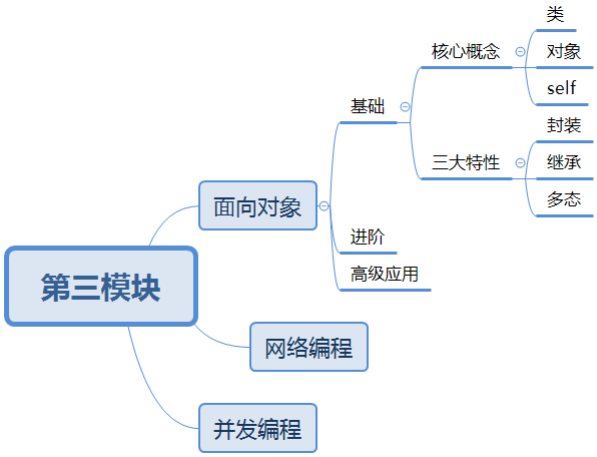
面向对象的编程

类对象中，传入父类是继承，传入子类是多态，传入参数是构造。



1.初始对象

- 目标：理解使用对象完成数据组织的思路
- 类的创建
  - 创建类
  - 创建方法
  - 创建一个实例化对象
  - 通过对象调用方法
- 如果程序中也可以和生活中一样

```
1. 可以设计表格
2. 可以将设计的表格打印出来
3. 可以将打印好的表格供人填写内容
```

1.1程序中使用对象组织数据

在程序中是可以做到和生活中那样，设计表格，生产表格，填写表格的组织形式的。

1. 在程序中设计表格，我们称之为设计类(class)，相当于设计一个表格的字段

```
class Student:
    name = None #记录学生姓名
```

注意：1.类名称首字母大写&驼峰式命名；2.python3之后默认类都继承object；3.在类中编写的函数称之为方法；4.每个方法的第一个参数是self.

2. 在程序中打印生产表格，我们称之为：创建对象 相当于创建表格中的样本数量

```
# 基于类创建对象
stu_1 = Student() #实例化一个对象，创建了一块区域。
stu_2 = Student()
```

3. 在程序中填写表格，我们称之为：对象属性赋值

```
stu_1.name = "周杰伦" #为学生1对象赋予名称属性值
stu_2.name = "林军杰" #为学生2对象赋予名称属性值
```

2.成员方法

2.1 掌握类的定义和使用语法

类可以封装属性，并基于类创建出一个个的对象来使用。

类的使用语法

```
class 类名称:    #class是关键字，表示要定义类
    类的属性     #类的属性，即定义在类中的变量（成员变量）
    类的行为     #类的行为，及定义在类中的函数（成员方法）

#创建类对象的语法 创建了一块区域
对象 = 类名称()

类class中分为两部分
    1. 属性(数据)
    2. 行为(函数)
```

```
class Student:
    def send_email(self,eamil,content):
        dat = f"给{email}发邮件，内容是{content}"
student1 = Student()
student1.send_email('ag48a48g4a','许磊')
```

类中定义的属性(变量)，我们称之为：成员变量

类中定义的行为(函数)，我们称之为：成员方法

2.2掌握成员方法的使用

- 类中定义成员方法的语法

```
def 方法名(self,形参1,.....,形参N):
    方法体
```

2.3掌握self关键字的使用

在方法定义的参数列表中，有一个:self关键字
self关键字是成员方法定义的时候，必须填写的。

1. 它可以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
4. 访问外部传入的直接用

例子

```python
#设计一个类，相当于设计一个表格的字段
class Student:
    name = None
    def say_hi(self,msg):
        print(f'大家好，我是{self.name},{msg}')
#创建一个对象，相当于创建表格中的样本数量
stu_1 = Student()
# 对象属性象进行赋值
stu_1.name = '许磊'
stu_1.say_hi("哎哟不错哟")
```

2.4 总结

1. 类是由哪两部分组成呢?

   类的属性：成员变量（数据）

   类的方法：成员方法（函数）

2. 类和成语方法的定义语句

```python
class 类名:
    name = None 成员变量
    def 成员方法(self,参数列表):
        成员方法体

创建对象
对象=类名()
```

3. self的作用

   表示类对象本身的意思

   self将类中的所有成员变量和成员方法传给self

   只有通过self,成员方法才能访问类的成员变量

   self出现在形参列表中，但是不用占用参数位置无需理会

4. self的理解

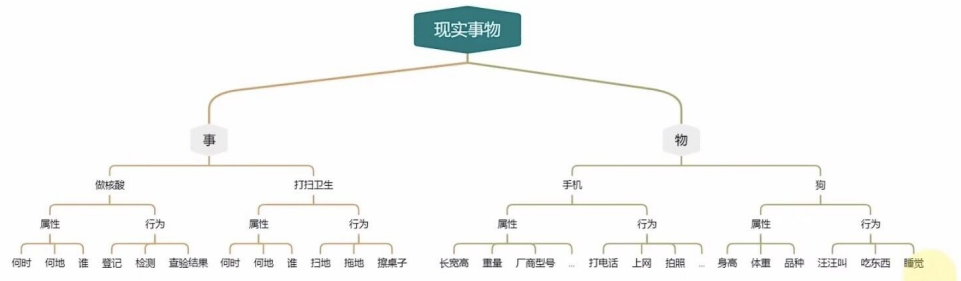
- self本质上就是一个参数。这个参数是Python内部提供，其实本质上就是调用当前方法的对象。
- 对象：基于类实例化出来“一块内存”，默认没有数据；经过类的`__init__方法`可以在内存中初始化一些数据。

3.类和对象

3.1掌握使用类描述现实世界事物的思想

现实可以分为属性和行为

现实世界的事物和类



面向对象编程的核心点：

> 设计类，基于类创建对象，有对象去做具体的工作

3.2掌握类和对象的关系

1. 类和对象的关系是什么？
    - 类是程序中的“设计图纸”
    - 对象是基于图纸生成的具体实体

3.3理解什么是面向对象

什么是面向对象编程

- 面向对象编程就是，使用对象进行编程。
- 即设计类，基于类创建对象，有对象去做具体的工作。

3.4 总结

> 1. 现实世界的事物由什么组成？
>
> - 属性
> - 行为
>
> 类也可以包含属性和行为，所以使用类描述现实世界事物是非常合适的
>
> 2. 类和对象的关系是什么？
>
> - 类是程序中的“设计图纸”
> - 对象是基于图纸生成的具体实体
>
> 3. 什么是面向对象编程
>
> - 面向对象编程就是，使用对象进行编程。
> - 即设计类，基于类创建对象，有对象去做具体的工作。

