# File Handling In Python

By Bhimashankar Takalki

In Python, there are various built-in functions and methods for working with files. These functions and methods allow you to perform different operations on files, such as opening, reading, writing, and closing files. Here are some of the commonly used file functions and methods in Python:

## 1. `open()`

The `open()` function is used to open a file and returns a file object that can be used to perform file operations.

python Copy code

```python
file = open("example.txt", "r")
```

## 2. `close()`

The `close()` method is used to close a file that was opened using the `open()` function. It's important to close files after performing file operations to release system resources.

python Copy code

```
file.close()
```

## 3. `read()`

The `read()` method is used to read the entire contents of a file.

python Copy code

```
content = file.read()
```

## 4. `readline()`

The `readline()` method is used to read a single line from a file.

python Copy code

```
line = file.readline()
```

## 5. `readlines()`

The `readlines()` method is used to read all lines from a file and return them as a list.

python Copy code

```
lines = file.readlines()
```

## 6. `write()`

The `write()` method is used to write data to a file.

python Copy code

```
file.write("Hello, world!")
```

## 7. `writelines()`

The `writelines()` method is used to write a list of lines to a file.

python Copy code

```
lines = ["Line 1\n", "Line 2\n", "Line 3\n"]
file.writelines(lines)
```

## 8. `seek()`

The `seek()` method is used to change the current position of the file pointer.

python Copy code

```
file.seek(0)
```

## 9. `tell()`

The `tell()` method is used to return the current position of the file pointer.

python Copy code

```
position = file.tell()
```

## 10. `flush()`

The `flush()` method is used to flush the internal buffer of the file.

```python
file.flush()
```

## 11. `truncate()`

The `truncate()` method is used to truncate the file to a specified size.

```python
file.truncate(10)
```

File handling in Python is a fundamental aspect of working with files on the operating system. Python provides built-in functions and methods for file input and output operations, making it easy to work with files.

## Opening and Closing Files:

You can open files using the `open()` function, which returns a file object. After performing file operations, it's important to close the file using the `close()` method to release the resources associated with it.

python Copy code

```python
# Open a file for reading ('r' mode)
file = open("example.txt", "r")

# Perform file operations
content = file.read()
print(content)

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

## Writing to Files:

To write to a file, open it in `'w'` mode. If the file doesn't exist, it will be created. If it does exist, its contents will be overwritten.

python Copy code

```python
# Open a file for writing ('w' mode)
file = open("example.txt", "w")

# Write content to the file
file.write("Hello, world!\n")
file.write("This is a new line.\n")

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

## Appending to Files:

To append content to an existing file, open it in `'a'` mode. If the file doesn't exist, it will be created.

python Copy code

```python
# Open a file for appending ('a' mode)
file = open("example.txt", "a")

# Append content to the file
file.write("Appending a new line.\n")

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

## Reading from Files:

To read from a file, open it in `'r'` mode. You can read the entire content or read line by line using methods like `read()`, `readline()`, or `readlines()`.

python Copy code

```python
# Open a file for reading ('r' mode)
file = open("example.txt", "r")

# Read the entire content
content = file.read()
print(content)

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

## Using `with` Statement (Context Manager):

Python's `with` statement provides a more elegant way to work with files. It automatically closes the file when the block inside the `with` statement is exited.

python Copy code

```python
# Using 'with' statement to open and read a file
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

## File Modes:

- `'r'`: Open a file for reading (default).
- `'w'`: Open a file for writing. If the file exists, its contents are overwritten. If the file does not exist, it is created.
- `'a'`: Open a file for appending. The new data is written to the end of the file.
- `'b'`: Open a file in binary mode.
- `'t'`: Open a file in text mode (default).
- `'+'`: Open a file for both reading and writing.

# Example:

python Copy code

```python
# Using 'with' statement to open, write, and read a file
with open("example.txt", "w") as file:
    file.write("Line 1\n")
    file.write("Line 2\n")

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

Handling CSV files in Python involves reading from and writing to files in the Comma-Separated Values (CSV) format, which is commonly used for tabular data storage. Python provides the `csv` module for working with CSV files efficiently.

