Методические указания

Урок 11.1. Работа с файлами

Задачи урока:

- Познакомиться с инструкцией with

0. Подготовка к уроку

До начала урока преподавателю необходимо:

1) Просмотреть, как ученики справились с домашним заданием
2) Прочитать методичку

1. Сохранение файлов в разных форматах

Учитель: Сегодня мы с вами рассмотрим чтение различных форматов файлов, а также немного познакомимся с модулем pickle.

Для начала вспомним, что на прошлом занятии мы научились считывать и записывать текстовые файлы, но помимо текстовых файлов мы можем например открывать изображения стандартными средствами нашей операционной системы. Для открытия изображения нам достаточно импортировать модуль os и открыть изображение с помощью system()

```
import os
os.system('1.jpg')
```

Таким же образом мы можем открывать и другие форматы файлов. Например текстовый

```
import os
os.system('1.txt')
```

os.system(команда) - позволяет выполнить какую либо команду для операционной системы. Это может быть не обязательно открытие какого либо файла, но например и создание папки

```
import os
os.system('mkdir test')
```

В данном примере создается папка test в текущем каталоге с помощью команды mkdir. Или же например мы можем запустить другой python файл, который у нас рядом с основным файлом

```
import os

os.system('python test.py')
```

Важно: в данном случае примеры выше рассчитаны, что у вас операционная система Windows, а для файлов, которые вы хотите открыть у вас уже назначена стандартная утилита.

Учитель: Давайте немного вернемся к текстовым файлам и представим, что у нас есть текстовый файл с языками программирования на отдельных строках

```
Python
C++
C#
Java
JavaScript
```

Предположим мы хотим считать этот файл и сохранить в виде строки через запятую

```
with open('1.txt') as file:
  str1 = file.read()

print(str1)
```

В данном случае данные из текстового файла сохранились в строку str1, но при записи сохранились и все отступы и переносы строк. Как решить данную проблему?

Можно конечно использовать readlines() и мы получим список строк, но у нас останутся символы переноса строки, от которых нам потребуется избавиться

```
with open('1.txt') as file:
  str1 = file.readlines()

print(str1)
```

Результат

['Python\n', 'C++\n', 'C#\n', 'Java\n', 'JavaScript']

Избавиться от этих символов можно, но давайте найдем другой вариант. На помощь нам придет метод split(), который по указанному разделителю возвращает нам список символов.

В данном случае мы хотим, чтобы у нас были в качестве элемента списка языки программирования мы оставим разделитель по умолчанию(по пробелу)

```
with open('1.txt') as file:
  str1 = file.read()

word_list = str1.split()

print(word_list)
```

Результат

['Python', 'C++', 'C#', 'Java', 'JavaScript']

Последнее что нам требуется это объединить их в строку через запятую. Попробуем перебрать список с помощью цикла и создадим новую строку

```
with open('1.txt') as file:
  str1 = file.read()

word_list = str1.split()
result = ''
for word in word_list:
  result += word + ','
print(result)
```

Результат

Python,C++,C#,Java,JavaScript,

Уже близко, но результат не совсем тот. Можно конечно убрать запятую в конце, но это не совсем удобный подход. В данном случае все, что нам требуется это вспомнить отличный метод join(), который принимает итерируемый объект и возвращает строку с использованием разделителя

```
with open('1.txt') as file:
  str1 = file.read()

word_list = str1.split()
result = ','.join(word_list)
print(result)
```

Результат

Python,C++,C#,Java,JavaScript

Учитель: Самыми популярными форматами являются:

- CSV (англ. Comma-Separated Values - значения, разделенные запятыми)
- JSON (англ. JavaScript Object Notation) - текстовый формат обмена данными, основанный на JavaScript

- XML (англ. eXtensible Markup Language - расширяемый язык разметки)
- YAML (англ. YAML Ain't Markup Language - «YAML - Не язык разметки»)
- INI (англ. Initialization file - файл инициализации);

Большинство форматов файлов поддерживается Python

Рассмотрим некоторые варианты работы с различными типами файлов.

JSON (англ. JavaScript Object Notation, 1999 г.) - текстовый формат обмена данными, основанный на JavaScript. Одно из преимуществ - JSON легко читается людьми (англ. human-readable).JSON используется при передачи и получении данных в сети.

JSON-текст представляет собой одну из двух структур:
- набор пар ключ: значение (словарь в Python), где ключ - строка, значение - любой тип;
- упорядоченный набор значений (список в терминологии Python).