4.构造方法（使用构造方法，少一步实例化对象）

4.1掌握使用构造方法向成员变量赋值

属性（成员变量）的赋值

```
class Student:
    name = None      # 名称
    age = None       # 年龄
    tel = None       # 手机号


student1 = Student()
student1.name = "周杰轮"
student1.age = 31
student1.tel = "18012340000"

student2 = Student()
student2.name = "周杰轮"
student2.age = 31
student2.tel = "18012340000"
```

左侧代码中，为对象的属性赋值需要依次进行，略显繁琐。

有没有更加高效的方式，能够一行代码就完成呢？

思考：`Student()`

这个括号，能否像函数（方法）那样，通过传参的形式对属性赋值呢？

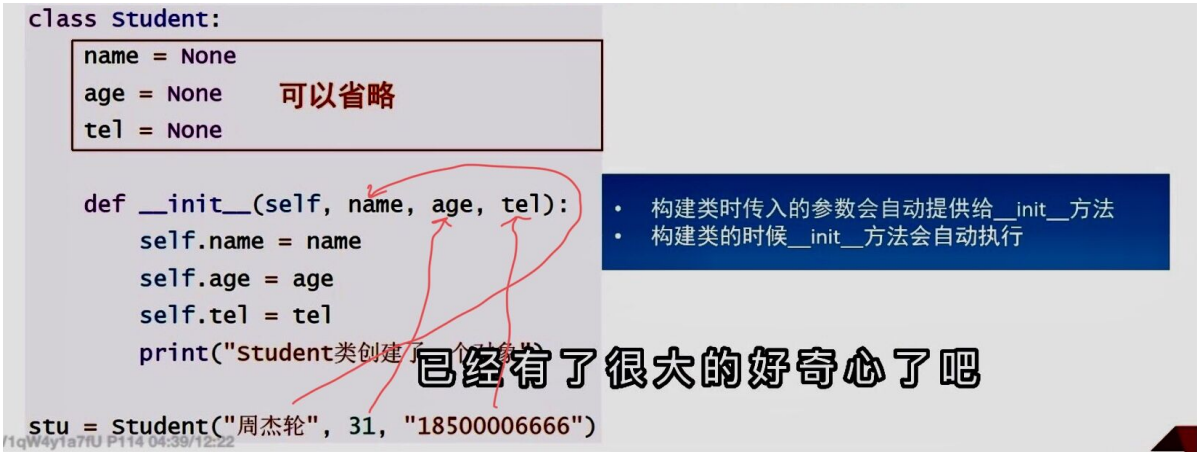
可以，需要使用构造方法：__init__()

4.2类的构造方法

python类可以使用：`__init__()`方法，称之为构造方法

可以实现：

- 在创建类对象时(构造类)的时候，会自动执行
- 在创建类对象时(构造类)的时候，将传入参数自动传递给`__init__`方法使用。



相比于减少了成员变量的赋值

例子

```
class Student:
    name = None
    age = None
    gender = None
    tel = None
    def __init__(self,name,age,gender,tel):
        self.name = name  #具有赋值和定义的功能
        self.age = age
        self.gender = gender
        self.tel = tel
        print("Student类创建了一个类对象")
```

```
12
13 stu_1 = Student("xulei",27,'男','18137405721')
14 #1.根据类型创建一个对象,内存的一块区域
15 #2.执行__init__方法，模块会创建的那块区域的内存地址当self参数传递进去，往这块区域
16 print(stu_1.name)
17 print(stu_1.age)
18 print(stu_1.gender)
19 print(stu_1.tel)
```

4.3构造方法的注意事项

- 构造方法的名称`__init__`
- 构造方法也是成员方法，不要忘记参数列表中提供的:self
- 构造方法内定义成员变量时，需要用到self关键字

```
1 def __init__(self,name,age,gender,tel):
2         self.name = name  #具有赋值和定义的功能
3         self.age = age
4         self.gender = gender
5         self.tel = tel
```

- 这是因为：变量是定义在**构造方法的内部**，如果要成为**成员变量**，需要使用self来表示。

4.4 总结

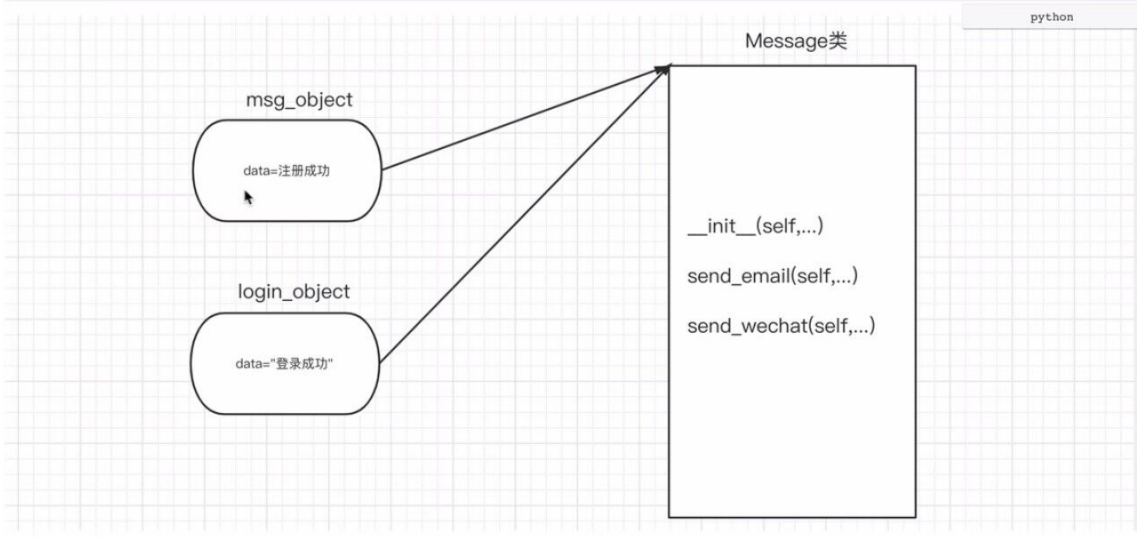
1. 构造方法的名称是：
    - `__init__`,注意init前后的2个下划线符号
2. 构造方法的作用

