CHAPTER 7

Strings in Python

Indexing • Slicing • String Methods • Formatting

Programming Principles

Chapter purpose
Learn how to inspect, extract, search, modify, split, combine, and format text in Python.

1. Learning Objectives

By the end of this chapter, you should be able to:

- Create strings using single, double, and triple quotes.
- Use common escape sequences such as newline, tab, backslash, single quote, and double quote.
- Combine strings with + and repeat them with *.
- Use len() and the “in” operator.
- Use positive and negative indexing.
- Use slicing with start, stop, and step.
- Reverse strings with [::-1].
- Use important methods: upper(), lower(), title(), capitalize(), swapcase(), strip(), find(), index(), rfind(), rindex(), replace(), count(), split(), and join().
- Understand that strings are immutable and methods normally return new strings.
- Use format() and f-strings, including alignment and decimal precision.
- Combine strings with loops, conditions, and functions to solve complete problems.

2. The Big Picture

A string is a sequence of characters. Because it is a sequence, Python lets us access individual characters and groups of characters by position. String methods provide convenient operations for searching, cleaning, changing, splitting, joining, and formatting text.

Task	Python	Example
Length	len(s)	len("Python") → 6
Membership	x in s	"py" in "Python" → False
One character	s[i]	"Python"[0] → P
Substring	s[a:b]	"Python"[1:4] → yth
Search	s.find(x)	"banana".find("na") → 2
Replace	s.replace(a,b)	"cat".replace("c","b") → bat
Split	s.split()	"a b c".split() → ["a","b","c"]
Join	sep.join(items)	"-".join(["a","b"]) → a-b
Format	f"{x}"	f"Score: {score}"

Core mental model
STRING → indexing selects one character → slicing selects a range → methods perform useful operations → formatting communicates the result.

3. Creating Strings

3.1 Single and double quotes

Single and double quotes are functionally equivalent for ordinary strings.

```
x = "a string"
y = 'a string'
print(x == y)       # True
```

3.2 Multi-line strings

```
message = """Line one
Line two
Line three"""
print(message)
```

3.3 Escape sequences

Sequence	Meaning
\n	new line
\t	tab
\\	backslash
\'	single quote
\"	double quote

3.4 Concatenation and repetition

```
x = "Mission"
y = "Control"
print(x + " " + y)    # Mission Control
print("Go! " * 3)         # Go! Go! Go!
```

4. Useful String Operations

4.1 len()

len(s) returns the number of characters, including spaces and punctuation.

```
print(len("Python"))      # 6
print(len("Hi there"))     # 8
```

4.2 Membership with in

```
s = "mission control"
print("mission" in s)    # True
print("control" in s)    # True
print("rocket" in s)     # False
print("ssi" in s)        # True
```

Remember

The in operator produces True or False.

5. String Indexing

Python uses zero-based indexing: the first character is at index 0. Negative indexes count from the end, with -1 being the last character.

```
s = "abcde"
print(s[0])    # a
print(s[1])    # b
print(s[4])    # e
print(s[-1])   # e
print(s[-2])   # d
```

Index	0	1	2	3	4	-5	-4	-3	-2	-1
Value	a	b	c	d	e	a	b	c	d	e

5.1 Index errors

```
s = "abc"
print(s[3])   # IndexError
```

For a 3-character string, valid indexes are 0, 1, 2, -3, -2, and -1.

6. String Slicing

The general form is s[start:stop:step]. Start is included; stop is excluded.

```
s = "abcde"
print(s[0:3])    # abc
print(s[:3])     # abc
print(s[2:])     # cde
print(s[-3:])    # cde
print(s[1:4])    # bcd
```

6.1 The stop value is excluded

```
s = "abcdef"
print(s[1:4])   # bcd
# indexes 1, 2 and 3 are included; index 4 is not
```

6.2 Step and reverse

```
s = "abcde"
print(s[::2])     # ace
print(s[1::2])    # bd
print(s[::-1])    # edcba
```

Easy rule

start = where to begin; stop = where to stop (not included); step = how far to move each time.

