**import** random  
**import** string  
  
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** Task1():  
 *"""  
 Создать словарь адресной книги, содержащий ФИО и адрес. Заполнить его 50 элементами, реализовать поиск по адресу  
 """* **def** \_\_init\_\_(self):  
 self.dict\_generator()  
 self.search()  
  
 **def** dict\_generator(self):  
 n = UNIC\_NAMES\_LIST  
 self.d = {}  
 **for** i **in** range(0, len(n), 2):  
 self.d[**"адрес №"** + str(i)] = [n[i], n[i + 1]]  
  
 **def** search(self):  
 s = input(**"Введите адрес для поиска ->"**)  
 **if** s **in** self.d:  
 print(**"Адрес найден!\nПроживающие по адресу:"**)  
 [print(x) **for** x **in** self.d[s]]  
 **else**:  
 **for** k, v **in** self.d.items():  
 **for** p **in** v:  
 **if** s **in** p:  
 print(**"Возможно вы имели ввиду '"** + p + **"' по '"** + k + **"'"**)  
  
  
**class** Task2():  
 *"""  
 Cоздать словарь телефонного справочника. Заполнить его 50 элементами. Реализовать поиск по телефону  
 """* **def** \_\_init\_\_(self):  
 self.dict\_generator()  
 self.search()  
  
 **def** locale\_random(self, n):  
 **return** str(random.randint(10 \*\* (n - 1), (10 \*\* n) - 1))  
  
 **def** dict\_generator(self):  
 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** Task3(object):  
 *"""  
 реализовать проверку на существующие записи в предыдущих заданиях с возможностью дополнения  
 """* **def** \_\_init\_\_(self):  
 d = {  
 **"1"**: Task1Plus,  
 **"2"**: Task2Plus,  
 }  
 e = input(**"Какой номер вы хотите дополнить ?"**)  
 **if** e **in** d:  
 d[e]()  
 **else**:  
 print(**"Такого номера не существует!"**)  
  
  
**class** Task1Plus(Task1):  
 *"""  
 Класс для дополнения задания №1  
 """* **def** \_\_init\_\_(self):  
 self.dict\_generator()  
 self.search()  
  
 **def** search(self):  
 s = input(**"Введите адрес для поиска ->"**)  
 **if** s **in** self.d:  
 print(**"Адрес найден!\nПроживающие по адресу:"**)  
 [print(x) **for** x **in** self.d[s]]  
 **else**:  
 print(**"Адрес не найден, но мы его добавим в систему"**)  
 names = input(**"Введите ФИО людей, проживающих по этому адресу через заптую -> "**)  
 self.d[s] = names.split(**","**)  
 print(**"Обновлённый словарь:"**)  
 **for** k, v **in** self.d.items():  
 print(k, v)  
  
  
**class** Task2Plus(Task2):  
 *"""  
 Класс для дополнения задания №2  
 """* **def** \_\_init\_\_(self):  
 self.dict\_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)  
  
  
**class** Task4(object):  
 *"""  
 создать словарь на свободную тему, включающий в себя кортеж в качестве ключа, реализовать поиск  
 """* **def** \_\_init\_\_(self):  
 self.d = {}  
 self.max\_values = 100  
 self.hashtable\_generator()  
 self.results\_searcher()  
  
 **def** all\_counter(self, \*t):  
 res = 0  
 **for** i **in** t:  
 res += i  
 **return** res  
  
 **def** all\_multipy(self, \*t):  
 res = 1  
 **for** i **in** t:  
 res \*= i  
 **return** res  
  
 **def** hashtable\_generator(self):  
 **for** i **in** range(self.max\_values):  
 **for** j **in** range(self.max\_values):  
 **for** k **in** range(self.max\_values):  
 self.d[(i, j, k)] = [  
 {**"multiplication"**: self.all\_multipy(i, j, k), **"sum"**: self.all\_counter(i, j, k)}]  
  
 **def** check\_digit(self, e):  
 **try**:  
 **return** int(e)  
 **except**:  
 **return** e  
  
 **def** results\_searcher(self):  
 out\_str = **"Введите 3 числа через пробел от 0 до 100 для быстрого подсчёта их суммы и произведения -> "** r = self.d[tuple([self.check\_digit(x) **for** x **in** input(out\_str).split(**" "**)])][0]  
 print(**"Произведение: "** + str(r[**"multiplication"**]) + **"\nСумма: "** + str(r[**"sum"**]))  
  
