Reading config files in python:

The configparser module from Python's standard library defines functionality for reading and writing configuration files as used by Microsoft Windows OS. Such files usually have .INI extension.

The INI file consists of sections, each led by a [section] header. Between square brackets, we can put the section's name. Section is followed by key/value entries separated by = or : character. It may include comments, prefixed by # or ; symbol.

```
[Settings]
# Set detailed log for additional debugging info
DetailedLog=1
RunStatus=1
StatusPort=6090
StatusRefresh=10
Archive=1
# Sets the location of the MV_FTP log file
LogFile=/opt/ecs/mvuser/MV_IPTel/log/MV_IPTel.log
Version=0.9 Build 4
ServerName=Unknown

[FTP]
# set the FTP server active
RunFTP=1
# defines the FTP control port
FTPPort=21
# Sets the location of the FTP data directory
FTPDir=/opt/ecs/mvuser/MV_IPTel/data/FTPdata
# set the admin Name
UserName=admin
# set the Password
Password=admin
```

The configparser module has ConfigParser class. It is responsible for parsing a list of configuration files, and managing the parsed database.

Object of ConfigParser is created by following statement –

```
parser = configparser.ConfigParser()
```

script to reads and parses the 'sampleconfig.ini' file:

```
import configparser
parser = configparser.ConfigParser()
parser.read('sampleconfig.ini')
for sect in parser.sections():
   print('Section:', sect)
   for k,v in parser.items(sect):
      print(' {} = {}'.format(k,v))
   print()
```

The write() method is used to create a configuration file. Following script configures the parser object and writes it to a file object representing 'test.ini'

```
import configparser
parser = configparser.ConfigParser()
parser.add_section('Manager')
parser.set('Manager', 'Name', 'Ashok Kulkarni')
parser.set('Manager', 'email', 'ashok@gmail.com')
parser.set('Manager', 'password', 'secret')
fp=open('test.ini','w')
parser.write(fp)
fp.close()
```

Writing log files in python

1. Create and configure the logger. It can have several parameters. But importantly, pass the name of the file in which you want to record the events.
2. Here the format of the logger can also be set. By default, the file works in **append** mode but we can change that to write mode if required.

3. Also, the level of the logger can be set which acts as the threshold for tracking based on the numeric values assigned to each level. There are several attributes that can be passed as parameters.
4. The list of all those parameters is given in Python Library. The user can choose the required attribute according to the requirement. After that, create an object and use the various methods like:

```
# importing module
import logging

# Create and configure logger
logging.basicConfig(filename="newfile.log",
                                    format='%(asctime)s %(message)s',
                                    filemode='w')

# Creating an object
logger = logging.getLogger()

# Setting the threshold of logger to DEBUG
logger.setLevel(logging.DEBUG)

# Test messages
logger.debug("Harmless debug Message")
logger.info("Just an information")
logger.warning("Its a Warning")
logger.error("Did you try to divide by zero")
logger.critical("Internet is down")
```

Understanding read functions:

Python provides inbuilt functions for creating, writing and reading files. There are two types of files that can be handled in python, normal text files and binary files.

- **Text files:** In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default.
- **Binary files:** In this type of file, there is no terminator for a line and the data is stored after converting it into machine-understandable binary language.

Access mode

Access modes govern the type of operations possible in the opened file. It refers to how the file will be used once it’s opened. These modes also define the location of the File Handle in the file. File handle is like a cursor, which

defines from where the data has to be read or written in the file. Different access modes for reading a file are –

1. **Read Only ('r') :** Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises I/O error. This is also the default mode in which file is opened.
2. **Read and Write ('r+') :** Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exists.
3. **Append and Read ('a+') :** Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

Opening a File

It is done using the open() function. No module is required to be imported for this function.

```
file1 = open("MyFile.txt", "r")
or
file2 = open(r"D:\Text\MyFile2.txt", "r+")
```

Reading from a file

- **read() :** Returns the read bytes in form of a string. Reads n bytes, if no n specified, reads the entire file.

```
File_object.read([n])
```

- **readline() :** Reads a line of the file and returns in form of a string.For specified n, reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.

```
File_object.readline([n])
```

- **readlines() :** Reads all the lines and return them as each line a string element in a list.

```
File_object.readlines()
```

```
Example :

# Program to show various ways to
```

```
# read data from a file.

# Creating a file
file1 = open("myfile.txt", "w")
L = ["This is Delhi \n", "This is Paris \n", "This is London \n"]

# Writing data to a file
file1.write("Hello \n")
file1.writelines(L)
file1.close() # to change file access modes

file1 = open("myfile.txt", "r+")

print("Output of Read function is ")
print(file1.read())
print()

# seek(n) takes the file handle to the nth
# byte from the beginning.
file1.seek(0)

print("Output of Readline function is ")
print(file1.readline())
print()

file1.seek(0)

# To show difference between read and readline
print("Output of Read(9) function is ")
print(file1.read(9))
```

```
print()


file1.seek(0)


print("Output of Readline(9) function is ")
print(file1.readline(9))
print()


file1.seek(0)


# readlines function
print("Output of Readlines function is ")
print(file1.readlines())
print()
file1.close()
```

Understanding write functions

`write()` function

The write() function will write the content in the file without adding any extra characters.

```
file = open("Employees.txt", "w")

for i in range(3):
name = input("Enter the name of the employee: ")
file.write(name)
file.write("\n")

file.close()

print("Data is written into the file.")
```

writelines() function

This function writes the content of a list to a file.

```
file1 = open("Employees.txt", "w")
lst = []
for i in range(3):
        name = input("Enter the name of the employee: ")
        lst.append(name + '\n')

file1.writelines(lst)
file1.close()
print("Data is written into the file.")
```

Manipulating file pointer using seek

seek() Method

Python seek() method is used for changing the current location of the file handle. The file handle is like a cursor, which is used for defining the location of the data which has to be read or written in the file.

- **0:** The 0 value is used for setting the whence argument at the beginning of the file.
- **1:** The 1 value is used for setting the whence argument at the current position of the file.
- **2:** The 2 value is used for setting the whence argument at the end of the file.

The from_where argument is set to 0 by default. Therefore, the reference point cannot be set to the current position or end position, in the text mode, except when the offset is equal to 0.

Example:

```
f = open("GfG.txt", "r")

f.seek(20)

# prints current position
```

```
print(f.tell())

print(f.readline())

f.close()
```