Module-4

Maheshkumar N

Module-4

- **Files:** Types of Files, Creating and Reading Text Data, File Methods to Read and Write Data, Reading and Writing Binary Files, The Pickle Module, Reading and Writing CSV Files,
- **Regular Expression Operations:** Using Special Characters, Regular Expression Methods, Named Groups in Python Regular Expressions, Regular Expression with glob Module.

Files:

- In everyday life, the term “file” is something of a catchall.
- It is used for things that are not only written, but also used to describe things that don’t have any words in them at all, like pictures.
- A file is the common storage unit in a computer, and all programs and data are “written” into a file and “read” from a file.
- A file extension, sometimes called a file suffix or a filename extension, is the character or group of characters after the period that makes up an entire file name.
- The file extension helps your computer’s operating system, like Windows, determine which program will be associated with the file.
- For example, the file assignments.docx ends in docx, a file extension that is associated with Microsoft Word on your computer

Files:

- File extensions also often indicate the file type, or file format, of the file but not always.
- Any file's extensions can be renamed, but that will not convert the file to another format or change anything about the file other than this portion of its name.
- File extensions and file formats are often spoken about interchangeably.
- However, the file extension is just whatever characters are after the period, while the file format illustrates the way in which the data in the file is organized and what sort of file it is.
- For example, in the file name pop.csv, the file extension csv indicates that this is a CSV file.

Files:

- All the data on your hard drive consists of files and directories.
- The fundamental difference between the two is that files store data, while directories store files and other directories.
- The folders, often referred to as directories, are used to organize files on your computer.
- The directories themselves take up virtually no space on the hard drive.
- Files, on the other hand, can range from a few bytes to several gigabytes.
- Directories are used to organize files on your computer

Files:

- Python supports two types of files – text files and binary files.
- These two file types may look the same on the surface but they encode data differently.
- While both binary and text files contain data stored as a series of bits (binary values of 1s and 0s), the bits in text files represent characters, while the bits in binary files represent custom data.
- Binary files typically contain a sequence of bytes or ordered groupings of eight bits.
- When creating a custom file format for a program, a developer arranges these bytes into a format that stores the necessary information for the application.
- Binary file formats may include multiple types of data in the same file, such as image, video, and audio data.
- This data can be interpreted by supporting programs but will show up as garbled text in a text editor.

Files:

- Text files are more restrictive than binary files since they can only contain textual data.
- However, unlike binary files, they are less likely to become corrupted.
- While a small error in a binary file may make it unreadable, a small error in a text file may simply show up once the file has been opened.
- A typical plain text file contains several lines of text that are each followed by an End-of-Line (EOL) character.
- An End-of-File (EOF) marker is placed after the final character, which signals the end of the file.
- Text files include a character encoding scheme that determines how the characters are interpreted and what characters can be displayed.
- Since text files use a simple, standard format, many programs are capable of reading and editing text files. Common text editors include Microsoft Notepad and WordPad, which are bundled with Windows, and Apple TextEdit, which is included with Mac OS X.

File Paths:

- All operating systems follow the same general naming conventions for an individual file: a base file name and an optional extension, separated by a period.
- Note that a directory is simply a file with a special attribute designating it as a directory, but otherwise must follow all the same naming rules as a regular file.
- To make use of files, you have to provide a file path, which is basically a route so that the user or the program knows where the file is located.
- The path to a specified file consists of one or more components, separated by a special character (a backslash for Windows and forward slash for Linux), with each component usually being a directory name or file name, and possibly a volume name or drive name in Windows or root in Linux.
- If a component of a path is a file name, it must be the last component.

Valid Names for files and directories:

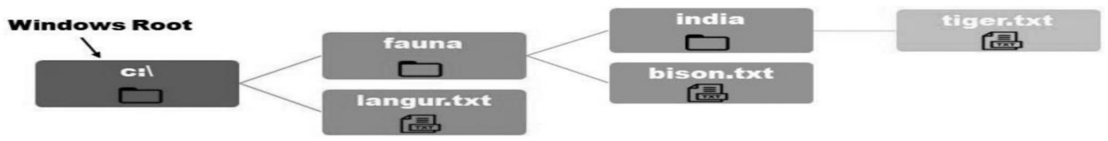
- The following fundamental rules enable applications to create and process valid names for files and directories in both Windows and Linux operating systems unless explicitly specified:
- Use a period to separate the base file name from the extension in the file name.
- In Windows use backslash (\) and in Linux use forward slash (/) to separate the components of a path. The backslash (or forward slash) separates one directory name from another directory name in a path and it also divides the file name from the path leading to it.
- Backslash (\) and forward slash (/) are reserved characters and you cannot use them in the name for the actual file or directory.
- Do not assume case sensitivity. File and Directory names in Windows are not case sensitive while in Linux it is case sensitive. For example, the directory names ORANGE, Orange, and orange are the same in Windows but are different in Linux Operating System.
- In Windows, volume designators (drive letters) are case-insensitive. For example, "D:\" and "d:\" refer to the same drive.
- The reserved characters that should not be used in naming files and directories are < (less than), > (greater than),: (colon), " (double quote), / (forward slash), \ (backslash), | (vertical bar or pipe), ? (question mark) and * (asterisk).
- In Windows Operating system reserved words like CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9 should not be used to name files and directories

Fully Qualified Path:

- A file path can be a fully qualified path name or relative path.
- The fully qualified path name is also called an Absolute path.
- A path is said to be a fully qualified path if it points to the file location, which always contains the root and the complete directory list.
- The current directory is the directory in which a user is working at a given time.
- Every user is always working within a directory.
- Note that the current directory may or may not be the root directory depending on what it was set to during the most recent “change directory” operation on that disk.
- The root directory, sometimes just called as root is the "highest" directory in the hierarchy.
- You can also think of it in general as the start or beginning of a particular directory structure.
- The root directory contains all other directories in the drive and can of course also contain files.
- For example, the root directory of the main partition on your Windows system will be C:\ and the root directory on your Linux system will be / (forward slash).
- Examples of a fully qualified path are given below.
- "C:\langur.txt" refers to a file named "langur.txt" under root directory C:\.
- "C:\fauna\bison.txt" refers to a file named "bison.txt" in a subdirectory fauna under
- root directory C:\.

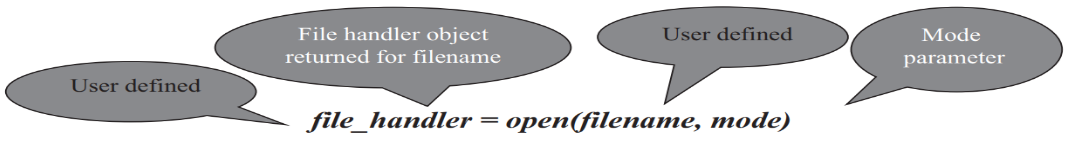
Relative Path:

- A path is also said to be a relative path if it contains “double-dots”; that is, two consecutive periods together as one of the directory components in a path or “single-dot”; that is, one period as one of the directory components in a path. These two consecutive periods are
- used to denote the directory above the current directory, otherwise known as the “parent
- directory.” Use a single period as a directory component in a path to represent the current
- directory.
- Examples of the relative path are given below:
- • "..\langur.txt" specifies a file named "langur.txt" located in the parent of the current
- directory fauna.
- • ".\bison.txt" specifies a file named "bison.txt" located in a current directory named fauna.
- • "..\..\langur.txt" specifies a file that is two directories above the current directory india



Creating and Opening Text Files:

- Files are not very useful unless you can access the information they contain.
- All files must be opened first before they can be read from or written to using the Python's built-in open() function.
- When a file is opened using open() function, it returns a file object called a file handler that provides methods for accessing the file.
- The syntax of open() function is given below


File handler object returned for filename
User defined
User defined
Mode parameter
file_handler = open(filename, mode)

- The open() function returns a file handler object for the file name.
- The open() function is commonly used with two arguments, where the first argument is a string containing the file name to be opened which can be absolute or relative to the current working directory.
- The second argument is another string containing a few characters describing the way in which the file will be used as shown in the next slide.

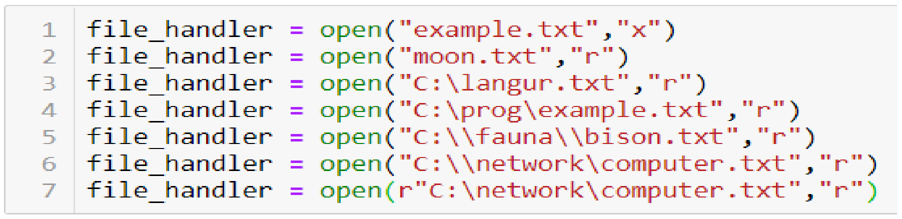
Access Modes of the Files:

Access Modes of the Files

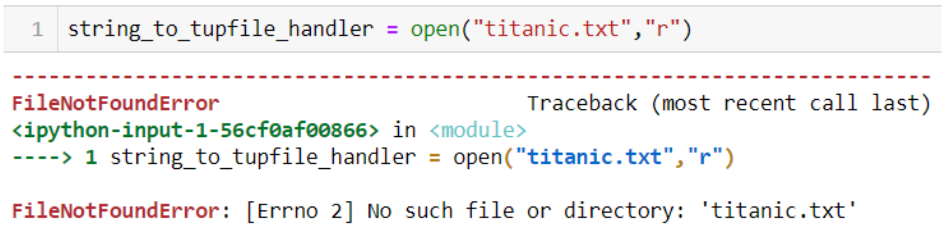
Mode	Description
“r”	Opens the file in read only mode and this is the default mode.
“w”	Opens the file for writing. If a file already exists, then it’ll get overwritten. If the file does not exist, then it creates a new file.
“a”	Opens the file for appending data at the end of the file automatically. If the file does not exist it creates a new file.
“r+”	Opens the file for both reading and writing.
“w+”	Opens the file for reading and writing. If the file does not exist it creates a new file. If a file already exists then it will get overwritten.
“a+”	Opens the file for reading and appending. If a file already exists, the data is appended. If the file does not exist it creates a new file.
“x”	Creates a new file. If the file already exists, the operation fails.
“rb”	Opens the binary file in read-only mode.
“wb”	Opens the file for writing the data in binary format.
“rb+”	Opens the file for both reading and writing in binary format.

