NLP Mastery – Day 02: Regex & Text Handling

By Shahriar Mahmud Sabuj

“Regex is the scalpel of text — master it, and you can dissect language with surgical precision.”

Table of Contents

Exercise 1 – Match All Capitalized Words

Problem Statement

Find all words that start with a capital letter, including hyphenated names and acronyms.

Final Regex Pattern

```
\b(?:[A-Z][a-z]+|[A-Z]{2,})(?:-[A-Z][a-z]+)*\b
```

Step-by-Step Explanation

- `\b` – Start at a word boundary.
- `(?:[A-Z][a-z]+|[A-Z]{2,})` – Capitalized word OR all■caps acronym.
- `(?:-[A-Z][a-z]+)*` – Optional hyphenated parts.
- `\b` – End at a word boundary.

Example Inputs & Outputs

```
import re
pattern = r"\b(?:[A-Z][a-z]+|[A-Z]{2,})(?:-[A-Z][a-z]+)*\b"
text = "Alice met Dr. Brown in New-York; NASA sent them an invite."
print(re.findall(pattern, text))
# ['Alice', 'Dr', 'Brown', 'New-York', 'NASA']
```

Key Takeaways

- Use non■capturing groups (?:...) to group without capturing.
- Word boundaries keep matches as whole words.

Exercise 2 – Extract All Emails from Text

Problem Statement

Extract all valid email addresses.

Final Regex Pattern

```
[\w\.-]+@[\w\.-]+\.[a-zA-Z]{2,}\b
```

Step-by-Step Explanation

- `[\w\.-]+` – Username part.
- `@` – Literal at sign.
- `[\w\.-]+` – Domain name.
- `\.[a-zA-Z]{2,}` – TLD with at least 2 letters.
- `\b` – Word boundary to avoid trailing punctuation.

Example Inputs & Outputs

```
import re
text = "Contact: hello@example.com, support@mail-service.org, a.b@co.uk"
pattern = r"[\w\.-]+@[\w\.-]+\.[a-zA-Z]{2,}\b"
print(re.findall(pattern, text))
```

Key Takeaways

- Prefer raw strings `r'...'` for regex in Python.
- Add lookbehind `(?<!\w)` at start if you want a left boundary.

Exercise 3 – Extract All URLs from HTML Content

Problem Statement

Pull out every HTTP/HTTPS (and optionally www.) link from mixed text/HTML without trailing punctuation.

Final Regex Pattern

```
(?<!\w)(?:https?://|www\.)[^ \t\r\n\"'<>)]+
```

Step-by-Step Explanation

- `(?<!\w)` – Negative lookbehind to avoid mid■word matches.
- `(?:https?://|www\.)` – Accept scheme or bare www.
- `[^ \t\r\n\"'<>)] +` – Consume URL chars, stop at spaces/quotes/brackets.

Example Inputs & Outputs

```
import re
html = 'See https://example.com/a?x=1#y and www.site.co.uk/page).'
pattern = r"(?<!\w)(?:https?://|www\.)[^ \t\r\n\"'<>\)]+"
urls = re.findall(pattern, html)
cleaned = [u.rstrip('.,);:!?"\'") for u in urls]
print(cleaned)
```

Key Takeaways

- `Post■process to strip punctuation at end if needed.`
- `For real HTML, BeautifulSoup is safer than regex.`

Exercise 4 – Extract All Hashtags from a Tweet

Problem Statement

Find every hashtag like #NLP, #AI_2025.

Final Regex Pattern

```
(?<!\w)#\w+
```

Step-by-Step Explanation

- `(?<!\w)` – Ensure # is not in the middle of a word.
- `#` – Literal hash.
- `\w+` – One or more word characters after #.

Example Inputs & Outputs

```
import re
tweet = "Learning #Python and #AI_2025. Not a#tag."
print(re.findall(r"(?<!\w)#\w+", tweet))
```

Key Takeaways

- Use re.IGNORECASE if you want to ignore case (hashtags are case■insensitive visually).

Exercise 5 – Extract All Numbers from a Paragraph

Problem Statement

Find whole numbers, decimals, and comma■grouped numbers (with optional leading minus).

