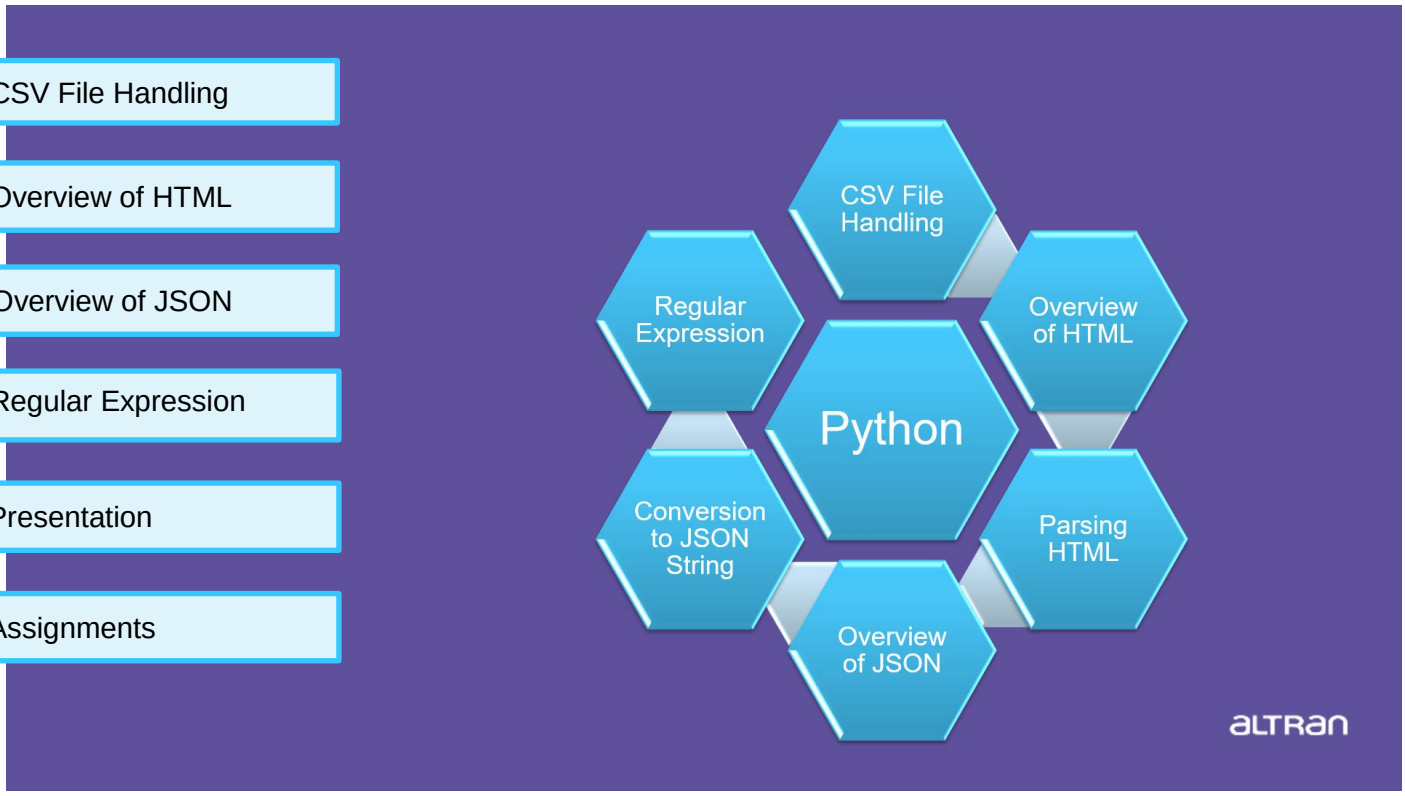

CSV File Reading and Writing

- CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases.
- The python csv module implements classes to read and write tabular data in CSV format.
- To read and write data in sequences the reader and writer functions are used.
- DictReader class can be used to create an object that operates like a regular reader but maps the information in each row to an OrderedDict whose keys are given by the optional *fieldnames* parameter.
- DictWriter class can be used to create an object which operates like a regular writer but maps dictionaries onto output rows.
- To make it easier to specify the format of input and output records, specific formatting parameters are grouped together into dialects.