Access Modes of the Files:

- Examples:

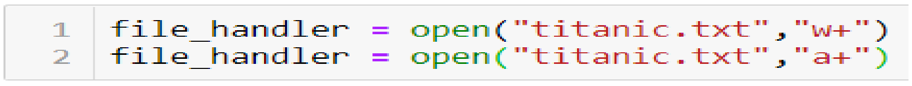


```
file_handler = open("example.txt","x")
file_handler = open("moon.txt","r")
file_handler = open("C:\langur.txt","r")
file_handler = open("C:\prog\example.txt","r")
file_handler = open("C:\\fauna\\bison.txt","r")
file_handler = open("C:\\network\computer.txt","r")
file_handler = open(r"C:\network\computer.txt","r")
```



```
string_to_tupfile_handler = open("titanic.txt","r")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-1-56cf0af00866> in <module>
----> 1 string_to_tupfile_handler = open("titanic.txt","r")

FileNotFoundError: [Errno 2] No such file or directory: 'titanic.txt'
```



```
file_handler = open("titanic.txt","w+")
file_handler = open("titanic.txt","a+")
```

Access Modes of the Files:

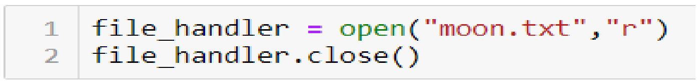
- Examples:
- Python represents backslashes in strings as \\ because the backslash is an escape character—for instance, \n represents a new line, \t represents a tab, \f represents ASCII Formfeed, and \b represents ASCII Backspace.
- Because of this, there needs to be a way to tell Python you really want two characters of \f and \b rather than Form feed and Backspace, and you do that by escaping the backslash itself with another one.

File close() Method:

- Opening files consume system resources, and, depending on the file mode, other programs may not be able to access them.
- It is important to close the file once the processing is completed.
- After the file handler object is closed, you cannot further read or write from the file.
- Any attempt to use the file handler object after being closed will result in an error.
- The syntax for close() function is,

file_handler.close()

- For example,

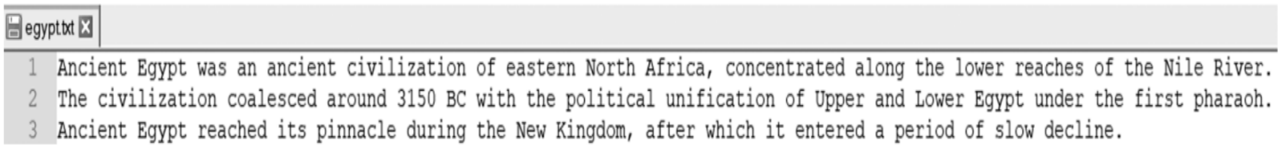


```python
file_handler = open("moon.txt","r")
file_handler.close()
```

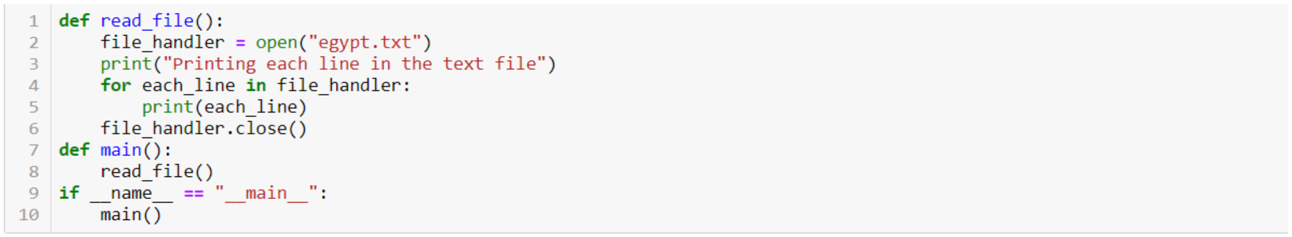
- You should call file_handler.close() to close the file

Program:

- Write Python Program to Read and Print Each Line in "egypt. txt" file. Sample Content of "egypt.txt" File is Given Below.



```
Ancient Egypt was an ancient civilization of eastern North Africa, concentrated along the lower reaches of the Nile River.
The civilization coalesced around 3150 BC with the political unification of Upper and Lower Egypt under the first pharaoh.
Ancient Egypt reached its pinnacle during the New Kingdom, after which it entered a period of slow decline.
```



```python
def read_file():
    file_handler = open("egypt.txt")
    print("Printing each line in the text file")
    for each_line in file_handler:
        print(each_line)
    file_handler.close()
def main():
    read_file()
if __name__ == "__main__":
    main()
```

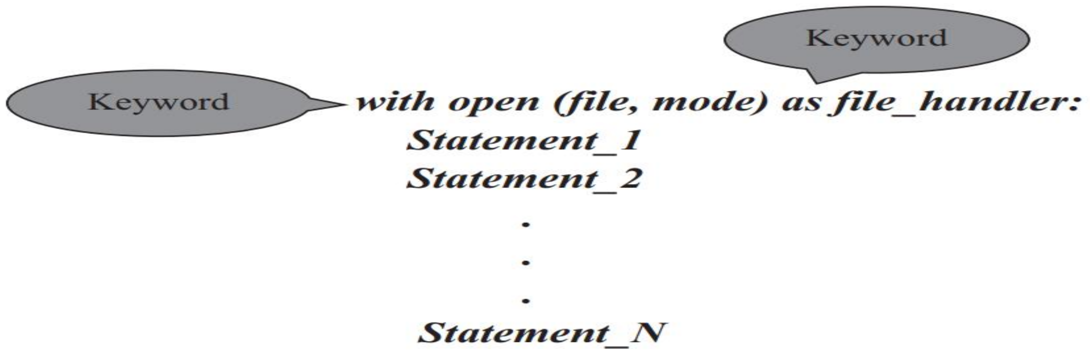
```
Printing each line in the text file
Ancient Egypt was an ancient civilization of eastern North Africa, concentrated along the lower reaches of the Nile River.

The civilization coalesced around 3150 BC with the political unification of Upper and Lower Egypt under the first pharaoh.

Ancient Egypt reached its pinnacle during the New Kingdom, after which it entered a period of slow decline.
```

Program:

- In the output, notice a blank space between each line of the file.
- Understand that at the end of each line, a newline character (\n) is present which is invisible and it indicates the end of the line.
- The print() function by default always appends a newline character.
- This means that if you want to print data that already ends in a newline, we get two newlines, resulting in a blank space between the lines.
- In order to overcome this problem, pass an end argument to the print() function and initialize it with an empty string (with no spaces).
- So, changing line ⑤ as print(each_line, end="") removes the blank spaces between the lines in the output

Use of with Statements to Open and Close Files:

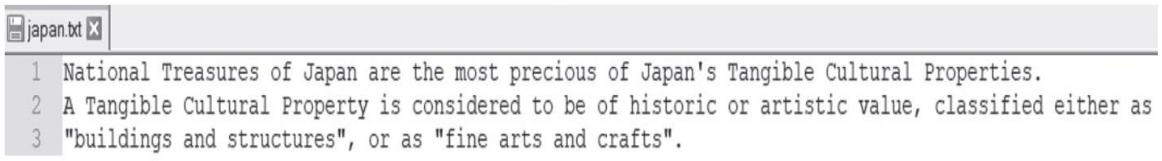
- Instead of using try-except-finally blocks to handle file opening and closing operations, a much cleaner way of doing this in Python is using the with statement.
- You can use a with statement in Python such that you do not have to close the file handler object.
- The syntax of the with statement for the file I/O is,



```
with open (file, mode) as file_handler:
    Statement_1
    Statement_2
        .
        .
        .
    Statement_N
```

Program:

- Program to Read and Print Each Line in "japan.txt" File Using with Statement. Sample Content of "japan.txt" File is Given Below.



```
National Treasures of Japan are the most precious of Japan's Tangible Cultural Properties.
A Tangible Cultural Property is considered to be of historic or artistic value, classified either as
"buildings and structures", or as "fine arts and crafts".
```

```python
def read_file():
    print("Printing each line in text file")
    with open("japan.txt") as file_handler:
        for each_line in file_handler:
            print(each_line, end="")
def main():
    read_file()
if __name__ == "__main__":
    main()
```

```
Printing each line in text file
National Treasures of Japan are the most precious of Japan's Tangible Cultural Properties.
A Tangible Cultural Property is considered to be of historic or artistic value, classified either as
"buildings and structures", or as "fine arts and crafts".
```

File Object Attributes:

- When the Python open() function is called, it returns a file object called a file handler.
- Using this file handler, you can retrieve information about various file attributes

List of File Attributes

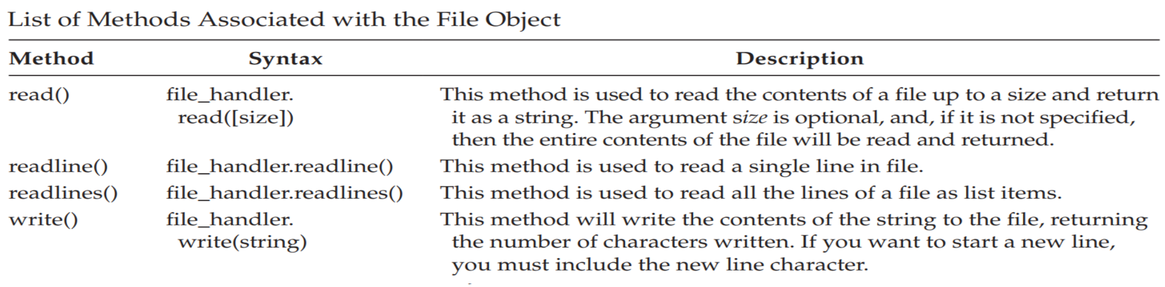
Attribute	Description
file_handler.closed	It returns a Boolean True if the file is closed or False otherwise.
file_handler.mode	It returns the access mode with which the file was opened.
file_handler.name	It returns the name of the file.

```python
file_handler = open("computer.txt", "w")
print(f"File Name is {file_handler.name}")
print(f"File State is {file_handler.closed}")
print(f"File Opening Mode is {file_handler.mode}")
```

```
File Name is computer.txt
File State is False
File Opening Mode is w
```

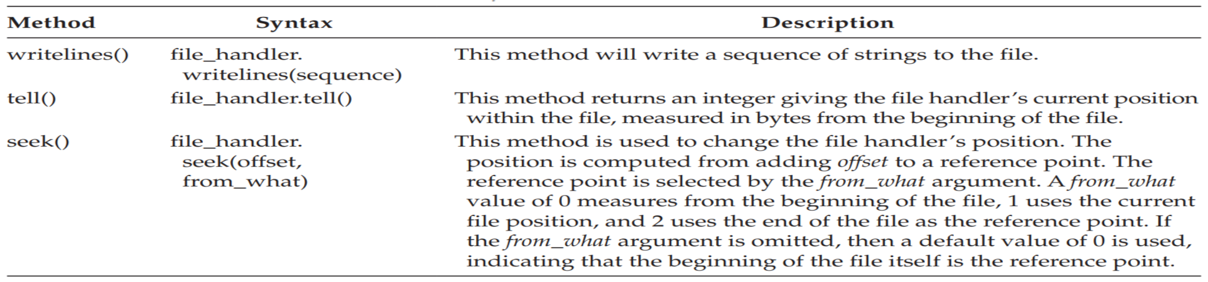
File Methods to Read and Write Data:

- When you use the open() function a file object is created.
- Here is the list of methods that can be called on this object



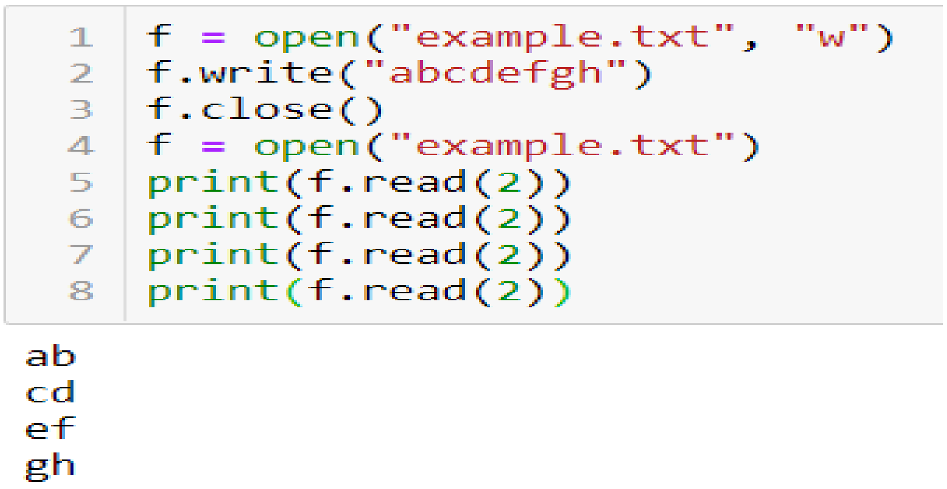
List of Methods Associated with the File Object

Method	Syntax	Description
read()	file_handler. read([size])	This method is used to read the contents of a file up to a size and return it as a string. The argument *size* is optional, and, if it is not specified, then the entire contents of the file will be read and returned.
readline()	file_handler.readline()	This method is used to read a single line in file.
readlines()	file_handler.readlines()	This method is used to read all the lines of a file as list items.
write()	file_handler. write(string)	This method will write the contents of the string to the file, returning the number of characters written. If you want to start a new line, you must include the new line character.



Method	Syntax	Description
writelines()	file_handler. writelines(sequence)	This method will write a sequence of strings to the file.
tell()	file_handler.tell()	This method returns an integer giving the file handler's current position within the file, measured in bytes from the beginning of the file.
seek()	file_handler. seek(offset, from_what)	This method is used to change the file handler's position. The position is computed from adding *offset* to a reference point. The reference point is selected by the *from_what* argument. A *from_what* value of 0 measures from the beginning of the file, 1 uses the current file position, and 2 uses the end of the file as the reference point. If the *from_what* argument is omitted, then a default value of 0 is used, indicating that the beginning of the file itself is the reference point.

File Methods Example:

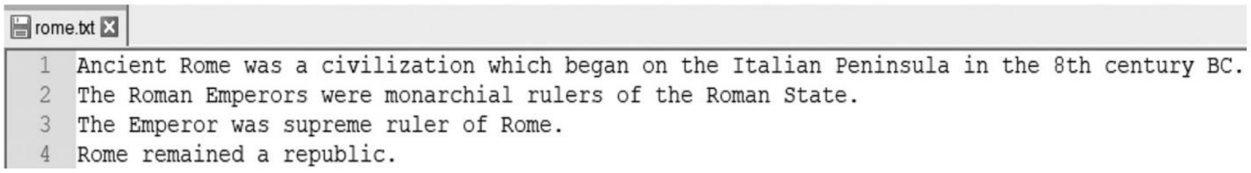


```python
f = open("example.txt", "w")
f.write("abcdefgh")
f.close()
f = open("example.txt")
print(f.read(2))
print(f.read(2))
print(f.read(2))
print(f.read(2))
```

```
ab
cd
ef
gh
```

Program:

- Write Python Program to Read "rome.txt" File Using read() Method. Sample Content of "rome.txt" File is Given Below



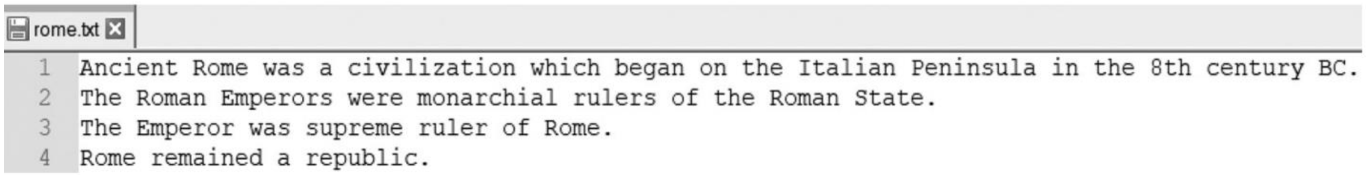
```
Ancient Rome was a civilization which began on the Italian Peninsula in the 8th century BC.
The Roman Emperors were monarchial rulers of the Roman State.
The Emperor was supreme ruler of Rome.
Rome remained a republic.
```

```python
def main():
    with open("rome.txt") as file_handler:
        print("Print entire file contents")
        print(file_handler.read(), end=" ")
if __name__ == "__main__":
    main()
```

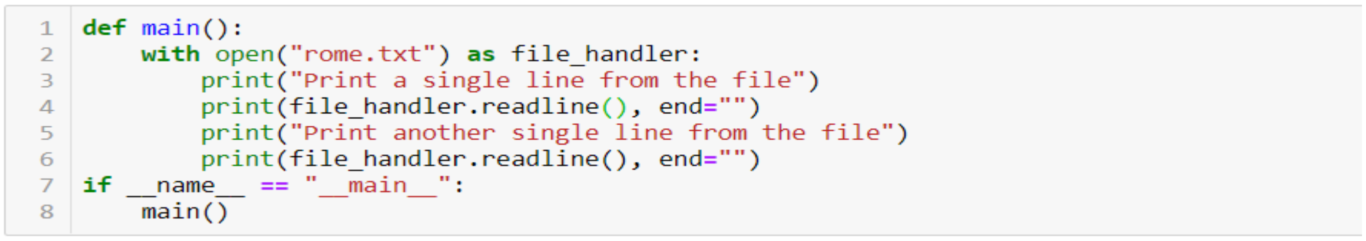
```
Print entire file contents
Ancient Rome was a civilization which began on the Italian Peninsula in the 8th century BC.
The Roman Emperors were monarchial rulers of the Roman State.
The Emperor was supreme ruler of Rome.
Rome remained a republic.
```

Program:

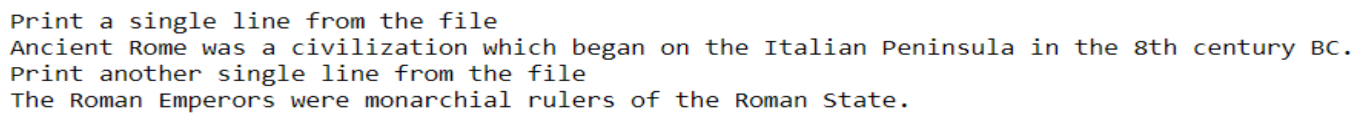
- Consider the "rome.txt" File. Write Python Program to Read "rome.txt" file Using readline() Method.



```
Ancient Rome was a civilization which began on the Italian Peninsula in the 8th century BC.
The Roman Emperors were monarchial rulers of the Roman State.
The Emperor was supreme ruler of Rome.
Rome remained a republic.
```



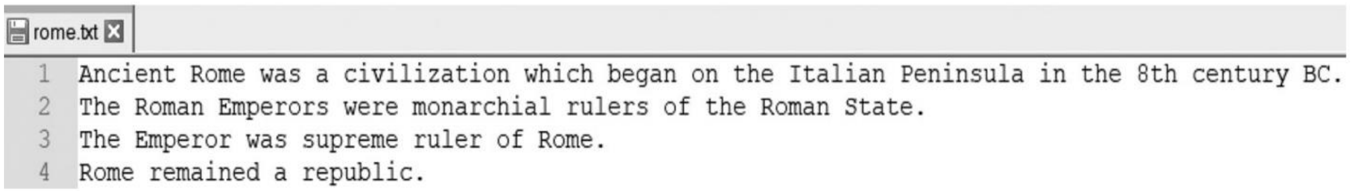
```python
def main():
    with open("rome.txt") as file_handler:
        print("Print a single line from the file")
        print(file_handler.readline(), end="")
        print("Print another single line from the file")
        print(file_handler.readline(), end="")
if __name__ == "__main__":
    main()
```



```
Print a single line from the file
Ancient Rome was a civilization which began on the Italian Peninsula in the 8th century BC.
Print another single line from the file
The Roman Emperors were monarchial rulers of the Roman State.
```

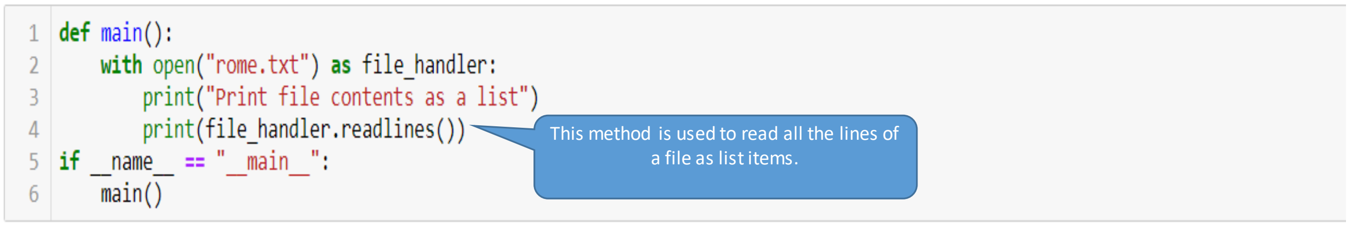
Program:

- Consider the "rome.txt" File. Write Python Program to Read "rome.txt" File Using readlines() Method.



rome.txt

```
Ancient Rome was a civilization which began on the Italian Peninsula in the 8th century BC.
The Roman Emperors were monarchial rulers of the Roman State.
The Emperor was supreme ruler of Rome.
Rome remained a republic.
```

