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 №2.16**

**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**: «Работа с данными формата JSON в языке Python»

**Цель**: Приобретение навыков по работе с данными формата JSON с помощью языка программирования Python версии 3.x.

Порядок выполнения работы

Пример. Для примера 1 лабораторной работы 2.8 добавить возможность сохранения списка в файл формата JSON и чтения данных из файла.

Таблица 1 – Код программы example.py

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  import json  import sys  from datetime import date  def get\_worker():      """      Запросить данные о работнике.      """      name = input("Фамилия и инициалы? ")      post = input("Должность? ")      year = int(input("Год поступления? "))      # Создать словарь.      return {          'name': name,          'post': post,          'year': year,      }  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():      """      Главная функция программы.      """      # Список работников.      workers = []      # Организовать бесконечный цикл запроса команд.      while True:          # Запросить команду из терминала.          command = input(">>> ").lower()          # Выполнить действие в соответствие с командой.          if command == "exit":              break          elif command == "add":              # Запросить данные о работнике.              worker = get\_worker()              # Добавить словарь в список.              workers.append(worker)              # Отсортировать список в случае необходимости.              if len(workers) > 1:                  workers.sort(key=lambda item: item.get('name', ''))          elif command == "list":              # Отобразить всех работников.              display\_workers(workers)          elif command.startswith("select "):              # Разбить команду на части для выделения стажа.              parts = command.split(maxsplit=1)              # Получить требуемый стаж.              period = int(parts[1])              # Выбрать работников с заданным стажем.              selected = select\_workers(workers, period)              # Отобразить выбранных работников.              display\_workers(selected)          elif command.startswith("save "):              # Разбить команду на части для выделения имени файла.              parts = command.split(maxsplit=1)              # Получить имя файла.              file\_name = parts[1]              # Сохранить данные в файл с заданным именем.              save\_workers(file\_name, workers)          elif command.startswith("load "):              # Разбить команду на части для выделения имени файла.              parts = command.split(maxsplit=1)              # Получить имя файла.              file\_name = parts[1]              # Сохранить данные в файл с заданным именем.              workers = load\_workers(file\_name)          elif command == 'help':              # Вывести справку о работе с программой.              print("Список команд:\n")              print("add - добавить работника;")              print("list - вывести список работников;")              print("select <стаж> - запросить работников со стажем;")              print("help - отобразить справку;")              print("load - загрузить данные из файла;")              print("save - сохранить данные в файл;")              print("exit - завершить работу с программой.")          else:              print(f"Неизвестная команда {command}", file=sys.stderr)  if \_\_name\_\_ == '\_\_main\_\_':      main() |

