

作业

1、实现温度的处理

- 1 实现华氏温度和摄氏温度的转换。
- 2 $^{\circ}\text{C} = 5 \times (^{\circ}\text{F} - 32) / 9$
- 3 $^{\circ}\text{F} = 9 \times ^{\circ}\text{C} / 5 + 32$
- 4
- 5 完成以上转换后，增加与开氏温度的转换， $\text{K} = ^{\circ}\text{C} + 273.15$

温度转换方法可以使用实例的方法，也可以使用类方法，使用类方法的原因是，为了不创建对象，就可以直接进行温度转换计算，这个类设计像个温度工具类。

先实现工具类

```
1 # 温度转换工具类
2 class Temperature:
3     # 温度转换
4     @classmethod
5     def c2f(cls, c):
6         return 9 * c / 5 + 32
7
8     @classmethod
9     def f2c(cls, f):
10        return (f - 32) * 5 / 9
11
12    @classmethod
13    def c2k(cls, c):
14        return c + 273.15
15
16    @classmethod
17    def k2c(cls, k):
18        return k - 273.15
19
20    # 华氏温度和开氏温度如何转换?
21
22    print(Temperature.c2f(40))
23    print(Temperature.f2c(104))
24    print(Temperature.c2k(40))
25    print(Temperature.k2c(313.15))
```

```
1 # 温度转换工具类
2 class Temperature:
3     # 温度转换
4     @classmethod
5     def c2f(cls, c):
6         return 9 * c / 5 + 32
7
8     @classmethod
9     def f2c(cls, f):
10        return (f - 32) * 5 / 9
11
12    @classmethod
```

```

13     def c2k(cls, c):
14         return c + 273.15
15
16     @classmethod
17     def k2c(cls, k):
18         return k - 273.15
19
20     # 华氏温度和开氏温度转换
21     @classmethod
22     def f2k(cls, f):
23         return cls.c2k(cls.f2c(f))
24
25     @classmethod
26     def k2f(cls, k):
27         return cls.c2f(cls.k2c(k))
28
29     print(Temperature.c2f(40))
30     print(Temperature.f2c(104))
31     print(Temperature.c2k(40))
32     print(Temperature.k2c(313.15))
33     print(Temperature.f2k(104))
34     print(Temperature.k2f(313.15))

```

给定一个温度值，先存着，用的时候再转

假定一般情况下，使用摄氏度为单位，传入温度值。

如果不给定摄氏度，一定会把温度值转换到摄氏度。

```

1  # 温度类，包含转换方法
2  class Temperature:
3      def __init__(self, t, unit='c'):
4          self._c = None
5          self._f = None
6          self._k = None
7
8      # 都要先转换到摄氏度，以后访问再计算其它单位的温度值
9      if unit == 'f':
10         self._f = t
11         self._c = self.f2c(t)
12     elif unit == 'k':
13         self._k = t
14         self._c = self.k2c(t)
15     else:
16         self._c = t
17
18     # 温度转换
19     @classmethod
20     def c2f(cls, c):
21         return 9 * c / 5 + 32
22
23     @classmethod
24     def f2c(cls, f):
25         return (f - 32) * 5 / 9
26
27     @classmethod
28     def c2k(cls, c):
29         return c + 273.15

```

```

30
31     @classmethod
32     def k2c(cls, k):
33         return k - 273.15
34
35     # 华氏温度和开氏温度如何转换?
36     @classmethod
37     def f2k(cls, f):
38         return cls.c2k(cls.f2c(f))
39
40     @classmethod
41     def k2f(cls, k):
42         return cls.c2f(cls.k2c(k))
43
44     print(Temperature.c2f(40))
45     print(Temperature.f2c(104))
46     print(Temperature.c2k(40))
47     print(Temperature.k2c(313.15))
48     print(Temperature.f2k(104))
49     print(Temperature.k2f(313.15))
50     print('-' * 30)
51
52     t = Temperature(104, 'f')
53     print(t.__dict__)

```

但是上面代码使用温度不方便，使用property装饰器构建属性

```

1  # 温度类，包含转换方法
2  class Temperature:
3      def __init__(self, t, unit='c'):
4          self._c = None
5          self._f = None
6          self._k = None
7
8      # 都要先转换到摄氏度，以后访问再计算其它单位的温度值
9      if unit == 'f':
10         self._f = t
11         self._c = self.f2c(t)
12     elif unit == 'k':
13         self._k = t
14         self._c = self.k2c(t)
15     else:
16         self._c = t
17
18     @property
19     def c(self):
20         return self._c
21
22     @property
23     def f(self): # 华氏温度
24         if self._f is None:
25             self._f = self.c2f(self._c)
26         return self._f
27
28     @property
29     def k(self): # 开氏温度
30         if self._k is None:

