Data Handling in Python

May 15, 2026

Contents

1 Introduction to Data Handling

1.1 What is Data Handling?

Data handling means collecting, organizing, storing, and managing data so that it can be used effectively. In programming, it involves reading, writing, and processing data using computer programs.

1.2 Importance of Data Handling in Programming

Data handling is vital because it helps programs interact with real-world information. Whether storing student records, reading sensor inputs, or saving game progress, efficient data handling makes software useful and dynamic.

1.3 Real-Life Applications of Data Handling

- Managing attendance in schools.
- Storing patient records in hospitals.
- Processing bank transactions.
- Handling user data in apps and websites.

These examples show how data handling connects programming with everyday tasks.

2 Types of Data

2.1 Structured Data

Structured data is organized in a fixed format like rows and columns in tables or spreadsheets. It is easy to store and search.

2.2 Unstructured Data

Unstructured data has no fixed format, like images, videos, or text files. It requires special processing to extract meaningful information.

2.3 Numeric and Textual Data

Data can be:

- **Numeric:** Numbers used for calculations (e.g., 5, 3.14).
- **Textual:** Words and sentences (e.g., ”Hello”, ”Python”).

Programming languages like Python provide special ways to handle each type.

3 File Handling in Python

3.1 What are Files?

A **file** is a storage unit on a computer that stores data permanently. Unlike variables, data in files stays even after the program ends.

3.2 Importance of File Handling

File handling allows programs to save data, read saved data, and share information between programs or with users.

3.3 Types of Files

Text Files	Store data in readable characters (letters, numbers). Example: `.txt` files.
Binary Files	Store data in binary form (0s and 1s). Used for images, audio, or compiled programs.

4 Opening and Closing Files

4.1 open() Function

The `open()` function is used to open a file and returns a file object to work with.

Syntax:

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

4.2 close() Method

Once done, always close the file using `close()` to free system resources.

4.3 File Opening Modes

Mode	Purpose
`r`	Open file for reading. File must exist.
`w`	Open file for writing. Creates file if not exists, overwrites if exists.
`a`	Open file for appending. Adds data at end if file exists, else creates.
`rb`	Open file for reading in binary mode.
`wb`	Open file for writing in binary mode.

5 Reading Data from Files

5.1 read()

Reads the entire file content as a string.

5.2 readline()

Reads one line from the file each time it is called.

5.3 readlines()

Reads all lines and returns a list of strings.

Example: Reading a File

```
file = open('sample.txt', 'r')
content = file.read()
print(content)
file.close()
```

This program opens `sample.txt`, reads all text, prints it, and closes the file.

6 Writing Data to Files

6.1 write()

Writes a string to the file.

6.2 writelines()

Writes a list of strings to the file.

6.3 Appending Data

Using `a` mode, new data is added at the end without deleting existing content.

Example: Writing to a File

```
file = open('output.txt', 'w')
file.write('Hello, Python!\n')
file.close()
```

This writes the string to `output.txt`. If the file exists, it replaces the old content.

7 File Pointer Functions

7.1 tell()

Returns the current position of the file pointer (in bytes).

7.2 seek()

Moves the file pointer to a specified position.

Example: Using tell() and seek()

```
file = open('sample.txt', 'r')
print(file.read(5))     # Read 5 characters
print(file.tell())      # Show pointer position
file.seek(0)            # Move pointer to start
print(file.readline())  # Read first line
file.close()
```

8 Working with CSV Files

8.1 Introduction to CSV

CSV (Comma Separated Values) files store tabular data, where each line is a record and fields are separated by commas.

8.2 Reading CSV Files

Python has a `csv` module to read and write CSV files easily.

8.3 Writing CSV Files

You can write lists or dictionaries to CSV files using the `csv` module.

Basic CSV Usage

```
import csv

```

```
# Reading
with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)

# Writing
with open('data.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Name', 'Age'])
    writer.writerow(['Alice', 17])
```

Explanation: The above code reads rows from `data.csv` and writes new rows.

9 Exception Handling in File Operations

9.1 try, except, finally

Use these blocks to handle errors and ensure files close properly.

Example: Handling File Errors

```
try:
    file = open('missing.txt', 'r')
    content = file.read()
except FileNotFoundError:
    print('File not found!')
finally:
    try:
        file.close()
    except:
        pass
```

This avoids program crash if the file is missing.

10 Data Handling Using Lists

10.1 Storing Data in Lists

Lists can hold multiple data items ordered by index.

10.2 Processing List Data

You can add, remove, and modify list elements.

10.3 Searching and Counting Data

Python lists provide methods to find and count specific items.

Example: Searching and Counting

```
students = ['Alice', 'Bob', 'Alice', 'David']
count_alice = students.count('Alice')
print('Alice appears:', count_alice, 'times')
```

11 Basic Data Analysis Concepts

11.1 Counting Values

Count how many times values appear (like above).

