*"""  
Деменчук Георгий, ПИ19-4  
Задания 1-12  
"""***import** random  
**from** itertools **import** permutations  
  
  
**class** Task1(object):  
 *"""  
 Нaпишите программу, на вход которой подаётся список чисел одной строкой.  
 Программа должна для каждого элемента этого списка вывести сумму двух его cоседей.  
 Для элeментов списка, являющиxся крайними, одним из соседей считается элемент, находящий на противоположном конце этого списка.  
 Например, если на вход подаётся cписок «1 3 5 6 10», то на выход ожидается cписок «13 6 9 15 7».  
 Если на вход пришло только однo число, надо вывести его же.  
 Вывoд должен содержать одну строку с чиcлами новoго списка, разделёнными пробeлом.  
 """* **def** \_\_init\_\_(self):  
 out\_text = **"Введите список чисел через запятую -> "** self.input\_l = [int(x) **for** x **in** input(out\_text).split(**","**)]  
 self.processing()  
 print(self.result)  
  
 **def** processing(self):  
 out\_l = []  
 l = self.input\_l  
 **if** len(l) == 1:  
 self.result = l[0]  
 **else**:  
  
 *# В начало добавляем сумму последнего и первого элемента* out\_l.append(l[-1] + l[1])  
 print(l[0], **"пары: "**, l[-1], l[1])  
 **for** i **in** range(1, len(l) - 1):  
 print(l[i], **"пары: "**, l[i - 1], l[i + 1])  
 out\_l.append(l[i - 1] + l[i + 1])  
  
 *# В конец добавляем сумму предпоследнего и нулевого элемента* out\_l.append(l[-2] + l[0])  
 print(l[-1], **"пары: "**, l[-2], l[0])  
  
 self.result = out\_l  
  
  
**class** Task2(object):  
 *"""  
 Нaпишите прогрaмму, котoрая принимает на вход спиcок чисел в одной cтроке и выводит на экран в   
 oдну строкy значения, котoрые повторяются в нём бoлее одного раза.  
 Выводимые числа не дoлжны повторяться, пoрядок их вывода может быть произвольным.  
 Нaпример: 4 8 0 3 4 2 0 3  
 """* **def** \_\_init\_\_(self):  
 out\_text = **"Введите список чисел через запятую -> "** self.input\_l = [int(x) **for** x **in** input(out\_text).split(**","**)]  
 self.processing()  
 print(self.result)  
  
 **def** processing(self):  
 d = {}  
 result = []  
 l = self.input\_l  
  
 **for** e **in** l:  
 d[e] = 0  
  
 **for** e **in** l:  
 d[e] += 1  
  
 **for** k, v **in** d.items():  
 **if** v > 1:  
 result.append(k)  
  
 self.result = **"Повторяющиеся значения:\n"** + **""**.join(str(x) + **" " for** x **in** result)  
  
  
**class** Task3(object):  
 *"""  
 Выполните oбработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов.  
 Все элeменты имeют целый тип. Дано целое число H.  
 Опрeделите, какие столбцы имeют хотя бы однo такое число, а какие не имeют.  
 """* **def** \_\_init\_\_(self):  
 **try**:  
 self.n = int(input(**"Введите количество строк N в матрице -> "**))  
 self.m = int(input(**"Введите количество столбцов M в матрице -> "**))  
 **except**:  
 print(**"Ошибка ввода данных"**)  
 **return** self.matrix\_gen()  
 self.element\_search()  
  
 **def** matrix\_gen(self):  
 m = [[random.randint(10, 99) **for** c **in** range(self.m)] **for** r **in** range(self.n)]  
 print(**"Исходная матрица:"**)  
 **for** e **in** m:  
 print(e)  
 self.matrix = m  
  
 **def** element\_search(self):  
 d = {}  
 **try**:  
 number = int(input(**"Введите число H для поиска по столбцам -> "**))  
 **except**:  
 print(**"Ошибка ввода данных"**)  
 **return  
  
 for** i **in** range(len(self.matrix[0])):  
 d[i] = 0  
  
 **for** i **in** range(len(self.matrix[0])):  
 **for** j **in** range(len(self.matrix)):  
 **if** self.matrix[j][i] == number:  
 d[i] += 1  
  
