Задание 1.

*"""  
Реализовать решение квадратного уравнения через дискриминант  
"""***import** math  
  
  
**class** DescrSolver():  
  
 **def** \_\_init\_\_(self, input\_str):  
 self.input\_str = input\_str  
 self.parse\_exp()  
 self.get\_descriminant()  
  
 **def** is\_digital(self, number):  
 **try**:  
 float(number)  
 **return True  
 except**:  
 **return False  
  
 def** parse\_exp(self):  
 input\_str = self.input\_str  
  
 *# Определяем коэффициент a* symbols = 1  
 return\_flag = **False  
 while** return\_flag == **False**:  
  
 index\_a = input\_str[:symbols]  
 **if** self.is\_digital(index\_a) == **False and** symbols != 1:  
 return\_flag = **True** symbols += 1  
  
 index\_a = input\_str[:symbols - 2]  
  
 *# Определяем коэффициент b* symbols = input\_str.rindex(**"x"**)  
 buf\_index\_b = input\_str[:symbols]  
 symbols = 1  
 return\_flag = **False  
 while** return\_flag == **False**:  
  
 index\_b = buf\_index\_b[symbols:]  
  
 **if** self.is\_digital(index\_b) == **True**:  
 return\_flag = **True** symbols += 1  
  
 *# Определяем коэффициент c* symbols = input\_str.rindex(**"="**)  
 buf\_index\_c = input\_str[:symbols]  
 symbols = 1  
 return\_flag = **False  
 while** return\_flag == **False**:  
  
 index\_c = buf\_index\_c[symbols:]  
  
 **if** self.is\_digital(index\_c) == **True**:  
 return\_flag = **True** symbols += 1  
  
 print(**"Коэффициент a ="**, index\_a)  
 print(**"Коэффициент b ="**, index\_b)  
 print(**"Коэффициент c ="**, index\_c)  
  
 self.index\_a = float(index\_a)  
 self.index\_b = float(index\_b)  
 self.index\_c = float(index\_c)  
  
 **def** get\_descriminant(self):  
 a = self.index\_a  
 b = self.index\_b  
 c = self.index\_c  
  
 D = b \*\* 2 - 4 \* a \* c  
 print(**"D ="**, D)  
  
 **if** D == 0:  
 A1 = -b / (2 \* a)  
 A2 = A1  
 print(**"\nОтвет: "**)  
 print(**"Корень А1 ="**, A1)  
 print(**"Корень A2 ="**, A2)  
  
 **elif** D > 0:  
 A1 = (-b + math.sqrt(D)) / (2 \* a)  
 A2 = (-b - math.sqrt(D)) / (2 \* a)  
 print(**"\nОтвет: "**)  
 print(**"Корень А1 ="**, A1)  
 print(**"Корень A2 ="**, A2)  
  
 **else**:  
 print(**"Нет решения, D < 0"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 input\_str = input(**"Введите квадратное выражение вида ax^2+bx+c=0 -> "**)  
 DescrSolver(input\_str)

Задание 2.

*"""  
Реализовать создание, запись, чтение и удаление файла с данными о пользователе.  
пользователь выбирает действие самостоятельно, а так же указывает путь к размещению файла.  
"""***import** os  
  
  
**class** FileProcessing():  
 **def** \_\_init\_\_(self):  
  
 select\_d = {  
 **"1"**: self.file\_add,  
 **"2"**: self.file\_remove,  
 **"3"**: self.file\_read,  
 **"4"**: self.file\_write,  
 }  
  
 self.file\_name = input(**"Введите название файла для записи -> "**)  
 input\_str = **""  
 while** input\_str != **"0"**:  
 input\_str = input(  
 **"1. Добавление файла\n2. Удаление файла\n3. Чтение из файла\n4. Запись файла\n0. Выход\n-> "**)  
 **if** input\_str **in** select\_d:  
 select\_d[input\_str]()  
 **elif** input\_str != **"0"**:  
 print(**"Нет введёного пункта меню"**)  
  
 **def** file\_add(self):  
 f = open(self.file\_name, **"w"**)  
 f.close()  
  
 **def** file\_remove(self):  
 os.remove(self.file\_name)  
  
 **def** file\_write(self):  
 *"""  
 Запись исходного выражения в файл  
 """* user\_info = input(**"Введите строку для записи -> "**)  
 **with** open(self.file\_name, **"w"**) **as** f:  
 f.write(user\_info)  
  
 **def** file\_read(self):  
 *"""  
 Чтение файла self.file\_name  
 """* **try**:  
 **with** open(self.file\_name, **"r"**) **as** f:  
 print(f.read())  
 **except** FileNotFoundError:  
 print(**"Ошибка чтения файла. Файла не существует"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 FileProcessing()

Задание 3. Модуль main.py

*#* ***TODO Рандомная генерация файла + сохранение расписания в файл****#* ***TODO Можно обратиться к полю self.content при покупке билетов. Это избавит от повторного чтения файла****"""  
Для задания 6 из предыдущей практики реализовать:  
[OK] 3.1 применение функций (не менее 5 штук)  
 3.2 расписание полетов или поездов задается файлом, свободные места в вашон  
 е или салоне указываются в файле (при продаже билета изменяете файл)  
   
