🔥 Python Exception Handling! 🔥

**What is Exception Handling?**  
When writing Python code, **errors (exceptions)** can occur at runtime, causing the program to crash. Exception handling helps us **gracefully handle errors** instead of abruptly stopping the program.

**📌 1. Understanding Exceptions in Python**

Python has several **built-in exceptions**, such as:  
🔴 **ZeroDivisionError** → Dividing by zero.  
🔴 **ValueError** → Invalid input value.  
🔴 **TypeError** → Wrong data type used.  
🔴 **FileNotFoundError** → Trying to open a missing file.

💡 **Example: Without Exception Handling**

num1 = int(input("Enter a number: ")) # User enters a letter instead of a number

num2 = int(input("Enter another number: "))

result = num1 / num2 # If num2 is 0, it crashes!

print("Result:", result)

❌ If you enter **"hello"** or **0**, the program crashes.

✅ **Let's handle it properly using try-except.**

**📌 2. Handling Exceptions with try-except**

We use **try** to attempt running code and **except** to catch errors.

try:

num1 = int(input("Enter a number: "))

num2 = int(input("Enter another number: "))

result = num1 / num2

print("Result:", result)

except ZeroDivisionError:

print("❌ Cannot divide by zero!")

except ValueError:

print("❌ Please enter only numbers!")

**✨ How it Works?**

✅ If **valid input**: Runs smoothly.  
✅ If **num2 = 0**: Catches ZeroDivisionError.  
✅ If **input is text**: Catches ValueError.

**📌 3. Using finally: Always Execute Code**

The **finally** block runs **no matter what happens** (error or no error).

try:

num = int(input("Enter a number: "))

print(100 / num)

except ZeroDivisionError:

print("❌ Cannot divide by zero!")

except ValueError:

print("❌ Invalid number!")

finally:

print("✅ Program execution completed.")

🔹 **finally ensures cleanup, like closing files or releasing resources.**

**📌 4. Using else: Runs if No Error**

The **else** block executes **only if no exception occurs.**

try:

num = int(input("Enter a number: "))

result = 100 / num

except ZeroDivisionError:

print("❌ Cannot divide by zero!")

except ValueError:

print("❌ Invalid number!")

else:

print(f"✅ Success! The result is {result}")

finally:

print("✅ Program execution completed.")

✅ **If no error → Runs else and finally.**  
❌ **If error → Skips else, but finally still runs.**

**📌 5. Catching Multiple Exceptions**

You can catch **multiple exceptions** in a **single except block.**

try:

num = int(input("Enter a number: "))

print(10 / num)

except (ZeroDivisionError, ValueError) as e:

print(f"❌ Error Occurred: {e}")

💡 **Here, e stores the exact error message.**

**📌 6. Raising Custom Exceptions (raise)**

You can **create your own exceptions** using raise.

def withdraw(amount):

if amount < 0:

raise ValueError("❌ Withdrawal amount cannot be negative!")

print(f"✅ Successfully withdrew ${amount}")

try:

withdraw(-100)

except ValueError as e:

print(e)

🔹 **Prevents invalid operations like withdrawing negative money.**

**📌 7. Creating Custom Exceptions**

Python allows **custom exception classes** for better error handling.

class CustomError(Exception):

def \_\_init\_\_(self, message):

self.message = message

super().\_\_init\_\_(self.message)

try:

raise CustomError("This is a custom exception!")

except CustomError as e:

print(f"❌ Caught Custom Error: {e}")

✅ **Use case:** Defining unique errors for specific situations.

**📌 8. Real-World Example: File Handling**

try:

with open("data.txt", "r") as file:

content = file.read()

print(content)

except FileNotFoundError:

print("❌ File not found! Please check the filename.")

✅ **Prevents crashing if the file doesn’t exist.**

**📌 9. Best Practices for Exception Handling**

✔️ **Use Specific Exceptions** (Avoid except Exception: as it catches everything).  
✔️ **Use finally for Cleanup** (Closing files, releasing resources).  
✔️ **Don’t Overuse Exceptions** (Validate inputs before errors occur).  
✔️ **Use raise for Business Logic** (e.g., stopping negative withdrawals).

**🎯 Quick Assessment (Test Yourself!)**

1️⃣ What will happen in the below code if the user enters 0?

try:

num = int(input("Enter a number: "))

print(10 / num)

except ValueError:

print("❌ Invalid input!")

finally:

print("✅ Program finished!")

**2️⃣ How can you modify the code to catch both ZeroDivisionError and ValueError?**

3️⃣ Create a program that:

* Asks a user for **two numbers**.
* If any input is invalid, catch ValueError.
* If **dividing by zero**, catch ZeroDivisionError.
* Always print "✅ Operation Completed." at the end.