 **for** k, v **in** d.items():  
 **if** v == 0:  
 print(**"Столбец №"** + str(k + 1) + **" - значений нет"**)  
 **else**:  
 print(**"Столбец №"** + str(k + 1) + **" - повторение значения "** + str(v) + **" раз(а)"**)  
  
  
**class** Task4(object):  
 *"""  
 Список задается пользователем с клавиатуры. Определите, является ли список симметричным .  
 """* **def** \_\_init\_\_(self):  
 **try**:  
 self.n = int(input(**"Введите размерность матрицы -> "**))  
 **except**:  
 print(**"Ошибка ввода данных"**)  
 **return** self.matrix\_input()  
 self.symmetry\_detect()  
  
 **def** check\_digit(self, e):  
 **try**:  
 **return** int(e)  
 **except**:  
 **return** e  
  
 *# TODO СДЕЛАТЬ В ОДНУ СТРОЧКУ* **def** matrix\_input(self):  
 l = []  
 **for** i **in** range(self.n):  
 l.append([])  
 **for** j **in** range(self.n):  
 l[i].append(self.check\_digit(input(**"Введите элемент ["** + str(i) + **"]["** + str(j) + **"] ->"**)))  
  
 print(**"\nИсходная матрица:"**)  
 **for** e **in** l:  
 print(e)  
 self.out\_l = l  
  
 **def** symmetry\_detect(self):  
  
 d = {  
 **True**: **"Список является симметричным"**,  
 **False**: **"Список НЕ является симметричным"** }  
  
 l = self.out\_l  
 flag = **True  
 for** i **in** range(len(l)):  
 **for** j **in** range(len(l)):  
 **if** l[i][j] != l[j][i]:  
 flag = **False** print(d[flag])  
  
  
**class** Task5(object):  
 *"""  
 Список задается пользователем с клавиатуры.  
 Определите, можно ли удалить из списка каких-нибудь два элемента так,   
 чтобы новый список оказался упорядоченным  
 """* **def** \_\_init\_\_(self):  
 self.l = list(set([self.check\_digit(e) **for** e **in** input(**"Введите элементы списка через запятую -> "**).split(**","**)]))  
 self.processing()  
  
 **def** check\_digit(self, e):  
 **try**:  
 **return** int(e)  
 **except**:  
 **return** e  
  
 **def** processing(self):  
 this\_list = self.l  
 perm = permutations(this\_list, 2)  
 **for** e **in** list(perm):  
 print(e)  
 buf\_list = this\_list[:]  
 buf\_list.remove(e[0])  
 buf\_list.remove(e[1])  
 buf\_list1 = buf\_list[:]  
 buf\_list1.sort()  
  
 **if** (buf\_list1 == buf\_list):  
 print(**"Удалили элементы"**, e[0], **"и"**, e[1], **"\nПолучили:"**, buf\_list)  
 **break  
  
  
class** Task6(object):  
 *"""  
 Список задается пользователем с клавиатуры.  
 Определите, сколько различных значений содержится в списке.  
 """* **def** \_\_init\_\_(self):  
 self.processing()  
  
 **def** check\_digit(self, e):  
 **try**:  
 **return** int(e)  
 **except**:  
 **return** e  
  
 **def** processing(self):  
 s = **"Введите элементы списка через запятую ->"** r = len(set([self.check\_digit(e) **for** e **in** input(s).split(**","**)]))  
 print(**"Уникальных значений в списке:"**, r)  
  
  
**class** Task7(object):  
 *"""  
 Список задается пользователем с клавиатуры.  
 Удаление из списка элементов, значения которых уже встречались в предыдущих элементах  
 """* **def** \_\_init\_\_(self):  
 self.processing()  
  
 **def** check\_digit(self, e):  
 **try**:  
 **return** int(e)  
 **except**:  
 **return** e  
  
 **def** processing(self):  
 s = **"Введите элементы списка через запятую ->"** r = list(set([self.check\_digit(e) **for** e **in** input(s).split(**","**)]))  
 print(**"Список без повторных значений: "**, r)  
  
