Lab Program 1: Automate File Renaming

Aim: To automate the renaming of multiple files in a directory by adding a prefix.

Procedure:
1. Import the necessary modules.
2. Define the directory and the prefix.
3. Iterate through the files in the directory and rename them.

Code: python
```
import os

# Define the directory and prefix
directory = 'path/to/directory'
prefix = 'new_'

# Iterate through the files and rename them
for filename in os.listdir(directory):
new_name = prefix + filename
os.rename(os.path.join(directory, filename), os.path.join(directory, new_name))

print("Files renamed successfully!")
```

Solution: This script automates renaming files by adding a specified prefix to each file name in the given directory.

Result: The files in the specified directory will have the prefix added to their names.

Output:

```
# Automate file renaming
import os

directory ='/content/newfolder'
prefix = 'alok_'
for filename in os.listdir(directory):
    new_name = prefix + filename
    os.rename(os.path.join(directory, filename), os.path.join(directory, new_name))
print("Files renamed successfully")
```

```
Files renamed successfully
```

Lab Program 2: Automate Email Sending

Aim: To automate the process of sending emails using Python.

Procedure:
1. Import the necessary modules.
2. Define the sender, receiver, subject, and body of the email.
3. Use the `smtplib` library to send the email.

Code: python
```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# Email details
sender_email = "sa5436@srmist.edu.in"
receiver_email = "shobhitadhya@gmail.com"
password = "#shobhitkumar"
subject = "Automated Email"
body = "This is an automated email sent using Python."

# Create the email
msg = MIMEMultipart()
msg['From'] = sender_email
msg['To'] = receiver_email
msg['Subject'] = subject
msg.attach(MIMEText(body, 'plain'))

# Send the email
server = smtplib.SMTP('smtp.example.com', 587)
server.starttls()
server.login(sender_email, password)
text = msg.as_string()
server.sendmail(sender_email, receiver_email, text)
server.quit()

print("Email sent successfully!")
```

Solution: This script automates the sending of an email by logging into an SMTP server and sending the email to the specified recipient.

Result: The email will be sent to the specified receiver.

Output:

```
server.quit()
print("Email sent successfully")
```

```
Email sent successfully
```

Lab Program 3: Automate Web Scraping

Aim: To automate web scraping to extract data from a webpage.

Procedure:
1. Import the necessary modules.
2. Define the URL of the webpage.
3. Use `requests` to fetch the webpage content.
4. Use `BeautifulSoup` to parse the HTML content.
5. Extract and display the desired data.

Code: python
```
import requests
from bs4 import BeautifulSoup

# URL of the webpage to scrape
url = 'https://example.com'

# Fetch the webpage content
response = requests.get(url)
webpage = response.content

# Parse the HTML content
soup = BeautifulSoup(webpage, 'html.parser')

# Extract data (example: all paragraph texts)
paragraphs = soup.find_all('p')
for p in paragraphs:
 print(p.text)
```

Solution: This script automates the extraction of data from a webpage by fetching and parsing the HTML content.

Result: The paragraph texts from the specified webpage will be displayed.

Output:

```
Intelligent automation (IA), sometimes called cognitive automation, is the use of automat
Intelligent automation simplifies processes, frees up resources and improves operational
Intelligent automation comprises 3 cognitive technologies. The integration of these compo
This integration leads to a transformative solution that streamlines processes and simpli
Learn how intelligent automation can make your business operations a competitive advantag
Register for TEI Report for IBM Robotic Process Automation
Intelligent automation platforms provide many benefits across industries as a result of u
Explore the IBM Cloud Paks for Automation
Read the blog: Intelligent automation gives humans the gift of time
Read the IDC study: Big payoffs from big bets in AI-powered automation
```

Lab Program 4: Automate Data Backup

Aim: To automate the backup of files from one directory to another.

Procedure:
1. Import the necessary modules.
2. Define the source and destination directories.
3. Iterate through the files in the source directory and copy them to the destination directory.

Code: python
```
import shutil
import os

# Define source and destination directories
source_dir = 'path/to/source_directory'
destination_dir = 'path/to/destination_directory'

# Iterate through files and copy them
for filename in os.listdir(source_dir):
 shutil.copy(os.path.join(source_dir, filename), destination_dir)

print("Files backed up successfully!")
```

Solution: This script automates the backup of files by copying them from the source directory to the destination directory.

Result: Files from the source directory will be copied to the destination directory.

Output:

```
import shutil
import os
source_dir='/content/newfolder'
dest_dir='/content/backupdir'
for filename in os.listdir(source_dir):
        shutil.copy(os.path.join(source_dir,filename),dest_dir)
print("Files backed up successfully")
```

```
Files backed up successfully
```

Lab Program 5: Automate Database Backup

Aim: To automate the backup of a MySQL database.

