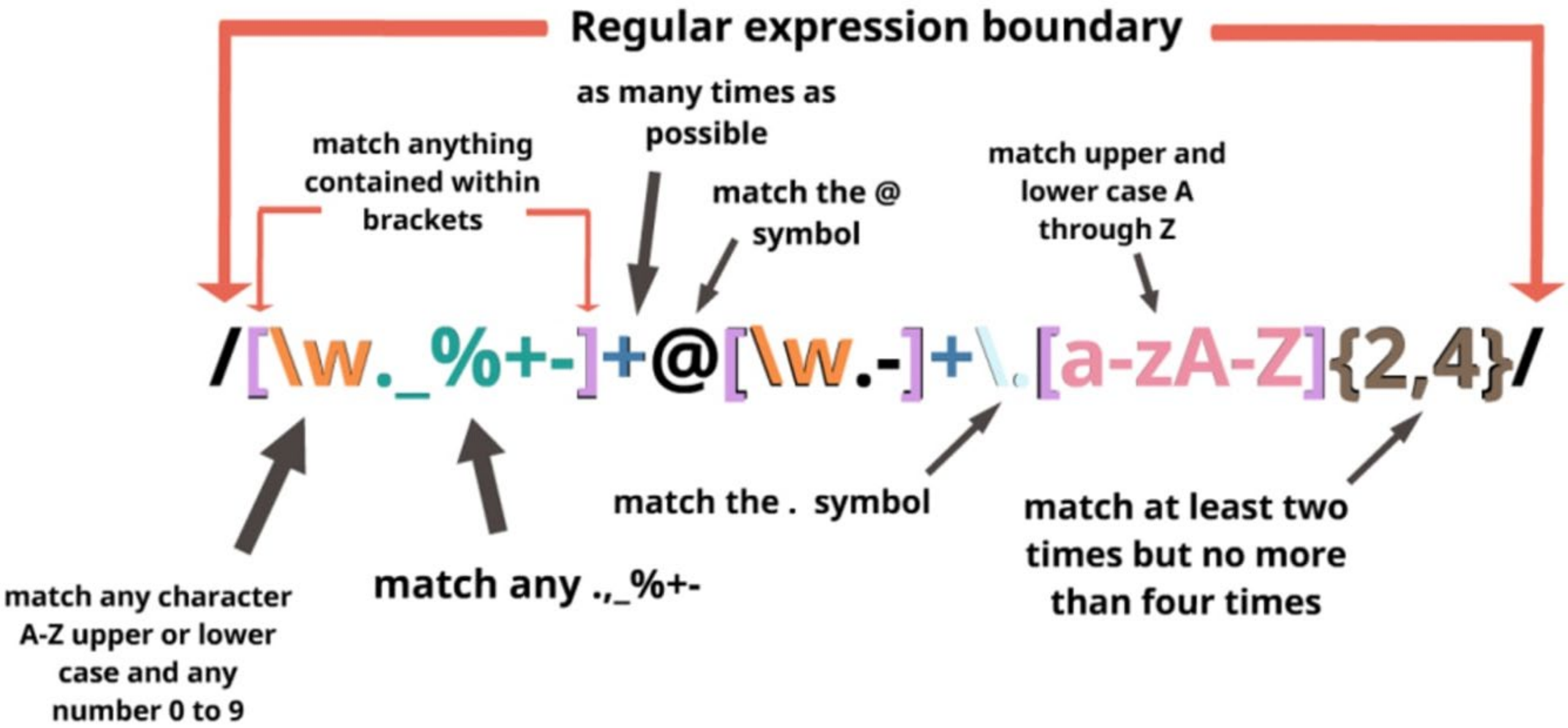
Chapter 2

Regular Expressions and Word Tokenization

Outline

- Introduction to regular expressions (**regex**)
- Which pattern?
- Practicing regular expressions: re.**split**() and re.**findall**()
- Introduction to tokenization
- Word tokenization with **NLTK**
- More regex with re.**search**()
- Advanced tokenization with NLTK and regex
- Choosing a tokenizer
- Regex with NLTK tokenization
- Non-ascii tokenization
- Charting word length with NLTK
- Charting practice



Regular expression boundary

as many times as possible

match anything contained within brackets

match the @ symbol

match upper and lower case A through Z

```
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
```

match the . symbol

match at least two times but no more than four times

match any character A-Z upper or lower case and any number 0 to 9

match any .,_%+-

Introduction to regular expressions

What are regular expressions?

- Regular expressions (**Regex**) are strings with a **special syntax**.
- Regex allow us to **match** patterns in other strings.
- **Applications** of Regex
  - Find all web links in a document
  - Parse email addresses
  - Remove/Replace unwanted characters

```
import re
re.match('abc','abcdef’)
```

```
<re.Match object; span=(0, 3), match='abc’>
```

```
word_regex = '\w+'
re.match(word_regex, 'hi there!’)
```

```
<re.Match object; span=(0, 2), match='hi'>
```

Common regex patterns

- A **pattern** is a series of letters/symbols which can **map** to an actual text/words/punctuation.
- Examples

Pattern	Matches	Example
\w+	Word	‘Magic’
\d	Digit	9
\s	Space	‘ ’
\S	Not space	‘no_spaces’
[a-z]	Lowercase group	‘abc’

- The wildcard will match ANY letter or symbol
- The + and * characters allow things to become greedy, grabbing repeats of single letters or whole patterns.

