1. **How can you write a data stream to a file in binary mode in Python?**
2. **What is the difference between os.rename() and shutil.move() when working with files in Python?**
3. **How can you delete a file in Python if it is opened by another process?**
4. **What is the difference between the read() and readlines() functions in Python file handling?**
5. **How can you append text to an existing file in Python, preserving the previous content of the file?**
6. **How can you set file permissions when creating a file in Python?**
7. **How can you read a specific number of characters from a file in Python?**
8. **How can you copy the content of one file to another in Python, without using any built-in functions?**
9. **How can you implement a file lock in Python to prevent multiple processes from accessing the same file simultaneously?**
10. **How can you efficiently read a large file in chunks in Python, without reading the entire file into memory at once?**

1. To write a data stream to a file in binary mode in Python, you can use the wb (write binary) mode when opening the file. For example:

```
with open('binary_file.bin', 'wb') as binary_file:
    binary_file.write(b'binary data')
```

2. os.rename() is used to rename a file or directory, whereas shutil.move() is used to move a file from one location to another (which may include renaming it). The difference between the two is that shutil.move() will automatically handle the case where the source and destination are on different file systems, whereas os.rename() will raise an error in this case.
3. To delete a file in Python that is opened by another process, you can use the os module to check if the file is still locked, and retry the deletion until it succeeds. Here's an example:

```
import os
import time
filename = 'file.txt'
while True:
    try:
        os.remove(filename)
        break
    except PermissionError:
        time.sleep(0.5)
```

4. The read() function in Python file handling reads the entire content of a file as a string, while readlines() reads the entire content of a file as a list of strings, where each element of the list corresponds to a line in the file. For example:

```
with open('file.txt', 'r') as f:
    content = f.read() # reads entire content as a string
with open('file.txt', 'r') as f:
    lines = f.readlines() # reads entire content as a list of strings, each string is a line
```

5. To append text to an existing file in Python and preserve the previous content of the file, you can use the a (append) mode when opening the file. For example:

```
with open('file.txt', 'a') as f:
    f.write('new text to be appended')
```

6. To set file permissions when creating a file in Python, you can use the os module and its

```
with open('file.txt', 'w') as f:
        pass
import os
os.chmod('file.txt', 0o755) # sets file permissions to 755
```

7. To read a specific number of characters from a file in Python, you can use the read() function and specify the number of characters you want to read. For example:

```
with open('file.txt', 'r') as f:
    content = f.read(100) # reads 100 characters from the file
```

8. To copy the content of one file to another in Python without using any built-in functions, you can use a loop to read the content of the source file in chunks and write it to the destination file. For example:

```
with open('source.txt', 'rb') as src, open('destination.txt', 'wb') as dst:
    chunk_size = 4096
    while True:
        chunk = src.read(chunk_size)
        if not chunk:
            break
        dst.write(chunk)
```

9. To implement a file lock in Python to prevent multiple processes from accessing the same file simultaneously, you can use the fcntl module to set a lock on the file before accessing it. Here's an example:

```
import fcntl
with open('file.txt', 'r') as f:
        fcntl.flock(f, fcntl.LOCK_EX)
        # Access the file
        fcntl.flock(f, fcntl.LOCK_UN)
```

10. To efficiently read a large file in chunks in Python without reading the entire file into memory at once, you can use a loop to read the file in small chunks, and process each chunk individually. For example:

```
with open('large_file.txt', 'r') as f:
        chunk_size = 1024
        while True:
                chunk = f.read(chunk_size)
                if not chunk:
                        break
        # Process the chunk
```