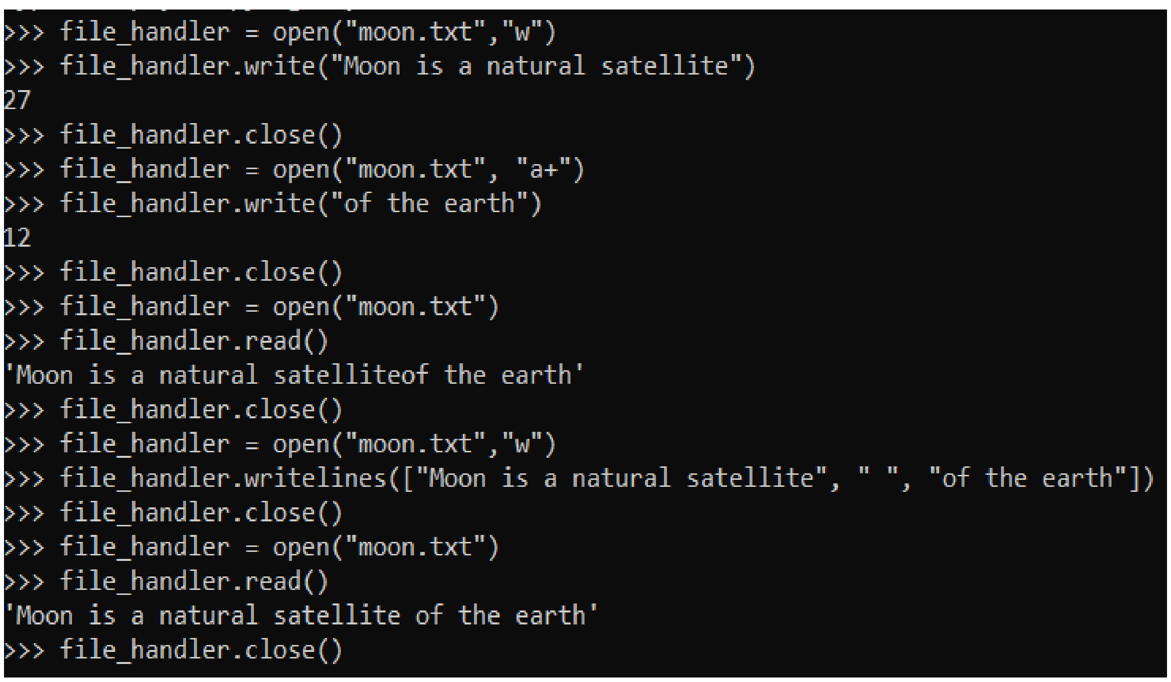


```python
def main():
    with open("rome.txt") as file_handler:
        print("Print file contents as a list")
        print(file_handler.readlines())
if __name__ == "__main__":
    main()
```

This method is used to read all the lines of a file as list items.

```
Print file contents as a list
['Ancient Rome was a civilization which began on the Italian Peninsula in the 8th century BC.\n', 'The Roman Emperors were mona
rchial rulers of the Roman State.\n', 'The Emperor was supreme ruler of Rome.\n', 'Rome remained a republic.']
```

write() method:



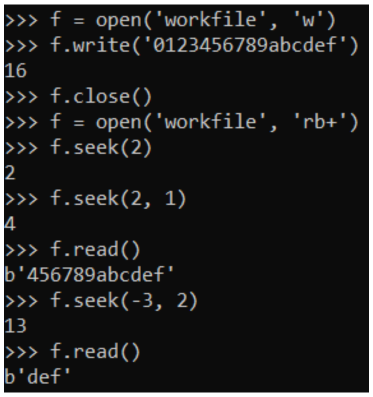
```
>>> file_handler = open("moon.txt","w")
>>> file_handler.write("Moon is a natural satellite")
27
>>> file_handler.close()
>>> file_handler = open("moon.txt", "a+")
>>> file_handler.write("of the earth")
12
>>> file_handler.close()
>>> file_handler = open("moon.txt")
>>> file_handler.read()
'Moon is a natural satelliteof the earth'
>>> file_handler.close()
>>> file_handler = open("moon.txt","w")
>>> file_handler.writelines(["Moon is a natural satellite", " ", "of the earth"])
>>> file_handler.close()
>>> file_handler = open("moon.txt")
>>> file_handler.read()
'Moon is a natural satellite of the earth'
>>> file_handler.close()
```

The seek() an tell() method:

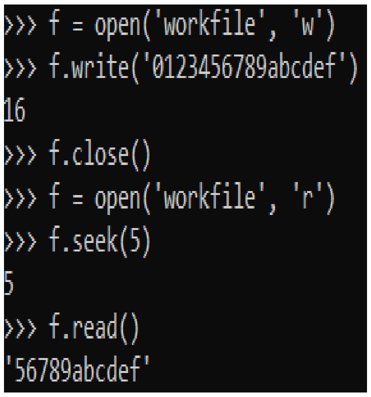
tell()	file_handler.tell()	This method returns an integer giving the file handler's current position within the file, measured in bytes from the beginning of the file.
seek()	file_handler. seek(offset, from_what)	This method is used to change the file handler's position. The position is computed from adding *offset* to a reference point. The reference point is selected by the *from_what* argument. A *from_what* value of 0 measures from the beginning of the file, 1 uses the current file position, and 2 uses the end of the file as the reference point. If the *from_what* argument is omitted, then a default value of 0 is used, indicating that the beginning of the file itself is the reference point.

The seek() method:

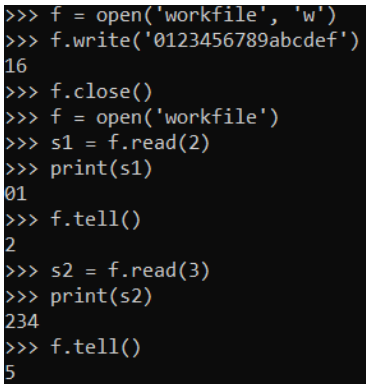
- The seek() method is used to set the file handler’s current position.
- Never forget that when managing files, there’ll always be a position inside that file where you are currently working on.
- When you open a file, that position is the beginning of the file, but as you work with it, you may advance.
- The seek() method will be useful to you when you need to work with that open file.
- The tell() method returns the file handler’s current position.



```
>>> f = open('workfile', 'w')
>>> f.write('0123456789abcdef')
16
>>> f.close()
>>> f = open('workfile', 'rb+')
>>> f.seek(2)
2
>>> f.seek(2, 1)
4
>>> f.read()
b'456789abcdef'
>>> f.seek(-3, 2)
13
>>> f.read()
b'def'
```



```
>>> f = open('workfile', 'w')
>>> f.write('0123456789abcdef')
16
>>> f.close()
>>> f = open('workfile', 'r')
>>> f.seek(5)
5
>>> f.read()
'56789abcdef'
```



```
>>> f = open('workfile', 'w')
>>> f.write('0123456789abcdef')
16
>>> f.close()
>>> f = open('workfile')
>>> s1 = f.read(2)
>>> print(s1)
01
>>> f.tell()
2
>>> s2 = f.read(3)
>>> print(s2)
234
>>> f.tell()
5
```

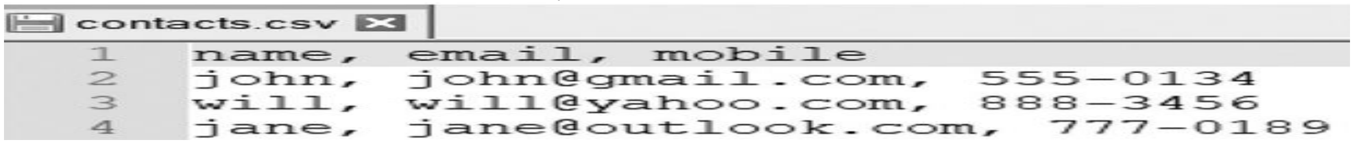
Reading and Writing Binary Files:

- We can usually tell whether a file is binary or text based on its file extension.
- This is because by convention the extension reflects the file format, and it is ultimately the file format that dictates whether the file data is binary or text.
- The string 'b' appended to the mode opens the file in binary mode and now the data is read and written in the form of bytes objects.
- **Write Python Program to Create a New Image from an Existing Image**

```python
def main():
    with open("rose.jpg", "rb") as existing_image, open(
        "new_rose.jpg", "wb"
    ) as new_image:
        for each_line_bytes in existing_image:
            new_image.write(each_line_bytes)


if __name__ == "__main__":
    main()
```

CSV Files:

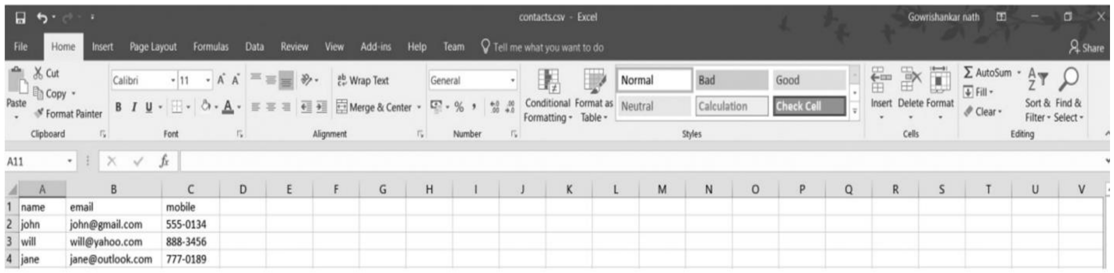
- CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases.
- Since a comma is used to separate the values, this file format is aptly named Comma Separated Values. CSV files have .csv extensions
- Consider the "contacts.csv" file, which when opened in a text editor, the CSV file looks like this



contacts.csv

```
name, email, mobile
john, john@gmail.com, 555-0134
will, will@yahoo.com, 888-3456
jane, jane@outlook.com, 777-0189
```

- Opened in Excel, our example CSV file "contacts.csv" looks like this



	A	B	C
1	name	email	mobile
2	john	john@gmail.com	555-0134
3	will	will@yahoo.com	888-3456
4	jane	jane@outlook.com	777-0189

Reading and Writing CSV Files:

- To read from a CSV file use csv.reader() method. The syntax is,

```
csv.reader(csvfile)
```

- where csv is the module name and csvfile is the file object.
- To write to a CSV file, use the csv.writer() method. The syntax is,

```
csv.writer(csvfile)
```

- where csv is the module name and csvfile is the file object.
- The syntax for writerow() method is,

```
csvwriter.writerow(row)
```

Reading and Writing CSV Files:

- where the csvwriter is the object returned by the writer() method and writerow() method will write the row argument to the writer() method's file object.
- The syntax for writerows() method is,

csvwriter.writerows(rows)

- Here, the writerows() method will write all the rows argument (a list of row objects) to the writer() method's file object.
- Programmers can also read and write data in dictionary format using the DictReader and DictWriter classes, which makes the data much easier to work with.
- The syntax for DictWriter is,

class csv. DictWriter(f, fieldnames, extrasaction='raise')

Reading and Writing CSV Files:

- The syntax for DictReader is,

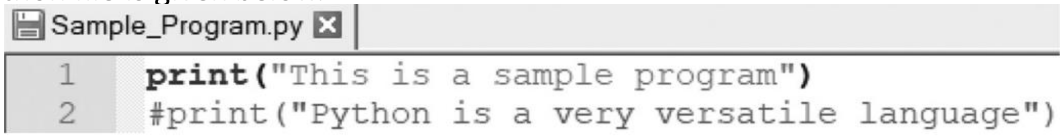
class csv. DictReader(f, fieldnames=None, restkey = None)

- The syntax for DictWriter is,

class csv. DictWriter(f, fieldnames, extrasaction=‘raise’)

Program:

- Consider "Sample_Program.py" Python file. Write Python program to remove the comment character from all the lines in a given Python source file. Sample content of "Sample_Program.py" Python file is given below.