More information: https://docs.python.org/3/library/re.html

re module

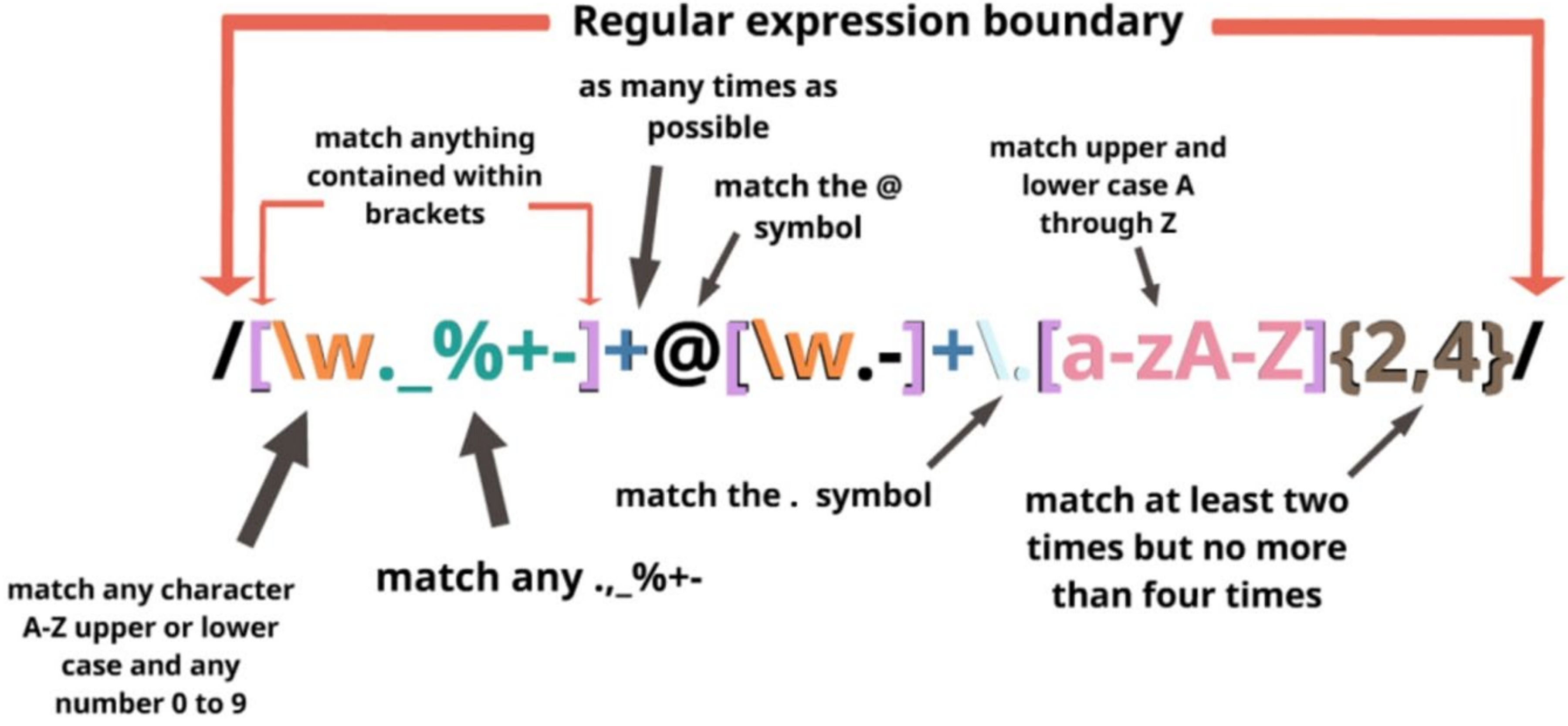
- re module
  - split: split a string on regex
  - findall: find all patterns in a string
  - search: search for a pattern
  - match: match an entire string or substring based on a pattern

```
re.split('\s+', 'Split on spaces.')
```

```
['Split', 'on', 'spaces.']
```

```
word_regex2 = 'spaces!'
print(re.findall(word_regex2,'Split on spaces!'))
print(re.search(word_regex2,'Split on spaces!'))
print(re.match(word_regex2,'Split on spaces!'))
```

```
['spaces']
<re.Match object; span=(9, 15), match='spaces'>
None
```


Regular expression boundary
as many times as possible
match anything contained within brackets
match the @ symbol
match upper and lower case A through Z
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
match the . symbol
match at least two times but no more than four times
match any character A-Z upper or lower case and any number 0 to 9
match any .,_%+-

Let’s practice!

Which pattern?

- Which of the following Regex patterns results in the following text?

```
my_string = "Let's write RegEx!"
re.findall(PATTERN, my_string)
```

```
['Let', 's', 'write', 'RegEx']
```

- Possible Answers
  a) PATTERN = r"\s+“
  b) PATTERN = r"\w+“
  c) PATTERN = r"[a-z]“
  d) PATTERN = r"\w"

re.split() and re.findall()

- The regular expression module re which is imported from you
- Set `my_string = "Let's write RegEx!  Won't that be fun?  I sure think so.  Can you find 4 sentences?  Or perhaps, all 19 words?"`
- Note: It's important to prefix your regex patterns with r to ensure that your patterns are interpreted in the way you want them to. Else, you may encounter problems to do with **escape sequences** in strings.
  - For example, **"\n"** in Python is used to indicate a new line, but if you use the r prefix, it will be interpreted as the raw string "\n" - that is, the character "\" followed by the character **"n"** - and not as a new line.

re.split() and re.findall()

- Split **my_string** on each sentence ending. To do this:
  - Write a pattern called **sentence_endings** to match sentence endings (**.?!**).
  - Use **re.split()** to split **my_string** on the pattern and print the result.

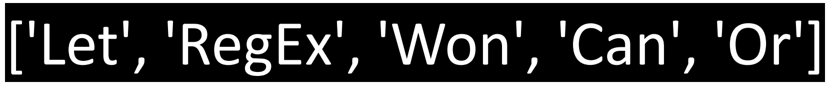
```
# Write a pattern to match sentence endings: sentence_endings
sentence_endings = r"[___]“

# Split my_string on sentence endings and print the result
print(re.____(____, ____))
```

```
["Let's write RegEx", "  Won't that be fun", '  I sure think so', '  Can you find 4 sentences',
'  Or perhaps, all 19 words', '']
```

re.split() and re.findall()

- Find and print all capitalized words in **my_string** by writing a pattern called **capitalized_words** and using **re.findall()**.
  - Remember the **[a-z]** pattern to match lowercase groups? Modify that pattern appropriately in order to match uppercase groups.

```
# Find all capitalized words in my_string and print the result
capitalized_words = r"___\w+"
print(re. ___(___, ___))
```

