# 9. String Manipulation

## ❖Understanding how to access and manipulate strings.

In Python, strings are sequences of characters enclosed in single (') or double (") quotes. They are immutable, meaning their content cannot be changed after creation. However, Python provides a rich set of tools to access, manipulate, and format strings effectively.

| Method | Description | Example |
|---|---|---|
| `upper()` | Converts all characters to uppercase | `"hello".upper()` → `'HELLO'` |
| `lower()` | Converts all characters to lowercase | `"HELLO".lower()` → `'hello'` |
| `capitalize()` | Capitalizes the first character | `"hello".capitalize()` → `'Hello'` |
| `title()` | Capitalizes the first character of each word | `"hello world".title()` → `'Hello World'` |
| `strip()` | Removes leading and trailing whitespace | `" hello ".strip()` → `'hello'` |
| `replace(old, new)` | Replaces occurrences of a substring | `"hello".replace('e', 'a')` → `'hallo'` |
| `split(sep)` | Splits the string into a list at each separator | `"a,b,c".split(',')` → `['a', 'b', 'c']` |
| `join(iterable)` | Joins elements of an iterable into a string | `','.join(['a', 'b', 'c'])` → `'a,b,c'` |

## ❖Basic operations: concatenation, repetition, string methods (upper(), lower(), etc.).

In Python, strings are immutable sequences of characters that support various operations for manipulation and transformation. Here's an overview of fundamental string operations: concatenation, repetition, and common string methods.

### String Concatenation :

Concatenation combines two or more strings into a single string. Python offers several ways to concatenate strings:

### 1. Using the + Operator

This is the most straightforward method:

```
s1 = "Hello"
s2 = "World"
s3 = s1 + " " + s2
print(s3)  # Output: Hello World
```

### 2. Using += Operator

Appends a string to an existing string:

```
text = "Hello"
```

```
text += " World"
print(text)  # Output: Hello World
```

## 3. Using join() Method

Efficient for concatenating a list of strings:

```
words = ["Hello", "World"]
sentence = " ".join(words)
print(sentence)  # Output: Hello World
```

## 4. Using f-strings (Python 3.6+)

Provides a readable and concise way to embed expressions inside string literals:

```
name = "Alice"
greeting = f"Hello, {name}!"
print(greeting)  # Output: Hello, Alice!
```

## String Repetition :

Repeating a string multiple times can be achieved using the * operator:

```
s = "Hi "
print(s * 3)  # Output: Hi Hi Hi
```

## String Methods :

Python provides a rich set of string methods for various operations:

- **upper()**: Converts all characters to uppercase.

```
s = "hello"
print(s.upper())  # Output: HELLO
```

- **lower()**: Converts all characters to lowercase.

```
s = "HELLO"
print(s.lower())  # Output: hello
```

- **capitalize()**: Capitalizes the first character and lowercases the rest.

```
s = "hello"
print(s.capitalize())  # Output: Hello
```

- **title()**: Capitalizes the first character of each word.

```
s = "hello world"
print(s.title())  # Output: Hello World
```

## ❖ String slicing.

In Python, string slicing is a powerful technique that allows you to extract substrings from a given string using a specific syntax. This operation is particularly useful in text processing, data manipulation, and various algorithms.

**Syntax of String Slicing :**

substring = string[start:end:step]

- **start**: The index where the slice begins (inclusive). If omitted, defaults to 0.
- **end**: The index where the slice ends (exclusive). If omitted, defaults to the length of the string.
- **step**: The interval between each index in the slice. If omitted, defaults to 1.