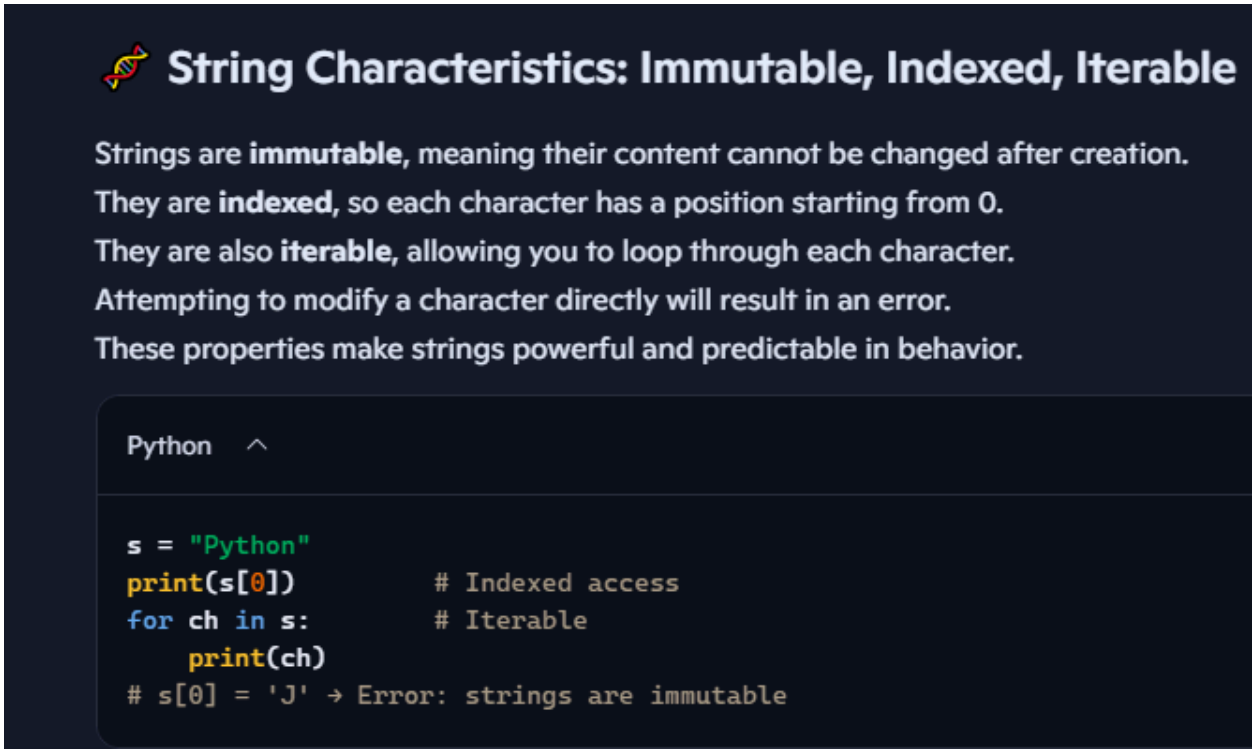
🧵 What is a String?

A string in Python is a sequence of characters enclosed within single (' '), double (" "), or triple quotes (''' '''). Strings are used to represent textual data such as words, sentences, or even entire paragraphs. They are objects of the str class and support various operations and methods. Strings can include letters, digits, symbols, and whitespace. Triple quotes are especially useful for multi-line strings or documentation (docstrings).

```
# Example: Defining strings
single = 'Hello'
double = "World"
multi_line = '''This is
a multi-line
string.'''
print(single, double)
print(multi_line)
```



🔍 Accessing String Elements: Indexing & Slicing

Indexing allows access to individual characters using their position.
Slicing extracts a portion of the string using `start:end` syntax.
Negative indexing starts from the end of the string.
Slicing does not modify the original string—it returns a new one.
This is useful for extracting substrings or reversing strings.

Python

```python
s = "Programming"
print(s[0])       # 'P'
print(s[3:6])     # 'gra'
print(s[-1])      # 'g'
print(s[:4])      # 'Prog'
print(s[::-1])    # Reverse string
```

🔁 Looping Through Strings

Since strings are iterable, you can use loops to process each character.
This is useful for tasks like counting vowels, formatting, or encryption.
You can use `for` or `while` loops depending on the logic.
Each iteration gives access to one character at a time.
Looping is often combined with conditions for filtering or transformation.

Python

```python
s = "Loop"
for ch in s:
    print(ch)
# Output: L o o p (each on a new line)
```

🔎 Searching in String: `find()`, `index()`, `count()`

- `find()` returns the first index of a substring or -1 if not found.
- `index()` is similar but raises an error if the substring is missing.
- `count()` returns how many times a substring appears.

These methods help in locating and analyzing string content.
They are case-sensitive and work only on exact matches.