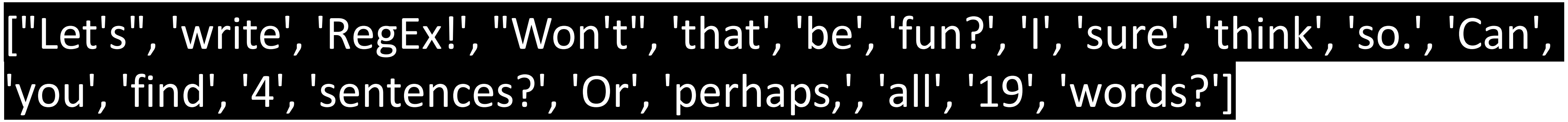


```
['Let', 'RegEx', 'Won', 'Can', 'Or']
```

re.split() and re.findall()

- Write a pattern called **spaces** to match one or more spaces **("\s+")** and then use **re.split()** to split **my_string** on this pattern, keeping all punctuation intact. Print the result.

```
# Split my_string on spaces and print the result
spaces = r"___"
print(re. ___(___, ___))
```

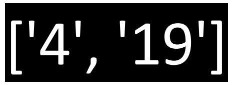


```
["Let's", 'write', 'RegEx!', "Won't", 'that', 'be', 'fun?', 'I', 'sure', 'think', 'so.', 'Can',
'you', 'find', '4', 'sentences?', 'Or', 'perhaps,', 'all', '19', 'words?']
```

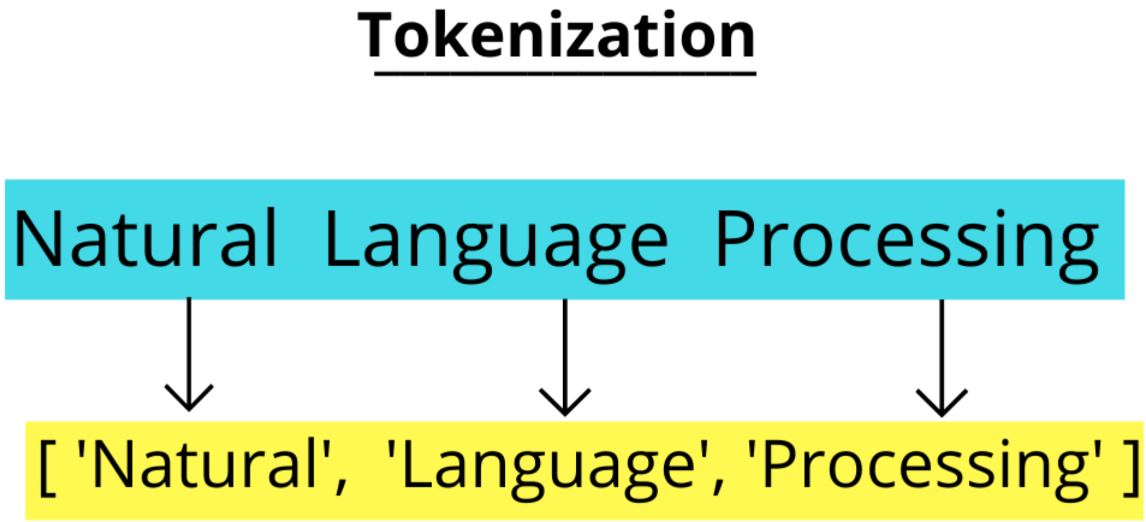
re.split() and re.findall()

- Find all digits in **my_string** by writing a pattern called digits **("\d+")** and using **re.findall()**. Print the result.

```
# Find all digits in my_string and print the result
digits = r"___"
print(re.findall(___, ___))
```



```
['4', '19']
```



Tokenization

Natural Language Processing

['Natural', 'Language', 'Processing']

Introduction to tokenization

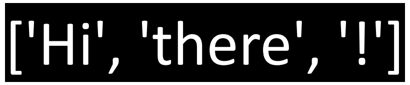
What is tokenization?

- Turning a string or document into **tokens** (smaller chunks).
- One step for preprocessing a text in NLP
- Many different theories and rules of tokenization, so, everyone can create their rules using Regex.
- Examples:
    - break out words or sentences
    - separate punctuation
    - separating all hashtags in a Tweet

nltk library

- nltk: natural language toolkit
  - py –m pip install nltk
- example of using the **word_tokenize** method to break down a string into tokens.

```python
from nltk.tokenize import word_tokenize
word_tokenize("Hi there!")
```



```
['Hi', 'there', '!']
```

Why tokenize?

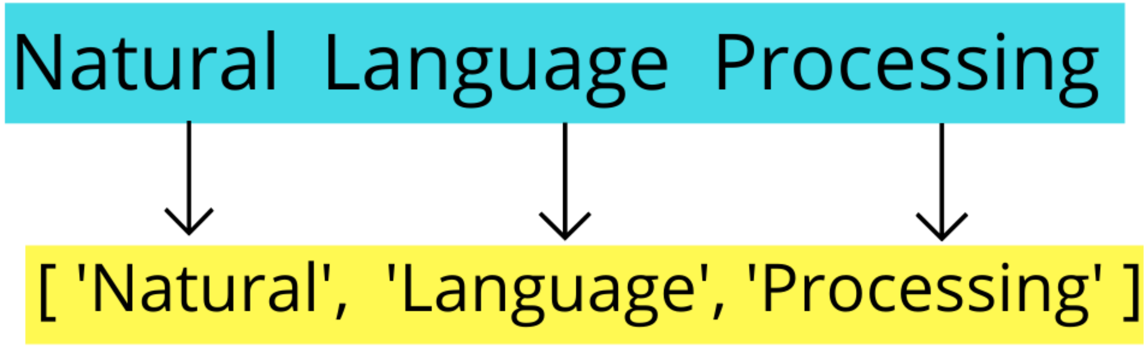
- Easier to map part of speech
- Matching common words
- Removing unwanted tokens
- Example:
  - The sentence is: `"I don't like Sam's shoes."`
  - Result after tokenization is:

```
['I', 'do', "n't", 'like', 'Sam', "'s", 'shoes', '.']
```

Other nltk tokenizers

- sent_tokenize: tokenize a document into sentences
- regexp_tokenize: tokenize a string or document based on a regular expression pattern
- TweetTokenizer: special class just for tweet tokenization, allowing you to separate hashtags, mentions and lots of exclamation points (!!!).

