Regular expressions & word tokenization

LaTeX, Puteaux, 2020, 2021

```
##############################################
##                                          ##
##  Natural Language Processing in Python   ##
##                                          ##
##############################################
```

```
§1 Introduction to Natural Language Processing in Python
```

```
§1.1 Regular expressions & word tokenization
```

1 Introduction to regular expressions

1.1 What is Natural Language Processing?

- The field of study Natural Language Processing (NLP) focused on making sense of language using statistics and computers.
- The basics of NLP include:
  - *topic identification*
  - *text classification*
- NLP applications include:
  - *chatbots*
  - *translation*
  - *sentiment analysis*
  - *and many more*

1.2 What exactly are regular expressions?

- Strings with a special syntax
- Allow matching patterns in other strings, e.g.,
  - *find all web links in a document*
  - *parse email addresses*
  - *remove/replace unwanted characters*

1.3 Code of the applications of regular expressions:

[1]:
```
import re

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

[1]:
```
<re.Match object; span=(0, 3), match='abc'>
```

[2]:
```
word_regex = '\w+'
re.match(word_regex, 'hi there!')
```

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

1.4 What are the common regex patterns?

pattern	matches	example
\w+	word	'Magic'
\d	digit	9
\s	space	' '
.*	wildcard	'username74'
+ or *	greedy match	'aaaaaa'
\S	**not** space	'no_spaces'
[a-z]	lowercase group	'abcdefg'

1.5 How to use Python's `re` module?

- `re` module:
  - `split`: split a string on regex
  - `findall`: find all pa erns in a string
  - `search`: search for a pattern
  - `match`: match an entire string or substring based on a pattern
- Parameterize the pattern first and parameterize the string second.
- May return an iterator, string, or match object.

1.6 Code of Python's `re` module:

[3]:
```
re.split('\s+', 'Split on spaces.')
```

[3]:
```
['Split', 'on', 'spaces.']
```

1.7 Practice question for finding out the corresponding 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']
```

☐ PATTERN = r"\s+".

☒ PATTERN = r"\w+".

☐ PATTERN = r"[a-z]".

☐ PATTERN = r"\w".

▶ Package pre-loading:

[4]:
```
import re
```

▶ Data pre-loading:

[5]:
```
my_string = "Let's write RegEx!"
```

▶ Question-solving method:

[6]:
```
PATTERN = r"\s+"
re.findall(PATTERN, my_string)
```

[6]:
```
[' ', ' ']
```

[7]:
```
PATTERN = r"\w+"
re.findall(PATTERN, my_string)
```

[7]:
```
['Let', 's', 'write', 'RegEx']
```

[8]:
```
PATTERN = r"[a-z]"
re.findall(PATTERN, my_string)
```

[8]:
```
['e', 't', 's', 'w', 'r', 'i', 't', 'e', 'e', 'g', 'x']
```

[9]:
```
PATTERN = r"\w"
re.findall(PATTERN, my_string)
```

[9]:
```
['L', 'e', 't', 's', 'w', 'r', 'i', 't', 'e', 'R', 'e', 'g', 'E', 'x']
```

1.8 Practice exercises for introduction to regular expressions:

▶ Package pre-loading:

[10]:
```
import re
```

▶ **Data pre-loading:**

[11]:
```
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?"
```

▶ **Regular expressions (`re.split()` and `re.findall()`) practice:**

[12]:
```
# Write a pattern to match sentence endings: sentence_endings
sentence_endings = r"[\.\?!]"

# Split my_string on sentence endings and print the result
print(re.split(sentence_endings, my_string))

# Find all capitalized words in my_string and print the result
capitalized_words = r"[A-Z]\w+"
print(re.findall(capitalized_words, my_string))

# Split my_string on spaces and print the result
spaces = r"\s+"
print(re.split(spaces, my_string))

# Find all digits in my_string and print the result
digits = r"\d+"
print(re.findall(digits, 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', '']
['Let', 'RegEx', 'Won', 'Can', 'Or']
["Let's", 'write', 'RegEx!', "Won't", 'that', 'be', 'fun?', 'I', 'sure',
'think', 'so.', 'Can', 'you', 'find', '4', 'sentences?', 'Or', 'perhaps,',
'all', '19', 'words?']
['4', '19']
```