Python

```python
s = "banana"
print(s.find("a"))     # 1
print(s.index("n"))    # 2
print(s.count("a"))    # 3
```

⚖️ String Comparison

Strings can be compared using relational operators like `==`, `!=`, `<`, `>`.
Comparison is based on Unicode values of characters.
It is case-sensitive unless converted using methods like `lower()`.
Useful in sorting, filtering, and conditional logic.
Lexicographical order determines which string is "greater".

Python

```python
a = "apple"
b = "banana"
print(a == b)       # False
print(a < b)        # True
print(a > b)        # False
```

🛠️ String Methods

Python provides many built-in methods to manipulate strings. These include case conversion (upper(), lower()), formatting (title(), capitalize()), and replacement (replace()). Splitting and joining strings is done using split() and join(). Validation methods like isalpha() and isdigit() check content type. Prefix/suffix checks are done using startswith() and endswith().

upper()

- Converts all lowercase letters in a string to uppercase.
- Non-alphabetic characters remain unchanged.
- Useful for standardizing user input or formatting text.
- Does not modify the original string—returns a new one.
- Commonly used in comparisons or display formatting.

Python

```python
s = "hello world"
print(s.upper())  # Output: 'HELLO WORLD'
```

lower()

- Converts all uppercase letters in a string to lowercase.
- Non-alphabetic characters are unaffected.
- Helps in case-insensitive comparisons.
- Returns a new string; original remains unchanged.
- Often used in search and filtering operations.

Python

```python
s = "HELLO World"
print(s.lower())  # Output: 'hello world'
```

`title()`

- Capitalizes the first letter of each word in the string.
- Words are separated by whitespace or punctuation.
- Useful for formatting names, titles, or headings.
- Returns a new string with title case.
- Does not affect acronyms or special cases.

Python

```python
s = "python programming language"
print(s.title())  # Output: 'Python Programming Language'
```

`swapcase()`

- Swaps the case of each character: uppercase becomes lowercase and vice versa.
- Useful for toggling case in user input or formatting.
- Non-alphabetic characters are unchanged.
- Returns a new string.
- Can be used for playful or stylistic text transformations.

Python

```python
s = "PyThOn"
print(s.swapcase())  # Output: 'pYtHoN'
```

`capitalize()`

- Converts the first character to uppercase and the rest to lowercase.
- Ideal for formatting single-word titles or names.
- Only affects the first character of the string.
- Returns a new string.
- Useful in UI or form validation.

Python

```python
s = "python"
print(s.capitalize())  # Output: 'Python'
```

`replace(old, new)`

- Replaces all occurrences of a substring with another.
- Can be used to clean or modify text.
- Returns a new string with replacements.
- Case-sensitive by default.
- You can also specify a count to limit replacements.

Python

```python
s = "banana"
print(s.replace("a", "o"))  # Output: 'bonono'
```

✂️ `split(separator)`

- Splits a string into a list using the specified separator.
- Default separator is whitespace.
- Useful for parsing CSV, logs, or user input.
- Returns a list of substrings.
- Does not modify the original string.

Python

```python
s = "apple,banana,cherry"
print(s.split(","))  # Output: ['apple', 'banana', 'cherry']
```

🔗 `join(iterable)`

- Joins elements of an iterable (like a list) into a single string.
- The string used to call `join()` becomes the separator.
- Useful for formatting output or rebuilding strings.
- All elements must be strings.
- Returns a new string.

Python

```python
words = ['Python', 'is', 'fun']
print(" ".join(words))  # Output: 'Python is fun'
```

`startswith(prefix)`

- Checks if the string starts with the specified prefix.
- Returns `True` or `False`.
- Case-sensitive by default.
- Useful in filtering or validation.
- Can specify start and end positions.

Python

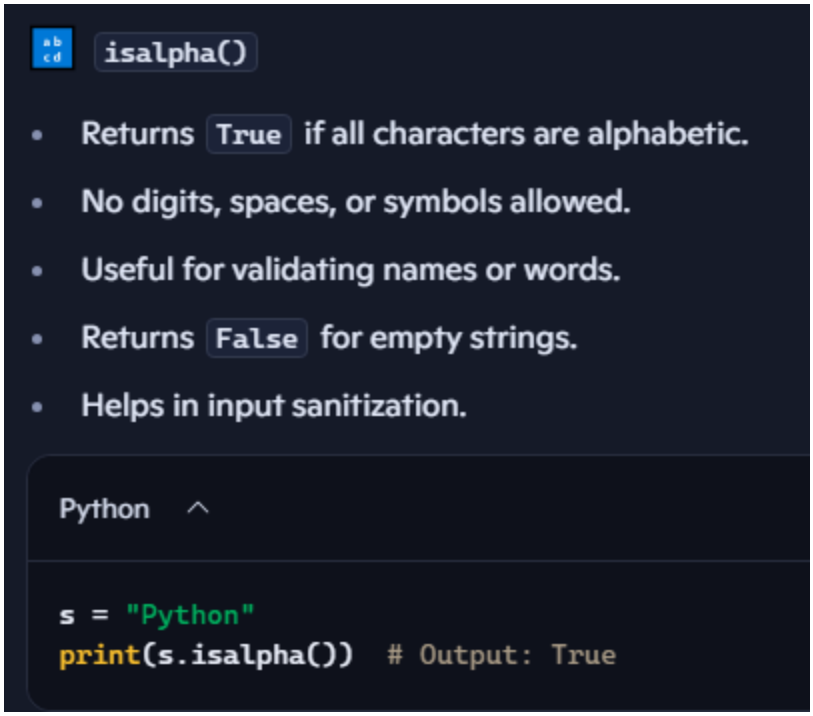
```python
s = "hello world"
print(s.startswith("hello"))  # Output: True
```

