PLP-PYTHON-Week-4-ASSIGMENT-File-Handling-and-Exception-Handling
File Read & Write Challenge

Step 1: Understand the Requirements

The program should:

1. Prompt the user for a filename to read.
2. Handle exceptions if the file does not exist or cannot be read.
3. Read the content of the file, modify it (e.g., convert to uppercase), and write the modified content to a new file.

Step 2: Set Up the Program Structure

Organize your code into functions for clarity and reusability.

Example Structure:

```
def main():
    # Main function to run the file handling program
    pass
```

Step 3: Get User Input for the Filename

Use the input() function to ask the user for the filename they want to read.

Example:

```
python
def get_filename():
    return input("Please enter the filename to read: ")
```

Step 4: Read the File with Error Handling

Use a try and except block to handle potential errors when opening and reading the file. This will allow you to catch exceptions like FileNotFoundError.

Example:

```
python
def read_file(filename):
    try:
        with open(filename, 'r') as file:
            content = file.read()
        return content
    except FileNotFoundError:
        print(f"Error: The file '{filename}' does not exist.")
        return None
    except IOError:
        print(f"Error: The file '{filename}' cannot be read.")
        return None
```

Step 5: Modify the Content

Create a function to modify the content read from the file. For this example, let's convert the text to uppercase.

Example:

```python
def modify_content(content):
    return content.upper()
```

Step 6: Write the Modified Content to a New File

Create a function to write the modified content to a new file. You can prompt the user for the new filename.

Example:

```python
def write_file(new_filename, content):
    try:
        with open(new_filename, 'w') as file:
            file.write(content)
        print(f"Modified content written to '{new_filename}'.")
    except IOError:
        print(f"Error: The file '{new_filename}' cannot be written.")
```

Step 7: Combine Everything in the Main Function

Integrate all the functions into the main() function to create a complete program.

Full Program:

```python
def get_filename():
    return input("Please enter the filename to read: ")

def read_file(filename):
    try:
        with open(filename, 'r') as file:
            content = file.read()
        return content
    except FileNotFoundError:
        print(f"Error: The file '{filename}' does not exist.")
        return None
    except IOError:
        print(f"Error: The file '{filename}' cannot be read.")
        return None

def modify_content(content):
    return content.upper()
```

```
def write_file(new_filename, content):
    try:
        with open(new_filename, 'w') as file:
            file.write(content)
        print(f"Modified content written to '{new_filename}'.")
    except IOError:
        print(f"Error: The file '{new_filename}' cannot be written.")

def main():
    filename = get_filename()
    content = read_file(filename)

    if content is not None:
        modified_content = modify_content(content)
        new_filename = input("Please enter the new filename to save the modified content: ")
        write_file(new_filename, modified_content)

if __name__ == "__main__":
    main()
```

Step 8: Test the Program

Run the program and test it with various filenames, including:

- A valid file that exists.
- A file that does not exist.
- A file that cannot be read due to permissions.

Step 9: Follow Best Practices

- Use clear and descriptive variable and function names.
- Handle exceptions specifically to provide meaningful error messages.
- Ensure that the program is user-friendly and prompts for input clearly.

By following these steps, you can create a robust file handling program in Python that effectively reads, modifies, and writes files while handling potential errors gracefully.