**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5760 Natural Language Processing**

**Fall 2025**

**Homework 1.**

**Student name:**

**Venkata Nanda Krishna Yaram**

**700765514**

**Submission Requirements:**

* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Submit your GitHub link on the Bright Space.
* Comment your code appropriately ***IMPORTANT.***
* Any submission after provided deadline is considered as a late submission.

**Q1. Regex**

**Task:** Write a regex to find

1. **U.S. ZIP codes (disjunction + token boundaries)**  
   Match 12345 **or** 12345-6789 **or** 12345 6789 (hyphen **or** space allowed for the +4 part). Make sure you only match whole tokens (not inside longer strings).
2. **Negation in disjunction (word start rules)**  
   Find all **words** that **do not** start with a capital letter. Words may include internal apostrophes/hyphens like don’t, state-of-the-art.
3. **Convenient aliases (numbers, a bit richer)**  
   Extract all numbers that may have:
   1. optional sign (+/-),
   2. optional thousands separators (commas),
   3. optional decimal part,
   4. optional scientific notation (e.g., 1.23e-4).
4. **More disjunction (spelling variants)**  
   Match any spelling of “email”: email, e-mail, or e mail. Accept either a **space** or a **hyphen** (including en‐dash –) between e and mail, and be case-insensitive.
5. **Wildcards, optionality, repetition (with punctuation)**  
   Match the interjection go, goo, gooo, … (one or more o), **as a word**, and allow an optional trailing punctuation mark ! . , ? (e.g., gooo!).
6. **Anchors (line/sentence end with quotes)**  
   Match lines that **end with a question mark** possibly followed only by closing quotes/brackets like ")”’] and spaces.

**CODE**

**import re**

**# 1. U.S. ZIP codes**

**zip\_pattern = re.compile(r"\b\d{5}(?:[-\s]\d{4})?\b")**

**zip\_tests = ["12345", "12345-6789", "12345 6789", "123456", "abc12345"]**

**print("ZIP:", [m.group() for t in zip\_tests for m in zip\_pattern.finditer(t)])**

**# 2. Words not starting with a capital letter**

**word\_pattern = re.compile(r"\b(?A-Z])[A-Za-z]+(?:['’-][A-Za-z]+)\*\b")**

**word\_tests = ["dog", "Cat", "don’t", "state-of-the-art", "Hello"]**

**print("Non-capitalized words:", [m.group() for t in word\_tests for m in word\_pattern.finditer(t)])**

**# 3. Numbers (signs, commas, decimals, scientific notation)**

**num\_pattern = re.compile(r"[+-]?(?:\d{1,3}(?:,\d{3})\*|\d+)(?:\.\d+)?(?:[eE][+-]?\d+)?")**

**num\_tests = ["123", "+123.45", "-1,234", "1.23e-4", "12,345,678.90e+10"]**

**print("Numbers:", [m.group() for t in num\_tests for m in num\_pattern.finditer(t)])**

**# 4. Email spelling variants**

**email\_pattern = re.compile(r"(?i)\be[-\s–]?mail\b")**

**email\_tests = ["email", "E-mail", "e mail", "E–mail", "Mail"]**

**print("Email variants:", [m.group() for t in email\_tests for m in email\_pattern.finditer(t)])**

**# 5. Interjection go/goo/gooo... with optional punctuation**

**go\_pattern = re.compile(r"\bgo+[\!\.\,\?]?\b")**

**go\_tests = ["go", "goo", "gooo!", "go?", "gone", "gooo,"]**

**print("Go variants:", [m.group() for t in go\_tests for m in go\_pattern.finditer(t)])**

**# 6. Lines ending with question mark + optional closing symbols**

**q\_pattern = re.compile(r"\?[)"'\]\s]\*$")**

**q\_tests = [**

**"Is this working?",**

**"What time is it?\") ",**

**"Really?'",**

**"No way!",**

**]**

**print("Questions:", [t for t in q\_tests if q\_pattern.search(t)])**

**OUTPUT**

**![A screenshot of a computer program

AI-generated content may be incorrect.**

**Q2. Programming Question:**

**1. Tokenize a paragraph**  
Take a short paragraph (3–4 sentences) in **your language** (e.g., from news, a story, or social media).

