Regular expressions & word tokenization

Puteaux, Fall/Winter 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 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*

1.2 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'>
```

1.3 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'

1.4 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.

1.5 Code of Python's `re` module:

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

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

1.6 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']
```

1.7 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']
```

```
##############################################
##                                          ##
##  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:

[13]:
```
from nltk.tokenize import word_tokenize

word_tokenize("Hi there!")
```

[13]:
```
['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()`):

[14]:
```
import re

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

[14]:
```
<re.Match object; span=(0, 3), match='abc'>
```

[15]:
```
re.search('abc', 'abcde')
```

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

[16]:
```
re.match('cd', 'abcde')
```

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

[17]:
```
<re.Match object; span=(2, 4), match='cd'>
```

2.7 Practice exercises for introduction to tokenization:

▶ Data pre-loading:

[18]:
```
scene_one = open("ref2. Monty Python and the Holy Grail.txt").read()
```

▶ NLTK word tokenization with practice:

[19]:
```
# 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)
```

```
{'Ewing', 'dirty', 'Yeah', 'And', 'handle', 'Erm', 'horn', 'ere', 'request',
'dorsal', 'exploiting', 'Try', ']', 'One', 'heard', 'GREEN', 'Running',
'pestilence', "'aaggggh", 'vache', 'Anarcho-syndicalism', 'suit', 'Over',
'Throw', 'With', 'rest', 'least', 'rodent', 'scene', 'Amen', 'Stand', 'quarrel',
'fly', 'return', 'uhh', 'mashed', 'period', 'ninepence', 'time-a',
'illegitimate-faced', 'PRINCE', 'behold', 'looking', 'water', 'unhealthy',
'Together', 'masses', 'awfully', 'will', "'old", 'Iiiiives', 'affairs', 'oral',
'Does', 'false', 'aquatic', 'back', 'bum', 'woosh', "'is", 'presence', 'Grail',
'BLACK', 'Hill', 'nice-a', 'once', 'empty', 'employed', 'assault', 'swords',
'yel', 'bid', 'routines', 'Packing', 'charged', 'your', 'elderberries',
'biggest', 'lucky', 'seem', 'ju', 'LUCKY', 'held', 'gouged', 'lord',
'Pendragon', 'died', 'woman', 'nervous', 'CHARACTERS', 'guard', 'remain',
'knights', 'favor', 'shit', 'Lady', 'hidden', 'CUSTOMER', 'smashing', 'Supreme',
'bitching', 'profane', 'Crapper', 'design', 'lived', 'Oh', 'Shall', 'bold',
'England', 'wound', "'Ni", 'dare', 'gave', 'feast', 'chance', 'haaa', 'attend',
'system', 'spake', 'WINSTON', 'stay', 'asks', 'yours', 'sigh', 'wait', 'doubt',
'Sir', 'call', 'work', "'d", 'Aah', 'most', 'high-pitched', 'signifying', 'I',
'pray', 'keeper', 'Build', 'questions', 'wipers', 'scrape', 'known', 'Very',
'Summer', 'Patsy', 'leg', 'union', 'Agh', 'ehh', 'folk', 'how', 'lapin',
'Thpppt', 'As', 'Table', 'basis', 'ever', 'sloths', 'Lead', 'sequin', 'Guy',
'thine', 'Not-appearing-in-this-film', 'walk', 'bridgekeeper', 'by',
'LAUNCELOT', 'guarded', 'when', 'bastards', 'stop', 'example', 'Death',
'ordinary', 'bring', 'shelter', 'lies', 'which', 'bless', '--', 'repressing',
