# Q1. What is an Exception in Python?

## Difference between Exceptions and Syntax Errors

**Exception**
An *exception* is a runtime error that occurs **during program execution**. It interrupts the normal flow of the program.

**Example:**

```
x = 10 / 0   # ZeroDivisionError
```

**Syntax Error**
A *syntax error* occurs when Python cannot understand the code due to **invalid syntax**. The program does not run at all.

**Example:**

```
if x > 5
    print(x)   # SyntaxError (missing colon)
```

## Difference Table

| Syntax Errors | Exceptions |
|---|---|
| Occur at compile time | Occur at runtime |
| Program does not start | Program starts but crashes |
| Caused by incorrect syntax | Caused by logical/runtime issues |
| Cannot be handled using try-except | Can be handled using try-except |

---

# Q2. What happens when an exception is not handled?

If an exception is **not handled**, Python:

1. Stops program execution
2. Displays an error message (traceback)
3. Terminates the program abruptly

**Example:**

```
x = 10
y = 0
print(x / y)
print("This will not execute")
```

**Output:**

```
ZeroDivisionError: division by zero
```

The last `print()` statement is never executed.

---

## Q3. Which Python statements are used to catch and handle exceptions?

Python uses the following statements:

- `try`
- `except`
- `else`
- `finally`

**Example:**

```
try:
    x = int(input("Enter a number: "))
```

```
    y = 10 / x
except ZeroDivisionError:
    print("Cannot divide by zero")
except ValueError:
    print("Invalid input")
else:
    print("Result:", y)
finally:
    print("Execution completed")
```

---

# Q4. Explain with examples

## `try` and `else`, `finally`, `raise`

### 1. `try` and `else`

- `try` → code that may raise an exception
- `else` → executes only if **no exception occurs**

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ZeroDivisionError:
    print("Division by zero error")
else:
    print("Result:", result)
```

---

### 2. `finally`

- Executes **always**, whether an exception occurs or not
- Used for cleanup (closing files, releasing resources)

```
try:
    f = open("test.txt", "r")
    print(f.read())
except FileNotFoundError:
    print("File not found")
finally:
    print("File operation attempted")
```

---

### 3. `raise`

- Used to **manually trigger** an exception

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

---

## Q5. What are Custom Exceptions in Python?

### Why do we need Custom Exceptions?

**Custom Exceptions** are user-defined exception classes used to handle **specific application-level errors**.

### Why we need them:

- To make error handling more meaningful
- To represent domain-specific problems
- To improve code readability and maintenance

**Example:**

```
class InsufficientBalanceError(Exception):
    pass
```

```
balance = 500
withdraw = 1000

if withdraw > balance:
    raise InsufficientBalanceError("Not enough balance")
```

---

## Q6. Create a custom exception class and use it

### Custom Exception Example

```
class InvalidMarksError(Exception):
    def __init__(self, message):
        super().__init__(message)

try:
    marks = int(input("Enter marks: "))
    if marks < 0 or marks > 100:
        raise InvalidMarksError("Marks must be between 0 and 100")
    print("Marks entered:", marks)

except InvalidMarksError as e:
    print("Custom Exception:", e)
```

**Output (if marks = 120):**

```
Custom Exception: Marks must be between 0 and 100
```