* Do **naïve space-based tokenization**.
* Manually correct the tokens by handling punctuation, suffixes, and clitics.  
  Submit both versions and highlight differences.

**2. Compare with a Tool**  
Run the paragraph through an NLP tool that supports your language (e.g., **NLTK, spaCy**, or any open-source tokenizer if available).

* Compare tool output vs. your manual tokens.
* Which tokens differ? Why?

**3. Multiword Expressions (MWEs)**  
Identify at least **3 multiword expressions (MWEs)** in your language. Example:

* Place names, idioms, or common fixed phrases.
* Explain why they should be treated as single tokens.

**4. Reflection (5–6 sentences)**

* What was the hardest part of tokenization in your language?
* How does it compare with tokenization in English?
* Do punctuation, morphology, and MWEs make tokenization more difficult?

ANSWER

## Original Paragraph:

The quick brown fox jumps over the lazy dog.   
It’s a classic sentence used in typing practice.   
However, many people don’t realize its history.

## 1. Naïve Space-based Tokenization:

['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog.', 'It’s', 'a', 'classic', 'sentence', 'used', 'in', 'typing', 'practice.', 'However,', 'many', 'people', 'don’t', 'realize', 'its', 'history.']

## 1. Manually Corrected Tokens:

['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '.', 'It', '’s', 'a', 'classic', 'sentence', 'used', 'in', 'typing', 'practice', '.', 'However', ',', 'many', 'people', 'don’t', 'realize', 'its', 'history', '.']

## 2. spaCy Tool Tokens:

['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog', '.', 'It', '’s', 'a', 'classic', 'sentence', 'used', 'in', 'typing', 'practice', '.', 'However', ',', 'many', 'people', 'do', 'n’t', 'realize', 'its', 'history', '.']

## 2. Differences (Manual vs spaCy):

Manual: don’t | spaCy: do

Manual: realize | spaCy: n’t

Manual: its | spaCy: realize

Manual: history | spaCy: its

Manual: . | spaCy: history

Extra spaCy tokens: ['.']

## 3. Multiword Expressions (MWEs):

- 'New York City' should be treated as one token (meaning is lost if split).

- 'kick the bucket' should be treated as one token (meaning is lost if split).

- 'high school' should be treated as one token (meaning is lost if split).

## 4. Reflection:

The hardest part of tokenization in English was handling contractions like "don’t" or "it’s",   
since they may be split differently depending on the tool. Punctuation also introduces   
challenges, because naïve space-based methods attach punctuation marks to words. Compared   
to English, morphologically rich languages (like Turkish or Hindi) are harder, since suffixes   
and word forms require morphological analysis. Multiword expressions add another difficulty   
because their meaning disappears when split. Tools like spaCy generally perform well,   
but sometimes make tokenization choices that differ from manual expectations.

CODE

import spacy

# Load spaCy English model

nlp = spacy.load("en\_core\_web\_sm")

# Paragraph (3–4 sentences)

text = """The quick brown fox jumps over the lazy dog.

It’s a classic sentence used in typing practice.

However, many people don’t realize its history."""

print("=== Q2. Tokenization Assignment ===\n")

print("Original Paragraph:\n", text, "\n")

# ----------------------------------------------------------------------

# 1. Naïve space-based tokenization

naive\_tokens = text.split()

print("1. Naïve space-based tokens:")

print(naive\_tokens, "\n")

# Manually corrected tokens

manual\_tokens = [

"The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog", ".",

"It", "’s", "a", "classic", "sentence", "used", "in", "typing", "practice", ".",

"However", ",", "many", "people", "don’t", "realize", "its", "history", "."

