Q1. Why do we use the `Exception` class while creating a Custom Exception?

In Python, **`Exception` is the base class for all built-in, non-system exceptions**.
When we create a custom exception, we inherit from the `Exception` class so that:

Reasons:

1. **Integration with Python's exception-handling mechanism**
   Only classes derived from `BaseException` (usually `Exception`) can be raised and caught using `try-except`.

2. **Consistency and compatibility**
   Custom exceptions behave like built-in exceptions and can be handled using `except Exception`.

3. **Access to useful features**
   The `Exception` class provides standard behavior like error messages and traceback support.

Example:

```
class MyCustomError(Exception):
    pass

raise MyCustomError("This is a custom exception")
```

If we do **not** inherit from `Exception`, Python will not treat the class as an exception.

Q2. Write a Python program to print Python Exception Hierarchy

Python provides the `__subclasses__()` method to inspect the exception hierarchy.

Program:

```
def print_exception_hierarchy(exception, level=0):
    print(" " * level + exception.__name__)
    for subclass in exception.__subclasses__():
        print_exception_hierarchy(subclass, level + 4)

print_exception_hierarchy(BaseException)
```

Output (partial):

```
BaseException
    Exception
        ArithmeticError
            ZeroDivisionError
            OverflowError
        LookupError
            IndexError
            KeyError
```

Q3. What errors are defined in the `ArithmeticError` class?

Explain any two with an example.

`ArithmeticError` is the **base class for arithmetic-related exceptions**.

Common errors under `ArithmeticError`:

1. `ZeroDivisionError`
2. `OverflowError`
3. `FloatingPointError`

1. ZeroDivisionError

Occurs when division by zero is attempted.

```
x = 10 / 0
```

Error: `ZeroDivisionError: division by zero`

---

2. OverflowError

Occurs when a calculation exceeds the maximum limit for a numeric type.

```
import math
print(math.exp(1000))
```

Error: `OverflowError: math range error`

Q4. Why is `LookupError` class used?

Explain `KeyError` and `IndexError` with examples.

`LookupError` is the **base class for errors raised when a lookup operation fails** (like indexing or key access).

Subclasses of `LookupError`:

- `IndexError`
- `KeyError`

KeyError

Raised when a **dictionary key does not exist**.

```
data = {"a": 1}
print(data["b"])
```

Error: `KeyError: 'b'`

IndexError

Raised when accessing an **invalid list index**.

```
nums = [1, 2, 3]
print(nums[5])
```

Error: `IndexError: list index out of range`

Q5. Explain ImportError. What is ModuleNotFoundError?

ImportError

Raised when:

- A module exists but cannot be imported properly
- A specific name cannot be imported from a module

```
from math import square
```

Error: `ImportError: cannot import name 'square'`

ModuleNotFoundError

A **subclass of ImportError** raised when the module itself does not exist.

```
import mymodule
```

Error: `ModuleNotFoundError: No module named 'mymodule'`

Q6. Best Practices for Exception Handling in Python

1. **Catch specific exceptions, not generic ones**

```
except ZeroDivisionError:
```

2. **Avoid using bare `except`**

```
except Exception as e:
```

3. **Use `finally` for cleanup**

- Close files
- Release resources

4. **Do not suppress exceptions silently**

```
except Exception:
    pass   # Bad practice
```

5. **Use custom exceptions for application-specific errors**

6. **Keep try blocks minimal**

- Only wrap code that may raise an exception

7. **Use meaningful error messages**

```
raise ValueError("Invalid age entered")
```