Test Task

Please implement a program that synchronizes two folders: source and replica. The program should maintain a full, identical copy of source folder at replica folder.
Solve the test task by writing a program in one of these programming languages:

------- Python -------

C/C++

C#

- Synchronization must be one-way: after the synchronization content of the replica folder should be modified to exactly match content of the source folder;

```
# Function to calculate file MD5
def file_md5(filename):
    hash_md5 = hashlib.md5()
    with open(filename, "rb") as f:
        for chunk in iter(lambda: f.read(4096), 
b""):
            hash_md5.update(chunk)
    return hash_md5.hexdigest()

# Function to synchronize folders
def sync_folders(source, replica):
    # Ensure trailing slash for correct path 
replacement
    source = os.path.join(source, '')
    replica = os.path.join(replica, '')

    # Copy or update files from source to replica
    for root, dirs, files in os.walk(source):
        replica_root = root.replace(source, 
replica)
        if not os.path.exists(replica_root):
            os.makedirs(replica_root)
```

```python
            logging.info(f"Directory created: {replica_root}")

        for file in files:
            source_file = os.path.join(root, file)
            replica_file = os.path.join(replica_root, file)
            if not os.path.exists(replica_file) or file_md5(source_file) != file_md5(replica_file):
                logging.info(f"Copying: {source_file} to {replica_file}")
                shutil.copy2(source_file, replica_file)

    # Remove files and directories no longer present in source
    for root, dirs, files in os.walk(replica, topdown=False):
        source_root = root.replace(replica, source)
        for file in files:
            replica_file = os.path.join(root, file)
            source_file = os.path.join(source_root, file)
            if not os.path.exists(source_file):
                logging.info(f"Removing: {replica_file}")
                os.remove(replica_file)
        for dir in dirs:
            replica_dir = os.path.join(root, dir)
            if not os.listdir(replica_dir):
                logging.info(f"Removing directory: {replica_dir}")
                os.rmdir(replica_dir)

# Main loop to run synchronization periodically
if __name__ == "__main__":
    while True:
        logging.info("Starting synchronization")
        sync_folders(args.source, args.replica)
        logging.info("Synchronization complete.
Waiting for the next interval.")
        time.sleep(args.interval)
```

- Synchronization should be performed periodically.

```
# Main loop to run synchronization periodically
if __name__ == "__main__":
    while True:
        logging.info("Starting synchronization")
        sync_folders(args.source, args.replica)
        logging.info("Synchronization complete. Waiting for
the next interval.")
        time.sleep(args.interval)
```

- File creation/copying/removal operations should be logged to a file and to the console output;

```
# Parse command line arguments
parser = argparse.ArgumentParser(description="Synchronize
two folders.")
parser.add_argument("source", help="Source folder path")
parser.add_argument("replica", help="Replica folder path")
parser.add_argument("interval", type=int,
help="Synchronization interval in seconds")
parser.add_argument("log_file", help="Path to the log file")
args = parser.parse_args()
```

```
# Setup logging
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(name)s -
%(levelname)s - %(message)s',
                    handlers=[
                        RotatingFileHandler(args.log_file,
maxBytes=5*1024*1024, backupCount=2),
                        logging.StreamHandler()
                    ])
```

- Folder paths, synchronization interval and log file path should be provided using the command line arguments;

```
# EXAMPLE OF COMMAND FOR SHELL
#    & C:/Python312/python.exe
c:/Users/Administrator/Desktop/5454g/aaaa/qa-testing.py
"C:\Users\Administrator\Desktop\SourceFolder"
"C:\Users\Administrator\Desktop\ReplicaFolder" 300
"C:\Users\Administrator\Desktop\sync_log.log"
# 300s OF INTERVAL OF SINC (THE 300 VALUE)
# I CHOSE TO DROP THE LOG ON DESKTOP, CHOSE YOUR OWN DESIRED
FOLDER

# THOSE ARE MY FOLDERS, ADAPT FOR TESTING!
```

- It is undesirable to use third-party libraries that implement folder synchronization;

  Okaaay

- It is allowed (and recommended) to use external libraries implementing other well-known algorithms. For example, there is no point in implementing yet another function that calculates MD5 if you need it for the task – it is perfectly acceptable to use a third-party (or built-in) library.

```
import argparse
```

```python
import logging
import os
import shutil
import time
import hashlib
from logging.handlers import RotatingFileHandler
```

```python

# Daniel T K
# Code for QA - Python version

# EXAMPLE OF COMMAND FOR SHELL
#    & C:/Python312/python.exe
c:/Users/Administrator/Desktop/5454g/aaaa/qa-testing.py
"C:\Users\Administrator\Desktop\SourceFolder"
"C:\Users\Administrator\Desktop\ReplicaFolder" 300
"C:\Users\Administrator\Desktop\sync_log.log"
# 300s OF INTERVAL OF SINC (THE 300 VALUE)
# I CHOSE TO DROP THE LOG ON DESKTOP, CHOSE YOUR OWN DESIRED
FOLDER

# THOSE ARE MY FOLDERS, ADAPT FOR TESTING!

import argparse
import logging
import os
import shutil
import time
import hashlib
from logging.handlers import RotatingFileHandler
```

```python
# Parse command line arguments
parser = argparse.ArgumentParser(description="Synchronize
two folders.")
parser.add_argument("source", help="Source folder path")
parser.add_argument("replica", help="Replica folder path")
parser.add_argument("interval", type=int,
help="Synchronization interval in seconds")
parser.add_argument("log_file", help="Path to the log file")
args = parser.parse_args()

# Setup logging
logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(name)s -
%(levelname)s - %(message)s',
                    handlers=[
                        RotatingFileHandler(args.log_file,
maxBytes=5*1024*1024, backupCount=2),
                        logging.StreamHandler()
                    ])

# Function to calculate file MD5
def file_md5(filename):
    hash_md5 = hashlib.md5()
    with open(filename, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b""):
            hash_md5.update(chunk)
    return hash_md5.hexdigest()

# Function to synchronize folders
def sync_folders(source, replica):
    # Ensure trailing slash for correct path replacement
    source = os.path.join(source, '')
    replica = os.path.join(replica, '')

    # Copy or update files from source to replica
    for root, dirs, files in os.walk(source):
        replica_root = root.replace(source, replica)
        if not os.path.exists(replica_root):
            os.makedirs(replica_root)
            logging.info(f"Directory created:
{replica_root}")

        for file in files:
            source_file = os.path.join(root, file)
            replica_file = os.path.join(replica_root, file)
            if not os.path.exists(replica_file) or
file_md5(source_file) != file_md5(replica_file):
                logging.info(f"Copying: {source_file} to
{replica_file}")
                shutil.copy2(source_file, replica_file)

    # Remove files and directories no longer present in
source
    for root, dirs, files in os.walk(replica,
topdown=False):
        source_root = root.replace(replica, source)
        for file in files:
            replica_file = os.path.join(root, file)
            source_file = os.path.join(source_root, file)
            if not os.path.exists(source_file):
                logging.info(f"Removing: {replica_file}")
                os.remove(replica_file)
        for dir in dirs:
            replica_dir = os.path.join(root, dir)
            if not os.listdir(replica_dir):
                logging.info(f"Removing directory:
{replica_dir}")
                os.rmdir(replica_dir)

# Main loop to run synchronization periodically
if __name__ == "__main__":
    while True:
        logging.info("Starting synchronization")
        sync_folders(args.source, args.replica)
        logging.info("Synchronization complete. Waiting for the next interval.")
        time.sleep(args.interval)
```