 3.3 реализовать заполнение шаблонов билетов (шаблон билетов разрабатывается  
 самостоятельно) данными о рейсе, ФИО пассажира, вагоне, месте, времени и дат  
 е отправления на весь путь(с учетом пересадок)  
  
+ Редактирование и сдачу билетов. Но они есть, не забывайте что в файлах храним информацию   
о пассажирах и при этих операциях требуется изменять данные в соответствующих файлах.  
Меню в консоли на выбор действия. При сдаче указаваем процент, который удерживается с пассажира   
(делаем вывод, что стоимость билета нам тоже желательно хранить)  
  
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  
**import** random  
**from** os **import** path  
  
**import** file\_writer\_module  
**import** texttable  
**import** ticket\_module  
**import** universal\_module  
**import** waysearcher\_module  
**import** yaml  
  
  
**class** GetAllInfoClass():  
 *"""  
 Класс для отдачи всей информации, связанной с именем  
 - Время отправления и прибытия поездов  
 - Индексы поездов в общем словаре для последующего обращения  
 - Информация о билетах: вагон, место, стоимость, тип, оплата  
 """* **def** \_\_init\_\_(self, d, name):  
  
 *# Поля для обращения к ним* self.ways\_index = []  
 self.tickets = []  
  
 self.d = d  
 self.name = name  
  
 self.d\_payment\_formater = {  
 0: **"Ожидает оплаты"**,  
 1: **"Оплачено"**,  
 }  
 self.check\_reservers\_by\_name()  
  
 **def** check\_reservers\_by\_name(self):  
 *"""  
 Метод находит по ФИО брони все билеты в коллекции  
 - Возвращает время отправления и прибытия этих поездов  
 - Возвращает индексы  
 """* d = self.d  
 ways\_index = []  
 tickets = []  
  
 **for** i **in** range(len(d)):  
 train\_begin\_time = d[i][**"time\_begin"**]  
 train\_time\_finish = d[i][**"time\_finish"**]  
 **for** car **in** d[i][**"train"**]:  
 **for** place **in** d[i][**"train"**][car][**"cars"**]:  
 current\_place\_dict = d[i][**"train"**][car][**"cars"**][place]  
 **if** current\_place\_dict[**"name"**] == self.name:  
 price\_str = str(current\_place\_dict[**"price"**])  
  
 ways\_index.append(i)  
 tickets.append([  
 d[i][**"from"**] + **" -> "** + d[i][**"to"**],  
 car,  
 place,  
 price\_str + **" руб."**,  
 current\_place\_dict[**"type"**],  
 self.d\_payment\_formater[current\_place\_dict[**"payment"**]],  
 train\_begin\_time,  
 train\_time\_finish,  
 ])  
  
 self.ways\_index = ways\_index  
 self.tickets = tickets  
  
  
**class** FileGeneratorClass():  
 *"""  
 Класс-генератор свободных мест на рейсы и запись в файл  
 """* **def** \_\_init\_\_(self, all\_train\_dict, file\_name, places\_count=55, car\_count=16):  
  
 self.file\_name = file\_name  
 self.all\_train\_dict = all\_train\_dict  
 self.places\_count = places\_count  
 self.car\_count = car\_count  
 self.processing()  
 self.file\_writer()  
  
 **def** get\_time\_ranges(self, train):  
 *"""  
 Возврат диапазонов времени отправлений поезда  
 - Принимает словарь поезда  
 - Отдаёт временные промежутки отправления и прибытия поезда  
 """* train\_begin\_time = datetime.datetime.strptime(train[**"begin\_time"**], **"%d.%m.%Y %H:%M"**)  
 train\_begin\_time\_str = train\_begin\_time.strftime(**"%H:%M:%S %d/%m/%Y"**)  
 train\_finish\_time = train\_begin\_time + datetime.timedelta(minutes=train[**"time\_range"**])  
 train\_finish\_time\_str = train\_finish\_time.strftime(**"%H:%M:%S %d/%m/%Y"**)  
 **return** train\_begin\_time\_str, train\_finish\_time\_str  
  
 **def** processing(self):  
  
 print(**"Перегенерация исходного файла.."**)  
 *# Каждый путь с каждым поездом и каждым вагоном и каждым местом* sum\_train\_list = []  
 *# Все пути от self.d* all\_train\_dict = self.all\_train\_dict  
  
 **for** first\_point **in** all\_train\_dict:  
 **for** second\_point **in** all\_train\_dict[first\_point]:  
 time\_begin, time\_finish = self.get\_time\_ranges(second\_point)  
 total\_free\_places = (self.places\_count - 1) \* (self.car\_count - 1)  
 sum\_train\_list.append(  
 {  
 **"from"**: first\_point,  
 **"to"**: second\_point[**"name"**],  
 **"time\_begin"**: time\_begin,  
 **"time\_finish"**: time\_finish,  
 **"train"**: {},  
 **"info"**:  
 {  
 **"places\_free"**: total\_free\_places,  
 **"places\_count"**: self.places\_count - 1,  
 **"car\_count"**: self.car\_count - 1,  
 }  
 }  
 )  
  
 **for** i **in** range(len(sum\_train\_list)):  
  
 *# Список поезда* train\_dict = {}  
 **for** j **in** range(self.car\_count):  
  
