习题参考

1、随机整数生成类

可以先设定一批生成数字的**个数**，可设定指定生成的数值的**范围**

几种实现如下

```
import random

# 1 普通类实现
class RandomGen:
    def __init__(self, start=1, stop=100, count=10):
        self.start = start
        self.stop = stop
        self.count = count

    def generate(self):
        return [random.randint(self.start, self.stop) for i in range(self.count)]

print(RandomGen().generate())


# 2 作为工具类来实现，提供类方法
class RandomGen:
    @classmethod
    def generate(cls, start=1, stop=100, count=10):
        return [random.randint(start, stop) for i in range(count)]


print(RandomGen().generate())

# 3 生成器实现
class RandomGen:
    def __init__(self, start=1, stop=100, count=10):
        self.start = start
        self.stop = stop
        self.count = count
        self._gen = self._generate()

    def _generate(self):
        while True:
            yield random.randint(self.start, self.stop)

    def generate(self):
        return [next(self._gen) for i in range(self.count)]

print(RandomGen().generate())

# 变形
```

```
class RandomGen:
    def __init__(self, start=1, stop=100, count=10):
        self.start = start
        self.stop = stop
        self.count = count
        self._gen = self._generate()

    def _generate(self):
        while True:
            yield random.randint(self.start, self.stop)

    def generate(self):
        yield from (next(self._gen) for i in range(self.count))

print(list(RandomGen().generate()))
```

随机整数生成类，可以先设定一批生成数字的个数，可设定指定生成的数值的范围。**运行时还可以调整每批生成数字的个数。**
使用生成器实现，如下：

```
import random


# 3 生成器实现
class RandomGen:
    def __init__(self, start=1, stop=100, count=10):
        self.start = start
        self.stop = stop
        self.count = count
        self._gen = self._generate()

    def _generate(self):
        while True:
            yield random.randint(self.start, self.stop)

    def generate(self, count=0): # 可以后期在产生数据时控制个数
        count = self.count if count <= 0 else count
        return [next(self._gen) for i in range(count)]

print(RandomGen().generate(5))
print(RandomGen().generate())
```

换个思路
能否由 _generate()方法 一次性产生一批数据

```
import random


# 4 生成器另一种实现
class RandomGen:
    def __init__(self, start=1, stop=100, count=10):
        self.start = start
        self.stop = stop
        self._count = count # 保护
        self._gen = self._generate()

    def _generate(self):
        while True: # 一次yield一批
            yield [random.randint(self.start, self.stop) for _ in range(self._count)]

    def generate(self, count=0): # 可以后期在产生数据时控制个数
        if count > 0:
            self._count = count
        return next(self._gen)

print(RandomGen().generate(5))
print(RandomGen().generate())
```

```
import random


# 4 生成器另一种实现, property
class RandomGen:
    def __init__(self, start=1, stop=100, count=10):
        self.start = start
        self.stop = stop
        self._count = count # 保护
        self._gen = self._generate()

    def _generate(self):
        while True: # 一次yield一批
            yield [random.randint(self.start, self.stop) for _ in range(self._count)]

    def generate(self):
        return next(self._gen)

    @property
    def count(self):
        return self._count

    @count.setter
    def count(self, count):
        self._count = count

r = RandomGen()
print(r.count)
print(r.generate())
r.count = 3
print(r.generate())
```

2、打印坐标

使用上题中的类，随机生成20个数字，两两配对形成二维坐标系的坐标，把这些坐标组织起来，并打印输出

```
import random


# 4 生成器另一种实现, property
class RandomGen:
    def __init__(self, start=1, stop=100, count=10):
        self.start = start
        self.stop = stop
        self._count = count # 保护
        self._gen = self._generate()

    def _generate(self):
        while True: # 一次yield一批
            yield [random.randint(self.start, self.stop) for _ in range(self._count)]

    def generate(self):
        return next(self._gen)

    @property
    def count(self):
        return self._count

    @count.setter
    def count(self, count):
        self._count = count


# 坐标类
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

r = RandomGen()
points = [Point(x,y) for x,y in zip(r.generate(), r.generate())]

for p in points:
    print("{:2}:{:2}".format(p.x, p.y))
```

3、车辆信息

记录车的品牌mark、颜色color、价格price、速度speed等特征，并实现车辆管理，能增加车辆、显示全部车辆的信息功能

```
class Car: # 记录单一车辆
    def __init__(self, mark, speed, color, price):
        self.mark = mark
        self.speed = speed
        self.color = color
        self.price = price
```

```
class CarInfo:
    def __init__(self):
        self.__info = []

    def addcar(self, car: Car):
        self.__info.append(car)

    def getall(self):
        return self.__info

ci = CarInfo()
car = Car('audi', 400, 'red', 100)
ci.addcar(car)

ci.getall() # 返回所有数据，此时再实现格式打印
```

4、实现温度的处理

```
实现华氏温度和摄氏温度的转换。
℃ = 5 × (℉ - 32) / 9
℉ = 9 × ℃ / 5 + 32

完成以上转换后，增加与开氏温度的转换，K = ℃ + 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

```
# 温度转换工具类
class Temperature:
    # 温度转换
    @classmethod
    def c2f(cls, c):
        return 9 * c / 5 + 32

