- **面向对象三大特性**

    1. 封装
    2. 继承
    3. 多态

- **面向对象三大特性**

1. 封装

   将属性和方法封装在一个对象中，对外隐藏内部实现细节

封装的好处:

- 使用便捷
- 保证数据安全
- 利于代码维护

- **面向对象三大特性**

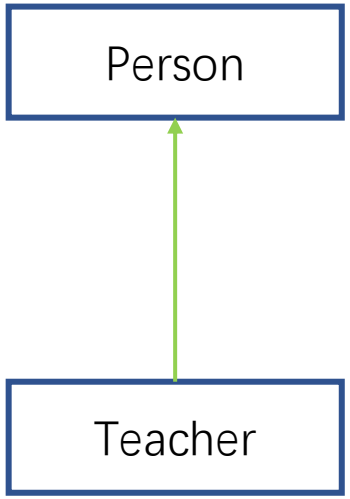
  2. 继承

     - 单继承
     - 多重继承

- **面向对象三大特性**

2. 继承

- 单继承　　仅继承了一个父类



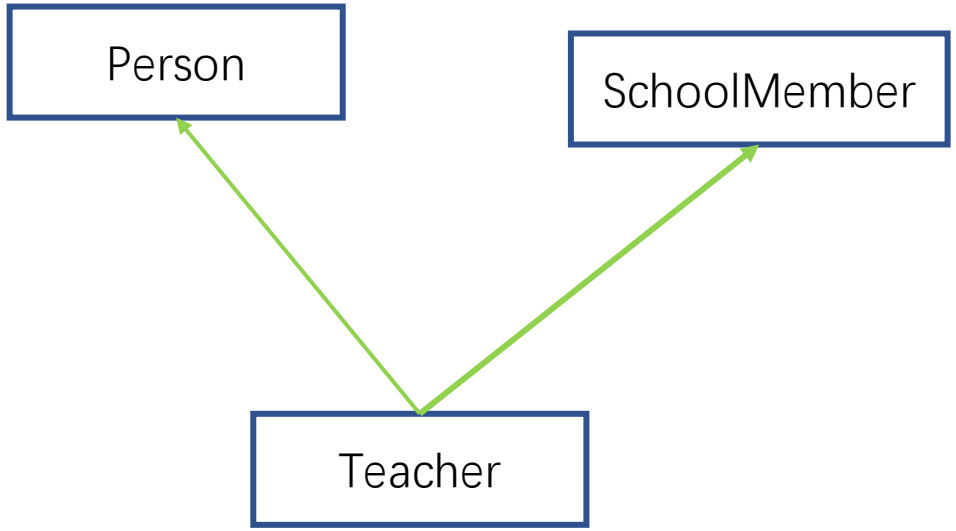

```python
class Person:
    def __init__(self, sn):
        self.sn = sn


class Teacher(Person):
    def __init__(self, stuff_no, sn):
        super(Teacher, self).__init__(sn)
        self.stuff_no = stuff_no
```

- **面向对象三大特性**

2. 继承

   - 多重继承　　继承多个父类




```python
class Person:
    def __init__(self, sn):
        self.sn = sn

class SchoolMember:
    def __init__(self, school_card_no):
        super().__init__(school_card_no)
        self.school_card_no = school_card_no

class Teacher(SchoolMember, Person):
    def __init__(self, sn):
        super().__init__(sn)
        self.stuff_no = sn
```

- **面向对象三大特性**

2. 继承
   - 使用__bases__查看父类

     ```
     print(Teacher.__bases__)
     ```

     (<class '__main__.SchoolMember'>, <class '__main__.Person'>,)

   - 调用父类方法
     1. 直接使用父类的类名调用
     2. 使用super

直接使用父类的类名调用父类方法

```
class Person:
    def __init__(self, sn):
        self.sn = sn


class Student(Person):
    def __init__(self, student_no):
        Person.__init__(self, student_no)
        self.student_no = student_no
```

Unbound Method(未绑定的方法）：ClassName.MemberFunctionName

super常用方式（以__init__函数为例)

- super(type, object).__init__()

```
class Teacher(SchoolMember, Person):
    def __init__(self, sn):
        super(Teacher, self).__init__(sn)
```

- super().__init__ 等价于 super(self.__class__, self).__init__()
  self.__class__ == Teacher
  super().__init__ == super(self.__class__, self)
  == super(Teacher, self)

使用super调用父类方法

- 理解mro(method resolution order)：定义了调用方法时，搜索的顺序。也被称为祖先树
- super根据mro生成一个代理对象，并按照mro定义的顺序进行父类方法调用

```python
class Person:
    def __init__(self, sn):
        self.sn = sn

class SchoolMember:
    def __init__(self, school_card_no):
        super().__init__(school_card_no)
        self.school_card_no = school_card_no

class Teacher(SchoolMember, Person):
    def __init__(self, sn):
        super().__init__(sn)
        self.stuff_no = sn
```

```python
print(Teacher.__mro__)
```

(<class '__main__.Teacher'>, <class '__main__. SchoolMember'>,
<class '__main__.Person'>, <class 'object'>)

super会根据type的__mro__生成代理，从object所属类的后一个开始进行调用

super_1.py && super_2.py

mro生成规则

mro从类自身开始，加入其所有父类，在加入父类的父类，一直循环直至基类object为止 (bfs)

菱形继承 mro.py

mro

当我们调用一个对象的方法时，也会根据mro进行搜索。所以在较深层次的继承关系中，函数调用的性能也会受到影响。

可以使用组件component的方式解决。

两种调用方式的优劣

直接调用父类方法	super
简单直观，方便处理参数不同的情况	对继承关系变化不敏感 方便实现遍历
对变化敏感，继承关系发生改变时需要修改代码	需要对不同参数的情况对特殊处理

参考资料:
https://rhettinger.wordpress.com/2011/05/26/super-considered-super/

- **面向对象三大特性**

  3.多态

  - 一个类所眼神的多种形态，以及调用时的多种形态
  - 调用时的多种形态指的是：在继承的前提下，使用不同的子类调用统一方法，会产生不同的效果