 *# Словарь вагонов* car\_places\_dict = {**"cars"**: {}, **"places\_free"**: self.places\_count - 1}  
 place\_type\_dict = {  
 0: **"нижнее (боковое)"**,  
 1: **"верхнее (боковое)"**,  
 2: **"верхнее"**,  
 3: **"нижнее"**,  
 4: **"верхнее"**,  
 5: **"нижнее"**,  
 }  
 place\_type\_counter = 0  
 **for** k **in** range(self.places\_count):  
 price = random.randint(999, 3300)  
 place\_type = place\_type\_dict[place\_type\_counter]  
 car\_places\_dict[**"cars"**][str(k + 1)] = {**"name"**: **None**, **"price"**: price, **"type"**: place\_type,  
 **"payment"**: 0}  
 place\_type\_counter += 1  
 **if** place\_type\_counter == 6:  
 place\_type\_counter = 0  
  
 train\_dict[str(j + 1)] = car\_places\_dict  
  
 sum\_train\_list[i][**"train"**] = train\_dict  
  
 self.result = sum\_train\_list  
  
 **def** file\_writer(self):  
 **with** open(self.file\_name, **'w'**) **as** outfile:  
 yaml.safe\_dump(self.result, outfile, allow\_unicode=**True**)  
 print(**"Перегенерация завершена"**)  
  
  
**class** MainClass():  
  
 **def** \_\_init\_\_(self):  
 *"""  
 Конструктор класса  
 Формирует словарь железнодорожных сообщений + вызов всех методов  
 """* self.file\_name = **"tickets.yml"** date = datetime.datetime.now().date().strftime(**"%d.%m.%Y "**)  
 *# TODO Чтение из YAML, если date != текущей date, то регенерейт.  
 # Если файла нет, то тоже его заного создаём* 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"**},  
 ],  
 }  
  
 fileflag = path.exists(self.file\_name)  
 **if** fileflag == **False**:  
 FileGeneratorClass(self.d, self.file\_name)  
 **else**:  
 reg\_str = input(**"Хотите обнулить все забронированные места и перегенерировать исходный файл? (Да/Нет)\n-> "**)  
 **if** reg\_str == **"Y" or** reg\_str == **"y" or** reg\_str == **"Да"**:  
 FileGeneratorClass(self.d, self.file\_name)  
  
 self.mainmenu\_show()  
  
 **def** buying\_ticket\_processing(self):  
 *"""  
 Управляющая логика для покупки билетов  
 """* user\_input = input(**"\nХотите купить билет сейчас? (Да/Нет) -> "**)  
 **if** user\_input == **"Да"**:  
  
 all\_ways = self.all\_ways  
 way\_number\_input = input(**"Выберите номер маршрута для покупки -> "**)  
 **if** way\_number\_input **in** all\_ways:  
  
 self.new\_name = input(**"Введите ФИО пассажира -> "**)  
 print(**"Загрузка.."**)  
 obj = universal\_module.FileClass(self.file\_name, 2)  
 self.content = obj.get\_text()  
 processing\_ways\_list = all\_ways[way\_number\_input]  
 auto\_selecter\_input = input(  
 **"Выбор режима покупки\nХотите, чтоб система оформила наиболее дешевые билеты автоматически для каждого отдельного пути?\nЕсли нет, то вам придётся вручную делать оформление каждого отдельного билета (Да/Нет) -> "**)  
  
 **if** auto\_selecter\_input == **"Да"**:  
 automate\_flag = **True  
 elif** auto\_selecter\_input == **"Нет"**:  
 automate\_flag = **False  
 else**:  
 print(**"Некорректный ввод данных.."**)  
 **return  
  
 for** way **in** processing\_ways\_list:  
 ticket\_module.AddTicketClass(self.content, self.file\_name, self.new\_name, way[0], way[1],  
 automate\_flag)  
  
 *# print("Хотите купить билет(ы) на весь указанный путь? (Да/Нет)  
 # if "Да":  
 # print("Выбор режима покупки\nХотите, чтоб система оформила наиболее дешевые билеты автоматически для каждого отдельного пути? Если нет, то вам придётся вручную делать оформление каждого отдельного билета (Да/Нет) ->")  
 # if "Да":  
 # Автоматически поиск самых дешевых  
 # if "Нет"  
 # Пусть вбивает всё сам  
  
 # ticket\_module.AddTicketClass(self.file\_name)* **else**:  
 print(**"Введенный маршрут не найден"**)  
 **else**:  
 print(**"Хорошо, вы можете сделать это позже в пункте 'Покупка билетов'"**)  
  
 **def** mainmenu\_show(self):  
 *"""  
 Метод для вывода меню действий, связанных с 3 заданием 5 практики  
 """* input\_value = **""  
 while** input\_value != **"0"**:  
 menu\_str = **"\nВыберите действие:\n1. Покупка билетов\n2. Управление моими билетами\n0. Выход из программы\n-> "** input\_value = input(menu\_str)  
  