'became', 'Really', 'forty-three', 'Rheged', 'Eee', 'you', 'length', 'smack',
'use', 'dogma', 'bonk', 'totally', 'wedding', 'Good', 'heeh', 'commune', 'vary',
```

```
'Hold', 'guided', 'wind', 'mud', 'daft', 'donkey-bottom', 'Saint', 'flint',
'Uhh', 'suggesting', 'consulted', 'frighten', 'non-migratory', '#', 'sacrifice',
'stayed', 'warm', 'passed', 'Have', '10', 'dangerous', 'carries', 'visually',
'escape', 'sweet', 'KING', 'courage', 'g', 'went', 'saying', 'war', 'nineteen-
and-a-half', 'Hello', 'spooky', 'cross', 'mightiest', 'count', 'favorite',
'model', 'Un', 'Bristol', 'obviously', 'color', 'Burn', 'these', 'distributing',
'Hallo', 'wet', 'Action', 'CART-MASTER', 'whether', 'smashed', 'surprise',
'Now', 'Perhaps', ':', 'remembered', 'disheartened', 'body', 'recover', 'rewr',
'Divine', 'main', 'as', 'temptation', 'remember', 'Hmm', 'answer', 'ARTHUR',
'Open', 'forward', 'bed', 'shut', 'anarcho-syndicalist', 'sawwwww', 'havin',
'Bread', 'Are', 'pansy', 'meant', 'scared', 'gra', 'run', 'door', 'Knight',
'cover', 'Steady', 'quest', 'Winter', 'carved', 'master', 'ni', 'might', 'pond',
'BRIDGEKEEPER', 'seen', 'kick', 'winter', 'Blue', 'GUARDS', 'brain', 'Cider',
'drillllll', 'Charge', 'Gable', 'young', 'late', 'African', 'emperor', 'Robin',
'spanking', 'Black', 'may', 'On', 'owns', 'bite', 'dark', 'SECOND', 'witches',
'snore', 'dying', 'offensive', 'very', 'fight', 'CHARACTER', 'Bravest',
'k-nnniggets', 'snows', 'retreat', 'Caerbannog', 'whoever', 'plain',
'Camaaaaaargue', 'high', 'regulations', 'must', 'running', 'find', 'raped',
'pweeng', 'wide', 'rock', 'medieval', 'utterly', 'again', 'merger', 'Assyria',
'ROBIN', 'bugger-folk', 'well', 'plover', 'matter', 'lunged', 'shivering',
'setting', 'spanked', 'orangutans', 'him', 'busy', 'Providence', 'Dis-mount',
"'Aauuuuugh", 'daughter', 'chickening', 'wishes', 'forth', 'eight',
'differences', 'NARRATOR', 'fourth', 'BEDEVERE', 'Greetings', 'stress',
'Churches', 'dictatorship', 'newt', 'has', 'keep', 'Hee', 'name', 'shows',
'sad', 'somebody', 'Anyway', 'CONCORDE', 'Ridden', 'ungallant', 'Quiet', 'Um',
'idiom', 'heads', 'same', 'it', 'zone', 'va.', 'centuries', 'quick', 'saved',
'baaaa', 'Prepare', 'l', 'Hiyya', 'Woa', 'scenes', 'Castle', 'Enchanter',
'penalty', 'large', 'further', 'met', 'Welcome', 'na', 'inferior', 'little',
'Huh', 'imperialist', 'SOLDIER', ';', 'rhymes', 'Mind', 'cost', "e'er",
'swallow', 'Thy', 'Remove', 'worse', 'inherent', 'suffered', 'aaugh',
'Auuuuuuuugh', 'groveling', 'thou', 'wonderful', 'unsingable', 'yet', 'expect',
'burn', 'band', 'spoken', 'cast', 'south', 'even', 'thirty-seven', 'What', "'s",
'certainly', 'Iesu', 'come', 'dead', 'anything', 'safety', 'protect', "'Here",
'nearly', 'Knights', 'liege', 'headoff', 'THE', 'Must', 'north-east', 'ai',
'haste', 'KNIGHTS', 'those', 'How', 'rich', 'Away', 'have', 'people', 'awhile',
'ARMY', 'vests', 'Cornwall', 'Ulk', 'mad', "'", 'sing', 'talk', 'rode', 'bois',
'12', 'watery', 'bridges', 'Aaaaaaaah', 'object', 'treat', 'guest', 'Brave',
'set', 'towards', 'aptly', 'life', 'GUEST', 'snuff', 'knight', 'thank', 'stood',
'Riiight', 'changed', 'shall', 'bangin', 'hang', 'witch', 'window-dresser',
'can', 'joyful', 'donaeis', 'fwump', 'sacred', 'soon', 'give', 'MONKS', 'Uugh',
'keepers', 'trough', 'think', 'table', 'bastard', 'got', 'progress', 'Uuh',
