Практическая работа №3

Задание

Краткое задание

Разработать программу для взаимодействия с файлами в различных форматах с возможностью их модификации, создания или удаления. Разработанный код должен иметь возможность его повторного использования и модификации. Код должен быть стойким к ошибкам ввода (предполагается использование конструкций обработки ошибок `try:... except:...` и дополнительных проверок входных данных). Программа должна содержать интерфейс для ввода данных и выбора действий, тип интерфейса: графический или консольный.

Рекомендуемый язык программирования: Python

Форма отчёта: файл с титульной страницей в формате PDF или DOCX (с ФИО, группой), копируемый исходный код. Прикреплённый отчёт в СДО.

Формат защиты: после прикрепления отчёта в СДО, подходить к преподавателю лично, для демонстрации полученных результатов. Для наглядности можно показать работоспособность кода в среде разработки (с компьютера, ноутбука, телефона). Суметь ответить на поясняющие вопросы.

Срок сдачи работы: до 5-го практического занятия

Детальное задание

1. Вывести информацию в консоль о логических дисках, именах, метке тома, размере и типе файловой системы (модуль psutil).

2. Работа с файлами (модуль os, shutil)

* Создать файл

* Записать в файл строку, введённую пользователем

* Прочитать файл в консоль

* Удалить файл

3. Работа с форматом JSON (модуль json)

* Создать файл формате JSON в любом редакторе или с использованием данных, введенных пользователем

* Создать новый объект. Выполнить сериализацию объекта в формате JSON и записать в файл.

* Прочитать файл в консоль

* Удалить файл

4. Работа с форматом XML (модуль xml.etree.ElementTree)

* Создать файл формате XML из редактора

* Записать в файл новые данные из консоли.

* Прочитать файл в консоль.

* Удалить файл.

5. Создание zip архива, добавление туда файла, определение размера архива (модуль zipfile)

* Создать архив в форматер zip

* Добавить файл, выбранный пользователем, в архив

* Разархивировать файл и вывести данные о нем

* Удалить файл и архив

Теоретическая часть

Среды разработки Python:

* PyCharm Community

* VScode

* Онлайн-компилятор CodeChef

* Компилятор Online Python

* PyDroid (на Android, доступен в Play Market)

Установка дополнительных пакетов: `pip install название_пакета`

JSON-документ содержит текст, фигурные и квадратные скобки, двоеточия, запятые, двойные кавычки и, может быть, некоторые другие символы.

Самая заметная вещь в JSON — то, что его данные состоят из пар имя/значение и отражают вложенную структуру с данными. Данные записываются в виде пар «ключ — значение» и разделяются запятыми. Ключи — строковые переменные, а значения могут быть строками, числами, булевыми значениями (true/false), объектами ({"key": "value"}), массивами ([1, 2, 3]) или null.

Значениями могут быть вложенные JSON-объекты: {"people": {"gender": "male", "age": 20}}

Пример JSON-формата:

```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
        "vOffset": 250,
        "alignment": "center"
    },
```

```
    "text": {
        "data": "Click Here",
        "size": 36,
        "style": "bold",
        "name": "text1",
        "hOffset": 250,
        "vOffset": 100,
        "alignment": "center",
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
    }
}}
```

Возвести строку с JSON-данными в JSON-подобный словарь Python можно командой `json.loads`:

```
# Модуль работы с JSON
import json

# Создание текстовой переменной, структура которой является json
str_json = '{ "id": 321, "fio": "Alexander M.A.", "discipline":
"Programming", "passed": null}'

# Конвертировать строку JSON в пайтоновский dict
python_dict = json.loads(str_json)

# Print Dictionary
print(python_dict) # {'id': 321, 'fio': 'Alexander M.A.',
'discipline': 'Programming', 'passed': None}

# Получить отдельные значения по их ключам в пайтоновском dict
print(python_dict['fio']) # Alexander M.A.
print(python_dict['id'])  # 321
```

Обратите внимание, пайтоновский dict использует вместо JSON-овского `null` – значение `None`, вместо `false` – `False`, а вместо `true` – `True`.

