File Handling and Exception Handling in Python

Dr. Awais Ahmed

24 March 2024

Introduction

- **File Handling**: Refers to the process of creating, reading, updating, and deleting files stored on a filesystem.
- **Exception Handling**: A mechanism to handle runtime errors, ensuring the program can gracefully manage unexpected situations or errors.

1 Part 1: File Handling in Python

1.1 Basic Concepts

- **File**: A file is a collection of data stored on a storage device.
- **File Modes**: File handling in Python allows different modes of operations:
  - `r`: Read (default mode)
  - `w`: Write (overwrites file)
  - `a`: Append (adds to the end of file)
  - `b`: Binary mode (e.g., `rb`, `wb`)
  - `t`: Text mode (default)

1.2 Opening a File

Use the built-in `open()` function.
Syntax: `open(file_path, mode)`

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

1.3 Reading from a File

Methods:

- `read(size)`: Reads the specified number of bytes.
- `readline()`: Reads a single line.
- `readlines()`: Reads all lines and returns a list.

Example:

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

1.4 Writing to a File

Methods:

- `write(string)`: Writes a string to the file.
- `writelines(list)`: Writes a list of strings.

Example:

```
with open("output.txt", "w") as file:
    file.write("Hello, World!\n")
    file.writelines(["Line 1\n", "Line 2\n"])
```

1.5 File Closure

Files should be properly closed after operations; using `with` statement ensures files are closed automatically.

```
file = open("example.txt", "r")
# operations
file.close()  # Manual close; not needed if using "with"
```

1.6 Deleting a File

Use the `os` module to delete files.

Example:

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

1.7 Working with Directories

- **Creating a Directory**: `os.mkdir("directory_name")`
- **Removing a Directory**: `os.rmdir("directory_name")`
- **Listing Files in a Directory**: `os.listdir("directory_name")`

2 Part 2: Exception Handling in Python

2.1 What is an Exception?

An exception is an error that occurs during the execution of a program. Common examples: `ZeroDivisionError`, `FileNotFoundError`, `ValueError`.

2.2 Basic Exception Handling

Use `try`, `except` blocks to catch and handle exceptions.
Example:

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You cannot divide by zero.")
```

2.3 Multiple Except Blocks

Handle different exceptions using multiple `except` blocks.
Example:

```
try:
    file = open("nonexistent.txt", "r")
except FileNotFoundError:
    print("File not found.")
except Exception as e:
    print(f"An error occurred: {e}")
```

2.4 Finally Block

The `finally` block will execute regardless of whether an exception was raised or not.
Example:

```
try:
    file = open("example.txt", "r")
except FileNotFoundError:
    print("File not found.")
finally:
    file.close()  # Ensures the file is closed
```

2.5 Raising Exceptions

Use the `raise` keyword to raise exceptions explicitly.
Example:

```
def check_positive(number):
    if number < 0:
        raise ValueError("Number must be positive.")
    return number

try:
    check_positive(-5)
except ValueError as ve:
    print(ve)
```

2.6 Custom Exceptions

You can create your own exceptions by extending the built-in `Exception` class.

Example:

```
class MyCustomError(Exception):
    pass

def check_value(value):
    if value < 10:
        raise MyCustomError("Value is too small.")

try:
    check_value(5)
except MyCustomError as e:
    print(e)
```

Conclusion

- **File Handling**: Essential for interacting with files, allowing operations such as reading, writing, and deleting.
- **Exception Handling**: Crucial for making your program robust and capable of handling errors gracefully.

Understanding both file and exception handling is vital for effective and reliable programming in Python, enabling developers to create applications that can handle a variety of input sources and unexpected situations with ease.