- 构建类对象时会自动运行
- 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

3. 注意事项

- 构造方法不要忘记self关键字
- 在方法内使用成员变量需要使用self

```python
class Message:

    def __init__(self, content):
        self.data = content

    def send_email(self, email):
        data = "给{}发邮件，内容是：{}".format(email, self.data)
        print(data)

    def send_wechat(self, vid):
        data = "给{}发微信，内容是：{}".format(vid, self.data)
        print(data)

# 对象 = 类名() # 自动执行类中的 __init__ 方法。

# 1. 根据类型创建一个对象，内存的一块 区域 。
# 2. 执行__init__方法，模块会将创建的那块区域的内存地址当self参数传递进去。     往区域中(data="注册成功")
msg_object = Message("注册成功")


msg_object.send_email("wupeiqi@live.com")
msg_object.send_wechat("武沛齐")
```

- 

4.5 面向对象的示例程序

1. 将数据封装到一个对象中，便于以后使用

```python
class UserInfo:
    def __init__(self,name,pwd):
        self.name = name
        self.password = pwd

def run():
    user_object_list = []
    # 用户注册
    while True:
        user = input("用户名")
        if user.upper() == "Q":
            break
        pwd = input("密码")

        # user_object对象中有：name/password
        user_object = UserInfo(user,pwd)
        #将每一个对象信息添加到列表中
        user_object_list.append(user_object)

    #展示用户信息
    for obj in user_object_list:
        print(obj.name,obj.password)
总结：
```

```
-将数据封装到对象，以后再去获取。
-规范数据（约束）
```

注意：用字典也可以实现封装，只不过字典在操作值是还需要自己写key，面向对象只需要`.`即可获取对象中封装的数据。

2. 将数据分装到对象中，在方法中对原始数据进行加工处理。（显示页面）

```python
#先创建用户
user_list = ["用户-{}".format(i) for i in range(1,3000)]
#分页显示，每页显示10条
while True:
    page = int(input("请输入页码:"))

    start_index = (page - 1)*10
    end_index = page*10

    page_data_list = user_list[start_index : end_index]
    for item in page_data_list:
        print(item)
```

```python
class Pagintation:
    def __init__(self,current_page,per_page_num=10):
        self.per_page_num = per_page_num
        self.current_page = current_page

        #判断页码是否是数字
        if not current_page.isdecimal():
            self.current_page = 1
            return
        current_page = int(current_page)
        if current_page < 1:
            self.current_page = 1
            return
        self.current_page = current_page
        #以上对不符合规格的数据进行处理
    def start(self):
        """
        计算起始索引
        :return:
        """
        return (self.current_page - 1) * self.per_page_num
    def end(self):
        """
        计算结束索引
        :return:
        """
        return self.current_page * self.per_page_num

user_list = [f"用户-{i}" for i in range(1,3000)]

#分页显示，每页显示10条
while True:
    page = input("请输入页码:")
```

```
    #page，当前访问页面码
    #10，每页显示10条数据
    #内部执行Pagination类的init方法。
    pg_object = Pagintation(page,10)
    page_data_list = user_list[pg_object.start():pg_object.end()]
    for item in page_data_list:
        print(item)
封装一些数据
通过方法改变一些成员变量
```

5.其他内置方法

5.1掌握几种常用的类的内置方法

`__init__`构造方法，是python类内置方法之一

这些内置的类方法，各自有各自特殊的功能，这些内置方法称之为：魔术方法

常用的魔术方法

> 1. `__init__`构造方法
> 2. `__str__`字符串方法
> 3. `__lt__`小于，大于符号比较方法
> 4. `__le__`小于等于，大于等于符号比较
> 5. `__eq__`==符号比较

5.2`__str__`字符串方法（类对象的输出）

作用：输出类对象，将控制类转换成字符串。将类对象的所有属性都显示出来

类对象转换为字符串的行为

__str__ 字符串方法

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

student = Student("周杰轮", 11)
print(student)      # 结果：<__main__.Student object at 0x000002200CFD7040>
print(str(student)) # 结果：<__main__.Student object at 0x000002200CFD7040>
```

当类对象需要被转换为字符串之时，会输出如上结果（内存地址）

内存地址没有多大作用，我们可以通过__str__方法，控制类转换为字符串的行为。

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"Student类对象，name={self.name}, age={self.age}"

student = Student("周杰轮", 11)
print(student)      # 结果：Student类对象，name=周杰轮, age=11
print(str(student)) # 结果：Student类对象，name=周杰轮, age=11
```

- 方法名：__str__
- 返回值：字符串
- 内容：自行定义

去简单的演示一下吧

5.3 __lt__小于符号比较方法（类对象比较）

__lt__ 小于符号比较方法

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

stu1 = Student("周杰轮", 11)
stu2 = Student("林军杰", 13)
print(stu1 < stu2)
```

```
Traceback (most recent call last):
  File "D:\python-learn\test.py", line 11, in <module>
    print(stu1 < stu2)
TypeError: '<' not supported between instances of 'Student' and 'Student'
```

直接对2个对象进行比较是不可以的，但是在类中实现__lt__方法，即可同时完成：小于符号 和 大于符号 2种比较

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __lt__(self, other):
        return self.age < other.age

stu1 = Student("周杰轮", 11)
stu2 = Student("林军杰", 13)
print(stu1 < stu2)  # 结果：True
print(stu1 > stu2)  # 结果：Falue
```

- 方法名：__lt__
- 传入参数：other，另一个类对象
- 返回值：True 或 False
- 内容：自行定义

好的还是很神奇的吧

5.4 __le__小于等于比较符号比较方法

```
class Student:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def __le__(self,other):
        return self.age <= other.age    #两个类对象的比较依据
stu_1 = Student('李四',29)
stu_2 = Student('杨威',30)
print(stu_2 <= stu_1)
```

