**МИНИCTEPCTBO НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное**

**образовательное учреждение высшего образования**

**«Северо-Кавказский федеральный университет»**

**Кафедра инфокоммуникаций**

**Отчет по лабораторной работе №2.17**

**по дисциплине «Основы программной инженерии»**

**Выполнил студент группы ПИЖ-б-о-20-1**

**Примаков В. Д. « »\_\_\_\_\_\_\_\_\_20\_\_г.**

**Подпись студента\_\_\_\_\_\_\_\_\_\_\_**

**Работа защищена « »\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.**

**Проверил Воронкин Р.А. \_\_\_\_\_\_\_\_**

**(подпись)**

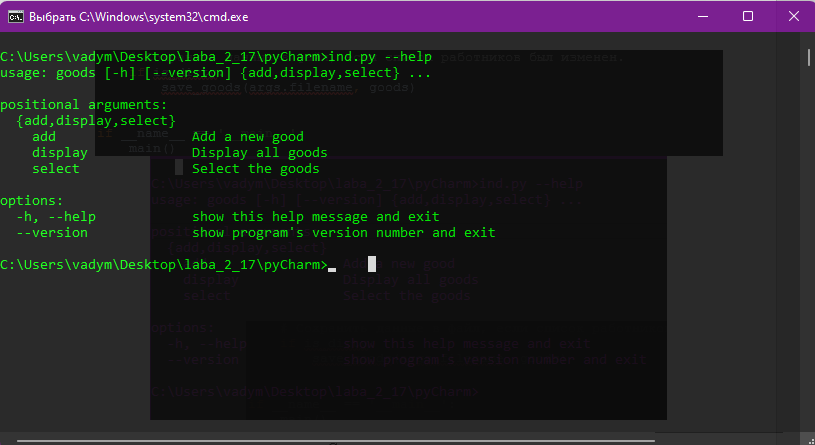
**ВЫПОЛНЕНИЕ**

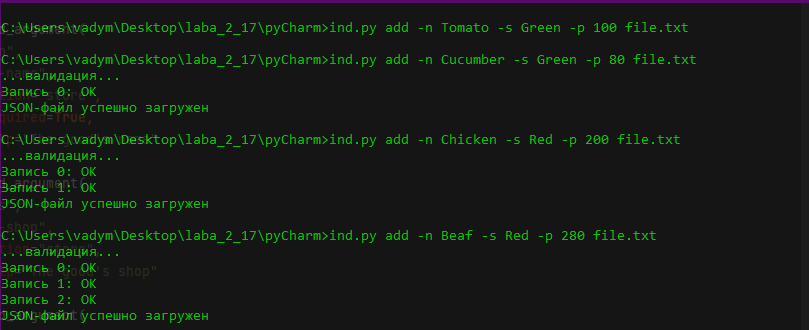
Пример 1:

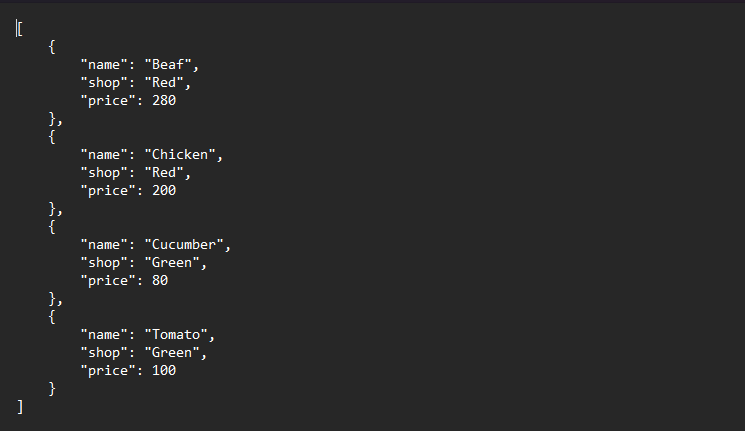
#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
  
import argparse  
import json  
import os.path  
import sys  
from datetime import date  
  
  
def add\_worker(staff, name, post, year):  
 *"""  
 Добавить данные о работнике.  
 """* staff.append(  
 {  
 "name": name,  
 "post": post,  
 "year": year  
 }  
 )  
  
 return staff  
  
  
def display\_workers(staff):  
 *"""  
 Отобразить список работников.  
 """* # Проверить, что список работников не пуст.  
 if staff:  
 # Заголовок таблицы.  
 line = '+-{}-+-{}-+-{}-+-{}-+'.format(  
 '-' \* 4,  
 '-' \* 30,  
 '-' \* 20,  
 '-' \* 8  
 )  
 print(line)  
 print(  
 '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(  
 "№",  
 "Ф.И.О.",  
 "Должность",  
 "Год"  
 )  
 )  
 print(line)  
  