6.3 Common slicing patterns

Expression	Meaning
s[:n]	first n characters
s[n:]	from index n to the end

s[-n:]	last n characters
s[:-n]	everything except the last n characters
s[::2]	every second character
s[::-1]	reverse the string

7. Strings Are Immutable

A string cannot be changed character-by-character after it has been created. Operations and methods normally produce a new string.

```
s = "cat"
# s[0] = "b"        # TypeError
s = "b" + s[1:]
print(s)            # bat
```

Common mistake
s.upper() computes an uppercase string but does not replace s. Use s = s.upper() if you want to store the result.

8. String Methods

Methods are called with dot notation, such as s.lower().

8.1 Changing case

```
s = "Hello Python"
print(s.upper())        # HELLO PYTHON
print(s.lower())        # hello python
print(s.title())        # Hello Python
print(s.capitalize())   # Hello python
print(s.swapcase())     # hELLO pYTHON
```

8.2 Removing spaces

```
s = "   mission   "
print(s.strip())    # "mission"
print(s.lstrip())   # "mission   "
print(s.rstrip())   # "   mission"
```

8.3 Alignment

```
s = "MARS"
print(s.center(10))
print(s.ljust(10))
print(s.rjust(10))
```

8.4 Searching

find() returns the first matching index, or -1 if absent. index() is similar, but raises ValueError if absent. rfind() and rindex() search from the right.

```
s = "banana"
print(s.find("na"))        # 2
```

```
print(s.find("xy"))        # -1
print(s.index("na"))       # 2
print(s.rfind("na"))       # 4
print(s.rindex("na"))      # 4
```

8.5 replace()

```
s = "the quick brown fox"
print(s.replace("o", "--"))
# the quick br--wn f--x
```

8.6 count()

```
s = "banana"
print(s.count("a"))    # 3
print(s.count("na"))   # 2
```

8.7 split()

split() breaks a string into a list. With no argument, whitespace is used.

```
s = "one two three"
words = s.split()
print(words)    # ["one", "two", "three"]

s = "red,green,blue"
print(s.split(","))    # ["red", "green", "blue"]
```

8.8 join()

join() combines strings using a separator.

```
words = ["one", "two", "three"]
print(" ".join(words))
print("--".join(["1", "2", "3"]))
```

Very important
join belongs to the separator: " ".join(words), not words.join(" ").

9. String Formatting

For modern Python, f-strings are usually the clearest choice (format() vs. f-strings).

9.1 format()

```
name = "Alex"
score = 87
print("Student: {}, Score: {}".format(name, score))
print("{1} scored {0}".format(87, "Alex"))
```

9.2 f-strings

```
name = "Alex"
score = 87
print(f"Student: {name}, Score: {score}")
speed = 28000
hours = 3
print(f"Distance: {speed * hours} km")
```

9.3 Number and alignment formatting

```
temperature = 37.7777
print(f"{temperature:.2f}")       # 37.78

name = "MARS"
print(f"{name:<10}")
print(f"{name:>10}")
print(f"{name:^10}")
```

Format	Meaning	Example
:<10	left aligned, width 10	f"{x:<10}"
:>10	right aligned, width 10	f"{x:>10}"
:^10	centered, width 10	f"{x:^10}"
:.2f	2 decimal places	f"{x:.2f}"
:10.2f	width 10, 2 decimal places	f"{x:10.2f}"

10. Worked Examples

Example 1 — Last three characters

Read a string and print its last three characters.

```
text = input("Enter a string: ")
print(text[-3:])
```

Example 2 — Count a character

Count the number of letter a characters.

```
text = input("Enter a string: ")
print(text.count("a"))
```

Example 3 — Clean and format a name

Remove outer spaces, capitalise each word, then print a message.

```
name = input("Enter your name: ")
name = name.strip().title()
print(f"Welcome, {name}!")
```

Example 4 — Reverse a string

Print a string backwards.

```
text = input("Enter a string: ")
print(text[::-1])
```

Example 5 — Search safely

Find where Python first occurs.

```
text = input("Enter a string: ")
pos = text.find("Python")
if pos == -1:
    print("Python was not found.")
else:
    print(f"Python starts at index {pos}.")
```

11. Sample Questions and Answers

Problem 1 — Longest string until A

Problem: Write a program that prompts for and reads strings until a string that starts with the letter "A" is entered (inclusive), then prints the longest string that was entered. Example run:

```
Enter a string: In realms where dreams and magic meet,

Enter a string: Where wonders never fail,

Enter a string: There lies a world of fantasy,

Enter a string: the land of fairytale.

Enter a string: With castles tall and dragons fierce,

Enter a string: And forests dark and deep,

Longest was: In realms where dreams and magic meet,
```

Testing: Test your code for the example input shown above.

```
longest = ""

while True:
    text = input("Enter a string: ")

    if len(text) > len(longest):
        longest = text

    if text.startswith("A"):
        break

print("Longest was:", longest)
```

Key ideas: startswith() tests the beginning; len() compares lengths; break stops the loop.