 **if** input\_value == **"1"**:  
 obj = waysearcher\_module.SearcherClass(self.d)  
 all\_ways = obj.ways  
 **if** all\_ways != {}:  
 *# Вынесли логику в отдельный метод* self.all\_ways = all\_ways  
 self.buying\_ticket\_processing()  
 **else**:  
 print(**"Маршруты не найдены"**)  
  
 **elif** input\_value == **"2"**:  
 print(**"Загрузка.."**)  
 obj = universal\_module.FileClass(self.file\_name, 2)  
 self.content = obj.get\_text()  
 self.new\_name = input(**"Введите ФИО пассажира -> "**)  
  
 exit\_flag = **False  
 while** exit\_flag == **False**:  
 obj\_info = GetAllInfoClass(self.content, self.new\_name)  
 ways\_indexes = obj\_info.ways\_index  
 check\_name\_tuple = obj\_info.tickets  
  
 ticket\_list = []  
 **if** check\_name\_tuple != []:  
  
 table = texttable.Texttable(180)  
 table\_list = [  
 [**"№"**, **"Поезд"**, **"№ вагона"**, **"№ места"**, **"Цена"**, **"Тип места"**, **"Статус"**, **"Время отправления"**,  
 **"Время прибытия"**], ]  
 **for** i **in** range(len(check\_name\_tuple)):  
 ticket\_list.append(str(i + 1))  
 buf\_list = [i + 1] + check\_name\_tuple[i]  
 table\_list.append(buf\_list)  
  
 table.add\_rows(table\_list)  
 print(table.draw() + **"\n"**)  
  
 reserve\_input = input(  
 **"Введите номер бронирования для управления им или 0 для выхода из подменю -> "**)  
 **if** reserve\_input == **"0"**:  
 exit\_flag = **True  
  
 elif** reserve\_input **in** ticket\_list:  
 current\_reserve = check\_name\_tuple[int(reserve\_input) - 1]  
 *# Флаг для проверки на ввод 2 пункта* payment\_show = **False  
 if** current\_reserve[5] == **"Ожидает оплаты"**:  
  
 payment\_show = **True** print(  
 **"Бронирование №"** + reserve\_input + **"\nДоступные действия:\n1. Отмена бронирования\n2. Формирование электронного билета\n3. Оплата билета"**)  
  
 **else**:  
 print(  
 **"Бронирование №"** + reserve\_input + **"\nДоступные действия:\n1. Отмена бронирования\n2. Формирование электронного билета"**)  
  
 input\_command = input(**"Введите номер действия -> "**)  
  
 way\_index = ways\_indexes[int(reserve\_input) - 1]  
 **if** input\_command == **"1"**:  
 ticket\_obj = ticket\_module.RemoveTicketClass(self.file\_name, self.content,  
 self.new\_name, way\_index,  
 current\_reserve[1], current\_reserve[2])  
 self.content = ticket\_obj.content  
  
 **elif** input\_command == **"2"**:  
  
 *# Формируем электронный билет об оплате* date\_now = datetime.datetime.now().strftime(**"%H.%M.%S %d:%m:%Y"**)  
 report\_filename = **"Электронный билет "** + self.new\_name + **" от "** + date\_now + **".pdf"** header\_str = **"Электронный билет"** main\_text\_str\_name = **"ФИО клиента: "** + self.new\_name + **"\nДата и время формирования: "** + date\_now  
  
 current\_ticket = table\_list[int(reserve\_input)]  
  
 main\_text\_str\_ticket = **"\nБилет\nСтанция и время отправления: "** + \  
 self.content[way\_index][**"from"**] + **" "** + self.content[way\_index][  
 **"time\_begin"**] + **"\n"** + **"Место и время прибытия: "** + \  
 self.content[way\_index][**"to"**] + **" "** + self.content[way\_index][  
 **"time\_finish"**] + **"\nМесто: №"** + current\_ticket[  
 3] + **" вагон "** + current\_ticket[2] + **" ["** + current\_ticket[  
 5] + **"]"** main\_text\_str\_payment = **"\nОплата\nСтоимость билета: "** + str(  
 current\_ticket[4]) + **"\n"** + **"Статус оплаты: "** + current\_ticket[6]  
 main\_text\_str = main\_text\_str\_name + **"\n"** + main\_text\_str\_ticket + **"\n"** + main\_text\_str\_payment  
 qr\_text = main\_text\_str  
 PDF\_obj = file\_writer\_module.PDFWriter(header\_str, main\_text\_str, qr\_text,  
 report\_filename)  
 **if** PDF\_obj.processed\_flag == **True**:  
 print(**"Электронный билет сформирован"**)  
  
  
 **elif** input\_command == **"3" and** payment\_show == **True**:  
 payment\_obj = ticket\_module.PaymentClass()  
  
 **if** payment\_obj.result == **True**:  
  
 self.content[way\_index][**"train"**][current\_reserve[1]][**"cars"**][current\_reserve[2]][  
 **"payment"**] = 1  
  
