SUM_Automation.py Code Breakdown

Necessary modules are imported. `Python files` is the folder that contains certain support files required for the script. hence the path is added to the current script path. `uninstall_ofed` is the python script that contains function to uninstall the ofed from the servers.

```
import os
import sys
import time
import sys
sys.path.append("Python_files")
from uninstall_ofed imort *
```

Main function

Parser would be the object of option parser class (Used for handling parameters and conditions). `add_options` is the method used to add the parameter conditions. `add_help_option=False` is used to deprecate the default help message and use the new help message created.

```
if __name__=='__main__':
    parser = OptionParser(add_help_option=False)
    add_options(parser)
```

Options are added to the parser object. help defines the help message that has to be displayed. We have overwritten the help message with the new help message stored in a string format in help_message variable. Hence the string mentioned in help section below will be deprecated.

```
def add_options (parser):
    parser.add_option("-h","--help",  help=help_message ,
action="store_true",default = False)
    parser.add_option("-o","--prerequisite_only", help = "Executes only the
prerequisites", action="store_true", default = False)
    parser.add_option("-p","--prerequisite", help = "Executes the prerequisites
along with the main automation process", action="store_true", default = False)
    parser.add_option("-i","--input", help = "Executes only automation process
(no prerequisites)", action="store", type="string")
```

The below line retrieves the properties of the parser object. `options` will be a dictionary containing key as option name and value as its action performed. (action would be true if user uses that option as parameter while executing the script). `args` wil contain the arguments that are passed with the options.

```
(options, args) = parser.parse_args()
```

If user choses -h/--help as the option, then help message will be printed.

```
if options.help:
        print(help_message)
        sys.exit()
```

```
help_message = "syntax for parameters:\n\
                -o,--prerequisite_only <path to input file>     :     For running
only prerequisites \n\
                -p,--prerequisite <path to input file>          :     For running
the script with prerequisites\n\
                -i,--input <path to input file>                 :     For running
the script without prerequisites\n"
```

Refer the initialize method for details.

```
initialize(args)
```

If user choses to run only prerequisites, then the robot file is executed with parameters `input:no` and `input_file:<path_to_input_file>`.

> 1. `input:no` implies that there is no need of input required from the user further. The script will execute on its own.
> 2. `-l repo_clear_log.html` will name the log file as repo_clear_log (by default it will be log.html)
> 3. `-r repo_clear_report.html` will name the report file as repo_clear_report.html (by default it will be report.html)
>
>   The above naming is used to prevent the log and report overlapping with the log and report generated by the `SUM_Automation.robot` script.
>
>   Output generated will be collected in `repo_clear_debug_details` file.

```
if options.prerequisite_only:
    print("Clearing iLO repository...")
    os.system('cmd /c "robot -v input:no -v input_file:"'+args[0]+'" -l
repo_clear_log.html -r repo_clear_report.html iLO_repo.robot >
repo_clear_debug_details"')
    time.sleep(300)
    print("Uninstalling OFED...")
    print(uninstall(args[0]))
```

uninstall(input_file_path) will uninstall the ofed from the servers mentioned in the input file. This method is in `uninstall_ofed.py` file within `Python_files` folder. Refer the uninstall method for more details.

```
uninstall(args[0])
```

If user choses to run the script along with the prerequisites, then the following code will be executed. First the robot file which clears the repository. Second the python script to uninstall the ofed from server. Thirdly the main `SUM_Automation.robot` script.

time delay is required after ofed uninstallation because the system will be rebooted after that process.

```
elif options.prerequisite:
        print("Clearing iLO repository...")
        os.system('cmd /c "robot -v input:no -v input_file:"'+args[0]+'" -l
repo_clear_log.html -r repo_clear_report.html iLO_repo.robot >
repo_clear_debug_details"')
        time.sleep(300)
        print("Uninstalling OFED...")
        print(uninstall(args[0]))
        os.system('cmd /c "py Python_files/uninstall_ofed.py"')
        time.sleep(300)
        print("SUM Automation in progres...")
        os.system('cmd /c "robot -v input:no -v input_file:"'+args[0]+'" -l
sum_automation_log.html -r sum_automation_report.html SUM_Automation.robot >
sum_automation_debug_details"')
```

If user choses to run the automation script alone, then directly the robot script SUM_Automation.robot will be executed.

```
elif options.input:
    print("SUM Automation in progres...")
    os.system('cmd /c "robot -v input:no -v input_file:"'+args[0]+'" -l
sum_automation_log.html -r sum_automation_report.html SUM_Automation.robot >
sum_automation_debug_details"')
```

If user does not select any option then, UI will be popped up asking for the user inputs.

