**Assignment 10**

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

**shutil.copy()** and **shutil.copytree()** are both functions in the Python standard library **shutil** module that allow you to copy files and directories in your file system. The main difference between them is that **shutil.copy()** is used to copy individual files, while **shutil.copytree()** is used to copy directories and their contents recursively.

Here's a brief overview of each function:

* **shutil.copy(src, dst, \*, follow\_symlinks=True)** - This function copies the file at the path **src** to the path **dst**. The **follow\_symlinks** argument determines whether symbolic links are followed or copied as links. If it's **True**, the function will follow symbolic links and copy the file or directory they point to. If it's **False**, symbolic links will be copied as links instead.
* **shutil.copytree(src, dst, symlinks=False, ignore=None, copy\_function=copy2, ignore\_dangling\_symlinks=False, dirs\_exist\_ok=False)** - This function recursively copies the directory at the path **src** and all its contents to the directory at the path **dst**. The **symlinks** argument determines whether symbolic links are copied as links or followed. If it's **True**, symbolic links will be copied as links. If it's **False**, they will be followed and the files or directories they point to will be copied. The **ignore** argument can be used to specify a callable that determines which files or directories should be ignored during the copy. The **copy\_function** argument can be used to specify a different function to use for copying files (the default is **shutil.copy2()**). The **ignore\_dangling\_symlinks** argument determines whether dangling symbolic links (those that point to non-existent files or directories) should be ignored during the copy. The **dirs\_exist\_ok** argument determines whether it's okay for the destination directory to already exist; if it's **True**, the function will still copy the source directory into the destination directory, but won't overwrite any files that already exist.

In summary, **shutil.copy()** is used to copy individual files, while **shutil.copytree()** is used to copy directories and their contents recursively. **shutil.copy()** also provides more control over how symbolic links are handled during the copy.

2. What function is used to rename files??

In Python, the **os.rename()** function is used to rename files.

Here's the basic syntax of **os.rename()**:

pythonCopy code

import os os.rename(src, dst)

Here, **src** is the current file name/path, and **dst** is the new file name/path. When you call **os.rename()**, the file at the **src** path is renamed to the name/path specified in **dst**.

Here's an example of how to use **os.rename()** to rename a file:

pythonCopy code

import os old\_filename = "my\_file.txt" new\_filename = "new\_file.txt" os.rename(old\_filename, new\_filename)

In this example, the file "my\_file.txt" will be renamed to "new\_file.txt".

Note that if the file at the **dst** path already exists, it will be overwritten by the file being renamed. If you want to avoid overwriting existing files, you should check whether the **dst** file already exists before calling **os.rename()**.

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

The **send2trash** and **shutil** modules both provide functions for deleting files and directories, but there are some important differences between them.

The main difference is that the **send2trash** module sends files and directories to the operating system's trash or recycle bin, while the **shutil** module permanently deletes them.

Here's a brief overview of the delete functions in each module:

* **send2trash.send2trash(path)** - This function sends the file or directory at the path **path** to the operating system's trash or recycle bin. The **path** argument can be a string representing a file or directory path, or a **Path** object from the **pathlib** module. If the file or directory at **path** is successfully sent to the trash, the function returns **None**. If an error occurs, it raises an exception.
* **shutil.rmtree(path, ignore\_errors=False, onerror=None)** - This function permanently deletes the directory at the path **path** and all its contents, recursively. The **ignore\_errors** argument determines whether errors should be ignored during the deletion; if it's **True**, errors will be ignored and the deletion will continue. The **onerror** argument can be used to specify a callable that will be called with the arguments **(func, path, exc\_info)** for each error that occurs during the deletion.
* **os.remove(path)** - This function permanently deletes the file at the path **path**. If the file is successfully deleted, the function returns **None**. If an error occurs, it raises an exception.

In summary, the **send2trash** module sends files and directories to the operating system's trash or recycle bin, while the **shutil** module permanently deletes them. The **send2trash** module is useful if you want to allow the user to recover deleted files, while the **shutil** module should be used when you want to permanently delete files or directories.

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

The **ZipFile** class in Python's **zipfile** module provides a method called **ZipFile()** which is equivalent to the **open()** method of the file object. The **ZipFile()** method allows you to open a zip archive file and access its contents.

Here's an example of how to use the **ZipFile()** method to open a zip archive file:

pythonCopy code

import zipfile with zipfile.ZipFile('my\_archive.zip', 'r') as my\_zip: # Access files and directories inside the zip archive my\_zip.extractall()

In this example, we create a **ZipFile** object called **my\_zip** by calling the **ZipFile()** method with the file name "my\_archive.zip" and the mode "r" (for reading). We then use the **extractall()** method to extract all the contents of the zip archive to the current working directory.

Note that just like with file objects, you should always use the **with** statement when working with **ZipFile** objects to ensure that the file is closed properly after you're done working with it. The **close()** method of **ZipFile** can also be used to close the zip archive file, but it's better to use the **with** statement as it automatically handles closing the file for you.

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.

Here's an example program that searches for all PDF files in a directory tree and copies them to a new folder:

pythonCopy code

import os import shutil # Set the source and destination directories source\_dir = 'path/to/source/directory' dest\_dir = 'path/to/destination/directory' # Set the file extension to search for file\_extension = '.pdf' # Walk through the directory tree and copy files with the specified extension for root, dirs, files in os.walk(source\_dir): for file in files: if file.endswith(file\_extension): # Construct the full file paths for the source and destination files src\_file\_path = os.path.join(root, file) dest\_file\_path = os.path.join(dest\_dir, file) # Copy the file to the destination directory shutil.copy2(src\_file\_path, dest\_file\_path) print(f"All {file\_extension} files have been copied to {dest\_dir}")

Here's what the program does:

1. First, it sets the source and destination directories as well as the file extension to search for.
2. It then uses the **os.walk()** function to recursively walk through the source directory and all its subdirectories, looking for files with the specified extension.
3. For each file that matches the specified extension, it constructs the full file paths for the source and destination files.
4. Finally, it uses the **shutil.copy2()** function to copy the file from the source directory to the destination directory.
5. Once all files have been copied, the program prints a message indicating that the operation is complete.

Note that in this example, the **shutil.copy2()** function is used to copy the files. This function preserves the original file's metadata such as creation time, modification time, and permissions. If you don't need to preserve this metadata, you can use the simpler **shutil.copy()** function instead.