5.5__eq__比较运算符实现方法

__eq__，比较运算符实现方法

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

stu1 = Student("周杰轮", 11)
stu2 = Student("林军杰", 11)
print(stu1 == stu2) # 结果：True
```

- 方法名：__eq__
- 传入参数：other，另一个类对象
- 返回值：True 或 False
- 内容：自行定义

➢ 不实现__eq__方法，对象之间可以比较，但是是比较内存地址，也即是：不同对象==比较一定是False结果。

➢ 实现了__eq__方法，就可以按照自己的想法来决定2个对象是否相等了。

5.6总结

常用的魔术方法

1. __init__构造方法
2. __str__字符串方法
3. __lt__小于，大于符号比较方法
4. __le__小于等于，大于等于符号比较
5. __eq__==符号比较

6.封装

6.1理解封装的概念

面向对象编程，是许多编程语言都支持的一种编程思想。

简单理解是：基于模板(类)去创建实体(对象，使用对象完成开发

面向对象包含三大主要特性：

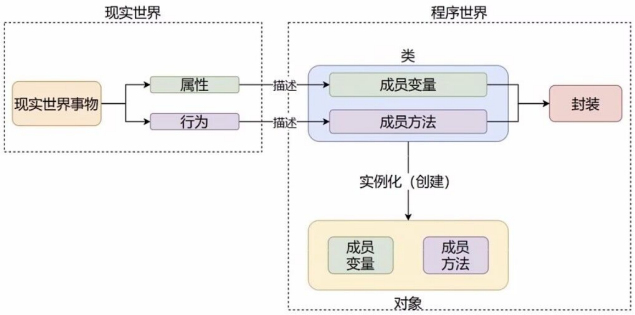
1. 封装
2. 继承
3. 多态

封装

封装

封装表示的是，将现实世界事物的：

- 属性
- 行为



封装到类中，描述为：

- 成员变量
- 成员方法

从而完成程序对现实世界事物的描述

6.2掌握私有成员的使用

不能类对象直接使用的属性和行为

私有成员

既然现实事物有不公开的属性和行为，那么作为现实事物在程序中映射的类，也应该支持。

类中提供了私有成员的形式来支持。

- 私有成员变量
- 私有成员方法

定义私有成员的方式非常简单，只需要：

- 私有成员变量：变量名以__开头（2个下划线）
- 私有成员方法：方法名以__开头（2个下划线）

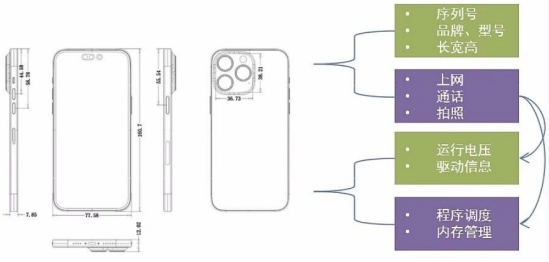
即可完成私有成员的设置

```
class Phone:
    IMEI = None                 # 序列号
    producer = None             # 厂商

    __current_voltage = None    # 当前电压   私有成员变量

    def call_by_5g(self):
        print("5g通话已开启")

    def __keep_single_core(self):
        print("让CPU以单核模式运行以节省电量")   私有成员方法
```

类对象无法直接使用私有成员，但内部的其它成员可以使用私有成员。私有的可以内部自己使用

使用私有成员

私有成员无法被类对象使用，但是可以被其它的成员使用。




```
class Phone:
    __current_voltage = 0.5         # 当前手机运行电压


    def __keep_single_core(self):
        print("让CPU以单核模式运行")


    def call_by_5g(self):
        if self.__current_voltage >= 1:
            print("5g通话已开启")
        else:
            self.__keep_single_core()
            print("电量不足，无法使用5g通话，并已设置为单核运行进行省电。")
```

6.3总结

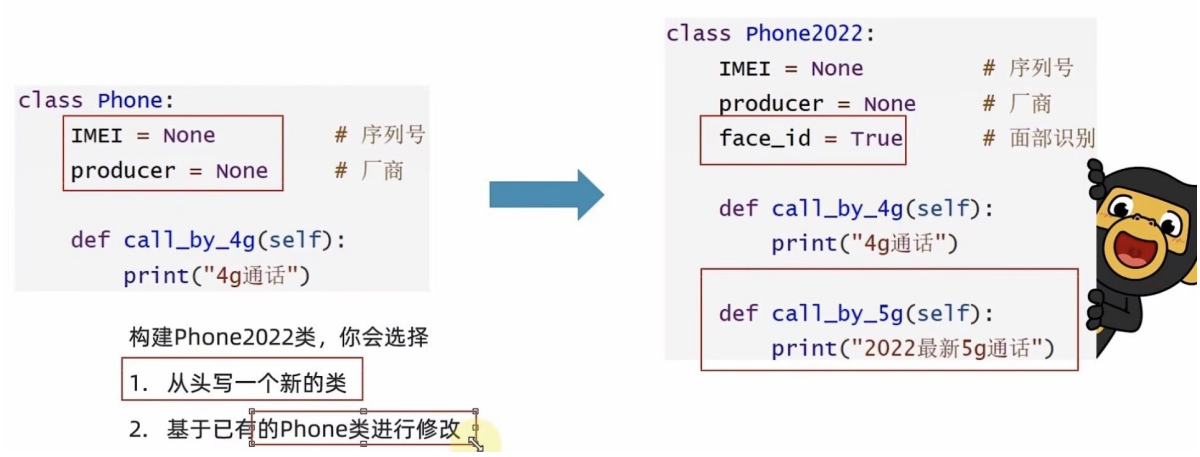
> 1. 封装的概念是指?
>   - 将现实世界事物在类中描述为属性和方法，即为封装。
> 2. 什么是私有成员？为什么需要私有成员
>   - 现实事物有部分属性和行为是不公开对使用者开放的。同样在类中描述属性和方法的时候也需要达到这个要求，就需要定义私有成员了
> 3. 如何定义私有成员
>   - 成员变量和成员方法的命名均已 __作为开头即可
> 4. 私有成员的访问限制
>   - 类对象无法访问私有成员
>   - 类中的其他成员可以访问私有成员

私有成员的意义：

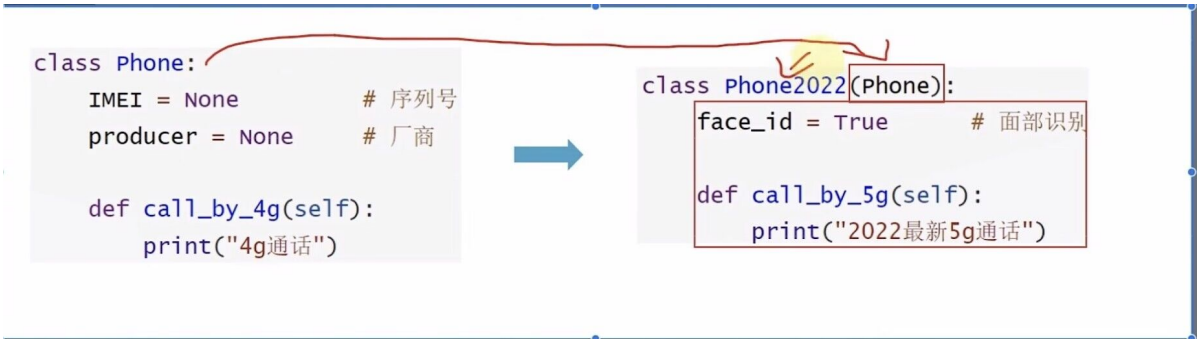
在类中提供仅供内部使用的属性和方法，而不对外开放(类对象无法使用)

7.继承的基础语法

7.1 理解继承的概念



7.2 掌握继承的使用方式

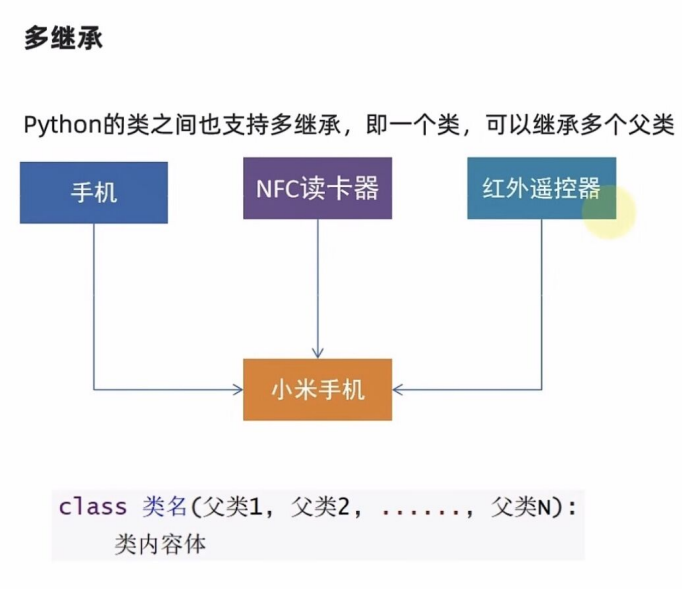


- 继承的语法