Tokenization



Natural Language Processing

['Natural', 'Language', 'Processing']

Let’s practice!

Word tokenization with NLTK

- Utilize **word_tokenize** and **sent_tokenize** from **nltk.tokenize** to tokenize both words and sentences from Python strings.
- Import the **sent_tokenize** and **word_tokenize** functions from **nltk.tokenize**

```
# Import necessary modules
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
```

- Read txt file and keep in **scene_one**

```
#Read TXT file
f = open("scene_one.txt", "r")
scene_one = f.read()
```

Word tokenization with NLTK

- Tokenize all sentences in **scene_one** using the **sent_tokenize()** function.

```
# Split scene_one into sentences: sentences
sentences = ___________(______)
```

- Tokenize the fourth sentence in **sentences**, which you can access as **sentences[3]**, using the **word_tokenize()** function.

```
# Use word_tokenize to tokenize the fourth sentence:
tokenized_sent
tokenized_sent = ___________(_______)
```

Word tokenization with NLTK

- Find the unique tokens in the entire scene by using **word_tokenize()** on **scene_one** and then converting it into a set using **set()**.

```
# Make a set of unique tokens in the entire scene: unique_tokens
unique_tokens = _____(_________(_______))
```

- Print the unique tokens found.

```
# Print the unique tokens result
print(unique_tokens)
```

More regex with re.search()

- Utilize **re.search()** and **re.match()** to find specific tokens.
- Use **re.search()** to search for the first occurrence of the word "coconuts" in **scene_one**. Store the result in match.

```
# Search for the first occurrence of "coconuts" in
scene_one: match
match = re.________(_________, scene_one)
```

- Print the start and end indexes of **match** using its **.start()** and **.end()** methods, respectively.

```
# Print the start and end indexes of match
print(____________, ____________)
```

More regex with re.search()

- Write a regular expression called **pattern1** to find anything in square brackets.

```
# Write a regular expression to search for anything in
square brackets: pattern1
pattern1 = r"\[____\]"
```

- Use **re.search()** with the pattern to find the first text in **scene_one** in square brackets in the scene. Print the result.

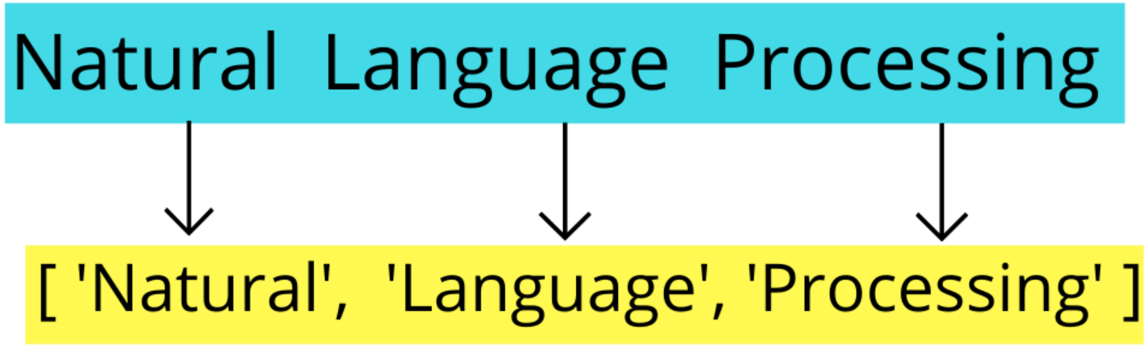
```
# Use re.search to find the first text in square brackets
print(re.________(_________, __________))
```

More regex with re.search()

- Create a pattern to match the script notation (e.g. **Character:**), assigning the result to **pattern2**. Remember that you will want to match any words or spaces that precede the **:** (such as the space within **SOLDIER #1:**).
- Use **re.match()** with your new pattern to find and print the script notation in the fourth line. The tokenized sentences are available in your namespace as **sentences**.

```
# Find the script notation at the beginning of the fourth
sentence and print it
pattern2 = r"[____]+:"
print(re._____(_________, ________[3]))
```

Tokenization



Natural Language Processing

['Natural', 'Language', 'Processing']

Advanced tokenization with NLTK and regex

Regex groups using or "|"

- **OR** method is represented using **|**
- Define a group using **()**
- Define explicit character ranges using **[]**

```
import re
match_digits_and_words = ('(\d+|\w+)')
re.findall(match_digits_and_words, 'He has 11 cats')
```

```
['He', 'has', '11', 'cats']
```

Regex ranges and groups

- Examples

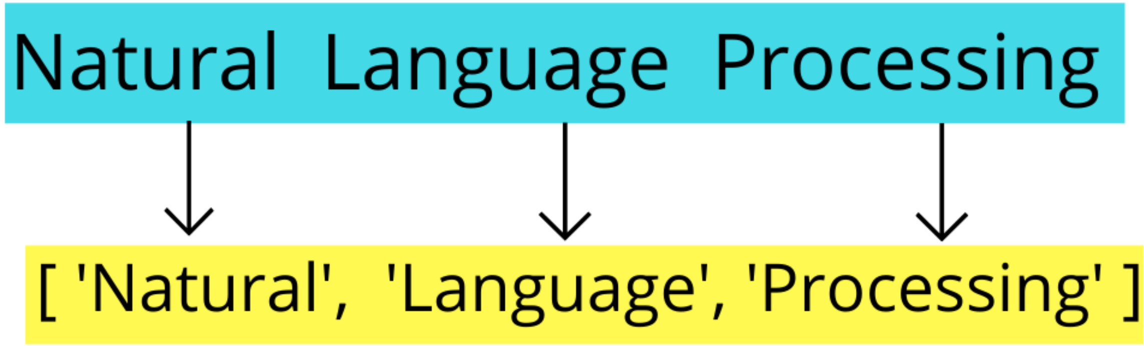
Pattern	Matches	Example
[A-Za-z]+	Upper and lowercase English alphabet	‘ABCDEFghijk”
[0-9]	Number from 0-9	9
[A-Za-z\-\.]+	Upper and lowercase English alphabet, - and .	‘My-Website.com’
(a-z)	a, - and z	‘a-z’
(\s+|,)	Spaces or a comma	‘, ’

Character range with `re.match()’

```python
import re
my_str = 'match lowercase spaces nums like 12, but no commas’
match_str = ('[a-z0-9 ]+')
print(re.match(match_str, my_str))
```

```
<re.Match object; span=(0, 35), match='match lowercase spaces nums like 12'>
```

Tokenization



Natural Language Processing

['Natural', 'Language', 'Processing']

Let’s practice!

