1. How do you distinguish between shutil.copy() and shutil.copytree()?

Ans : The shutil.copy() and shutil.copytree() functions are both part of the shutil module in Python, and they serve different purposes for copying files and directories.

1. **shutil.copy(src, dst)**: This function is used to copy a single file from the source (**src**) to the destination (**dst**). Here are some characteristics of **shutil.copy()**:
   * It copies the file pointed to by **src** to the location specified by **dst**.
   * If **dst** specifies an existing file, it will be overwritten.
   * Parent directories for the destination file must already exist.
   * Metadata and permissions of the source file are preserved in the copied file.

Example:

import shutil shutil.copy('source\_file.txt', 'destination\_file.txt')

1. **shutil.copytree(src, dst)**: This function is used to recursively copy an entire directory from the source (**src**) to the destination (**dst**). Here are some characteristics of **shutil.copytree()**:
   * It copies the entire directory tree rooted at **src** to the location specified by **dst**.
   * If **dst** specifies an existing directory, the contents of the source directory are copied into the destination directory.
   * If **dst** specifies a non-existent directory, it will be created.
   * Metadata and permissions of files and directories are preserved in the copied directory tree.

Example:

import shutil shutil.copytree('source\_directory', 'destination\_directory')

In summary, **shutil.copy()** is used to copy individual files, while **shutil.copytree()** is used to copy entire directory trees. The choice between the two functions depends on your specific needs, whether you want to copy a single file or copy a directory along with its contents.

2. What function is used to rename files??

Ans : The function used to rename files in Python is os.rename(src, dst). It is part of the os module, which provides various operating system-related functionalities.

The **os.rename()** function is used to rename a file or directory. It takes two arguments: **src** and **dst**. Here's how it works:

* **src**: The source is the current name of the file or directory you want to rename.
* **dst**: The destination is the new name you want to assign to the file or directory.

Example of renaming a file:

import os os.rename('old\_filename.txt', 'new\_filename.txt')

In the above example, the file with the name **'old\_filename.txt'** is renamed to **'new\_filename.txt'**.

Please note that the **os.rename()** function can also be used to move a file or directory to a different location by specifying the new path in the **dst** argument.

3. What is the difference between the delete functions in the send2trash and shutil modules?

Ans : The send2trash and shutil modules in Python provide different approaches for deleting files or directories, and they have distinct differences:

1. **send2trash** module:
   * The **send2trash** module provides a safer way to delete files or directories by moving them to the operating system's trash or recycle bin instead of permanently deleting them.
   * The **send2trash.send2trash(path)** function moves the specified file or directory to the trash or recycle bin, allowing for potential recovery if needed.
   * It is useful when you want to delete files or directories but still have the option to restore them later.
   * Example:

from send2trash import send2trash send2trash('file\_to\_delete.txt')

1. **shutil** module:
   * The **shutil** module provides various file operations, including the **shutil.rmtree(path)** function, which is used to recursively delete an entire directory tree.
   * The **shutil.rmtree()** function permanently deletes the specified directory and all its contents. It does not move them to the trash or recycle bin.
   * It is useful when you want to completely remove a directory and its contents without the possibility of recovery.
   * Example:

import shutil shutil.rmtree('directory\_to\_delete')

In summary, the **send2trash** module allows you to send files or directories to the trash or recycle bin for potential recovery, while the **shutil** module provides functions to permanently delete files or directories without the possibility of recovery. The choice between the two depends on your specific requirements for handling file deletion.

4.ZipFile objects have a close() method just like File objects’ close() method. What ZipFile method is equivalent to File objects’ open() method?

Ans : he equivalent method in ZipFile objects to File objects' open() method is ZipFile.open(filename, mode='r', pwd=None).

The **ZipFile.open()** method is used to open a specific file within a ZIP archive and returns a file-like object representing that file. It allows you to access and read the contents of the individual file within the ZIP archive without extracting the entire archive.

Here's how you can use **ZipFile.open()** to open a file within a ZIP archive:

import zipfile with zipfile.ZipFile('archive.zip', 'r') as zf: with zf.open('file\_inside\_archive.txt', 'r') as file: # Perform operations on the file data = file.read() print(data)

In the above example, **zipfile.ZipFile()** is used to open the ZIP archive, and then **ZipFile.open()** is used to open the specific file **'file\_inside\_archive.txt'** within the archive. The resulting file-like object is then used to read the contents of the file.

Similar to **open()** method in **File** objects, the **ZipFile.open()** method provides a way to access the contents of a specific file within a ZIP archive without extracting it, giving you more flexibility when working with ZIP archives in Python.

5. Create a programme that searches a folder tree for files with a certain file extension (such as .pdf or .jpg). Copy these files from whatever location they are in to a new folder.

Ans : Certainly! Here's an example program that searches a folder tree for files with a specified file extension and copies them to a new folder:

import os import shutil def copy\_files\_by\_extension(source\_folder, target\_folder, file\_extension): # Create the target folder if it doesn't exist if not os.path.exists(target\_folder): os.makedirs(target\_folder) # Walk through the source folder tree and copy files with the given extension for root, dirs, files in os.walk(source\_folder): for file in files: if file.endswith(file\_extension): source\_path = os.path.join(root, file) target\_path = os.path.join(target\_folder, file) shutil.copy2(source\_path, target\_path) print(f"Copied: {source\_path} -> {target\_path}") # Example usage source\_folder = '/path/to/source\_folder' target\_folder = '/path/to/target\_folder' file\_extension = '.pdf' copy\_files\_by\_extension(source\_folder, target\_folder, file\_extension)

In this program:

1. Define the source folder (**source\_folder**), the target folder (**target\_folder**), and the desired file extension (**file\_extension**).
2. The **copy\_files\_by\_extension()** function is defined to handle the file copying operation.
3. The target folder is created if it doesn't exist using **os.makedirs()**.
4. The **os.walk()** function is used to traverse the source folder tree, including all subdirectories.
5. For each file encountered, the **endswith()** method is used to check if it has the specified file extension.
6. If the file matches the extension, it is copied using **shutil.copy2()** to the target folder.
7. The source and target paths are printed to provide feedback on the copied files.

Remember to replace **/path/to/source\_folder** and **/path/to/target\_folder** with the actual paths to your source and target folders, respectively. Additionally, modify the **file\_extension** variable to the desired file extension (e.g., **'.pdf'**, **'.jpg'**).

This program will search the source folder and all its subdirectories for files with the specified file extension and copy them to the target folder, preserving the directory structure.