Procedure:
1. Import the necessary modules.
2. Define the database connection details and the backup file path.
3. Use `mysqldump` to backup the database.

Code: python
```
import os

# Database connection details
user = 'shobhit'
password = '#shobhitkumar'
database = 'employee'
backup_file = 'path/to/backup_file.sql'

# Command to backup the database
command = f"mysqldump -u {user} -p{password} {database} > {backup_file}"
os.system(command)

print("Database backed up successfully!")
```

Solution: This script automates the backup of a MySQL database using the `mysqldump` command.

Result: The database will be backed up to the specified file.

Output:

```
import os

# Database connection details
user = 'shobhit'
password = '#shobhitkumar'
database = 'employee'
backup_file = 'path/to/backup_file.sql'

# Command to backup the database
command = f"mysqldump -u {user} -p{password} {database} > {backup_file}" os.system(command)

print("Database backed up successfully!")
```

```
Database backed up successfully
```

Lab Program 6: Automate System Monitoring

Aim: To automate the monitoring of system resources such as CPU and memory usage.

Procedure:
1. Import the necessary modules.
2. Use `psutil` to fetch system resource usage data.
3. Display the data.

Code: python
```
import psutil

# Fetch and display CPU usage
cpu_usage = psutil.cpu_percent(interval=1)

print(f"CPU Usage: {cpu_usage}%")

# Fetch and display memory usage
memory_info = psutil.virtual_memory()

print(f"Memory Usage: {memory_info.percent}%")
```

Solution: This script automates the monitoring of system resources by fetching and displaying CPU and memory usage data.

Result: The CPU and memory usage data will be displayed.

Output:

```
[9] import psutil
    cpu_usage=psutil.cpu_percent(interval=1)
    print(f'CPU Usage:{cpu_usage}%')

    memory_info=psutil.virtual_memory()
    print(f'Memory Usage:{memory_info.percent}%')

CPU Usage:3.0%
Memory Usage:8.9%
```

Lab Program 7: Automate File Compression

Aim: To automate the compression of files into a ZIP archive.

Procedure:
1. Import the necessary modules.
2. Define the directory containing the files and the output ZIP file path.
3. Use `zipfile` to compress the files.

Code: python
```
import zipfile
import os

# Define the directory and output ZIP file path
directory = 'path/to/directory'
zip_file = 'path/to/output.zip'

# Create a ZIP file and add files
with zipfile.ZipFile(zip_file, 'w') as zipf:
 for foldername, subfolders, filenames in os.walk(directory):  for filename in filenames:
 file_path = os.path.join(foldername, filename)
 zipf.write(file_path, os.path.relpath(file_path, directory))

print("Files compressed successfully!")
```

Solution: This script automates the compression of files into a ZIP archive.

Result: The files in the specified directory will be compressed into the output ZIP file.

Output:

```
import zipfile
import os
# Define the directory and output ZIP file path
directory = '/content/sample_data/california_housing_train.csv'
zip_file = '/content/sample_data/untitled'
# Create a ZIP file and add files
with zipfile.ZipFile(zip_file, 'w') as zipf:
 for foldername, subfolders, filenames in os.walk(directory):
  for filename in filenames:
    file_path = os.path.join(foldername, filename)
    zipf.write(file_path, os.path.relpath(file_path, directory))
print("Files compressed successfully!")
```

```
Files compressed successfully!
```

Lab Program 8: Automate Image Processing

Aim: To automate the process of resizing multiple images.

Procedure:
1. Import the necessary modules.
2. Define the directory containing the images and the desired size.
3. Use `PIL` to resize the images.

Code: python
```
from PIL import Image
import os

# Define the directory and desired size
directory = 'path/to/images'
size = (800, 600)

# Iterate through the images and resize them
for filename in os.listdir(directory):
 if filename.endswith(('.jpg', '.png')):
 img = Image.open(os.path.join(directory, filename))  img = img.resize(size)
 img.save(os.path.join(directory, 'resized_' + filename))

print("Images resized successfully!")
```

Solution: This script automates the resizing of images to the specified size.

Result: The images in the specified directory will be resized to the desired size.

Output:

```
from PIL import Image
import os
# Define the directory and desired size
directory = '/content/files'
size = (800, 600)
# Iterate through the images and resize them
for filename in os.listdir(directory):
 if filename.endswith(('.jpg', '.png')):
  img = Image.open(os.path.join(directory, filename))
  img = img.resize(size)
  img.save(os.path.join(directory, 'resized_' + filename))
print("Images resized successfully!")
```

```
Images resized successfully!
```