DATA SCIENCE COURSE TUTORIAL # 32

3.19 Error and Exception Handling

What is Error Handling?

Error handling in Python is the process of managing errors that occur during program execution. Errors can stop the program from running if not handled properly. Python provides a powerful mechanism called **exception handling** to manage these errors and allow the program to continue running smoothly.

Types of Errors

1. **Syntax Errors:**
    - Occur when Python code is written incorrectly.
    - These are detected before the program runs.

    Example:

    ```
    print('Hello'   # Missing closing parenthesis
    ```

2. **Runtime Errors (Exceptions):**
    - Occur during program execution.
    - Example: dividing by zero, accessing invalid indexes, etc.

    Example:

    ```
    print(10 / 0)   # ZeroDivisionError
    ```

What is an Exception?

An **exception** is an event that occurs during the execution of a program, interrupting the normal flow of instructions. Exceptions can be handled to prevent the program from crashing.

Using try and except Blocks

Python uses `try` and `except` blocks to handle exceptions.

Syntax:

```
try:
    # Code that may cause an error
```

```
except ExceptionType:
    # Code to handle the error
```

Example:

```
try:
    num = int(input("Enter a number: "))
    print(10 / num)
except ZeroDivisionError:
    print("You cannot divide by zero!")
except ValueError:
    print("Please enter a valid number!")
```

Using Multiple except Blocks

You can handle different exceptions separately.

Example:

```
try:
    a = int(input("Enter a number: "))
    b = int(input("Enter another number: "))
    print(a / b)
except ZeroDivisionError:
    print("Division by zero is not allowed.")
except ValueError:
    print("Invalid input! Please enter numbers only.")
except Exception as e:
    print("An unexpected error occurred:", e)
```

Using else Block

The `else` block runs only if no exception occurs in the `try` block.

Example:

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

Using finally Block

The `finally` block always runs, whether an exception occurs or not. It is often used for cleanup actions (like closing files or releasing resources).

Example:

```
try:
    file = open("data.txt", "r")
    content = file.read()
except FileNotFoundError:
    print("File not found!")
finally:
    file.close()
    print("File closed.")
```

Raising Exceptions Manually

You can use the `raise` keyword to throw an exception manually.

Example:

```
def check_age(age):
    if age < 18:
        raise ValueError("Age must be 18 or above.")
    else:
        print("Access granted.")

check_age(15)
```

Output:

```
ValueError: Age must be 18 or above.
```

Using try-except-else-finally Together

You can combine all four blocks for complete error control.

Example:

```
try:
    number = int(input("Enter a number: "))
    result = 10 / number
except ZeroDivisionError:
```

```
    print("Division by zero error.")
except ValueError:
    print("Invalid input.")
else:
    print("Result is:", result)
finally:
    print("Execution completed.")
```

Common Built-in Exceptions in Python

Exception Name	Description
`ZeroDivisionError`	Raised when dividing by zero.
`ValueError`	Raised when a function receives invalid data.
`TypeError`	Raised when an operation is applied to the wrong type.
`IndexError`	Raised when a list index is out of range.
`KeyError`	Raised when a key is not found in a dictionary.
`FileNotFoundError`	Raised when a file operation fails.
`NameError`	Raised when a variable is not defined.
`AttributeError`	Raised when an invalid attribute is accessed.

Creating Custom Exceptions

You can also raise your own custom exceptions using the built-in `Exception` class with a custom message, without defining a new class.

Example:

```
def check_marks(marks):
    if marks < 0 or marks > 100:
        raise Exception("Marks must be between 0 and 100.")
    else:
        print("Valid marks.")

check_marks(120)
```

Output:

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

Summary of Error and Exception Handling

- `try` block tests for errors.
- `except` block handles errors.
- `else` block executes if no error occurs.
- `finally` block executes regardless of what happens.
- `raise` is used to manually trigger exceptions.
- Exceptions prevent programs from crashing and make them more reliable.

Key Points to Remember

- Always handle predictable errors.
- Avoid using a general `except` without specifying an error type.
- Use `raise` for custom error messages.
- Use `finally` for cleanup tasks.
- Handle exceptions properly to make programs user-friendly and stable.