CH-9: Files and Exception Handling in Python

December 23, 2025

Learning Objectives

- Understand file handling in Python

Learning Objectives

- Understand file handling in Python
- Perform text file processing and updates

Learning Objectives

- Understand file handling in Python
- Perform text file processing and updates
- Learn exception handling mechanisms

Learning Objectives

- Understand file handling in Python
- Perform text file processing and updates
- Learn exception handling mechanisms
- Apply finally clause and custom exceptions

Introduction to Files

- Variables, lists, tuples, dictionaries, sets, arrays, Series, DataFrames $\longrightarrow$ data lost when program ends.

Introduction to Files

- Variables, lists, tuples, dictionaries, sets, arrays, Series, DataFrames $\longrightarrow$ data lost when program ends.
- A **file** is a collection of data stored permanently

Introduction to Files

- Variables, lists, tuples, dictionaries, sets, arrays, Series, DataFrames $\longrightarrow$ data lost when program ends.
- A **file** is a collection of data stored permanently
- Used for storing large data and program output

Introduction to Files

- Variables, lists, tuples, dictionaries, sets, arrays, Series, DataFrames $\longrightarrow$ data lost when program ends.
- A **file** is a collection of data stored permanently
- Used for storing large data and program output
- Python treats files as **objects**

Types of Files

- **Text files** (`.txt`, `.csv`)

Types of Files

- **Text files** (`.txt`, `.csv`)
- **Binary files** (`.dat`, `.bin`)

Types of Files

- **Text files** (`.txt`, `.csv`)
- **Binary files** (`.dat`, `.bin`)

(We'll focus mainly on text files as per syllabus)

Standard File Objects in Python

- When a Python program starts, it creates three standard file objects

Standard File Objects in Python

- When a Python program starts, it creates three standard file objects
- These are created automatically by the `sys` module

Standard File Objects in Python

- When a Python program starts, it creates three standard file objects
- These are created automatically by the `sys` module

- `sys.stdin` — standard input

Standard File Objects in Python

- When a Python program starts, it creates three standard file objects
- These are created automatically by the `sys` module

- `sys.stdin` — standard input
- `sys.stdout` — standard output

Standard File Objects in Python

- When a Python program starts, it creates three standard file objects
- These are created automatically by the `sys` module

- `sys.stdin` — standard input
- `sys.stdout` — standard output
- `sys.stderr` — standard error

Standard File Objects in Python

- When a Python program starts, it creates three standard file objects
- These are created automatically by the `sys` module

- `sys.stdin` — standard input
- `sys.stdout` — standard output
- `sys.stderr` — standard error

- They behave like file objects

Standard File Objects in Python

- When a Python program starts, it creates three standard file objects
- These are created automatically by the `sys` module

- `sys.stdin` — standard input
- `sys.stdout` — standard output
- `sys.stderr` — standard error

- They behave like file objects
- By default, they are connected to the command line

input() and print(): Internal Working

- `input()` reads data from `sys.stdin`

input() and print(): Internal Working

- `input()` reads data from `sys.stdin`
- `print()` sends output to `sys.stdout`

input() and print(): Internal Working

- `input()` reads data from `sys.stdin`
- `print()` sends output to `sys.stdout`
- Errors and tracebacks go to `sys.stderr`

input() and print(): Internal Working

- `input()` reads data from `sys.stdin`
- `print()` sends output to `sys.stdout`
- Errors and tracebacks go to `sys.stderr`
- Using `sys` explicitly is rare

input() and print(): Internal Working

- `input()` reads data from `sys.stdin`
- `print()` sends output to `sys.stdout`
- Errors and tracebacks go to `sys.stderr`
- Using `sys` explicitly is rare
- `input()` and `print()` handle this implicitly

File Representation and EOF

- Python treats:

File Representation and EOF

- Python treats:
  - **Text files** as a sequence of **characters**

File Representation and EOF

- Python treats:
  - **Text files** as a sequence of **characters**
  - **Binary files** as a sequence of **bytes**

File Representation and EOF

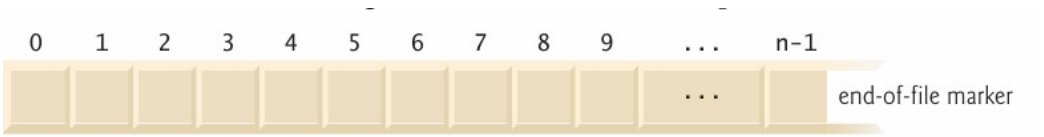
- Python treats:
  - **Text files** as a sequence of **characters**
  - **Binary files** as a sequence of **bytes**
- File positions start at **0** (zero-based indexing)

