Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра инфокоммуникаций

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.17**

**дисциплины «Анализ данных»**

**Вариант 13**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Выполнил:  Иващенко Олег Андреевич  2 курс, группа ИВТ-б-о-22-1,  09.03.02 «Информационные и вычислительные машины», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |  |
|  | | Руководитель практики:  Воронкин Роман Александрович, доцент кафедры инфокоммуникаций  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |  |
|  |  | |  | |

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ставрополь, 2024 г.

**Тема**: «Разработка приложений с интерфейсом командной строки (CLI) в Python3»

**Цель**: Приобретение навыков построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Порядок выполнения работы

Пример. Для примера 1 лабораторной работы 2.16 разработать интерфейс командной строки.

Таблица 1 – Код программы example.py:

|  |
| --- |
| #!/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(command\_line)        # 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 рааботников.      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() |

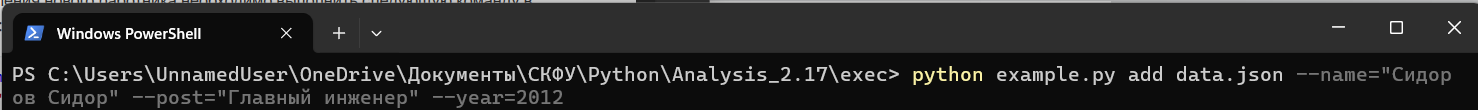


Рисунок 1.1 – Добавление новой записи в файл data.json

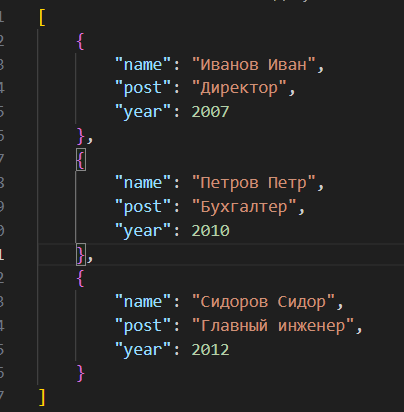


Рисунок 1.2 – Содержимое файла data.json

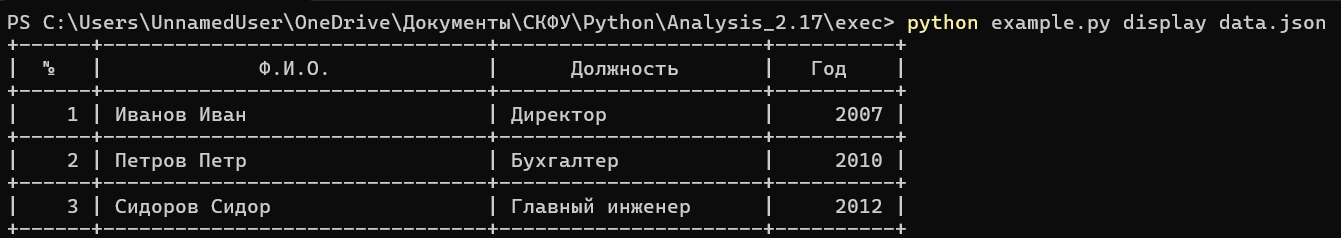


Рисунок 1.3 – Вывод содержимого файла data.json

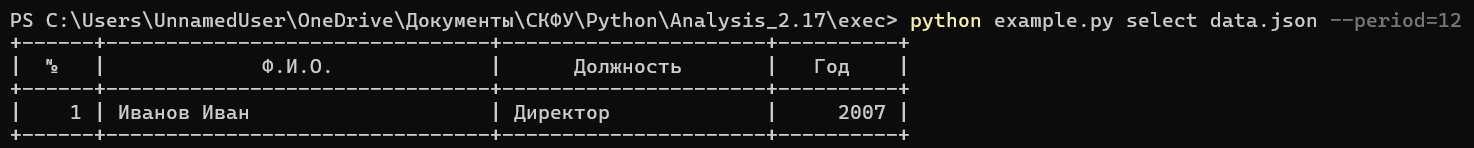


Рисунок 1.4 – Выборка из файла

Индивидуальное задание. Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

Таблица 2 – Код программы

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-    import json  import sys  from datetime import datetime  import argparse  import os.path    def print\_help():      """      Функция вывода доступных пользователю команд      """        print("list - вывод всех добавленных записей")      print("add - добавление новых записей")      print("find - найти запись по фамилии")      print("exit - завершение работы программы")    def add\_worker(workers, surname, name, phone, date):      """      Функция добавления новой записи, возвращает запись      """        workers.append(          {              "surname": surname,              'name': name,              'phone': phone,              'date': date          }      )        return workers    def print\_list(list):      """      Функция выводит на экран список всех существующих записей      """        for member in list:          print(f"{member['surname']} {member['name']} | "                  f"{member['phone']} | {member['date']}")      def find\_member(workers, period):      """      Функция для вывода на экран всех записей, чьи фамилии совпадают      с введённой (не возвращает никаких значений)      """        count = 0      members = []        for member in workers:          year = datetime.strptime(member['date'], "%d.