Strings

A string is text inside quotes — learn indexing, slicing, useful methods, and f-strings for building messages.

Introduction

A **string** is text. Any word, sentence, or even a single letter, written inside quotes, is a string in Python. Your name is a string. A greeting is a string. Even a number inside quotes, like `"100"`, is a string — Python treats it as text, not as a number you can add.

Strings are everywhere in real programs: names on forms, messages to users, file names, and search words. So Python gives you many tools for working with them. You can take one letter out of a string, cut out a part of it, change it to capital letters, remove extra spaces, and join it with other values to make a full sentence.

In this lesson you will learn the four skills that cover most string work: reading single letters, cutting slices, cleaning and changing text with methods, and building messages with f-strings.

(Roman Urdu: String ka matlab hai text — jo bhi lafz ya jumla quotes ke andar likha ho, woh string hai, chahay woh number jaisa hi kyun na lagay.)

Two ideas cause most string confusion. First, a string is not the thing it looks like: `"100"` is text, not a number, even though it has digits in it. Second, string methods never change the string you started with — they hand you a new one. If you forget to catch that new string, your work silently disappears.

Key points

- **A string uses single or double quotes.** `"Ali"` and `'Ali'` are the same. Pick one style and stay consistent.
- **Indexing reads one letter.** Counting starts at 0, not 1. So in `"Ali"`, index 0 is `A`, index 1 is `l`, and index 2 is `i`.
- **Negative indexing counts from the end.** Index -1 is the last letter, -2 is the second-last, and so on.
- **Slicing cuts out a part.** `text[0:3]` takes letters from index 0 up to — but not including — index 3.
- **Strings cannot be changed in place.** You cannot do `text[0] = "B"`; instead you build a new string. This is called immutability.
- **Methods are built-in string tools.** `.upper()` makes capitals, `.lower()` makes small letters, `.strip()` removes spaces from both ends, and `.replace()` swaps one part for another.
- **F-strings build messages.** Writing `f"Hello {name}"` puts the value of `name` inside the sentence.
- **`len()` counts the length.** `len("Ali")` is 3, because there are three letters.
- **The `+` operator joins strings.** `"Good " + "morning"` gives `"Good morning"`.
- **`.title()` capitalises each word.** `"ali raza".title()` gives `"Ali Raza"` — handy for cleaning up names typed in small letters.

- `.split()` **breaks a string into a list of words.** `"I love Python".split()` gives `['I', 'love', 'Python']`, splitting on spaces by default.

Examples

Example 1 — Indexing and slicing

```
word = "Python"
print(word[0])
print(word[1])
print(word[-1])
print(word[0:3])
print(word[2:])
print(len(word))
```

What you will get:

```
P
y
n
Pyt
thon
6
```

`word[0:3]` takes indexes 0, 1 and 2 — it stops before index 3. `word[2:]` takes everything from index 2 to the end.

Example 2 — String methods

```
message = "  hello world  "
print(message.strip())
print(message.strip().upper())
print(message.strip().replace("world", "Python"))
print("hello".upper())
print("HELLO".lower())
```

What you will get:

```
hello world
HELLO WORLD
hello Python
HELLO
hello
```

`strip()` only removes spaces at the two ends, not spaces in the middle. Methods do not change the original string — they return a new one, so you must print or store the result.

Example 3 — F-strings for real messages

```
name = "Ayesha"
marks = 85
print(f"Hello {name}, you scored {marks} marks.")
print(f"Next year {name} wants {marks + 5} marks.")
```

What you will get:

```
Hello Ayesha, you scored 85 marks.
Next year Ayesha wants 90 marks.
```

Inside the curly brackets you can put a variable, or even a small calculation like `marks + 5`. The `f` before the opening quote is what turns an ordinary string into an f-string — without it, the brackets print literally.

Example 4 — Counting, splitting and joining

Three more methods cover a lot of everyday string work: counting how often a letter appears, splitting a sentence into words, and joining pieces with a separator:

```
sentence = "I love Python programming"
print(sentence.count("o"))
print(sentence.split())
print("-".join(["2026", "10", "27"]))
print(sentence.startswith("I"))
```

What you will get:

```
3
['I', 'love', 'Python', 'programming']
2026-10-27
True
```

The letter `o` appears three times: once in "love", once in "Python", and once in "programming". `join()` is called on the separator (`"-"`), not on the list — that order surprises beginners, so note it well.