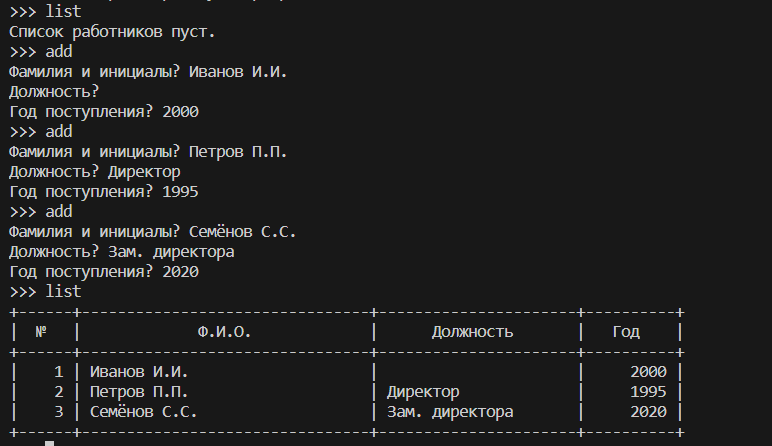


Рисунок 1.1 – Добавление новых записей

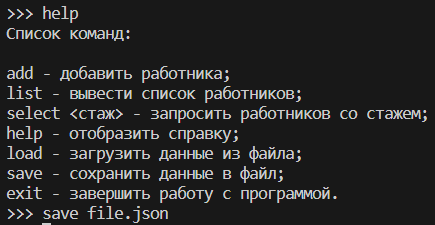


Рисунок 1.2 – Вывод команды help и сохранение записей в файл

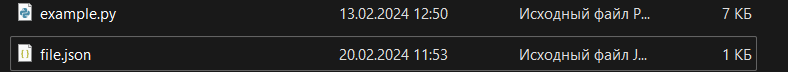


Рисунок 1.3 – Сохранённый файл

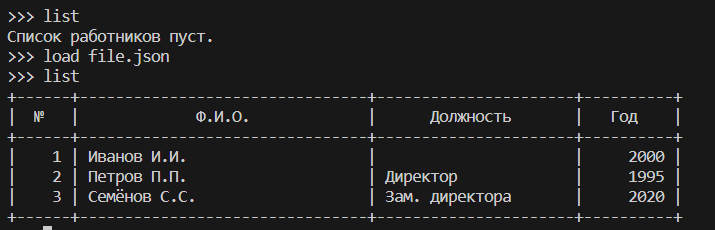


Рисунок 1.4 – Загрузка данных из файла

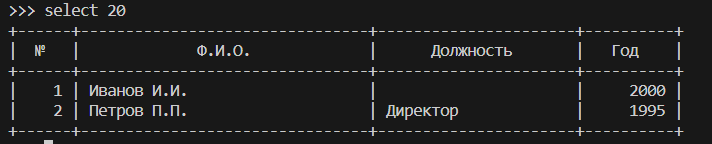


Рисунок 1.5 – Вывод сотрудников по стажу

Индивидуальное задание. Для своего варианта лабораторной работы 2.8 необходимо дополнительно реализовать сохранение и чтение данных из файла формата JSON. Необходимо также проследить за тем, чтобы файлы, генерируемые этой программой, не попали в репозиторий лабораторной работы.

Таблица 2 – Код программы individual.py

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  import json  import sys  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():      """      Функция добавления новой записи, возвращает запись      """        surname = input("Введите фамилию: ")      name = input("Введите имя: ")      phone = input("Введите номер телефона: ")      date = tuple(map(int, input("Введите дату рождения: ").split('.')))         new\_member = {'surname': surname,                      'name': name,                      'phone': phone,                      'date': date                  }        return new\_member  def print\_list(list):      """      Функция выводит на экран список всех существующих записей      """      for member in member\_list:          print(f"{member['surname']} {member['name']}, "                  f"{member['phone']}, {member['date']}")  def find\_member(surname):      """      Функция для вывода на экран всех записей, чьи фамилии совпадают      с введённой (не возвращает никаких значений)      """        count = 0     for member in member\_list:          if member['surname'] == surname:              print(f"{member['surname']} {member['name']}, "                  f"{member['phone']}, {member['date']}")              count += 1            if count == 0:              print("Записи не найдены")  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)  if \_\_name\_\_ == "\_\_main\_\_":      """      Основная программа      """        member\_list = []      while True:          cmd = input(">>> ")          if cmd == "help":              print\_help()         elif cmd == "add":              member\_list.append(add())              member\_list.sort(key=lambda item: item.get('phone')[:3])          elif cmd == "list":              print\_list(member\_list)            elif cmd == "find":              find\_member(input("Введите фамилию: "))          elif cmd.startswith("save"):              data = cmd.split(" ")              save\_file(data[1], member\_list)          elif cmd.startswith("load"):              data = cmd.split(" ")              member\_list = load\_file(data[1])          elif cmd == "exit":              print("Завершение работы программы...")              break          else:              print(f"Команды {cmd} не существует") |