1.9 Version checking:

[13]:
```
import sys

print('The Python version is {}.'.format(sys.version.split()[0]))
```

```
The Python version is 3.7.9.
```

```
#############################################
##                                         ##
##  Natural Language Processing in Python  ##
##                                         ##
#############################################

§1 Introduction to Natural Language Processing in Python
```

§1.1 Regular expressions & word tokenization

2 Introduction to tokenization

2.1 What is tokenization?

- It turns a string or document into tokens (smaller chunks).
- It's one step in preparing a text for NLP.
- It has many different theories and rules.
- Users can create their own rules using regular expressions.
- There are some examples:
  - *breaking out words or sentences*
  - *separating punctuation*
  - *separating all hashtags in a tweet*

2.2 What is the NLTK library?

- NLTK: Natural Language Toolkit

2.3 Code of the NLTK library:

[14]:
```
from nltk.tokenize import word_tokenize

word_tokenize("Hi there!")
```

[14]:
```
['Hi', 'there', '!']
```

2.4 Why tokenize?

- Easier to map part of speech.
- To match common words.
- To remove unwanted tokens.
- E.g.,

```
>>> word_tokenize("I don't like Sam's shoes.")
['I', 'do', "n't", 'like', 'Sam', "'s", 'shoes', '.']
```

2.5 What are the 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 separate hashtags, mentions, and lots of exclamation points, such as '!!!'.

2.6 Code of regex practice (the difference between `re.search()` and `re.match()`):

[15]:
```
import re

re.match('abc', 'abcde')
```

[15]:
```
<re.Match object; span=(0, 3), match='abc'>
```

[16]:
```
re.search('abc', 'abcde')
```

[16]:
```
<re.Match object; span=(0, 3), match='abc'>
```

[17]:
```
re.match('cd', 'abcde')
```

[18]:
```
re.search('cd', 'abcde')
```

[18]:
```
<re.Match object; span=(2, 4), match='cd'>
```

2.7 Practice exercises for introduction to tokenization:

▶ Data pre-loading:

[19]:
```
scene_one = open("ref2. Scene 1 of Monty Python and the Holy Grail.txt").read()
```

▶ NLTK word tokenization with practice:

[20]:
```
# Import necessary modules
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize

# Split scene_one into sentences: sentences
sentences = sent_tokenize(scene_one)

# Use word_tokenize to tokenize the fourth sentence: tokenized_sent
tokenized_sent = word_tokenize(sentences[3])

# Make a set of unique tokens in the entire scene: unique_tokens
unique_tokens = set(word_tokenize(scene_one))

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

```
{'velocity', 'England', 'if', 'pound', 'coconuts', 'coconut', 'temperate',
'held', 'horse', 'back', 'are', 'must', "'d", 'zone', 'Pendragon', 'join',
'two', 'could', 'carry', 'Will', 'in', 'five', 'Patsy', 'all', 'and', 'bangin',
'it', 'its', 'course', 'What', 'King', 'Supposing', '[', 'since', 'wants',
'servant', 'master', 'air-speed', 'But', 'creeper', 'grip', 'other', 'swallow',
'here', 'So', 'warmer', 'climes', 'question', 'every', 'tropical', 'not',
```