Choosing a tokenizer

```
['SOLDIER', '#1', 'Found', 'them', '?', 'In', 'Mercea', '?', 'The',
'coconut', 's', 'tropical', '!']
```

```python
my_string = "SOLDIER #1: Found them? In Mercea? The coconut's tropical!"
```

- Given the following string, which of the below patterns is the best tokenizer? If possible, you want to retain sentence punctuation as separate tokens, but have '#1' remain a single token.
- The string is available in your workspace as **my_string**, and the patterns have been pre-loaded as **pattern1**, **pattern2**, **pattern3**, and **pattern4**, respectively.
- Additionally, **regexp_tokenize** has been imported from **nltk.tokenize**. You can use **regexp_tokenize(string, pattern)** with **my_string** and one of the patterns as arguments to experiment for yourself and see which is the best tokenizer.

Possible Answers

a) r"(\w+|\?|!)"

b) r"(\w+|#\d|\?|!)"

c) r"(#\d\w+\?!)"

d) r"\s+"

Regex with NLTK tokenization

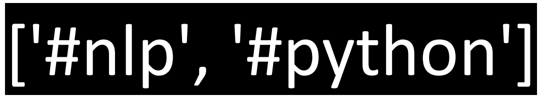
- Twitter is a frequently used source for NLP text and tasks. In this exercise, you'll build a more complex tokenizer for tweets with hashtags and mentions using **nltk** and regex. The **nltk.tokenize.TweetTokenizer** class gives you some extra methods and attributes for parsing tweets.
- From **nltk.tokenize**, import **regexp_tokenize** and **TweetTokenizer** .

```
# Import the necessary modules
from nltk.tokenize import ______________
from nltk.tokenize import ______________
```

Regex with NLTK tokenization

- Set `tweets = ['This is the #nlp exercise! #python', '#NLP is super fun! <3 #learning', 'Thanks @fitmkmutnb :) #nlp #python']`
- A regex pattern to define hashtags called **pattern1** has been defined for you. Call **regexp_tokenize()** with this hashtag pattern on the first tweet in **tweets** and assign the result to **hashtags**.
- Print **hashtags**

```
# Define a regex pattern to find hashtags: pattern1
pattern1 = r"#\w+"
# Use the pattern on the first tweet in the tweets list
hashtags = ______________(________[_], _________)
print(hashtags)
```



```
['#nlp', '#python']
```

Regex with NLTK tokenization

- Write a new pattern called **pattern2** to match mentions and hashtags. A mention is something like **@fitmkmutnb**.
- Then, call **regexp_tokenize()** with your new hashtag pattern on the **last** tweet in **tweets** and assign the result to **mentions_hashtags**.
  - You can access the last element of a list using **-1** as the index, for example, **tweets[-1]**.
- Print **mentions_hashtags**

```
# Write a pattern that matches both mentions (@) and hashtags
pattern2 = r"([__]\w+)"
# Use the pattern on the last tweet in the tweets list
mentions_hashtags = ___________(______[__], _________)
print(mentions_hashtags)
```

```
['@fitmkmutnb', '#nlp', '#python']
```

Regex with NLTK tokenization

- Create an instance of **TweetTokenizer** called **tknzr** and use it inside a list comprehension to tokenize each tweet into a new list called **all_tokens**.
- To do this, use the **.tokenize()** method of **tknzr**, with **t** as your iterator variable.
- Print **all_tokens** .

```
# Use the TweetTokenizer to tokenize all tweets into one list
tknzr = _______________()
all_tokens = [_______._________(_) for t in tweets]
print(all_tokens)
```

```
[['This', 'is', 'the', '#nlp', 'exercise', '!', '#python'], ['#NLP', 'is', 'super', 'fun', '!', '<3',
'#learning'], ['Thanks', '@fitmkmutnb', ':)', '#nlp', '#python']]
```

Non-ascii tokenization

- Practice advanced tokenization by tokenizing some non-ascii based text.
- The following modules have been pre-imported from **nltk.tokenize**: **regexp_tokenize** and **word_tokenize**.

```
from nltk.tokenize import regexp_tokenize
from nltk.tokenize import word_tokenize
```

- Set `german_text = "Wann gehen wir Pizza essen? 🍕 Und fährst du mit Über? 🚕"`
- Unicode ranges for emoji are:
  - ('\U0001F300'-'\U0001F5FF'), ('\U0001F600-\U0001F64F'), ('\U0001F680-\U0001F6FF'), and ('\u2600'-\u26FF-\u2700-\u27BF').

Non-ascii tokenization

- Tokenize all the words in **german_text** using **word_tokenize()**, and print the result.

```
# Tokenize and print all words in german_text
all_words = ______________(___________)
print(all_words)
```

```
['Wann', 'gehen', 'wir', 'Pizza', 'essen', '?', '🍕', 'Und', 'fährst', 'du', 'mit', 'Über', '?', '🚕']
```

Non-ascii tokenization

- Tokenize only the capital words in **german_text** .
  - First, write a pattern called **capital_words** to match only capital words. Make sure to check for the German **Ü** ! To use this character in the exercise, copy and paste it from these instructions.
  - Then, tokenize it using **regexp_tokenize()** .