  
**class** Task5(object):  
 *"""  
 создать словарь авиарейсов, в возможностью поиска маршрута из точки А в точку В с учётом 1 пересадки  
 """* **def** \_\_init\_\_(self):  
 self.d = {  
 **"Москва"**: [**"Лондон"**, **"Владивосток"**, **"Санкт-Петербург"**],  
 **"Лондон"**: [**"Москва"**, **"Сингапур"**],  
 **"Сингапур"**: [**"Лондон"**],  
 **"Калининград"**: [**"Санкт-Петербург"**],  
 **"Санкт-Петербург"**: [**"Калининград"**, **"Москва"**],  
 **"Владивосток"**: [**"Москва"**, **"Норильск"**],  
 **"Норильск"**: [**"Владивосток"**],  
 }  
 self.flight\_generator()  
 self.search()  
  
 **def** get\_random\_flight(self):  
  
 letters = string.ascii\_uppercase  
 chars = **''**.join(random.choice(letters) **for** i **in** range(3))  
 numbers = str(random.randint(1000, 9999))  
 **return** chars + **"-"** + numbers  
  
 **def** flight\_generator(self):  
 self.flight\_d = {}  
 **for** e **in** self.d:  
 **for** city **in** self.d[e]:  
 self.flight\_d[self.get\_random\_flight()] = {**"from"**: e, **"to"**: city}  
  
 **def** search(self):  
  
 search\_flag = **False** point\_a = input(**"Введите точку А -> "**)  
 point\_b = input(**"Введите точку В -> "**)  
  
 *# Проверка на то, если ли изначальная точка в словаре* begining\_flag = **False  
 for** flight, value **in** self.flight\_d.items():  
 **if** value[**"from"**] == point\_a:  
 begining\_flag = **True  
  
 if** begining\_flag == **False**:  
 print(**"Нет указанной точки вылета в словаре!"**)  
 **return** *# Проверяем на перелёт без вложенности* **for** flight, value **in** self.flight\_d.items():  
 **if** value[**"from"**] == point\_a **and** value[**"to"**] == point\_b:  
 search\_flag = **True** print(**"План перелёта\nСуществует прямой рейс №"** + flight + **"\n"** + point\_a + **" -> "** + point\_b)  
  
 **if** search\_flag == **False**:  
 buf\_list = []  
 **for** flight, value **in** self.flight\_d.items():  
 **if** value[**"from"**] == point\_a:  
 buf\_list.append((flight, value[**"from"**], value[**"to"**]))  
  
 **for** element **in** buf\_list:  
 **for** flight, value **in** self.flight\_d.items():  
 **if** element[2] == value[**"from"**] **and** point\_b == value[**"to"**]:  
 search\_flag = **True** print(**"План перелёта:\nНа рейсе №"** + element[0] + **" "** + element[1] + **" -> "** + element[2])  
 print(**"На рейсе №"** + flight + **" "** + value[**"from"**] + **" -> "** + value[**"to"**])  
  
 **if** search\_flag == **False**:  
 print(**"По вашему запросу ничего не найдено"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 d = {  
 **"1"**: Task1,  
 **"2"**: Task2,  
 **"3"**: Task3,  
 **"4"**: Task4,  
 **"5"**: Task5  
 }  
 s = input(**"Введите номер задания -> "**)  
 **if** s **in** d:  
 d[s]()  
 **else**:  
 print(**"Такого задания нет!"**)

*"""  
Создать словарь железнодорожных сообщений с учетом более одной но менее 4 пересадок, с рекомендацией оптимального маршрута по времени  
Между переходами разница в 30 мин  
"""***import** datetime  
  
*# Время для пересадки между поездами*TIME\_WAIT = 30  
  
  
**class** UniversalClass():  
  
 @staticmethod  
 **def** get\_train\_time(\*time\_ranges):  
 *"""  
 Метод для вычисления времени прибытия поезда  
 Необходим для проверки на то, успевает ли человек на свой поезд  
 """* out\_list = []  
 **for** time\_range **in** time\_ranges:  
 time = time\_range[0]  
 begin\_time = time\_range[1]  
  
 train\_begin\_time = datetime.datetime.strptime(begin\_time, **"%d.%m.%Y %H:%M"**)  
 train\_arrive\_time = train\_begin\_time + datetime.timedelta(minutes=time)  
  
 out\_list.append((train\_begin\_time, train\_arrive\_time))  
 **return** out\_list  
  