%m.%Y").year          if datetime.now().year - period >= year:              members.append(member)              count += 1        if count == 0:          print("Записи не найдены")      else:          return members    def save\_file(filename, data):      """      Сохранение списка сотрудников в файл формата JSON      """        with open(filename, "w", encoding="utf-8") as file:          json.dump(data, file, ensure\_ascii=False, indent=4)    def load\_file(filename):      """      Загрузка данных о сотрудниках из указанного JSON-файла      """        with open(filename, "r", encoding="utf-8") as file:          return json.load(file)      def parse\_datetime(value):      try:          return datetime.strptime(value, "%d.%m.%Y")      except ValueError:          print("Error")    def main(command\_line=None):      file\_parser = argparse.ArgumentParser(add\_help=False)      file\_parser.add\_argument(          "filename",          action="store",          help="The data file name"      )        parser = argparse.ArgumentParser("workers")      parser.add\_argument(          "--version",          action="version",          version="%(prog)s 0.1.0"      )        subparsers = parser.add\_subparsers(dest="command")        add = subparsers.add\_parser(          "add",          parents=[file\_parser],          help="Add a new worker"      )      add.add\_argument(          "-s",          "--surname",          action="store",          required=True,          help="The worker's surname"      )      add.add\_argument(          "-n",          "--name",          action="store",          required=True,          help="The worker's name"      )      add.add\_argument(          "-p",          "--phone",          action="store",          help="The worker's phone"      )      add.add\_argument(          "-d",          "--date",          action="store",          required=True,          help="The date of hiring"      )        \_ = subparsers.add\_parser(          "display",          parents=[file\_parser],          help="Display all workers"      )        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"      )        args = parser.parse\_args(command\_line)        is\_dirty = False      if os.path.exists(args.filename):          workers = load\_file(args.filename)      else:          workers = []        if args.command == "add":          workers = add\_worker(              workers,              args.surname,              args.name,              args.phone,              args.date          )          is\_dirty = True        elif args.command == "display":          print\_list(workers)        elif args.command == "select":          selected = find\_member(workers, args.period)          print\_list(selected)        if is\_dirty:          save\_file(args.filename, workers)    if \_\_name\_\_ == "\_\_main\_\_":      """      Основная программа      """      main() |

