# Module 7 Cheat Sheet

## File Modes and Operations

- File Modes:
  - `'r'` - Read mode, opens the file for reading.
  - `'w'` - Write mode, opens the file for writing. Creates a new file or truncates the existing file.
  - `'a'` - Append mode, opens the file for writing and appends to the end if it exists.
  - `'x'` - Exclusive creation mode, creates a new file and fails if the file already exists.
  - `'rb'` - Read mode in binary format.
  - `'wb'` - Write mode in binary format.
  - `'r+'` - Read and write mode.
- File Methods:
  - `file.read()` - Reads the entire file.
  - `file.readline()` - Reads a single line from the file.
  - `file.readlines()` - Reads all lines into a list.
  - `file.write(data)` - Writes a single string to the file.
  - `file.writelines(list)` - Writes a list of strings to the file.
  - `file.tell()` - Returns the current file position.
  - `file.seek(offset, from_what)` - Moves the file cursor to a specific position.

## Handling Files with `with` Statement

- `with` Statement:
  - Ensures proper acquisition and release of resources.
  - Automatically closes the file after the block is executed.

```
with open('file.txt', 'r') as file:
   data = file.read()
```

## Checking File Existence

- Using `os` module:
  - `os.path.isfile('file.txt')` - Checks if a file exists and is a file.
  - `os.path.exists('file.txt')` - Checks if a path exists.

```
wimport os
if os.path.isfile('file.txt'):
   print("File exists")
```

Reading and Writing Binary Files

- Binary Mode:
  - Open file in binary mode for reading: `open('file.txt', 'rb')`
  - Open file in binary mode for writing: `open('file.txt', 'wb')`

Appending to a File

- Append Mode:
  - Opens the file in append mode: `open('file.txt', 'a')`

File Paths

- Paths:
  - Absolute Path: Starts from the root directory.
  - Relative Path: Relative to the current working directory.

File Reading and Writing Techniques

- Reading Specific Characters:
  - `file.read(10)` - Reads the first 10 characters of a file.
- Handling Missing Files:
  - `try-except` Block:
    - Use `try-except` to handle file-related exceptions.

```
try:
   with open('file.txt', 'r') as file:
       data = file.read()
except FileNotFoundError:
   print("File not found")
```

Context Managers and Efficiency

- Context Managers:
  - Ensure that resources are properly managed and closed.

```
with open('file.txt', 'w') as file:
   file.write('Hello World')
```

### Working with File Cursors

- File Cursor:
  - Move cursor to the beginning: `file.seek(0)`

### Example Operations

- Copying File Content:
  - Read from one file and write to another.

```
with open('file1.txt', 'r') as src, open('file2.txt', 'w') as
dst:
   dst.write(src.read())
```

## Key Concepts:

- File Modes:
  - Determine how files are opened and operated on.
- File Methods:
  - Provide various ways to read, write, and manipulate file data.
- Context Managers:
  - Ensure that files are properly closed after operations.
- File Paths:
  - Understanding absolute and relative paths is crucial for file operations.
- Exception Handling:
  - Use `try-except` blocks to handle file-related errors gracefully.