Problem 2 — Replace digits with underscores

Problem: Write a program with a **function** that converts all numerical digits in a string to the underline character. Example run:

```
Input a string? Australia has 59,736 km coastal line.
Output: Australia has __,___ km coastal line.

Input a string? 10 x 10 = 20
Output: __ x __ = __

Input a string? 10 4 2
Output: __ _ _
```

```
def replace_digits(text):
    result = ""

    for ch in text:
        if ch.isdigit():
            result += "_"
```

```
        else:
            result += ch

    return result


text = input("Input a string: ")
print("Output:", replace_digits(text))
```

Key ideas: process one character at a time, use isdigit(), build a new string, and return it.

Problem 3 — Celsius/Fahrenheit

Problem: Write a function that converts degrees in Celsius to degrees in Fahrenheit, and function that converts degrees in Fahrenheit to degrees in Celsius. The main program should accept user input, call the relevant function, and print the converted degree in Celsius or Fahrenheit.

The following formulas can be used for the degree conversion:

$$°F = (°C \times 1.8) + 32 \text{ or } °F = °C \times \frac{9}{5} + 32$$

$$°C = (°F - 32) \times \frac{5}{9}$$

Example input and output:

Enter the degrees and the unit (C/F): 100F 100F is 37.78C   Enter the degrees and the unit (C/F): 35C 35C is 95F

```
def celsius_to_fahrenheit(c):
    return c * 9 / 5 + 32


def fahrenheit_to_celsius(f):
    return (f - 32) * 5 / 9


text = input("Enter the degrees and the unit (C/F): ").strip()

unit = text[-1].upper()
degrees = float(text[:-1])

if unit == "C":
    result = celsius_to_fahrenheit(degrees)
    print(f"{degrees:g}C is {result:.2f}F")
elif unit == "F":
    result = fahrenheit_to_celsius(degrees)
    print(f"{degrees:g}F is {result:.2f}C")
```

```
else:
    print("Invalid unit.")
```

String connection: text[-1] extracts the unit; text[:-1] extracts everything before the unit.

Supplied examples
100F → 37.78C
35C → 95F
20F → -6.67C

Problem 4 — Lowercase and sort words

Problem: Write a program that reads strings (without digits or symbols in the string) typed by the user until an empty string is entered. For each string, convert all words to lowercase, then sort and print the words into descending order lexicographically. Hint: use split function to split a string into a list, then sort it with a method.

```
Enter a string: Three Rings for the eleven kings under the sky
Output: under three the the sky rings kings for eleven
Enter a string: Seven for the Dwarf lords in their halls of stone
Output: their the stone seven of lords in halls for dwarf
Enter a string: Nine for Mortal Men doomed to die
Output: to nine mortal men for doomed die
Enter a string: One for the Dark Lord on his dark throne
Output: throne the one on lord his for dark dark
Enter a string: In the land of Mordor where the Shadows lie
Output: where the the shadows of mordor lie land in
Enter a string:
```

Testing: Test your code with the example output above.

```
while True:
    text = input("Enter a string: ")

    if text == "":
        break

    words = text.lower().split()
    words.sort(reverse=True)

    print("Output:", " ".join(words))
```

Processing pipeline: string → lowercase string → list of words → descending sort → output string.

Problem 5 — Happy or lonely g's

Problem: We'll say that a lowercase 'g' in a string is "happy" if there is another 'g' immediately to its left or right. Write a function to print 'HAPPY' if all the g's in the given string are happy, otherwise, print 'LONELY'.

