CANDIDATE'S FULL NAME: Daniel Mula Tarancón

'''You work at a company that receives daily data files from external partners. These files need to be processed and analyzed, but first, they need to be transferred to the company's internal network.

The goal of this project is to automate the process of transferring the files from an external FTP server to the company's internal network.

Here are the steps you can take to automate this process:

Use the ftplib library to connect to the external FTP server and list the files in the directory.

Use the os library to check for the existence of a local directory where the files will be stored.

Use a for loop to iterate through the files on the FTP server and download them to the local directory using the ftplib.retrbinary() method.

Use the shutil library to move the files from the local directory to the internal network.

Use the schedule library to schedule the script to run daily at a specific time.

You can also set up a log file to keep track of the files that have been transferred and any errors that may have occurred during the transfer process.'''

[illegible]

INDEX

1. Python libraries used

A) The **os** library

The ‘os’ library is a built-in Python module that provides a portable way of using operating system dependent functionality. It allows you to interact with the file system and perform various operations on files and directories, such as creating, deleting, renaming, moving, and copying files, and creating, listing, and deleting directories.

Some of the commonly used functions in the ‘os’ library include:

- os.path.join(): Joins one or more path components together and returns a new path. This is useful for creating file paths that are compatible with the current operating system.

- os.listdir(): Returns a list of all files and directories in the specified directory.

- os.mkdir(): Creates a new directory with the specified name.

- os.rmdir(): Deletes the specific directory.

- os.rename(): Renames a file or directory.

- os.remove(): Deletes a file.

- os.path.exists(): Checks if a file or directory exists.

These functions make it easy to work with the file system in Python and provide a convenient way to automate file and directory operations.

B) The ‘ftplib’ library is a built-in Python module that provides a way to interact with FTP (File Transfer Protocol) serves. It allows you to perform various operations on FTP servers, such as downloading and uploading files, creating and deleting directories, and navigating the directory structure of the FTP server.

Some of the commonly used functions in the ‘ftplib’ library include:

ftplib.FTP(): Creates an FTP instance that can be used to connect to an FTP server.

FTP.login(): Logs in to the FTP server with the specified username and password.

FTP.cwd(): Changes the current working directory on the FTP server.

FTP.nlst(): Returns a list of filenames in the current working directory on the FTP server.

FTP.retrbinary(): Downloads a file from the FTP server and writes it to a local file.

FTP.storbinary(): Uploads a file to the FTP server.

These functions make it easy to work with FTP servers in Python and provide a convenient way to automate FTP operations.

C) The shutil library is a built-in Python module that provides a set of high-level operations on files and collections of files. It allows you to perform various file and directory operations such as copying, moving, renaming, and deleting files and directories.

Some of the commonly used functions in the ‘shutil’ library include:

shutil.copy(): Copies a file from one location to another.

shutil.copytree(): Copies a directory tree (including all files and subdirectories) from one location to another.

shutil.move(): Moves a file or directory from one location to another.

shutil.rmtree(): Deletes a directory tree (including all files and subdirectories).

These functions make it easy to work with files and directories in Python and provide a convenient way to automate file and directory operations. Additionally, the ‘shutil’ library provides cross-platform compatibility, so you can use the same code to perform these operations on Windows, macOS, and Linux.

D) The schedule library

The schedule library is a Python library that provides a simple and convenient way to schedule recurring tasks at specific times or intervals. It allows you to automate tasks that need to be performed regularly, such as checking for updates, running backups, or sending reports.

Some of the commonly used functions in the ‘schedule’ library include:

schedule.every(): Returns a ‘Job’ instance that can be used to schedule a task.
Job.run(): Runs the schedule task.

Job.do(): Specifies the function that will be called when the task is run.

schedule.every().day.at(): Schedules a task to run at a specific time each day.

schedule.every().hour: Schedules a task to run every hour.

schedule.every().minute: Schedules a task to run every minute.

These functions make it easy to schedule tasks in Python and provide a convenient way to automate repetitive tasks. Additionally, the ‘schedule’ library is lightweight and easy to use, making it a popular choice for scheduling tasks in Python.

E) The logging library

The logging library is a built-in Python module that provides a way to record and output log messages from your Python code. It allows you to capture and store information about what your code is doing, making it easier to debug problems and understand how your code is working.

Some of the commonly used functions in the ‘logging’ library include:

logging.basicConfig(): Configures the logging system with basic configuration settings.

logging.getLogger(): Gets a logger instance that can be used to emit log messages.