 *# Формируем квитанцию об оплате* date\_now = datetime.datetime.now().strftime(**"%H.%M.%S %d:%m:%Y"**)  
 report\_filename = **"Квитанция об оплате "** + self.new\_name + **" от "** + date\_now + **".pdf"** header\_str = **"Квитанция об оплате заказа от\n"** + date\_now  
 main\_text\_str\_name = **"ФИО клиента: "** + self.new\_name  
  
 current\_ticket = table\_list[int(reserve\_input)]  
 main\_text\_str\_ticket = **"\nБилет\nСтанция и время отправления: "** + \  
 self.content[way\_index][**"from"**] + **" "** + \  
 self.content[way\_index][  
 **"time\_begin"**] + **"\n"** + **"Место и время прибытия: "** + \  
 self.content[way\_index][**"to"**] + **" "** + \  
 self.content[way\_index][**"time\_finish"**] + **"\nМесто: №"** + \  
 current\_ticket[3] + **" вагон "** + current\_ticket[2] + **" ["** + \  
 current\_ticket[5] + **"]"** main\_text\_str\_payment = **"\nОплата\nСтоимость билета: "** + str(  
 current\_ticket[4]) + **"\n"** + **"Статус оплаты: ОПЛАЧЕНО"** main\_text\_str = main\_text\_str\_name + **"\n"** + main\_text\_str\_ticket + **"\n"** + main\_text\_str\_payment  
 qr\_text = main\_text\_str  
 PDF\_obj = file\_writer\_module.PDFWriter(header\_str, main\_text\_str, qr\_text,  
 report\_filename)  
 **if** PDF\_obj.processed\_flag == **True**:  
 print(**"Квитанция об оплате успешно сформирована"**)  
  
 *# Записываем все в файл* print(**"Записываем изменения.."**)  
 writer\_obj = universal\_module.FileClass(self.file\_name)  
 writer\_obj.set\_file(self.content)  
 **else**:  
 print(**"Оплата не прошла.."**)  
  
 **elif** input\_command != **"0"**:  
 print(**"Нет такого пункта в меню"**)  
  
 **else**:  
 print(**"Введенного номера бронирования не существует"**)  
 *# Проверка на то, что можно выводить в управлении* **else**:  
 print(**"Броней, связанных с введенными ФИО не найдено"**)  
 exit\_flag = **True  
  
 elif** input\_value != **"0"**:  
 print(**"Такого пункта нет в меню"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 MainClass()

Задание 3. Модуль file\_writer\_module.py

**import** qrcode  
**from** fpdf **import** FPDF  
  
  
**class** PDFWriter():  
  
 **def** \_\_init\_\_(self, header, main\_text, qr\_text, pdf\_filename\_to\_save):  
  
 self.pdf\_filename\_to\_save = pdf\_filename\_to\_save  
 self.processed\_flag = **False** self.header = header.split(**"\n"**)  
 self.main\_text = main\_text.split(**"\n"**)  
 self.qr\_text = qr\_text  
 self.processing()  
  
 **def** processing(self):  
  
 img = qrcode.make(self.qr\_text)  
 img.save(**"./img/qr.png"**)  
  
 pdf = FPDF()  
 pdf.add\_page()  
 pdf.add\_font(**'Times'**, **'B'**, **'./img/times.ttf'**, uni=**True**)  
 pdf.set\_font(**'Times'**, **'B'**, 16)  
  
 pdf.image(**'./img/logo.gif'**, 10, 10, 33)  
 pdf.image(**'./img/qr.png'**, 160, 8, 45)  
 pdf.image(**'./img/footer.png'**, 10, 220, 200)  
  
 **for** header **in** self.header:  
 pdf.cell(75)  
 pdf.cell(30, 10, header, 0, 1, **"C"**)  
  
 pdf.ln(20)  
 pdf.add\_font(**'Times'**, **''**, **'./img/times.ttf'**, uni=**True**)  
 pdf.set\_font(**'Times'**, **''**, 14)  
  
 **for** t **in** self.main\_text:  
 pdf.cell(0, 10, t, 0, 1)  
  
 pdf.output(**'./pdf/'** + self.pdf\_filename\_to\_save, **'F'**)  
 self.processed\_flag = **True**

Задание 3. Модуль ticket\_module.py

*"""  
Модуль для работы с билетами  
- Заказ, бронирование, покупка  
- Отмена заказа  
"""***import** datetime  
**import** time  
  
**import** file\_writer\_module  
**import** texttable  
**import** universal\_module  
  
  
**class** PaymentClass():  
  
 **def** \_\_init\_\_(self):  
 self.payment\_processing()  
 self.result = **True  
  
 def** payment\_processing(self):  
 print(**"Оплата..\n"**)  
 **for** i **in** range(3):  
 time.sleep(1)  
 print(**"."**)  
 print(**"Оплата прошла успешно!"**)  
  
  
**def** get\_min\_max\_price\_of\_car(car):  
 max\_price = car[**"cars"**][**"1"**][**"price"**]  
 min\_price = max\_price  
  
 **for** place **in** car[**"cars"**]:  
 locale\_price = car[**"cars"**][place][**"price"**]  
 **if** locale\_price > max\_price:  
 max\_price = locale\_price  
 **if** locale\_price < min\_price:  
 min\_price = locale\_price  
  