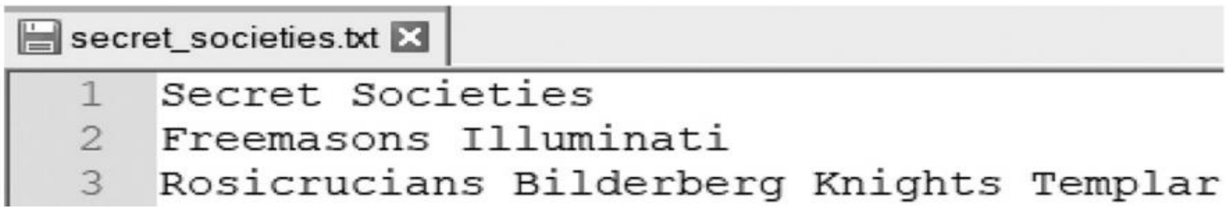
```
print("This is a sample program")
#print("Python is a very versatile language")
```

```python
def main():
    with open("Sample_Program.py") as file_handler:
        for each_row in file_handler:
            each_row = each_row.replace("#", "")
            print(each_row, end="")
if __name__ == "__main__":
    main()
```

```
print("This is a sample program")
print("Python is a very versatile language")
```

Program:

- Write Python Program to Reverse Each Word in "secret_societies.txt" file. Sample Content of "secret_societies.txt" is Given Below.



secret_societies.txt

```
Secret Societies
Freemasons Illuminati
Rosicrucians Bilderberg Knights Templar
```

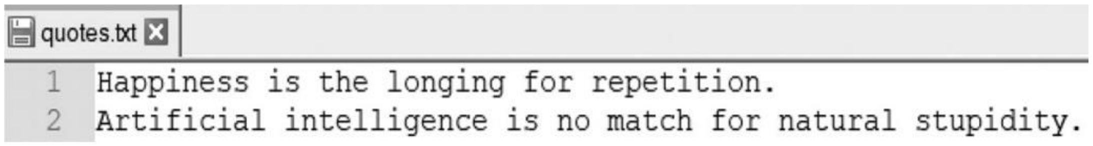
- Write a program to get a following result

- Expected result

```
terceS seiteicoS
snosameerF itanimullI
snaicurcisoR grebredliB sthginK ralpmeT
```

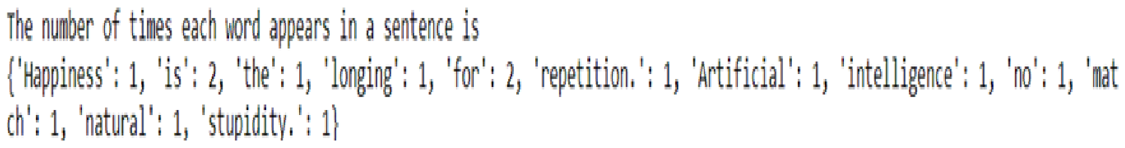
Program:

- Write Python Program to Count the Occurrences of Each Word and Also Count the Number of Words in a "quotes.txt" File. Sample Content of "quotes.txt" File is Given Below



```
Happiness is the longing for repetition.
Artificial intelligence is no match for natural stupidity.
```

- Write a program to get a following result
- Expected result



```
The number of times each word appears in a sentence is
{'Happiness': 1, 'is': 2, 'the': 1, 'longing': 1, 'for': 2, 'repetition.': 1, 'Artificial': 1, 'intelligence': 1, 'no': 1, 'mat
ch': 1, 'natural': 1, 'stupidity.': 1}
```

Program:

- Write Python Program to Find the Longest Word in a File. Get the File Name from User. (Assume User Enters the File Name as "animals.txt" and its Sample Contents are as Below)

```python
def read_file(file_name):
    with open(file_name) as file_handler:
        longest_word = ""
        for each_row in file_handler:
            word_list = each_row.rstrip().split()
            for each_word in word_list:
                if len(each_word) > len(longest_word):
                    longest_word = each_word
    print(f"The longest word in the file is {longest_word}")


def main():
    file_name = input("Enter file name: ")
    read_file(file_name)


if __name__ == "__main__":
    main()
```

```
Enter file name: animals.txt
The longest word in the file is Rhinocerose
```

Program:

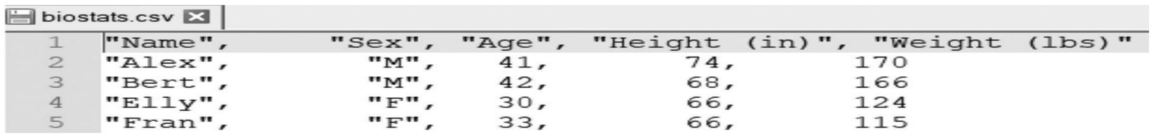
- Consider a File Called "workfile". Write Python Program to Read and Print Each Byte in the Binary File.

```python
def main():
    with open("workfile", "wb") as f:
        f.write(b"abcdef")
    with open("workfile", "rb") as f:
        byte = f.read(1)
        print("Print each byte in the file")
        while byte:
            print(byte)
            byte = f.read(1)


if __name__ == "__main__":
    main()
```

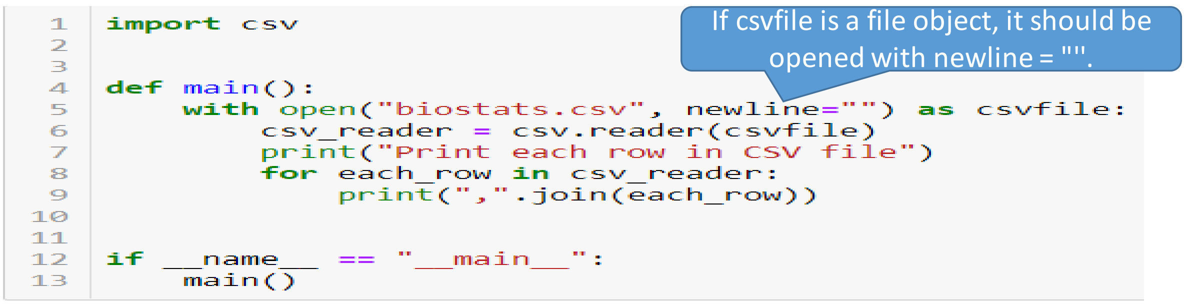
```
Print each byte in the file
b'a'
b'b'
b'c'
b'd'
b'e'
b'f'
```

Program:

- Write Python program to read and display each row in "biostats.csv" CSV file. Sample content of "biostats.csv" is given below.

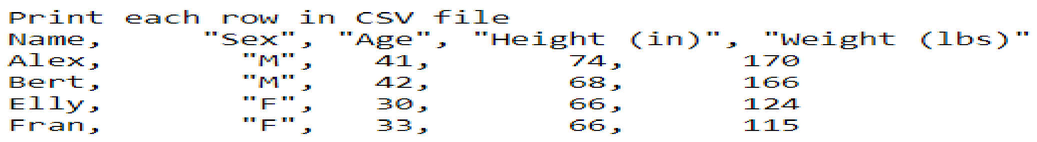


```
"Name",     "Sex", "Age", "Height (in)", "Weight (lbs)"
"Alex",       "M",   41,       74,      170
"Bert",       "M",   42,       68,      166
"Elly",       "F",   30,       66,      124
"Fran",       "F",   33,       66,      115
```



If csvfile is a file object, it should be opened with newline = "".

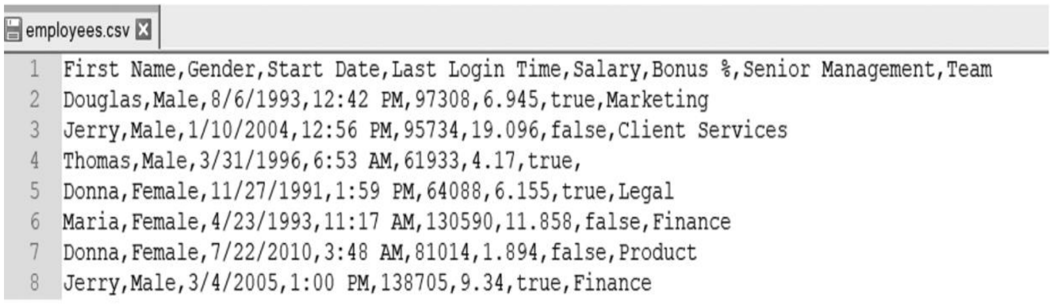
```python
import csv


def main():
    with open("biostats.csv", newline="") as csvfile:
        csv_reader = csv.reader(csvfile)
        print("Print each row in CSV file")
        for each_row in csv_reader:
            print(",".join(each_row))


if __name__ == "__main__":
    main()
```



```
Print each row in CSV file
Name,     "Sex", "Age", "Height (in)", "Weight (lbs)"
Alex,       "M",   41,       74,      170
Bert,       "M",   42,       68,      166
Elly,       "F",   30,       66,      124
Fran,       "F",   33,       66,      115
```

Program:

- Write Python program to read and display rows in "employees.csv" CSV file that start with employee name "Jerry". Sample content of "employees.csv" is given below.