```
class 类名(父类名):
    类内容体

继承分为：单继承和多继承
使用上图语法，可以完成类的单继承
```

继承表示：将从父类哪里继承(复制)成员变量和成员方法(不含私有)。

- **多继承**




```
class Phone:
    IMEI = None          # 序列号
    producer = None      # 厂商

    def call_by_5g(self):
        print("5g通话")

class NFCReader:
    nfc_type = "第五代"
    producer = "HM"

    def read_card(self):
        print("读取NFC卡")

    def write_card(self):
        print("写入NFC卡")

class RemoteControl:
    rc_type = "红外遥控"

    def control(self):
        print("红外遥控开启")

class MyPhone(Phone, NFCReader, RemoteControl):
    pass
```

多继承中，如果父类有同名方法或属性，先继承的优先级高于后继承。

7.3 掌握pass关键字的作用

pass用来补全语法，表示无内容

7.4总结

1. 什么是继承?
    - 继承就是一个类，继承另一个类的成员变量和成员方法
    - 语法:

```
class 类名(父类[,父类2,.....,父类N]):
    类内容体
```

    - 子类构建的类对象，可以
        - 有自己的成员变量和成员方法
        - 使用父类的成员变量和成员方法
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，按照顺序从左到右依次继承
    - 多继承中：如果父类有同名的方法和属性，先继承的优先级高于后继承
3. Pass关键字的作用是什么

- pass只是一个普通的占位语句，用来保证我们的函数或者方法或者类定义的完整性，表示无内容，空的意思

8.继承的复写和使用父类成员

8.1 掌握复写父类成员的语法

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。

```
class Phone:
    IMEI = None          # 序列号
    producer = "ITCAST" # 厂商

    def call_by_5g(self):
        print("父类的5g通话")

class MyPhone(Phone):
    proucer = "ITHEIMA"          # 复写父类属性

    def call_by_5g(self):        # 复写父类方法
        print("子类的5g通话")
```

8.2 掌握如何在子类中调用父类成员

- 调用父类同名成员

如果直接调用同名成员，回调用到复写的子类中的同名成员，那么如何调用父类中与子类中同名的成员呢?

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的成员，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()

```
class Phone:
    IMEI = None          # 序列号
    producer = "ITCAST" # 厂商

    def call_by_5g(self):
        print("父类的5g通话")

class MyPhone(Phone):
    proucer = "ITHEIMA"

    def call_by_5g(self):
        # 方式1调用父类成员
        print(f"父类的品牌是：{Phone.producer}")
        Phone.call_by_5g(self)

        # 方式2调用父类成员
        print(f"父类的品牌是：{super().producer}")
        super().call_by_5g()

        print("子类的5g通话")
```

8.3 继承相关示例练习

1. 示例1——继承基础语法

```
class Base:
    def f1(self):
        print('Base.f1')

class Foo(Base):
    def f2(self):
        print('foo.f2')

object1 = Foo()
object1.f1()
object1.f2()
```

2.子类和父类如果同名，优先找子类的成员变量或成员方法

```
class Base:
    def f1(self):
        print('base.f1')
class Foo(Base):
    def f2(self):
        print('before')
        self.f1()
        print('foo.f2')
    def f1(self):
        print('foo.f1')
obj = Foo()
obj.f1()


