Minor in AI

File Handling in Python

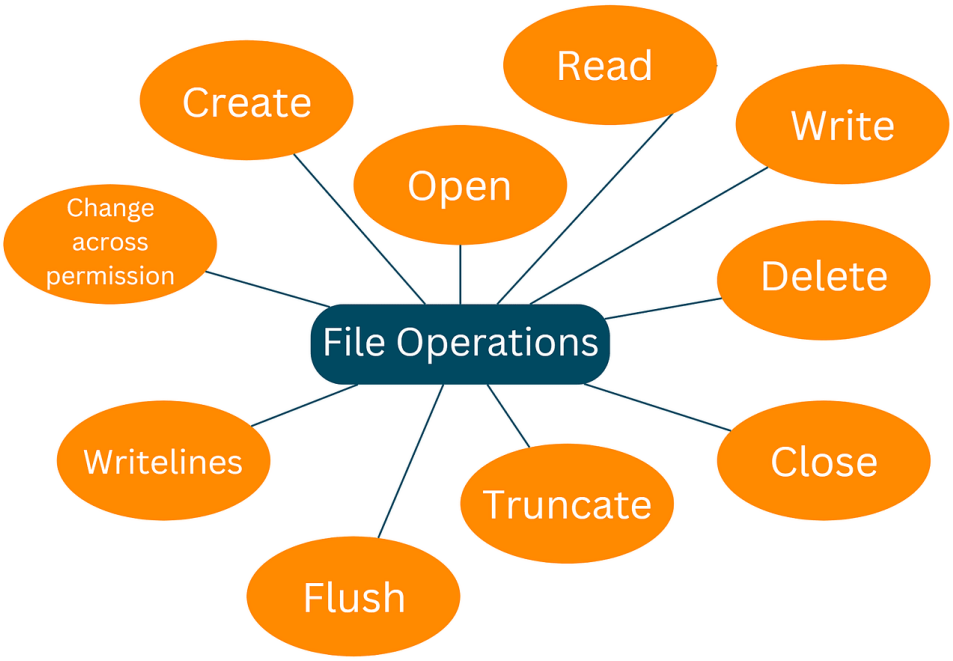

Figure 1: File Handling

1 Smart Book-Keeping with Python File Handling

Imagine you're a teacher managing records for hundreds of students. You need to store their names, grades, and attendance. How would you handle this data efficiently? This is where file handling in Python comes to the rescue.

File handling allows us to store and retrieve large amounts of data persistently. Unlike variables that lose their values when a program ends, files retain information even after the program closes. This makes files crucial for managing data in real-world applications.

2 Understanding File Handling

File handling in Python involves working with external files for storing and retrieving data. It's a fundamental concept in programming that allows us to work with persistent data.

2.1 Key Concepts

- **Opening a file**: Before we can work with a file, we need to open it using the `open()` function.
- **File modes**: These determine how we interact with the file (read, write, append).
- **Reading from a file**: Extracting data from the file.
- **Writing to a file**: Adding/Appending new data to the file.
- **Closing a file**: It's crucial to close files after operations to free up system resources.

2.2 File Modes

- 'r': Read mode
- 'w': Write mode (overwrites existing content)
- 'a': Append mode (adds to existing content)
- 'r+': Read and write mode without overwriting
- 'w+': Read and write mode with overwriting
- 'a+': Append and read mode

3 Code Implementation

Let's implement a simple student record system to demonstrate file handling:

```
# Open file in write mode and initialize with a header
with open('student_records.txt', 'w') as file:
    file.write("Student Records\n")  # Writing the title of the records
    file.write("------------------\n")  # Adding a separator line

# Function to add a student record to the file
def add_student(name, grade):
    with open('student_records.txt', 'a') as file:  # Open file in
        append mode
        file.write(f"{name}: {grade}\n")  # Write the student's name and
            grade

# Function to read and display all student records from the file
def read_records():
    with open('student_records.txt', 'r') as file:  # Open file in read
        mode
        print(file.read())  # Read and print the contents of the file

# Adding student records
add_student("Alice", "A")    # Adding Alice with grade A
add_student("Bob", "B")      # Adding Bob with grade B
add_student("Charlie", "A-") # Adding Charlie with grade A-

# Reading and displaying the records
read_records()  # Display all student records
```

This code creates a file, adds student records, and then reads and displays them.

4 Diving Deeper

4.1 Exception Handling

4.1.1 Basic Example: Handling File Not Found

```
# Attempting to open a file that may not exist
try:
    with open('nonexistent_file.txt', 'r') as file:
        content = file.read()  # Trying to read the file's content
except FileNotFoundError:
    # Handles the case where the file does not exist
    print("The file does not exist.")
```

4.1.2 Handling Multiple Exceptions

```
try:
    num = int(input("Enter a number: "))  # Taking user input and
        converting to integer
    result = 10 / num  # Performing division
except ValueError:
    # Handles the case where input is not a valid integer
    print("Invalid input! Please enter a number.")
except ZeroDivisionError:
    # Handles division by zero error
    print("Cannot divide by zero!")
```

4.1.3 Using `finally` for Cleanup

```
try:
    file = open("example.txt", "r")  # Attempting to open a file
    content = file.read()  # Reading the file content
except FileNotFoundError:
    # Handles the case where the file does not exist
    print("File not found.")
finally:
    # This block always executes, regardless of exceptions
    print("Execution completed.")
```

4.2 Working with CSV Files

4.2.1 What is CSV?

CSV (Comma-Separated Values) files are commonly used for storing tabular data. Each line in a CSV file represents a row, and the columns are separated by commas. CSV files are widely used in data science and machine learning for storing datasets, as they provide an easy way to represent data in a simple text format.

4.2.2 Where is CSV?

In Machine Learning, CSV files are commonly used to store structured datasets, including both input data and target data. After reading the data from a CSV file, it is often preprocessed and used for training machine learning models.

4.2.3 Accessing CSV:

Using libraries like `csv` in Python, we can easily read and write these files for data preprocessing and model training.

```
import csv  # Importing the CSV module

# Writing to a CSV file
with open('students.csv', 'w', newline='') as file:
    writer = csv.writer(file)  # Create a writer object to write to the CSV file
    writer.writerow(["Name", "Grade"])  # Writing the header row
    writer.writerow(["Alice", "A"])  # Writing data for Alice
    writer.writerow(["Bob", "B"])  # Writing data for Bob

# Reading from a CSV file
with open('students.csv', 'r') as file:
    reader = csv.reader(file)  # Create a reader object to read the CSV file
    for row in reader:  # Iterating over each row in the CSV file
        print(row)  # Print each row of data
```

Explanation:

- `csv.writer` is used to write data to a CSV file.
- `csv.reader` is used to read data from a CSV file.
- The `writerow()` method writes a single row of data to the CSV file.
- The `for row in reader` loop reads each row from the file, which is then printed.

5 Conclusion

File handling in Python provides a powerful way to work with persistent data. It's essential for many real-world applications, from simple record-keeping to complex data analysis followed by ML and AI. By understanding file handling, one can create more robust and useful programs.

6 Additional Resources

- Google Colab File