'particular', 'dull', 'Thee', ')', 'today', 'speak', 'yelling', 'ha', 'doctors',
'forced', '1', 'mean', 'Hey', 'BRIDE', 'Chapter', 'lady', 'deal', 'effect',
'welcome', 'mooooooo', 'This', 'Order', 'nearer', 'So', 'food', 'tiny',
'Piglet', 'Great', 'than', 'armed', 'lad', 'bravest', "'ni", 'follow', 'bint',
'Hah', 'runes', 'yellow', 'government', 'looks', 'throwing', 'rescue', 'Listen',
'wayy', 'down', '4', 'great', 'father', '24', 'PRISONER', 'dungeon', 'gone',
'intermission', 'When', 'Clear', 'bride', 'tea', 'nine', 'about', 'tree',
```

```
'teeth', 'blood', 'doing', 'Shh', 'p', 'fought', 'Anybody', 'ones', 'Ooh',
'tit', 'approaching', 'deeds', 'forget', 'risk', 'fair', 'CRONE', 'grail-
shaped', 'spank', 'tart', 'sonny', 'testicles', 'MIDDLE', 'vouchsafed',
'wooden', 'Your', 'Uh', 'BORS', 'someone', '(', 'naughty', "'S", 'samite',
'side', 'chorus', 'fled', 'accent', 'preserving', 'Far', 'fortune', 'Bridge',
'Silence', 'any', 'icy', 'Five', "C'est", 'problems', 'Victory', 'enchanter',
'classes', 'force', 'accomplished', 'tops', 'mystic', 'snap', 'Ohh', 'sure',
'Mother', 'legally', 'automatically', 'Rather', 'week', 'rrrr', 'LEFT',
'supreme', 'Saxons', 'Aaaaugh', 'apart', 'Arimathea', 'die', 'Anthrax', 'pig-
dogs', 'feel', 'Olfin', 'Hurry', 'duty', 'jump', 'No', 'conclusions', 'My',
'splat', 'previous', 'either', 'SCENE', 'am', 'outside', 'gained', 'rocks',
'writing', 'anyone', 'chord', 'draw', 'science', 'rejoicing', 'resumes', 'liar',
'lot', 'Pin', 'capital', 'thump', 'Bloody', 'tail', 'CARTOON', 'Supposing',
"'ll", 'because', "'cause", 'Galahad', 'un', 'and', 'twin', 'HISTORIAN',
'Alice', 'knew', 'manner', 'unladen', 'It', 'silly', 'twang', 'who', 'vicious',
'leads', 'sex', 'crone', 'Thpppppt', 'Consult', 'told', 'Hang', 'ho', 'dancing',
'ruffians', 'purest', 'clllank', 'Swamp', 'long', 'but', 'beds', '?',
'traveller', 'formed', 'Would', 'bet', 'outdoors', 'number', 'workers', 'vital',
'birds', 'exciting', 'cave', 'stupid', 'enemies', 'zoosh', 'Gawain', 'guards',
'wave', 'interested', 'pimples', 'bleeder', 'identical', 'TIM', 'or', 'Of',
'open', 'other', 'made', 'one', '7', 'uh', 'bottoms', 'Man', 'dictating',
'left', 'All', 'Three', 'Dragon', 'OTHER', 'home', 'art', 'Zoot', 'eat', 'away',
'8', 'mayhem', 'compared', 'thing', 'tropical', 'laden', 'leap', '2', 'Off',
'Tall', 'Augh', 'right', 'scales', 'pure', 'smelt', 'But', 'clank', 'become',
'beacon', ',', 'gentle', 'some', 'day', 'unplugged', 'binding', 'full',
'amazes', 'miserable', 'much', 'more', 'say', 'bringing', 'fine', 'eis',
'maybe', 'legs', 'Monsieur', 'Even', 'at', 'duck', 'anywhere', 'Hyy',
'advancing', 'Dramatically', 'ask', 'pay', 'grin', 'language', 'Walk', 'head',
'kingdom', 'Nothing', 'they', 'enter', 'off', 'accompanied', 'tired', 'nasty',
'See', 'Then', 'mumble', 'easy', 'far', 'foe', 'OFFICER', 'her', 'Farewell',
'only', 'best', 'hospitality', 'husk', 'gallantly', 'makes', 'bats', 'so',
'tell', 'them', 'two-thirds', 'using', "'m", 'types-a', 'separate', 'largest',
'nobody', 'Skip', 'etc', 'badger', 'Other', 'entrance', 'last', 'y', 'Pure',
'Everything', 'Lake', 'grovel', 'glass', 'quack', 'ugly', 'wherein', 'arm',
'excuse', 'third', 'tale', 'W', 'breakfast', 'supports', 'ill.', 'Whoa',
'clack', 'Brother', 'nor', 'Two', 'verses', 'training', 'anchovies', 'breath',
'Britons', 'closest', '13', 'sons', 'old', 'north', 'working', 'understanding',