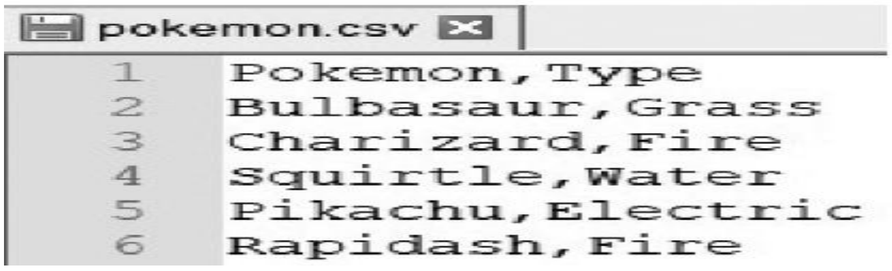


```
First Name,Gender,Start Date,Last Login Time,Salary,Bonus %,Senior Management,Team
Douglas,Male,8/6/1993,12:42 PM,97308,6.945,true,Marketing
Jerry,Male,1/10/2004,12:56 PM,95734,19.096,false,Client Services
Thomas,Male,3/31/1996,6:53 AM,61933,4.17,true,
Donna,Female,11/27/1991,1:59 PM,64088,6.155,true,Legal
Maria,Female,4/23/1993,11:17 AM,130590,11.858,false,Finance
Donna,Female,7/22/2010,3:48 AM,81014,1.894,false,Product
Jerry,Male,3/4/2005,1:00 PM,138705,9.34,true,Finance
```

- Write a program to get a following result
- Expected result

Program:

- Write Python Program to Read Data from "pokemon.csv" csv File Using DictReader. Sample Content of "pokemon.csv" is Given Below


pokemon.csv

```
Pokemon,Type
Bulbasaur,Grass
Charizard,Fire
Squirtle,Water
Pikachu,Electric
Rapidash,Fire
```

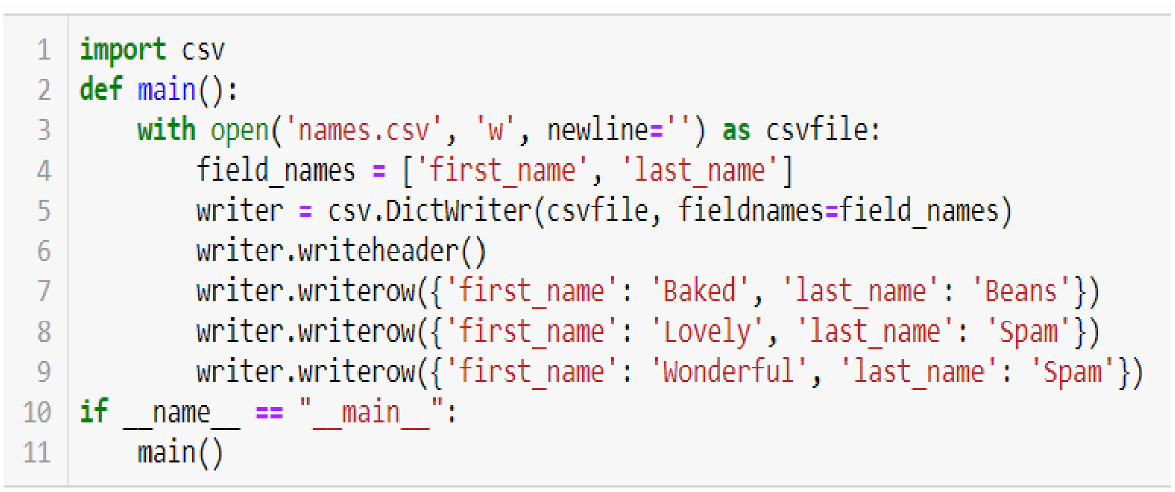
- Write a program to get a following result

- Expected result

```
Bulbasaur, Grass
Charizard, Fire
Squirtle, Water
Pikachu, Electric
Rapidash, Fire
```

Program:

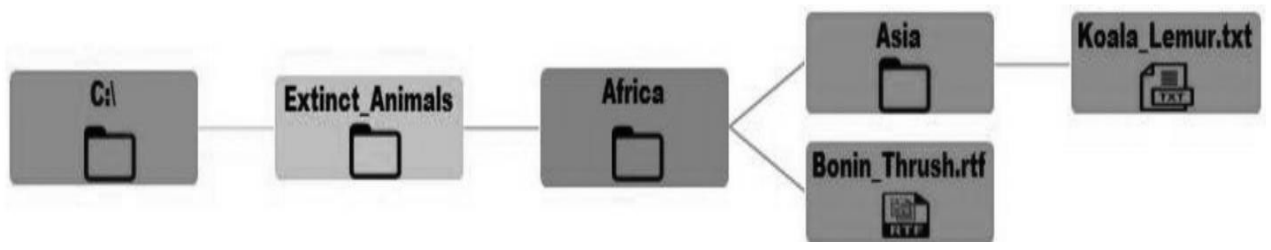
- Write Python program to demonstrate the writing of data to a CSV file using DictWriter class



```python
import csv
def main():
    with open('names.csv', 'w', newline='') as csvfile:
        field_names = ['first_name', 'last_name']
        writer = csv.DictWriter(csvfile, fieldnames=field_names)
        writer.writeheader()
        writer.writerow({'first_name': 'Baked', 'last_name': 'Beans'})
        writer.writerow({'first_name': 'Lovely', 'last_name': 'Spam'})
        writer.writerow({'first_name': 'Wonderful', 'last_name': 'Spam'})
if __name__ == "__main__":
    main()
```

Program:

- Consider the File Structure Given Below. Write Python Program to Delete All the Files and Subdirectories from the Extinct_Animals Directory

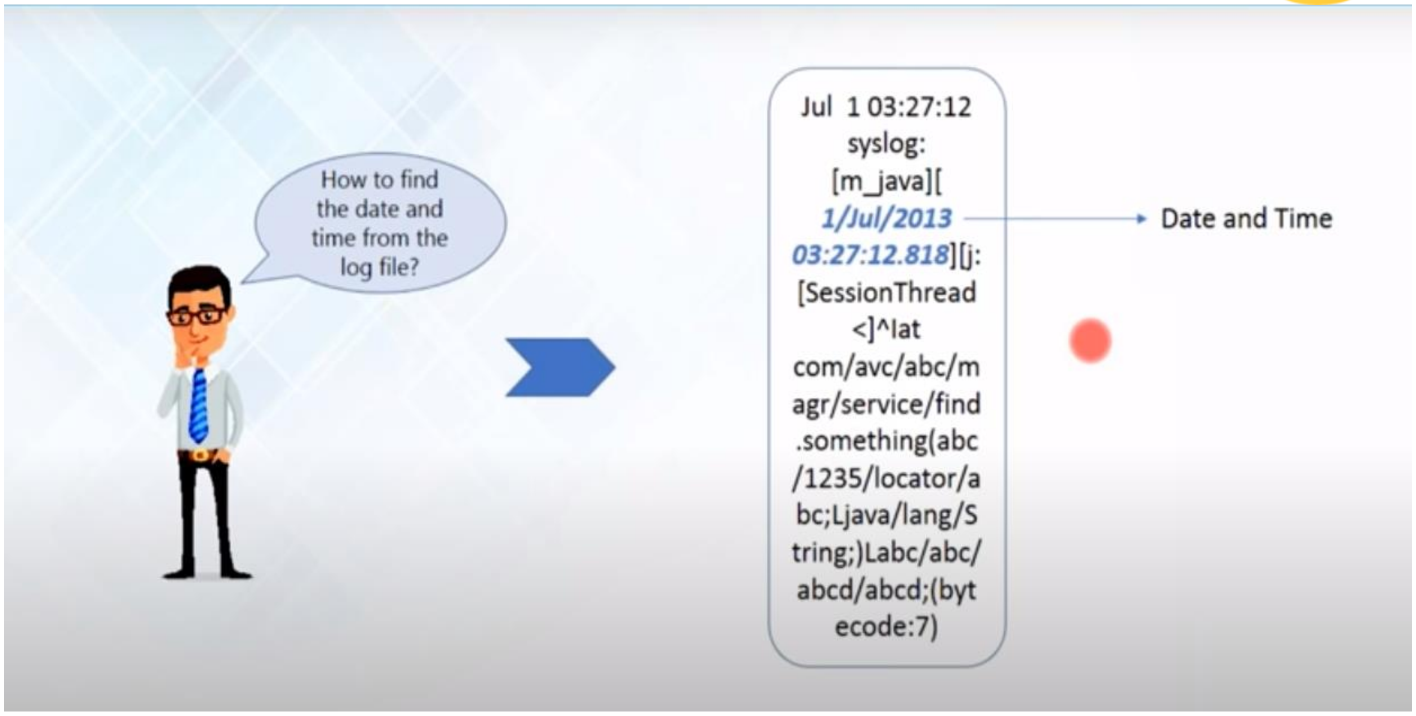


- Write a program to delete all the files

Online Materials:

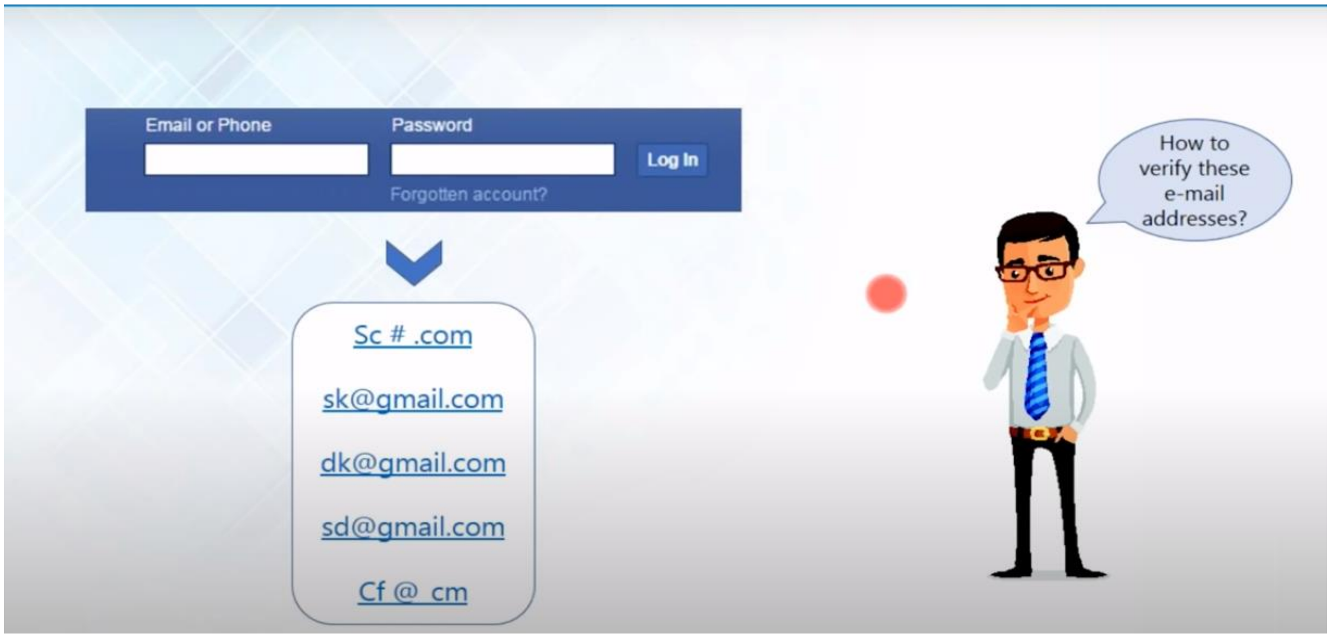
- https://www.w3schools.com/python/python_file_handling.asp
- https://www.programiz.com/python-programming/file-operation
- https://www.geeksforgeeks.org/file-handling-python/
- https://www.tutorialspoint.com/python/python_files_io.htm
- https://www.guru99.com/reading-and-writing-files-in-python.html

