UNIT III: Regular Expression and Exception Handling

1. Regular Expressions (REs)

1.1 Introduction to Regular Expressions

A **Regular Expression (RE)** is a powerful tool for searching, matching, and manipulating text based on specific patterns. Regular expressions are used for tasks like input validation, data extraction, and text transformation.

Features of Regular Expressions:

1. **Pattern Matching**: Matches specific patterns in a string.
2. **Flexible and Powerful**: Can handle complex text-processing needs.
3. **Used for Validation**: Validate inputs like email addresses, phone numbers, etc.

Example

```
import re

pattern = r"\d{3}-\d{2}-\d{4}"  # Pattern for a social security number
text = "123-45-6789"
match = re.match(pattern, text)
if match:
    print("Match found!")
else:
    print("No match.")
```

1.2 Special Symbols and Characters for REs

Regular expressions use **special symbols** to define patterns. Here's a list of commonly used symbols:

Symbol	Meaning	Example
`.`	Matches any single character	`a.b` matches `a1b`, `acb`
`^`	Matches the start of a string	`^Hello` matches `Hello World`
`$`	Matches the end of a string	`World$` matches `Hello World`
`*`	Matches 0 or more repetitions	`ab*` matches `a`, `ab`, `abbb`
`+`	Matches 1 or more repetitions	`ab+` matches `ab`, `abbb`
`?`	Matches 0 or 1 occurrence	`ab?` matches `a`, `ab`
`{n}`	Matches exactly `n` repetitions	`a{3}` matches `aaa`
`{n,}`	Matches `n` or more repetitions	`a{2,}` matches `aa`, `aaa`
`[ ]`	Matches any character inside brackets	`[abc]` matches `a`, `b`, `c`
` ` `	` ` `	Logical OR
`\d`	Matches any digit	Matches `0-9`
`\w`	Matches any word character (alphanumeric)	Matches `a-z`, `A-Z`, `0-9`
`\s`	Matches any whitespace	Matches spaces, tabs

1.3 REs and Python

Python provides the `re` module to work with regular expressions. Here are some important functions:

Common Functions:

1. **`re.match()`**: Matches a pattern at the beginning of the string.
2. **`re.search()`**: Searches for a pattern anywhere in the string.
3. **`re.findall()`**: Returns all non-overlapping matches as a list.
4. **`re.sub()`**: Replaces occurrences of a pattern with a specified string.
5. **`re.split()`**: Splits the string by occurrences of a pattern.

Example

```
import re

text = "My phone number is 123-456-7890"
pattern = r"\d{3}-\d{3}-\d{4}"

# Search for the pattern
match = re.search(pattern, text)
if match:
    print("Found:", match.group())  # Output: Found: 123-456-7890
```

---

2. Exception Handling

2.1 Introduction to Exceptions

An **exception** is an event that occurs during the execution of a program, disrupting its normal flow. Exceptions are typically errors caused by invalid operations, such as division by zero or accessing a non-existent file.

2.2 Detecting and Handling Exceptions

Python provides a **try-except** mechanism to handle exceptions gracefully.

Syntax

```
try:
    # Code that may raise an exception
except ExceptionType:
    # Code to handle the exception
```

Example

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

2.3 Exceptions as Strings

Python allows custom messages for exceptions. These messages help identify the type and cause of the exception.

Example

```
try:
    x = int("abc")
except ValueError as e:
    print(f"ValueError occurred: {e}")
```

2.4 Raising Exceptions

You can use the `raise` keyword to throw an exception manually when a specific condition is met.

Example

```
x = -5
if x < 0:
    raise ValueError("Value must be non-negative.")
```

2.5 Assertions

Assertions are a debugging aid to test assumptions in your program. If the condition in an assertion is false, Python raises an `AssertionError`.

Syntax

```
assert condition, "Error Message"
```

Example

```
x = 10
assert x > 0, "x should be greater than zero"
```

2.6 Standard Exceptions

Python provides many built-in exception classes. Here are some common ones:

Exception	Description
`IndexError`	List index out of range
`KeyError`	Dictionary key not found
`ValueError`	Invalid value for a given operation
`ZeroDivisionError`	Division by zero
`FileNotFoundError`	File or directory not found
`TypeError`	Invalid type used in operation
`NameError`	Variable or function name not found

Example of Multiple Exceptions

```
try:
    my_list = [1, 2, 3]
    print(my_list[5])  # IndexError
except IndexError:
    print("Index out of range!")
except Exception as e:
    print(f"An error occurred: {e}")
```

2.7 Advantages of Exception Handling

1. **Graceful Error Recovery**: Prevents abrupt termination of the program.
2. **Improved Debugging**: Provides meaningful error messages.
3. **Separation of Concerns**: Cleanly separates error-handling logic from normal code.
4. **Flexibility**: Can handle specific types of errors.
