# -\*- coding: utf-8 -\*-  
  
# Эмуляция операций и операторов python с помощью специальных методов  
  
# Эмуляция операторов сравнения  
#  
# object.\_\_eq\_\_(self, other) - равенство двух объектов ==  
# object.\_\_ne\_\_(self, other) - не равно !=  
# object.\_\_lt\_\_(self, other) - строго меньше <  
# object.\_\_le\_\_(self, other) - меньше или равно <=  
# object.\_\_gt\_\_(self, other) - строго больше >  
# object.\_\_ge\_\_(self, other) - больше или равно >=  
#  
# должны возвращать boolean - True/False  
  
class Backpack:  
 *""" Рюкзак """* def \_\_init\_\_(self, gift=None):  
 self.content = []  
 if gift:  
 self.content.append(gift)  
  
 def \_\_eq\_\_(self, other):  
 return self.content == other.content  
  
  
my\_backpack = Backpack(gift='бутерброд')  
son\_backpack = Backpack(gift='бутерброд')  
  
if my\_backpack == son\_backpack:  
 print('Как мы похожи...')  
  
if Backpack.\_\_eq\_\_(self=my\_backpack, other=son\_backpack):  
 print('Как мы похожи...')  
  
  
# Эмуляция математических операций  
# 2 + 2  
# my\_car + truck  
#  
# object.\_\_add\_\_(self, other) - сложение +  
# object.\_\_sub\_\_(self, other) - вычитание -  
# object.\_\_mul\_\_(self, other) - умножение \*  
# object.\_\_truediv\_\_(self, other) - деление /  
# object.\_\_floordiv\_\_(self, other) - целочисленное деление //  
# object.\_\_mod\_\_(self, other) - остаток от деления %  
# object.\_\_pow\_\_(self, other) - возведение в степень \*\*  
# object.\_\_lshift\_\_(self, other) - побитовый сдвиг влево <<  
# object.\_\_rshift\_\_(self, other) - побитовый сдвиг вправо >>  
# object.\_\_and\_\_(self, other) - побитовое И &  
# object.\_\_xor\_\_(self, other) - побитовое исключающее ИЛИ ^  
# object.\_\_or\_\_(self, other) - побитовое ИЛИ |  
#  
# должны возвращать объект  
  
class Backpack:  
 *""" Рюкзак """* def \_\_init\_\_(self, gift=None):  
 self.content = []  
 if gift:  
 self.content.append(gift)  
  
 def \_\_str\_\_(self):  
 return 'Backpack: ' + ', '.join(self.content)  
  
 def \_\_add\_\_(self, other):  
 new\_obj = Backpack()  
 new\_obj.content.extend(self.content)  
 new\_obj.content.extend(other.content)  
 return new\_obj  
  
  
my\_backpack = Backpack(gift='бутерброд')  
son\_backpack = Backpack(gift='банан')  
new\_backpack = my\_backpack + son\_backpack  
print(new\_backpack)  
  
# other\_backpack = my\_backpack + ['яблоко', 'апельсин', ]  
# print(other\_backpack)  
  
  
# для операций расширенного присвоения служат методы  
# object.\_\_iadd\_\_(self, other) - +=  
# object.\_\_isub\_\_(self, other) - -=  
# object.\_\_imul\_\_(self, other) - \*=  
# object.\_\_itruediv\_\_(self, other) - /+  
# object.\_\_ifloordiv\_\_(self, other) - //=  
# object.\_\_imod\_\_(self, other) - %=  
# object.\_\_ipow\_\_(self, other) - \*\*=  
# object.\_\_ilshift\_\_(self, other) - <<=  
# object.\_\_irshift\_\_(self, other) - >>=  
# object.\_\_iand\_\_(self, other) - &=  
# object.\_\_ixor\_\_(self, other) - ^=  
# object.\_\_ior\_\_(self, other) - |=  
#  
# они изменяют сам объект (по месту, inplace)  
  
class Backpack:  
 *""" Рюкзак """* def \_\_init\_\_(self, gift=None):  
 self.content = []  
 if gift:  
 self.content.append(gift)  
  
 def \_\_str\_\_(self):  
 return 'Backpack: ' + ', '.join(self.content)  
  
 def \_\_iadd\_\_(self, other):  
 self.content.extend(other.content)  
 return self  
  
  
my\_backpack = Backpack(gift='бутерброд')  
son\_backpack = Backpack(gift='банан')  
my\_backpack += son\_backpack  
print(my\_backpack)  
  
  
# Не обязательно возвращать объект такого же класса(типа)  
class Bread:  
  
 def \_\_str\_\_(self):  
 return 'Я хлеб'  
  
 def \_\_add\_\_(self, other):  
 return Sandwich(part1=self, part2=other)  
  
  
class Sausage:  
  
 def \_\_str\_\_(self):  
 return 'Я колбаса'  
  
 def \_\_add\_\_(self, other):  
 return Sandwich(part1=self, part2=other)  
  
  
class Sandwich:  
  
 def \_\_init\_\_(self, part1, part2):  
 self.part1 = part1  
 self.part2 = part2  
  
 def \_\_str\_\_(self):  
 return 'Я бутерброд. Состою из ' + str(self.part1) + ' и ' + str(self.part2)  
  
  
borodinsky = Bread()  
salami = Sausage()  
result = borodinsky + salami  
print(result)  
  
  
# эмуляция вызова функции - это когда объект ведет себя как функция  
# object.\_\_call\_\_(self[, args...]) - вызов как функции  
  
def func(\*args, \*\*kwargs):  
 print(args, kwargs)  
  
  
print(func)  
func(a=2, b=2)  
  
  
class MyFunction:  
  
 def \_\_call\_\_(self, \*args, \*\*kwargs):  
 print(args, kwargs)  
  
  
func = MyFunction()  
print(func)  
  
func(a=2, b=2)  
  
  
# это используется для немного странных и увлекательных вещей :)  
# погрузимся чуть-чуть в функциональный стиль  
class Multyplier:  
  
 def \_\_init\_\_(self, factor=2):  
 self.factor = factor  
  
 def \_\_call\_\_(self, \*args):  
 res = []  
 for item in args:  
 res.append(item \* self.factor)  
 return res  
  
  
mul\_by\_27 = Multyplier(factor=27)  
result = mul\_by\_27(1, 2, 3, 4)  
print(result)  
  
# multipiers = []  
# for factor in (2, 3, 4, 5):  
# mul = Multyplier(factor=factor)  
# multipiers.append(mul)  
# print(multipiers)  
#  
# for mul in multipiers:  
# print(mul(10, 20, 30))  
  
  
# все специальные методы перечислены в  
# https://docs.python.org/3/reference/datamodel.html#special-method-names