 **return** max\_price, min\_price  
  
  
**class** AddTicketClass():  
 *"""  
 Класс для добавления билетов. Выщывается как при прямых рейсов, так и для ресов с пересадками  
 """* **def** \_\_init\_\_(self, content, file\_name, name, way\_from, way\_to, automate\_flag=**False**):  
  
 self.file\_name = file\_name  
 self.way\_from = way\_from  
 self.way\_to = way\_to  
 self.name = name  
  
 self.content = content  
  
 **if** automate\_flag == **True**:  
 self.auto\_components\_reserve()  
 **else**:  
 self.mechanical\_components\_reserve()  
  
 **def** auto\_components\_reserve(self):  
 way\_index = 0  
 price\_dict = {}  
 content = self.content  
 **for** i **in** range(len(content)):  
 **if** content[i][**"from"**] == self.way\_from **and** content[i][**"to"**] == self.way\_to:  
 way\_index = i  
 **for** car **in** content[i][**"train"**]:  
 **for** place **in** content[i][**"train"**][car][**"cars"**]:  
 **if** content[i][**"train"**][car][**"cars"**][place][**"name"**] == **None**:  
 locale\_price = content[i][**"train"**][car][**"cars"**][place][**"price"**]  
 price\_dict[(car, place)] = locale\_price  
  
 min\_price = price\_dict[(**'1'**, **'1'**)]  
 min\_key = (**'1'**, **'1'**)  
 **for** k, v **in** price\_dict.items():  
 **if** v < min\_price:  
 min\_price = v  
 min\_key = k  
  
 way\_index  
 price = min\_price  
 car, place = min\_key  
  
 selected\_train = content[way\_index]  
 train\_places\_free = selected\_train[**"info"**][**"places\_free"**] - 1  
 car\_places\_free = selected\_train[**"train"**][car][**"places\_free"**] - 1  
 print(  
 **"Зарезервировали "** + place + **" место в вагоне "** + car + **" по маршруту "** + selected\_train[**"from"**] + **" -> "** +  
 selected\_train[**"to"**] + **" за "** + str(price) + **" руб"**)  
 print(**"Отправление в "** + selected\_train[**"time\_begin"**] + **", прибытие - "** + selected\_train[**"time\_finish"**])  
  
 content[way\_index][**"info"**][**"places\_free"**] = train\_places\_free  
 content[way\_index][**"train"**][car][**"places\_free"**] = car\_places\_free  
 content[way\_index][**"train"**][car][**"cars"**][place][**"name"**] = self.name  
 *# Записываем все в файл* writer\_obj = universal\_module.FileClass(self.file\_name)  
 writer\_obj.set\_file(content)  
 self.content = content  
  
 **def** mechanical\_components\_reserve(self):  
 *"""  
 Выбор вагона для брони в поезде  
 """* exit\_flag = **False  
 while** exit\_flag == **False**:  
 content = self.content  
  
 **for** i **in** range(len(content)):  
 **if** content[i][**"from"**] == self.way\_from **and** content[i][**"to"**] == self.way\_to:  
 print(**"Поезд найден "** + content[i][**"from"**] + **" -> "** + content[i][**"to"**])  
 print(**"Всего в поезде "** + str(content[i][**"info"**][**"car\_count"**]) + **" вагонов и "** + str(  
 content[i][**"info"**][**"places\_free"**]) + **" свободных мест\nМест по вагонам:"**)  
 table = texttable.Texttable(180)  
 table\_list = [[**"Вагон"**, **"Мест свободно"**, **"Диапазон цен на места"**], ]  
  
 buf\_car\_list = []  
  
 **for** car **in** content[i][**"train"**]:  
 price\_min, price\_max = get\_min\_max\_price\_of\_car(content[i][**"train"**][car])  
 price\_range = str(price\_max) + **" - "** + str(price\_min) + **" руб."** table\_list.append([car, content[i][**"train"**][car][**"places\_free"**], price\_range])  
 buf\_car\_list.append(car)  
  
 *# Вывод* table.add\_rows(table\_list)  
 print(table.draw() + **"\n"**)  
  
 selected\_car = input(**"Выберите вагон (0 для выхода из подменю) -> "**)  
 **if** selected\_car == **"0"**:  
 exit\_flag = **True  
 elif** selected\_car **in** buf\_car\_list:  
 self.place\_searcher(i, selected\_car)  
 **else**:  
 print(**"Нет такого вагона, выход из подпрограммы.."**)  
 exit\_flag = **True** self.content = content  
  
 **def** place\_searcher(self, way, selected\_car):  
 *"""  
 Выбор места в вагоне для брони  
 """* exit\_flag = **False  
  
 while** exit\_flag == **False**:  
 content = self.content  
 table = texttable.Texttable(180)  
 print(**"Места в вагоне:"**)  
 table\_list = [[**"№"**, **"Статус"**, **"Цена"**, **"Тип"**], ]  
 buf\_place\_list = []  
 **for** place **in** content[way][**"train"**][selected\_car][**"cars"**]:  
 buf\_place\_list.append(place)  
 locale\_place = content[way][**"train"**][selected\_car][**"cars"**][place]  
  
 reserved\_type = **"свободно"  
 if** locale\_place[**"name"**] != **None**:  
 reserved\_type = **"забронировано"** price = str(locale\_place[**"price"**]) + **" руб."** table\_list.append([place, reserved\_type, price, locale\_place[**"type"**]])  
  
 table.add\_rows(table\_list)  
 print(table.draw() + **"\n"**)  
  
 selected\_place = input(**"Введите номер места для бронирования (0 для выхода из подменю) -> "**)  
 **if** selected\_place == **"0"**:  
 exit\_flag = **True  
 elif** selected\_place **in** buf\_place\_list:  
 **if** content[way][**"train"**][selected\_car][**"cars"**][selected\_place][**"name"**] == **None**:  
 question\_string = **"Вы действительно хотите забронировать место №"** + selected\_place + **" в вагоне "** + selected\_car + **" поезда "** + self.way\_from + **" - "** + self.way\_to + **" на имя '"** + self.name + **"'? (Да/Нет) -> "** user\_reply = input(question\_string)  
 **if** user\_reply == **"Да" or** user\_reply == **"Y" or** user\_reply == **"y"**:  
  
