

Operators, Strings, Indexing, Slicing & String Methods

Master Python's operators (arithmetic, comparison, logical, assignment, membership, identity, bitwise) and work with text: string indexing and slicing, immutability, every string method an analyst needs (split, join, strip, replace, find, startswith, case methods, isdigit...), escape sequences, and cleaning real messy text.

● Python classes 3–4

✦ Python 3.13

Beginner

🕒 120 min

By the end of this lesson you can

- Use arithmetic, comparison, logical, membership and identity operators and explain operator precedence
- Index and slice strings forwards and backwards, including steps, and explain immutability
- Apply string methods to clean, split, search, join and format text
- Build small text-cleaning functions and check results on real data
- Recognise common string pitfalls (encoding, whitespace, case, escape characters)

Datasets used

customers.csv

Operators

Arithmetic

Operator	Meaning	Example	Result
<code>+</code> <code>-</code> <code>*</code>	Add, subtract, multiply	<code>7 * 3</code>	21
<code>/</code>	True division (always float)	<code>7 / 2</code>	3.5
<code>//</code>	Floor division	<code>7 // 2</code> , <code>-7 // 2</code>	3, -4
<code>%</code>	Remainder (modulo)	<code>7 % 2</code> , <code>-7 % 3</code>	1, 2
<code>**</code>	Power	<code>2 ** 10</code>	1024

Python

```
print(7 / 2, 7 // 2, -7 // 2, 7 % 2, -7 % 3, 2 ** 10, 9 ** 0.5)
print(divmod(17, 5))           # (quotient, remainder) in one call
hours = 135; print(f"{hours // 60} h {hours % 60} min")      # split minutes into hours
and minutes
print(10 % 3 == 1, 12 % 2 == 0)      # test odd/even and divisibility
```

OUTPUT

```
3.5 3 -4 1 2 1024 3.0
(3, 2)
2 h 15 min
True True
```

Comparison and logical

Python

```
x = 15
print(x > 10, x == 15, x != 15, 10 < x <= 20)      # chained comparison: 10 < x and x <= 20
print(x > 10 and x < 20, x < 5 or x > 12, not x > 10)
print("apple" < "banana", "Zebra" < "apple")      # strings compare by character codes:
uppercase before lowercase
```

OUTPUT

```
True True False True
True True False
True True
```

Short-circuit evaluation: `A and B` stops at A if A is falsy; `A or B` stops at A if A is truthy. They return the **deciding operand**, not just True/False:

Python

```
print(0 or "default", "" or "n/a", "x" and "y", None and 1)
name = ""; print(name or "Anonymous")           # a common idiom for default values
print(5 > 3 and print("evaluated") is None)      # right side runs only when needed
```

OUTPUT

```
default n/a y None
Anonymous
evaluated
True
```

Assignment, membership, identity

Python

```
count = 10; count += 5; count *= 2; count -= 1; count /= 4; count **= 2
print(count)
print(3 in [1, 2, 3], "py" in "python", "z" not in "python", "a" in {"a": 1})    #
membership
a = [1, 2]; b = [1, 2]; c = a
print(a == b, a is b, a is c, None is None)                                     #
equality vs identity
```

OUTPUT

```
49
True True True True
True False True True
```

Bitwise (awareness)

`&` AND, `|` OR, `^` XOR, `~` NOT, `<<` `>>` shifts: used for flags and low-level work; in pandas/NumPy `&` and `|` combine boolean arrays (use brackets).

Python

```
print(5 & 3, 5 | 3, 5 ^ 3, 5 << 1, 5 >> 1, bin(5))
```

OUTPUT

```
1 7 6 10 2 0b101
```

Precedence

From highest to lowest: `**` → unary `-x` → `*` / `//` / `%` → `+` / `-` → comparisons (`<` / `=` / `in` / `is`) → `not` → `and` → `or`. When in doubt, use brackets.

Python

```
print(2 + 3 * 4 ** 2, (2 + 3) * 4 ** 2, -2 ** 2, (-2) ** 2, not 1 == 2 or False)
```

OUTPUT

```
50 80 -4 4 True
```

Walrus operator (3.8+)

`:=` assigns inside an expression: `if (n := len(items)) > 10: print(f"{n} items")`. Useful in loops and comprehensions; do not overuse.

Strings

A **string** is an immutable sequence of Unicode characters. Create with single, double or triple quotes.