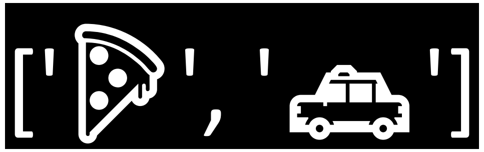
```
# Tokenize and print only capital words
capital_words = r"[___]\w+"
print(____________(__________, __________))
```

```
['Wann', 'Pizza', 'Und', 'Über']
```

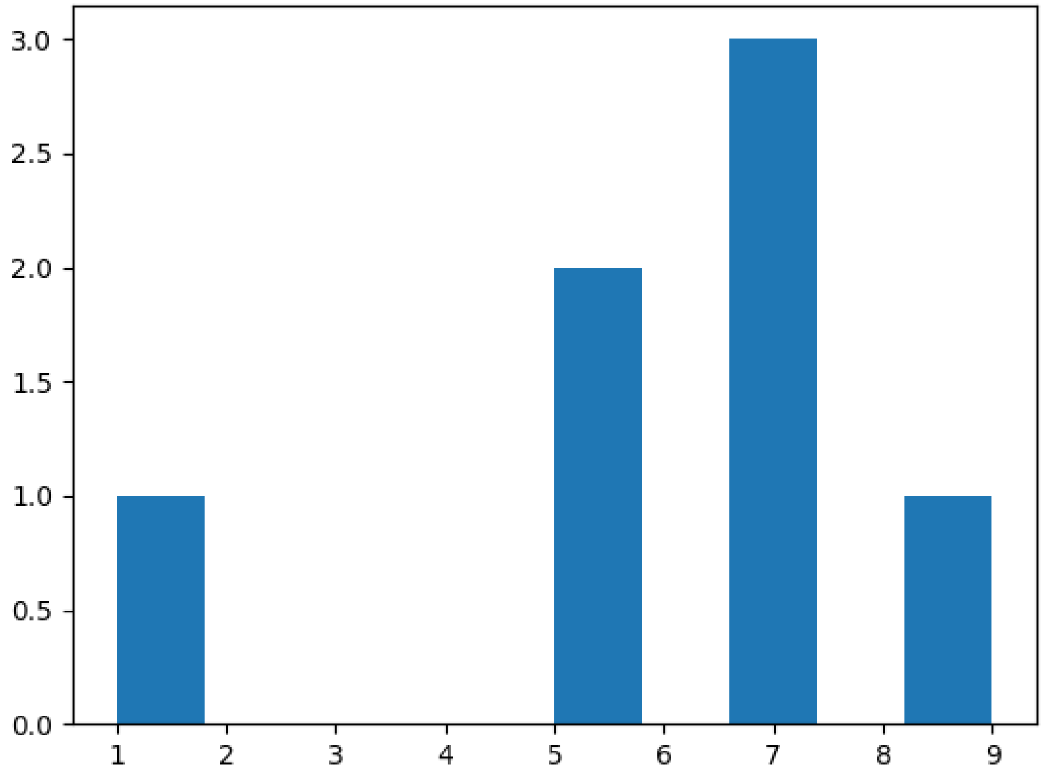
Non-ascii tokenization

- Tokenize only the emoji in **german_text**. The pattern using the unicode ranges for emoji given in the assignment text has been written for you. Your job is to use **regexp_tokenize()** to tokenize the emoji.

```
# Tokenize and print only emoji
emoji = "['\U0001F300-\U0001F5FF'|'\U0001F600-
\U0001F64F'|'\U0001F680-\U0001F6FF'|'\u2600-
\u26FF\u2700-\u27BF']"
print(_____________(_____________, _______))
```



```
['🍕', '🚕']
```



Charting word length with nltk

Getting started with **matplotlib**

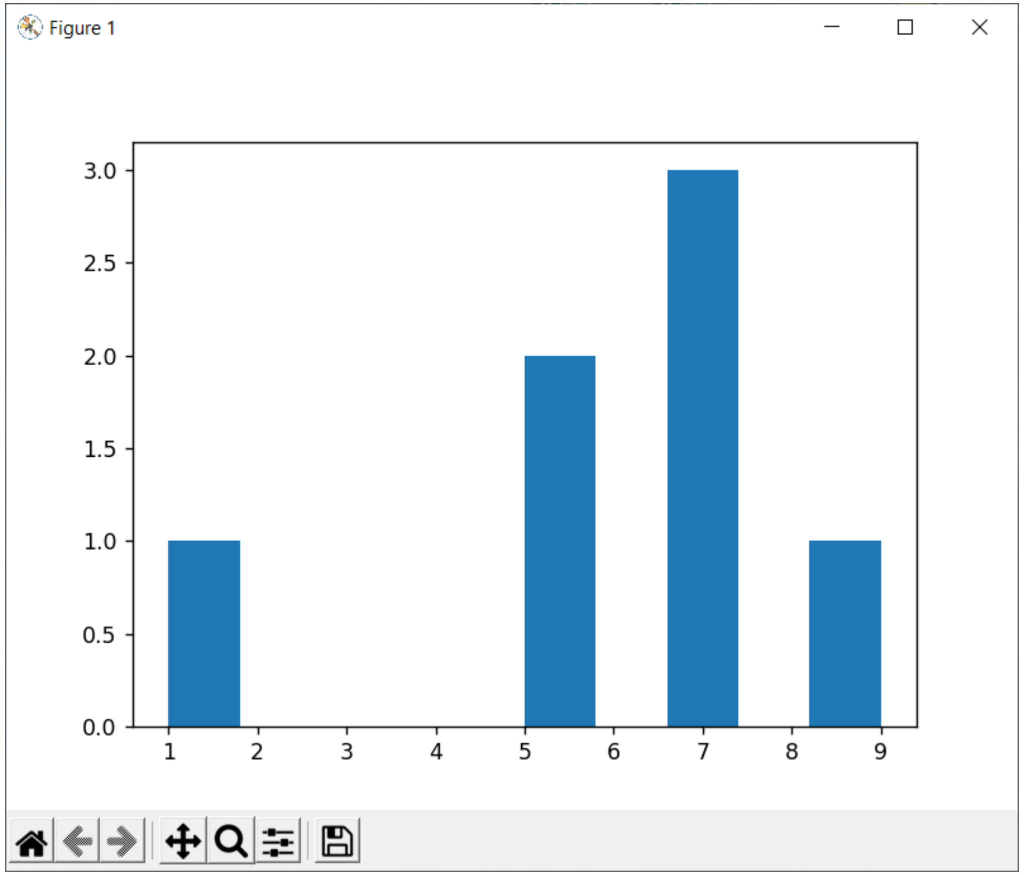
- Matplotlib is a charting library used by many open-source Python projects
- Straightforward functionality with lots of options
  - histograms,
  - bar charts,
  - line charts and
  - scatter plots.
- And also advanced functionality like generating 3D graphs and animations.