Regular Expressions:

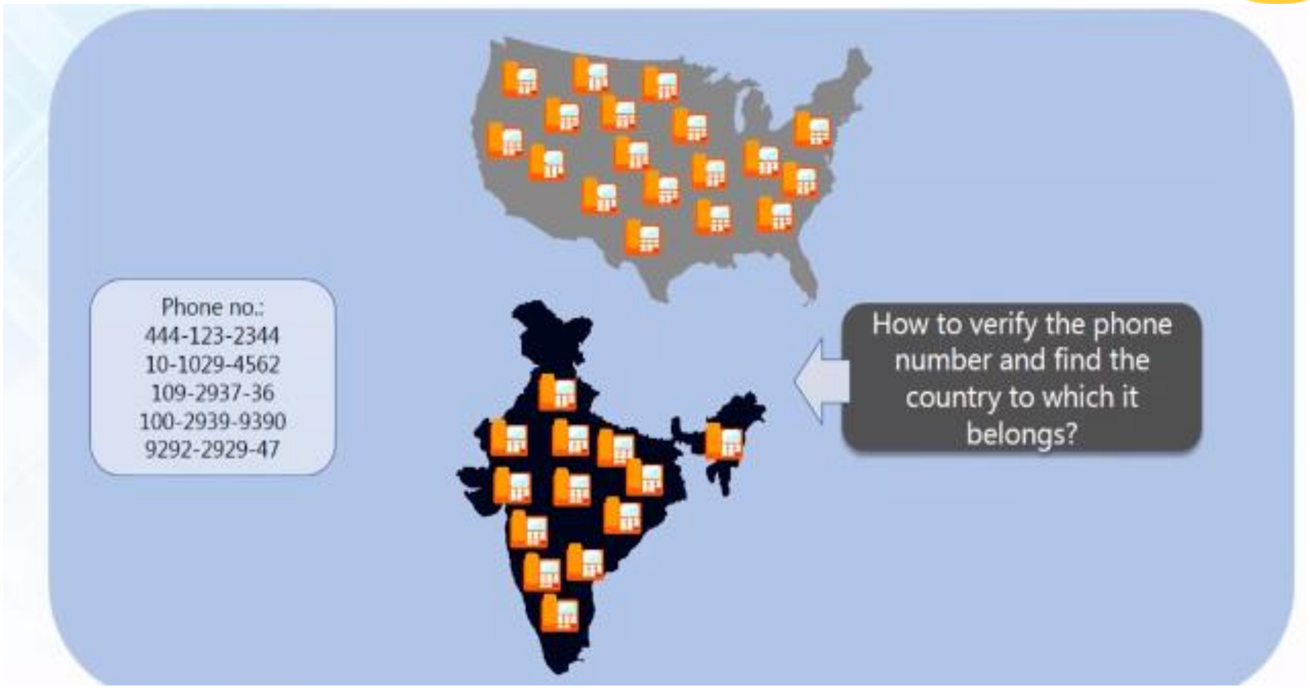


```
Jul  1 03:27:12
syslog:
[m_java][
1/Jul/2013
03:27:12.818][j:
[SessionThread
<]^lat
com/avc/abc/m
agr/service/find
.something(abc
/1235/locator/a
bc;Ljava/lang/S
tring;)Labc/abc/
abcd/abcd;(byt
ecode:7)
```

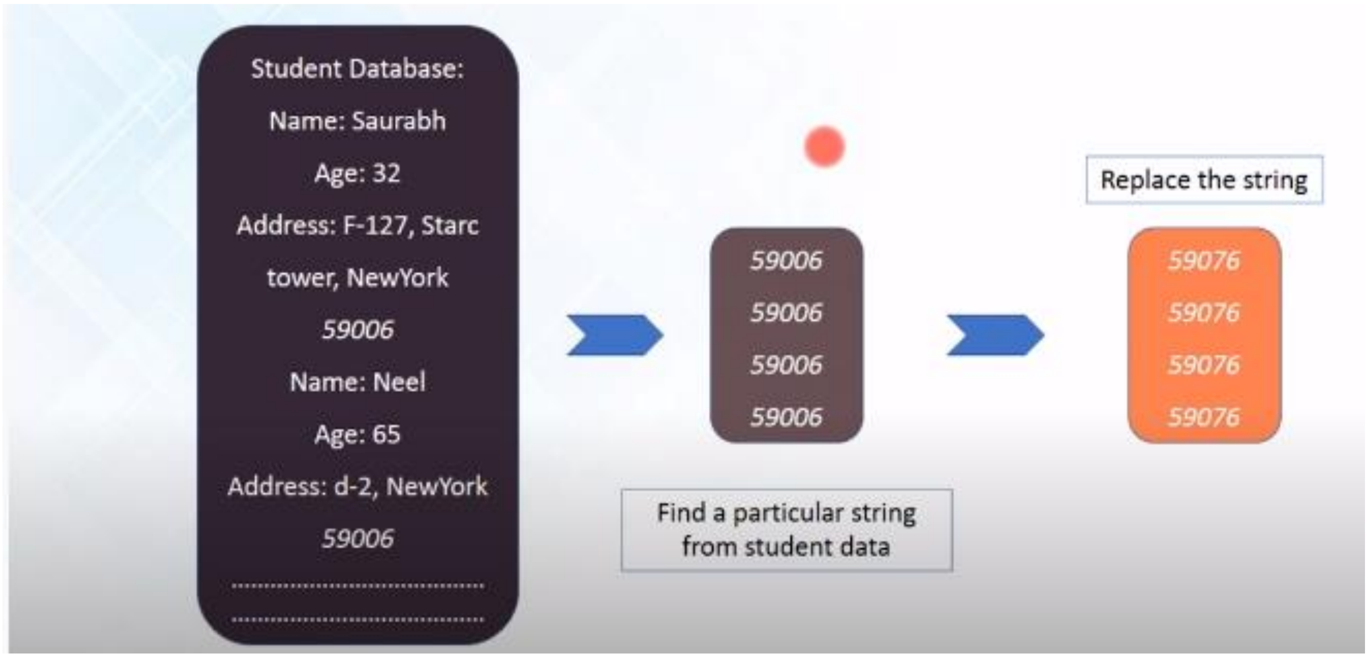
Regular Expressions:



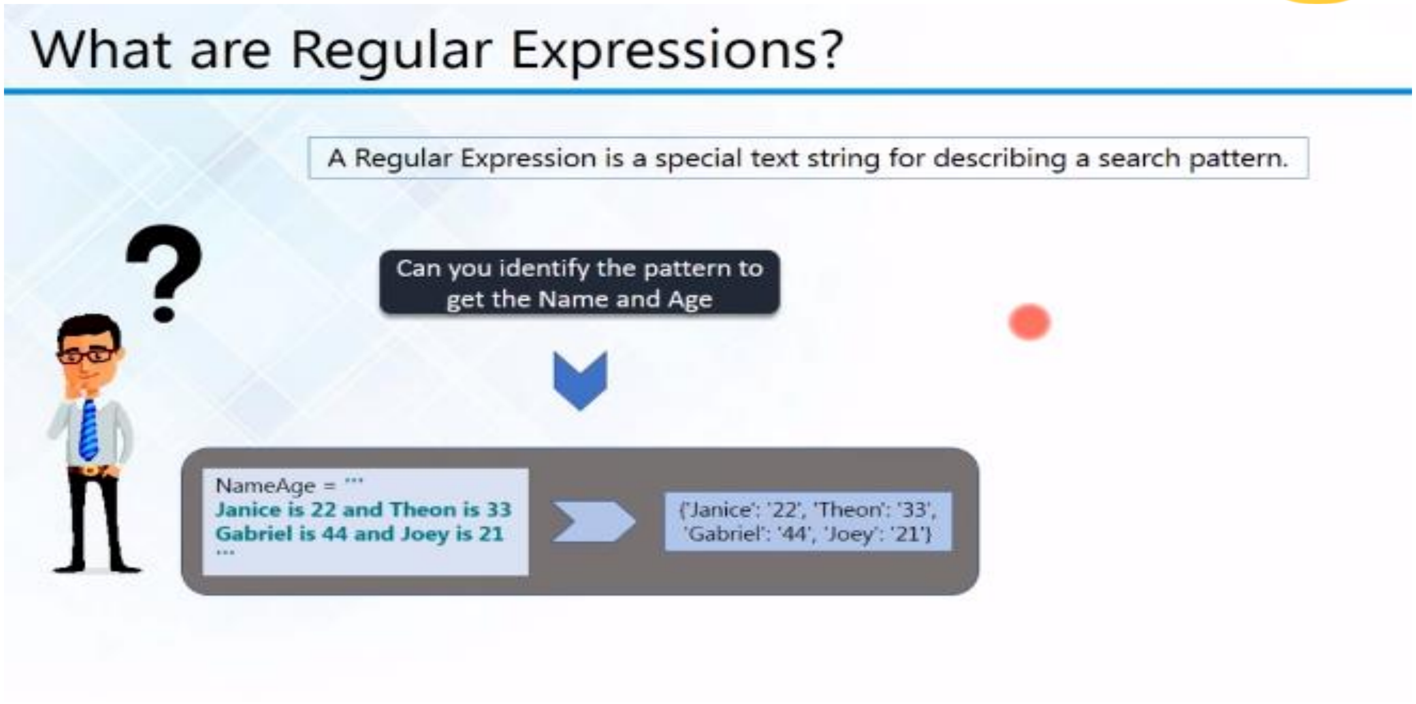
Regular Expressions:


Phone no.:
444-123-2344
10-1029-4562
109-2937-36
100-2939-9390
9292-2929-47
How to verify the phone number and find the country to which it belongs?