```
'Listen', 'Found', 'have', "'ve", 'Where', 'will', 'anyway', 'Camelot', 'is',
'Whoa', 'by', 'with', 'yeah', 'grips', 'Well', 'Halt', 'order', 'one', 'ounce',
'guiding', 'the', 'found', 'on', 'wings', 'interested', 'clop', 'south',
'these', 'strangers', 'SOLDIER', ']', 'Arthur', 'simple', 'non-migratory',
'bring', 'using', 'land', 'martin', 'snows', 'he', 'just', 'sun', 'who', 'Who',
'house', 'suggesting', 'Court', 'got', 'migrate', 'It', 'son', 'search',
'Please', 'or', 'second', 'Uther', '…', 'halves', 'maybe', 'line', 'right',
'?', 'ARTHUR', 'I', 'strand', "n't", 'winter', 'may', 'minute', 'Wait', 'am',
'lord', 'European', 'together', '--', 'husk', 'plover', 'length', 'yet', 'We',
'A', 'of', 'carrying', 'speak', "'", 'SCENE', 'that', 'a', 'forty-three', ',',
'fly', 'you', '2', 'ask', '.', 'kingdom', 'The', 'there', 'mean', "'re", 'tell',
'wind', 'breadth', 'Oh', ':', 'beat', 'swallows', 'Britons', 'needs', 'matter',
'be', 'court', 'trusty', 'ratios', 'You', 'my', 'but', 'agree', 'they', 'Pull',
'through', 'go', 'Not', 'then', 'why', 'seek', "'m", '!', 'this', 'does',
'dorsal', 'use', 'feathers', 'Mercea', 'KING', 'our', 'Saxons', 'Am', 'Ridden',
'No', "'s", 'times', 'They', 'That', 'covered', 'get', 'point', 'to',
'maintain', 'from', 'me', 'Yes', 'them', 'ridden', 'do', 'African', 'goes', '#',
'carried', 'weight', 'sovereign', 'your', "'em", 'bird', '1', 'In', 'castle',
'under', 'empty', 'where', 'an', 'defeator', 'at', 'knights', 'Are'}
```

▶ Package pre-loading:

[21]:
```
import re
```

▶ Regex (re.search()) practice:

[22]:
```
# Search for the first occurrence of "coconuts" in scene_one: match
match = re.search("coconuts", scene_one)

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

```
580 588
```

[23]:
```
# Write a regular expression to search for anything in square brackets: pattern1
pattern1 = r"\[.*\]"

# Use re.search to find the first text in square brackets
print(re.search(pattern1, scene_one))
```

```
<re.Match object; span=(9, 32), match='[wind] [clop clop clop]'>
```

[24]:
```
# Find the script notation at the beginning of the fourth sentence and print it
pattern2 = r"[\w\s#]+:"
print(re.match(pattern2, sentences[3]))
```

```
<re.Match object; span=(0, 7), match='ARTHUR:'>
```

2.8 Version checking:

[25]:
```
import sys
import nltk

print('The Python version is {}.'.format(sys.version.split()[0]))
print('The NLTK version is {}.'.format(nltk.__version__))
```

```
The Python version is 3.7.9.
The NLTK version is 3.5.
```

```
##############################################
##                                          ##
##  Natural Language Processing in Python   ##
##                                          ##
##############################################
```

```
§1 Introduction to Natural Language Processing in Python

§1.1 Regular expressions & word tokenization
```

3 Advanced tokenization with NLTK and regex

3.1 How to make regex groups and how to indicate "OR"?

- "OR" is represented using `|`.
- It is possible to define a group using `()`.
- It is possible to define explicit character ranges using `[]`.

3.2 Code of regex groups and the indication of "OR":

[26]:
```
import re

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

[26]:
```
['He', 'has', '11', 'cats']
```

3.3 What are regex ranges and groups?

pattern	matches	example
[A-Za-z]+	upper and lowercase English alphabet	'ABCDEFghijk'
[0-9]	numbers from 0 to 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	', '

3.4 Code of character range with `re.match()`:

[27]:
```
my_str = 'match lowercase spaces nums like 12, but no commas'
re.match('[a-z0-9 ]+', my_str)
```

[27]:
```
<re.Match object; span=(0, 35), match='match lowercase spaces nums like 12'>
```

3.5 Practice question for choosing a tokenizer:

- Given the following string, which of the below patterns is the best tokenizer? It is better to retain sentence punctuation as separate tokens if possible but have `'#1'` remain a single token.

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

  ☐ `r"(\w+|\?|!)"`.

  ☒ `r"(\w+|#\d|\?|!)"`.

  ☐ `r"(#\d\w+\?!)"`.

  ☐ `r"\s+"`.

▶ Package pre-loading:

[28]:
```
from nltk.tokenize import regexp_tokenize
```

▶ Data pre-loading:

[29]:
```
my_string = "SOLDIER #1: Found them? In Mercea? The coconut's tropical!"
string = my_string

pattern1 = r"(\w+|\?|!)"
pattern2 = r"(\w+|#\d|\?|!)"
pattern3 = r"(#\d\w+\?!)"
pattern4 = r"\s+"
```

▶ Question-solving method:

[30]:
```
pattern = pattern1
print(regexp_tokenize(string, pattern))
```

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

[31]:
```
pattern = pattern2
print(regexp_tokenize(string, pattern))
```

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

[32]:
```
pattern = pattern3
print(regexp_tokenize(string, pattern))
```

