Модуль task

*"""  
№1 создайте коллекцию на свободную тему и выполните:  
№2 подсчет количества членов коллекции с помощью функции len()  
№3 проверку принадлежности элемента данной коллекции c помощью оператора in  
№4 выполните поиск подстроки  
№5 обход коллекции с применением оператора цикла  
№6 найдите максимальный, минимальный̆ элементы коллекции и сумму элементов  
№7 Найдите количество определённого пользователем элемента коллекции  
№8 выполните конвертацию типа созданной вами коллекции  
№9 выполните сортировку элементов коллекции  
№10 реализовать любые два практических задания из темы словари с применением коллекции  
№11 реализовать два задания из практической работы по спискам и кортежам с применением коллекций  
"""***from** collections.abc **import** MutableSequence  
  
**import** task\_10\_module  
**import** task\_11\_module  
  
  
**class** DEMKACollection(MutableSequence):  
 *"""  
 Кастомная коллекция  
 """* **def** \_\_init\_\_(self, data=**None**):  
  
 *# Дёргаем конструктор MutableSequence* super(DEMKACollection, self).\_\_init\_\_()  
 **if** (data **is not None**):  
 self.\_list = list(data)  
 **else**:  
 self.\_list = list()  
  
 **def** \_\_repr\_\_(self):  
 **return "<{0} {1}>"**.format(self.\_\_class\_\_.\_\_name\_\_, self.\_list)  
  
 **def** \_\_len\_\_(self):  
 **return** len(self.\_list)  
  
 **def** \_\_getitem\_\_(self, ii):  
 **return** self.\_list[ii]  
  
 **def** \_\_delitem\_\_(self, ii):  
 **del** self.\_list[ii]  
  
 **def** \_\_setitem\_\_(self, ii, val):  
 self.\_list[ii] = val  
  
 **def** \_\_str\_\_(self):  
 **return** str(self.\_list)  
  
 **def** sort(self):  
 **return** self.\_list.sort()  
  
 **def** insert(self, ii, val):  
 self.\_list.insert(ii, val)  
  
 **def** append(self, val):  
 self.insert(len(self.\_list), val)  
  
  
**class** MainClass():  
 **def** \_\_init\_\_(self):  
 self.my\_collection = DEMKACollection([1, 5, 3, 2, 5])  
 self.task\_2()  
 self.task\_3()  
 self.task\_4\_5()  
 self.task\_6()  
 self.task\_7()  
 self.task\_8()  
 self.task\_9()  
 self.task\_10()  
 self.task\_11()  
  
 **def** task\_2(self):  
 print(**"len:"**, len(self.my\_collection))  
  
 **def** task\_3(self):  
 **for** i **in** range(11):  
 flag = **True if** i **in** self.my\_collection **else False** print(i, flag)  
  
 **def** task\_4\_5(self):  
 locale\_collection = DEMKACollection(list(self.my\_collection) + [**"abc"**])  
 print(locale\_collection.\_\_getitem\_\_(0))  
 **for** i **in** range(len(locale\_collection)):  
 **if "ab" in** str(locale\_collection.\_\_getitem\_\_(i)):  
 print(**"Подстрока \"ab\" есть в элементе №"** + str(i) + **" с содержимым \""** + locale\_collection[i] + **"\""**)  
  
 **def** task\_6(self):  
 print(self.my\_collection)  
 c = self.my\_collection  
 print(**"Сумма:"**, sum(c))  
 print(**"Максимальный элемент:"**, max(c))  
 print(**"Минимальный элемент:"**, min(c))  
  
 **def** task\_7(self):  
 search\_element = int(input(**"Введите элемент для подсчёта кол-ва -> "**))  
 count = self.my\_collection.count(search\_element)  
 print(**"Количество элементов"**, search\_element, **":"**, count)  
  
 **def** task\_8(self):  
 print(type(self.my\_collection))  
 print(type(tuple(self.my\_collection)))  
 print(type(frozenset(self.my\_collection)))  
  
 **def** task\_9(self):  
 c = self.my\_collection  
 print(c)  
 c.sort()  
 print(c)  
  
 **def** task\_10(self):  
 task\_10\_module.Task10\_1()  
 task\_10\_module.Task10\_2()  
  
 **def** task\_11(self):  
 task\_11\_module.Task11\_1(DEMKACollection)  
 task\_11\_module.Task11\_2(DEMKACollection)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 MainClass()