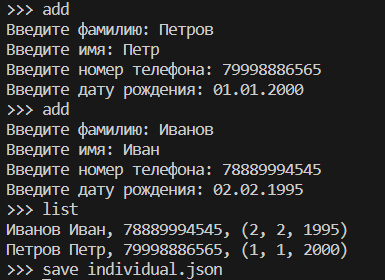


Рисунок 2.1 – Добавление данных и их сохранение в файл

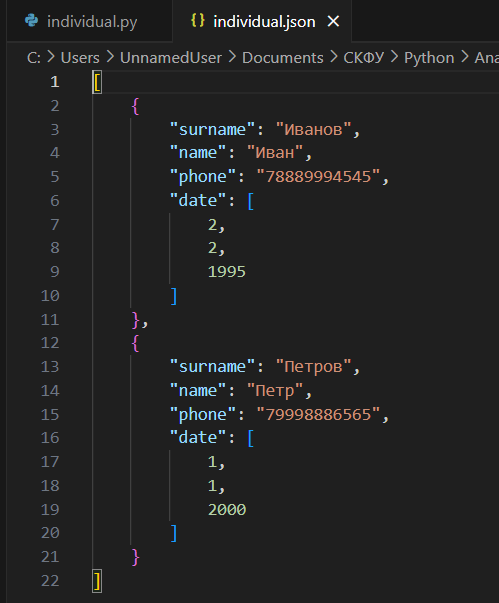


Рисунок 2.2 – Данные в сохранённом файле

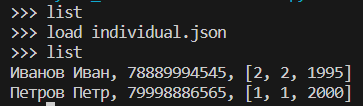


Рисунок 2.3 – Загрузка данных из файлов в программу

Задание повышенной сложности. Необходимо после загрузки из файла JSON выполнять валидацию загруженных данных. Валидацию данных необходимо производить с использованием спецификации JSON Schema. Одним из возможных вариантов работы с JSON Schema является использование пакета jsonschema. Таким образом необходимо реализовать валидацию загруженных данных с помощью спецификации JSON Schema.

Таблица 3 – Код программы

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  import json  import sys  import os  from jsonschema import validate  from datetime import date  new\_schema = {      "type": "object",      "properties": {          "name": {"type": "string"},          "post": {"type": "string"},          "year": {"type": "number"}      },      "required": ["name", "post", "year"],  }  def get\_worker():      """      Запросить данные о работнике.      """      name = input("Фамилия и инициалы? ")      post = input("Должность? ")      year = int(input("Год поступления? "))      worker\_data = {'name': name, 'post': post, 'year': year}      try:          validate(instance=worker\_data, schema=new\_schema)          return {          'name': name,          'post': post,          'year': year,          }      except Exception as e:          print("Ошибка данных")  def display\_workers(staff):      """      Отобразить список работников.      """      if staff:          line = '+-{}-+-{}-+-{}-+-{}-+'.format(              '-' \* 4,              '-' \* 30,              '-' \* 20,              '-' \* 8          )          print(line)          print(              '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(                  "№",                  "Ф.И.О.",                  "Должность",                  "Год"              )          )          print(line)          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.      """      today = date.today()      result = []      for employee in staff:          if today.year - employee.get('year', today.year) >= period:              result.append(employee)      return result  def save\_workers(file\_name, staff):      """      Сохранить всех работников в файл JSON.      """      with open(file\_name, "w", encoding="utf-8") as fout:          json.dump(staff, fout, ensure\_ascii=False, indent=4)  def load\_workers(file\_name):      """      Загрузить всех работников из файла JSON.      """      if os.path.exists(file\_name):          with open(file\_name, "r", encoding="utf-8") as fin:              file\_data = json.load(fin)              try:                  for data in file\_data:                      validate(instance=data, schema=new\_schema)                  return file\_data              except Exception as e:                  print("Ошибка загрузки данных")                  return None      else:          print("Файл не существует")          return None  def main():      """      Главная функция программы.      """      workers = []      while True:          command = input(">>> ").lower()          if command == "exit":              break          elif command == "add":              worker = get\_worker()              workers.append(worker)              if len(workers) > 1:                  workers.sort(key=lambda item: item.get('name', ''))          elif command == "list":              display\_workers(workers)          elif command.startswith("select "):              parts = command.split(maxsplit=1)              period = int(parts[1])              selected = select\_workers(workers, period)              display\_workers(selected)          elif command.startswith("save "):              parts = command.split(maxsplit=1)              file\_name = parts[1]             save\_workers(file\_name, workers)          elif command.startswith("load "):              parts = command.split(maxsplit=1)              file\_name = parts[1]             workers = load\_workers(file\_name)          elif command == 'help':              print("Список команд:\n")              print("add - добавить работника;")              print("list - вывести список работников;")              print("select <стаж> - запросить работников со стажем;")              print("help - отобразить справку;")              print("load - загрузить данные из файла;")              print("save - сохранить данные в файл;")              print("exit - завершить работу с программой.")          else:              print(f"Неизвестная команда {command}", file=sys.stderr)  if \_\_name\_\_ == '\_\_main\_\_':      main() |