> input : yes is made as the parameter for the robot file since UI is required in this scenario.

```
else:
    os.system('cmd /c "robot -v input:yes -l sum_automation_log.html -r
sum_automation_report.html SUM_Automation.robot >
sum_automation_debug_details"')
```

The Initialize Method

Initialize method is used to do to things:

1. Extract the data from the input file.
2. Mounting the SPP to the required drive.

```
initialize(args)
```

As discussed `args` contains the arguments that are passed. User should provide the path of the input file as the argument. Upon identifying the input file location read every line (`each`) and every value (`word`) and assign it to the two dimensional matrix `data`.

> **data** : 2D matrix : contains details of the input file

```
def initialize(args):
    file = open(args[0],"r")
    for each in file:
        word = each.split()
        data.append([])
        for every in word:
            data[-1].append(every)
```

`data\[1]\[1]` contains the path to the folder of spp. `data\[1]\[1]\[0]` represents the drive in which spp is stored.

In order to mount the spp, the script must go to drive in which the spp is present. Hence the drive name is extracted from the directory path. Since the first letter of the string will always be drive name, `data\[1]\[1]\[0]` is chosen.

```
try:
    val=str(data[1][1][0])+':'
    spp_name=os.path.basename(data[2][1])
    os.system('cmd /c "c: & '+str(val)+' & cd "'+str(data[1][1])+'" &
"'+spp_name)
except:
    print('Failed to mount the spp')
    sys.exit()
```

uninstall_ofed.py Script

The uninstall Method

Reading the input file and storing the information in the list `data`.

```
def uninstall(input_file):
    file = open(input_file,"r")
    data=[]
    for each in file:
        word = each.split()
        data.append([])
        for every in word:
            data[-1].append(every)
    file.close()
```

`entry` is a Boolean variable used for the conditional flow of the for loop. `string` is the variable used to store the statements that needs to be printed after the execution. string will be returned to calling function. Below if statements takes care of the blank spaces that are inserted in the input file.

```
entry = False
    for elem in data:
        string = ''
        if len(elem)==0:
            continue
        if elem[0]=='':
            continue
```

`entry` will be true only if script encounters the word `#SERVER`. This is to make sure that only server details are used for the further processing leaving behind the other details such as SPP path, drive name etc. If you generalize the for loop for every element in the list then you will get index out of bound exception as previous details will not be of same length of the server details.

`flag` is a Boolean variable used to check the condition that whether the server is Linux based and is not excluded or not. If the server is Linux and is included for automation then the flag is made True.

Refer the ssh_command method for more details.

Execution of command `ofed_uninstall.sh --force` is wrapped under try catch block because if the ofed has already been uninstalled then this would throw an error. So in order to handle that error try catch is used. If error is thrown, then the statements ofed already been uninstalled will be added to the string. If there is no error, then this means two possibilities. One being successful uninstallation of ofed and other being unsuccessful uninstallation. These two conditions are handled by the if else statements.

`if '#SERVER' in elem[0]:` condition takes care that entry is made true only after the encounter of SERVER DETAILS.

```
if entry == True:
    flag=False
        try:
            if elem[3]=='Linux' and elem[0][0]!='#':
                flag=True
                value = sshCommand(elem[0],22,elem[1],elem[2],'ofed_uninstall.sh
--force')
                compare = value.rstrip("\n")
                if compare=='Uninstall finished successfully':
                    string += elem[0]+' : ofed uninstallation was successful'
                else:
                    string +=  elem[0]+' : ofed uninstallation was unsuccessful'
        except:
            if flag==True and elem[0][0]!='#':
                string += elem[0]+' : ofed already been uninstalled'
        if elem[0][0]!='#' and elem[3]=='Linux' or elem[3]=='Windows' or
elem[3]=='iLO-ESXi':
            sshCommand(elem[0],22,elem[1],elem[2],'reboot')

    if '#SERVER' in elem[0]:
            entry = True
return string
```

The ssh_command Method

`Paramiko` is the library that carry outs the ssh command from the python script. First 3 lines is the initialization of the sshclient.

`stdout.readlines()` will return the list with each line of the output of executed command as its element.

`''.join(stdout.readlines())` returns the exact output of the command in the string format.

`result[-1]` contains the last line of the output which says whether the ofed installation was successful or nor.

```
def sshCommand(hostname, port, username, password, command):
    sshClient = paramiko.SSHClient()
    sshClient.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    host = sshClient.get_host_keys()
    host.clear()
    sshClient.connect(hostname, port, username, password)
    stdin, stdout, stderr = sshClient.exec_command(command)
    result = stdout.readlines()
    #result = ''.join(stdout.readlines())
    return  result[-1].strip()
```