What is a Regular Expression?

A Regular Expression (regex) is a pattern that describes a set of strings.

In Python, we use it for searching, matching, and manipulating text using the re module.

```
import re
```

Main Functions in Python's re Module

1. re.search(pattern, string)

Searches for first occurrence of the pattern anywhere in the string.

Returns a Match object if found, else None.

```
import re
txt = "Python is powerful"
result = re.search(r"power", txt)
if result:
    print("Found:", result.group())  # Found: power
```

2. re.match(pattern, string)

Matches pattern only at the start of the string.

```
txt = "Python is powerful"
print(re.match(r"Python", txt))  # Match object
print(re.match(r"power", txt))   # None
```

3. re.findall(pattern, string)

Returns all matches as a list of strings.

```
txt = "I have 2 apples and 5 bananas"
print(re.findall(r"\d+", txt))  # ['2', '5']
```

4. re.finditer(pattern, string)

Returns an iterator of Match objects for all matches.

Useful when you also need positions.

```
txt = "I have 2 apples and 5 bananas"
for match in re.finditer(r"\d+", txt):
    print(match.group(), "at position", match.start())
```

Output:

```
2 at position 7
5 at position 18
```

Components of Regex Patterns

A. Literal Characters

Matches exactly the characters you type.

```
re.search(r"cat", "A black cat")  # Matches "cat"
```

B. Meta Characters (special meanings)

Symbol	Meaning	Example	Matches
.	Any character except newline	c.t	cat, cot, cut
^	Start of string	^Hello	Matches "Hello" at start
$	End of string	world$	Matches "world" at end
[]	Match any one char inside	[aeiou]	Any vowel
[^]	Match any char NOT inside	[^0-9]	Not a digit
`	`	OR	`cat
()	Group	(ab)+	ab, abab, ababab

C. Character Classes

Class	Meaning	Example
\d	Digit (0–9)	\d+ → "123"
\D	Not a digit	\D+ → "abc"
\w	Word char (letters, digits, _)	\w+ → "Python3"
\W	Not a word char	\W+ → "@!"
\s	Whitespace	\s+ → " "
\S	Not whitespace	\S+ → "Python"

D. Quantifiers

Symbol	Meaning	Example	Matches
*	0 or more	go*	g, go, goo
+	1 or more	go+	go, goo
?	0 or 1	go?	g, go
{n}	Exactly n	\d{4}	2025
{n,}	n or more	\d{2,}	12, 123
{n,m}	Between n and m	\d{2,4}	12, 1234

E. Anchors

^pattern → Match at start

pattern$ → Match at end

```
re.search(r"^Hello", "Hello World")  # Match
re.search(r"World$", "Hello World")  # Match
```

F. Grouping & Capturing

Grouping: () groups part of a pattern.

Capturing: Store matched text for later use.

```
text = "Name: Python, Age: 25"
match = re.search(r"Name: (\w+), Age: (\d+)", text)
print(match.group(1))  # Python
print(match.group(2))  # 25
```

G. Backreferences

Reuse captured groups inside the regex.

```
text = "He said that that was fine."
match = re.search(r"\b(\w+)\s+\1\b", text)
print(match.group())  # that that
```

Practical Examples

Validate Email

```
pattern = r"^[\w\.-]+@[\w\.-]+\.\w+$"
print(re.match(pattern, "user@example.com"))  # Match object
```

Extract Dates

```
txt = "Event on 2025-08-10 and 2025-08-15"
print(re.findall(r"\d{4}-\d{2}-\d{2}", txt))
```

Mask Phone Numbers

```
txt = "Call me at 9876543210"
print(re.sub(r"\d{6}$", "******", txt))  # Call me at 9876******
```

Beginner Tip:

Use regex101.com to test patterns interactively.