Logger.setLevel(): Sets the logging level for a longer.

Logger.debug(), logger.info(), ‘Logger.warning(), Logger.error()’, ‘Logger.critical()’: Methods for emitting log messages at different severity levels.

logging.handlers: A module that provides various handlers for emitting log messages to different destinations, such as the console, a file, or a network socket.

These function make it easy to add logging to your Python code and provide a convenient way to capture and store information about what your code is doing. Additionally, the ‘logging’ library provides a flexible and configurable logging system, making it easy to customize the logging behavior to suit your needs.

2. Script Structure

The script is basically composed by one big function that make all the process.

Outside that function some necessary variables are defined, the loggin.basicConfig and the schedule library so the scripts runs periodically.

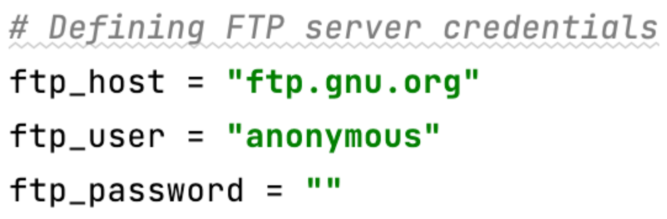
The scripts is controlled using a ‘dunder main’ block’.

So when we execute the script that ‘dunder main’ block is triggered.

3. Getting into details

In this section, I'll explain each part of the code. Why it is necessary and what it does.

A) Defining the server credentials

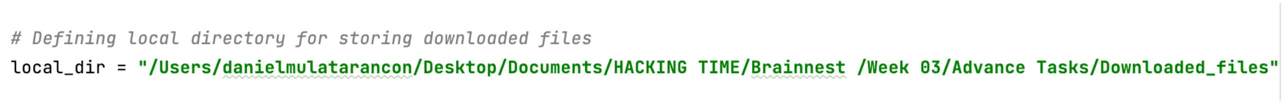


```
# Defining FTP server credentials
ftp_host = "ftp.gnu.org"
ftp_user = "anonymous"
ftp_password = ""
```

So I decided to use a free open source FTP server, ftp.gnu.org in this case.

And I defined the credentials as you can see in the image.

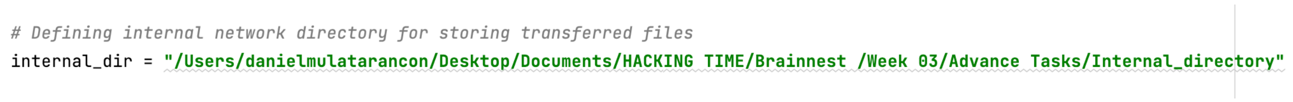
B) Defining the local directory for storing downloaded files



```
# Defining local directory for storing downloaded files
local_dir = "/Users/danielmulatarancon/Desktop/Documents/HACKING TIME/Brainnest /Week 03/Advance Tasks/Downloaded_files"
```

This is the local directory I use in my case. Of course, you'll have to use your own. Just putting the global path into a variable.