 # Вывести данные о всех сотрудниках.  
 for idx, worker in enumerate(staff, 1):  
 print(  
 '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(  
 idx,  
 worker.get('name', ''),  
 worker.get('post', ''),  
 worker.get('year', 0)  
 )  
 )  
 print(line)  
  
 else:  
 print("Список работников пуст.")  
  
  
def select\_workers(staff, period):  
 *"""  
 Выбрать работников с заданным стажем.  
 """* # Получить текущую дату.  
 today = date.today()  
  
 # Сформировать список работников.  
 result = []  
 for employee in staff:  
 if today.year - employee.get('year', today.year) >= period:  
 result.append(employee)  
  
 # Возвратить список выбранных работников.  
 return result  
  
  
def save\_workers(file\_name, staff):  
 *"""  
 Сохранить всех работников в файл JSON.  
 """* # Открыть файл с заданным именем для записи.  
 with open(file\_name, "w", encoding="utf-8") as fout:  
 # Выполнить сериализацию данных в формат JSON.  
 # Для поддержки кирилицы установим ensure\_ascii=False  
 json.dump(staff, fout, ensure\_ascii=False, indent=4)  
  
  
def load\_workers(file\_name):  
 *"""  
 Загрузить всех работников из файла JSON.  
 """* # Открыть файл с заданным именем для чтения.  
 with open(file\_name, "r", encoding="utf-8") as fin:  
 return json.load(fin)  
  
  
def main(command\_line=None):  
 # Создать родительский парсер для определения имени файла.  
 file\_parser = argparse.ArgumentParser(add\_help=False)  
 file\_parser.add\_argument(  
 "filename",  
 action="store",  
 help="The data file name"  
 )  
  
 # Создать основной парсер командной строки.  
 parser = argparse.ArgumentParser("workers")  
 parser.add\_argument(  
 "--version",  
 action="version",  
 version="%(prog)s 0.1.0"  
 )  
  
 subparsers = parser.add\_subparsers(dest="command")  
  
 # Создать субпарсер для добавления работника.  
 add = subparsers.add\_parser(  
 "add",  
 parents=[file\_parser],  
 help="Add a new worker"  
 )  
 add.add\_argument(  
 "-n",  
 "--name",  
 action="store",  
 required=True,  
 help="The worker's name"  
 )  
 add.add\_argument(  
 "-p",  
 "--post",  
 action="store",  
 help="The worker's post"  
 )  
 add.add\_argument(  
 "-y",  
 "--year",  
 action="store",  
 type=int,  
 required=True,  
 help="The year of hiring"  
 )  
  
 # Создать субпарсер для отображения всех работников.  
 \_ = subparsers.add\_parser(  
 "display",  
 parents=[file\_parser],  
 help="Display all workers"  
 )  
  
 # Создать субпарсер для выбора работников.  
 select = subparsers.add\_parser(  
 "select",  
 parents=[file\_parser],  
 help="Select the workers"  
 )  
  
 select.add\_argument(  
 "-P",  
 "--period",  
 action="store",  
 type=int,  
 required=True,  
 help="The required period"  
 )  
  
 # Выполнить разбор аргументов командной строки.  
 args = parser.parse\_args()  
  
 # Загрузить всех работников из файла, если файл существует.  
 is\_dirty = False  
 if os.path.exists(args.filename):  
 workers = load\_workers(args.filename)  
 else:  
 workers = []  
  
 # Добавить работника.  
 if args.command == "add":  
 workers = add\_worker(  
 workers,  
 args.name,  
 args.post,  
 args.year  
 )  
 is\_dirty = True  
  
 # Отобразить всех работников.  
 elif args.command == "display":  
 display\_workers(workers)  
  
 # Выбрать требуемых работников.  
 elif args.command == "select":  
 selected = select\_workers(workers, args.period)  
 display\_workers(selected)  
  
 # Сохранить данные в файл, если список работников был изменен.  
 if is\_dirty:  
 save\_workers(args.filename, workers)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