To know more about CSV File Reading & Writing: - 

(to refer documentation)

Conceptual Video on CSV File Handling: - 

(to watch the video)

Reading data from csv file to a dictionary: -

```
import csv
INPUT_FILENAME = "people.csv"
def read_csv_todict(csv_filename):
    """reading csv to dict"""
    with open(csv_filename, newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            print(row['Name'], row['Email'], row['City'], row['Qualification'] )
def main():
    """main function"""
    read_csv_todict(INPUT_FILENAME)
if __name__ == "__main__":
    main()
```

people.csv

Points to note:

- DictReader is used here.
- By default, the values in the first row of file will be used as the fieldnames.
- In python the with keyword is used when working with unmanaged resources like open files. This ensures that a resource is "cleaned up" when the code that uses it finishes running, even if exceptions are thrown

Code to write data from a dictionary to CSV file: -

```
"""Writing data from a dictionary to csv file"""
import csv
OUTPUT_FILENAME = "output.csv"
def write_dict_to_csv(csv_filename, fields, rows):
    """Writing dictionary to csv"""

    with open(csv_filename, 'w', newline='') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fields)
        writer.writeheader()
        for row in rows:
            writer.writerow(row)
def main():
    """main function"""
    fields = ['Name', 'Email', 'City', 'Qualification']
    row1 = {'Name': 'Amitabh', 'Email': 'amitabh@world.com','City': 'Tokyo','Qualification': 'BA'}
    row2 = {'Name': 'Rajib', 'Email': 'rajib@world.com','City': 'Delhi','Qualification': 'BSC'}
    row3 = {'Name': 'Gaurav', 'Email': 'gaurav@world.com','City': 'Colombo','Qualification': 'PHD'}
    list_of_rows = [row1, row2, row3]
    write_dict_to_csv(OUTPUT_FILENAME, fields, list_of_rows)
    print("Data Written")
if __name__ == "__main__":
    main()
```

Points to note:

- DictWriter class is used here.
- Note that unlike the DictReader class, the *fieldnames* parameter of the DictWriter class is not optional.
- writeheader writes row with the field names (as specified in the constructor) of DictWriter.

Overview of HTML

What is HTML (Hypertext Mark-up Language)?

- HTML is a simple language made up of elements.
- HTML can be applied to pieces of text:
    - To give them different meaning in a document (Typically Webpage) Is it a paragraph? Is it a bulleted list? Is it part of a table?
    - Structure a document into logical sections (Does it have a header? Three columns of content? A navigation menu?),
    - Embed content such as images and videos into a web page.

What is an element?

- An element is a part of a webpage. In XML and HTML, an element may contain a data item or a chunk of text or an image, or perhaps nothing.

A typical element includes an opening tag with some attributes, enclosed text content, and a closing tag.

To know more about HTML: - 

(to refer documentation)

Parsing HTML in Python

Html.parser Module

- In Python standard library ***html.parser*** module defines a class HTMLParser which serves as the basis for parsing text files formatted in HTML and XHTML.
- An HTMLParser instance is fed HTML data and calls handler methods when start tags, end tags, text, comments, and other markup elements are encountered.

Programmer should subclass HTMLParser and override its methods to implement the desired behavior.

To know more about HTML Parser: - 

(to refer documentation)

A basic HTML Parser application

```
from html.parser import HTMLParser
class MyHTMLParser(HTMLParser):
   def handle_starttag(self, tag, attrs):
      print("Encountered a start tag:", tag)
   def handle_endtag(self, tag):
      print("Encountered an end tag :", tag)
   def handle_data(self, data):
      print("Encountered some data  :", data)
parser = MyHTMLParser()
parser.feed('<html><head><title>Test</title></head>'
        '<body><h1>Parse me!</h1></body></html>')
```

Points to note:

- A derived class MyHTMLParser is created from the base class HTMLParser.
- handle_starttag, handle_endtag and handle_data are methods of the base class which are overridden in this class.
- The handler methods are called automatically when html is given as input through feed method.
- HTML which is given as input must be string.
- The argument attrs in *handle_starttag(self, tag, attrs*) is a list of (name, value) pairs containing the attributes found inside the tag's <> brackets.