 *# Успешно меняем текущий словарь* content[way][**"train"**][selected\_car][**"cars"**][selected\_place][**"name"**] = self.name  
 question\_input = input(**"Место успешно зарезервировано\nОплатить его сейчас? Да/Нет -> "**)  
 **if** question\_input == **"Да"**:  
 pay\_obj = PaymentClass()  
 **if** pay\_obj.result == **True**:  
 *# Устанавливаем флаг того, что мы всё оплатили* content[way][**"train"**][selected\_car][**"cars"**][selected\_place][**"payment"**] = 1  
  
 *# Формируем квитанцию об оплате* date\_now = datetime.datetime.now().strftime(**"%H.%M.%S %d:%m:%Y"**)  
 report\_filename = **"Квитанция об оплате "** + self.name + **" от "** + date\_now + **".pdf"** header\_str = **"Квитанция об оплате заказа от\n"** + date\_now  
 main\_text\_str\_name = **"ФИО клиента: "** + self.name  
  
 buf\_place = content[way][**"train"**][selected\_car][**"cars"**][selected\_place]  
  
 main\_text\_str\_ticket = **"\nБилет\nСтанция и время отправления: "** + content[way][  
 **"from"**] + **" "** + content[way][**"time\_begin"**] + **"\n"** + **"Место и время прибытия: "** + \  
 content[way][**"to"**] + **" "** + content[way][  
 **"time\_finish"**] + **"\nМесто: №"** + selected\_place + **" вагон "** + selected\_car + **" ["** + \  
 buf\_place[**"type"**] + **"]"** main\_text\_str\_payment = **"\nОплата\nСтоимость билета: "** + str(  
 buf\_place[**"price"**]) + **" руб. \n"** + **"Статус оплаты: ОПЛАЧЕНО"** main\_text\_str = main\_text\_str\_name + **"\n"** + main\_text\_str\_ticket + **"\n"** + main\_text\_str\_payment  
 qr\_text = main\_text\_str  
 PDF\_obj = file\_writer\_module.PDFWriter(header\_str, main\_text\_str, qr\_text,  
 report\_filename)  
 **if** PDF\_obj.processed\_flag == **True**:  
 print(**"Квитанция об оплате успешно сформирована"**)  
  
 **elif** question\_input == **"Нет"**:  
 print(**"Хорошо, оплатить билет вы можете позже в пункте 'Управление моими билетами'"**)  
  
 *# Обновляем количество мест* train\_places\_free = content[way][**"info"**][**"places\_free"**] - 1  
 car\_places\_free = content[way][**"train"**][selected\_car][**"places\_free"**] - 1  
  
 content[way][**"info"**][**"places\_free"**] = train\_places\_free  
 content[way][**"train"**][selected\_car][**"places\_free"**] = car\_places\_free  
  
 *# Записываем все в файл* writer\_obj = universal\_module.FileClass(self.file\_name)  
 writer\_obj.set\_file(content)  
  
 **else**:  
 print(**"Введенное место уже забронировано"**)  
 **else**:  
 print(**"Введенное место не найдено, выход из подпрограммы.."**)  
 exit\_flag = **True** self.content = content  
  
  
**class** RemoveTicketClass():  
 *"""  
 Класс для отмены бронирования билетов  
 """* **def** \_\_init\_\_(self, file\_name, content, name, way, car, place):  
 self.file\_name = file\_name  
 self.content = content  
 self.name = name  
 self.way = way  
 self.car = car  
 self.place = place  
 self.refund\_percent = 5  
  
 self.ticket\_remover()  
  
 **def** ticket\_remover(self):  
  
 print(**"Отмена бронирования.."**)  
 percent = self.refund\_percent  
 content = self.content  
 way = self.way  
 car = self.car  
 place = self.place  
  
 way\_str = content[way][**"from"**] + **" -> "** + content[way][**"to"**]  
 selected\_place = content[way][**"train"**][car][**"cars"**][place]  
 percent\_price = (selected\_place[**"price"**] / 100) \* percent  
 refound\_price = selected\_place[**"price"**] - percent\_price  
  
 refound\_str = **"Вам вернется "** + str(refound\_price) + **" руб. Сервис удержит комиссию в виде "** + str(  
 percent\_price) + **" руб."** place\_str = **"\nВы действительно хотите отменить бронирование на "** + place + **" место "** + car + **" вагона поезда "** + way\_str + **" ? (Да/Нет) -> "** confirm\_input = input(refound\_str + place\_str)  
 **if** confirm\_input == **"Да" or** confirm\_input == **"Y" or** confirm\_input == **"y"**:  
  