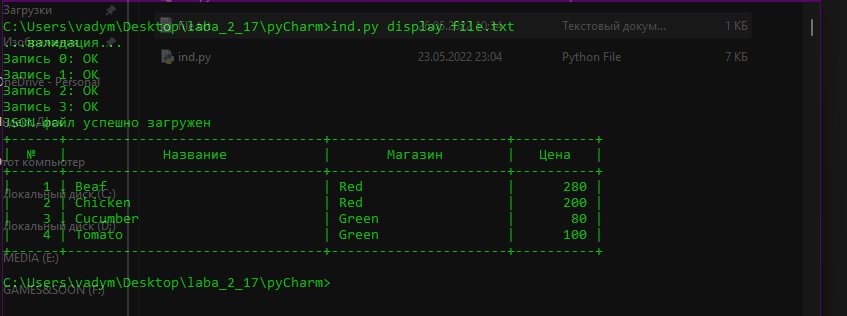
Индивидуальное задание:

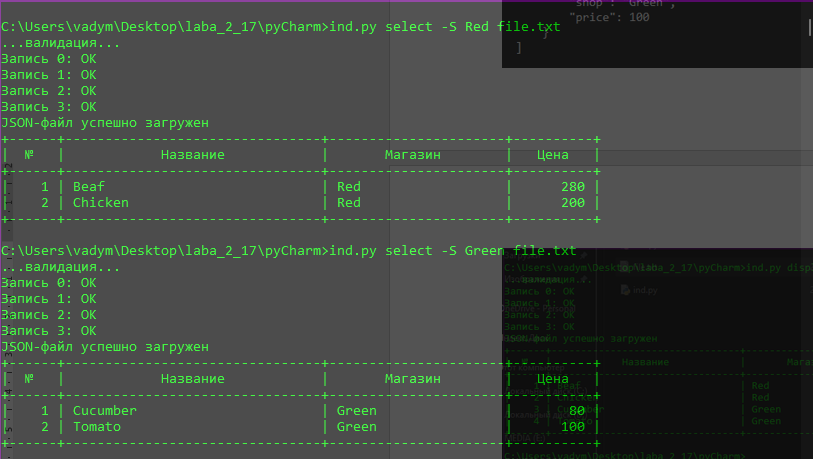
#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
  
import argparse  
import os  
import sys  
import json  
import jsonschema  
from jsonschema import validate  
  
schema = {  
 "type" : "object",  
 "properties" : {  
 "price" : {"type" : "number"},  
 "name" : {"type" : "string"},  
 "shop" : {"type" : "string"}  
 },  
}  
  
  
def add\_goods(goods, args):  
 # Создать словарь.  
 good = {  
 'name': args.name,  
 'shop': args.shop,  
 'price': args.price,  
 }  
 # Добавить словарь в список.  
 goods.append(good)  
 # Отсортировать список в случае необходимости.  
 if len(goods) > 1:  
 goods.sort(key=lambda item: item.get('name', ''))  
  
  
def display\_goods(goods):  
 *"""  
 Отобразить список товаров.  
 """* # Проверить, что список товаров не пуст.  
 if goods:  
 # Заголовок таблицы.  
 line = '+-{}-+-{}-+-{}-+-{}-+'.format(  
 '-' \* 4,  
 '-' \* 30,  
 '-' \* 20,  
 '-' \* 8  
 )  
 print(line)  
 print(  
 '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(  
 "№",  
 "Название",  
 "Магазин",  
 "Цена"  
 )  
 )  
 print(line)  
 # Вывести данные о всех товарах.  
 for idx, good in enumerate(goods, 1):  
 print(  
 '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(  
 idx,  
 good.get('name', ''),  
 good.get('shop', ''),  
 good.get('price', 0)  
 )  
 )  
 print(line)  
  
 else:  
 print("Список товаров пуст.")  
  
  
def select\_goods(goods, shop):  
 *"""  
 Выбрать товары магазина.  
 """* # Счетчик записей.  
 count = 0  
  
 # Сформировать список товаров.  
 result = []  
  
 for good in goods:  
 if shop == good.get('shop', shop):  
 count += 1  
 result.append(good)  
  
 # Проверка на отсутствие товаров или выбранного магазина.  
 if count == 0:  
 print("Такого магазина не существует либо нет товаров.")  
 else:  
 # Возвратить список выбранных товаров.  
 return result  
  
  
def save\_goods(file\_name, goods):  
 *"""  
 Сохранить все магазины в файл JSON.  
 """* # Открыть файл с заданным именем для записи.  
 with open(file\_name, "w", encoding="utf-8") as fout:  
 # Выполнить сериализацию данных в формат JSON.  
 # Для поддержки кирилицы установим ensure\_ascii=False  
 json.dump(goods, fout, ensure\_ascii=False, indent=4)  
  