Brief Overview of JSON

What is JSON?

- JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax.
- It is commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).
- JSON is purely a data format — it contains only properties, no methods.
- JSON has six kind of values: objects (unordered container of name/value pairs), arrays (ordered sequence of values), string, numbers, Booleans (true and false), and the special value null.
- JSON requires double quotes to be used around strings and property names. single quotes are not valid.

To know more about JSON: - 

(to refer documentation)

Conceptual Video on JSON: - 

Encoding and Decoding JSON Through Python: -

To know more about JSON encoder & decoder module: - 

(to refer documentation)

- Functions load/loads decodes (deserialize) JSON document to Python Object, using the following table by default.
- Functions dump/dumps encodes (serialize) Python Object to JSON document using the following table by default.
- It is also possible to create custom encoders and decoders by extending JSONEncoder and JSONDecoder classes.

Pattern Matching using Regular Expression

What are regular expressions?

- Regular expressions are a powerful feature to match texts.
- These patterns are internally converted (by a regular expression engine) to recognizers (called finite automata).
- The Finite automata (basically a mathematical model, implemented by the programming language) can match strings conforming to a pattern

Some types of common patterns that we need to match are as follows,

- Matching a single character.
- Matching a group of characters.
- Matching repeating characters.
- Matching character(s) from the start of a given string.
- Matching character(s) from the end of a given string.
- Matching character(s) anywhere in the given string.

An example:

Create a regular expression to check if a given string is a *valid identifier* in a programming.

A valid identifier satisfies the following

- Can start only with _, or English alphabets (Uppercase or Lowercase)
- After the first character valid character(s) are _, number (0, ..9) or English alphabets (Uppercase or Lowercase)
- Length of the identifier can't exceed 31

The regular expression which satisfies all these will be *[_a-zA-Z][_a-zA-Z0-9]{0,30}*

re module in Python

The Python "re" module provides regular expression support

The general steps to use this module are

- Building the regular expression (it means forming the string representing the regular expression)
- Create a regular expression object using the “compile” method of the “re” module
- Invoke methods of this object to check if match(s) was/were found
- Get those “matched” strings using methods of this object.

These steps are followed in the example below:

Problem Statement:

Suppose we want to match only “Mumbai” or “muMbai” or “MuMbai” or “mumbai” out of total 64 combinations of the 6-character word “Mumbai” (every character can be in either case).

The regular expression for this will be [Mm]u[Mm]bai

The [] (square bracket) represents “character class”. It means either of the characters inside the brackets can be matched. So basically the above regular expression is a concatenation of 6 regular expressions that is ***[Mm] . a. [Mm].b.a.i***

To know more about Pattern Matching: - 

(to refer documentation)

```
"""Showing steps of using re module"""
import re
def main():
    """main function"""
    mystr = 'MUMBAI mumbai MuMbai Mumbai muMbai mumbaI MUmbai MuMBai mumbAi mUmbai'
    mypattern = '[Mm]u[Mm]bai'
    re_print_match(mystr, mypattern)

def re_print_match(mystr, mypattern):
    """Accepts a string and a pattern and prints the first matched component"""
    mypatobj = re.compile('[Mm]u[Mm]bai')
    mymatch = mypatobj.search(mystr)
    if mymatch:
        print('The regexpr. matched :{}'.format(mymatch.group()))
    else:
        print('No Match found')
if __name__ == "__main__":
    main()
```

Points to note:

- If the search is successful, search() returns a match object (which holds the first match) or “None” otherwise
- If the match was successful then matchobj.group() returns the matched text
- In the above example the output will be the first match that is “mumbai” (All the strings that could have been matched won’t be available).

Conceptual Video on Pattern Matching: - 

More Concepts: -

Request Library Web Pages, download images, POST Data, Read JSON	**Click Here**
Web Scrapping with Requests- HTML	**Click Here**
Web Scrapping with BeautifulSoup and Requests	**Click Here**

Presentation 

Assignments