



A pattern describes text; `re` finds it, splits on it or replaces it. Always write patterns as raw strings (`r"\d+"`) so backslashes reach `re` untouched. Examples use `import re` and `s = "Order 66 shipped 2026-03-14 to ada@dev.io"`.

THE FUNCTIONS

```
re.findall(r"\d+", s)
→ ['66', '2026', '03', '14']
every match, as a list of strings

re.search(r"\d+", s).group()
the first match anywhere, or None

re.match(r"\d+", s)
match only looks at the start

bool(re.fullmatch(r"\d{4}", "2026"))
→ True
the whole string must match

re.sub(r"\d", "#", "a1b22")
→ 'a#b##'
replace every match

re.split(r"[,;]\s*", "a, b;c")
→ ['a', 'b', 'c']
split on a pattern

[m.span() for m in re.finditer(r"\d+",
"a1 b22")]
→ [(1, 2), (4, 6)]
match objects: where, not just what

DATE = re.compile(r"\d{4}-\d\d-\d\d")
DATE.search(s)
compile once when a pattern is reused
```

CHARACTER CLASSES

```
\d \w \s
digit, word, space; \D \W \S mean NOT

re.findall(r"\w+", "it's ok_2")
→ ['it', 's', 'ok_2']
\w is letters, digits and _

re.findall(r"[aeiou]", "regex")
→ ['e', 'e']
any one of these

re.findall(r"[^a-z]", "ab C1")
→ ['C', '1']
^ inside [] means not these

re.findall(r"a.c", "abc a-c ac")
→ ['abc', 'a-c']
. is any character but a new line

re.findall(r"\.", "v1.2.3")
→ ['.', '.']
escape . to mean a real dot
```

HOW MANY

```
re.findall(r"lo+l", "ll lol loool")
→ ['lol', 'loool']
+ one or more

re.findall(r"lo*l", "ll lol")
→ ['ll', 'lol']
* zero or more

re.findall(r"colou?r", "color colour")
→ ['color', 'colour']
? optional

re.findall(r"\d{3}", "12345678")
→ ['123', '456']
{3} exactly three

re.findall(r"\d{2,3}", "12345678")
→ ['123', '456', '78']
{2,3} two to three, as many as fit

re.findall(r"<.+>", "<a><b>")
→ ['<a><b>']
greedy: as much as possible

re.findall(r"<.+?>", "<a><b>")
→ ['<a>', '<b>']
+? lazy: as little as possible
```

ANCHORS

```
bool(re.search(r"^$\d+$", "2026"))
^ start, $ end: whole-string check

re.findall(r"\bcat\b", "cat concat cat.")
→ ['cat', 'cat']
\b word boundary: whole words only

re.findall(r"^[w+]", "a b\nc d", re.M)
→ ['a', 'c']
re.M: ^ and $ work per line
```

GROUPS

```
m.group(1), m.group(2)
→ ('ada', 'dev.io')
() captures part of the match

m.group(0)
0 is the whole match

re.findall(r"(\d+)-(\d+)", "1-2 3-4")
→ [(1, 2), (3, 4)]
with groups, findall gives tuples

re.findall(r"(?:ab)+", "abab x ab")
→ ['abab', 'ab']
(?:) groups without capturing

re.findall(r"cat|dog", "cat, dog, cow")
→ ['cat', 'dog']
| either side

re.findall(r"(\w)\1", "book keeper")
→ ['o', 'e']
\1 the same text again
```

NAMED GROUPS

```
m = re.search(
    r"(?P<y>\d+)-(?P<m>\d+)", s)
(?P<name>...) Labels a group

m["y"]
same as m.group("y")

m.groupdict()
→ {'y': '2026', 'm': '03'}
every named group at once
```

LOOK AROUND

```
re.findall(r"\d+(?=px)", "10px 2em 5px")
→ ['10', '5']
(?=) followed by, not included

re.findall(r"(?<=\$)\d+", "$5 and 7")
→ ['5']
(?<=) preceded by

re.findall(r"\b\d+\b(?:!%)", "5% 7")
→ ['7']
(?:!) not followed by
```

REPLACE

```
re.sub(r"\s+", " ", "a b\n c")
tidy runs of whitespace

re.sub(r"(\w+)-(\w+)", r"\2-\1", "a-b")
→ 'b-a'
\1 \2 put the groups back

re.sub(r"(?P<d>\d)", r"[\g<d>]", "a1")
→ 'a[1]'
\g<name> for named groups

re.sub(r"\d", lambda m: m[0] * 2, "a1")
→ 'a11'
a function decides each replacement

re.sub(r"o", "0", "foo", count=1)
only the first
```

FLAGS

```
re.findall(r"py", "Py PY", re.I)
→ ['Py', 'PY']
re.I ignores case; combine: re.I | re.M

re.findall(r"(?i)py", "Py")
→ ['Py']
or set the flag inside the pattern

re.findall(r"a.b", "a\nb", re.S)
→ ['a\nb']
re.S lets . match a new line too

re.compile(r"""
    (\d{3}) # area
    -(\d{4}) # number
""", re.X)
re.X: spaces and # comments ignored
```

USEFUL PATTERNS

```
re.findall(r"[\w.+-]+@[ \w-]+\.[\w.]+", s)
→ ['ada@dev.io']
email, loose: good for finding

re.findall(r"-?\d+(?:\.\d+)?", "-3.5 2")
→ ['-3.5', '2']
signed numbers

re.findall(r"\d{4}-\d{2}-\d{2}", s)
→ ['2026-03-14']
ISO date

re.findall(r"https?://\S+", "x http://a")
→ ['http://a']
URL, loose

re.sub(r"<[^>]+>", "", "<b>hi</b>")
→ 'hi'
strip simple tags
```

WATCH OUT

```
if m := re.search(r"\d+", s):
    print(m.group())
no match is None, and None.group() raises
AttributeError

re.match(r"\d+", s)
match only tries the start: use search

re.findall(r"\d+.\d+", "1x5")
an unescaped . matches anything: write \.

re.findall(r"(\d+)px", "5px 9px")
→ ['5', '9']
one group: findall returns only that part

re.split(r"(", "a,b")
a group in split keeps the separators

re.findall(re.escape("a+b"), "a+b")
→ ['a+b']
escape() makes user text literal

re.findall("\bhi", "hi")
no r prefix: \b turned into a backspace

re.findall(r"[A-Z]", "^_")
A-z also covers [\^_]: use A-Za-z
```