    @classmethod
    def f2c(cls, f):
        return (f - 32) * 5 / 9

    @classmethod
    def c2k(cls, c):
        return c + 273.15

    @classmethod
    def k2c(cls, k):
        return k - 273.15

    # 华氏温度和开氏温度如何转换?

print(Temperature.c2f(40))
print(Temperature.f2c(104))
print(Temperature.c2k(40))
print(Temperature.k2c(313.15))
```

```
# 温度转换工具类
class Temperature:
    # 温度转换
    @classmethod
    def c2f(cls, c):
        return 9 * c / 5 + 32

    @classmethod
    def f2c(cls, f):
        return (f - 32) * 5 / 9

    @classmethod
    def c2k(cls, c):
        return c + 273.15

    @classmethod
    def k2c(cls, k):
        return k - 273.15

    # 华氏温度和开氏温度转换
    @classmethod
    def f2k(cls, f):
        return cls.c2k(cls.f2c(f))

    @classmethod
    def k2f(cls, k):
        return cls.c2f(cls.k2c(k))

print(Temperature.c2f(40))
print(Temperature.f2c(104))
print(Temperature.c2k(40))
print(Temperature.k2c(313.15))
print(Temperature.f2k(104))
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。

```
# 温度类，包含转换方法
class Temperature:
    def __init__(self, t, unit='c'):
        self._c = None
        self._f = None
        self._k = None
        # 都要先转换到摄氏度，以后访问再计算其它单位的温度值
        if unit == 'f':
            self._f = t
            self._c = self.f2c(t)
        elif unit == 'k':
            self._k = t
            self._c = self.k2c(t)
        else:
            self._c = t

    # 温度转换
    @classmethod
    def c2f(cls, c):
        return 9 * c / 5 + 32

    @classmethod
    def f2c(cls, f):
        return (f - 32) * 5 / 9

    @classmethod
    def c2k(cls, c):
        return c + 273.15

    @classmethod
    def k2c(cls, k):
        return k - 273.15

    # 华氏温度和开氏温度如何转换?
    @classmethod
    def f2k(cls, f):
        return cls.c2k(cls.f2c(f))

    @classmethod
    def k2f(cls, k):
        return cls.c2f(cls.k2c(k))

print(Temperature.c2f(40))
print(Temperature.f2c(104))
print(Temperature.c2k(40))
print(Temperature.k2c(313.15))
print(Temperature.f2k(104))
print(Temperature.k2f(313.15))
print('-' * 30)

t = Temperature(104, 'f')
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

```
# 温度类，包含转换方法
class Temperature:
    def __init__(self, t, unit='c'):
        self._c = None
        self._f = None
        self._k = None

        # 都要先转换到摄氏度，以后访问再计算其它单位的温度值
        if unit == 'f':
            self._f = t
            self._c = self.f2c(t)
        elif unit == 'k':
            self._k = t
            self._c = self.k2c(t)
        else:
            self._c = t

    @property
    def c(self):
        return self._c

    @property
    def f(self): # 华氏温度
        if self._f is None:
            self._f = self.c2f(self._c)
        return self._f

    @property
    def k(self): # 开氏温度
        if self._k is None:
            self._k = self.c2k(self._c)
        return self._k

    # 温度转换
    @classmethod
    def c2f(cls, c):
        return 9 * c / 5 + 32

    @classmethod
    def f2c(cls, f):
        return (f - 32) * 5 / 9

    @classmethod
    def c2k(cls, c):
        return c + 273.15

    @classmethod
    def k2c(cls, k):
        return k - 273.15

    # 华氏温度和开氏温度如何转换?
    @classmethod
    def f2k(cls, f):
        return cls.c2k(cls.f2c(f))

    @classmethod
```

```python
    def k2f(cls, k):
        return cls.c2f(cls.k2c(k))

print(Temperature.c2f(40))
print(Temperature.f2c(104))
print(Temperature.c2k(40))
print(Temperature.k2c(313.15))
print(Temperature.f2k(104))
print(Temperature.k2f(313.15))
print('-' * 30)

t = Temperature(104, 'f')
print(t.__dict__)
print(t.c, t.k, t.f)
print(t.__dict__)
```

5、模拟购物车购物

思路
购物车购物，分解得到两个对象`购物车`、`物品`，一个操作`购买`。
购买不是购物车的行为，其实是人的行为，但是对于购物车来说就是`增加add`。
商品有很多种类，商品的属性多种多样，怎么解决？
购物车可以加入很多不同的商品，如何实现？

```python
class Color:
    RED = 0
    BLUE = 1
    GREEN = 2
    GOLDEN = 3
    BLACK = 4
    OTHER = 1000

class Item:
    def __init__(self, **kwargs):
        self.__spec = kwargs

    def __repr__(self):
        return str(sorted(self.__spec.items()))

class Cart:
    def __init__(self):
        self.items = []

    def additem(self,item:Item):
        self.items.append(item)

    def getallitems(self):
        return self.items

mycart = Cart()
myphone = Item(mark='Huawei', color=Color.GOLDEN, memory='4G')
mycart.additem(myphone)
```

```python
mycar = Item(mark='Red Flag', color=Color.BLACK, year=2017)
mycart.additem(mycar)

print(mycart.getallitems())
```

注意，以上代码只是一个非常简单的一个实现，生产环境实现购物车的增删改查，要考虑更多。