11.2 Searching Records

Find if certain data exists in your dataset.

11.3 Filtering Data

Select only data meeting certain criteria.

11.4 Simple Calculations

Calculate sums, averages on numeric data.

12 Practical Python Programs

Below are 12 practical programs for common data handling tasks.

12.1 1. Read a Text File

```
file = open('sample.txt', 'r')
print(file.read())
file.close()
```

Reads and prints entire file content.

12.2 2. Write Data into File

```
file = open('output.txt', 'w')
file.write('Hello World\n')
file.close()
```

Writes ”Hello World” to a file.

12.3 3. Count Words in File

```
file = open('sample.txt', 'r')
text = file.read()
file.close()
words = text.split()
print('Word count:', len(words))
```

Counts words by splitting text on spaces.

12.4 4. Append Data to File

```
file = open('output.txt', 'a')
file.write('Appending this line.\n')
file.close()
```

Adds new line at end without deleting old data.

12.5 5. Read CSV File

```
import csv
with open('data.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Prints each row from CSV.

12.6 6. Search Word in File

```
word_to_find = 'Python'
with open('sample.txt', 'r') as file:
    content = file.read()
if word_to_find in content:
    print(f"'{word_to_find}' found!")
else:
    print(f"'{word_to_find}' not found.")
```

Checks if a word exists in the file.

12.7 7. Count Lines in File

```
with open('sample.txt', 'r') as file:
    lines = file.readlines()
print('Total lines:', len(lines))
```

Counts number of lines in file.

12.8 8. Write Multiple Lines to File

```
lines = ['Line 1\n', 'Line 2\n', 'Line 3\n']
with open('output.txt', 'w') as file:
    file.writelines(lines)
```

Writes a list of lines to file.

12.9 9. Use tell() and seek()

```
file = open('sample.txt', 'r')
print(file.read(10))
print(file.tell())
file.seek(0)
```

```
print(file.readline())
file.close()
```

Shows file pointer position manipulation.

12.10 10. Read Binary File

```
with open('image.jpg', 'rb') as file:
    data = file.read(20)
print(data)
```

Reads first 20 bytes of binary file.

12.11 11. Write Binary File

```
data = b'\x00\x01\x02\x03'
with open('output.bin', 'wb') as file:
    file.write(data)
```

Writes binary data to file.

12.12 12. Exception Handling in File Reading

```
try:
    with open('missing.txt', 'r') as file:
        print(file.read())
except FileNotFoundError:
    print('File missing!')
```

Handles missing file error gracefully.

13 Common Errors in Data Handling

Error	Description
File Not Found Error	Happens when the file does not exist but program tries to open it in `r` mode.
Permission Error	Occurs if your program tries to write to a file without permission.
Syntax Mistakes	Errors in Python code like missing colons or wrong indentation.
Wrong File Mode Usage	Using `write` mode on a file that should be read or vice versa can cause loss of data.

14 Important Viva Questions

- **What is file handling?**
  Answer: File handling is reading and writing data to files using a programming language.
- **Difference between text and binary file?**
  Answer: Text files store data as readable characters; binary files store data in binary form.
- **What is a CSV file?**
  Answer: A CSV file stores tabular data with values separated by commas.

15 Exam-Oriented Questions

15.1 Short Answer Questions

- What does the `open()` function do in Python?
- Name two file opening modes and their uses.
- How do you read one line at a time from a file?

15.2 Long Answer Questions

- Explain the difference between `read()`, `readline()`, and `readlines()` methods.
- Describe the steps to write data into a file and append new data.
- How does exception handling improve file operations? Give an example.

15.3 Multiple Choice Questions (MCQs)

1. Which mode opens a file for appending data?
   (a) `r` (b) `w` (c) `a` (d) `rb`
   Answer: (c)

2. What does `csv.reader()` return?
   (a) A string (b) A list of lists (c) A file object (d) None
   Answer: (b)

3. Which method closes a file?
   (a) `close()` (b) `end()` (c) `stop()` (d) `exit()`
   Answer: (a)

16 Revision Notes

- Data handling means managing data with programs.
- Files help store data permanently.
- `open()` opens files; always `close()` after use.
- Modes: `r` (read), `w` (write), `a` (append).
- Use `read()`, `readline()`, `readlines()` to read files.
- Use `write()` and `writelines()` to write files.
- `tell()` and `seek()` control file pointer.
- CSV files store table data; use `csv` module to work with them.
- Use `try-except-finally` to handle file errors safely.
- Lists store data for processing and analysis.
- Common errors: File not found, permission, syntax mistakes.

Keep practicing file operations and Python programs regularly to master data handling!