Exception Handling in Python

What is Exception Handling?

Gracefully handles unexpected issues (e.g., wrong input, missing files) without crashing the program.

> **Example:** ATM machine prompts errors like “Invalid Amount” instead of shutting down.

Types of Errors

1. Syntax Error

Occurs before execution (e.g., missing brackets, wrong keywords).

```
print("Hello"   # ❌ Missing closing bracket
```

2. Runtime Error

Occurs during execution (e.g., divide by 0).

```
num = 10 / 0  # ❌ DivisionByZeroError
```

Try-Except Block

Used to catch and handle exceptions.

```
try:
        # Prompt the user to enter a number
        num = int(input("Enter a number: "))
        # Attempt to divide 10 by the entered number
        print(10 / num)
        except ZeroDivisionError:
        # Handle the case where the user enters zero
        print("Error! You cannot divide by zero.")
```

Handling Multiple Exceptions

```
Try:
        # Prompt the user to enter the first number
        num1 = int(input('Enter first number: '))
        # Prompt the user to enter the second number
        num2 = int(input('Enter second number: '))
        # Attempt to divide the first number by the second number
        result = num1 / num2
        # Print the result of the division
        print('Result:', result)
except ZeroDivisionError:
```

```
        # Handle the case where the second number is zero
        print('Error: Cannot divide by zero!')
except ValueError:
        # Handle the case where the input is not a valid integer
        print('Error: Invalid input! Please enter numbers only.')
```

Generic Exception Handling

```
Try:
        # Code that may raise an exception
        num = int(input("Enter a number: "))
        result = 10 / num # May raise ZeroDivisionError
        print("Result:", result)
except Exception as e: # Catching all exceptions
        print("An error occurred:", e)
```

Finally Block

Executes regardless of try/except result—used for cleanup like file closing.

```
Try:
        # Attempt to open the file in read mode
        file = open("example.txt", "r")
        # Read the content of the file
        content = file.read()
except FileNotFoundError:
        # Handle the case where the file does not exist
        print("File not found!")
finally:
        # This block runs no matter what
        print("Closing file...")
        file.close()
```

Custom Exceptions

```
def withdraw(amount):
        # Check if the amount is negative
        if amount < 0:
                # Raise a ValueError if the amount is negative
                raise ValueError("Amount cannot be negative!")
        # Print the withdrawal amount
        print(f"Withdrawing ${amount}")
try:
```

```
        # Attempt to withdraw a negative amount
        withdraw(-100)
except ValueError as e:
        # Handle the ValueError and print the error message
        print(f"Error: {e}")
```

Best Practices

- Always close files (file.close() or with open())
- Use try-except for error resilience
- Use custom messages for better UX
- Avoid silent failures (e.g., empty except: blocks)

Final Thoughts

File and JSON handling are essential for building scalable systems, and exception handling ensures the system remains user-friendly and error-resilient. Practice coding with try, except, finally, and experiment with real datasets and files.