  
**class** Task8(object):  
 *"""  
 Пользователь вводит упорядоченный список книг (заданной длины по алфавиту).  
 Добавить новую книгу, сохранив  
 упорядоченность списка по алфавиту  
 """* **def** \_\_init\_\_(self):  
 self.add\_values()  
 self.add\_new\_value()  
  
 **def** add\_values(self):  
 books\_list = input(**"Введите книги через запятую -> "**).split(**","**)  
 **for** i **in** range(len(books\_list)):  
 **if** books\_list[i][0] == **" "**:  
 books\_list[i] = books\_list[i][1:]  
 self.books\_list = sorted(books\_list, key=str.lower)  
 print(**"Введенный list:\n"** + str(self.books\_list))  
  
 **def** add\_new\_value(self):  
 self.new\_book = input(**"Введите название книги для добавления в существующий список ->"**)  
 self.add\_book\_to\_list()  
  
 **def** add\_book\_to\_list(self):  
 buf\_list = [e.lower() **for** e **in** self.books\_list]  
 input\_element = self.new\_book.lower()  
  
 **for** i **in** range(len(buf\_list)):  
 **if** buf\_list[i] > input\_element:  
 index = i  
 **break** print(**"Индекс для вставки:"**, index)  
 out\_list = self.books\_list[:index] + [self.new\_book] + self.books\_list[index:]  
 print(**"Результирующий list:\n"** + str(out\_list))  
  
 *# Driver function   
 # list = [1, 2, 4]  
 # n = 3  
  
 # print(insert(list, n))***class** Task9(object):  
 *"""  
 Дан список целых чисел. Упорядочьте по возрастанию только:  
 а) положительные числа;  
 б) элементы с четными порядковыми номерами в списке.  
 """* **def** \_\_init\_\_(self):  
  
 **try**:  
 n = int(input(**"Введите размерность списка ->"**))  
 **except**:  
 print(**"Что-то пошло не так при вводе данных"**)  
 **return** self.l = [random.randint(-10, 10) **for** \_ **in** range(n)]  
 print(**"Исходная матрица:\n"**, self.l)  
 self.a\_processing()  
 self.b\_processing()  
 print(**"Упорядочьте по возрастанию только положительные числа:\n"**, self.a\_l)  
 print(**"Упорядочьте по возрастанию только элементы с четными порядковыми номерами в списке:\n"**, self.b\_l)  
  
 **def** a\_processing(self):  
 buf\_list = []  
 matrix = self.l  
 **for** i **in** range(len(matrix)):  
 **if** matrix[i] > 0:  
 buf\_list.append(matrix[i])  
 buf\_list.sort()  
  
 index = 0  
 **for** i **in** range(len(matrix)):  
 **if** matrix[i] > 0:  
 matrix[i] = buf\_list[index]  
 index += 1  
 self.a\_l = matrix  
  
 **def** b\_processing(self):  
 buf\_list = []  
 matrix = self.l  
 **for** i **in** range(len(matrix)):  
 **if** i % 2 == 0:  
 buf\_list.append(matrix[i])  
 buf\_list.sort()  
  
 index = 0  
 **for** i **in** range(len(matrix)):  
 **if** i % 2 == 0:  
 matrix[i] = buf\_list[index]  
 index += 1  
 self.b\_l = matrix  
  
  
**class** Task10(object):  
 *"""  
 Даны два списка. Определите, совпадают ли множества их элементов.   
 """* **def** \_\_init\_\_(self):  
 self.l1 = []  
 self.l2 = []  
 self.input\_data()  
 self.comparator()  
  
 **def** input\_data(self):  
  
 **try**:  
 n1 = int(input(**"Введите размерность списка №1 ->"**))  
 n2 = int(input(**"Введите размерность списка №2 ->"**))  
 **except**:  
 print(**"Что-то пошло не так при вводе данных"**)  
 **return** print(**"\*Заполение списка №1\*"**)  
 **for** i **in** range(n1):  
 self.l1.append(input(**"Введите элемент списка №"** + str(i) + **" -> "**))  
  
 print(**"\*Заполение списка №2\*"**)  
 **for** i **in** range(n2):  
 self.l2.append(input(**"Введите элемент списка №"** + str(i) + **" -> "**))  
  
