1. Basic Exception Handling (try-except)

Program

```
try:
   num = 10 / 0
   print(num)
except ZeroDivisionError:
   print("Error: Cannot divide by zero.")
```

Explanation

- The try block contains code that may generate an exception.
- Dividing by zero raises a ZeroDivisionError.
- The except block catches the error and displays a message instead of terminating the program.

Output

```
Error: Cannot divide by zero.
```

---

2. Handling Multiple Exceptions

Program

```
try:
   num = int(input("Enter a number: "))
   result = 10 / num
   print("Result:", result)

except ValueError:
   print("Please enter a valid integer.")

except ZeroDivisionError:
   print("Division by zero is not allowed.")
```

Explanation

- ValueError occurs if the user enters non-numeric data.
- ZeroDivisionError occurs if the user enters 0.
- Different exceptions are handled using separate except blocks.

Example Output

```
Enter a number: 0
Division by zero is not allowed.
```

---

3. Using `else` with Exception Handling

Program

```
try:
   num = int(input("Enter a number: "))
except ValueError:
   print("Invalid input.")
else:
   print("You entered:", num)
```

Explanation

- The `else` block executes only if no exception occurs in the `try` block.
- It is useful for code that should run when everything succeeds.

Example Output

```
Enter a number: 25
You entered: 25
```

---

4. Using `finally`

Program

```
try:
   file = open("sample.txt", "r")
   content = file.read()
   print(content)

except FileNotFoundError:
   print("File not found.")

finally:
   print("Program execution completed.")
```

Explanation

- `finally` always executes, whether an exception occurs or not.
- It is commonly used for cleanup tasks such as closing files or database connections.

Output (if file doesn't exist)

```
File not found.
Program execution completed.
```

---

5. Raising an Exception Using `raise`

Program

```
age = int(input("Enter your age: "))

if age < 18:
   raise ValueError("Age must be 18 or above.")

print("You are eligible.")
```

Explanation

- The `raise` keyword is used to generate an exception manually.
- If the age is below 18, a `ValueError` is raised.

Example Output

```
Enter your age: 15
ValueError: Age must be 18 or above.
```

---

6. Custom Exception

Program

```
class InvalidAgeError(Exception):
   pass

try:
   age = int(input("Enter age: "))

   if age < 18:
      raise InvalidAgeError("You must be at least 18 years old.")

   print("Access granted.")

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

Explanation

- A custom exception class is created by inheriting from `Exception`.
- The exception is raised when the age condition is not met.
- The custom exception is then caught and handled.

Example Output

```
Enter age: 16
Custom Exception: You must be at least 18 years old.
```

---

7. Catching Any Exception

Program

```
try:
    num = int(input("Enter a number: "))
    result = 100 / num
    print(result)

except Exception as e:
    print("An error occurred:", e)
```

Explanation

- `Exception` is the base class for most built-in exceptions.
- `as e` stores the exception object so its message can be displayed.

Example Output

```
Enter a number: 0
An error occurred: division by zero
```

Summary of Exception Handling Keywords in Python

Keyword	Purpose
`try`	Contains code that may cause an exception
`except`	Handles exceptions
`else`	Executes if no exception occurs
`finally`	Always executes
`raise`	Manually raises an exception
`Exception`	Base class for most exceptions

Complete Example

```
try:
    num = int(input("Enter a number: "))
    result = 100 / num

except ValueError:
    print("Invalid number entered.")

except ZeroDivisionError:
    print("Cannot divide by zero.")

else:
    print("Result =", result)

finally:
    print("Execution finished.")
```

This complete example demonstrates `try`, `except`, `else`, and `finally` together in one program.

1. Basic Exception Handling (try-except)

Program

```
try:
   num = 10 / 0
   print(num)
except ZeroDivisionError:
   print("Error: Cannot divide by zero.")
```

Explanation

- The try block contains code that may generate an exception.
- Dividing by zero raises a ZeroDivisionError.
- The except block catches the error and displays a message instead of terminating the program.

Output

```
Error: Cannot divide by zero.
```

---

2. Handling Multiple Exceptions

Program

```
try:
   num = int(input("Enter a number: "))
   result = 10 / num
   print("Result:", result)

except ValueError:
   print("Please enter a valid integer.")

except ZeroDivisionError:
   print("Division by zero is not allowed.")
```

Explanation

- ValueError occurs if the user enters non-numeric data.
- ZeroDivisionError occurs if the user enters 0.
- Different exceptions are handled using separate except blocks.

Example Output

```
Enter a number: 0
Division by zero is not allowed.
```

---

3. Using `else` with Exception Handling

Program

```
try:
   num = int(input("Enter a number: "))
except ValueError:
   print("Invalid input.")
else:
   print("You entered:", num)
```

Explanation

- The `else` block executes only if no exception occurs in the `try` block.
- It is useful for code that should run when everything succeeds.

Example Output

```
Enter a number: 25
You entered: 25
```

---

4. Using `finally`

Program

```
try:
   file = open("sample.txt", "r")
   content = file.read()
   print(content)

except FileNotFoundError:
   print("File not found.")

finally:
   print("Program execution completed.")
```

Explanation

- `finally` always executes, whether an exception occurs or not.
- It is commonly used for cleanup tasks such as closing files or database connections.

Output (if file doesn't exist)

```
File not found.
Program execution completed.
```

---

5. Raising an Exception Using `raise`

Program

```
age = int(input("Enter your age: "))

if age < 18:
   raise ValueError("Age must be 18 or above.")

print("You are eligible.")
```

Explanation

- The `raise` keyword is used to generate an exception manually.
- If the age is below 18, a `ValueError` is raised.

Example Output

```
Enter your age: 15
ValueError: Age must be 18 or above.
```

---

6. Custom Exception

Program

```
class InvalidAgeError(Exception):
   pass

try:
   age = int(input("Enter age: "))

   if age < 18:
      raise InvalidAgeError("You must be at least 18 years old.")

   print("Access granted.")