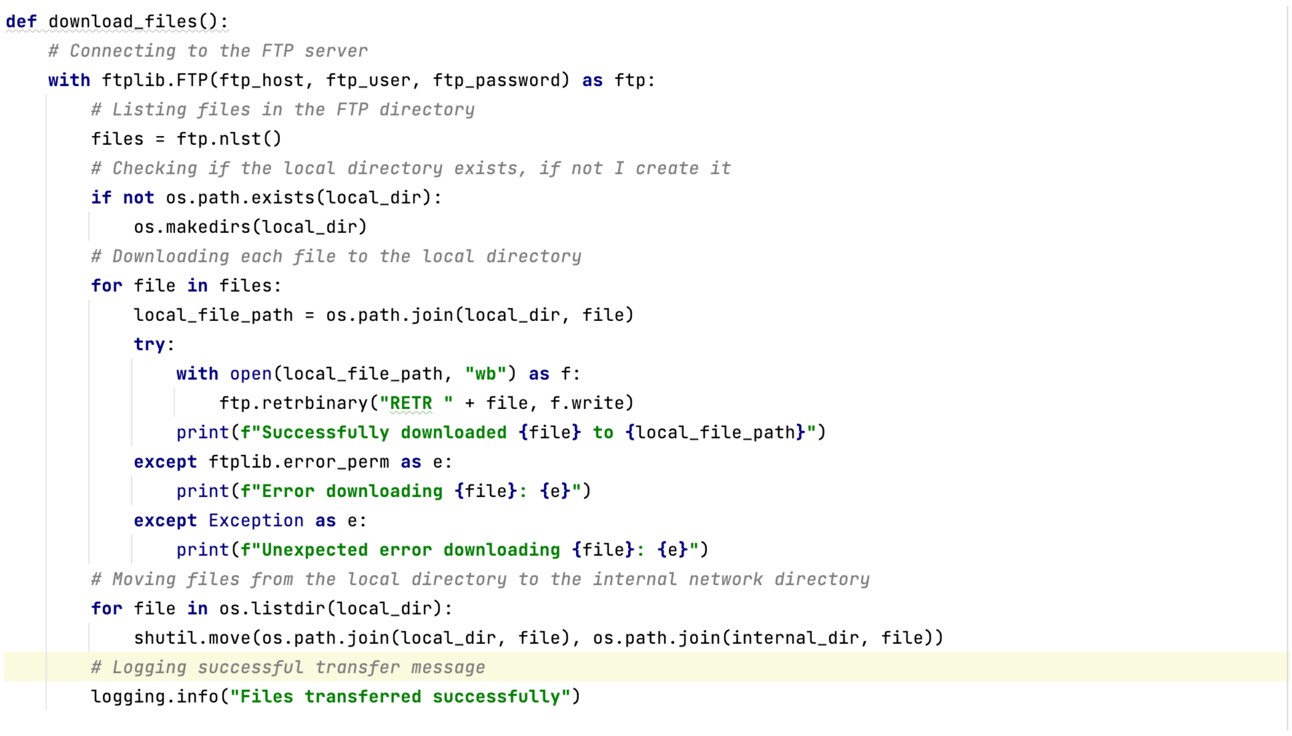
C) Defining internal network directory for storing transferred files



```
# Defining internal network directory for storing transferred files
internal_dir = "/Users/danielmulatarancon/Desktop/Documents/HACKING TIME/Brainnest /Week 03/Advance Tasks/Internal_directory"
```

Similar operation here when defining the internal directory.

D) Defining a function to downloading files from the FTP server



```python
def download_files():
    # Connecting to the FTP server
    with ftplib.FTP(ftp_host, ftp_user, ftp_password) as ftp:
        # Listing files in the FTP directory
        files = ftp.nlst()
        # Checking if the local directory exists, if not I create it
        if not os.path.exists(local_dir):
            os.makedirs(local_dir)
        # Downloading each file to the local directory
        for file in files:
            local_file_path = os.path.join(local_dir, file)
            try:
                with open(local_file_path, "wb") as f:
                    ftp.retrbinary("RETR " + file, f.write)
                print(f"Successfully downloaded {file} to {local_file_path}")
            except ftplib.error_perm as e:
                print(f"Error downloading {file}: {e}")
            except Exception as e:
                print(f"Unexpected error downloading {file}: {e}")
        # Moving files from the local directory to the internal network directory
        for file in os.listdir(local_dir):
            shutil.move(os.path.join(local_dir, file), os.path.join(internal_dir, file))
        # Logging successful transfer message
        logging.info("Files transferred successfully")
```

d1) Firstly we establish connection to the server.

The code 'with ftplib.FTP(ftp_host, ftp_user, ftp_password) as ftp: 'creates a new 'FTP' object and establishes a connection to an FTP server.

The 'FTP' class is part of the 'ftplib' library and represents an FTP connection. It provides methods for interacting with an FTP server, such as logging in, changing the current working directory, listing files in a directory, and downloading or uploading files.

The 'with' statement is a Python language construct that is used to create a context in which a resource, such as a file or network connection, is used. In this case, the 'with' statement creates a context in which the 'FTP' connection is used. The 'as' keyword is used to assign the 'FTP' object to a variable named 'ftp', which can then be used to interact with the FTP server.

