

Report

Directory Compression and Decompression

This report provides an overview of two Python scripts that enable directory compression into a ZIP file and directory decompression from a ZIP file. The first script, "Compressing Files," allows the user to compress files within a directory into a ZIP archive, calculate the compression ratio, and display file sizes. The second script, "Decompressing Files," enables users to extract the contents of a ZIP file into a specified directory.

- Compressing Directory
 - The script prompts the user to enter the path to a directory containing files to be compressed.
 - It verifies whether the input directory exists. If not, it displays an error message and exits.
 - The user is prompted to specify the path and filename for the output ZIP file.
 - The script utilizes the `'zipfile'` module to compress the directory into a ZIP file while preserving the directory structure. The `'ZIP_DEFLATED'` compression method is used.
 - Original file sizes within the directory and the compressed ZIP file size are calculated using the `'os.path.getsize'` function.
 - The compression ratio, defined as the ratio of compressed file size to original file size, is calculated and displayed.
 - A success message indicates the completion of directory compression.
- Decompressing ZIP File
 - The script prompts the user to enter the path to the ZIP file to be decompressed.
 - It checks whether the specified ZIP file exists. If the file is not found, it displays an error message and exits.
 - The user is prompted to specify the path for extracting the ZIP file's contents.
 - The script ensures that the extraction path exists or creates it if necessary.
 - The `'zipfile'` module is used to decompress the ZIP file, extracting its contents.
 - A success message indicates the completion of ZIP file extraction.

These scripts offer practical functionality for managing files and directories:

- **Compressing Files:** Useful for bundling multiple files into a single compressed archive, reducing storage space, and simplifying file transfer.
 - **Data Backup and Archiving:** Users can compress directories containing valuable data, creating efficient backups that consume less storage space.
 - **File Transfer:** Compressed ZIP files are ideal for sending large sets of files over networks or via email, as they simplify the transfer process and reduce transmission times.
 - **Software Packaging:** Developers use directory compression to package software and its associated resources into a single distributable ZIP archive.

- **Decompressing Files:** Essential for extracting the contents of ZIP archives, whether for data recovery, file access, or installation purposes.
 - **Data Recovery:** Decompression is vital for recovering data from compressed archives, making it possible to access files that may have been backed up in ZIP format.
 - **Software Installation:** Many software packages are distributed in ZIP archives, and decompression is the first step in the installation process.
 - **File Access:** ZIP files are commonly used to group related files, and decompression provides a convenient method for accessing these files without extracting the entire archive.

In conclusion, these Python scripts provide a straightforward and user-friendly means of compressing directories into ZIP files and decompressing ZIP files into specified directories. They demonstrate the capabilities of the `'zipfile'` module and the `'os'` module for file handling and manipulation. These tools can be invaluable for file management tasks in various applications, including data backup, software distribution, and data extraction.

Compression

The screenshot displays a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project structure under 'PYTHON LABS' with folders 'Module-1', 'Module-2', 'Lab-1', 'Lab-2', 'Lab-3', and 'Task-1', 'Task-2', 'Task-3'. The 'Task-3' folder is selected, showing a 'Testing-Folder' and a 'Compressed-File.zip' file. The 'compression.py' file is open in the editor.

The code in 'compression.py' defines a function 'compress_directory' and a 'main' function. The 'main' function prompts the user for the input directory and output ZIP file path, checks if the directory exists, compresses it, and prints the original size, compressed size, and compression ratio.

```

24
25 def compress_directory(input_dir, output_zip):
26     # Compresses a directory into a ZIP file
27     with zipfile.ZipFile(output_zip, 'w', zipfile.ZIP_DEFLATED) as zipf:
28         for root, _, files in os.walk(input_dir):
29             for file in files:
30                 file_path = os.path.join(root, file)
31                 arcname = os.path.relpath(file_path, input_dir)
32                 zipf.write(file_path, arcname)
33
34 def main():
35     # Prompt the user to enter the path to the directory containing files to be compressed
36     input_dir = input("Enter the path to the directory to compress: ")
37
38     # Check if the input directory exists
39     if not os.path.exists(input_dir):
40         print("Directory not found.")
41         return
42
43     # Prompt the user to enter the path and filename for the output ZIP file
44     output_zip = input("Enter the path and filename for the output ZIP file: ")
45
46     # Compress the directory
47     compress_directory(input_dir, output_zip)
48
49     # Get file sizes
50     original_size = sum(os.path.getsize(os.path.join(root, file)) for root, _, files in os.walk(input_dir))
51     compressed_size = os.path.getsize(output_zip)
52
53     # Calculate compression ratio
54     compression_ratio = (compressed_size / original_size) * 100
55
56     # Display file sizes and compression ratio
57     print(f"Original directory size: {original_size} bytes")
58     print(f"Compressed ZIP file size: {compressed_size} bytes")
59     print(f"Compression ratio: {compression_ratio:.2f}%")
60
61     # Print a success message
62     print("Directory compression successful.")
63
64 if __name__ == "__main__":
65     main()
66