Чтобы из файла с JSON-форматом загрузить данные в пайтоновский dict, можно использовать функцию `json.load()`, используя менеджер контекста `with` и дескриптор файла `open('namefile')`:

```
import json

# Открытие JSON-файла 'data.json' в менеджере контекста with
with open('data.json') as json_file:
    data = json.load(json_file)

    # Проверка, что преобразовалось в пайтоновский dict
    print("Type:", type(data))

    # Извлекаем и печатаем нужные данные
    rez=data['widget']['window']['title']
    print(rez)
```

Для сериализации JSON в файл, из пайтоновского dict, достаточно преобразовать пайтоновский dict в строку JSON формата (через `json.dump`), и записать полученное значение в файл. Чтобы файл получился читаемым, рекомендуется прописать доп.параметры `indent=4, ensure_ascii=False`:

```
import json

# Создание пайтоновского dict
d = {
"data": "Click Here",
"size": 36,
"style": "bold",
"name": "text1",
"hOffset": 250,
"vOffset": 100,
"alignment": "center",
"onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
}

# В менеджере контекста создаём файл sample.json на запись
with open("sample.json", "w") as outfile:

    # Преобразовываем в JSON dict-переменную "d", в файл outfile, с
читаемым выводом
    json.dump(d, outfile, indent=4, ensure_ascii=False)
```

Вывести информацию, какие буквы дисков присутствуют в системе, при помощи модуля `psutil`, функции `disk_partitions`:

```
# Для того, чтобы узнать съёмные и несъёмные диски устройства
from psutil import disk_partitions

# Вывести, какие диски присутствуют на устройстве
disk_list = disk_partitions()
for i in disk_list:
    print(i.device, end=' ') # C:\ D:\ E:\ F:\
```

XML — популярный формат для обмена данными между различными системами и приложениями. Его универсальность делает XML важным инструментом в веб-разработке, настройке программного обеспечения и других областях. Расширяемый язык разметки, предназначенный для хранения и передачи данных в структурированном виде. Данные находятся внутри тегов, которые помогают организовать данные в логической иерархии, понятной человеку и компьютеру. Например:

```
<message>
   Привет, мир!
</message>
```

`<message>` и `</message>` - теги, определяющие начало и конец элемента. Фраза `Привет, мир!` — это данные внутри тега, которые необходимо сохранить или передать.

Допустим, из Python нужно создать XML следующего содержания:

```
<config>
   <database>
       <host>localhost</host>
       <port>3306</port>
       <username>root</username>
       <password>password</password>
   </database>
```

```
    </config>
```

Тогда в Python это будет выглядеть так:

```
# Импорт библиотеки по работе с XML
import xml.etree.ElementTree as ET

# Создаём корневой тег
element1=ET.Element('config')

# Создаём вложенный тег
element1_1=ET.SubElement(element1, 'database')

# Можно создать элемент через Element отдельно, потом добавив его
через append
element1_1_1=ET.Element('host')
element1_1_1.text='localhost'
element1_1.append(element1_1_1)

# А можно использовать SubElement – будет добавлять автоматически
element1_1_2=ET.SubElement(element1_1, 'port')
element1_1_2.text = '3306'

# Добавление третьего тега внутри тега element1_1 (database)
element1_1_3 = ET.SubElement(element1_1, 'password')
element1_1_3.text = '12345'

# Напечатать текущий результат
ET.dump(element1)

# Сохранить в файл текущий результат - через ElementTree, указав
корневой тег, применив write()
ET.ElementTree(element1).write('test.xml')
```

Чтобы прочитать XML-файл и отредактировать его содержимое, например, поменять пароль на 54321, в Python это будет представлено следующим образом:

```
import xml.etree.ElementTree as ET

# Загрузить XML в Python
tree = ET.parse('test.xml')

# Получить корневой тег
root = tree.getroot()

# Изменение значения первого элемента <password>
password_element = root.find('.//password')
```

```
if password_element is not None:
    password_element.text = '54321'  # Новое значение

# Перезаписать файл с учетом изменений
tree.write('test.xml')
```