**Python Assignment 10**

Q1. How do you distinguish between shutil.copy() and shutil.copytree()?

Ans. shutil.copy() and shutil.copytree() are both functions in Python's shutil module that can be used to copy files and directories, but they are used for different purposes.

shutil.copy(src, dst) is used to copy a single file from one location to another. It takes two arguments - src, which is the path of the file to be copied, and dst, which is the destination path for the copied file. If the dst parameter specifies a directory, the file will be copied to that directory. If dst specifies a filename, the file will be copied to that location with the new name. shutil.copy() preserves the metadata of the original file, such as the creation and modification timestamps, permissions, etc.

shutil.copytree(src, dst, symlinks=False, ignore=None) is used to recursively copy an entire directory tree from one location to another. It takes two arguments - src, which is the source directory to be copied, and dst, which is the destination directory to copy to. The symlinks parameter is optional and is used to control how symbolic links in the source tree are treated. If it's set to True, symbolic links will be copied as links; if it's set to False (the default), symbolic links will be copied as regular files. The ignore parameter is also optional and allows for filtering of files and directories to exclude from the copy operation. shutil.copytree() creates a new directory at the destination location and copies all of the files and subdirectories of the source directory to the destination directory.

In summary, shutil.copy() is used to copy a single file, while shutil.copytree() is used to copy an entire directory tree.

Q2. What function is used to rename files??

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

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

import os

old\_name = "old\_file.txt"

new\_name = "new\_file.txt"

os.rename(old\_name, new\_name)

In this example, the os.rename() function takes two arguments: the old filename (old\_name) and the new filename (new\_name). When the function is executed, it renames the file with the old name to the new name.

Note that if the file with the new name already exists in the same directory, the os.rename() function will overwrite it. So, it's important to use caution when renaming files to avoid accidentally overwriting important data.

Q3. What is the difference between the delete functions in the send2trash and shutil modules?

Ans. Both the send2trash and shutil modules provide functions for deleting files and directories in Python, but there are some differences between them.

The send2trash module provides a cross-platform implementation for sending files and directories to the operating system's trash or recycle bin instead of permanently deleting them. This is a safer alternative to permanently deleting files, as it allows users to easily restore files that were accidentally deleted. The send2trash module only works if the operating system has a trash or recycle bin functionality.

The send2trash module has only one function, send2trash, which takes the path of the file or directory to be deleted as its argument. Here's an example:

import send2trash

filename = "example.txt"

send2trash.send2trash(filename)

On the other hand, the shutil module provides a variety of file and directory operations, including copying, moving, and deleting files and directories. The shutil module's rmtree function is commonly used for deleting directories and their contents. Unlike send2trash, shutil.rmtree permanently deletes files and directories, and cannot be used to send files to the trash or recycle bin.

Here's an example of how to use shutil.rmtree to delete a directory and its contents:

import shutil

dir\_path = "/path/to/directory"

shutil.rmtree(dir\_path)

In summary, the main difference between the delete functions in the send2trash and shutil modules is that send2trash sends files and directories to the trash or recycle bin, while shutil.rmtree permanently deletes them.

Q4. ZipFile objects have a close() method just like File objects’ close() method. What ZipFile method is equivalent to File objects’ open() method?

Ans. The ZipFile method that is equivalent to the File object's open() method is also called ZipFile.open(). This method is used to open a file within the ZIP archive and return a file-like object that can be used to read or write the file's contents.

Here's an example of how to use ZipFile.open() to read the contents of a file within a ZIP archive:

import zipfile

with zipfile.ZipFile('example.zip') as myzip:

with myzip.open('example.txt') as myfile:

contents = myfile.read()

print(contents)

In this example, the ZipFile.open() method is used to open the file 'example.txt' within the ZIP archive 'example.zip'. The with statement is used to automatically close the file-like object returned by ZipFile.open() when the block is exited.

Once the file is open, its contents are read using the read() method of the file-like object returned by ZipFile.open(). Finally, the contents of the file are printed to the console.

Note that like the File object's open() method, ZipFile.open() returns a file-like object, not a file object. This means that it does not have a close() method, and must be used within a with statement or manually closed using the close() method of the file-like object.

Q5. 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. Here's a Python program that searches a folder tree for files with a certain file extension and copies them to a new folder:

import os

import shutil

# Set the source folder and file extension

source\_folder = "/path/to/source/folder"

file\_extension = ".pdf"

# Set the destination folder

destination\_folder = "/path/to/destination/folder"

# Iterate through the source folder tree and copy files with the specified extension

for root, dirs, files in os.walk(source\_folder):

for filename in files:

if filename.endswith(file\_extension):

# Construct the full source path and destination path for the file

source\_path = os.path.join(root, filename)

destination\_path = os.path.join(destination\_folder, filename)

# Copy the file to the destination folder

shutil.copy2(source\_path, destination\_path)

In this program, the os.walk() function is used to recursively iterate through the folder tree starting from source\_folder. For each file in the tree, the program checks if its extension matches file\_extension. If it does, the program constructs the full source and destination paths for the file using the os.path.join() function and copies the file to destination\_folder using the shutil.copy2() function.

Note that the shutil.copy2() function preserves the file's metadata, such as its creation time and permissions, when copying it to the destination folder. If you don't need to preserve this metadata, you can use the shutil.copy() function instead.