_5g(self):
#         print("开启5g，使用5G拨打电话")
#         print("拨打电话")
#         print("关闭5g，使用4G拨打电话")
#
#
# # 创建子类对象
# my_phone = MyPhone()
```

```
# my_phone.call_by_5g()
# print(my_phone.producer)

# ******************调用**********************
class Phone:
    ID = None    # 序列号
    producer = None  # 厂商
    def call_by_5g(self):
        print("使用5G拨打电话****")

# 定义子类重写父类成员
class MyPhone(Phone):
    # producer = "数加"  # 复写父类的成员属性
    #
    # def call_by_5g(self):
    #     print("开启5g！！！")
    #     print("可以上网！！！")
    #     # 使用方式1调用父类成员
    #     # print(f"父类的厂商是：{Phone.producer}")   # None
    #     # Phone.call_by_5g(self)  # TypeError: Phone.call_by_5g() missing 1 required po
    #     # print("关闭5g！！！")
    #
    #     # 使用方式2调用父类成员
    #     print(f"父类的厂商是：{super().producer}")
    #     super().call_by_5g()
    #     print("关闭5g！！！")
    pass



# 创建子类对象
my_phone = MyPhone()
my_phone.call_by_5g()
print(my_phone.producer)


"""
类型的注解
1、变量的类型注解
学习目标：
    （1）理解为什么使用类型注解
    （2）掌握变量的类型注解语法
```

类型注解：
python3.5版本的时候引入了类型注解，以方便静态类型检察工具、IDE等第三方工具。
类型注解：在代码中涉及数据交互的地方，提供数据类型的注解（显示说明）。
主要功能：
    帮助第三方IDE工具（pycharm）对代码进行类型推断，协助做代码提示
    帮助开发者自身对变量进行类型注解
支持：
    1、变量的类型注解
    2、函数(方法)形参列表和返回值的类型注解

类型注解的语法：
为变量设置类型注解语法：
变量:类型
除了使用　变量:类型　为变量做注解外，还可以在注释中为其做注解
type:类型

类型注解的限制：
类型注解主要功能在于：
    (1)帮助第三方IDE工具对代码进行类型推断，协助代码提示
    (2)帮助开发者自身对变量进行类型注解（备注）
并不会真正对类型做验证和判断，也就是说，类型注解仅仅是提示性的，不是决定性的。

函数(方法)的类型注解
学习目标：
    1、掌握函数（方法）形参进行类型注解
    2、掌握函数（方法）返回值进行类型注解

```
def func(data)
    data.xxx
func()
```

在编写函数（方法），使用形参data的时候，工具是没有任何提示的
在调用函数（方法），传入参数的时候，工具无法提示参数类型
这是因为，我们在定义函数的时候，没有给形参泪飙进行注解。

函数和方法的形参类型注解语法：
```
def 函数（方法）名(形参名:类型,形参名:类型,形参名:类型,...):
    pass
```

函数和方法的返回值类型注解语法：
```
def 函数（方法）名(形参名:类型,形参名:类型,形参名:类型,...) ->返回值类型:
    pass
```

```
"""
import random
import json

# import random
# random.randint()
#
# my_list = [1,2,3,4,5]
# my_list.append()


# 基础数据类型注解
a = 10
var_1:int = 10
print(type(var_1))  # <class 'int'>
var_2:str = "shujia"
print(type(var_2))
var_3:bool = True

# 类对象类型注解
class Student:
    pass
stu:Student = Student()


# 基础容器类型注解
# my_list:list = [1,2,3,4,5]
# my_tuple:tuple = (1,2,3,4,5)
# my_dict:dict = {"name":"shujia","age":18}
# my_set:set = {1,2,3,4,5}
# my_str:str = "lanzhishujia"
#
# print(my_list)
# print(my_tuple)
# print(my_dict)
# print(my_set)
# print(my_str)


my_list:list[int] = [1,2,3,4,"5"]
my_tuple:tuple[int,int,int] = (1,2,3)
```

```python
my_dict:dict[str,int] = {"age":18,"name":"shujia"}
my_set:set[int] = {1,2,3,4,5}
print(my_list)
print(my_tuple)
print(my_dict)
print(my_set)