Значением может являться:
- строка (в кавычках);
- число;
- логическое значение (true/false);
- null;
- одна из структур.

json.dumps - Сериализует объект obj, возвращая строку в JSON-формате.
Синтаксис:
json.dumps(obj, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw)
- obj – сериализуемый объект;
- ensure_ascii – если равен False, запись не-ASCII значений происходит в файл «как есть», без преобразования в Unicode;
- indent – величина отступа для вложенных структур.

json.loads - Десериализует объект (в том числе файловый) s, возвращая структуру в Python.
Синтаксис:
json.loads(s, encoding=None, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw)
exceptionjson.JSONDecodeError(msg, doc, pos, end=None) - Класс исключения, возбуждаемый при ошибке в работе некоторых функций пакета.

Пример работы с json

```
import json

filename = "file.json"
```

```
info = {
  "ФИО": "Иванов Иван Иванович",
  "Оценки": {
    "Математика": 4,
    "Физика": 5,
    "Информатика": 5
  },
  "Хобби": ["Программирование", "Плавание"],
  "Возраст": 14,
  "ДомЖивотные": None
}

# Запись структуры в файл в JSON-формате
with open(filename, "w", encoding="utf-8") as file:
  file.write(json.dumps(info, ensure_ascii=False, indent=4))

# Чтение из файла JSON-формата
info_2 = []
with open(filename, encoding="utf-8") as file:
  info_2 = json.loads(file.read())

print(info_2)
```

CSV (от англ. Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми.
В Python работа с CSV-файлами поддерживается стандартным модулем csv
Основные методы:
csv.reader - Создает и возвращает объект для чтения последовательности из CSV-файла.
Синтаксис:
csv.reader(csvfile, dialect='excel', **fmtparams)

- csvfile – итерируемый объект, возвращающий строку на каждой итерации (например, файловый объект в текстовом режиме доступа);
- dialect – диалект CSV (набор специальных параметров);
- fmtparams – дополнительные настройки (совокупность кавычек, разделителей и т.д.).

csv.writer - создает и возвращает объект для записи последовательности в CSV-файл.
Синтаксис:
csv.writer(csvfile, dialect='excel', **fmtparams)

- csvfile – любой объект, поддерживающий метод записи write()
- dialect – аналогично csv.reader()
- fmtparams – аналогично csv.reader()

classcvs.DictReader - Создает и возвращает объект для чтения данных из CSV-файла как словаря значений.

Синтаксис:
classcsv.DictReader(csvfile, fieldnames=None, restkey=None, restval=None, dialect='excel', *args, **kwds)

- csvfile – итерируемый объект, возвращающий строку на каждой итерации (например, файловый объект в текстовом режиме доступа)
- fieldnames – список наименований столбцов (если не задан, используется первая строка файла).

classcsv.DictWriter - Создает и возвращает объект для записи данных как словаря значений в CSV-файл.
classcsv.DictWriter(csvfile, fieldnames, restval='', extrasaction='raise', dialect='excel', *args, **kwds)

- csvfile – любой объект, поддерживающий метод записи write()
- fieldnames – список наименований столбцов.

classcsv.Writer

- **writerow(row)** - Записывает последовательность row в CSV-файл.
- **writerows(rows)** - Записывает список последовательностей rows в CSV-файл.

classcsv.DictWriter

- **writeheader()** - Записывает в файл заголовки файла, переданные при создании класса.
- **writerow(row)** - Записывает словарь row в CSV-файл.
- **writerows(rows)** - Записывает список словарей rows в CSV-файл.

exceptioncsv.Error - Класс исключения, возбуждаемый при ошибке в работе любой из функций модуля.

Рассмотрим несколько вариантов работы с csv: построчно и запись в словарь
Построчно:

```
import csv

filename = "test.csv"

shop_list = {"картофель": [2, 100], "яблоки": [3, 250], "морковь": [1, 35]}

# Запись в файл
with open(filename, "w", encoding="utf-8", newline="") as file:
  writer = csv.writer(file, quoting=csv.QUOTE_ALL)
  writer.writerow(["Наименование", "Вес", "Цена/кг."])  # Заголовки столбца
  for name, values in sorted(shop_list.items()):
    writer.writerow([name, *values])
  writer.writerow(["мука", "4", "70"])  # Допишем произвольную запись

# Чтение файла
rows = []
with open(filename, "r", encoding="utf-8") as file:
  reader = csv.reader(file)
```

```
  rows = list(reader)  # reader - итерируемый объект и может быть преобразован в список строк

for row in rows:
  print(row)
```