 @staticmethod  
 **def** detect\_station\_waiting\_time(station\_name, all\_wait\_list):  
 *"""  
 Определяем, есть ли станция в словаре ожидания  
 - Если есть такая станция, то отдаём True и время ожидания  
 - Если нет, то False и None  
 """* **for** way **in** all\_wait\_list[**"ways"**]:  
 **if** station\_name == way[**"point"**]:  
 **return True**, way[**"time"**]  
 **return False**, **None** @staticmethod  
 **def** get\_ways\_string(ways\_list):  
 *"""  
 Отдаёт единую строку путей для красивого вывода в result\_outputer  
 """* buf\_list = []  
 **for** e **in** ways\_list:  
 buf\_list.extend([e[**"way\_from"**], e[**"way\_to"**]])  
 results = []  
 **for** item **in** buf\_list:  
 **if** results **and** item == results[-1]:  
 results.pop()  
 results.append(item)  
 **return " -> "**.join(results)  
  
  
**class** Task6():  
  
 **def** \_\_init\_\_(self):  
 *"""  
 Конструктор класса  
 Формирует словарь железнодорожных сообщений + вызов всех методов  
 """* self.all\_ways\_list = []  
 date = datetime.datetime.now().date().strftime(**"%d.%m.%Y "**)  
 self.d = {  
 **"Одинцово"**: [  
 {**"begin\_time"**: date + **"21:30"**, **"time\_range"**: 30, **"name"**: **"Белорусский Вокзал"**},  
 {**"begin\_time"**: date + **"14:40"**, **"time\_range"**: 3, **"name"**: **"Баковка"**},  
 {**"begin\_time"**: date + **"10:30"**, **"time\_range"**: 15, **"name"**: **"Отрадное"**},  
 ],  
  
 **"Баковка"**: [  
 {**"begin\_time"**: date + **"07:45"**, **"time\_range"**: 3, **"name"**: **"Одинцово"**},  
 {**"begin\_time"**: date + **"14:20"**, **"time\_range"**: 10, **"name"**: **"Курский Вокзал"**},  
 {**"begin\_time"**: date + **"08:10"**, **"time\_range"**: 25, **"name"**: **"Савёловский Вокзал"**},  
 ],  
 **"Отрадное"**: [  
 {**"begin\_time"**: date + **"08:10"**, **"time\_range"**: 15, **"name"**: **"Одинцово"**},  
 {**"begin\_time"**: date + **"09:40"**, **"time\_range"**: 60, **"name"**: **"Курский Вокзал"**},  
 {**"begin\_time"**: date + **"18:21"**, **"time\_range"**: 38, **"name"**: **"Савёловский Вокзал"**},  
 ],  
 **"Белорусский Вокзал"**: [  
 {**"begin\_time"**: date + **"11:15"**, **"time\_range"**: 30, **"name"**: **"Одинцово"**},  
 {**"begin\_time"**: date + **"13:50"**, **"time\_range"**: 10, **"name"**: **"Курский Вокзал"**},  
 {**"begin\_time"**: date + **"20:52"**, **"time\_range"**: 5, **"name"**: **"Савёловский Вокзал"**},  
 ],  
 **"Курский Вокзал"**: [  
 {**"begin\_time"**: date + **"17:02"**, **"time\_range"**: 10, **"name"**: **"Белорусский Вокзал"**},  
 {**"begin\_time"**: date + **"17:10"**, **"time\_range"**: 10, **"name"**: **"Баковка"**},  
 {**"begin\_time"**: date + **"15:58"**, **"time\_range"**: 60, **"name"**: **"Отрадное"**},  
 ],  
 **"Савёловский Вокзал"**: [  
 {**"begin\_time"**: date + **"11:18"**, **"time\_range"**: 5, **"name"**: **"Белорусский Вокзал"**},  
 {**"begin\_time"**: date + **"15:26"**, **"time\_range"**: 25, **"name"**: **"Баковка"**},  
 {**"begin\_time"**: date + **"19:10"**, **"time\_range"**: 38, **"name"**: **"Отрадное"**},  
 ],  
 }  
 self.way\_inputer()  
 self.way\_recognizer()  
 **if** self.all\_ways\_list:  
 self.waiting\_time\_detector()  
 self.main\_time\_detector()  
 self.result\_outputer()  
  