 selected\_place[**"payment"**] = 0  
 selected\_place[**"name"**] = **None** *# Обновляем количество мест* train\_places\_free = content[way][**"info"**][**"places\_free"**] + 1  
 car\_places\_free = content[way][**"train"**][car][**"places\_free"**] + 1  
  
 content[way][**"info"**][**"places\_free"**] = train\_places\_free  
 content[way][**"train"**][car][**"places\_free"**] = car\_places\_free  
  
 date\_now = datetime.datetime.now().strftime(**"%H.%M.%S %d:%m:%Y"**)  
 report\_filename = **"Возврат "** + self.name + **" от "** + date\_now + **".pdf"** header\_str = **"Документ об оформлении возврата срeдств от\n"** + date\_now  
  
 main\_text\_str\_name = **"ФИО клиента: "** + self.name  
 main\_text\_str\_ticket = **"\nБилет\nСтанция и время отправления: "** + content[way][**"from"**] + **" "** + content[way][  
 **"time\_begin"**] + **"\n"** + **"Место и время прибытия: "** + content[way][**"to"**] + **" "** + content[way][  
 **"time\_finish"**] + **"\nМесто: №"** + place + **" вагон "** + car + **" ["** + selected\_place[  
 **"type"**] + **"]"** main\_text\_str\_payment = **"\nВозврат средств\nСтоимость билета: "** + str(  
 selected\_place[**"price"**]) + **" руб. \n"** + **"Стоимость комиссии: "** + str(  
 percent\_price) + **" руб. \nВозвращённая сумма: "** + str(refound\_price) + **" руб."** main\_text\_str = main\_text\_str\_name + **"\n"** + main\_text\_str\_ticket + **"\n"** + main\_text\_str\_payment  
 qr\_text = main\_text\_str  
  
 PDF\_obj = file\_writer\_module.PDFWriter(header\_str, main\_text\_str, qr\_text, report\_filename)  
 **if** PDF\_obj.processed\_flag == **True**:  
 print(**"Документ о возврате успешно сформирован"**)  
  
 content[way][**"train"**][car][**"cars"**][place] = selected\_place  
 print(**"Запиcь изменений.."**)  
 writer\_obj = universal\_module.FileClass(self.file\_name)  
 writer\_obj.set\_file(content)  
 self.content = content  
 print(**"Успешно"**)  
  
 **else**:  
 print(**"Хорошо, все оставили без изменений"**)

Задание 3. Модуль universal\_module.py

**import** datetime  
  
**import** yaml  
  
  
**class** FileClass():  
 *"""  
 Класс для работы с файлом file\_name, который передаётся в конструктр из main  
 - Записывает данные в файл  
 - Читает данные из файла   
 """* **def** \_\_init\_\_(self, file\_name, method=1):  
 self.file\_name = file\_name  
  
 **if** method == 2:  
 self.read\_file()  
  
 **def** set\_file(self, content):  
 **with** open(self.file\_name, **'w'**) **as** outfile:  
 yaml.safe\_dump(content, outfile, allow\_unicode=**True**)  
  
 **def** read\_file(self):  
 **with** open(self.file\_name, **'r'**) **as** outfile:  
 self.content = yaml.safe\_load(outfile)  
  
 **def** get\_text(self):  
 **return** self.content  
  
  
**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)

Задание 3. Модуль waysearcher\_module.py

*"""  
Модуль для поиска наиболее рациональных путей по веремени.  
Неокогда основной процессинг 4 практики 6 задания  
"""***import** datetime  
  
**import** universal\_module  
  
  
**class** SearcherClass():  
 **def** \_\_init\_\_(self, d):  
 self.d = d  
 self.TIME\_WAIT = 30  
 self.all\_ways\_list = []  
 self.way\_inputer()  
 self.way\_recognizer()  
  
 *# Поля для последующего обращения с main* self.ways = {}  
  
 **if** self.all\_ways\_list:  
 self.waiting\_time\_detector()  
 self.main\_time\_detector()  
 self.result\_outputer()  
 self.ways\_outer()  
  
 **def** ways\_outer(self):  
 *"""  
 Метод необходим для формирования словаря self.ways со всеми путями  
 """* locale\_ways = {}  
 r = self.final\_out\_list  
 **for** i **in** range(len(r)):  
 buf\_list = []  
 **for** e **in** r[i][**"ways"**]:  
 buf\_list.append([e[**"way\_from"**], e[**"way\_to"**]])  
  
 locale\_ways[str(i + 1)] = buf\_list  
  
 self.ways = locale\_ways  
  
 **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=self.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=universal\_module.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=universal\_module.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"**: universal\_module.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"**: universal\_module.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"**)  
  
 **def** result\_outputer(self):  
 *"""  
 Вывод всех результатов на экран + отдача значений в self.ways  
 """* r = self.final\_out\_list  
 **for** i **in** range(len(r)):  
 print(**"\n"** + **"\*"** \* 10 + **"Маршрут №"** + str(i + 1) + **"\*"** \* 10)  
 print(universal\_module.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 = universal\_module.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