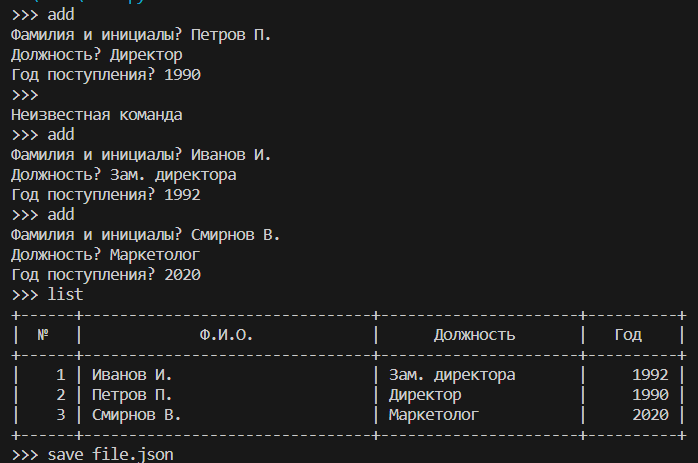


Рисунок 3.1 – Заполнение и сохранение данных

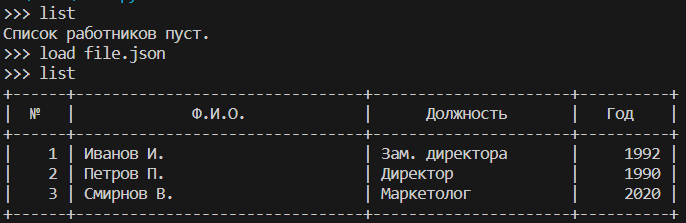


Рисунок 3.2 – Загрузка данных из файла

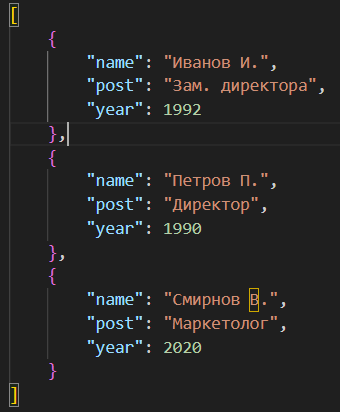


Рисунок 3.3 – Содержимое файла file.json

Контрольные вопросы

1. Для чего используется JSON?

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. Формат JSON был разработан Дугласом Крокфордом.

Легко читаемый и компактный JSON представляет собой хорошую альтернативу XML и требует куда меньше форматирования контента.

1. Какие типы значений используются в JSON?

JSON значения находятся с правой стороны от двоеточия. Если быть точным, то им нужно быть одним из шести типов данных:

* Строка очень похожа на литерал одноимённого типа данных в языке JavaScript. Строка – заданная последовательность из нуля и больше символов Юникода, заключённая в две двойные кавычки, например: {"firstName": "Tom"};
* Число в JSON должно быть целым или с плавающей запятой, например: {"age": 30};
* Объект содержит ключ и значение. После каждого ключа состоит двоеточие, а после каждого значения – запятая, которая также различает каждый объект. Оба они находятся внутри кавычек. Объект как значение должен подчиняться тому же правилу, что и объект, например:

{

"employees": {"fistName": "Tom", "lastName": "Jackson"}

}

Здесь employees – ключ, а всё, что находится внутри фигурных скобок - объект.

* Массив – это упорядоченная коллекция значений. Он заключён в квадратные скобки [], а каждое значение внутри разделено запятой. Значение массива может содержать объекты JSON, что означает, что он использует ту же концепцию пар ключей/значений, например:

{

"students":[

{"firstName":"Tom", "lastName":"Jackson"},

{"firstName":"Linda", "lastName":"Garner"},

{"firstName":"Adam". "lastName":"Cooper"}

]

}

Информация в квадратных скобках – это массива, в котором есть три объекта.