 **def** result\_outputer(self):  
 *"""  
 Вывод всего этого веселья на экран  
 """* r = self.final\_out\_list  
 **for** i **in** range(len(r)):  
 print(**"\n"** + **"\*"** \* 10 + **"Маршрут №"** + str(i + 1) + **"\*"** \* 10)  
 print(UniversalClass.get\_ways\_string(r[i][**"ways"**]))  
 print(**"Общее время: "**+str(r[i][**"total\_time"**]))  
 **for** way **in** r[i][**"ways"**]:  
 print(way[**"way\_from"**]+**" -> "**+way[**"way\_to"**]+**", время: "**+str(way[**"train\_time"**]))  
 **if** way[**"waiting\_time"**] **is not None**:  
 print(**"Ожидание: "**+str(way[**"waiting\_time"**]))  
  
 **def** main\_time\_detector(self):  
  
 *"""  
 Метод вычисляет общее время, которое включает в себя:  
 - Время поездки на поезда  
 - Время ожидания между поездами  
 Формуирет форматированный список рейсов, отсортированных по убыванию  
 """* all\_ways\_list = self.all\_ways\_list  
 all\_wait\_list = self.all\_wait\_list  
  
 out\_list = []  
  
 **for** i **in** range(len(all\_ways\_list)):  
  
 final\_way = {**"total\_time"**: **None**, **"ways"**: []}  
  
 points = all\_ways\_list[i][**"points"**]  
 times = all\_ways\_list[i][**"times"**]  
 total\_time = **None** little\_way\_dict = []  
 **for** j **in** range(len(times)):  
 train\_time = times[j][1] - times[j][0]  
 checked\_value = UniversalClass.detect\_station\_waiting\_time(points[j + 1], all\_wait\_list[i])  
  
 **if** checked\_value[0]:  
 little\_way\_dict.append({**"way\_from"**: points[j], **"way\_to"**: points[j + 1], **"train\_time"**: train\_time,  
 **"waiting\_time"**: checked\_value[1]})  
 train\_time += checked\_value[1]  
 **else**:  
 little\_way\_dict.append({**"way\_from"**: points[j], **"way\_to"**: points[j + 1], **"train\_time"**: train\_time,  
 **"waiting\_time"**: **None**})  
  
 **if** total\_time **is None**:  
 total\_time = train\_time  
 **else**:  
 total\_time += train\_time  
  
 final\_way[**"total\_time"**] = total\_time  
 final\_way[**"ways"**] = little\_way\_dict  
 out\_list.append(final\_way)  
 out\_list.sort(key=**lambda** x: (x[**"total\_time"**]))  
  
 self.final\_out\_list = out\_list  
  
 **def** waiting\_time\_detector(self):  
 *"""  
 Метод для определения времени ожидания след поезда между станциями + TIME\_WAIT мин на пересадку  
 """* all\_ways\_list = self.all\_ways\_list  
 all\_wait\_list = []  
  
 **for** way **in** all\_ways\_list:  
 *# Словарь для хранения маршрутов и времени для дальнейшего вывода* wait\_dict = {**"ways"**: []}  
 *# Список с парами прибытия предыдущего и отправления следующего поезда* pairs\_time = []  
 *# Т.к. есть прямой маршрут, то время одно от точки 0 до 1  
 # Индивидуальный подход кароч, если маршрут прямой* **if** len(way[**"times"**]) == 1:  
 pairs\_time.append([way[**"times"**][0][0], way[**"times"**][0][1]])  
  
 **if** pairs\_time[0][0] > pairs\_time[0][1]:  
 pairs\_time[0][1] = pairs\_time[0][1] + datetime.timedelta(days=1)  
 total\_time = pairs\_time[0][1] - pairs\_time[0][0]  
 **else**:  
 total\_time = pairs\_time[0][1] - pairs\_time[0][0]  
 wait\_dict[**"ways"**].append({**"point"**: way[**"points"**], **"time"**: total\_time})  
 **else**:  
 **for** i **in** range(1, len(way[**"times"**])):  
 pairs\_time.append([way[**"times"**][i - 1][1], way[**"times"**][i][0]])  
  
 **for** i **in** range(len(pairs\_time)):  
  
 *# Если не успевает на след поезд от премени предыдущего поезда + 30 мин, то + 24 часа* **if** pairs\_time[i][0] + datetime.timedelta(minutes=TIME\_WAIT) > pairs\_time[i][1]:  
 pairs\_time[i][1] = pairs\_time[i][1] + datetime.timedelta(days=1)  
 diff\_time = pairs\_time[i][1] - pairs\_time[i][0]  
  
