Python RegEx

Regular expressions in Python live in the `re` module. Always write patterns as raw strings (`r"..."`) so backslashes reach the regex engine instead of being eaten by Python's string parser.

The `re` Module

Call	Returns / does
`re.search(p, s)`	first match anywhere, else `None`
`re.match(p, s)`	match only at the **start** of the string
`re.fullmatch(p, s)`	match only if the **whole** string matches
`re.findall(p, s)`	list of all matches (or group tuples)
`re.finditer(p, s)`	iterator of match objects
`re.sub(p, repl, s)`	replace all matches
`re.split(p, s)`	split the string on the pattern
`re.compile(p)`	pre-compile a pattern for reuse

Extract, then use the match

```
import re

m = re.search(r"(\d{4})-(\d{2})-(\d{2})", text)
if m:
    year, month, day = m.groups()
    print(m.group(0))   # the full match
```

Character Classes

Token	Matches
`.`	any character except newline
`\d / \D`	a digit / a non-digit
`\w / \W`	word char (a-z, 0-9, _) / non-word
`\s / \S`	whitespace / non-whitespace
`\b`	a word boundary
`[abc]`	any one of a, b, c
`[^abc]`	any char except a, b, c
`[a-z0-9]`	a range: lowercase letter or digit

Anchors & Quantifiers

Token	Meaning
`^ / $`	start / end of string (or line with `re.M`)
`*`	0 or more (greedy)
`+`	1 or more
`?`	0 or 1 (optional)
`{3}`	exactly 3
`{2,5}`	between 2 and 5
`{2,}`	2 or more
`*? +? ??`	lazy — match as few as possible

> **Greedy by default.** `.*` grabs as much as it can and backtracks. For `<.*>` on `<a><b>` that's the whole string — use the lazy `<.*?>` to stop at the first `>`.

Groups & Lookarounds

Token	Meaning
`(abc)`	capturing group — numbered from 1
`(?:abc)`	group without capturing
`(?P<name>abc)`	named group — read via `m.group('name')`
`a	b`
`(?=abc)`	lookahead: followed by abc
`(?!abc)`	negative lookahead: not followed by abc
`(?⩽abc)`	lookbehind: preceded by abc
`\1`	backreference to group 1

Named groups + sub with a backreference

```
# swap "First Last" → "Last, First"
re.sub(r"(?P<f>\w+)\s+(?P<l>\w+)",
       r"\g<l>, \g<f>",
       "Grace Hopper")
# → "Hopper, Grace"
```

Flags

Flag	Effect
`re.I (IGNORECASE)`	case-insensitive matching
`re.M (MULTILINE)`	`^` and `$` match at each line
`re.S (DOTALL)`	`.` also matches newlines
`re.X (VERBOSE)`	ignore whitespace/comments in the pattern

Combine flags with |

```
re.findall(r"^error:.*", log, re.I | re.M)
```