# Week 2 – PRACTICAL

# Traditional Linguistic Methods: Regex, Tokenisation, Morphology and Segmentation

## Introduction

This practical introduces the traditional linguistic tools that still power many modern NLP pipelines. You will apply regular expressions, tokenisation, stemming and lemmatisation, and segmentation.

- Tasks are grouped by difficulty level so you can progress at your own pace.
- Everyone should complete Level 1 in class.
- Levels 2 can be continued during the online hour and later.
- Make use of the Regex Reference Sheets when necessary
- Install all necessary python libraries. Use Anaconda if necessary to maintain clean environment

## Tools Overview

- **`re` (Python Regular Expressions)** – pattern-based text cleaning and matching.
- **spaCy** – modern NLP library with efficient tokenisation.
- **NLTK (Natural Language Toolkit)** – classic NLP library for morphology and corpus tools.
- **Jieba** – Chinese word segmentation library.
- **Transformers tokenisers** – optional for token inflation study.

## Level 1 – Core Tasks (Build Foundations)

Run, observe and explain basic outputs.

Q1). Regex – Text Cleaning

Extract email addresses and hashtags from the sample text.

```
import re
text = "Email: jane.doe@uni.edu NLP AI 机器学习 <nlp@northampton.ac.uk>"
emails = re.findall(r"\b[\w\.-]+@[\w\.-]+\.\w+\b", text)
hashtags = re.findall(r"(?<!\w)#[\w]+", text)
print("Emails:", emails)
print("Hashtags:", hashtags)
```

- Observe the output, how many hashtags and emails where identified?
- Add new emails and hashtags. Re-run and observe output
- Reflection: What would happen if the text contained a Chinese hashtag or a multi-dot domain?

## Q2). Tokenisation – Observe

Tokenise a short sentence with spaCy and compare with NLTK.

```
# Installation
%pip install -U spacy nltk
import nltk
nltk.download("punkt_tab")
nltk.download("punkt")
```

```
# Load spaCy model
import spacy
from nltk.tokenize import word_tokenize
nlp = spacy.load("en_core_web_sm")

# Text to analyze
text = "Dr. Olu's AI-based model outperforms others in 2025."

# Tokenize with both libraries
print("spaCy:", [t.text for t in nlp(text)])
print("NLTK:", word_tokenize(text))
```

- Run the code and observe the difference between the Spacy and NLTK
- Question: Which tokenizer handled contractions or hyphens better?
- Replace the text with "He said, "Let's deploy version 2.0—no delays this time," before leaving at 5:45 p.m." Run and observe the difference

## Q3). Stemming and Lemmatisation – Run and Notice

Use NLTK to stem and lemmatise several words.

```
from nltk.stem import PorterStemmer, WordNetLemmatizer
import nltk; nltk.download("wordnet")
ps, wnl = PorterStemmer(), WordNetLemmatizer()

words = ["studies","studying","better","ate","flies"]
for w in words:
    print(w, "→", ps.stem(w), "→", wnl.lemmatize(w, "v"))
```

- Reflection: Which version keeps meaning better? When might the rougher one still be useful?

Q4). Segmentation – Observe

Run Jieba on a simple Chinese sentence.

```
#installation
%pip install jieba
```

```
import jieba
text = "我喜欢学习人工智能"
print(list(jieba.cut(text)))
```

Question: Why must languages like Chinese be segmented before tokenisation?

## 🟦 Level 2 – Extended Tasks (Apply and Compare)

### Extended Exercise

Q5). Regex – Extract Twitter Handles (Exclude Emails)

Write `extract_handles(text)` to find Twitter @handles without capturing the `@` inside email addresses.

```
import re

def extract_handles(text):
    # Match @ at start or after space, 1-15 letters/digits/underscore
    # Ignore @ inside emails
    pattern = r'(?:(?<=\s)|^)@[A-Za-z0-9_]{1,15}\b'
    return re.findall(pattern, text)

sample = """
Follow @nlp_lab and @Research_AI.
Email team@uni.edu or admin@example.com.
"""

print(extract_handles(sample))
```

Task:

- Change the text so a handle appears after punctuation (e.g. ".@user") – does it still work?
- What would you change to fix that?

Q6). Tokenisation – Compare Rule-based and Library

```
import re, spacy
nlp = spacy.load("en_core_web_sm")

def simple_tokenize(text):
    pattern = r"[A-Za-z]+(?:'[A-Za-z]+)?|[0-9]+|[^\w\s]"
    return re.findall(pattern, text)

text = "Dr. A. I. Jones co-authored a study, didn't he?"
print("Simple:", simple_tokenize(text))
print("spaCy :", [t.text for t in nlp(text)])
```

Task:

- Add one more rule (e.g. keep hyphenated words together) and re-run.
- Which version do you prefer for this sentence, and why?

Q7). Morphology – Stem vs Lemma

```
import nltk
from nltk.stem import PorterStemmer, WordNetLemmatizer
ps, wnl = PorterStemmer(), WordNetLemmatizer()

words = ["studies","studying","better","ate","flies"]
for w in words:
    print(w, "→", ps.stem(w), "→", wnl.lemmatize(w, "v"))
```

- Add two more verbs or nouns of your own.
- Which output keeps the meaning best?

Q8). Segmentation – Greedy vs Jieba

```
LEX = {"我","喜欢","学习","人工智能","自然语言","处理"}
def max_match(s, lex):
    out, i = [], 0
    while i < len(s):
        for j in range(len(s), i, -1):
            if s[i:j] in lex:
                out.append(s[i:j]); i = j; break
        else:
            out.append(s[i]); i += 1
    return out
```

```
import jieba
sent = "我喜欢学习人工智能"
print("Greedy:", max_match(sent, LEX))
print("Jieba :", list(jieba.cut(sent)))
```

Task:

- Add "自然语言处理" to LEX and compare again.
- When does greedy segmentation fail?

Q9). Mini Extension – Fairness and Tokens

```
%pip install transformers
```

```
from transformers import GPT2Tokenizer
tok = GPT2Tokenizer.from_pretrained("gpt2")
en = "I love natural language processing."
zh = "我喜欢自然语言处理。"
print("EN:", len(tok.tokenize(en)), "ZH:", len(tok.tokenize(zh)))
```

Question:

Why might different token counts across languages matter for AI cost or fairness?

Q10). Regex Creation and Decoding

a. Create a Regex that supports **Markdown links** [text](https://url)
b. Create a Regex that supports **Hex colours** #RGB or #RRGGBB
c. Decode the following Regex and suggest the kind of strings it would support
    i. \b(19|20)\d{2}[-/.](0[1-9]|1[0-2])[-/.](0[1-9]|[12]\d|3[01])\b
    ii. \b(?:[01]?\d|2[0-3]):[0-5]\d(?::[0-5]\d)?\b