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

Answer: shutil.copy() and **shutil.copytree()** are both functions in Python's **shutil** module used for copying files and directories, but they have different purposes:

1. **shutil.copy(src, dst)**:
   - This function is used to copy a single file from the source (**src**) to the destination (**dst**).
   - If **dst** is a file path, the file will be copied to that path with the same name.
   - If **dst** is a directory, the file will be copied into that directory with the same name.

   Example:
   ```
   import shutil
   shutil.copy("source_file.txt", "destination_folder")  # Copies file to destination folder
   ```

2. **shutil.copytree(src, dst)**:
   - This function is used to recursively copy an entire directory tree from the source (**src**) to the destination (**dst**).
   - If **dst** directory does not exist, it will be created.
   - If **dst** directory already exists, the contents of **src** will be copied into it.
   - It preserves the directory structure and copies all files and subdirectories within the source directory.

   Example:
   ```
   import shutil
   shutil.copytree("source_folder", "destination_folder")  # Copies entire directory tree
   ```

   In summary, **shutil.copy()** is used to copy individual files, while **shutil.copytree()** is used to copy entire directory trees, including all files and subdirectories within the source directory.

2. **What function is used to rename files?**

Answer: In Python, the **os.rename()** function is used to rename files or directories. It takes two arguments: the current name of the file or directory and the new name you want to assign to it.

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

```
import os

# Current file name
old_name = "old_file.txt"

# New file name
new_name = "new_file.txt"

# Rename the file
os.rename(old_name, new_name)
```

This code will rename the file **"old_file.txt"** to **"new_file.txt"**. Make sure to provide the correct paths for the old and new names, including any necessary directories if the file is not in the current working directory. Additionally, be cautious when renaming files to avoid overwriting existing files unintentionally.

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

Answer: The **send2trash** and **shutil** modules in Python provide different ways to delete files and directories, each with its own characteristics:

1. **send2trash module**:
   - The **send2trash** module provides a platform-independent way to send files and directories to the recycle bin or trash instead of permanently deleting them.
   - This module is useful when you want to give users a chance to recover deleted files or when you want to avoid accidental permanent deletions.
   - It does not permanently delete files but instead moves them to the recycle bin or trash, depending on the operating system.

   Example:
   ```
   import send2trash
   send2trash.send2trash("file_to_delete.txt")  # Sends file to recycle bin or trash
   ```

2. **shutil module**:
   - The **shutil** module provides functions for high-level file operations, including copying, moving, and deleting files and directories.
   - The **shutil.rmtree()** function is commonly used to recursively delete a directory and its contents, permanently removing them from the file system.
   - Unlike **send2trash**, **shutil** functions typically result in permanent deletions and do not provide a way to recover deleted files unless you have a backup.

   Example:
   ```
   import shutil
   shutil.rmtree("directory_to_delete")  # Deletes directory and its contents permanently
   ```

In summary, the main difference between the delete functions in **send2trash** and **shutil** is that **send2trash** moves files to the recycle bin or trash for potential recovery, while **shutil** functions typically result in permanent deletions from the file system.

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

Answer: The equivalent method to File objects' **open()** method for ZipFile objects is **ZipFile()** itself.

Just like how you use **open()** to open a file for reading or writing, you use **ZipFile()** to open a ZIP file for reading, writing, or appending.

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

```
import zipfile

# Open a ZIP file for reading
with zipfile.ZipFile('example.zip', 'r') as zip_file:
    # Perform operations on the ZIP file
    pass
```

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.**

Answer: You can accomplish this task using Python with the **os** module for traversing the directory tree and **shutil** module for copying files. Below is a Python script that searches for files with a specified extension in a folder tree and copies them to a new folder:

```
import os
import shutil

def search_and_copy_files(source_folder, target_folder, file_extension):
    # Create the target folder if it does not exist
    if not os.path.exists(target_folder):
        os.makedirs(target_folder)

    # Traverse the directory tree
    for root, _, files in os.walk(source_folder):
        for file in files:
            # Check if the file has the specified extension
            if file.endswith(file_extension):
                # Construct the source and target file paths
                source_file_path = os.path.join(root, file)
                target_file_path = os.path.join(target_folder, file)

                # Copy the file to the target folder
                shutil.copy(source_file_path, target_file_path)
                print(f"Copied: {source_file_path} -> {target_file_path}")

            # Example usage
            source_folder = "source_folder"a
            target_folder = "target_folder"
```

```
file_extension = ".pdf"

search_and_copy_files(source_folder, target_folder, file_extension)
```

In this script:

- The **search_and_copy_files()** function takes the source folder path, target folder path, and file extension as arguments.
- It traverses the directory tree using **os.walk()** to search for files.
- For each file found with the specified extension, it constructs the source and target file paths and copies the file to the target folder using **shutil.copy()**.
- The **print()** statement provides feedback on which files are being copied.

You can modify the **source_folder**, **target_folder**, and **file_extension** variables according to your requirements.