>>>foo.f1
obj.f2()
>>>before
>>>foo.f1
>>>foo.f2
总结：
    执行对象.方法时，优先去当前对象关联的类中寻找，没有的话去父类中查找
    Python支持多继承：先继承左边，在继承右边
    self到底是谁？去self对应的那个类中去获取成员，没有按继承关系寻找
```

8.4总结

1. 复写表示:
    - 对父类的成员属性或成员方法进行重新的定义
2. 复写的语法
    - **在子类中重新实现同名成员方法或成员属性即可**
3. 在子类中，如何调用父类
    - 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：只可以在子类内部调用父类的同名成员，子类的实体类对象调用默认是调用子类复写的。

9.类型注解

- 理解为什么使用类型注解
- 掌握变量的类型注解语法

类型注解的主要功能：

- 帮助第三方IDE工具(如pycharm)对代码进行类型推断，协助做代码提示
- 帮助开发着自身对变量进行类型注释

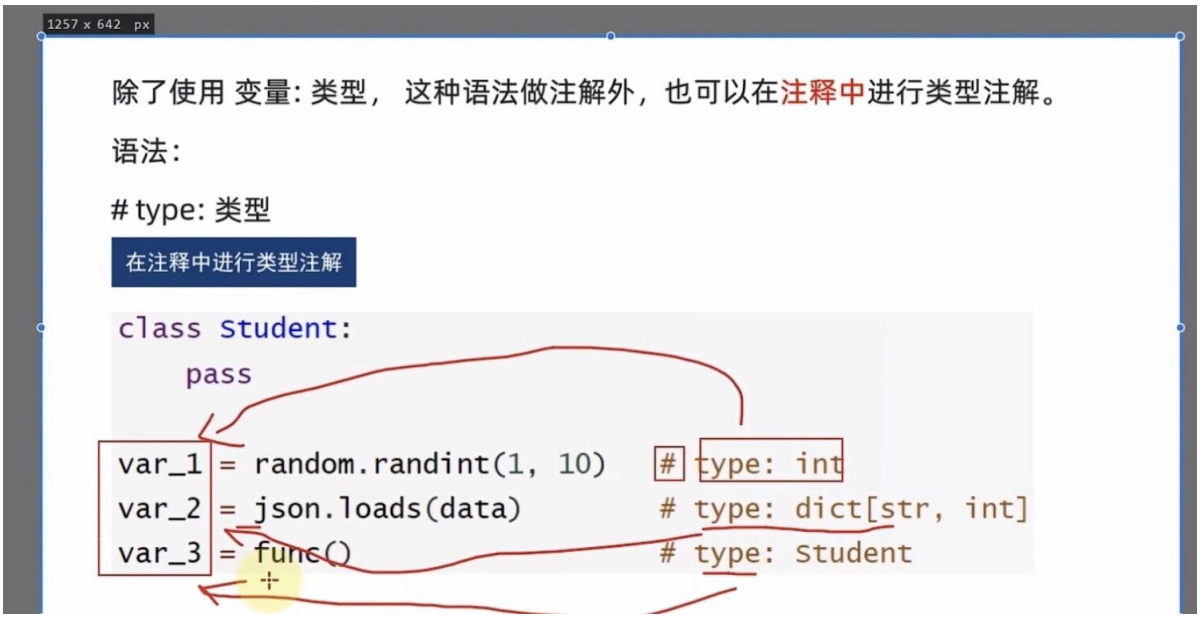
支持

- 变量的类型注解
- 函数(方法)形参列表和返回值的类型注解

9.1变量的类型注解

为变量设置类型注解

基础语法：变量:类型



- 变量的类型注解语法
- 语法1:变量:类型
- 语法2:在注释中，#type:类型
- 注解类型只是提示性的，并非决定性的。数据类型和注解类型无法对应也不会报错。

9.2函数(方法)的类型注解

9.2.1掌握为函数(方法)形参进行类型注解

- 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,.....):
    pass
```

9.2.2掌握为函数(方法)返回值进行类型注解

- 函数(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)->返回值类型:
    pass

def func(data:list) -> list:
    return data
```

==返回值类型注解的符号使用：->

9.3Union类型

9.3.1理解Union类型

- 对多种类型做注解

9.3.2掌握使用Union进行联合类型注解（注释混合类型数据）

```
from typing import Union
my_list: list[Union[str,int]]=[1,21,'itcast'] #表示数组中有str，int类型
my_dict: dict[str,Union[srt,int]] = {'name':'周杰伦','age':18}

使用Union[类型,类型,...,类型,类型]
可以定义联合类型注解
```

9.3.3总结

1. 什么是Union类型

使用Union可以定义联合类型注解

2. Union的使用方式

- 导包：from typing import Union
- 使用：Union[类型,类型,...,类型,类型]

10.多态

10.1多态的概念

- 多态：指的是多种状态，即完成某个行为时，使用不同的对象会得到不同的状态。
- 如何理解？

  类中传入不同的**子类对象**，得到不同的状态。

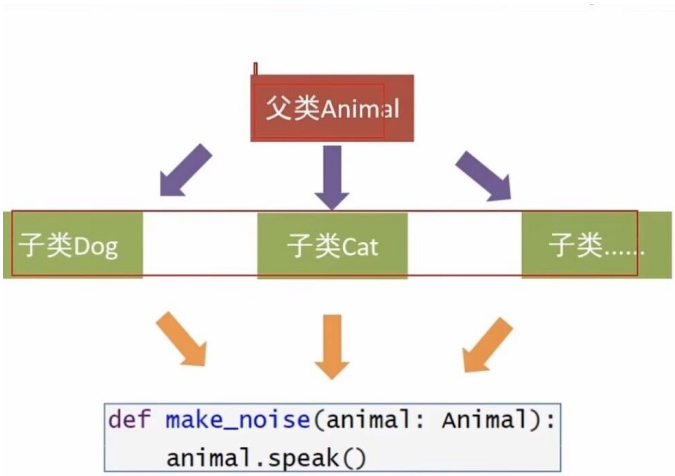
多态，指的是：多种状态，即完成某个行为时，使用不同的对象会得到不同的状态。

如何理解？