🚀 **Try answering these, and I'll review your solutions!**

**📌 Where to Use try-except in Python?**

We use try-except to **handle runtime errors** in our code.  
✅ The **try block** contains code that **may** raise an exception.  
✅ The **except block** handles specific errors if they occur.

**📝 General Rule**

💡 Use try-except **only around risky code** (e.g., file handling, user input, external APIs).  
🔹 **DO NOT** wrap the entire program in try-except, as it hides all errors.

**📌 Example 1: Handling User Input**

❌ **Bad practice:** Wrapping the entire input process.  
✅ **Good practice:** Wrapping only risky operations.

try:

num = int(input("Enter a number: ")) # Risky line (user may enter text)

result = 10 / num # Risky line (num may be 0)

print("Result:", result)

except ValueError:

print("❌ Please enter a valid number!")

except ZeroDivisionError:

print("❌ Cannot divide by zero!")

📌 **Why?**  
✔️ If input is **not a number**, ValueError is caught.  
✔️ If user enters 0, ZeroDivisionError is caught.  
✔️ Only necessary code is inside try.

**📌 Example 2: Handling File Operations**

Opening a file may fail if the file does not exist.

try:

with open("data.txt", "r") as file:

content = file.read()

print(content)

except FileNotFoundError:

print("❌ File not found! Please check the filename.")

📌 **Why?**  
✔️ Only open() and read() are inside try, since they may fail.

**📌 Example 3: Using else and finally**

✔️ **Use else** when the operation **succeeds**.  
✔️ **Use finally** to execute **cleanup code** (e.g., closing files).

try:

num = int(input("Enter a number: "))

result = 10 / num

except (ZeroDivisionError, ValueError) as e:

print(f"❌ Error Occurred: {e}")

else:

print(f"✅ Success! The result is {result}")

finally:

print("✅ Program execution completed.")

📌 **Why?**  
✔️ **If no error**, else runs.  
✔️ **Regardless of error or success**, finally runs.

**📌 Example 4: Handling API Calls**

When calling an API or database, we must **handle failures properly**.

import requests

try:

response = requests.get("https://example.com/data")

response.raise\_for\_status() # Raises an error for HTTP issues

data = response.json() # Risky operation (may fail)

except requests.exceptions.RequestException as e:

print(f"❌ Network error: {e}")

except ValueError:

print("❌ Failed to decode JSON response!")

📌 **Why?**  
✔️ **Handles connection failures** (e.g., no internet).  
✔️ **Handles invalid JSON responses** from the API.

**🎯 Quick Challenge (Test Yourself!)**

👉 Where should you use try-except in the following code?

import json

file = open("config.json", "r")

config = json.load(file)

print("API Key:", config["api\_key"])

file.close()

1️⃣ What **errors** can occur here?  
2️⃣ Where will you **add try-except**?

🚀 **Try answering, and I'll review your solution!**

Great start! You're on the right track! However, there are a couple of things to adjust. Here's how you can improve it:

**Key Issues:**

1. **Exception Name:** The correct exception for file not found is FileNotFoundError (not FileNotFoundException).
2. **Handling Multiple Errors:** Instead of a generic except e, we should catch specific exceptions like json.JSONDecodeError for issues in loading the JSON or KeyError if the api\_key is missing.
3. **finally Block Placement:** It's fine to place file.close() in the finally block, but the file needs to be opened **inside the try block** to ensure proper handling.

**Corrected Code:**

import json

try:

file = open("config.json", "r") # Opening the file (risky)

config = json.load(file) # Risky line (may fail if JSON is invalid)

print("API Key:", config["api\_key"]) # Risky (if key doesn't exist)

except FileNotFoundError:

print("❌ Error: File not found!")

except json.JSONDecodeError:

print("❌ Error: Failed to decode JSON!")

except KeyError:

print("❌ Error: 'api\_key' not found in the config file!")

except Exception as e: # Catch any other errors

print(f"❌ Some error occurred: {e}")

finally:

try:

file.close() # Close the file (if opened)

except NameError:

pass # File may not have been opened if there was an error

**Key Points:**

* **FileNotFoundError:** This catches the error when the file doesn't exist.
* **json.JSONDecodeError:** This handles any issues related to the JSON format (e.g., malformed JSON).
* **KeyError:** Catches cases when the key 'api\_key' is missing in the loaded JSON.
* **Generic Exception (Exception as e):** This catches any unexpected errors (e.g., permission issues).
* **Finally Block:** It makes sure that file.close() is called if the file was opened, and if there was an error before the file opened, it won't break the program due to a NameError.

**🎯 Key Takeaways:**

* Always be specific with exceptions to avoid hiding errors.
* Use finally to clean up resources like files or network connections.
* Use else if you want code to run only when the try block doesn't raise an exception.