Python

```
city = "Bengaluru"
quote = 'She said "hello"'
multi = """Line 1
Line 2"""
hindi = "नमस्ते" # Unicode works everywhere
print(len(city), len(hindi), multi.splitlines(), city.upper())
print("Ha" * 3, "data" + " " + "science", "-" * 20)
```

OUTPUT

```
9 6 ['Line 1', 'Line 2'] BENGALURU
HaHaHa data science -----
```

Indexing

Characters are numbered from 0 on the left and from -1 on the right.

index	0	1	2	3	4	5	6	7	8
	B	E	N	G	A	L	U	R	U
neg.	-9	-8	-7	-6	-5	-4	-3	-2	-1

Indexing a string: positive indexes count from the left (starting at 0), negative from the right (starting at -1).

Python

```
w = "BENGALURU"
print(w[0], w[3], w[-1], w[-3])
try: print(w[20])
except IndexError as e: print("IndexError:", e)
```

OUTPUT

```
B G U U
IndexError: string index out of range
```

Slicing: s[start:stop:step]

start is included, **stop** is **excluded**; omit a part to mean "from the beginning / to the end"; a negative step goes backwards.

Python

```
w = "BENGALURU"
print(w[0:3], w[:3], w[3:], w[3:6], w[-4:], w[:-1])
print(w[::2], w[::-1], w[1:3], w[8:2:-2])
print(w[2:100], repr(w[5:2])) # out-of-range slices are safe; an empty slice is ''
```

OUTPUT

```
BEN BEN GALURU GAL LURU BENGALUR
BNAUU URULAGNEB EAR UUA
NGALURU ''
```

Slice	Meaning
<code>s[:n]</code>	First n characters
<code>s[-n:]</code>	Last n characters
<code>s[a:b]</code>	From a up to (not including) b; length b – a
<code>s[::-1]</code>	Reversed
<code>s[::2]</code>	Every second character

Python

```
order_id = "IN-2025-00042"
country, year, number = order_id[:2], int(order_id[3:7]), int(order_id[-5:])
print(country, year, number)           # same as LEFT/MID/RIGHT in Excel and SQL
```

OUTPUT

IN 2025 42

String methods

Strings are immutable, so every method **returns a new string**; the original stays unchanged.

Case and cleaning

Python

```
s = "  new DELHI  "
print(repr(s.strip()), repr(s.lstrip()), repr(s.rstrip()))
print(s.strip().lower(), s.strip().upper(), s.strip().title(), s.strip().capitalize(),
s.strip().swapcase())
print("xxHixx".strip("x"), "2025-04-27".replace("-", "/"), "a,b,,c".replace(",", ", "))
print("Bengaluru".casefold() == "BENGALURU".casefold())           # robust case-insensitive
comparison
```

OUTPUT

```
'new DELHI' 'new DELHI' ' ' 'new DELHI'
new delhi NEW DELHI New Delhi New delhi NEW delhi
Hi 2025/04/27 a,b,c
True
```

Searching and testing

Python

```
t = "Delhi, Mumbai, Delhi, Pune"
print(t.find("Delhi"), t.rfind("Delhi"), t.find("Goa"), t.count("Delhi"),
t.index("Mumbai"))
print(t.startswith("Delhi"), t.endswith("Pune"), "Mumbai" in t)
print("12345".isdigit(), "12.5".isdigit(), "abc".isalpha(), "abc123".isalnum(), "
.isisspace(), "Abc".istitle(), "abc".islower())
```

OUTPUT

```
0 15 -1 2 7
True True True
True False True True True True True
```

`find` returns `-1` when not found; `index` raises `ValueError`. `isdigit` is False for decimals and negatives, so use `try: float(s)` to test numbers.

Splitting and joining

Python

```
csv_line = "E101,Kavya Iyer,HR,97000"
parts = csv_line.split(",")
print(parts, len(parts))
print("one two three".split(), "a-b-c".split("-", 1), "line1\nline2".splitlines(),
"path/to/file.csv".rsplit("/", 1))
print(", ".join(["Pune", "Delhi", "Goa"]), "".join(["a", "b"]), " | ".join(str(n) for n in
[1, 2, 3]))
print("abc".partition("b"), "IN-2025-00042".partition("-"))
```

OUTPUT

```
['E101', 'Kavya Iyer', 'HR', '97000'] 4
['one', 'two', 'three'] ['a', 'b-c'] ['line1', 'line2'] ['path/to', 'file.csv']
Pune, Delhi, Goa ab 1 | 2 | 3
('a', 'b', 'c') ('IN', '-', '2025-00042')
```

`split()` with no argument splits on any whitespace and ignores repeats: the best way to tokenise text. `join` is called on the **separator**, with a list of strings.

