Working with Strings in Python

1. Introduction:

Strings are sequences of characters enclosed within quotes. They are one of the most essential and frequently used data types in Python, suitable for handling textual data like names, messages, identifiers, etc.

Python supports single (' '), double (" "), and triple quotes ("' "' or) for defining strings.

Example:

```
s1 = 'Hello'
s2 = "World"
s3 = "'This is a
multiline string"'
```

2. Immutability of Strings:

Strings in Python are immutable, meaning once a string is created, it cannot be changed. Any operation that modifies a string will return a new string object.

Example:

```
s = "Hello"
s[0] = "h"   # This will raise a TypeError
```

Instead:

```
s = "h" + s[1:]  # Creates a new string
```

3. String Concatenation and Repetition:

- Concatenation: Joining strings using `+`
- Repetition: Repeating a string using `*`

Example:

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

```
print(a * 3)              # Output: HelloHelloHello
```

4. Accessing Characters in Strings:

You can access individual characters using indexing and slicing.

Indexing:

```
text = "Python"
print(text[0])    # Output: P
print(text[-1])   # Output: n
```

Slicing:

```
print(text[0:3])  # Output: Pyt
print(text[:4])   # Output: Pyth
print(text[2:])   # Output: thon
```

5. Useful String Methods:

Python provides several built-in string methods:

- upper(), lower(), title()
- strip(), lstrip(), rstrip()
- find(), index()
- replace(old, new)
- split(delimiter), join(iterable)
- isdigit(), isalpha(), isalnum()

Example:

```
text = "  Python Programming  "
print(text.strip())         # Removes leading/trailing spaces
print(text.upper())         # Converts to uppercase
print(text.replace("Python", "Java"))
```

6. String Formatting:

Python supports multiple ways to format strings:

- f-strings (Python 3.6+)
- format() method
- %-formatting (older style)

Example:

```
name = "Alice"
age = 24
print(f"My name is {name} and I am {age} years old.")
print("My name is {} and I am {} years old.".format(name, age))
```

7. Escape Characters:

Escape characters are used to include special characters in a string.

Examples:

\n New line
\t Tab space
\" Double quote
\\ Backslash

```
text = "Hello\nWorld"
print(text)
```

8. String Comparisons:

Strings can be compared using comparison operators (==, !=, <, >, etc.)

They compare lexicographically (dictionary order).

Example:

```
print("apple" < "banana")  # Output: True
```

9. Looping Through Strings:

You can iterate over each character in a string using loops.

Example:

```
for char in "Python":
    print(char)
```

10. Practical Use Cases:

- Validating user input (e.g., checking if email contains '@')
- Cleaning and formatting text for display
- Parsing logs or file data
- Creating dynamic messages in web apps

11. Common Mistakes:

- Forgetting that strings are immutable
- Using + for concatenation in loops (inefficient for large data)
- Misusing escape characters

12. Practice Questions:

1. Write a program to take a user's name and greet them with a custom message.
2. Check if a string is a palindrome.
3. Count the number of vowels in a given string.
4. Replace all spaces in a sentence with underscores.
5. Split a sentence into words and print each word on a new line.
6. Ask the user for an email and validate if it contains '@' and '.'

13. Summary:

Strings are powerful and versatile in Python. With a rich set of built-in methods and flexible syntax, string operations are easy to perform and are used across almost every Python applicationfrom web development to data analysis.