 **def** comparator(self):  
  
 d = {  
 **True**: **"Множества списокв совпадают"**,  
 **False**: **"Множества списков НЕ совпадают"**,  
 }  
  
 print(d[set(self.l1) == set(self.l2)])  
  
  
**class** Task11(object):  
 *"""  
 Дан список. После каждого элемента добавьте предшествующую ему часть списка.  
 """* **def** \_\_init\_\_(self):  
 self.l = input(**"Введите элементы списка через запятую -> "**).split(**","**)  
 self.processing()  
 print(self.result)  
  
 **def** processing(self):  
 s = self.l  
 counter = -1  
 output\_list = [s[0]]  
 **for** element\_first **in** s:  
 counter += 1  
 **if** counter == 0:  
 **pass  
 else**:  
 output\_list.append(element\_first)  
 **for** element\_alter **in** output\_list[:counter]:  
 output\_list.append(element\_alter)  
 counter += 1  
 self.result = output\_list  
  
  
**class** Task12(object):  
 *"""  
 Пусть элементы списка хранят символы предложения. Замените каждое вхождение слова 'itma  
 threpetitor' на 'silence'.  
 """* **def** \_\_init\_\_(self):  
 self.list = list(input(**"Введите строку для замены 'itmathrepetitor' на 'silence' -> "**))  
 self.sub\_list = list(**"itmathrepetitor"**)  
 self.replace\_list = list(**"silence"**)  
 self.processing()  
  
 **def** get\_sublist\_index(self):  
  
 sub = self.sub\_list  
 lst = self.list  
 sublen = len(sub)  
 first = sub[0] **if** sub **else** []  
 indx = -1  
  
 **while True**:  
 **try**:  
 indx = lst.index(first, indx + 1)  
 **except** ValueError:  
 **break  
 if** sub == lst[indx: indx + sublen]:  
 **return True**, indx, indx + len(sub)  
 **return False**, 0, 0  
  
 **def** processing(self):  
  
 print(**"Список до замены:\n"** + str(self.list))  
 processing\_flag = **True  
  
 while** processing\_flag == **True**:  
 index\_tuple = self.get\_sublist\_index()  
 **if** index\_tuple[0] == **True**:  
 print(**"Замена подсписка по индексам"**, index\_tuple[1], index\_tuple[2])  
 **del** self.list[index\_tuple[1]:index\_tuple[2]]  
 self.list[index\_tuple[1]:index\_tuple[1]] = self.replace\_list  
 **else**:  
 processing\_flag = **False** print(**"Список после замены:\n"** + str(self.list))  
  
  
**class** Task13(object):  
 *"""  
 Дан текстовый файл. Создайте двусвязный список, каждый элемент которого содержит  
 количество символов в соответствующей строке текста.  
 """* **def** \_\_init\_\_(self):  
 **pass  
  
  
def** main():  
 d = {  
 **"1"**: Task1,  
 **"2"**: Task2,  
 **"3"**: Task3,  
 **"4"**: Task4,  
 **"5"**: Task5,  
 **"6"**: Task6,  
 **"7"**: Task7,  
 **"8"**: Task8,  
 **"9"**: Task9,  
 **"10"**: Task10,  
 **"11"**: Task11,  
 **"12"**: Task12,  
 }  
  
 input\_str = input(**"Введите номер задания ->"**)  
 **if** input\_str **in** d:  
 d[input\_str]()  
 **else**:  
 print(**"Такого номера нет!"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

*"""  
Дан текстовый файл. Создайте двусвязный список, каждый элемент которого содержит  
количество символов в соответствующей строке текста.  
"""*TXT\_FILE = **"./text.txt"  
  
  
class** Node(object):  
  
 **def** \_\_init\_\_(self, value=**None**, next=**None**, previous=**None**):  
 self.value = value  
 self.next = next  
 self.previous = previous  
  