]

print("1. Manually corrected tokens:")

print(manual\_tokens, "\n")

# ----------------------------------------------------------------------

# 2. Compare with spaCy tool

doc = nlp(text)

tool\_tokens = [token.text for token in doc]

print("2. spaCy tool tokens:")

print(tool\_tokens, "\n")

# Show differences clearly

print("2. Differences (manual vs spaCy):")

i, j = 0, 0

while i < len(manual\_tokens) and j < len(tool\_tokens):

if manual\_tokens[i] == tool\_tokens[j]:

i += 1

j += 1

else:

print(f" Manual: {manual\_tokens[i]} | spaCy: {tool\_tokens[j]}")

# advance both, because it's a misalignment

i += 1

j += 1

# If extra tokens remain

if i < len(manual\_tokens):

print(" Extra manual tokens:", manual\_tokens[i:])

if j < len(tool\_tokens):

print(" Extra spaCy tokens:", tool\_tokens[j:])

print()

# 3. Multiword Expressions (MWEs)

MWEs = [

"New York City", # place name

"kick the bucket", # idiom

"high school" # fixed phrase

]

print("3. Multiword Expressions (MWEs):")

for expr in MWEs:

print(f"- '{expr}' should be treated as one token (meaning is lost if split).")

print()

# 4. Reflection

reflection = """

4. Reflection:

The hardest part of tokenization in English was handling contractions like "don’t" or "it’s",

since they may be split differently depending on the tool. Punctuation also introduces

challenges, because naïve space-based methods attach punctuation marks to words. Compared

to English, morphologically rich languages (like Turkish or Hindi) are harder, since suffixes

and word forms require morphological analysis. Multiword expressions add another difficulty

because their meaning disappears when split. Tools like spaCy generally perform well,

but sometimes make tokenization choices that differ from manual expectations.

"""

print(reflection)

OUTPUT

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Q3. Manual BPE on a toy corpus**

3.1 Using the same corpus from class:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

1. Add the end-of-word marker \_ and write the *initial vocabulary* (characters + \_).
2. Compute bigram counts and perform the **first three merges** by hand:
   * Step 1: most frequent pair → merge → updated corpus snippet (show at least 2 lines).
   * Step 2: repeat.
   * Step 3: repeat.
3. After each merge, list the new token and the updated vocabulary.

3.2 — Code a mini-BPE learner

1. Use the classroom code above (or your own) to learn BPE merges for the toy corpus.
   * Print the top pair at each step and the evolving vocabulary size.
2. Segment the words: new, newer, lowest, widest, and one word you invent (e.g., newestest).
   * Include the subword sequence produced (tokens with \_ where applicable).
3. In 5–6 sentences, explain:
   * How subword tokens solved the OOV (out-of-vocabulary) problem.
   * One example where subwords align with a meaningful morpheme (e.g., er\_ as English agent/comparative suffix).

3.3 — *Your language* (or English if you prefer)

Pick one short paragraph (4–6 sentences) in *your own language* (or English if that’s simpler).

1. Train BPE on that paragraph (or a small file of your choice).
   * Use end-of-word \_.
   * Learn at least 30 merges (adjust if the text is very small).
2. Show the five most frequent merges and the resulting five longest subword tokens.
3. Segment 5 different words from the paragraph:
   * Include one rare word and one derived/inflected form.
4. Brief reflection (5–8 sentences):
   * What kinds of subwords were learned (prefixes, suffixes, stems, whole words)?
   * Two concrete pros/cons of subword tokenization for your language

ANSWER

Submitted in github

**Q4. Word Pair:**

Sunday → Saturday

Tasks:

1. Find the minimum edit distance between *Sunday* and *Saturday* under both models:
   * Model A (Sub = 1, Ins = 1, Del = 1)
   * Model B (Sub = 2, Ins = 1, Del = 1)
2. Write out at least one valid edit sequence (step by step).
3. Compute the **first 3 rows and first 4 columns** of the Levenshtein DP table (sub=2) for:

* S1 = SUNDAY (rows)
* S2 = SATURDAY (cols)

Write down **D(0,0)…D(3,4)** and the value of **D(3,4)** specifically

1. Reflect (4–5 sentences):
   * Did both models give the same distance?
   * Which operations (insert/delete/substitute) were most useful here?
   * How would the choice of model affect applications like spell check vs. DNA alignment?

Hint: The words differ by only a couple of characters, but pay attention to the placement of “Sat-” in *Saturday*

ANSWER

Submitted in github