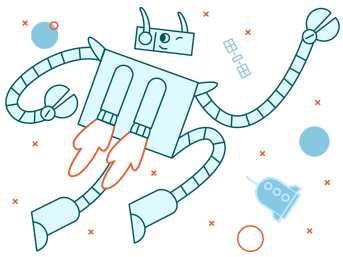
Data Safe



In a bustling digital world, where data is as precious as gold, a small startup named "DataSafe" is on a mission to safeguard their treasure trove of digital assets. Amidst their rapid growth, the team at DataSafe realized that their invaluable project files, customer data, and creative assets were at risk of being lost to the ether of digital calamity.

Recognizing the need for a guardian to protect their digital domain, they tasked their brightest developer with crafting a tool, a File Backup Utility, capable of not only guarding their assets through backups but also compressing the files to ensure their vaults would never overflow. Thus, the journey to forge this digital safeguard began, promising security and peace of mind in the ever-volatile realm of bytes and bits.

Objective

Create a utility script for backing up files to a specified directory. This utility should allow the user to specify source and destination directories, implement incremental backup (copy files that have changed since last backup), optionally compress backup files to save space, and log each backup operation's details.

Task

- **Source and Destination:** Prompt the user to specify the paths for the source directory (files to back up) and the destination directory (where backups are stored).
- **Incremental Backup:** Only copy files that have changed since the last backup (based on modification time).
- **Optional Compression:** Provide the option to compress the backup files using gzip to save space.

- **Logging:** Log the details of each backup operation, including time, number of files backed up, and total duration.

Guidelines

1. **Parsing Command Line Input:** Use `sys.argv` to get source and destination directory paths, and whether to compress the files.
    - If the word "compress" appears after the source and destination paths, the script should compress the files into a single zip.
2. **Incremental Backup Logic:** Use `os.path.getmtime()` to compare modification times and determine if a file should be backed up again.
3. **File Compression:** Utilize the `zipfile` module for optional file compression. Wrap files in a `.zip` format if the user chooses compression.
4. **Directory Management:** Employ `os.makedirs()` to create the backup directory structure as needed, ensuring that the full path exists before copying files.
5. **File Copying and Compression:** Leverage the `shutil` module for copying files (`shutil.copy2()`) and for file-object operations if compressing.
6. **Logging:** Write backup details to a log file. Consider including timestamp, number of files backed up, and operation duration.

Hints

- `shutil` **Module:** Use `shutil.copy2()` to copy files to the backup directory, preserving metadata. For copying file objects (e.g., when compressing), look into `shutil.copyfileobj()`.
- `zipfile` **Module:** To compress files, open the source file in read-binary (`'rb'`) mode, and create a zip file in write-binary (`'wb'`) mode.
- `os.makedirs()`: This function allows you to create the destination directory, including any necessary parent directories, with

`exist_ok=True` to prevent errors if the directory already exists.

- `os.walk()`: This function generates all the file names in a directory tree. Use it to gather all files from the source directory for backup, allowing you to replicate the directory structure or compress the files as needed.

Example Usage

The script should be executable from the command line, with user input guiding the process.

```
> python data_safe.py /path/to/source
/path/to/destination compress
Backup completed at YYYY-MM-DD HH:MM:SS, Duration:
XX.XX seconds, Files backed up: N
```

To Submit

Submit your `data_safe.py` script.