## Reading from a CSV File:

You can use the `csv.reader` class to read data from a CSV file. The `reader` object can be iterated over to access each row of the CSV file as a list.

python Copy code

```python
import csv

# Open the CSV file for reading
with open('data.csv', mode='r') as file:
    # Create a CSV reader object
    reader = csv.reader(file)

    # Iterate over each row in the CSV file
    for row in reader:
        print(row)
```

**Writing to a CSV File:**

To write data to a CSV file, you can use the csv.writer class. You need to provide an iterable (such as a list or tuple) containing the data for each row.

```
import csv

# Data to be written to the CSV file
data = [
    ['Name', 'Age', 'City'],
    ['John', 30, 'New York'],
    ['Alice', 25, 'Los Angeles'],
    ['Bob', 35, 'Chicago']
]

# Open the CSV file for writing
with open('output.csv', mode='w', newline='') as file:
    # Create a CSV writer object
    writer = csv.writer(file)

    # Write data to the CSV file
    writer.writerows(data)
```

## Using DictReader and DictWriter:

The csv.DictReader and csv.DictWriter classes can be used to read and write CSV files with dictionaries, respectively. This allows you to access data using column names instead of indices.

```
import csv

# Open the CSV file for reading
with open('data.csv', mode='r') as file:
    # Create a DictReader object
    reader = csv.DictReader(file)

    # Iterate over each row in the CSV file
    for row in reader:
        print(row['Name'], row['Age'], row['City'])

# Data to be written to the CSV file
data = [
    {'Name': 'John', 'Age': 30, 'City': 'New York'},
    {'Name': 'Alice', 'Age': 25, 'City': 'Los Angeles'},
    {'Name': 'Bob', 'Age': 35, 'City': 'Chicago'}
]
```

```
# Open the CSV file for writing
with open('output.csv', mode='w', newline='') as file:
    # Define column names for the CSV file
    fieldnames = ['Name', 'Age', 'City']

    # Create a DictWriter object
    writer = csv.DictWriter(file, fieldnames=fieldnames)

    # Write column headers to the CSV file
    writer.writeheader()

    # Write data to the CSV file
    writer.writerows(data)
```

**Output:**

```
ID of process running main program: 1141
Main thread name: MainThread
Task 1 assigned to thread: t1
ID of process running task 1: 1141
Task 2 assigned to thread: t2
ID of process running task 2: 1141
```

Working with JSON (JavaScript Object Notation) files in Python involves reading and writing data in JSON format. Python provides the `json` module to handle JSON data efficiently. Here's a best example code demonstrating JSON file handling in Python:

## Reading JSON from a File:

python Copy code

```python
import json

# Open the JSON file for reading
with open('data.json', 'r') as file:
    # Load JSON data from the file
    data = json.load(file)

# Display the loaded JSON data
print(data)
```

In this code:

- We open the JSON file `data.json` in read mode using the `open()` function.
- We use the `json.load()` function to load JSON data from the file into a Python dictionary named `data`.
- We then print the loaded JSON data.

## Writing JSON to a File:

python Copy code

```python
import json

# JSON data to be written to the file
data = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

# Open the JSON file for writing
with open('output.json', 'w') as file:
    # Write JSON data to the file
    json.dump(data, file, indent=4)
```

In this code:

- We define a Python dictionary named `data` containing some sample data.
- We open the JSON file `output.json` in write mode using the `open()` function.
- We use the `json.dump()` function to write the JSON formatted string representation of the `data` dictionary to the file.
- We specify the `indent=4` argument to format the JSON data with an indentation of 4 spaces for better readability.

**Example: Reading and Writing JSON Data**

```
import json

# JSON data to be written to the file
data = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

# Writing JSON data to a file
with open('data.json', 'w') as file:
    json.dump(data, file, indent=4)
```

```
# Reading JSON data from the file
with open('data.json', 'r') as file:
    loaded_data = json.load(file)

# Display the loaded JSON data
print(loaded_data)
```

In this example:

- We first write JSON data to a file using `` `json.dump()` ``.
- Then, we read JSON data from the file using `` `json.load()` ``.
- Finally, we display the loaded JSON data.

This example demonstrates how to read from and write to JSON files in Python using the `` `json` `` module. JSON files are commonly used for storing and exchanging data between different applications due to their simplicity and ease of use. Python's `` `json` `` module provides convenient functions for working with JSON data, making it easy to handle JSON file operations efficiently.