Introduction to Regular Expressions (RegEx) in Python

Open in Colab

Regular Expressions (RegEx) help us **search, extract, and manipulate** text efficiently. Python provides the `re` module to work with regex.

In this notebook, we'll cover **simple regex patterns with diverse examples**, perfect for beginners!

```
import re  # Import the regular expressions module
```

1. Searching for a Simple Character

Pattern: `a`

This pattern matches the letter 'a' anywhere in the string.

```
text = 'apple banana cat'
match = re.search(r'a', text)
if match:
    print(f'Found "a" at position: {match.start()}')  # Output: Position of first 'a'
else:
        print('No match found.')
```

```
Found "a" at position: 0
```

2. Matching an Exact Word

Pattern: `cat`

This pattern finds the word 'cat' in a string.

```
text = 'The cat sat on the mat.'
match = re.search(r'cat', text)
if match:
    print(f'Found "cat" at position: {match.start()}')
else:
    print('No match found.')
```

```
Found "cat" at position: 4
```

3. Matching Any Character

Pattern: `c.t`

The `.` (dot) matches any single character. This finds words like 'cat', 'cot', or 'cut'.

```
text = 'cat cot cut'
matches = re.findall(r'c.t', text)
print(f'Matching words: {matches}')  # Output: ['cat', 'cot', 'cut']
```

```
Matching words: ['cat', 'cot', 'cut']
```

4. Matching a Digit

Pattern: `\d`

The `\d` pattern matches any single digit (0-9).

```
text = 'My number is 5 and yours is 7.'
matches = re.findall(r'\d', text)
print(f'Found digits: {matches}')  # Output: ['5', '7']
```

```
Found digits: ['5', '7']
```

5. Matching a Word Character

Pattern: `\w`

The `\w` pattern matches any letter, number, or underscore (_).

```
text = 'Hello, Python_3!'
matches = re.findall(r'\w', text)
print(f'Word characters: {matches}')  # Output: ['H', 'e', 'l', 'l', 'o', ..., '3']
```

```
Word characters: ['H', 'e', 'l', 'l', 'o', 'P', 'y', 't', 'h', 'o', 'n', '_', '3']
```

6. Matching Whitespace

Pattern: `\s`

The `\s` pattern matches any space, tab, or newline character.

```
text = 'Hello World\nNew Line'
matches = re.findall(r'\s', text)
print(f'Whitespace characters found: {len(matches)}')  # Output: 3 (space, newline, space)
```

```
Whitespace characters found: 3
```

7. Matching Multiple Occurrences

Pattern: `o+`

The `+` means 'one or more' repetitions of the letter 'o'.

```
text = 'soooon moon good'
matches = re.findall(r'o+', text)
print(f'Found sequences: {matches}')  # Output: ['oooo', 'oo', 'oo']
```

```
Found sequences: ['ooooo', 'oo', 'oo']
```

8. Matching the Start of a String

Pattern: `^Hello`

The `^` symbol ensures the match is **only at the beginning** of the string.

```
text = 'Hello, world!'
match = re.match(r'^Hello', text)
if match:
    print('The string starts with "Hello"!')
else:
    print('No match at the start.')
```

```
The string starts with "Hello"!
```

9. Matching the End of a String

Pattern: `world!$`

The `$` symbol ensures the match is **only at the end** of the string.

```
text = 'This is my world!'
match = re.search(r'world!$', text)
if match:
    print('The string ends with "world!"')
else:
    print('No match at the end.')
```

```
The string ends with "world!"
```

10. Finding All Words with 'ing'

Pattern: `\b\w+ing\b`

This pattern finds full words ending in 'ing'.

```
text = 'I am running, singing, and playing today.'
matches = re.findall(r'\b\w+ing\b', text)
print(f'Words ending with "ing": {matches}')  # Output: ['running', 'singing', 'playing']
```

```
Words ending with "ing": ['running', 'singing', 'playing']
```

Practical Example

The script below uses regular expressions to automatically extract essential business details including:

- email subjects
- recipient names
- dates
- monetary amounts
- company names

from sample email text, organizing the results into a structured pandas DataFrame for easy analysis.

```
import re

email = """Subject: Meeting Request for Product Launch
    Dear John,

    We are excited to announce the launch of our new product next month. The project cost is estimated at $12,500,000.
    We would like to schedule a meeting on March 20th at 3 PM to discuss partnership opportunities.

    Best regards,
    Jane Doe
    MarketingPro Inc."""
```

Extracting the Subject of the Email

```
# Pattern: r"Subject:\s*(.*)"
# - Matches the word "Subject:" followed by any whitespace (`\s*`).
# - Captures the rest of the line (the actual subject) using `(.*)`,
#   which means "any character (.) zero or more times (*)".
# - This helps in extracting the subject line from the email content.

match = re.search(r"Subject:\s*(.*)", email)
subject = match.group(1) if match else "Not Found"

print(subject)
```

```
Meeting Request for Product Launch
```

Extracting the Recipient Name or Team

```
# Pattern: r"Dear\s+([\w\s]+),"
# - Matches the word "Dear" followed by one or more whitespace characters (`\s+`).
# - Captures the recipient’s name or team using `([\w\s]+)`,
#   which means "one or more word characters (`\w`) or spaces (`\s`)".
# - Ensures the match ends before the comma (`,`), marking the end of the salutation.

match = re.search(r"Dear\s+([\w\s]+),", email)
recipient = match.group(1) if match else "Not Found"

print(recipient)
```

```
John
```

Extracting Any Mentioned Date

```
# Pattern: r"(January|February|March|April|May|June|July|August|September|October|November|December) \d{1,2}"
# - Looks for any full month name (`January` to `December`).
# - Followed by a space and a **1 or 2 digit number** (`\d{1,2}`), representing the day.
# - This extracts dates like "March 20", "April 10", or "May 5" from the email content.

match = re.search(r"(January|February|March|April|May|June|July|August|September|October|November|December) \d{1,2}", email)
date_mentioned = match.group(1) if match else "Not Found"
print(date_mentioned)
```

```
March
```

Extracting Monetary Values (Currency Amounts)

```
# Pattern: r"\$\d{1,3}(?:,\d{3})*(?:\.\d{2})?"
# - Matches a dollar sign (`\$`).
# - Captures **1 to 3 digits** (`\d{1,3}`) at the start of the amount.
# - Optionally matches **comma-separated thousands** using `(?:,\d{3})*` (e.g., "$1,000" or "$15,000").
# - Optionally matches **decimal values** (`(?:\.\d{2})?`), allowing for cents like "$99.99".
# - This extracts monetary values such as "$15,000", "$95,000", or "$499.99" from the email.

match = re.search(r"\$\d{1,3}(?:,\d{3})*(?:\.\d{2})?", email)
amount_mentioned = match.group(0) if match else "Not Found"
print(amount_mentioned)
```

```
$12,500,000
```

Extracting Company Names

```
# Pattern: r"\n\s*([\w\s]+(?:Inc\.|Ltd\.|Corp\.|Giant))"
# - Ensures the match starts on a new line (`\n`).
# - Allows for any leading whitespace (`\s*`).
# - Captures the company name using `([\w\s]+)`, which allows letters and spaces.
# - Ensures the company name ends with a **common business suffix** (e.g., "Inc.", "Ltd.", "Corp.", or "Giant").
# - This extracts names like "MarketingPro Inc.", "TechInnovators Ltd.", and "GlobalCorp."

match = re.search(r"^\s*([\w ]+(?:Inc\.|Ltd\.|Corp\.|Giant))", email,re.MULTILINE)
company_name = match.group(0) if match else "Not Found"
print(company_name)

```

```
MarketingPro Inc.
```

Note:

If we want to match start-of-line within multi-line text we need to set the re.MULTILINE flag, otherwise it will match the first line only

The re.MULTILINE flag redefines ^ and $ so that:

Anchor Normal (default)	With re.MULTILINE
^	Start of string Start of any line
$	End of string End of any line