File Representation and EOF

- Python treats:
  - **Text files** as a sequence of **characters**
  - **Binary files** as a sequence of **bytes**
- File positions start at **0** (zero-based indexing)
- For a file with n characters/bytes, the last position is $n - 1$

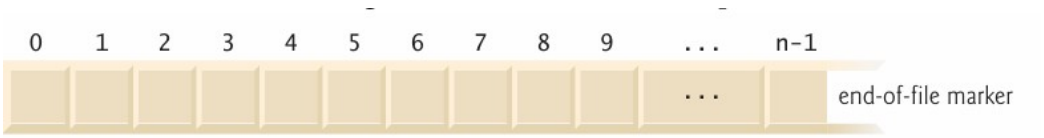
File Representation and EOF

- Python treats:
  - **Text files** as a sequence of **characters**
  - **Binary files** as a sequence of **bytes**
- File positions start at **0** (zero-based indexing)
- For a file with n characters/bytes, the last position is $n - 1$



File Representation and EOF

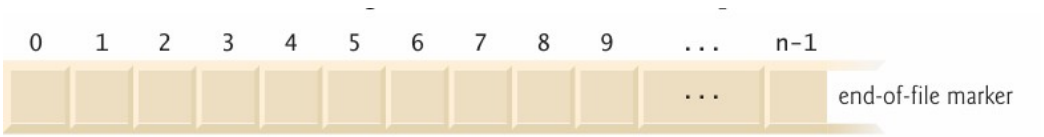
- Python treats:
  - **Text files** as a sequence of **characters**
  - **Binary files** as a sequence of **bytes**
- File positions start at **0** (zero-based indexing)
- For a file with n characters/bytes, the last position is $n - 1$



- Operating systems mark the **End of File (EOF)**

File Representation and EOF

- Python treats:
  - **Text files** as a sequence of **characters**
  - **Binary files** as a sequence of **bytes**
- File positions start at **0** (zero-based indexing)
- For a file with n characters/bytes, the last position is $n - 1$



- Operating systems mark the **End of File (EOF)**
- Python hides OS-level EOF details from the programmer

Opening and Closing a File

Opening and Closing a File

```
file = open("data.txt", "r")
# File operations
file.close()
```

Opening and Closing a File

```
file = open("data.txt", "r")
# File operations
file.close()
```

- `open(filename, mode)`

Opening and Closing a File

```
file = open("data.txt", "r")
# File operations
file.close()
```

- `open(filename, mode)`
- Always close the file after use

File Modes

- `"r"` : Read text file (default mode)

File Modes

- `"r"` : Read text file (default mode)
- `"w"` : Write text file (erases existing content)

File Modes

- `"r"` : Read text file (default mode)
- `"w"` : Write text file (erases existing content)
- `"a"` : Append at end (creates file if needed)

File Modes

- `"r"` : Read text file (default mode)
- `"w"` : Write text file (erases existing content)
- `"a"` : Append at end (creates file if needed)
- `"r+"` : Read and write

File Modes

- `"r"` : Read text file (default mode)
- `"w"` : Write text file (erases existing content)
- `"a"` : Append at end (creates file if needed)
- `"r+"` : Read and write
- `"w+"` : Read and write (erases existing content)

File Modes

- `"r"` : Read text file (default mode)
- `"w"` : Write text file (erases existing content)
- `"a"` : Append at end (creates file if needed)
- `"r+"` : Read and write
- `"w+"` : Read and write (erases existing content)
- `"a+"` : Read and append (creates file if needed)

Text File Processing – Reading

```
file = open("data.txt", "r")
content = file.read()
print(content)
file.close()
```

Reading from a Text File

- `read()` : Reads the **entire file** as one string

Reading from a Text File

- `read()` : Reads the **entire file** as one string
- `f.read(10)` $\rightarrow$ reads first 10 characters.

Reading from a Text File

- `read()` : Reads the **entire file** as one string
- `f.read(10)` $\rightarrow$ reads first 10 characters.

```
# grades.txt
1 Alice A
2 Bob B
3 Charlie A
```

Reading from a Text File

- `read()` : Reads the **entire file** as one string
- `f.read(10)` → reads first 10 characters.

```
# grades.txt
1 Alice A
2 Bob B
3 Charlie A
```

```
with open("grades.txt", "r") as f:
    content = f.read()     # whole file as one string
    print(content)
```

Reading from a Text File

- `read()` : Reads the **entire file** as one string
- `f.read(10)` → reads first 10 characters.

```
# grades.txt
1 Alice A
2 Bob B
3 Charlie A
```

```
with open("grades.txt", "r") as f:
    content = f.read()     # whole file as one string
    print(content)
```

```
1 Alice A
2 Bob B
3 Charlie A
```

Reading from a Text File

- `readline()` : Reads **one line at a time**

Reading from a Text File

- `readline()` : Reads **one line at a time**
- Each call moves the file pointer forward.

