Python 程序设计 实验 7：面向对象编程

注意事项：

（1） 实验报告提交**截止日期：2021.05.13， 23:59pm**，迟交扣 20%，缺交 0 分。

（2） 实验报告内容包括：解决问题的思路与方法（如代码的解释）、遇到的问题以及收获（简单描述即可）、代码运行结果的展示。

（3） 实验报告提交方法：**blackboard**。

（4） 提交要求：实验报告+源代码，打包上传，命名：学号_姓名_实验报告_7。

（5） **禁止抄袭，一经发现 0 分处理（包括抄袭者和提供代码或实验报告者）!**

1. 编写类 RegularPolygon，表示正 n 边形。类包括：
   - 私有成员 n，要求为整型，代表正 n 边形边的数量，注意 $n \geq 3$；
   - 私有成员 side，代表正 n 边形每条边的长度；
   - 私有成员 x，代表正 n 边形中心的坐标在 x 轴上的数值；
   - 私有成员 y，代表正 n 边形中心的坐标在 y 轴上的数值；
   - 构造函数，输入参数为 n（默认为 3），side（默认为 1），x（默认为 0），y（默认为 0）；
   - 方法 getPerimeter，返回正 n 边形的周长；
   - 方法 getArea，返回正 n 边形的面积；
   - 方法 distanceToPolygon，输入参数为另外一个正 n 边形，返回两个正 n 边形中心的距离。

   自行设计测试函数验证 RegularPolygon 类代码的正确性。

2. 自定义数据结构栈。栈是一种后进先出（Last-In-First-Out）的数据结构。编写类 Stack，实现入栈、出栈、判断栈是否为空，是否满栈、以及改变栈容量等操作。

   Stack 类包括：
   - 私有成员 content，为一个列表，代表栈里的数据；
   - 私有成员 size，要求为整型，代表栈的容量；
   - 私有成员 current，要求为整型，代表栈当前数据的个数；
   - 方法 isempty，判断栈是否为空，返回 True/False；
   - 方法 empty,置空栈；
   - 方法 setSize，输入参数为新的栈的容量。注意新的栈容量可能小于原有的栈容量，统一将后进的元素删除；

- 方法 isFull，判断栈是否为空，返回 True/False；
- 方法 push，入栈，输入参数为新的元素；
- 方法 pop，出栈；
- 方法 show，打印当前栈的数据。

自行设计测试函数验证 Stack 类代码的正确性。

3. 时间类：设计一个名为 Time 的类。该类包含：
   - 表示时间的私有成员 hour、minute 和 second。
   - 构造 Time 对象的构造函数，使用**当前时间 time.time()**初始化小时、分钟和秒。
   - hour、minute 和 second 的 get 方法。
   - 方法 setTime（elapseTime），设置经过了 elapseTime（以秒为单位）后的新时间。

4. 继承 1：补充代码 lab7_4.py,使得代码输出如下。注：不允许在类中添加新的方法。

```
if __name__ =='__main__':
    zhangsan = Person('Zhang San', 19, 'man')
    zhangsan.show()
    #Name: Zhang San
    #Age: 19
    #Sex: man

    lisi = Teacher('Li Xi',32, 'man', 'Math')
    lisi.show()
    #Name: Li Xi
    #Age: 32
    #Sex: man
    #Department: Math

    lisi.setAge(40)
    lisi.setName("Li Si")
    lisi.show()
    #Name: Li Si
    #Age: 40
```

```
#Sex: man
#Department: Math
```

5. 继承 2:研究以下代码，思考代码的输出，并解释。

```
class China:
    def __init__(self, given, family):
        self.given = given
        self.family = family
    def __str__(self):
        return self.given + ' '    + self.family + '\n' + self.get_description()
    def get_description(self):
        return 'From China'
    def execute(self):
        print(self.family)

class Guangdong(China):
    def __init__(self):
        China.__init__(self, 'Ming', 'Li')

class England(China):
    def __init__(self):
        China.__init__(self, 'David', 'Beckham')
    def get_description(self):
        return 'From England'

def test_person(person):
    print(person)

ming = Guangdong()
ming.execute()
test_person(ming)
test_person(England())
```