* Для данных в формате JSON допустим и булев тип. Можно использовать true или false в качестве значения, например: {"married": false};
* Значение null показывает отсутствие информации, например: {"bloodType": null}.

1. Как организована работа со сложными данными в JSON?

JSON может содержать другие вложенные обхекты в JSON, в дополнение к вложенным массивам. Такие объекты и массивы будут передаваться, как значения назначенные ключам и будут представлять собой связку ключ-значение.

* Вложенные объекты – в файле users.json для каждого из четырёх пользователей ("sammy", "jesse", "drew", "jamei") есть вложенный JSON объект, передающий значения для каждого из пользователей, со своими собственными вложенными ключами "username" и "location". Например:

|  |
| --- |
| { |
| "sammy" : { |
| "username" : "SammyShark", |
| "location" : "Indian Ocean", |
| "online" : true, |
| "followers" : 987 |
| }, |
| "jesse" : { |
| "username" : "JesseOctopus", |
| "location" : "Pacific Ocean", |
| "online" : false, |
| "followers" : 432 |
| }, |
| "drew" : { |
| "username" : "DrewSquid", |
| "location" : "Atlantic Ocean", |
| "online" : false, |
| "followers" : 321 |
| }, |
| "jamie" : { |
| "username" : "JamieMantisShrimp", |
| "location" : "Pacific Ocean", |
| "online" : true, |
| "followers" : 654 |
| } |
| } |

В примере фигурные скобки везде используются для форматирования вложенного JSON объекта с ассоциированными именами пользователей и данными локаций для каждого из них. Как и с любым другим значением, используя объекты, двоеточие используется для разделения элементов.

* Вложенные массивы – данные также могут быть вложены в формат JSON, используя JavaScript массивы, которые передаются как значения. JavaScript использует квадратные скобки [ ] для формирования массива. Массивы по своей сути – это упорядоченные коллекции и могут включать в себя значения совершенно разных типов данных.

Мы можем использовать массив при работе с большим количеством данных, которые могут быть легко сгруппированы вместе, как например, если есть несколько разных сайтов и профайлов в социальных сетях ассоциированных с одним пользователем. Пример:

|  |
| --- |
| { |
| "first\_name" : "Sammy", |
| "last\_name" : "Shark", |
| "location" : "Ocean", |
| "websites" : [ |
| { |
| "description" : "work", |
| "URL" : "https://www.digitalocean.com/" |
| }, |
| { |
| "description" : "tutorials", |
| "URL" : "https://www.digitalocean.com/community/tutorials" |
| } |
| ], |
| "social\_media" : [ |
| { |
| "description" : "twitter", |
| "link" : "https://twitter.com/digitalocean" |
| } |
| "description" : "facebook", |
| "link" : "https://facebook.com/DigitalOceanCloudHostion" |
| }, |
| { |
| "description" : "github", |
| "link" : "https://github.com/digitalocean" |
| } |
| ] |
| } |

Ключи "websites" и "social\_media" используют массив для вложения информации о сайтах пользователя и профайлов в социальных сетях. Мы знает, что это массивы – из-за квадратных скобок.

Использование вложенностей в JSON формате позволяет работать с наиболее сложными и иерархичными данными.

1. Самостоятельно ознакомьтесь с форматом данных JSONS. В чём отличие этого формата от формата данных JSON?

Формат данных JSONS (JSON Streaming) по сути представляет собой набор JSON-объектов, разделённых друг от друга символом новой строки (\n). Этот подход позволяет передавать и обрабатывать большие объемы данных постепенно, по мере их поступления, что делает его особенно полезным в сценариях потоковой обработки данных.

1. Какие средства языка программирования Python могут быть использованы для работы с данными в формате JSONS?

Для работы с данными в формате JSONS в Python можно использовать стандартные библиотеки для работы с JSON и потоковым вводом-выводом.

Стандартный модуль Python json предоставляет функции для сериализации (преобразования объектов Python в формат JSON) и десериализации (преобразования данных JSON в объекты Python). Методы json.loads() и json.dumps() могут использоваться для работы с данными JSONS.

1. Какие средства предоставляет язык Python для сериализации данных в формате JSON?

Сериализация данных в формат JSON:

* json.dump() – конвертирует python-объект в JSON и записывает в файл;
* json.dumps() – то же самое, но записывает в строку.