Regular Expressions:



Regular Expressions:



What are Regular Expressions?

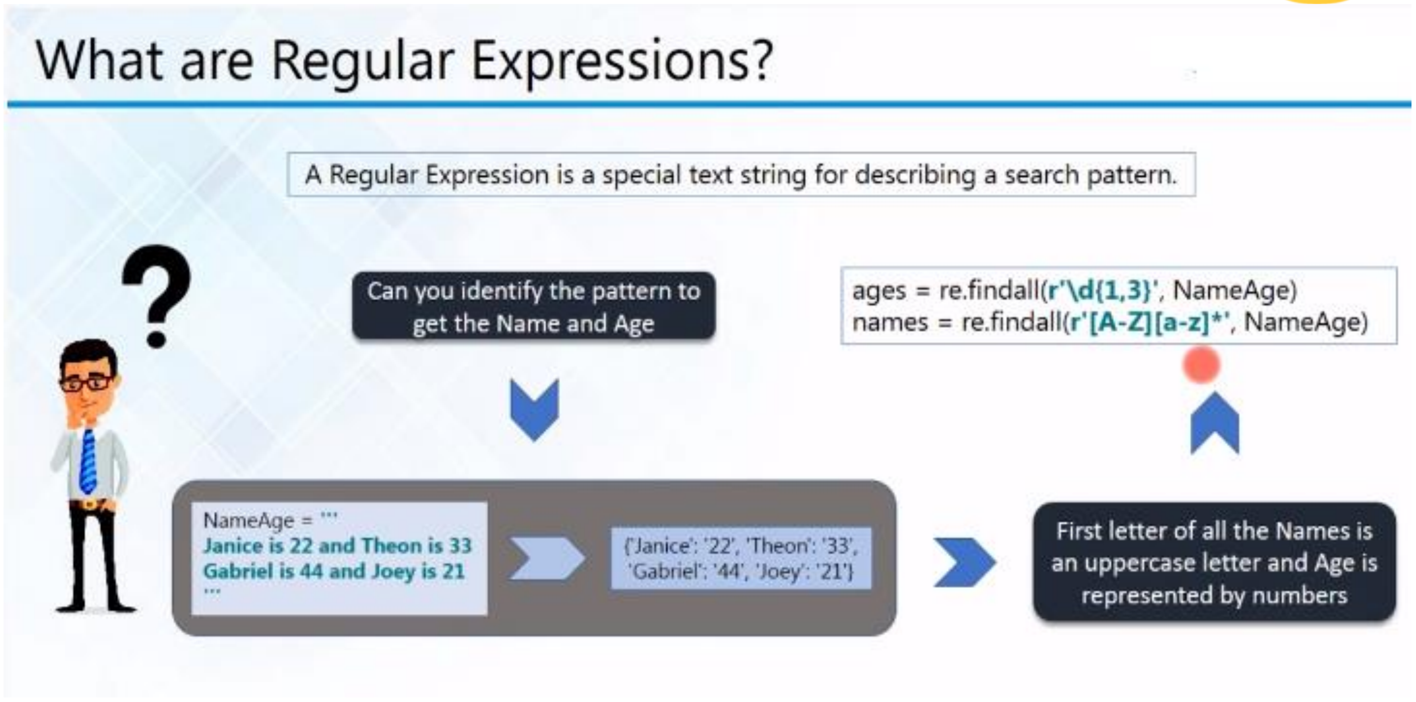
A Regular Expression is a special text string for describing a search pattern.

Can you identify the pattern to get the Name and Age

```
NameAge = '''
Janice is 22 and Theon is 33
Gabriel is 44 and Joey is 21
'''
```

```
{'Janice': '22', 'Theon': '33', 'Gabriel': '44', 'Joey': '21'}
```

Regular Expressions:



What are Regular Expressions?

A Regular Expression is a special text string for describing a search pattern.

Can you identify the pattern to get the Name and Age

```
ages = re.findall(r'\d{1,3}', NameAge)
names = re.findall(r'[A-Z][a-z]*', NameAge)
```

```
NameAge = '''
Janice is 22 and Theon is 33
Gabriel is 44 and Joey is 21
'''
```

```
{'Janice': '22', 'Theon': '33', 'Gabriel': '44', 'Joey': '21'}
```

First letter of all the Names is an uppercase letter and Age is represented by numbers

Regular Expressions:

- A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.
- RegEx can be used to check if a string contains the specified search pattern.
- Python has a built-in package called re, which can be used to work with Regular Expressions.
- Import the re module:

```python
import re
```

- When you have imported the re module, you can start using regular expressions:

Regular Expressions:

RegEx Functions

The `re` module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

Regular Expressions:

The findall() Function

The `findall()` function returns a list containing all matches.

Example

Print a list of all matches:

```
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

The list contains the matches in the order they are found.

If no matches are found, an empty list is returned:

Example

Return an empty list if no match was found:

```python
import re

txt = "The rain in Spain"
x = re.findall("Portugal", txt)
print(x)
```

Regular Expressions:

The search() Function

The `search()` function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned:

Example

Search for the first white-space character in the string:

```
import re

txt = "The rain in Spain"
x = re.search("\s", txt)

print("The first white-space character is located in position:", x.start())
```

The split() Function

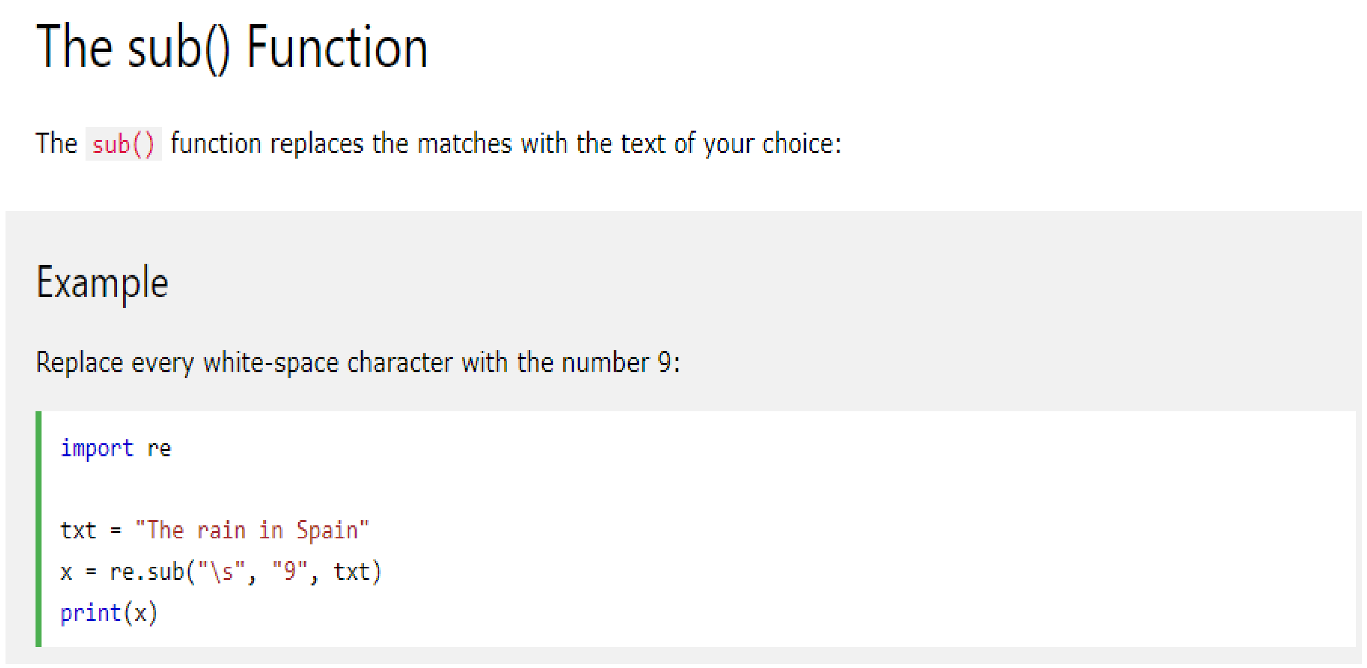
The `split()` function returns a list where the string has been split at each match:

Example

Split at each white-space character:

```python
import re

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

Regular Expressions:



The sub() Function

The `sub()` function replaces the matches with the text of your choice:

Example

Replace every white-space character with the number 9:

```
import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

Regular Expressions:

Compile() function

re module contains compile() function to compile a pattern into RegexObject.
Example: pattern= re.compile ("python")

finditer()
Returns an Iterator object which yields Match object for every Match
matcher=pattern.finditer(" programmin in python")

On Match object we can call the following methods:

start()-start index of the match
end()- end+1 indes of match
group()-: returns matched stringis

pattern=re.compile("ab")
matcher=pattern.finditer("abcababa")

Special Characters	Meaning
^	Matches the **start** of a string
$	Matches the **end** of the string
.	Matches **any single** character
\s	Matches a single **whitespace** character including space
\S	Matches any **single non-whitespace** character
*****	**Repeats or Matches** a character zero or more times of **preceding expression**
***?**	**Repeats or Matches** a character zero or more times (non-greedy) of **preceding expression**
+	**Repeats or Matches** a character one or more times of **preceding expression**
+?	**Repeats or Matches** a character one or more times (non-greedy) of **preceding expression**
[aeiou]	Matches any single character in the listed **bracket**
[^XYZ]	Matches any single character **not in** the listed **set or bracket**
[a-z0-9]	The set of characters can include a **range denoted by hyphen**
(	Indicates where string **extraction is to start**
)	Indicates where string **extraction is to end**
r	Use "r" at the start of the pattern string, it designates a **python raw string**
\w	The characters [a-zA-Z0-9_] are **word characters**. These are also matched by the **short-hand character class \w. Note that although "word" is the mnemonic for this, it only matches a single word character, not a whole word**
\W	Matches any **non-word character**
\d	Matches any **decimal digit** [0-9]
\D	Matches any **non-digit character**. Equivalent to [^0-9]
\b	Matches a **word boundary**
\B	Matches a **non-word boundary**
{m, n}	Where m and n are positive integers and m <= n. Matches at least **m and at most n occurrences** of the preceding expression
{m}	Matches exactly **m occurrences** of the preceding expression
|	A | B **Matches 'A', or 'B'** (if there is no match for 'A'), where A and B are regular expressions