 **def** \_\_str\_\_(self):  
 **return** str(self.value)  
  
  
**class** ProcessingClass(object):  
  
 **def** \_\_init\_\_(self):  
 self.file2text()  
 self.processing()  
  
 **def** file2text(self):  
 **with** open(TXT\_FILE, **'r'**) **as** stream:  
 list\_m = stream.readlines()  
 self.text = [e.replace(**"\n"**, **""**) **for** e **in** list\_m]  
  
 **def** print\_backward\_previous(self, lst):  
 **if** lst == **None**:  
 **return** head = lst  
 tail = lst.next  
 self.print\_backward\_previous(tail)  
 print(head, head.previous)  
  
 **def** print\_backward\_next(self, lst):  
 **if** lst == **None**:  
 **return** head = lst  
 tail = lst.next  
 self.print\_backward\_next(tail)  
 print(head, head.next)  
  
 **def** processing(self):  
  
 obj\_list = []  
 **for** txt **in** self.text:  
 obj\_list.append(Node(len(txt)))  
 print(**"Занесли len '"** + txt + **"' "** + str(len(txt)))  
  
 **for** i **in** range(len(obj\_list) - 1):  
 obj\_list[i].next = obj\_list[i + 1]  
 obj\_list[i + 1].previous = obj\_list[i]  
  
 **for** i **in** range(len(obj\_list)):  
 print(**"-------\n\*Элемент №"** + str(i) + **"\*"**)  
 print(**"Проход с начала в конец (next)"**)  
 self.print\_backward\_next(obj\_list[i])  
 print(**"Проход с конца в начало (previous)"**)  
 self.print\_backward\_previous(obj\_list[i])  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 ProcessingClass()

*"""  
Создайте двусвязный список групп факультета. Каждая группа представляет собой односвязн  
ый список студентов.  
"""***import** yaml  
  
  
**class** GetDataClass(object):  
 **def** \_\_init\_\_(self):  
 self.processing()  
  
 **def** processing(self):  
 **with** open(**"./d.yml"**, **'r'**) **as** stream:  
 self.d = yaml.safe\_load(stream)  
  
  
**class** Students(object):  
 *"""  
 Класс для создания коллекции студентов в виде односвязанного списка  
 Поля:  
 - value - значение объекта  
 - next - ссылка на след объект в коллекции  
 """* **def** \_\_init\_\_(self, value=**None**, next=**None**):  
 self.value = value  
 self.next = next  
  
 **def** \_\_str\_\_(self):  
 **return** str(self.value)  
  
  
**class** Groups(object):  
 *"""  
 Класс для создания коллекции групп в виде двусвязного списка  
 Поля:  
 - value - значение объекта  
 - next - ссылка на след объект в коллекции  
 - previous - ссылка на предыдущий объект коллекции  
 """* **def** \_\_init\_\_(self, value=**None**, next=**None**, previous=**None**):  
 self.value = value  
 self.next = next  
 self.previous = previous  
  
 **def** \_\_str\_\_(self):  
 **return** str(self.value)  
  
  
**class** ProcessingClass(object):  
  
 **def** \_\_init\_\_(self):  
 self.get\_data()  
 self.processing()  
  
 **def** get\_data(self):  
 settings = GetDataClass()  
 self.data = settings.d  
  
 **def** processing(self):  
 group\_list = []  
 **for** group **in** self.data:  
 student\_list = []  
 **for** student **in** self.data[group]:  
 student\_list.append(Students(student))  
 print(**"Занесли '"** + student + **"' "**)  
  
 *# Связываем между собой студентов группы* **for** i **in** range(len(student\_list) - 1):  
 student\_list[i].next = student\_list[i + 1]  
  
 group\_list.append((group, Groups(student\_list)))  
 print(**"Занесли студентов группы"**, group)  
  
 **for** i **in** range(len(group\_list) - 1):  
 group\_list[i][1].next = group\_list[i + 1][1]  
 group\_list[i + 1][1].previous = group\_list[i][1]  
  
 **for** group **in** group\_list:  
  
 print(**"\*"** \* 60)  
 print(group[1].previous, **"->"**, group[0], **"->"**, group[1].next)  
 print(**"Элементы группы:"**)  
 **for** student **in** group[1].value:  
 print(student.value, **"->"**, student.next)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 ProcessingClass()

*"""  
Дан список студентов. Элемент списка содержит фамилию, имя, отчество, год рождения, курс, номер группы,   
оценки по пяти предметам.   
Упорядочите студентов по курсу, причем студенты одного курса располагались в алфавитном порядке.  
  
