Python File Handling Cheat Sheet

Opening a File

Python provides the `open()` function to work with files.

Syntax:

```
file = open("filename", "mode")
```

File Modes:

- `"r"` → Read (default, errors if file doesn't exist)
- `"w"` → Write (overwrites if file exists)
- `"a"` → Append (adds content at the end)
- `"x"` → Create (errors if file exists)
- `"b"` → Binary mode (use with `r`, `w`, `a`)
- `"t"` → Text mode (default)

Reading a File

Read Entire File

```
with open("example.txt", "r") as file:
    content = file.read()
print(content)
```

Read Line by Line

```
with open("example.txt", "r") as file:
    for line in file:
        print(line.strip())
```

Read Specific Bytes

```
with open("example.txt", "r") as file:
    chunk = file.read(10)  # Reads first 10 characters
```

Read Into a List

```
with open("example.txt", "r") as file:
    lines = file.readlines()  # Returns a list of all lines
```

Writing to a File

Overwrite File

```
with open("example.txt", "w") as file:
    file.write("Hello, World!\n")
```

Warning: Using `"w"` deletes existing content!

Append to File

```
with open("example.txt", "a") as file:
    file.write("Appending this line!\n")
```

Binary Files (Images, Videos, etc.)

Reading Binary

```
with open("image.jpg", "rb") as file:
    data = file.read()
```

Writing Binary

```
with open("copy.jpg", "wb") as file:
    file.write(data)
```

Best Practice: `with open(...)`

✅ Automatically closes file, prevents memory leaks.

```
with open("example.txt", "r") as file:
    content = file.read()
```

❌ **Avoid manual close:**

```
file = open("example.txt", "r")
content = file.read()
file.close()  # Must manually close
```

Checking if a File Exists

```
import os
if os.path.exists("example.txt"):
    print("File exists!")
```

Deleting a File

```
import os
os.remove("example.txt")
```

Creating and Deleting Folders

Creating a Folder

```
import os
os.mkdir("new_folder")
```

Deleting an Empty Folder

```
import os
os.rmdir("new_folder")
```

Deleting Folder with Contents

```
import shutil
shutil.rmtree("new_folder")
```

`os.path.join` and File Paths

Why File Paths Matter

File operations need **absolute paths** in many cases, especially when working in different directories or OS environments. Using `os.path.join` ensures cross-platform compatibility.

Using `os.path.join`

```
import os
folder = "documents"
filename = "file.txt"
filepath = os.path.join(folder, filename)
print(filepath)  # Output: "documents/file.txt" (Linux/macOS) or "documents\\file.txt" (Windows)
```

Using `argparse` for Command-Line Arguments

When working with files, scripts often take file names as arguments.

Basic `argparse` Example

```
import argparse
parser = argparse.ArgumentParser()
parser.add_argument("filename", type=str, help="Path to the file")
args = parser.parse_args()
filename = args.filename

with open(filename, "r") as file:
    content = file.read()
    print(content)
```

Summary

- Use `with open()` for safe file handling.
- Always check file existence before reading/writing.
- Use `os.path.join` for platform-independent paths.
- Use `argparse` to handle command-line arguments dynamically.

🚀 **Now you're ready to handle files like a pro!**