Example 5 — Slicing tricks: first part, last part, reversed

Slicing with an empty start or end means "from the beginning" or "to the end". A step of -1 walks backwards, which reverses the string:

```
full_name = "Ali Raza"
print(full_name[:3])
print(full_name[4:])
print(full_name[::-1])
print(full_name.upper()[0:3])
```

What you will get:

```
Ali
Raza
azaR ilA
ALI
```

`full_name[:3]` is the same as `full_name[0:3]`. The last line chains two steps: first make everything capital, then take the first three letters. Chaining works because each method returns a new string for the next step to use.

Example 6 — Real-world case: cleaning a name typed into a form

People type names with extra spaces and small letters: `" ayesha khan "`. Before saving it, a real program cleans it up — strip the spaces, capitalise the words, and build a username from it:

```
raw_name = "  ayesha khan  "
clean_name = raw_name.strip().title()
print(f"Welcome, {clean_name}!")
print(f"Username: {clean_name.lower().replace(' ', '_')}")
```

What you will get:

```
Welcome, Ayesha Khan!
Username: ayesha_khan
```

This is the same pattern behind sign-up forms everywhere: take messy human input, clean it with string methods, and only then store or show it.

Quick recap

- A string is text inside quotes.
- Indexing starts at 0; negative indexes count from the end.
- Slicing `[start:end]` stops before the end index.
- `.upper()`, `.lower()`, `.strip()` and `.replace()` return new strings.
- F-strings join values into messages with `{ }`.

- `.title()` capitalises each word; `.split()` breaks a string into a list of words.
- Reversing uses a slice with a step of -1: `text[::-1]`.

Common mistakes

1. **Counting from 1 instead of 0** — thinking `word[1]` is the first letter. Fix: remember the first letter is always index 0.
2. **Expecting the slice end to be included** — `word[0:3]` does not include index 3. Fix: to include a letter, set the end one higher than its index.
3. **Forgetting the `f` in an f-string** — `"Hello {name}"` prints the brackets literally. Fix: write `f"Hello {name}"`.
4. **Thinking a method changes the original** — `message.upper()` alone changes nothing unless you store or print the result. Fix: write `message = message.upper()` or `print(message.upper())`.
5. **Trying to change one letter directly** — `word[0] = "B"` gives an error. Fix: build a new string, for example `word = "B" + word[1:]`.

1. **Counting spaces in `len()` by surprise** — `len("Ali ")` is 4, not 3, because the trailing space counts as a character. Fix: strip first, then measure: `len("Ali ".strip())`.
2. **Adding a string and a number** — `"Marks: " + 85` gives a TypeError, because `+` cannot join text and a number. Fix: convert first with `str(85)`, or use an f-string.

Practice

1. Store your full name in a variable. Print its first letter, its last letter, and its first three letters using slicing.
2. Take the string `" I Love Python "`. Print it stripped, then in all capitals, then with "Python" replaced by your favourite language.
3. Make two variables, `city` and `country`, and print one sentence with an f-string: "I live in [city], [country]."
4. Try `len()` on your name and on an empty string `""`. Write down both results and why the empty string has length 0.
5. Store your name, print it reversed with a slice, and print how many times the letter "a" appears in it (use `.count()`).
6. Take the sentence `"python is easy to learn"`. Print it with `.title()`, then split it into a list of words and print the list.
7. Build a username from your own name: strip spaces, make it small letters, and replace spaces with `_`, like Example 6. Print the result with an f-string.

Practice quiz

1. How do you make a string in Python?

- With round brackets only
- With single or double quotes around text

- With square brackets only
- With a # sign

Answer: With single or double quotes around text

2. What does "hello".upper() give?

- hello
- HELLO
- Hello
- hELLO

Answer: HELLO

3. What is an f-string used for?

- Making text bold
- Putting variables inside text easily
- Deleting text
- Hiding text

Answer: Putting variables inside text easily

Watch the video lesson: https://youtu.be/2YdHqt7Cx7g

Read online: https://scopetreasure.com/notes/py-strings · More free tools and notes at https://scopetreasure.com