In Python, `seek()` and `tell()` are methods used for file manipulation. They are primarily used when reading from or writing to files. Let me explain each of them:

`seek(offset, whence)`

The `seek()` method is used to change the current position of the file pointer within a file. The file pointer indicates the current position from where the next read or write operation will occur.

- **`offset`: It specifies the number of bytes to move the file pointer.**
- **`whence`: It specifies the reference position for the offset. It can take one of three values:**
  - **`0` (default): The offset is relative to the beginning of the file.**
  - **`1`: The offset is relative to the current position of the file pointer.**
  - **`2`: The offset is relative to the end of the file.**

Example:

python Copy code

```python
# Open a file in read mode
file = open("example.txt", "r")

# Move the file pointer to the 10th byte from the beginning
file.seek(10)

# Read the content from the current position
content = file.read()
print(content)

# Close the file
file.close()
```

`tell()`

The `tell()` method returns the current position of the file pointer within the file. It tells you where the file pointer is currently located.

Example:

```python
# Open a file in read mode
file = open("example.txt", "r")

# Read the content from the file
content = file.read()
print("Content:", content)

# Get the current position of the file pointer
position = file.tell()
print("Current position:", position)

# Close the file
file.close()
```

These methods are particularly useful when you need to navigate within a file or keep track of the current position while reading or writing data.

Code:

```
f = open("sample.txt")
f.seek(11)
print(f.tell())  # tell the file file pointer
print(f.readline())
# print(f.tell())
f.close()
```

Output:

```
C:\Users\test\PycharmProjects\project_1\.venv\Scripts\python.exe C:\Users\test\PycharmProjects\project_1\1.py
11
d!

This is a sample file.
```

Using With Block to open python files:

Using the `with` statement in Python is a best practice when working with files because it automatically handles closing the file when you are done with it, even if an error occurs. It's more concise and ensures that resources are properly released.

python Copy code

```python
# Writing to a file using with statement
with open("example.txt", "w") as file:
    file.write("Hello, world!\n")
    file.write("This is a test file.\n")

# Reading from a file using with statement
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

- **The `open()` function opens the file in write mode (`"w"`) for the first `with` statement and read mode (`"r"`) for the second `with` statement.**
- **Inside the `with` block, you perform the file operations (writing or reading).**
- **Once the code block inside the `with` statement is executed, the file is automatically closed, regardless of whether an error occurred or not. This is the beauty of using the `with` statement.**

Using the `with` statement ensures that the file is closed properly, even if an exception is raised during the execution of the block. It's a recommended practice for file handling in Python.

Code:

```
with open("sample.txt") as f:
    a = f.readlines()
    print(a)
```

Output:

```
C:\Users\test\PycharmProjects\project_1\.venv\Scripts\python.exe C:\Users\test\PycharmProjects\project_1\1.py
['Hello, world!\n', 'This is a sample file.\n', 'Python file I/O basics.\n', 'Appending new content to the file.']
```