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

Explanation

- A custom exception class is created by inheriting from `Exception`.
- The exception is raised when the age condition is not met.
- The custom exception is then caught and handled.

Example Output

```
Enter age: 16
Custom Exception: You must be at least 18 years old.
```

---

7. Catching Any Exception

Program

```
try:
    num = int(input("Enter a number: "))
    result = 100 / num
    print(result)

except Exception as e:
    print("An error occurred:", e)
```

Explanation

- `Exception` is the base class for most built-in exceptions.
- `as e` stores the exception object so its message can be displayed.

Example Output

```
Enter a number: 0
An error occurred: division by zero
```

Summary of Exception Handling Keywords in Python

Keyword	Purpose
`try`	Contains code that may cause an exception
`except`	Handles exceptions
`else`	Executes if no exception occurs
`finally`	Always executes
`raise`	Manually raises an exception
`Exception`	Base class for most exceptions

Complete Example

```
try:
    num = int(input("Enter a number: "))
    result = 100 / num

except ValueError:
    print("Invalid number entered.")

except ZeroDivisionError:
    print("Cannot divide by zero.")

else:
    print("Result =", result)

finally:
    print("Execution finished.")
```

This complete example demonstrates `try`, `except`, `else`, and `finally` together in one program.

1. Basic Exception Handling (try-except)

Program

```
try:
   num = 10 / 0
   print(num)
except ZeroDivisionError:
   print("Error: Cannot divide by zero.")
```

Explanation

- The try block contains code that may generate an exception.
- Dividing by zero raises a ZeroDivisionError.
- The except block catches the error and displays a message instead of terminating the program.

Output

```
Error: Cannot divide by zero.
```

---

2. Handling Multiple Exceptions

Program

```
try:
   num = int(input("Enter a number: "))
   result = 10 / num
   print("Result:", result)

except ValueError:
   print("Please enter a valid integer.")

except ZeroDivisionError:
   print("Division by zero is not allowed.")
```

Explanation

- ValueError occurs if the user enters non-numeric data.
- ZeroDivisionError occurs if the user enters 0.
- Different exceptions are handled using separate except blocks.

Example Output

```
Enter a number: 0
Division by zero is not allowed.
```

---

3. Using `else` with Exception Handling

Program

```
try:
   num = int(input("Enter a number: "))
except ValueError:
   print("Invalid input.")
else:
   print("You entered:", num)
```

Explanation

- The `else` block executes only if no exception occurs in the `try` block.
- It is useful for code that should run when everything succeeds.

Example Output

```
Enter a number: 25
You entered: 25
```

---

4. Using `finally`

Program

```
try:
   file = open("sample.txt", "r")
   content = file.read()
   print(content)

except FileNotFoundError:
   print("File not found.")

finally:
   print("Program execution completed.")
```

Explanation

- `finally` always executes, whether an exception occurs or not.
- It is commonly used for cleanup tasks such as closing files or database connections.

Output (if file doesn't exist)

```
File not found.
Program execution completed.
```

---

5. Raising an Exception Using `raise`

Program

```
age = int(input("Enter your age: "))

if age < 18:
   raise ValueError("Age must be 18 or above.")

print("You are eligible.")
```

Explanation

- The `raise` keyword is used to generate an exception manually.
- If the age is below 18, a `ValueError` is raised.

Example Output

```
Enter your age: 15
ValueError: Age must be 18 or above.
```

---

6. Custom Exception

Program

```
class InvalidAgeError(Exception):
   pass

try:
   age = int(input("Enter age: "))

   if age < 18:
      raise InvalidAgeError("You must be at least 18 years old.")

   print("Access granted.")

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

Explanation

- A custom exception class is created by inheriting from `Exception`.
- The exception is raised when the age condition is not met.
- The custom exception is then caught and handled.

Example Output

```
Enter age: 16
Custom Exception: You must be at least 18 years old.
```

---

7. Catching Any Exception

Program

```
try:
    num = int(input("Enter a number: "))
    result = 100 / num
    print(result)

except Exception as e:
    print("An error occurred:", e)
```

Explanation

- `Exception` is the base class for most built-in exceptions.
- `as e` stores the exception object so its message can be displayed.

Example Output

```
Enter a number: 0
An error occurred: division by zero
```

Summary of Exception Handling Keywords in Python

Keyword	Purpose
`try`	Contains code that may cause an exception
`except`	Handles exceptions
`else`	Executes if no exception occurs
`finally`	Always executes
`raise`	Manually raises an exception
`Exception`	Base class for most exceptions

Complete Example

```
try:
    num = int(input("Enter a number: "))
    result = 100 / num

except ValueError:
    print("Invalid number entered.")

except ZeroDivisionError:
    print("Cannot divide by zero.")

else:
    print("Result =", result)

finally:
    print("Execution finished.")
```

This complete example demonstrates `try`, `except`, `else`, and `finally` together in one program.