 **else**:  
 diff\_time = pairs\_time[i][1] - pairs\_time[i][0]  
  
 wait\_dict[**"ways"**].append({**"point"**: way[**"points"**][i + 1], **"time"**: diff\_time})  
  
 all\_wait\_list.append(wait\_dict)  
 self.all\_wait\_list = all\_wait\_list  
  
 **def** way\_inputer(self):  
 *"""  
 Ввод данных  
 """* self.point\_a = input(**"Введите точку А -> "**)  
 self.point\_b = input(**"Введите точку В -> "**)  
  
 **def** way\_recognizer(self):  
 *"""  
 Метод для поиска маршрутов по словарю d  
 Заносит результаты вычислений в all\_ways\_list  
 """* d = self.d  
 point\_a = self.point\_a  
 point\_b = self.point\_b  
  
 all\_ways\_list = []  
  
 **if** point\_a **in** d:  
  
 **for** first\_element **in** d[point\_a]:  
  
 **if** first\_element[**"name"**] == point\_b:  
 print(**"[Вложенность 0] "** + point\_a + **" - > "** + first\_element[**"name"**])  
  
 all\_ways\_list.append(  
 dict(points=[point\_a, first\_element[**"name"**]], times=UniversalClass.get\_train\_time(  
 (first\_element[**"time\_range"**], first\_element[**"begin\_time"**])), detector\_number=0))  
  
 **if** first\_element[**"name"**] **in** d:  
 **for** second\_element **in** d[first\_element[**"name"**]]:  
 **if** second\_element[**"name"**] == point\_b:  
 print(**"[Вложенность 1] "** + point\_a + **" - > "** + first\_element[**"name"**] + **" -> "** +  
 second\_element[**"name"**])  
  
 all\_ways\_list.append(  
 dict(points=[point\_a, first\_element[**"name"**], second\_element[**"name"**]],  
 times=UniversalClass.get\_train\_time(  
 (first\_element[**"time\_range"**], first\_element[**"begin\_time"**]),  
 (second\_element[**"time\_range"**], second\_element[**"begin\_time"**])  
 ), detector\_number=1))  
  
 **if** second\_element[**"name"**] **in** d:  
 **for** third\_element **in** d[second\_element[**"name"**]]:  
 **if** third\_element[**"name"**] == point\_b:  
 print(**"[Вложенность 2] "** + point\_a + **" - > "** + first\_element[**"name"**] + **" -> "** +  
 second\_element[**"name"**] + **" -> "** + third\_element[**"name"**])  
 all\_ways\_list.append(  
 {  
 **"points"**: [point\_a, first\_element[**"name"**], second\_element[**"name"**],  
 third\_element[**"name"**]],  
 **"times"**: UniversalClass.get\_train\_time(  
 (first\_element[**"time\_range"**], first\_element[**"begin\_time"**]),  
 (second\_element[**"time\_range"**], second\_element[**"begin\_time"**]),  
 (third\_element[**"time\_range"**], third\_element[**"begin\_time"**]),  
 ),  
 **"detector\_number"**: 2,  
 })  
  
 **if** third\_element[**"name"**] **in** d:  
 **for** fourth\_element **in** d[third\_element[**"name"**]]:  
 **if** fourth\_element[**"name"**] == point\_b:  
 print(**"[Вложенность 3] "** + point\_a + **" - > "** + first\_element[  
 **"name"**] + **" -> "** + second\_element[**"name"**] + **" -> "** + third\_element[  
 **"name"**] + **" -> "** + fourth\_element[**"name"**])  
 all\_ways\_list.append(  
 {  
 **"points"**: [point\_a, first\_element[**"name"**], second\_element[**"name"**],  
 third\_element[**"name"**], fourth\_element[**"name"**]],  
 **"times"**: UniversalClass.get\_train\_time(  
 (first\_element[**"time\_range"**], first\_element[**"begin\_time"**]),  
 (second\_element[**"time\_range"**], second\_element[**"begin\_time"**]),  
 (third\_element[**"time\_range"**], third\_element[**"begin\_time"**]),  
 (fourth\_element[**"time\_range"**], fourth\_element[**"begin\_time"**]),  
 ),  
 **"detector\_number"**: 3,  
 })  
  
 self.all\_ways\_list = all\_ways\_list  
  
 **else**:  
 print(**"Нет исходной точки в начале"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 Task6()