Reading from a Text File

- `readline()` : Reads **one line at a time**
- Each call moves the file pointer forward.
- Returns: string (includes `\n` at the end, except possibly for the last line).

Reading from a Text File

- `readline()` : Reads **one line at a time**
- Each call moves the file pointer forward.
- Returns: string (includes `\n` at the end, except possibly for the last line).

```
with open("grades.txt", "r") as f:
    line1 = f.readline()   # first line
    line2 = f.readline()   # second line
    print(line1, line2)
```

Reading from a Text File

- `readline()` : Reads **one line at a time**
- Each call moves the file pointer forward.
- Returns: string (includes `\n` at the end, except possibly for the last line).

```
with open("grades.txt", "r") as f:
    line1 = f.readline()   # first line
    line2 = f.readline()   # second line
    print(line1, line2)
```

```
1 Alice A
 2 Bob B
```

Reading from a Text File

- `readlines()` : Reads **all lines** into a list

Reading from a Text File

- `readlines()` : Reads **all lines** into a list

```
with open("grades.txt", "r") as f:
    lines = f.readlines()
    print(lines)
```

Reading from a Text File

- `readlines()` : Reads **all lines** into a list

```
with open("grades.txt", "r") as f:
    lines = f.readlines()
    print(lines)
```

```
['1 Alice A\n', '2 Bob B\n', '3 Charlie A\n']
```

Writing into a Text File

```
file = open("output.txt", "w")
file.write("Hello Python\n")
file.write("File Handling")
file.close()
```

Additional Notes Regarding Files

- Always close files to avoid data loss

Additional Notes Regarding Files

- Always close files to avoid data loss
- Prefer using the `with` statement

Additional Notes Regarding Files

- Always close files to avoid data loss
- Prefer using the `with` statement
- File pointer position matters

Additional Notes Regarding Files

- Always close files to avoid data loss
- Prefer using the `with` statement
- File pointer position matters
- Reset the pointer using `seek()` before reading

Additional Notes Regarding Files

- Always close files to avoid data loss
- Prefer using the `with` statement
- File pointer position matters
- Reset the pointer using `seek()` before reading

```
with open('output1.txt', 'w+') as f:
    f.write('This is first line.')
    f.write('This is second line.')
    f.seek(0) # reset pointer to start of file
    r = f.read()
    print(repr(r))
```

Additional Notes Regarding Files

- Always close files to avoid data loss
- Prefer using the `with` statement
- File pointer position matters
- Reset the pointer using `seek()` before reading

```
with open('output1.txt', 'w+') as f:
    f.write('This is first line.')
    f.write('This is second line.')
    f.seek(0) # reset pointer to start of file
    r = f.read()
    print(repr(r))
```

```
OUTPUT: 'This is first line.This is second line.'
```

Using with Statement

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

Using with Statement

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

- Automatically closes the file

Updating Text Files

- Formatted text files **cannot be updated in place**

Updating Text Files

- Formatted text files **cannot be updated in place**
- Reason: records and fields have **variable length**

Updating Text Files

- Formatted text files **cannot be updated in place**
- Reason: records and fields have **variable length**
- Overwriting longer data may **destroy other data**

Updating Text Files

- Formatted text files **cannot be updated in place**
- Reason: records and fields have **variable length**
- Overwriting longer data may **destroy other data**

Example:

- Original record: `300 White 0.00`

Updating Text Files

- Formatted text files **cannot be updated in place**
- Reason: records and fields have **variable length**
- Overwriting longer data may **destroy other data**

Example:

- Original record: `300 White 0.00`
- Updating to `Williams` corrupts the record

Safe Method to Update a Text File

- Create a temporary file

Safe Method to Update a Text File

- Create a temporary file
- Copy unchanged records

Safe Method to Update a Text File

- Create a temporary file
- Copy unchanged records
- Write updated record

Safe Method to Update a Text File

- Create a temporary file
- Copy unchanged records
- Write updated record
- Delete old file and rename temp file

Safe Method to Update a Text File

- Create a temporary file
- Copy unchanged records
- Write updated record
- Delete old file and rename temp file

```
import os

# Step 1: Open original file for reading
grades = open('grades.txt', 'r')

# Step 2: Open temporary file for writing
temp_file = open('temp_file.txt', 'w')
```

```
# Step 3: Use context manager to ensure both files close automatically
with grades, temp_file:
    for record in grades:
        serial, name, grade = record.split()

        if name != 'Bob':
            # Write unchanged record
            temp_file.write(record)
        else:
            # Replace Bob with David
            new_record = ' '.join([serial, 'David', grade])
            temp_file.write(new_record + '\n')

# Step 4: Replace old file with updated one
os.remove('grades.txt')
os.rename('temp_file.txt', 'grades.txt')

print("Update complete. Bob has been replaced by David.")
```