Оба эти функции принимают следующие необязательные аргументы:

* Если skipkeys = True, то ключи словаря не базового типа (str, int, float, bool, None) будут проигнорированы, вместо того, чтобы вызывать исключение TypeError;
* Если ensure\_ascii = True, все не-ASCII символы в выводе будут экранированы последовательностями \uxxxx, и результатом будет строка, содержащая только ASCII символы. Если ensure\_ascii = False, строки запишутся как есть;
* Если check\_circular = False, то проверка циклических ссылок будет пропущена, а такие ссылку будут вызывать OverflowError;
* Если allow\_nan = False, при попытке сериализовать значение с запятой, выходящее за допустимые пределы, будет вызываться ValueError (nan, inf, -inf) в строгом соответствии со спецификацией JSON, вместо того, чтобы использовать эквиваленты из JavaScript (NaN, Infinity, -Infinity);
* Если indent является неотрицательным числом, о массивы и объекты в JSON будут выводиться с этим уровнем отступа. Если уровень отступа 0, отрицательный или "", то вместо этого будут просто использоваться новые строки. Значения по умолчанию None отражает наиболее компактное представление. Если indent – строка, то она и будет использоваться в качестве отступа;
* Если sort\_keys = True, то ключи выводимого словаря будут отсортированы.

1. В чём отличие функций json.dump() и json.dumps()?

Отличие между функциями json.dump() и json.dumps() заключается в том, что json.dump() конвертирует python-объекты в JSON и записывает в файл, а json.dumps() записывает в строку.

1. Какие средства предоставляет язык Python для десериализации данных из формата JSON?

Десериализация данных из формата JSON:

* json.load() – прочитать JSON из файла и конвертировать в Python-объект;
* json.loads() – тоже самое, но из строки с JSON (s на конце от string).

Обе эти функции принимают следующие аргументы:

* object\_hook – опциональная функция, которая применяется к результату декодирования объекта (dict). Использоваться будет значение, возвращаемое этой функцией, а не полученный словарь;
* object\_pairs\_hook – опциональная функция, которая применяется к результату декодирования объекта с определённой последовательностью пар ключ/значение. Будет использован результат, возвращаемый функцией, вместо исходного словаря. Если задан так же object\_hook, то приоритет отдаётся object\_pairs\_hook;
* parse\_float, если определён, будет вызван для каждого значения JSON с плавающей точкой. По умолчанию, это эквивалентно float(nun\_str);
* parse\_int, если определён, будет вызван для строки JSON с числовым значение. По умолчанию эквивалентно int(num\_str);
* parse\_constant, если определён, будет вызван для следующих строк: "-Infinity", "Infinity", "NaN". Может быть использовано для возбуждения исключений при обнаружении ошибочныз чисел JSON.

Если не удастся десериализовать JSON, будет вощбуждено исключение ValueError.

1. Какие средства необходимо использовать для работы с данными формата JSON, содержащим кириллицу?

Для работы с данными формата JSON, содержащими кириллицу, можно использовать стандартный модуль json в Python, так как он поддерживает Unicode, включая символы кириллицы.

1. Самостоятельно ознакомьтесь со спецификацией JSON Schema. Что такое схема данных? Приведите схему данных для примера 1.

JSON Schema - это язык описания структуры данных в формате JSON. Он позволяет определить правила и ограничения для JSON-документов, определяя их структуру, типы данных, обязательные и необязательные поля, а также валидацию данных.

Схема данных - это формальное описание структуры данных, которое определяет типы данных, их отношения, ограничения и правила валидации. Схема данных определяет, как данные должны быть представлены и организованы, чтобы соответствовать заданным требованиям.

Схема данных для примера 1:

new\_schema = {

    "type": "object",

    "properties": {

        "id": {"type": "number"},

        "name": {"type": "string"},

"post": {"type": "string"},

        "year": {"type": "number"}

    },

    "required": ["name", "post", "year"],

}

**Выводы**: В процессе выполнения лабораторной работы были приобретены навыки по работе с данными формата JSON с помощью языка программирования Python версии 3.x, проработан пример и выполнены индивидуальное задание и задание повышенной сложности.