Plotting a histogram with matplotlib

Install **matplotlib**:

```
pip install matplotlib
```

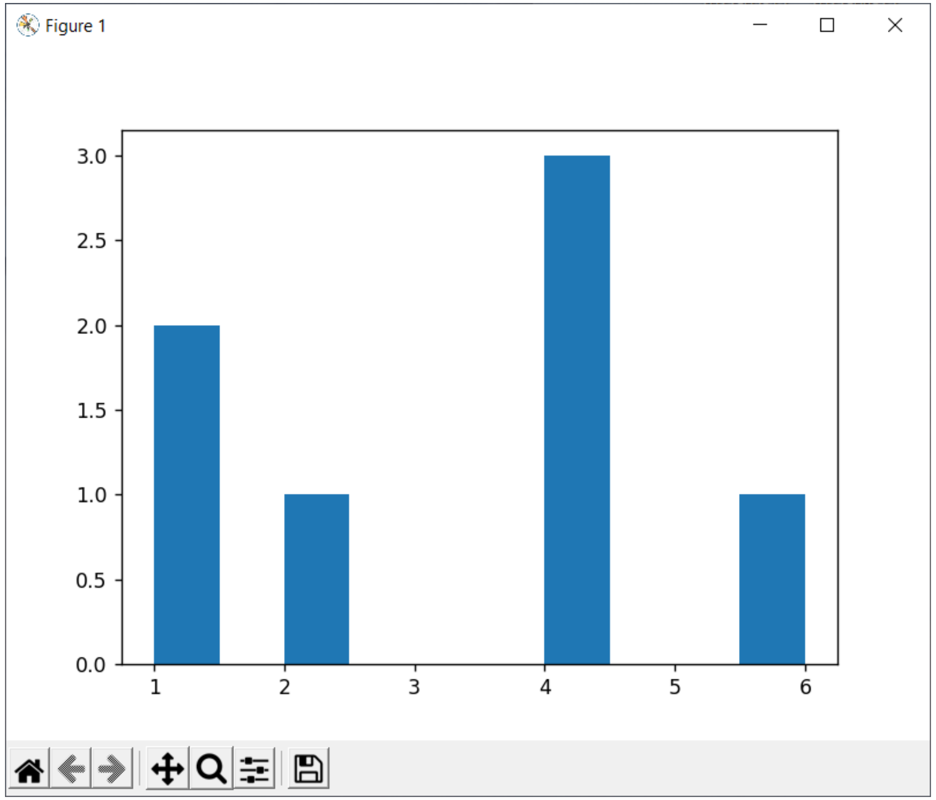
```
from matplotlib import pyplot as plt
plt.hist([1, 5, 5, 7, 7, 7, 9])

plt.show()
```



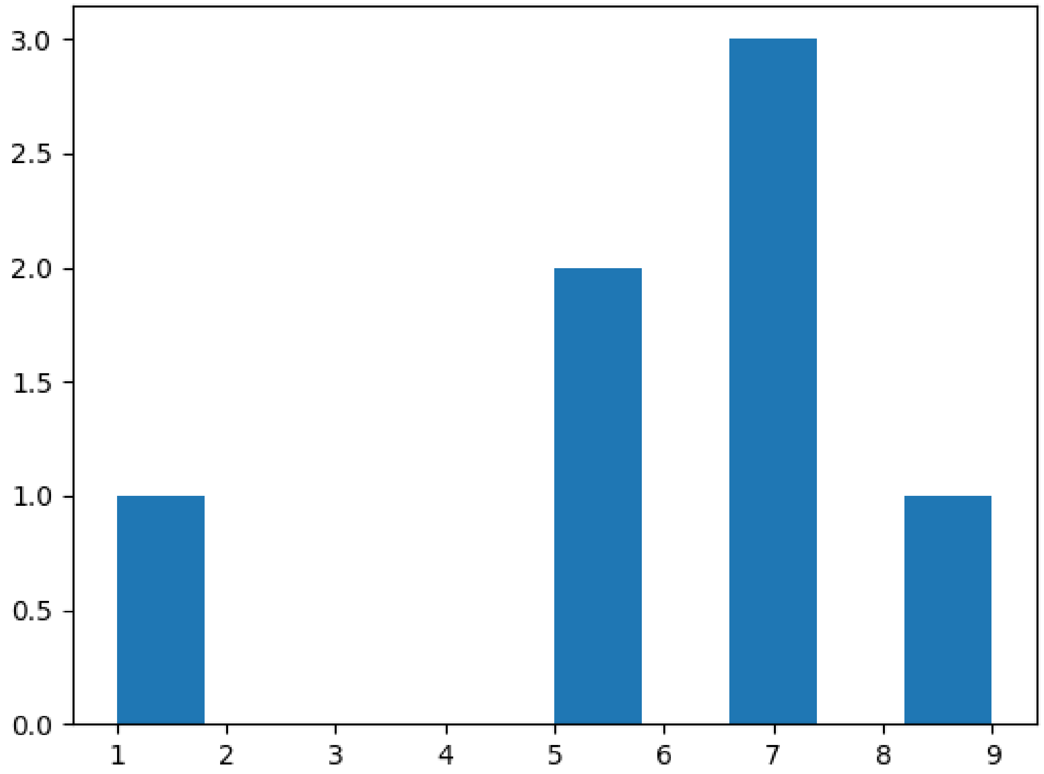
Generated histogram

Combining NLP data extraction with plotting

```
from matplotlib import pyplot as plt
from nltk.tokenize import word_tokenize

words = word_tokenize("This is a pretty
cool tool!")
word_lengths = [len(w) for w in words]
plt.hist(word_lengths)

plt.show()
```



Word length histogram



Let’s practice!