Модуль task10\_module

**import** random  
**from** collections.abc **import** MutableMapping  
  
UNIC\_NAMES\_LIST = [  
 **"Комова Елизавета Олеговна"**,  
 **"Репенков Сергей Алексеевич"**,  
 **"Анисимов Ефим Сергеевич"**,  
 **"Борисов Никита Алексеевич"**,  
 **"Исмоилова Милена Витальевна"**,  
 **"Рогов Владимир Алексеевич"**,  
 **"Савочкин Артём Дмитриевич"**,  
 **"Кривошеина Елена Олеговна"**,  
 **"Пономарев Михаил Александрович"**,  
 **"Абдулмеджидов Мирза Мурадович"**,  
 **"Егшатян Артем Кирович"**,  
 **"Гарайшин Тамерлан Тагирович"**,  
 **"Брусова Полина Игоревна"**,  
 **"Крылова Елизавета Алексеевна"**,  
 **"Гераськина Надежда Станиславовна"**,  
 **"Гиниятуллина Эвита Маратовна"**,  
 **"Малахов Иван Петрович"**,  
 **"Жилина Алена Алексеевна"**,  
 **"Королев Илья Алексеевич"**,  
 **"Пойкалайнен Александра Максимовна"**,  
 **"Щербак Станислав Валентинович"**,  
 **"Буркина Елизавета Сергеевна"**,  
 **"Мосолова Ксения Дмитриевна"**,  
 **"Кротов Олег Валерьевич"**,  
 **"Шаповалов Сергей Александрович"**,  
 **"Прищепа Екатерина Михайловна"**,  
 **"Артемьева Дарья Сергеевна"**,  
 **"Попова Софья Александровна"**,  
 **"Башмакова Анастасия Алексеевна"**,  
 **"Корнева Татьяна Андреевна"**,  
 **"Олзошкина Янжина Владленовна"**,  
 **"Касьянов Максим Евгеньевич"**,  
 **"Олейник Анастасия Александровна"**,  
 **"Сивухов Артём Олегович"**,  
 **"Груздев Всеволод Алексеевич"**,  
 **"Буковец Данила Андреевич"**,  
 **"Зелянина Алёна Геннадьевна"**,  
 **"Мерзляков Данила Артемович"**,  
 **"Карасёв Артём Владимирович"**,  
 **"Пономаренко Александр Павлович"**,  
 **"Курносиков Кирилл Андреевич"**,  
 **"Гуриков Дмитрий Олегович"**,  
 **"Котова Екатерина Дмитриевна"**,  
 **"Лихачев Марк Игоревич"**,  
 **"Волкова Татьяна Алексеевна"**,  
 **"Марунько Анна Сергеевна"**,  
 **"Пашкевич Денис Вячеславович"**,  
 **"Маркова Ольга Алексеевна"**,  
 **"Термышева Полина Евгеньевна"**,  
 **"Василевская Лидия Игоревна"**,  
]  
  
  
**class** DEMKAdict(MutableMapping):  
 *"""  
 Кастомная коллекция от dict  
 """* **def** \_\_init\_\_(self, \*args, \*\*kwargs):  
 self.store = dict()  
 self.update(dict(\*args, \*\*kwargs))  
  
 **def** \_\_getitem\_\_(self, key):  
 **return** self.store[self.\_\_keytransform\_\_(key)]  
  
 **def** \_\_setitem\_\_(self, key, value):  
 self.store[self.\_\_keytransform\_\_(key)] = value  
  
 **def** \_\_delitem\_\_(self, key):  
 **del** self.store[self.\_\_keytransform\_\_(key)]  
  
 **def** \_\_iter\_\_(self):  
 **return** iter(self.store)  
  
 **def** \_\_len\_\_(self):  
 **return** len(self.store)  
  
 **def** \_\_keytransform\_\_(self, key):  
 **return** key  
  
  
**class** Task10\_1():  
 *"""  
 Cоздать словарь телефонного справочника. Заполнить его 50 элементами. Реализовать поиск по телефону  
 """* **def** \_\_init\_\_(self):  
 self.custom\_generator()  
 self.search()  
  
 **def** locale\_random(self, n):  
 **return** str(random.randint(10 \*\* (n - 1), (10 \*\* n) - 1))  
  
 **def** custom\_generator(self):  
 self.d = DEMKAdict()  
 print(type(self.d))  
 **for** e **in** UNIC\_NAMES\_LIST:  
 key = **"+7"** + self.locale\_random(10)  
 self.d[key] = e  
 print(**"Сгенерировали ключ "** + key)  
  
 **def** search(self):  
 input\_key = input(**"Введите номер телефона для поиска ->"**)  
 **if** input\_key **in** self.d:  
 print(**"Значение для телефона "** + input\_key + **" -> "** + self.d[input\_key])  
 **else**:  
 print(**"Введённого номера телефона нет в базе"**)  
  
  
**class** Task10\_2(Task10\_1):  
 *"""  
 Реализовать проверку на существующие записи в предыдущих заданиях с возможностью дополнения  
 """* **def** \_\_init\_\_(self):  
 self.custom\_generator()  
 self.search()  
  
 **def** search(self):  
 s = input(**"Введите телефон для поиска ->"**)  
 **if** s **in** self.d:  
 print(**"Телефон найден!\nАбонент "** + self.d[s])  
 **else**:  
 print(**"Абонент не найден, но мы его добавим в систему"**)  
 name = input(**"Введите ФИО абонента -> "**)  
 self.d[s] = name  
  
 print(**"Обновлённый словарь:"**)  
 **for** k, v **in** self.d.items():  
 print(k, v)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 Task10\_1()  
 Task10\_2()

Модуль task11\_module

**class** Task11\_1():  
 *"""  
 Список задается пользователем с клавиатуры.  
 Удаление из списка элементов, значения которых уже встречались в предыдущих элементах  
 """* **def** \_\_init\_\_(self, DEMKA\_class):  
 self.DEMKA\_class = DEMKA\_class  
 self.processing()  
  
 **def** check\_digit(self, e):  
 **try**:  
 **return** int(e)  
 **except**:  
 **return** e  
  
 **def** processing(self):  
 s = **"Введите элементы списка через запятую ->"** DEMKAlist = self.DEMKA\_class([self.check\_digit(e) **for** e **in** input(s).split(**","**)])  
 r = self.DEMKA\_class(set(DEMKAlist))  
 print(type(r))  
 print(**"Список без повторных значений: "**, r)  
  
  
**class** Task11\_2():  
 *"""  
 Дан список. После каждого элемента добавьте предшествующую ему часть списка.  
 """* **def** \_\_init\_\_(self, DEMKA\_class):  
 self.l = DEMKA\_class(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