```
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        print("汪汪汪")

class Cat(Animal):
    def speak(self):
        print("喵喵喵")
```

```
def make_noise(animal: Animal):
    animal.speak()

dog = Dog()
cat = Cat()

make_noise(dog)      # 输出：汪汪汪
make_noise(cat)      # 输出：喵喵喵
```

同样的行为（函数），传入不同的对象，得到不同的状态



多态常作用在继承关系上.

比如

- 函数(方法)形参声明接收父类对象
- 实际传入父类的子类对象进行工作

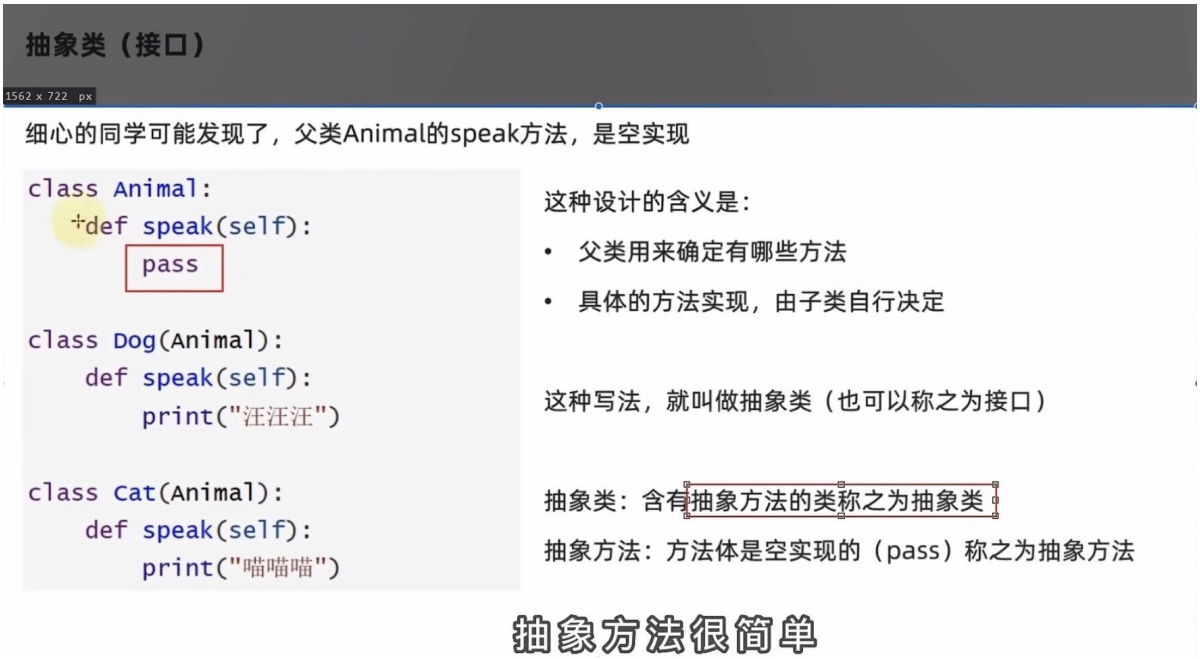
即:

- 以父类做定义声明
- 以子类做实际工作
- 用以获得同一行为,不同状态

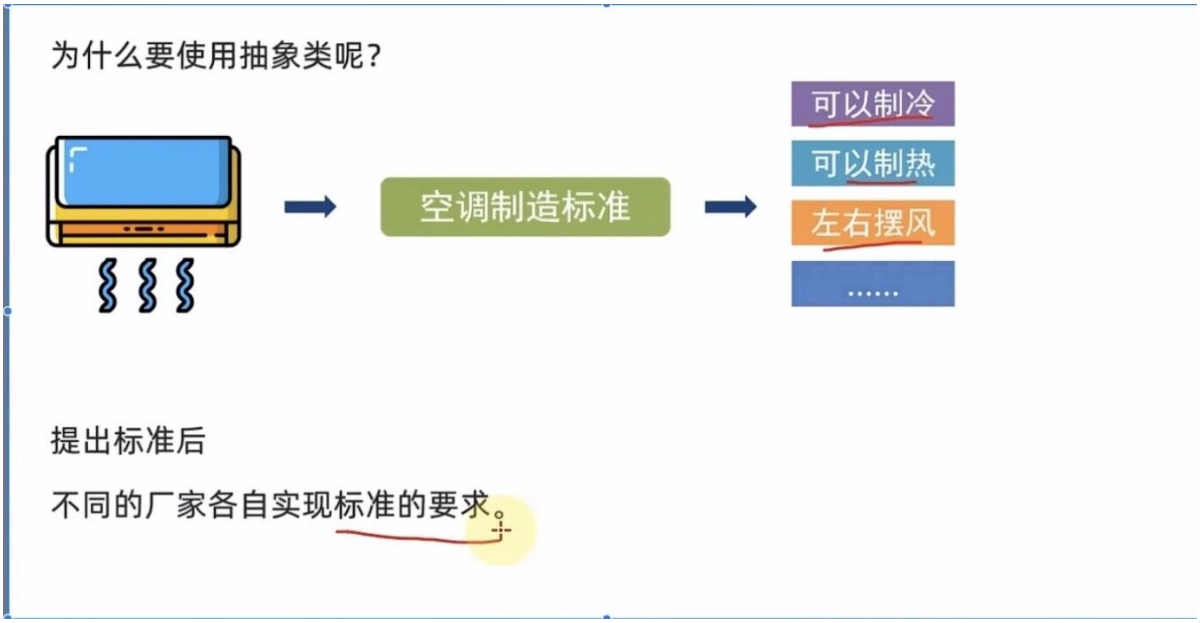
```
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        print("汪汪汪")

class Cat(Animal):
    def speak(self):
        print("喵喵喵")

#具体的函数，传入不同的对象，得到不同的状态
def make_noise(animal:Animal):
    animal.speak()

dog_object = Dog()
make_noise(dog_object)
>>>汪汪汪

cat_object = Cat()
make_noise(cat_object)
>>>喵喵喵
总结：同样的行为(函数)，传入不同的对象，得到不同的状态
```

- 普通的函数接受不同的对象，输出不同的结果

10.2 理解抽象类(接口)的编程思想



- 父类确定了有哪些方法，子类负责方法的实现



解决代码

```
#父类制定标准，不负责实现，顶层抽象设计
class AC:
    def cool_wind(self):
        """
        制冷
        :return:
        """
        pass
    def hot_wind(self):
        """
        制热
        :return:
        """
        pass
```