Charting practice

- Find and chart the number of words per line in the script using **matplotlib**.
- Read txt file (holy_grail.txt) and keep in **holy_grail**

```
#Read TXT file
f = open("holy_grail.txt", "r")
holy_grail = f.read()
```

- Split the script **holy_grail** into lines using the newline (**'\n'**) character.

```
# Split the script into lines: lines
lines = __________.________('\n')
```

Charting practice

- Use **re.sub()** inside a list comprehension to replace the prompts such as **ARTHUR:** and **SOLDIER #1**. The pattern has been written for you.
- Use a list comprehension to tokenize **lines** with **regexp_tokenize()**, keeping only words. Recall that the pattern for words is "**\w+**".

```
# Replace all script lines for speaker
pattern = "[A-Z]{2,}(\s)?(#\d)?([A-Z]{2,})?:"
lines = [re.____(_________, '', l) for l in lines]

# Tokenize each line: tokenized_lines
tokenized_lines = [_____________(___,______) for s in lines]
```

Charting practice

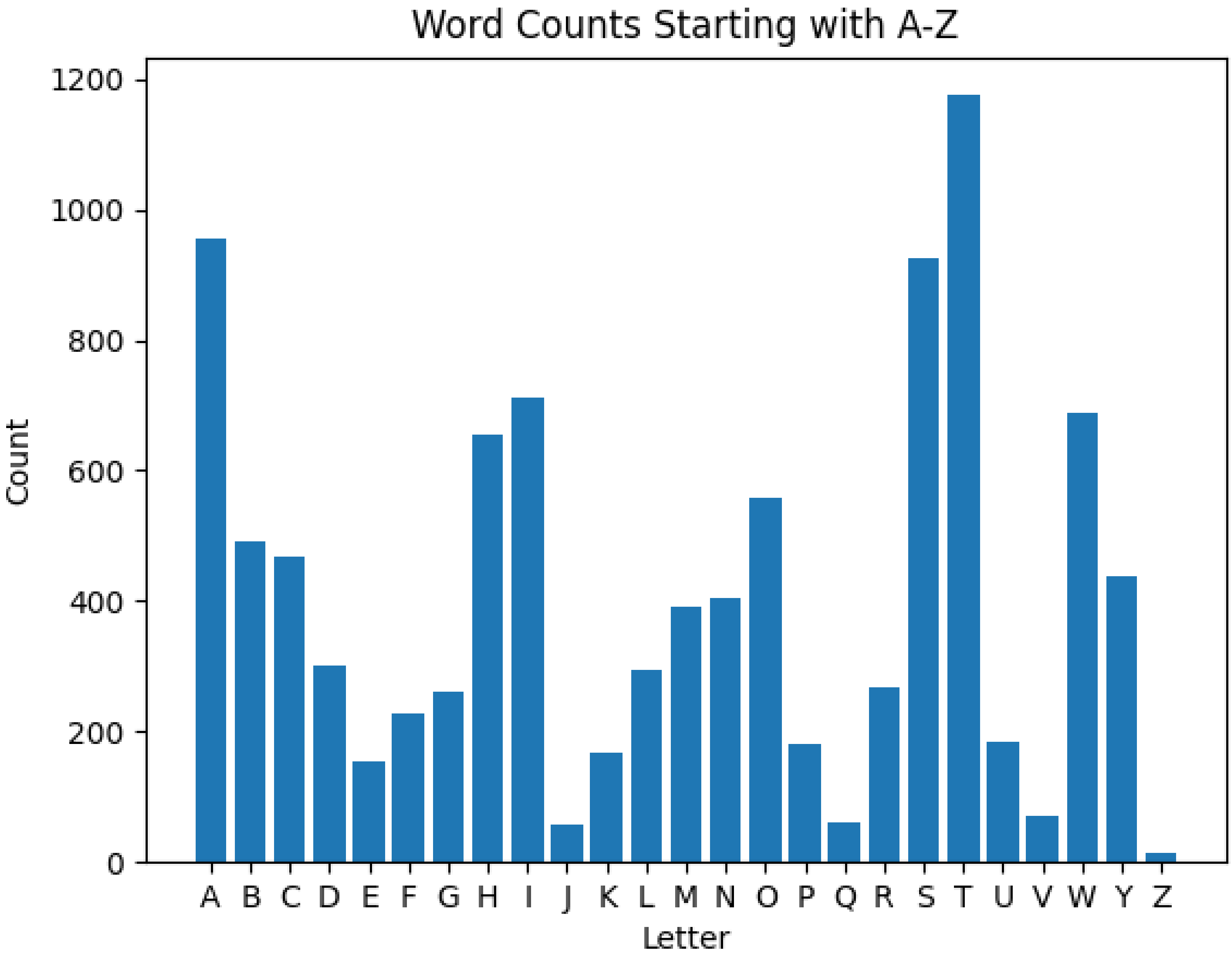
- Use a list comprehension to create a list of line lengths called **line_num_words**.
  - Use **t_line** as your iterator variable to iterate over **tokenized_lines**, and then **len()** function to compute line lengths.

```
# Make a frequency list of lengths: line_num_words
line_num_words = [_____(_____) for t_line in tokenized_lines]
```

- Plot a histogram of **line_num_words** using **plt.hist()**. Don't forgot to use **plt.show()** as well to display the plot.

```
# Plot a histogram of the line lengths
____._______(______________)
# Show the plot
____._______()
```

Work


Word Counts Starting with A-Z
Count
1200
1000
800
600
400
200
0
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
Y
Z
Letter

Questions

Reference: https://app.datacamp.com/learn