Final Regex Pattern

```
-?\d{1,3}(?:,\d{3})*(?:\.\d+)?
```

Step-by-Step Explanation

- `-? – Optional minus sign.`
- `\d{1,3}(?:,\d{3})* – 1–3 digits then groups of ,###.`
- `(?:\.\d+)? – Optional decimal part.`

Example Inputs & Outputs

```
import re
text = "1,234 and -12,345.67 plus 42 and 3.14"
print(re.findall(r"-?\d{1,3}(?:,\d{3})*(?:\.\d+)?", text))
```

Key Takeaways

- `Place scientific branch first if combining with scientific notation.`
- `Strip '%' after matching if you’re collecting percents separately.`

Exercise 6 – Validate if a String is a Phone Number

Problem Statement

US■style validation with optional country code.

Final Regex Pattern

```
^(\+\d{1,3}[- ]?)?(\(\d{3}\)|\d{3})[- ]?\d{3}[- ]?\d{4}$
```

Step-by-Step Explanation

- ^…$ – Anchor to entire string.
- (\+\d{1,3}[-]?)? – Optional +CC.
- (\(\d{3}\)|\d{3}) – Area code with or without parentheses.
- \d{3}[-]?\d{4} – Local number.

Example Inputs & Outputs

```
import re
nums = ["+1-202-555-0143","(202) 555-0143","202 555 0143","5550143"]
pat = r"^(\+\d{1,3}[- ]?)?(\(\d{3}\)|\d{3})[- ]?\d{3}[- ]?\d{4}$"
print([bool(re.match(pat,n)) for n in nums])
```

Key Takeaways

- For international formats, use flexible blocks like (\d{1,4}[-]?){2,4}.

Exercise 7 – Find All Words Starting with ‘a’

Problem Statement

Match every word that starts with ‘a’ or ‘A’.

Final Regex Pattern

```
\ba\w*
```

Step-by-Step Explanation

- `\b` – Word boundary at start.
- `a` – First letter (use IGNORECASE to catch A/a).
- `\w*` – Rest of the word.

Example Inputs & Outputs

```
import re
text = "Alice and Bob are amazing artists at an Academy."
print(re.findall(r"\ba\w*", text, flags=re.IGNORECASE))
```

Key Takeaways

- Use re.IGNORECASE for both 'a' and 'A'.
- If letters only: \b[aA][a-zA-Z]*.

Exercise 8 – Replace Multiple Spaces with a Single Space

Problem Statement

Normalize whitespace by collapsing runs of spaces/tabs/newlines to one space.

Final Regex Pattern

```
\s+
```

Step-by-Step Explanation

- `\s+` – One or more whitespace characters.

Example Inputs & Outputs

```
import re
text = "This   is    spaced\t\t out\n badly."
print(re.sub(r"\s+", " ", text).strip())
```

Key Takeaways

- Use .strip() to also remove leading/trailing spaces.
- Use literal space class [ ]+ if you want to keep newlines.

Exercise 9 – Remove All Punctuation from Text

Problem Statement

Strip punctuation, keep letters, digits, underscores, and whitespace.

Final Regex Pattern

```
[^\w\s]
```

Step-by-Step Explanation

- `[^\w\s]` – Anything that is NOT a word char or whitespace.

Example Inputs & Outputs

```
import re
text = "Hello, world! #NLP @AI_2025 (v2)."
print(re.sub(r"[^\w\s]", "", text))
```

Key Takeaways

- Keep hashtags/mentions by excluding # and @ in the class: `[^\w\s#@]`.

Exercise 10 – Split a Paragraph into Sentences

Problem Statement

Split after ., ?, or ! without losing the punctuation.

Final Regex Pattern

```
(?<=[.?!])\s+
```

Step-by-Step Explanation

- `(?<=…)` – Positive lookbehind ensures split happens AFTER punctuation.
- `\s+` – Consume following spaces.

Example Inputs & Outputs

```
import re
text = "Hello world! How are you? I'm fine."
print(re.split(r"(?<=[.?!])\s+", text))
```

Key Takeaways

- Abbreviations need NLP tokenizers for perfect accuracy.

Exercise 11 – Find All Dates in Format DD/MM/YYYY

Problem Statement

Extract valid dates with day 01–31 and month 01–12.