Example input and output:

```
String: xggt
Happy?: True

String: abgsx
Happy?: False

String: g
Happy?: False

String: brggsomr
Happy?: True
```

Testing: Test your code with the example output above.

```
def happy_gs(text):
    for i in range(len(text)):
        if text[i] == "g":
            left_happy = i > 0 and text[i - 1] == "g"
            right_happy = i < len(text) - 1 and text[i + 1] == "g"

            if not (left_happy or right_happy):
                return "LONELY"

    return "HAPPY"


text = input("String: ")
print(happy_gs(text))
```

Boundary checks prevent access outside the string. A g is happy if at least one immediate neighbour is another g.

```
# Example results
ggt       -> HAPPY
abgxx     -> LONELY
g         -> LONELY
brgggor   -> HAPPY
```

12. A Reliable Strategy for String Problems

1. Identify the input and the stopping condition.
2. Decide whether you need the whole string, one character, or a substring.
3. Use indexing when position matters; use slicing for a range.
4. Use a string method when it directly expresses the required operation.
5. If every character must be examined, consider for ch in text or indexed iteration.
6. Maintain variables such as longest, count, or result when the program must remember information.
7. Check boundary cases: empty strings, one-character strings, first/last characters, and no-match cases.
8. Test the program with several inputs before trusting the output.

13. Common Mistakes

Mistake	Fix

Thinking the first character is index 1.	The first character is s[0].
Including the stop index in a slice.	The stop is excluded: s[1:4] uses 1, 2, 3.
Using s[0:3] for the last three characters.	Use s[-3:].
Trying s.reverse() on a string.	Use s[::-1].
Expecting s.upper() to change s.	Use s = s.upper().
Using index() when a match may be absent.	Use find() when -1 is easier to handle.
Forgetting split() returns a list.	Use join() when you need to turn it back into a string.
Writing words.join(" ").	Write " ".join(words).
Trying to assign s[0] = "x".	Strings are immutable; create a new string.
Checking i-1 or i+1 without boundary checks.	Check i > 0 and i < len(s)-1 first.

14. Check Your Understanding — Answers

Question: What is the first index of a string?

Answer: 0.

Question: What does s[-1] mean?

Answer: The last character.

Question: What does s[2:5] return?

Answer: Indexes 2, 3 and 4.

Question: What does s[::-1] do?

Answer: Returns the string reversed.

Question: What does len(s) return?

Answer: The number of characters in s.

Question: What does x in s return?

Answer: True if x occurs in s; otherwise False.

Question: What does find() return when a substring is absent?

Answer: -1.

Question: What happens if index() cannot find the substring?

Answer: It raises ValueError.

Question: What does split() normally return?

Answer: A list of strings.

Question: What does join() do?

Answer: Combines strings using a separator.

Question: Are strings mutable?

Answer: No. Strings are immutable.

Question: What is the preferred modern way to insert variables into formatted output?

Answer: An f-string, such as f"Score: {score}".

15. Extra Practice

Task	Answer
Print the first four characters.	print(text[:4])
Print the last five characters.	print(text[-5:])
Print every second character.	print(text[::2])
Print backwards.	print(text[::-1])
Count spaces.	print(text.count(" "))
Trim spaces and lowercase.	text = text.strip().lower()
Check whether it starts with A.	print(text.startswith("A"))
Replace spaces with underscores.	print(text.replace(" ", "_"))
Turn a sentence into words.	words = text.split()
Turn a list of words into a sentence.	sentence = " ".join(words)

16. Quick Reference

Concept	Syntax	Purpose
Length	len(s)	number of characters
Membership	x in s	check whether x occurs
Index	s[i]	one character
Slice	s[a:b]	characters a through b-1
Step	s[a:b:c]	move by c
Reverse	s[::-1]	reverse
Case	s.upper(), s.lower()	change case
Trim	s.strip()	remove outer whitespace
Search	s.find(x)	first position or -1
Replace	s.replace(a,b)	replace occurrences
Count	s.count(x)	count occurrences
Split	s.split()	string → list
Join	" ".join(words)	list → string
Format	f"{x}"	insert values

17. Final Takeaway

The key progression this week is:

```
STRING
  ↓
INDEXING      → choose one character
  ↓
SLICING       → choose a range
  ↓
METHODS       → search, clean, replace, split, join
  ↓
FORMATTING    → communicate results clearly
```

Do not try to memorise every method. Learn to recognise the operation you need, choose the simplest tool, and test it with small examples.

Workshop / exam mindset
Be able to predict what an expression returns, explain why an index or slice works, choose an appropriate string method, and combine strings with loops, conditions, and functions.

End of Chapter 7