```
    def swing_l_r(self):
        """
        左右摆风
        :return:
        """
        pass

class Midea_AC(AC):
    def cool_wind(self):
        print("美的核心制冷科技")
    def hot_wind(self):
        print("美的空调电热丝加热")
    def swing_l_r(self):
        print("美的空调无风感自动摆动")


class Gree_AC(AC):
    def cool_wind(self):
        print("格力空调变频省电制冷")
    def hot_wind(self):
        print("格力空调电热丝加热")
    def swing_l_r(self):
        print("格力空调静音左右摆动")
'''
配合多态，完成
    抽象父类设计(设计标准)
    具体的子类实现(实现标准)
'''
def make_cool(ac:AC):
    ac.cool_wind()

gree_ac = Gree_AC()
midea_ac = Midea_AC()

make_cool(gree_ac)
>>> 格力空调变频省电制冷
make_cool(midea_ac)
>>> 美的核心制冷科技
```

10.3 多态的小结

1. 什么是多态?

- 多态指的是，同一行为(函数)，使用不同的子对象获取不同的状态。
- 如：定义函数(方法)，通过类型注解声明需要父类对象，实际传入子类对象进行工作，从而获得不同的工作状态。

2. 什么是抽象类(接口)

- 包含抽象方法的类，称之为抽象类。抽象方法是指：没有具体实现的方法(pass)称之为抽象方法。

3. 抽象类的作用

- 多用于作顶层设计(设计标准)，以便子类做具体实现。
- 也是对子类的一种软性约束，要求子类必须复写(实现)父类的一些方法

11.综合案例(数据分析案例)

1.使用面向对象的思想完成数据读取和处理

2.基于面向对象思想重新认识第三方库使用(PyErcharts)

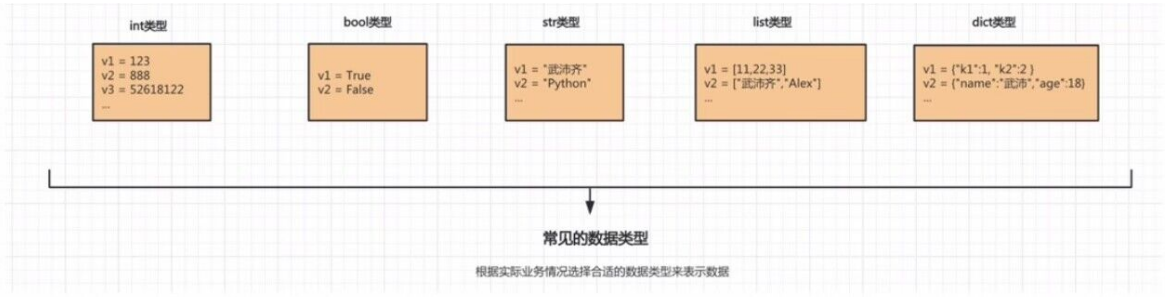
面向对象的三大特性：封装，继承，多态

- 封装：将方法封装到类中或将数据封装到对象中，便于以后使用
- 继承：将类中的公共的方法提取到基类中去实现
- 多态：Python默认支持多态(这种方式称之为鸭子类型)，最简单的基础下面的这段代码

```
def func(arg):
    v1 = arg.copy()#浅拷贝
    print(v1)
func("吴佩琦")
func(11,22,33,44)
```

12. 扩展：再看数据类型

在初步了解面向对象之后，再来看看之前学习的：str, list, dict等数据类型，他们其实都是一个类，根据类创建不同的对象。




```
# 实例化一个str类的对象v1
v1 = str("武沛齐")

# 通过对象执行str类中的upper方法。
data = v1.upper()

print(data)
```

str是一个类，按住CTRL+点击str，会跳到str这个类中。

查看str类中的方法：

- 进入到str类中，定位到空白处
- 点击放大镜，会出线str类中的所有方法

13.总结

13.1 面向对象编程的三大特性

- 封装
  - 将同一方法封装到一个类中，例如上述示例中：匪徒相关方法封装到Terrorist类中；警察的相关方法都封装到Police类中
  - **将数据封装到对象中**，可以通过`__init__`初始化方法在对象中封装一些数据，便于以后使用
- 继承
  - 传统理念中：儿子可以继承父亲的财产
  -
- 多态

作业

1.简述面向对象三大特性?

- 封装：将方法封装到类中或将数据封装到对象中
- 继承：将类中的公共的方法提取到基类中去实现

- 多态：

2.将下面的函数改写成类的方式，并调用：

```
def func(a1):
    print(a1)
```

3.面向对象中的self中指的是什么?

4.以下代码体现 向对象的什么特性?

```
class Person(object):
    def __init__(self,name,age,gender):
        self.name = name
        self.age = age
        self.gender = gender
obj = Person('吴佩琦',18,'男')
```

5.以下代码体现 面向对象的什么特点?

```
class Message(object):
    def email(self):
        """
        发送邮件
        """
        pass
    def msg(self):
        """
        发送短信
        """
```

```
        pass
    def wechat(self):
        """
        发送微信
        :return:
        """
        pass
```

6.看代码写结果

```
class Foo:
    def func(self):
        print('foo.func')
obj = Foo()
result = obj.func()
print(result)
结果：foo.func
```

7.看代码写结果

```
class Base1:
    def f1(self):
        print('base1.f1')

    def f2(self):
        print('base1.f2')

    def f3(self):
        print('base1.f3')
class Base2:
    def f1(self):
        print('base2.f1')
class Foo(Base1,Base2):
    def f0(self):
        print('foo.f0')
        self.f3()

obj = Foo()
obj.f0()
结果：
foo.f0
base1.f3
```

8.看代码写结果：

```
class Base:
    def f1(self):
        print('base.f1')

    def f3(self):
        self.f1()
        print('base.f3')

class Foo(Base):
```

```
    def f1(self):
        print('foo.f1')

    def f2(self):
        print('foo.f2')
        self.f3()

obj = Foo()
obj.f2()
```

9.补充代码实现

```
user_list = []
while True:
    user = input('请输入用户名：')
    pwd = input('请输入密码：')
    email = input('请输入邮箱：')
'''
#需求
1.while循环提示 户输入：户名，密码，邮箱(正则满，邮箱格式)
2.为每个 户创建 个对象，并添加到 表中。
3.当 表中的添加3个对象后，跳出循环并一次循环打印所有用户的姓名和邮箱
'''
```

10. 补充代码：实现 户注册和登录

```
class User:
    def __init__(self,name,pwd):
        self.name = name
        self.pwd = pwd

class Account:
    def __init__(self):
        #用户列表，数据格式:[user对象,user对象,user对象]
        self.user_list = []

    def login(self):
        '''
        用户登录，输入用户名和密码然后去self.user_list中校验用户合法性
        :return:
        '''
        pass
    def register(self):
        '''
        用户注册，每注册一个用户就创建一个user对象，然后添加到self.user_list中，表示创建成功
        :return:
        '''
        pass
    def run(self):
        '''
        主程序
        :return:
        '''
```

```
        pass

if __name__ == '__main__':
    obj = Account()
    obj.run()

```