```
[]
```

[33]:
```
pattern = pattern4
print(regexp_tokenize(string, pattern))
```

```
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
```

3.6 Practice exercises for advanced tokenization with NLTK and regex:

▶ **Data pre-loading:**

[34]:
```
tweets = [
    'This is the best #nlp exercise ive found online! #python',
    '#NLP is super fun! <3 #learning', 'Thanks @datacamp :) #nlp #python'
]
```

▶ **NLTK regex tokenization practice:**

[35]:
```
# Import the necessary modules
from nltk.tokenize import TweetTokenizer
from nltk.tokenize import regexp_tokenize
```

[36]:
```
# Import the necessary modules
from nltk.tokenize import regexp_tokenize
from nltk.tokenize import TweetTokenizer
# Define a regex pattern to find hashtags: pattern1
pattern1 = r"#\w+"
# Use the pattern on the first tweet in the tweets list
hashtags = regexp_tokenize(tweets[0], pattern1)
print(hashtags)
```

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

[37]:
```
# Import the necessary modules
from nltk.tokenize import regexp_tokenize
from nltk.tokenize import TweetTokenizer
# 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 = regexp_tokenize(tweets[-1], pattern2)
print(mentions_hashtags)
```

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

[38]:
```
# Import the necessary modules
from nltk.tokenize import regexp_tokenize
from nltk.tokenize import TweetTokenizer
# Use the TweetTokenizer to tokenize all tweets into one list
tknzr = TweetTokenizer()
all_tokens = [tknzr.tokenize(t) for t in tweets]
print(all_tokens)
```

```
[['This', 'is', 'the', 'best', '#nlp', 'exercise', 'ive', 'found', 'online',
'!', '#python'], ['#NLP', 'is', 'super', 'fun', '!', '<3', '#learning'],
['Thanks', '@datacamp', ':)', '#nlp', '#python']]
```

▶ Package pre-loading:

[39]:
```
from nltk.tokenize import word_tokenize
```

▶ Data re-pre-loading:

[40]:
```
german_text = 'Wann gehen wir Pizza essen?  Und fährst du mit Über? '
```

▶ Non-ASCII tokenization practice:

[41]:
```
# Tokenize and print all words in german_text
all_words = word_tokenize(german_text)
print(all_words)

# Tokenize and print only capital words
capital_words = r"[A-ZÜ]\w+"
print(regexp_tokenize(german_text, capital_words))

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

print(regexp_tokenize(german_text, emoji))
```

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

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

3.7 Version checking:

[42]:
```
import sys
import nltk

print('The Python version is {}.'.format(sys.version.split()[0]))
print('The NLTK version is {}.'.format(nltk.__version__))
```

```
The Python version is 3.7.9.
The NLTK version is 3.5.
```

```
##############################################
##                                          ##
##  Natural Language Processing in Python   ##
##                                          ##
##############################################