Определите самого старшего студента и самого младшего студентов.  
Найдите средний балл каждой группы по каждому предмету.   
Для каждой группы найдите лучшего с точки зрения успеваемости студента.  
"""***class** MainProcessingClass(object):  
 **def** \_\_init\_\_(self):  
  
 self.student\_list = [  
 [**"Деменчук"**, **"Георгий"**, **"Максимович"**, **"1999"**, 3, **"ПИ19-4"**, [2, 5, 3, 4, 5]],  
 [**"Анатолий"**, **"Анатолий"**, **"Анатольевич"**, **"1995"**, 3, **"ПИ19-2"**, [2, 5, 3, 4, 5]],  
 [**"Прищепа"**, **"Екатерина"**, **"Кот"**, **"2001"**, 1, **"ПИ19-2"**, [5, 4, 5, 3, 5]],  
 [**"Ваня"**, **"Ваня"**, **"Иванович"**, **"2000"**, 1, **"ПИ18-2"**, [2, 3, 3, 4, 5]],  
 ]  
 self.datadict = {**"oldest"**: 9999, **"youngest"**: 0, **"data"**: {}}  
  
 self.processing()  
 self.out\_data()  
  
 **def** processing(self):  
  
 datadict = self.datadict  
 student\_list = self.student\_list  
 student\_list.sort(key=**lambda** x: (x[4], x[0]))  
  
 **for** collection **in** student\_list:  
  
 *# Проверка на лучшего студента* best\_student = {  
 **"fio"**: collection[0] + **" "** + collection[1] + **" "** + collection[2],  
 **"date"**: collection[3],  
 **"course"**: collection[4],  
 **"sum"**: sum(collection[6]),  
 }  
  
 **if** collection[5] **in** datadict[**"data"**]:  
  
 **if** datadict[**"data"**][collection[5]][**"best"**][**"sum"**] < sum(collection[6]):  
 datadict[**"data"**][collection[5]] = {**"best"**: best\_student}  
  
 datadict[**"data"**][collection[5]][**"subjects"**][**"sum"**] = [  
 sum(x) **for** x **in** zip(collection[6], datadict[**"data"**][collection[5]][**"subjects"**][**"sum"**])  
 ]  
  
 datadict[**"data"**][collection[5]][**"subjects"**][**"count"**] += 1  
  
 **else**:  
 datadict[**"data"**][collection[5]] = {**"best"**: best\_student, **"subjects"**: {**"count"**: 1, **"sum"**: collection[6]}}  
  
 **if** int(datadict[**"youngest"**]) < int(collection[3]):  
 datadict[**"youngest"**] = collection[3]  
 **if** int(datadict[**"oldest"**]) > int(collection[3]):  
 datadict[**"oldest"**] = collection[3]  
  
 **for** stat **in** datadict[**"data"**]:  
 datadict[**"data"**][stat][**"subjects"**] = [x / datadict[**"data"**][stat][**"subjects"**][**"count"**] **for** x **in** datadict[**"data"**][stat][**"subjects"**][**"sum"**]]  
  
 self.student\_list = student\_list  
 self.datadict = datadict  
  
 **def** out\_data(self):  
 d = self.datadict  
 print(**"Отсортированный словарь:\n"**, self.student\_list, **"\n"**)  
 print(**"Самый старший студент "** + d[**"oldest"**] + **"-го года\nСамый младший студент "** + d[**"youngest"**] + **"-го года"**)  
 **for** group, data **in** d[**"data"**].items():  
 print(**"\n\*Группа "** + group + **"\*"**)  
 b = data[**"best"**]  
 print(**"Луший студент группы:\n"** + b[**"fio"**] + **" "** + b[**"date"**] + **","**, b[**"course"**], **"курс,"**, **"cумма баллов"**,  
 b[**"sum"**])  
 print(**"Средние баллы группы по предметам:"**, data[**"subjects"**])  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 MainProcessingClass()