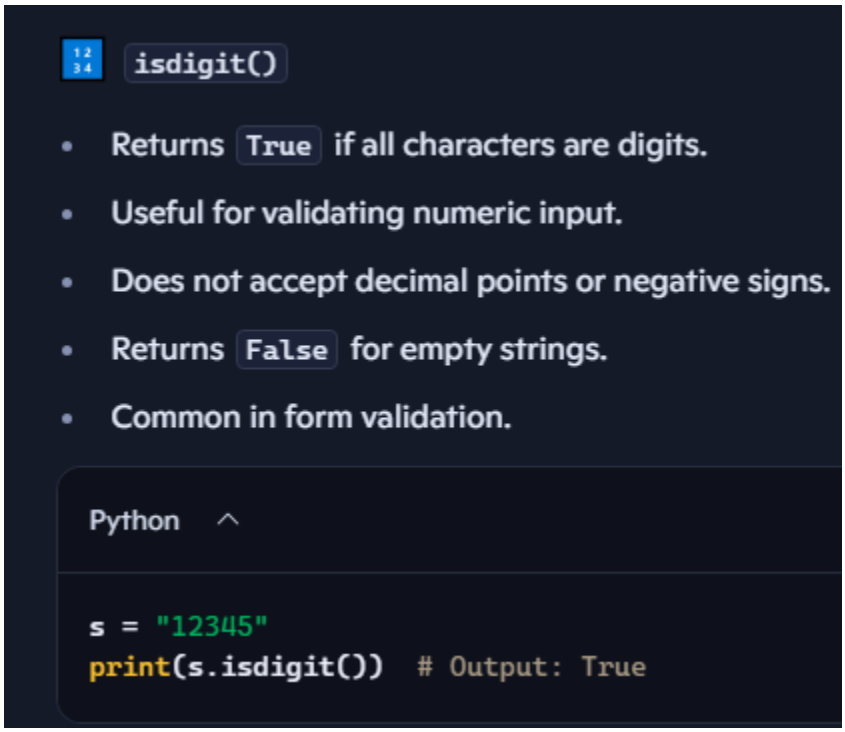
`endswith(suffix)`

- Checks if the string ends with the specified suffix.
- Returns `True` or `False`.
- Case-sensitive.
- Useful for file extensions or URL checks.
- Can specify start and end positions.

Python

```python
s = "document.pdf"
print(s.endswith(".pdf"))  # Output: True
```

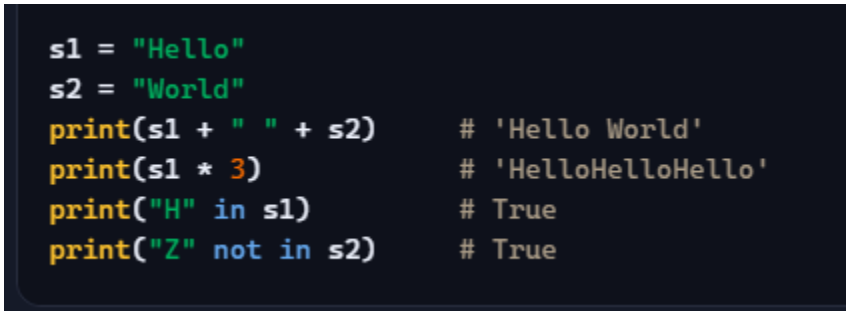




➕ **String Operations: Concatenation, Repetition, Membership**

- **Concatenation (+)** joins two strings together.
- **Repetition (*)** repeats a string multiple times.

- **Membership (**in**,** not in**)** checks if a substring exists. These operations are intuitive and commonly used in string handling. They help build dynamic messages and validate input.



```
s1 = "Hello"
s2 = "World"
print(s1 + " " + s2)      # 'Hello World'
print(s1 * 3)             # 'HelloHelloHello'
print("H" in s1)          # True
print("Z" not in s2)      # True
```