§1 Introduction to Natural Language Processing in Python

§1.1 Regular expressions & word tokenization
```

4 Charting word length with NLTK

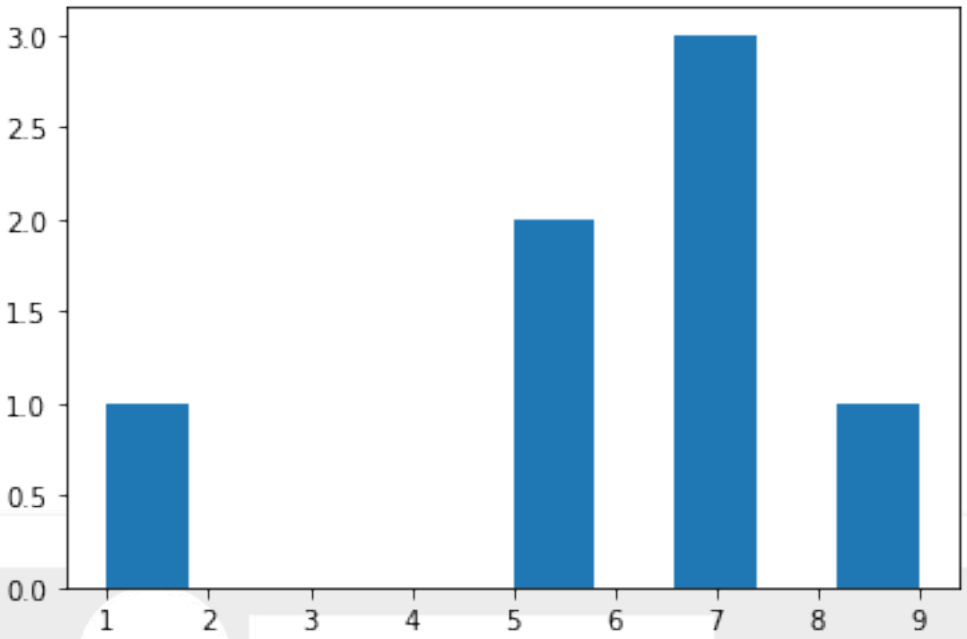
4.1 Why is it in need to get started with Matplotlib?

- It is a charting library used by many open-source Python projects.
- It has straightforward functionality with lots of options:
  - *histograms*
  - *bar charts*
  - *line charts*
  - *scatter plots*
- And also, it has advanced functionality like 3D graphs and animations!

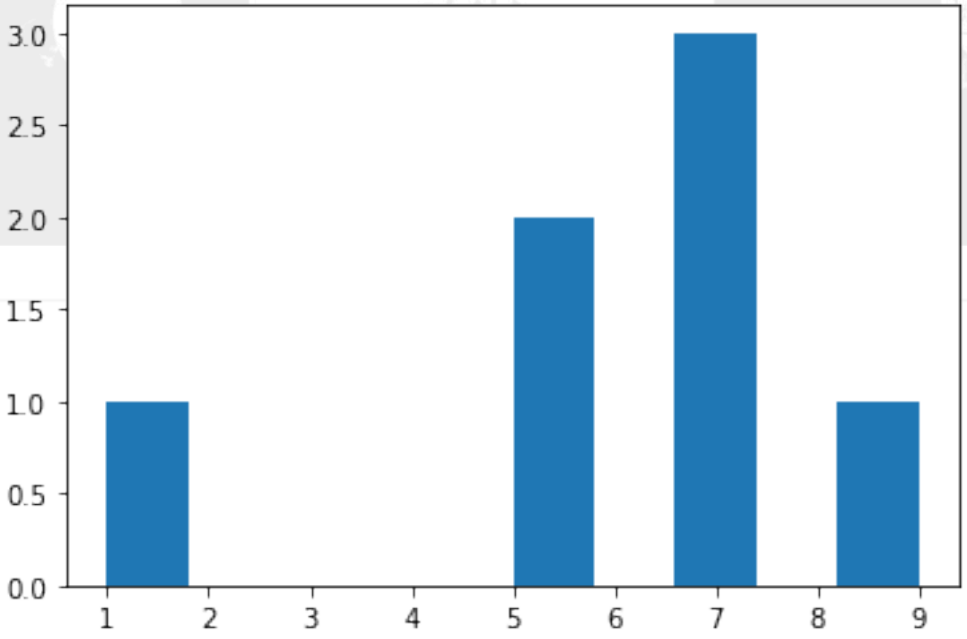
4.2 Code of plotting a histogram with Matplotlib:

[43]:
```
from matplotlib import pyplot as plt

