Exception Handling in Python (Class XI–XII)

1. What is an Exception?

An exception is a runtime error that interrupts the normal flow of a program.

2. Why Exception Handling is Needed

• Prevents abrupt program termination
• Handles invalid user input
• Improves program reliability
• Helps in debugging and error analysis

3. Types of Errors

• Syntax Error – detected before execution
• Runtime Error (Exception) – occurs during execution
• Logical Error – program runs but produces incorrect output

4. Common Built■in Exceptions

ZeroDivisionError, ValueError, TypeError, IndexError, KeyError, NameError, FileNotFoundError

5. try–except Block

Used to catch and handle exceptions gracefully.

Example: try–except

```
try:
    x = int(input("Enter number: "))
    print(10/x)
except ZeroDivisionError:
    print("Division by zero not allowed")
except ValueError:
    print("Invalid input")
```

6. else Block

The else block executes only if no exception occurs.

```
try:
    a = int(input())
    b = int(input())
    print(a/b)
except ZeroDivisionError:
    print("Error")
else:
```

```
    print("Successful execution")
```

7. finally Block

The finally block executes whether an exception occurs or not.

```
try:
    f = open("data.txt")
except FileNotFoundError:
    print("File not found")
finally:
    print("End of program")
```

8. raise Keyword

Used to raise an exception explicitly.

```
age = int(input("Enter age: "))
if age < 18:
    raise ValueError("Age must be 18 or above")
```

9. User■Defined Exception

```
class MyError(Exception):
    pass

try:
    raise MyError("Custom error occurred")
except MyError as e:
    print(e)
```

CBSE Exam Tips

• Always use specific exceptions
• try block must be followed by except or finally
• finally block always executes
• Exception handling improves program robustness