**File Handling and Exceptions**

**1. File Handling: Reading from and writing to text files**

Python provides built-in functions to work with files. Here are some of the most important functions and methods you will use:

Opening a File (open())

To open a file in Python, we use the built-in open() function. It returns a file object that allows us to interact with the file.

**Syntax:**

file = open("filename.txt", "mode")

Where "filename.txt" is the name of the file, and "mode" specifies the mode in which the file is opened. Common modes include:

* + "r": Read (default mode) – Opens the file for reading.
  + "w": Write – Opens the file for writing (creates the file if it doesn't exist).
  + "a": Append – Opens the file for appending content at the end.
  + "rb", "wb", "ab": Open in binary mode (useful for non-text files).

Reading a File (read(), readlines())

* read(): Reads the entire content of the file as a string.
* readlines(): Reads the file line by line and returns a list of lines.

Writing to a File (write())

* write(): Writes a string to the file.

**2. Exception Handling: Using try, except, else, and finally**

When working with files (or any other resource), errors can happen. For example:

* The file might not exist.
* You might not have permission to read or write the file.

Python provides a way to handle these errors gracefully using exceptions.

* try: Code that might raise an exception.
* except: Handles the exception if it occurs.
* else: Code that runs if no exception occurs.
* finally: Code that runs no matter what (even if there was an exception).

**Example:** Read a File, Count Words, and Handle Errors

Now, let’s write a program that reads a file, counts the words, and handles potential errors (like the file not being found or issues with file access).

**Example Code:**

def count\_words\_in\_file(filename):

try:

# Open the file in read mode

with open(filename, 'r') as file:

# Read the content of the file

content = file.read()

# Split the content into words (using whitespace as a separator)

words = content.split()

# Count the number of words

word\_count = len(words)

# Return the word count

return word\_count

except FileNotFoundError:

print(f"Error: The file '{filename}' was not found.")

except IOError:

print("Error: There was an issue with reading the file.")

else:

print("File read successfully.")

finally:

print("Execution completed.")

# Example usage:

filename = "example.txt" # Make sure the file exists in your working directory

word\_count = count\_words\_in\_file(filename)

if word\_count:

print(f"The file contains {word\_count} words.")

**Explanation:**

* with open(filename, 'r') as file:: This opens the file in read mode and ensures that the file is automatically closed when the block of code is finished. It’s a best practice to use with because it handles closing the file even if an error occurs.
* file.read(): Reads the entire content of the file.
* content.split(): Splits the content of the file into a list of words, using spaces as separators.
* len(words): Counts the number of words in the list.

**Error Handling:**

* FileNotFoundError: Raised if the file doesn't exist.
* IOError: A more general error for any input/output problems (e.g., permission errors).

**Output Examples:**

If the file exists and has content:

File read successfully.

The file contains 42 words.

Execution completed.

If the file doesn’t exist:

Error: The file 'example.txt' was not found.

Execution completed.

If there is another I/O issue (like permission problems):

Error: There was an issue with reading the file.

Execution completed.