When the ‘with’ block is exited, the ‘FTP’ connection is automatically closed and any resources associated with the connection, such as sockets or file handles, are released. This helps to ensure that resources are properly cleaned up and that the program does not leak resources or leave connections open.

d2) Listing the files in the FTP directory

The FTP.nlst() method is a function of the ‘ftplib’ library in Python that is used to list the files in the current working directory on an FTP server. The method returns a list of filenames as string, which can be used to iterate through and perform operations on the files.

d3) Checking if the local directory exits if not I create it

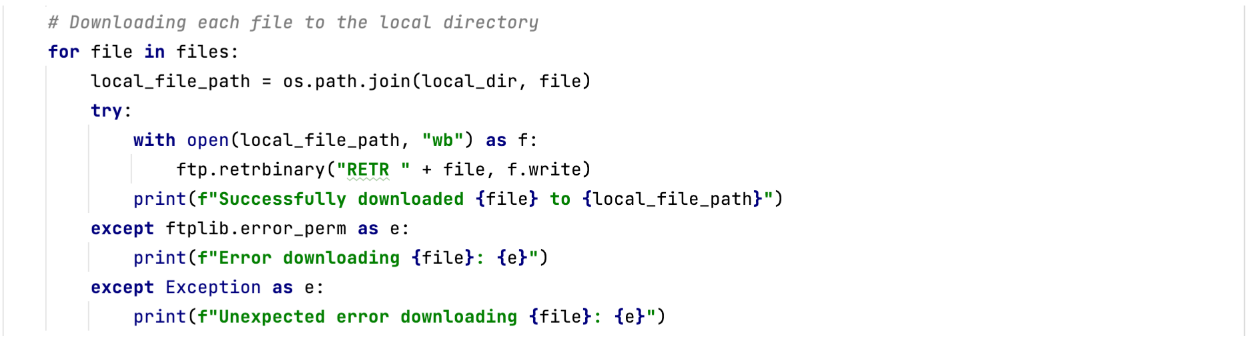
We do this by using the os library.

os.path.exists(local_dir)

If not —> os.makedirs(local_dir)

d4) Downloading each file to the local directory

I do that using the following block of code:



```python
# Downloading each file to the local directory
for file in files:
    local_file_path = os.path.join(local_dir, file)
    try:
        with open(local_file_path, "wb") as f:
            ftp.retrbinary("RETR " + file, f.write)
        print(f"Successfully downloaded {file} to {local_file_path}")
    except ftplib.error_perm as e:
        print(f"Error downloading {file}: {e}")
    except Exception as e:
        print(f"Unexpected error downloading {file}: {e}")
```

First there is a loop that will allow me to loop through all the files that I detected in the FTP server.

I create the variable local_file_path = os.path.join(local_dir, file). That will assign to each file the proper path.

Now the next block of code inside the 'try:'

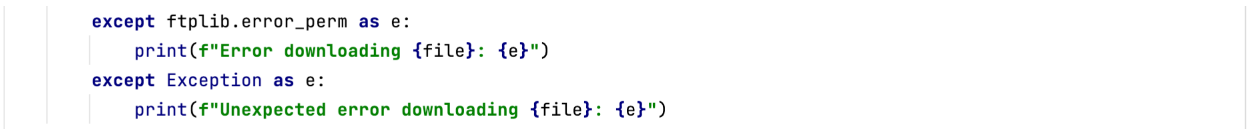
In this code block, the Python 'open()' function is used to create a local file object named 'f' in binary mode with the specified local file path and mode ("wb" means write binary mode). This file object will be used to write the contents of the file that is being download from the FTP server.

The 'with' statement is used to create a context in which the file object is used. The advantage of using the 'with' statement is that it automatically closes the file object when the block is exited, ensuring that resources are properly cleaned up.

The 'ftp.retrbinary()' method is then called to retrieve the contents of the file from the FTP server. The 'RETR' command is a standard FTP command that is used to retrieve a file from the server. The 'ftp.retrbinary()' sends the 'RETR' command to the server and downloads the file contents in binary mode, writing the contents to the local file object 'f' using its 'write()' method.

Overall, this code block downloads a file from an FTP server and writes its contents to a local file. The local file object is opened and close automatically using the 'with' statement, and the contents of the file are downloaded and written to the file using the ftp.retrbinary()' method.

except block —> Since I encounter some problems I had to create a try-except block that will catch any error and let the program run and accomplish the the task.



```
        except ftplib.error_perm as e:
            print(f"Error downloading {file}: {e}")
        except Exception as e:
            print(f"Unexpected error downloading {file}: {e}")
```

This block of code handles exceptions that may occur while downloading a file from a FTP server using the ‘ftplib’ library.