Final Regex Pattern

```
\b(0[1-9]|[12]\d|3[01])/(0[1-9]|1[0-2])/\d{4}\b
```

Step-by-Step Explanation

- Day: (0[1-9]|[12]\d|3[01]) – 01–31.
- Month: (0[1-9]|1[0-2]) – 01–12.
- Year: \d{4}.
- \b boundaries to avoid partial matches.

Example Inputs & Outputs

```
import re
text = "Valid 01/01/2025 and 31/08/2020; invalid 32/01/2020."
pat = r"\b(0[1-9]|[12]\d|3[01])/(0[1-9]|1[0-2])/\d{4}\b"
print(re.findall(pat, text))
# full match alternative:
pat2 = r"\b(?:0[1-9]|[12]\d|3[01])/(?:0[1-9]|1[0-2])/\d{4}\b"
print(re.findall(pat2, text))
```

Key Takeaways

- Switch inner groups to (?:…) to get full strings from findall.
- Use finditer to extract full match while keeping capture groups.

Exercise 12 – Extract Domain Name from an Email

Problem Statement

Return only the domain (including TLD) from each email address.

Final Regex Pattern

```
(?:[\w\.-]+@)([\w\.-]+\.\w+)
```

Step-by-Step Explanation

- Non■capturing for username + @.
- Capture the domain + TLD in one group.

Example Inputs & Outputs

```
import re
text = "test.user@example.com admin@my-site.io sales@company.co.uk"
print(re.findall(r"(?:[\w\.-]+@)([\w\.-]+\.\w+)", text))
```

Key Takeaways

- If you only want the registrable domain, post■process (split by dots).

Exercise 13 – Replace All Digits with

Problem Statement

Replace every digit with the # character.

Final Regex Pattern

```
\d
```

Step-by-Step Explanation

- `\d` – Single digit. Use `\d+` to replace per number.

Example Inputs & Outputs

```
import re
text = "Order 123 will arrive in 4 days at 2025 Main St."
print(re.sub(r"\d", "#", text))
```

Key Takeaways

- `Use \d+ to collapse each numeric run to a single #.`

Exercise 14 – Find Duplicate Words in a Sentence

Problem Statement

Detect immediate word repetitions using backreferences.

Final Regex Pattern

```
\b(\w+)\s+\1\b
```

Step-by-Step Explanation

- (\\w+) – Capture a word.
- \\1 – Backreference to the same word.
- \\s+ – One or more spaces between repeats.

Example Inputs & Outputs

```
import re
text = "This is is fine. Very very good!"
print(re.findall(r"\b(\w+)\s+\1\b", text, flags=re.IGNORECASE))
```

Key Takeaways

- Use (?:\s+\1){1,} to catch 2+ repeats (chains).

Exercise 15 – Extract All Words Ending with ing

Problem Statement

Find every word that ends with 'ing'.

Final Regex Pattern

```
\b\w+ing\b
```

Step-by-Step Explanation

- `\\b – Start boundary.`
- `\\w+ – One or more word chars.`
- `ing – Literal suffix.`
- `\\b – End boundary.`

Example Inputs & Outputs

```
import re
text = "running, singing, learning, eating pudding."
print(re.findall(r"\b\w+ing\b", text))
```

Key Takeaways

- `Use \b\w{4,}ing\b to require at least 4 chars before 'ing'.`

Regex Mastery Quick Reference

```
\d  – Digit [0-9]
\w  – Word character [a-zA-Z0-9_]
\s  – Whitespace (space, tab, newline)
.    – Any character (except newline)
^ / $ – Start / End of string
\b / \B – Word boundary / Non-boundary
Quantifiers – +, *, ?, {n}, {n,}, {n,m}
Character classes – [...], [^...] (negated)
Groups – ( ) capturing, (?: ) non-capturing
Alternation – a|b
Lookahead – (?=...), (?!...)
Lookbehind – (?<=...), (?<!...)
Flags – re.IGNORECASE, re.MULTILINE, re.DOTALL, re.VERBOSE
```

Closing Note

You’ve completed Day 02 of your NLP Mastery journey. Every pattern you learned today is a precision instrument. Keep practicing, build tiny utilities, and apply these patterns in real NLP preprocessing pipelines. Onward to tokenization, normalization, and beyond.