  
def load\_goods(file\_name):  
 *"""  
 Загрузить все магазины из файла JSON.  
 """* # Открыть файл с заданным именем для чтения.  
 with open(file\_name, "r", encoding="utf-8") as fin:  
 f = json.load(fin)  
 err\_count = 0  
 print("...валидация...")  
 for idx, item in enumerate(f):  
 try:  
 validate(item, schema)  
 sys.stdout.write("Запись {}: OK\n".format(idx))  
 except jsonschema.exceptions.ValidationError as ve:  
 sys.stderr.write("Запись {}: ОШИБКА\n".format(idx))  
 sys.stderr.write(str(ve) + "\n")  
 err\_count += 1  
 if err\_count > 0:  
 print("JSON-файл не прошел валидацию.\nФайл не будет загружен.")  
 else:  
 print("JSON-файл успешно загружен")  
 return f  
  
  
def main():  
 # Создать родительский парсер для определения имени файла.  
 file\_parser = argparse.ArgumentParser(add\_help=False)  
 file\_parser.add\_argument(  
 "filename",  
 action="store",  
 help="The data file name"  
 )  
  
 # Создать основной парсер командной строки.  
 parser = argparse.ArgumentParser("goods")  
 parser.add\_argument(  
 "--version",  
 action="version",  
 version="%(prog)s 0.1.0"  
 )  
  
 subparsers = parser.add\_subparsers(dest="command")  
  
 # Создать субпарсер для добавления товара.  
 add = subparsers.add\_parser(  
 "add",  
 parents=[file\_parser],  
 help="Add a new good"  
 )  
 add.add\_argument(  
 "-n",  
 "--name",  
 action="store",  
 required=True,  
 help="The good's name"  
 )  
 add.add\_argument(  
 "-s",  
 "--shop",  
 action="store",  
 help="The good's shop"  
 )  
 add.add\_argument(  
 "-p",  
 "--price",  
 action="store",  
 type=int,  
 required=True,  
 help="Price of the good"  
 )  
  
 # Создать субпарсер для отображения всех товаров.  
 \_ = subparsers.add\_parser(  
 "display",  
 parents=[file\_parser],  
 help="Display all goods"  
 )  
  
 # Создать субпарсер для выбора товаров.  
 select = subparsers.add\_parser(  
 "select",  
 parents=[file\_parser],  
 help="Select the goods"  
 )  
  
 select.add\_argument(  
 "-S",  
 "--shop\_select",  
 action="store",  
 required=True,  
 help="The required shop"  
 )  
  
 # Выполнить разбор аргументов командной строки.  
 args = parser.parse\_args()  
  
 # Загрузить всех работников из файла, если файл существует.  
 is\_dirty = False  
 if os.path.exists(args.filename):  
 goods = load\_goods(args.filename)  
 else:  
 goods = []  
  
 # Добавить работника.  
 if args.command == "add":  
 add\_goods(goods, args)  
 is\_dirty = True  
  
 # Отобразить всех работников.  
 elif args.command == "display":  
 display\_goods(goods)  
  
 # Выбрать требуемых работников.  
 elif args.command == "select":  
 selected = select\_goods(goods, args.shop\_select)  
 display\_goods(selected)  
  
 # Сохранить данные в файл, если список работников был изменен.  
 if is\_dirty:  
 save\_goods(args.filename, goods)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()











[Ссылки на репозитории](https://github.com/surai5a/FSE)

GitHub - <https://github.com/surai5a/laba_2_17>

Ответы на контрольные вопросы

1. Применяется в веб-приложениях как для обмена данными между браузером и сервером (AJAX), так и между серверами (программные HTTP-сопряжения).
2. Значение JSON может быть одним из шести типов данных: строкой, числом, объектом, массивом, булевым значением или null.
3. При работе с большим набором данных предпочтительней использовать запись в многострочном формате с использованием пробелов между парами.
4. JSON5 — предложенное расширение формата json в соответствии с синтаксисом ECMAScript 5, вызванное тем, что json используется не только для общения между программами, но и создаётся/редактируется вручную. Файл JSON5 всегда является корректным кодом ECMAScript 5. JSON5 обратно совместим с JSON. Для некоторых языков программирования уже существуют парсеры json5.
5. Для работы с JSON5 в python можно использовать библиотеку pyjson5.
6. Метод dump библиотеки json для python позволяет конвертировать python-объект в json и записать в файл. (dumps – для записи в строку)
7. Dump – запись в многострочном режиме, dump – запись в строку.
8. Метод load загружает данные json из многострочного режима, loads – из строки.
9. При записи объекта, содержащего кириллицу в файл json нужно использовать параметр ensure\_ascii с флагом true. Тогда все символы будут записаны корректно.
10. Схема данных определяет разрешенные типы данных для определенных полей, их количество и т.п.