Handling IO Exceptions

Handling I/O (Input/Output) exceptions is crucial in Python to ensure that your program gracefully responds to potential issues when reading from or writing to files, sockets, or other I/O operations. Common I/O exceptions include `FileNotFoundError`, `PermissionError`, and `IOError`. Here's an overview of handling I/O exceptions with examples:

1. **Using `try` and `except` Blocks:**
   The `try` and `except` blocks are used to catch and handle exceptions. You can specify the type of exception to catch or use a more general `except` block to catch any exception.

   ```python
   try:
       with open('nonexistent_file.txt', 'r') as file:
           content = file.read()
   except FileNotFoundError as e:
       print(f"File not found: {e}")
   except Exception as e:
       print(f"An unexpected error occurred: {e}")
   ```

   In this example, a `FileNotFoundError` is caught specifically, and a more general `Exception` block catches any other unexpected errors.

2. **Using `finally` Block:**
   The `finally` block is executed whether an exception occurs or not. It's commonly used for cleanup operations, such as closing files.

   ```python
   try:
       file = open('example.txt', 'r')
       content = file.read()
   except FileNotFoundError as e:
       print(f"File not found: {e}")
   except Exception as e:
       print(f"An unexpected error occurred: {e}")
   finally:
       file.close()
   ```

   The `finally` block ensures that the file is closed, even if an exception occurs during the try block.

3. **Handling Multiple Exceptions:**
   You can handle multiple exceptions in a single `except` block or use multiple `except` blocks for different exception types.

   ```python
   ```

```
try:
    with open('example.txt', 'r') as file:
        content = file.read()
        result = 1 / 0  # Intentionally raising a ZeroDivisionError
except (FileNotFoundError, ZeroDivisionError) as e:
    print(f"Error occurred: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Here, both `FileNotFoundError` and `ZeroDivisionError` are caught in the same `except` block.

4. **Using `else` Block:**
   The `else` block is executed if no exceptions are raised in the `try` block.

   ```python
   try:
       with open('example.txt', 'r') as file:
           content = file.read()
   except FileNotFoundError as e:
       print(f"File not found: {e}")
   except Exception as e:
       print(f"An unexpected error occurred: {e}")
   else:
       print("File read successfully.")
   ```

   The `else` block is executed if the file is read successfully without any exceptions.

5. **Handling Specific Exceptions:**
   You can handle specific exceptions based on the error message or content.

   ```python
   try:
       with open('example.txt', 'r') as file:
           content = file.read()
           if 'error' in content:
               raise ValueError("Custom error detected in file content.")
   except FileNotFoundError as e:
       print(f"File not found: {e}")
   except ValueError as e:
       print(f"ValueError: {e}")
   except Exception as e:
       print(f"An unexpected error occurred: {e}")
   ```

   Here, a `ValueError` is raised if the word "error" is found in the file content.

When handling I/O exceptions, it's important to provide informative error messages, log errors, and gracefully handle potential issues. Depending on the application, you may want to retry the operation, prompt the user for input, or take other appropriate actions. Always consider the specific requirements of your program when designing exception handling mechanisms.