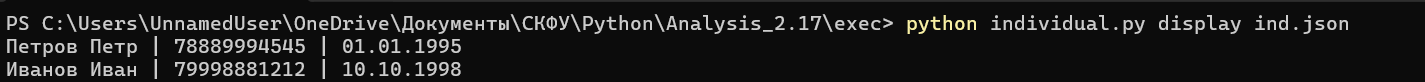


Рисунок 2.1 – Вывод изначального списка

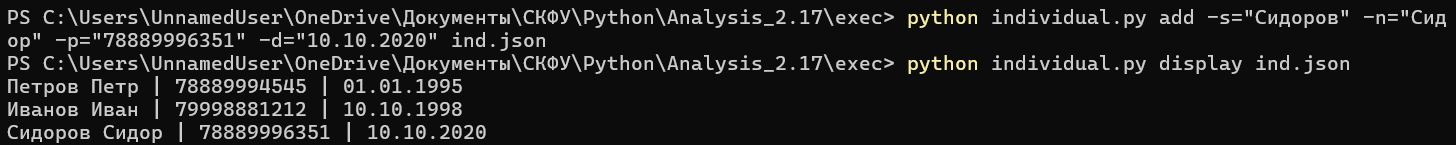


Рисунок 2.2 – Добавление новой записи и вывод списка

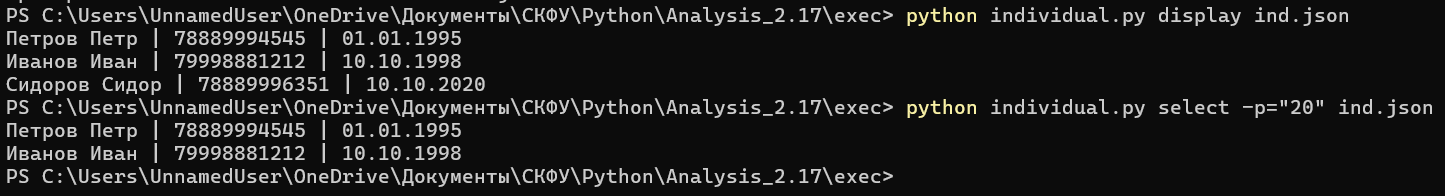


Рисунок 2.3 – Выборка из списка

Задание повышенной сложности. Самостоятельно изучить работу с пакетом click для построения интерфейса командной строки (CLI). Для своего варианта лабораторной работы 2.16 необходимо реализовать интерфейс командной строки с использованием пакета click.

Таблица 3 – Код программы

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-    import json  import sys  from datetime import datetime  import argparse  import os.path  import click    @click.group()  def cli():      pass    @cli.command()  @click.option('--filename', '-f', help='Имя файла')  def display(filename):      """      Функция (click) вывода списка существующих записей из указанного файла      """        try:          workers = load\_file(filename)          print\_list(workers)          print()      except:          print("Ошибка в имени файла\n")    @cli.command()  @click.option('--filename', '-f', help='Имя файла')  @click.option("--surname", "-s", help="Фамилия работника")  @click.option("--name", "-n", help="Имя работника")  @click.option("--phone", "-p", help="Номер телефона работника")  @click.option("--date", "-d", help="Дата приёма работника")  def add(filename, surname, name, phone, date):      """      Функция (click) добавления в указанный файл новой записи      """        workers = load\_file(filename)      workers = add\_worker(workers, surname, name, phone, date)      save\_file(filename, workers)    @cli.command()  @click.option('--filename', '-f', help='Имя файла-источника')  @click.option('--period', '-p', type=int, help='Искомый период (лет)')  def select(filename, period):      """      Функция (click) выбора записей из указанного файла по периоду      """        try:          workers = find\_member(load\_file(filename), period)          print\_list(workers)          print()      except TypeError:          print("Записи не найдены\n")    def add\_worker(workers, surname, name, phone, date):      """      Функция добавления новой записи, возвращает запись      """        workers.append(          {              "surname": surname,              'name': name,              'phone': phone,              'date': date          }      )        return workers    def print\_list(list):      """      Функция выводит на экран список всех существующих записей      """        for member in list:          print(f"{member['surname']} {member['name']} | "                  f"{member['phone']} | {member['date']}")      def find\_member(workers, period):      """      Функция для вывода на экран всех записей, чьи фамилии совпадают      с введённой (не возвращает никаких значений)      """        count = 0      members = []        for member in workers:          year = datetime.strptime(member['date'], "%d.%m.%Y").year          if datetime.now().year - period >= year:              members.append(member)              count += 1        if count != 0:          return members    def save\_file(filename, data):      """      Сохранение списка сотрудников в файл формата JSON      """        with open(filename, "w", encoding="utf-8") as file:          json.dump(data, file, ensure\_ascii=False, indent=4)    def load\_file(filename):      """      Загрузка данных о сотрудниках из указанного JSON-файла      """        with open(filename, "r", encoding="utf-8") as file:          return json.load(file)      def parse\_datetime(value):      try:          return datetime.strptime(value, "%d.%m.%Y")      except ValueError:          print("Error")    if \_\_name\_\_ == "\_\_main\_\_":      """      Основная программа      """      cli() |