```

The terminal output shows the execution of the script. The user enters the path to the directory to compress and the path and filename for the output ZIP file. The script prints the original directory size, compressed ZIP file size, and compression ratio, and finally prints a success message.

```

User: harshsiddhapura
Time: 2023-09-01 23:36:05.427590
Computer Info: posix
Enter the path to the directory to compress: /Users/harshsiddhapura/Harsh/Education/MS_IT/Sem-1/IFT510 - Architecture /Python Labs/Module-2/Lab-3/Task-3/Testing-Folder
Enter the path and filename for the output ZIP file: /Users/harshsiddhapura/Harsh/Education/MS_IT/Sem-1/IFT510 - Architecture /Python Labs/Module-2/Lab-3/Task-3/Compressed-File.zip
Original directory size: 113083 bytes
Compressed ZIP file size: 109854 bytes
Compression ratio: 97.14%
Directory compression successful.
(.venv) harshsiddhapura@Harshs-MacBook-Air Task-3 %

```

Decompression

The screenshot displays a VS Code editor window titled 'Python Labs'. The Explorer panel on the left shows the project structure:

- PYTHON LABS
 - > .env
 - > Module-1
 - > Module-2
 - > Lab-1
 - > Lab-2
 - > Lab-3
 - > Task-1
 - > Task-2
 - > Task-3
 - > Decompressed-Fold...
 - > Testing-Folder
 - Compressed-File.zip
 - Compression-Scre... U
 - compression.py U
 - decompression.py U
 - > Task-4

The main editor shows the file 'decompression.py' with the following code:

```

10 def print_system_info():
11     # Get user data
12     os.system('clear') # os.system('clear') for Linux
13     username = getpass.getuser()
14     # Get computer information
15     computer_info = os.name
16     # Get current date and time
17     current_time = datetime.datetime.now()
18     # Format log message
19     log_message = f"User: {username}\nTime:{current_time}\nComputer Info: {computer_info}"
20     # Print log message
21     print(log_message)
22 # Call the function to print the log
23 print_system_info()
24
25 def decompress_zip(zip_file, extract_path):
26     # Decompress a ZIP file to the specified directory
27     with zipfile.ZipFile(zip_file, 'r') as zip_ref:
28         zip_ref.extractall(extract_path)
29
30 def main():
31     # Prompt the user to enter the path to the ZIP file to be decompressed
32     zip_file = input("Enter the path to the ZIP file to be decompressed: ")
33
34     # Check if the ZIP file exists
35     if not os.path.exists(zip_file):
36         print("ZIP file not found.")
37         return
38
39     # Prompt the user to enter the path for extracting the contents
40     extract_path = input("Enter the path for extracting the contents: ")
41
42     # Check if the extraction path exists or create it if not
43     if not os.path.exists(extract_path):
44         os.makedirs(extract_path)
45
46     # Decompress the ZIP file
47     decompress_zip(zip_file, extract_path)
48
49     # Print a success message
50     print("ZIP file extraction successful.")
51
52 if __name__ == "__main__":
53     main()

```

The Terminal panel at the bottom shows the execution output:

```

zsh - Task-3
User: harshsiddhapura
Time:2023-09-02 00:15:05.465869
Computer Info: posix
Enter the path to the ZIP file to be decompressed: /Users/harshsiddhapura/Harsh/Education/MS_IT/Sem-1/IFT510 - Architecture
/Python Labs/Module-2/Lab-3/Task-3/Compressed-File.zip
Enter the path for extracting the contents: /Users/harshsiddhapura/Harsh/Education/MS_IT/Sem-1/IFT510 - Architecture /Python
Labs/Module-2/Lab-3/Task-3/Decompressed-Folder
ZIP file extraction successful.
(.venv) harshsiddhapura@Harshs-MacBook-Air Task-3 %

```

The status bar at the bottom indicates the current file is 'main.py', the encoding is 'UTF-8', the line length is 20, and the Python version is 3.9.6.