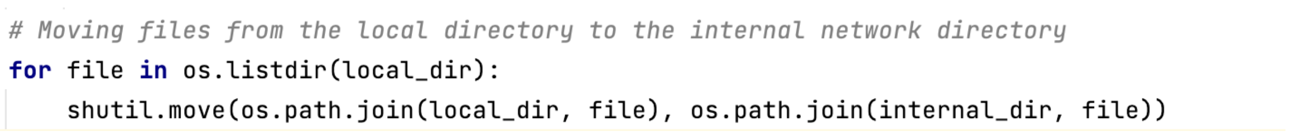
The ‘try’ block contains the code that may raise an exception. In this case, the exception that may be raised is ‘ftplib.error_perm’, which is raised when an FTP command fails with a permanent error response. For example, if the file does not exist on the server, the ‘RETR’ command will fail with a ‘550 Failed to open file’ error.

The ‘except’ block immediately following the ‘try’ block specifies the exception type that will be caught and handle. In this case, if an ‘ftplib.error_perm’ exception is raised, the code inside the ‘except’ block will be executed. The code inside the ‘except’ block prints an error message indicating that the file could not be downloaded and includes the specific error message that was raised.

The second ‘except’ block is a catch-all exception handler that will catch any other exceptions that were not caught by the first ‘except’ block. This is useful for handling unexpected errors that may occur during the download process, such as network errors or file system errors. The code inside this block also prints an error message that includes the specific error that was raised.

By handling exceptions in this way, the code is more robust and can recover from errors without crashing. It also provides useful feedback to the user about what went wrong and why the file could not be downloaded.

d5) Moving files from the local directory to the internal directory



```python
# Moving files from the local directory to the internal network directory
for file in os.listdir(local_dir):
    shutil.move(os.path.join(local_dir, file), os.path.join(internal_dir, file))
```

This will be easily done using the ‘shutil’ library.

Looping through all the files, I move them to the internal directory.

d6) Logging successful transfer message

```
# Logging successful transfer message
logging.info("Files transferred successfully")
```

We use one of the methods that the ‘logging’ library provides. So we can see the result of the whole process.

E) Scheduling the script to run daily at a specific time

```
# Scheduling the script to run daily at a specific time
schedule.every().day.at("16:53").do(download_files)
```

That sentence of code shedules a recurring task using the ‘schedule’ library in Python. The ‘schedule.every()’ function returns a ‘Schedule’ object that can be used to specify the frequency and timing of the schedule task.

In this case the ‘day’ method is used to specify that the task should run every day, and the ‘at’ method is used to specify the time of day at which the task should run, in 24-hour format.

Finally, the ‘do()’, method is used to specify the function that should be called when the task runs. In this case, the ‘download_files()’ function is specified, which is the function that downloads files from the FTP server and moves them to the internal network directory.

F) Setting up logging

```
# Setting up logging
logging.basicConfig(filename="file_transfer.log", level=logging.INFO)
```

This code configures the ‘logging’ library in Python to output log messages to a file named “file_transfer.log” and sets the logging level to ‘logging.INFO’.

The ‘logging.basicConfig()’ function is used to configure the logging system with basic configuration settings. In this case, the ‘filename’ parameter is used to specify the name of the log file that will be written to, and the ‘level’ parameter is used to set the minimum level of log messages that will be output to the file.

The ‘logging.INFO’ level specifies that messages with a severity level of ‘INFO’ or higher will be output to the log file. The available severity levels in the ‘logging’ library, in order of increasing severity, are ‘DEBUG’, ‘INFO’, ‘WARNING’, ‘ERROR’, and ‘CRITICAL’.

Overall, this code configures the logging system to output log messages to a file and sets the minimum severity level of messages that will be output to the file. This allows the program to capture and store information about what it is doing, making it easier to debug problems and understand how the program is working.

G) Defining the ‘dunder main’ block

```
# Defining the 'dunder main' block
if __name__ == '__main__':
    # Start scheduled script
    while True:
        schedule.run_pending()
```

Eventually I create a ‘dunder main’ block. So the execution of the scrip is under control.

And inside a loop I call ‘schedule.run_pending()’.

The ‘run_pending()’ method is used to check if any scheduled task are due to be run, and if so, to run them. When called, the ‘run_pending()’ method checks each scheduled task and runs any tasks whose schedule time has passed since the last time ‘run_pending()’ was called.

4. Conclusions

Despite of being consider as an advance task, it is resolved in less than 60 lines of code. The difficulty here is knowing how connect and with the FTP server and use properly the ftplib methods. If you know how to do that you can do it without problem.