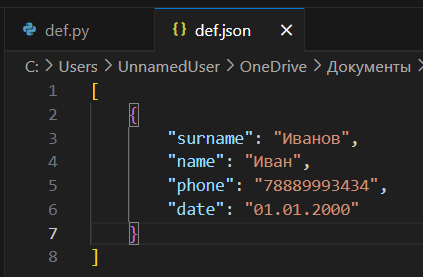


Рисунок 3.1 – Изначальное содержимое файла def.json

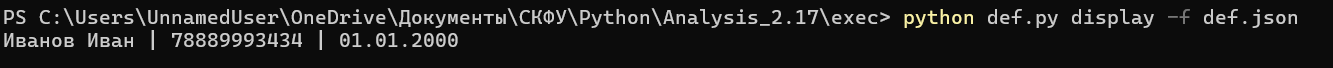


Рисунок 3.2 – Вывод содержимого файла в консоль

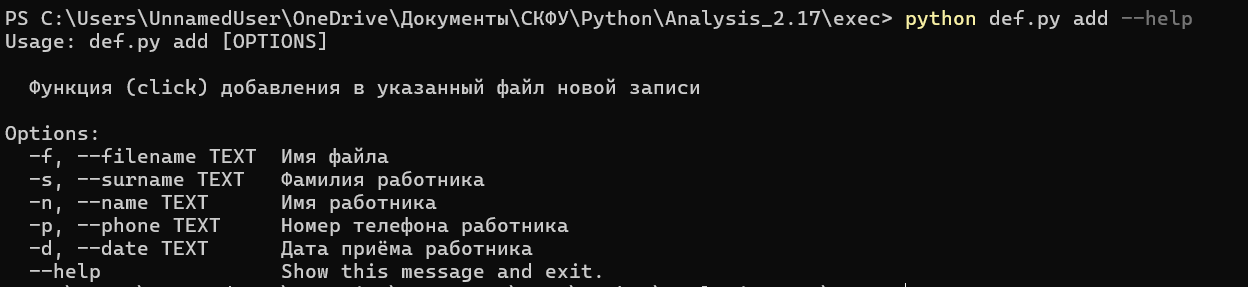


Рисунок 3.3 – Вывод подсказки команды add

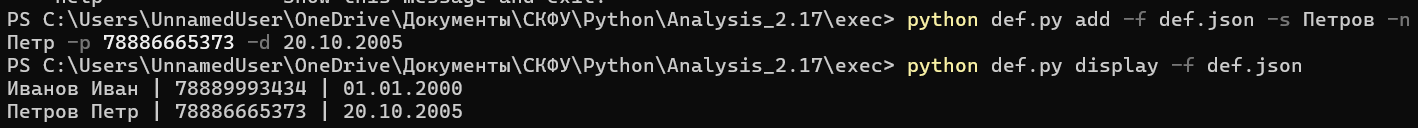


Рисунок 3.4 – Добавление новой записи

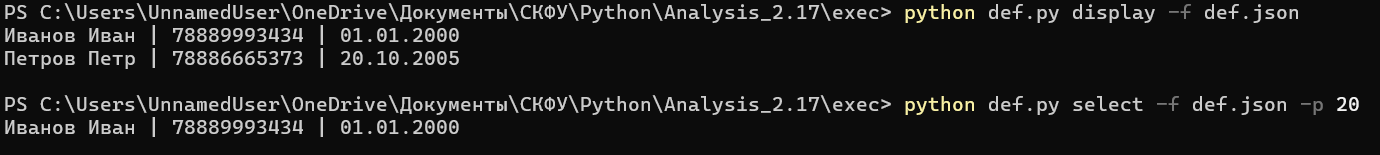


Рисунок 3.5 – Выборка записей по периоду

Контрольные вопросы

1. В чём отличие терминала и консоли?

Терминал – устройство или ПО, выступающее посредником между человеком и вычислительной системой. Обычно данный термин используется, когда точка доступа к системе вынесена в отдельное физическое устройство и предоставляет свой пользовательский интерфейс на основе внутреннего интерфейса (например, сетевых протоколов).

Консоль – исторически реализация терминала с клавиатурой и текстовым дисплеем. В настоящее время это слово часто используется как синоним сеанса работы или окна оболочки командной строки. В том же смысле иногда применяется и слово «терминал».

1. Что такое консольное приложение?

Консольное приложение – вид ПО, разработанный с расчётом на работу внутри оболочки командной строки, т.е. опирающийся на текстовый ввод-вывод.

1. Какие существуют средства языка программирования Python для построения приложений командной строки?

sys – модуль, предоставляющий доступ к некоторым переменным и функциям, специфичными для интерптератора Python. Хотя sys в первую очередь используется для управления интерпретатором и его окружением, его так же можно использовать для обработки аргументов командной строки, переданных при запуске скрипта, с помощью парса sys.argv.

getopt – модуль, предоставляющий функции для разработки аргументов командной строки в стиле Unix, когда аргументы могут быть короткими (однобуквенными) или длинными (полнословными). Этот модуль предоставляет функцию getopt.getopt, которая позволяет определить список опций и их значения, переданных скрипту через аргументы командной строки.