"""
注意：
1、元组类型设置类型详细注解，需要将每一个元素都标记出来。
2、字典类型设置类型详细注解，需要2个类型，第一个表示key，第二表示value
"""

# 在注释中进行类型注解
var_4 = random.randint(1,3)   # type:int
var_5 = json.loads('{"name":"zhangsan"}')  # type:dict[str,str]
print(var_4)
print(var_5)



# ********************函数类型注解************************
# def func(data):
#     pass
#
# func()



# 对形参进行类型注解
def add(x:int,y:int):
    return x + y

add(1,2)

# 对返回值进行类型注解
def func(data:list) ->list:
    return data
a = func([1,2,3,4,5,6])
print(type(a))

"""
注意：注解只是建议性的，并非强制性的。！！！！
```

```
"""

"""
Union类型：
学习目标：
    1、理解Union类型
    2、掌握使用Union类型进行联合类型注解


语法：
Union[类型,.....类型]
可以定义联合类型注解。
注意：union联合类型注解，在变量注解、函数（方法）形参和返回值注解中，均可使用
"""
from typing import Union

# my_list:list = [1,2,"shujia"]
my_dict = {
    "姓名":"张三",
    "年龄":"19"
}

# 使用union类型,必须先导包，才能使用
my_list:list[Union[int,str]] = [1,2,3,"4"]
print(my_list)


my_dict:dict[str,Union[str,int]] = {
    "姓名":"张三",
    "年龄":19
}
print(my_dict)


def func(data:Union[str,int]) -> Union[str,int]:
    pass
func(1)

"""
多态
学习目标：
    1、理解多态的概念
```

2、理解抽象类（接口）的编程思想

多态：多种状态，即完成某个行为时，使用不同的对象会得到不同的状态。

同样的行为（函数），传入不同的对象，得到的不同的状态。

多态主要常用在继承关系上。

比如：

（1）函数（方法）形参声明接收父类对象

（2）实际传入父类的子类对象进行工作

即：

1、以父类作定义声明

2、以子类作实际工作

3、用以获得同一行为，不同状态。

抽象类（接口）

实现的含义：

1、父类用来确定哪些方法的

2、具体的方法实现，由子类自行决定

这种写法，就叫做抽象类（接口）

抽象类：含有抽象方法的类称之为抽象类

抽象方法：方法体是空实现的（pass）称之为抽象方法

```
"""
class Animal:
    # 动物的行为会叫
    def speak(self):
        pass


# 定义狗
class Dog(Animal):
    def speak(self):
        print("狗叫：汪汪汪")

# 定义猫
class Cat(Animal):
    def speak(self):
        print("猫叫：喵喵喵")


# 制造噪声的函数
def make_noise(animal:Animal):
    animal.speak()
```

```python
# 使用2个子类对象来调用函数
dog = Dog()
cat = Cat()


make_noise(dog)
make_noise(cat)


# 抽象类
# 空调为例（小米、美的）
# 制作空调需要遵循制冷、制热、左右摆风
class AC:
    # 制冷  (抽象方法)
    def cool(self):
        pass
    # 制热  (抽象方法)
    def heat(self):
        pass
    # 左右摆风  (抽象方法)
    def swing(self):
        pass


# 小米空调
class Mi_AC(AC):
    def cool(self):
        print("小米空调制冷")
    def heat(self):
        print("小米空调制热")
    def swing(self):
        print("小米空调左右摆风")

# 美的空调
class Mei_AC(AC):
    def cool(self):
        print("美空调制冷")
    def heat(self):
        print("美空调制热")
    def swing(self):
        print("美空调左右摆风")
```

```
def make_cool(ac:AC):
    ac.cool()


# 创建小米空调对象
mi_ac = Mi_AC()
# 创建美的空调对象
mei_ac = Mei_AC()


make_cool(mi_ac)
make_cool(mei_ac)
```