9) String Manipulation

- **Understanding how to access and manipulate strings**

String Access Methods.

1. **Indexing**

- Strings are sequences of characters that can be accessed via zero-based indexing.
- Positive indexes start from the beginning (0 = first character).
- Negative indexes start from the end (-1 = last character).
- Attempting to access an out-of-range index raises an IndexError.

2. **Slicing**

- Syntax: string[start:stop:step]
- Extracts a substring from start to stop-1
- Default start is 0, default stop is length of string, default step is 1
- Omitting values: [:] creates a copy of the string
- Negative values count from the end
- Step value allows skipping characters ([::2] gets every second character)

String Manipulation Operations

1. **Concatenation**

    - Combining strings with + operator
    - * operator repeats strings ("a" * 3 produces "aaa")

2. **Immutability**

    - Strings are immutable - cannot be changed after creation.
    - Operations that appear to modify strings actually create new string objects
    - Requires creating new strings for any modifications

3. **Common Manipulation Methods**

    Case Conversion

    - lower() - converts to lowercase
    - upper() - converts to uppercase
    - title() - converts to title case
    - capitalize() - capitalizes first character
    - swapcase() - swaps cases

    Searching and Validation

    - find()/index() - locate substrings
    - startswith()/endswith() - check prefixes/suffixes

- isalpha()/isdigit()/isalnum() - character type checks

Transformation

- replace() - substring replacement
- split()/rsplit() - split into list
- join() - combine sequence into string
- format() - advanced string formatting

- **Basic operations: concatenation, repetition, string methods (upper(), lower(), etc.).**

1. **String Concatenation**

   Definition: Combining two or more strings into a single string

   Methods:

   - Using the + operator
   - Using the join() method (more efficient for multiple strings)
   - Using f-strings (Python 3.6+) or format() for embedded expressions

2. **String Repetition**

Definition: Creating new strings by repeating an existing string

Method:

- Using the * operator

String Methods :

1. Case Conversion Methods

    - upper() → Converts all characters to uppercase
    - lower() → Converts all characters to lowercase
    - title() → Converts first letter of each word to uppercase
    - capitalize() → Converts first character to uppercase, rest lowercase
    - swapcase() → Swaps uppercase to lowercase and vice versa
    - casefold() → Aggressive lowercase conversion (for case-insensitive comparisons)

2. Search & Validation Methods

    - find(sub) → Returns lowest index where substring is found (else -1)
    - index(sub) → Like find() but raises ValueError if not found
    - rfind(sub) → Highest index where substring is found
    - count(sub) → Counts occurrences of substring
    - startswith(prefix) → Checks if string starts with prefix
    - endswith(suffix) → Checks if string ends with suffix
    - isalnum() → Checks if all characters are alphanumeric
    - isalpha() → Checks if all characters are alphabetic
    - isdigit() → Checks if all characters are digits

- isnumeric() → Checks if all characters are numeric (including Unicode)
- isspace() → Checks if all characters are whitespace
- islower() → Checks if all characters are lowercase
- isupper() → Checks if all characters are uppercase
- istitle() → Checks if string follows title case rules

3. Formatting & Cleaning Methods

- strip([chars]) → Removes leading/trailing characters (default: whitespace)
- lstrip([chars]) → Removes leading characters
- rstrip([chars]) → Removes trailing characters
- center(width[, fillchar]) → Centers string in given width
- ljust(width[, fillchar]) → Left-justifies string in given width
- rjust(width[, fillchar]) → Right-justifies string in given width
- zfill(width) → Zero-pads string to given width
- expandtabs(tabsize) → Replaces tabs with spaces

4. Transformation Methods

- replace(old, new[, count]) → Replaces occurrences of substring
- split([sep[, maxsplit]]) → Splits string into list using separator
- rsplit([sep[, maxsplit]]) → Splits from right side
- splitlines([keepends]) → Splits at line breaks
- join(iterable) → Joins elements of iterable using string as separator

- encode(encoding, errors) → Returns encoded version of string
- translate(table) → Performs character-level translations

- **String slicing.**

String slicing allows you to extract portions of strings by specifying start, stop, and step values. Here's a complete explanation:

Basic Syntax

string[start:stop:step]

Key Components

1. Start Index (inclusive) - Where slicing begins (default: 0
2. Stop Index (exclusive) - Where slicing ends (default: end of string)
3. Step - Interval between characters (default: 1)

Common Use Cases

1. Extracting substrings
2. Reversing strings
3. Skipping characters
4. Processing fixed-width data formats
5. Removing prefixes/suffixes

String slicing is:

- Zero-based indexing
- Does not modify original string (strings are immutable)
- Returns a new string object