В словарь

```
import csv

filename = "test.csv"
# список покупок
shop_list = {"картофель": [2, 100], "яблоки": [3, 250], "морковь": [1, 35]}

# Запись в файл
with open(filename, "w", encoding="utf-8", newline="") as file:
  writer = csv.DictWriter(file, fieldnames=["name", "weight", "price"], quoting=csv.QUOTE_ALL)
  writer.writeheader()  # Записывает заголовки в файл
  for name, values in sorted(shop_list.items()):
    writer.writerow(dict(name=name, weight=values[0], price=values[1]))

# Чтение файла
rows = []
with open(filename, "r", encoding="utf-8") as file:
  reader = csv.DictReader(file)
  rows = list(reader)  # reader - итерируемый объект и может быть преобразован в список строк

for row in rows:
  print(row)
```

Учитель: Чтение/запись простых типов (например, чисел или строк) не представляет большого труда, однако с увеличением объема информации появляется необходимость эффективно сохранять/загружать более сложные структуры данных (например, словари). Кроме того, очень часто нам необходимо обмениваться данными между разными модулями, а также между приложениями в целом, для чего необходимо иметь возможность удобно обмениваться данными.

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Десериализация, соответственно обратный процесс.
Чаще всего сериализация используется для сохранения объектов в файлы или передачи их по сети.

Одним из вариантов, позволяющий сериализовать/десериализовать данные в Python, является использование стандартного модуля pickle, при помощи которого можно сохранять любой объект Python в двоичном файле, а затем извлекать его обратно .

Рассмотрим основные возможности pickle.
pickle.dump Сериализует объект obj и записывает его в файл file.

Синтаксис

pickle.dump(obj, file, protocol=None, *, fix_imports=True)

- obj – объект для записи;
- file – файловый объект;
- protocol – версия формата pickle.

pickle.load - Читает и десериализует содержимое файла file, возвращая созданный объект

Синтаксис

pickle.load(file, *, fix_imports=True, encoding='ASCII', errors='strict')

- file – файловый объект.

Рассмотрим небольшой пример

```
import pickle

filename = "test.txt"

shop_list = {"овощи": ["картофель", "капуста"],
        "бакалея": ["мука"],
        "бюджет": 500}

# Запись в файл
with open(filename, "wb") as file:
  pickle.dump(shop_list, file)  # помещаем объект в файл

# Считываем из хранилища
shop_list_2 = []
with open(filename, "rb") as file:
  shop_list_2 = pickle.load(file)  # загружаем объект из файла
print(shop_list_2)
```

Если мы посмотрим содержимое текстового файла, мы увидим

Простота работы с модулем, является явным плюсом, но существуют и минусы использования pickle

- специфичен для Python (не может быть использован, если файл будет читаться с использованием других языков программирования);
- небезопасен (десериализация готовых конструкций языка может привести к выполнению ненадежного кода).

2. Решение задач

Задача 1

Написать программу, которая запрашивает у пользователя имя и возраст и записывает в словарь. Ввод продолжается до тех пор, пока количество пар(ключ/значение) в словаре не равно 3. Если пользователь ввел повторное имя, то программа должна вывести соответствующее сообщение. После окончания ввода данные записываются в файл test.txt в формате json.

Решение

```
import json

dict_person = {}
while len(dict_person) != 3:
  name = input('Введите имя: ')
  age = input('Введите возраст: ')
  if name not in dict_person:
     dict_person[name] = age
  else:
     print('Данное имя уже существует')

with open('text.txt', "w", encoding="utf-8") as file:
  file.write(json.dumps(dict_person, ensure_ascii=False, indent=4))
```

Дополнительно

Если на уроке остается время, то ученикам можно предложить начать прорешивать домашнее задание.

Домашняя работа

Задача 1

Написать программу список дел, которая спрашивает у пользователя значение n, после этого запрашивает на ввод n строк различных дел и сохраняет их в список, а после записывает значения из списка через одно в файл в одну строку

Решение

```
n = int(input())
todo_list = []
for i in range(n):
    todo_list.append(input())
with open('example5.txt', 'a') as file:
    for todo in range(0, len(todo_list), 2):
        file.write(todo_list[todo])
```