Alignment, padding and formatting

Python

```
print("42".zfill(6), "7".rjust(4, "0"), "ab".ljust(5, ".") + "|", "ab".center(8, "*"))
print("Total".ljust(10) + "1,500".rjust(10))
print("{:>8} {:<8} {:^8}".format("right", "left", "mid"))
print("tab\tseparated", "new\nline stays literal", "unicode: \u20b9 \u2713")
```

OUTPUT

```
000042 0007 ab...| **ab**
Total          1,500
   right left      mid
tab    separated new\nline stays literal unicode: ₹ ✓
```

Translating and encoding

Python

```
table = str.maketrans({"₹": "Rs ", ",": ""})
print("₹1,25,000".translate(table))
print("naïve café".encode("utf-8"), "abc".encode("ascii"), b"abc".decode())
print(ord("A"), chr(97), "é".encode("utf-8"), len("é".encode("utf-8")))
```

OUTPUT

```
Rs 125000
b'na\xc3\xafve caf\xc3\xa9' b'abc' abc
65 a b'\xc3\xa9' 2
```

NOTE Encoding

Text on disk is **bytes**. Most files today are **UTF-8**; if you see `Ã©` or `💎`, the file was read with the wrong encoding. In pandas use `pd.read_csv(path, encoding="utf-8")` or `"latin-1"` / `"cp1252"` for old Windows exports.

Method cheat sheet

Goal	Methods
Change case	<code>lower()</code> , <code>upper()</code> , <code>title()</code> , <code>capitalize()</code> , <code>casefold()</code>
Trim	<code>strip()</code> , <code>lstrip()</code> , <code>rstrip()</code> , <code>removeprefix()</code> , <code>removesuffix()</code>
Replace	<code>replace(old, new, count)</code> , <code>translate()</code>
Search	<code>find()</code> , <code>rfind()</code> , <code>index()</code> , <code>count()</code> , <code>in</code> , <code>startswith()</code> , <code>endswith()</code>
Test	<code>isdigit()</code> , <code>isalpha()</code> , <code>isalnum()</code> , <code>isspace()</code> , <code>isupper()</code> , <code>islower()</code>
Split / join	<code>split()</code> , <code>rsplit()</code> , <code>splitlines()</code> , <code>partition()</code> , <code>join()</code>
Pad / align	<code>zfill()</code> , <code>ljust()</code> , <code>rjust()</code> , <code>center()</code>
Format	f-strings, <code>format()</code> , <code>format_map()</code>

Cleaning real text

Customer names from the course's `customers.csv` , cleaned with the same rules you used in Excel (`TRIM` , `PROPER`) and SQL:

Python

```
import pandas as pd
cust = pd.read_csv("customers.csv")
raw = [" asha RAO ", "ravi verma", "MEERA\tIYER", "Kabir singh "] # messy
examples
def clean_name(name):
    parts = name.split() # split() collapses all whitespace
    return " ".join(p.capitalize() for p in parts)
print([clean_name(n) for n in raw])
print(cust.customer_name.head(5).tolist())
first_names = [n.split()[0] for n in cust.customer_name]
print(len(first_names), "names; most common first names:",
pd.Series(first_names).value_counts().head(5).to_dict())
```

OUTPUT

```
['Asha Rao', 'Ravi Verma', 'Meera Iyer', 'Kabir Singh']
['Saanvi Khan', 'Meera Bose', 'Riya Shah', 'Diya Fernandes', 'Neha Sharma']
800 names; most common first names: {'Arjun': 36, 'Vikram': 31, 'Aarav': 31, 'Sneha':
30, 'Priya': 30}
```

Python

```
def initials(name): return "".join(w[0].upper() for w in name.split())
def mask_email(email):
    user, _, domain = email.partition("@")
    return user[0] + "*" * (len(user) - 2) + user[-1] + "@" + domain
print(initials("kavya iyer"), mask_email("kavya.iyer@abc.in"))
print("palindrome" if (s := "Level").lower() == s.lower()[::-1] else "not")
```

OUTPUT

```
KI k*****r@abc.in
palindrome
```

Counting characters and words

Python

```
text = "data science is fun and data engineering is fun too"
words = text.split()
freq = {}
for w in words: freq[w] = freq.get(w, 0) + 1
print(freq)
print(max(freq, key=freq.get), len(words), len(set(words)))
```

OUTPUT

```
{'data': 2, 'science': 1, 'is': 2, 'fun': 2, 'and': 1, 'engineering': 1, 'too': 1}
data 10 7
```

(`collections.Counter(words)` does this in one line: Lesson 8.)