plt.hist([1, 5, 5, 7, 7, 7, 9])
```

[43]:
```
(array([1., 0., 0., 0., 0., 2., 0., 3., 0., 1.]),
 array([1. , 1.8, 2.6, 3.4, 4.2, 5. , 5.8, 6.6, 7.4, 8.2, 9. ]),
 <BarContainer object of 10 artists>)
```



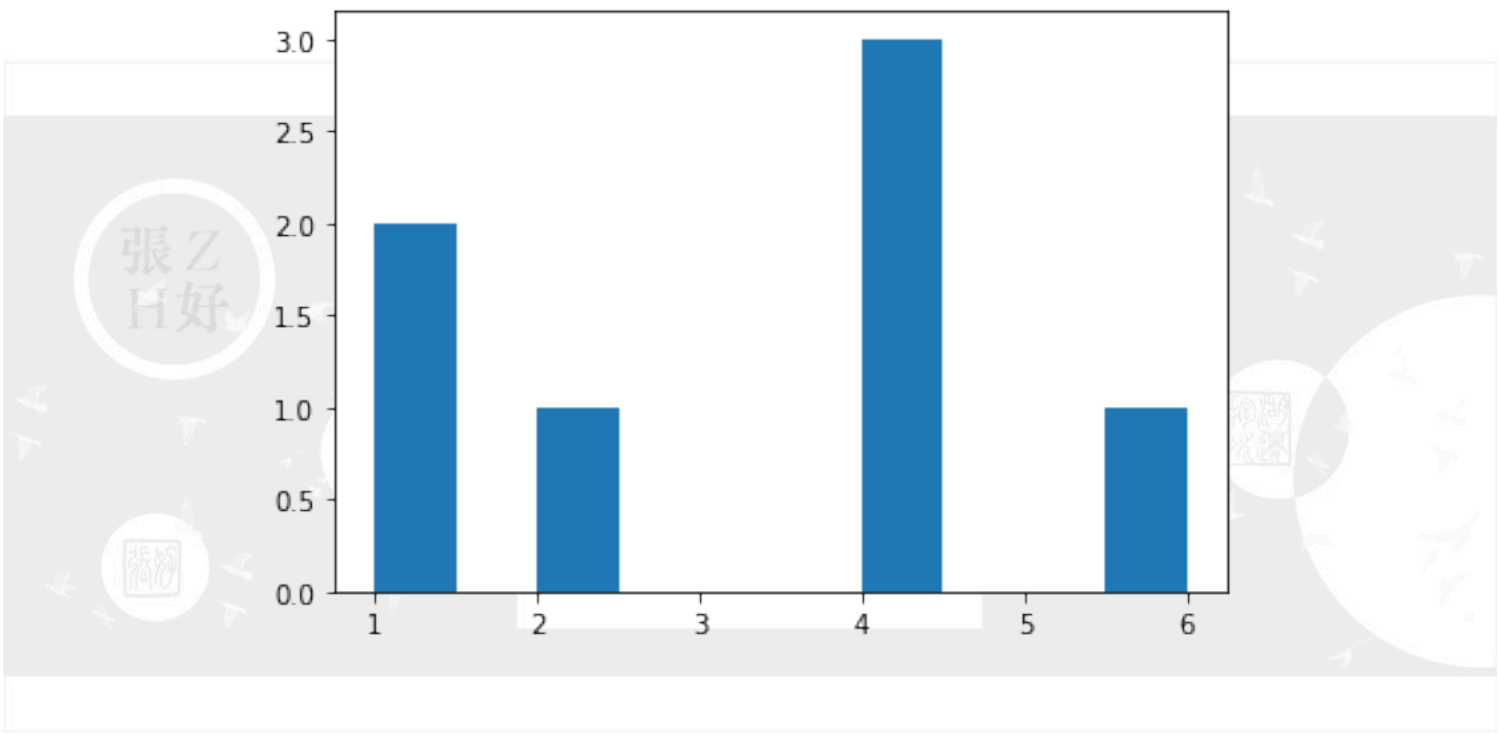
[44]:
```
plt.hist([1, 5, 5, 7, 7, 7, 9])
plt.show()
```



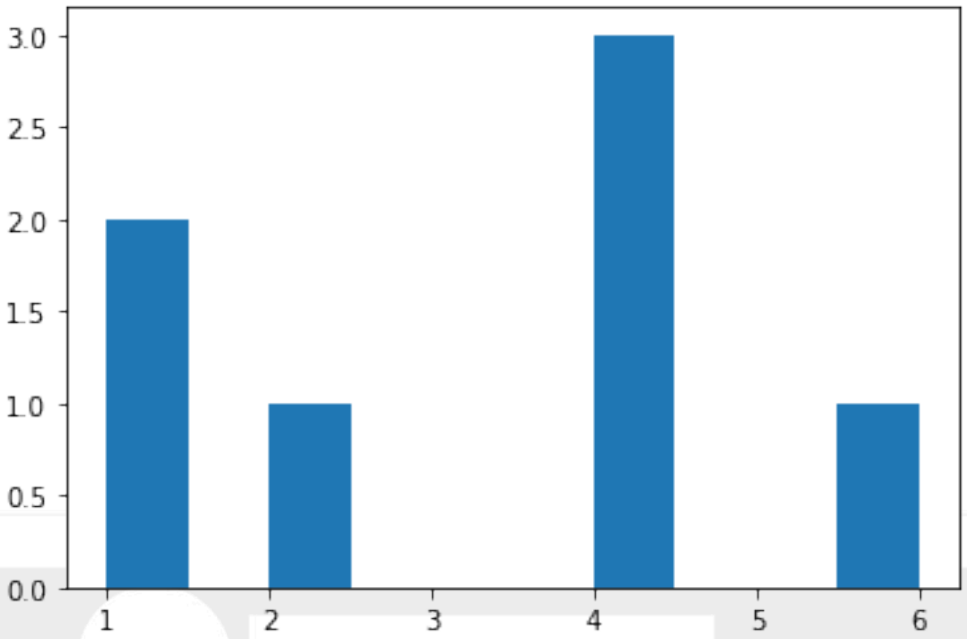
4.3 Code of combining NLP data extraction with plotting:

[45]:
```
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)
```

[45]:
```
(array([2., 0., 1., 0., 0., 0., 3., 0., 0., 1.]),
 array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. ]),
 <BarContainer object of 10 artists>)
```



[46]:
```
plt.hist(word_lengths)
plt.show()
```



4.4 Practice exercises for charting word length with NLTK:

▶ **Package pre-loading:**

[47]:
```
import re
from matplotlib import pyplot as plt
from nltk.tokenize import regexp_tokenize
```

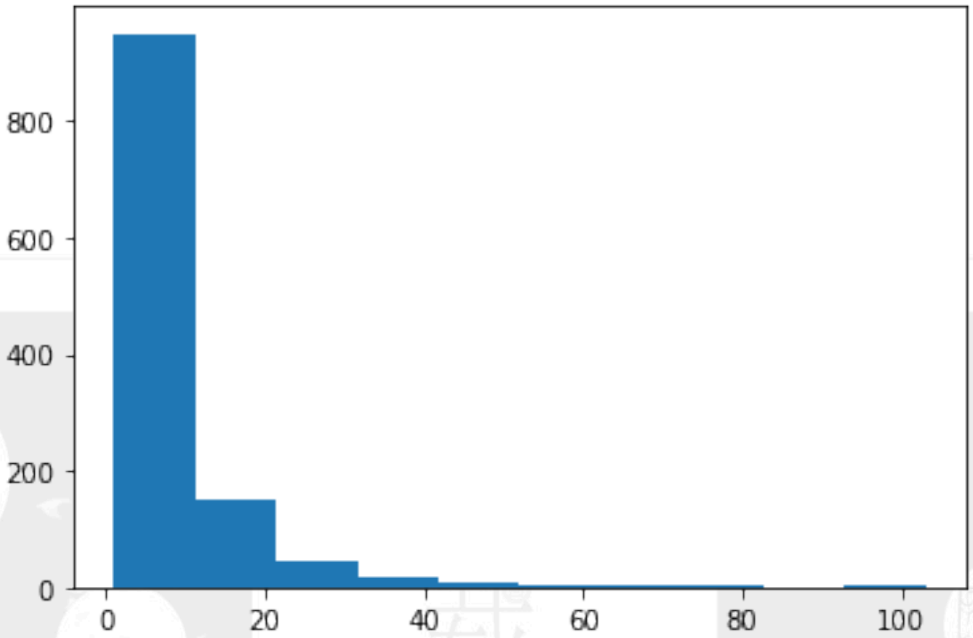
▶ **Data pre-loading:**

[48]:
```
holy_grail = open("ref4. Monty Python and the Holy Grail.txt").read()
```

▶ **Charting practice:**

[49]:
```
# Split the script into lines: lines
lines = holy_grail.split('\n')

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

# Tokenize each line: tokenized_lines
tokenized_lines = [regexp_tokenize(s, '\w+') for s in lines]

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

```
# Plot a histogram of the line lengths
plt.hist(line_num_words)

# Show the plot
plt.show()
```



4.5 Version checking:

[50]:
```
import sys
import nltk
import matplotlib

print('The Python version is {}.'.format(sys.version.split()[0]))
print('The NLTK version is {}.'.format(nltk.__version__))
print('The Matplotlib version is {}.'.format(matplotlib.__version__))
```

```
The Python version is 3.7.9.
The NLTK version is 3.5.
The Matplotlib version is 3.3.4.
```