argparse – стандартная библиотека Python, предназначенная для разбора аргументов командной строки и создания интерфейса командной строки. Она позволяет легко определять аргументы, поддерживает различные типы аргументов и автоматически генерирует справку для пользователя.

1. Какие особенности построения CLI с использованием модуля sys?

Использование sys.argv предоставляет простой способ доступа к аргументам командной строки без необходимости импортировать дополнительные библиотека. Однако нужно работать напрямую с аргументами командной строки, поэтому требуется вручную проверять и обрабатывать аргументы. Это может привести к большему объёму кода и более сложному управлению ошибками.

1. Какие особенности построения CLI с использованием модуля getopt?

Модуль getopt позволяет определять и использовать как коротки (однобуквенные), так и длинные (полные слова) опции командной строки. Но подобно sys.argv, getopt требует ручного разбора аргументов. Это может быть утомительным и привести к большему количеству кода, особенно при обработке различных опций и аргументов.

1. Какие особенности построения CLI с использованием модуля argparse?

Модуль argparce предоставляет высокоуровневый API для определения и обработки аргументов командной строки. Можно определять ожидаемые аргументы и опции с помощью декларативного синтаксиса, что делает код более читаемым и поддерживаемым. argparce автоматически генерирует справку на основе определённых аргументов и опций. Это упрощает документирование программы и делает её более доступной для пользователей. Также argparse позволяет определять различные типы аргументов, такие как целые числа, строки и другие. Она также обеспечивает проверку корректности введённых значений, что может сократить количество ошибок и упростить отладку.

**Выводы**: В процессе выполнения лабораторной работы были приобретены навыки построения приложений с интерфейсом командной строки с помощью языка программирования Python, был изучен пакет click, позволяющий другими методами создать интерфейс CLI. Был проработан пример, выполнено индивидуальное задание и задание повышенной сложности.