Common pitfalls

Pitfall	Example	Fix
Forgetting strings are immutable	<code>s.upper()</code> without assigning	<code>s = s.upper()</code>
Whitespace/case mismatches	<code>"Delhi " != "Delhi"</code>	<code>strip()</code> , <code>lower()</code> before comparing
Using <code>+</code> with numbers	<code>"Total: " + 5</code>	f-string or <code>str(5)</code>
<code>isdigit</code> for decimals	<code>"3.5".isdigit()</code> → False	try: <code>float(x)</code>
Off-by-one in slices	<code>s[0:3]</code> has 3 characters	stop is excluded
Backslashes in Windows paths	<code>"C:\new"</code> → newline!	raw string <code>r"C:\new"</code> or forward slashes
Huge strings in loops	<code>s += x</code> in a loop	build a list and <code>"".join(list)</code>
Wrong encoding	Ã© characters	<code>encoding="utf-8"</code>

DO IT WITH AI

Do it with AI

Ask an assistant to produce 10 messy phone numbers (fake) and then write a function to standardise them to `+91-XXXXX-XXXXX` . Test it yourself on edge cases (spaces, dashes, leading 0, +91 already present) and compare approaches: string methods first, then regular expressions (Lesson 14).



Summary

KEY TAKEAWAYS

- Operators: `/` true division, `//` floor division, `%` remainder, `**` power; comparisons chain `( 10 < x ≤ 20 )`; and `/ or` short-circuit and return an operand; use `in` , `not in` , `is` .
- Strings are **immutable** sequences: index from 0 or `-1`, slice `[start:stop:step]` (stop excluded).
- Methods return new strings: `strip` , `lower/upper/title` , `replace` , `split / join` , `find / count` , `startswith` , `zfill` , `partition` .
- Clean text by trimming, normalising case/whitespace and replacing variants before comparing or counting.
- Watch pitfalls: forgetting to assign, off-by-one slices, wrong encodings, Windows backslashes.

Quick quiz

6 questions · tap an answer to check it

1. What does `"BENGALURU"[2:5]` return?

A NGA

B ENG

C NGAL

D GAL

Answer: A. Index 2 is N, 3 is G and 4 is A; the stop index 5 is excluded.

2. What is `-7 // 2` in Python?

A `-3`

B `-4`

C `3`

D `-3.5`

Answer: B. Floor division rounds down towards minus infinity.

3. What does `0` or `"default"` return?

A `0`

B `True`

C `"default"`

D `None`

Answer: C. `or` returns the first truthy operand (or the last operand).

4. Which statement is true about strings?

A `s[0] = "X"` changes the first character

B Strings are immutable, so methods like `upper()` return a new string

C `split()` changes the original string

D `strip()` removes only spaces inside the text

Answer: B. Assign the result to keep it.

5. How do you split text on any whitespace and ignore repeated spaces?

A `text.split(" ")`

B `text.split()`

C `text.partition()`

D `text.strip()`

Answer: B. `split()` without an argument splits on runs of whitespace.

6. `"3.5".isdigit()` returns:

A True

B False

C 3.5

D An error

Answer: B. `isdigit()` is True only for strings of digits; use `float()` inside `try/except` for decimals.

EXERCISE String practice

(1) From "IN-2025-00042" extract the country, year (int) and sequence number (int) with slicing. (2) Write `is_valid_pin(text)` that returns True for six-digit PIN codes not starting with 0. (3) Reverse the words in "data science is fun" (word order, not letters). (4) Turn "rAHuL VERMA " into "Rahul Verma" .

Show solution (worked answer)

Python

```
oid = "IN-2025-00042"; print(oid[:2], int(oid[3:7]), int(oid[-5:]))
def is_valid_pin(t): return len(t) == 6 and t.isdigit() and t[0] != "0"
print(is_valid_pin("411001"), is_valid_pin("041100"), is_valid_pin("4110a1"))
print(" ".join("data science is fun".split()[::-1]))           # fun is science data
print(" ".join(w.capitalize() for w in " rAHuL VERMA ".split()))
```

INTERVIEW CORNER Questions you may be asked

Q: How do you reverse a string in Python? `s[::-1]` (slice with step -1) or `"".join(reversed(s))` .

Q: Why use `"".join(List)` instead of `+=` in a loop? Strings are immutable; each `+=` creates a new string (quadratic time). Building a list and joining once is linear.

Q: Difference between `find` and `index` ? `find` returns -1 if the substring is missing; `index` raises `ValueError` .

Q: What does `//` do with negative numbers? Floors toward minus infinity: `-7 // 2 = -4` , and `-7 % 3 = 2` , so `(a // b) * b + a % b == a` always holds.

Practise more online

Free, trusted places to test yourself on this topic. Links open the original site in a new tab.