```

```

31         self._k = self.c2k(self._c)
32         return self._k
33
34     # 温度转换
35     @classmethod
36     def c2f(cls, c):
37         return 9 * c / 5 + 32
38
39     @classmethod
40     def f2c(cls, f):
41         return (f - 32) * 5 / 9
42
43     @classmethod
44     def c2k(cls, c):
45         return c + 273.15
46
47     @classmethod
48     def k2c(cls, k):
49         return k - 273.15
50
51     # 华氏温度和开氏温度如何转换?
52     @classmethod
53     def f2k(cls, f):
54         return cls.c2k(cls.f2c(f))
55
56     @classmethod
57     def k2f(cls, k):
58         return cls.c2f(cls.k2c(k))
59
60     print(Temperature.c2f(40))
61     print(Temperature.f2c(104))
62     print(Temperature.c2k(40))
63     print(Temperature.k2c(313.15))
64     print(Temperature.f2k(104))
65     print(Temperature.k2f(313.15))
66     print('-' * 30)
67
68     t = Temperature(104, 'f')
69     print(t.__dict__)
70     print(t.c, t.k, t.f)
71     print(t.__dict__)

```

2、图形

- 1、有Shape基类，要求所有子类都必须提供面积的计算，子类有三角形、矩形、圆。
- 2、上题圆类的数据可序列化

三角形面积——海伦公式：

$$p = (a + b + c) / 2$$

$$S = \sqrt{p(p - a)(p - b)(p - c)}$$

Shape基类，要求所有子类都必须提供面积的计算，子类有三角形、矩形、圆。

```
1 import math
```

```

2
3 class Shape:
4     @property
5     def area(self):
6         raise NotImplementedError('基类未实现')
7
8 class Triangle(Shape):
9     def __init__(self, a, b, c):
10         self.a = a
11         self.b = b
12         self.c = c
13
14     @property
15     def area(self):
16         p = (self.a + self.b + self.c) / 2
17         return math.sqrt(p * (p-self.a) * (p-self.b) * (p-self.c))
18
19 class Rectangle(Shape):
20     def __init__(self, width, height):
21         self.width = width
22         self.height = height
23
24     @property
25     def area(self):
26         return self.width * self.height
27
28 class Circle(Shape):
29     def __init__(self, radius):
30         self.d = radius * 2
31
32     @property
33     def area(self):
34         return math.pi * self.d * self.d * 0.25
35
36
37 shapes = [Triangle(3,4,5), Rectangle(3,4), Circle(4)]
38 for s in shapes:
39     print('The area of {} = {}'.format(s.__class__.__name__, s.area))

```

上例中，每取一次面积就要计算一次。在上例基础上，可以做以下改动

```

1 import math
2
3 class Shape:
4     def __init__(self):
5         self._area = None
6
7     @property
8     def area(self):
9         raise NotImplementedError('基类未实现')
10
11 class Triangle(Shape):
12     def __init__(self, a, b, c):
13         super().__init__()
14         self.a = a
15         self.b = b
16         self.c = c

```

```

17         self._p = (self.a + self.b + self.c) / 2
18
19     @property
20     def area(self):
21         if self._area is None:
22             p = self._p
23             self._area = math.sqrt(p * (p-self.a) * (p-self.b) * (p-self.c))
24         return self._area
25
26 class Rectangle(Shape):
27     def __init__(self, width, height):
28         super().__init__()
29         self.width = width
30         self.height = height
31
32     @property
33     def area(self):
34         if self._area is None:
35             self._area = self.width * self.height
36         return self._area
37
38 class Circle(Shape):
39     def __init__(self, radius):
40         super().__init__()
41         self.d = radius * 2
42
43     @property
44     def area(self):
45         if self._area is None:
46             self._area = 3.14 * self.d * self.d * 0.25
47         return self._area
48
49
50 shapes = [Triangle(3,4,5), Rectangle(3,4), Circle(4)]
51 for s in shapes:
52     print('The area of {} = {}'.format(s.__class__.__name__, s.area))

```

圆类的数据可序列化

```

1  import json
2  import msgpack
3
4  class SerializableMixin:
5      def dumps(self, t='json'):
6          if t == 'json':
7              return json.dumps(self.__dict__)
8          elif t == 'msgpack':
9              return msgpack.packb(self.__dict__)
10         else:
11             raise NotImplementedError('没有实现的序列化')
12
13 class SerializableCircle(SerializableMixin, Circle):
14     pass
15
16 scm = SerializableCircle(4)

```

```
17 print(scm.area)
18 s = scm.dumps('msgpack')
19 print(s)
```