Handling Exceptions

- Runtime errors disrupt program flow

Handling Exceptions

- Runtime errors disrupt program flow
- Exception handling avoids program crash

try–except Block

```
try:
    x = int(input("Enter a number: "))
    print(10 / x)
except ZeroDivisionError:
    print("Division by zero not allowed")
```

try–except Block

```
try:
    x = int(input("Enter a number: "))
    print(10 / x)
except ZeroDivisionError:
    print("Division by zero not allowed")
```

- Multiple exceptions can be caught in one except clause using a tuple:

try–except Block

```
try:
    x = int(input("Enter a number: "))
    print(10 / x)
except ZeroDivisionError:
    print("Division by zero not allowed")
```

- Multiple exceptions can be caught in one except clause using a tuple:

```
except (ValueError, ZeroDivisionError):
```

`else` and `finally` Clauses

- `else` block executes **only if no exception occurs**; used for **normal logic**

`else` and `finally` Clauses

- `else` block executes **only if no exception occurs**; used for **normal logic**
- `finally` block executes **whether an exception occurs or not**; used for **cleanup**

`else` and `finally` Clauses

- `else` block executes **only if no exception occurs**; used for **normal logic**
- `finally` block executes **whether an exception occurs or not**; used for **cleanup**

```
f = None
try:
    f = open("grades.txt", "r")
    data = f.read()
except FileNotFoundError:
    print("File not found.")
else:
    print("File read successfully.")
    print(data)
finally:
    if f is not None:
        f.close()  # cleanup resource
    print("File closed.")
```

```
File read successfully.
1 Alice A
2 Bob B
3 Charlie A

File closed.
```

Explicitly Raising an Exception

- Programmer-defined exceptions

Explicitly Raising an Exception

- Programmer-defined exceptions
- Enforce logical constraints

raise Keyword

```
age = int(input("Enter age: "))
if age < 18:
    raise ValueError("Age must be 18 or above")
```

raise Keyword

```
age = int(input("Enter age: "))
if age < 18:
    raise ValueError("Age must be 18 or above")
```

```
INPUT: 17
OUTPUT:
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[2], line 3
      1 age = int(input("Enter your age: "))
      2 if age<18:
----> 3     raise ValueError("Age must be 18 or above!")

ValueError: Age must be 18 or above!
```

CSV Files in Python

- CSV (Comma-Separated Values) stores tabular data as text

CSV Files in Python

- CSV (Comma-Separated Values) stores tabular data as text
- Each line represents a record

CSV Files in Python

- CSV (Comma-Separated Values) stores tabular data as text
- Each line represents a record
- Fields are separated by commas

Writing to a CSV File

- Use the `csv` module to write CSV files

Writing to a CSV File

- Use the `csv` module to write CSV files

```
import csv

with open("data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["ID", "Name", "Marks"])
    writer.writerow([1, "Alice", 85])
    writer.writerow([2, "Bob", 90])
    writer.writerow([3, "Devid", 80])
```

Reading from a CSV File

- CSV files are read row-by-row

Reading from a CSV File

- CSV files are read row-by-row

```
import csv

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

Reading from a CSV File

- CSV files are read row-by-row

```
import csv

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

```
OUTPUT:
['ID', 'Name', 'Marks']
['1', 'Alice', '85']
['2', 'Bob', '90']
['3', 'Devid', '80']
```

Cautions While Using CSV Files

- Commas inside data fields must be quoted

Cautions While Using CSV Files

- Commas inside data fields must be quoted
- Missing commas cause incorrect field splitting

Cautions While Using CSV Files

- Commas inside data fields must be quoted
- Missing commas cause incorrect field splitting
- Extra commas create unwanted empty fields

Reading CSV Files into Pandas DataFrames

- Pandas simplifies CSV file handling

Reading CSV Files into Pandas DataFrames

- Pandas simplifies CSV file handling
- Data is stored as a DataFrame

Reading CSV Files into Pandas DataFrames

- Pandas simplifies CSV file handling
- Data is stored as a DataFrame

```
import pandas as pd
df = pd.read_csv("data.csv")
print(df)
```

Reading CSV Files into Pandas DataFrames

- Pandas simplifies CSV file handling
- Data is stored as a DataFrame

```
import pandas as pd
df = pd.read_csv("data.csv")
print(df)
```

```
OUTPUT:
   ID   Name  Marks
0   1  Alice     85
1   2    Bob     90
2   3  Devid     80
```