🔥 Exception Handling in Python (Complete Guide)

1️⃣ What is an Exception?

An **exception** is a runtime error that **stops normal program flow**.

Example (without handling)

```
a = 10
b = 0
print(a / b)
```

❌ Output:

```
ZeroDivisionError: division by zero
```

2️⃣ Why Exception Handling is Important (Big Data Context)

In **data engineering**, errors are common:

- File not found
- Corrupt data
- Network failure
- Database connection failure
- Type mismatch

👉 Instead of crashing pipelines, **we handle errors gracefully**.

3️⃣ Basic Syntax

```
try:
    # risky code
except:
    # runs if exception occurs
```

Example

```
try:
    print(10 / 0)
except:
    print("Error occurred")
```

4️⃣ Catching Specific Exceptions (VERY IMPORTANT)

Common Built-in Exceptions

Exception	When it occurs
ZeroDivisionError	divide by zero
ValueError	wrong value type
TypeError	wrong data type
FileNotFoundError	missing file
IndexError	invalid index
KeyError	missing dictionary key

Example

```
try:
    x = int("abc")
except ValueError:
    print("Conversion failed")
```

5️⃣ Multiple except Blocks

```
try:
    a = int(input())
    b = int(input())
    print(a / b)
except ZeroDivisionError:
    print("Cannot divide by zero")
except ValueError:
    print("Invalid input")
```

✔ Best practice: **specific → generic**

6️⃣ Generic Exception Handling

```
try:
    print(a)
except Exception as e:
    print("Error:", e)
```

📌 Exception is the **parent class of most errors**.

7️⃣ else Block (Runs if NO Exception)

```
try:
    a = 10
    b = 2
    print(a / b)
except ZeroDivisionError:
    print("Error")
else:
    print("Execution successful")
```

✔ Useful when success logic should run **only if no error**.

8️⃣ finally Block (ALWAYS Executes)

Used for **cleanup**.

```
try:
    f = open("data.txt", "r")
    print(f.read())
except FileNotFoundError:
    print("File missing")
finally:
    print("Closing resources")
```

📌 Used for:

- Closing files
- Closing DB connections

- Releasing memory

9️⃣ Complete Flow (try–except–else–finally)

```
try:
    x = int(input())
    print(10 / x)
except Exception as e:
    print("Error:", e)
else:
    print("No error occurred")
finally:
    print("Program ended")
```

🔟 Raising Exceptions (Manual Error Trigger)

You can **force an exception**.

```
age = -5
if age < 0:
    raise ValueError("Age cannot be negative")
```

📌 Useful in **data validation**.

1️⃣1️⃣ Custom Exceptions (ADVANCED)

Create your own exception.

```
class InvalidAgeError(Exception):
    pass


age = -1
if age < 0:
    raise InvalidAgeError("Invalid age")
```

✔ Used in:

- APIs
- Data validation layers
- Enterprise projects

1️⃣2️⃣ Exception Handling with Functions

```
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Invalid division"


print(divide(10, 0))
```

1️⃣3️⃣ Exception Handling with Files (Very Important for Data Engineers)

```
try:
    with open("input.csv", "r") as f:
        print(f.read())
except FileNotFoundError:
    print("Input file missing")
```

✔ with automatically closes file (better than finally).

1️⃣4️⃣ Exception Handling with Lists & Dicts

```
lst = [1, 2, 3]
try:
    print(lst[10])
except IndexError:
    print("Index out of range")
d = {"name": "Tarun"}
try:
```

```
    print(d["age"])
except KeyError:
    print("Key not found")
```

1️⃣5️⃣ Handling Multiple Exceptions Together

```
try:
    x = int("abc")
    print(10 / 0)
except (ValueError, ZeroDivisionError) as e:
    print("Error:", e)
```

1️⃣6️⃣ Nested try-except

```
try:
    try:
        print(10 / 0)
    except ZeroDivisionError:
        print("Inner handled")
except:
    print("Outer handled")
```

📌 Avoid nesting too much → reduces readability.

1️⃣7️⃣ Exception Propagation

```
def f1():
    return 10 / 0

def f2():
    f1()

try:
```

```
    f2()
except ZeroDivisionError:
    print("Handled at top level")
```

✔ Error bubbles up until handled.

1️⃣8️⃣ Best Practices (Interview Gold ⭐)

✅ Catch **specific exceptions**
❌ Don’t use empty except: blindly
✅ Use finally or with for cleanup
✅ Log errors in real projects
❌ Don’t hide critical exceptions

1️⃣9️⃣ Real-World Data Engineering Example

```
def read_data(file):
    try:
        with open(file, "r") as f:
            return f.read()
    except FileNotFoundError:
        return "File missing"
    except PermissionError:
        return "Access denied"
    except Exception as e:
        return str(e)

print(read_data("data.csv"))
```

2️⃣0️⃣ Interview Questions You WILL Get

1. Difference between except and Exception
2. Use of else block?
3. Difference between raise and assert

4. Custom exception use cases
5. Why finally is important?

🧠 One-Line Summary

Exception handling prevents program crashes and makes Python applications reliable, especially in data pipelines and production systems.