'nose', 'until', 'Hand', 'times', 'society', 'Aaah', 'guests', 'oui',
'creature', 'fold', 'was', 'Ah', 'WIFE', 'MAYNARD', 'Ninepence', 'Like',
'Yeaaah', 'peasant', 'Halt', 'word', 'eccentric', 'Practice', 'fart',
'coconuts', 'minutes', 'Eternal', 'wood', '21', 'summon', '5', 'bit', 'throat',
'pack', 'Wood', 'Lancelot', 'awaaay', 'Be', '[', 'wants', 'else', "'it",
'pointy', 'Book', 'aside', 'ptoo', 'convinced', 'Right', 'OF', 'proceed',
'king-a', 'lair', 'officer', 'out-clever', 'Today', 'occasion', 'Antioch',
'Thank', 'PARTY', 'heroic', 'weight', 'Aaaugh', 'girl', 'this', 's', 'packing',
'mine', 'Picture', 'whom', 'argue', 'NI', 'round', 'Once', "'Ooooooooh",
'angels', 'legendary', 'cough', 'happens', 'trusty', 'mandate', 'together',
'Quickly', 'pause', 'inside', 'Say', 'Spring', 'VOICE', 'animator', 'around',
```

```
'its', 'logically', 'present', "'Morning", 'helpful', 'carried', 'ANIMATOR',
'Found', 'outdated', 'arrows', 'Thppt', 'Since', 'derives', 'knows', 'violence',
'self-perpetuating', 'pissing', 'just', 'alarm', 'case', 'w', 'out', 'bird',
'breadth', 'need', 'Unfortunately', 'pull', 'Silly', 'Please', 'For', 'finest',
'frontal', 'committed', 'listen', 'stab', 'Bors', 'bosom', 'daring', 'Lie',
'mayest', 'says', 'Tower', 'BROTHER', 'carry', 'Aramaic', 'internal', 'song',
'sire', 'dub', 'take', 'himself', 'stretched', 'every', 'thud', 'seldom',
'hell', 'Ha', 'Oooo', 'aaaaaah', 'KNIGHT', 'witness', 'unclog', 'Dennis',
'scimitar', 'could', 'see', 'never', 'bathing', 'know', 'sir', 'God', 'He',
'Here', 'Meanwhile', 'idea', 'impersonate', 'each', 'Quite', 'DEAD', 'persons',
'Frank', "'Til", 'weighs', 'man', '6', 'cheesy', 'Thou', 'Why', 'special',
'keen', 'sister', 'fruit', 'Come', 'PERSON', 'next', 'flights', 'worry', "'Man",
'ROGER', 'fifty', 'fellows', 'Badon', 'Yay', 'GOD', 'Winston', 'general',
'repressed', 'swallows', 'lambs', 'mortally', 'brush', 'eats', 'electric',
'own', 'Aaauugh', 'married', 'To', 'explain', 'Midget', 'underwear', 'kings',
'ounce', 'refuse', 'Grenade', 'Cut', 'point', 'They', 'Ho', '19', 'c', 'valor',
'dressing', 'wise', 'Nu', 'dunno', 'through', 'formidable', 'grail', 'Lord',
'send', 'O', 'being', 'Old', 'couple', 'sixteen', 'outwit', 'leave', 'sign',
'go', 'all', 'turns', 'That', 'alive', 'retold', 'hundred-and-fifty', 'Thppppt',
'silence', 'HEADS', 'bang', 'feint', 'Yapping', 'lying', 'weapon', 'good',
'autonomous', 'sovereign', 'higher', 'power', 'k-nnnnniggets', 'knocked',
'European', 'watch', 'tiny-brained', 'thought', 'mistake', 'Autumn', 'SIR',
'sponge', 'over', 'spam', 'stuffed', 'Every', 'o', 'handsome', 'cop', 'GUESTS',
'liver', 'Himself', 'join', 'biscuits', 'entered', 'Chicken', 'Schools',
"'First", 'valleys', 'diaphragm', 'mer', 'Mmm', 'Princess', 'VILLAGER', 'peril',
'Hooray', 'strength', 'ALL', 'counting', 'he', 'needs', 'night', 'taunting',
'woods', 'expensive', 'chanting', 'Stop', 'bad-tempered', 'temperate',
'SHRUBBER', 'here', 'confuse', 'wart', 'sorry', 'dappy', 'apologise', 'b',
'reads', 'Beast', 'sell', 'please', 'Arthur', 'Tell', 'triumphs', 'slightly',
'burst', 'boom', 'Doctor', 'aloft', 'ours', "'To", 'harmless', 'chops',
'looney', 'history', 'biters', 'commands', 'By', 'Sorry', 'LOVELY', 'chickened',
'splash', 'Bedevere', 'gravy', 'English', 'conclusion', 'thonk', 'enough',
'RANDOM', 'Who', 'humble', 'to', 'Just', 'time', 'suspenseful', 'u', 'built',
'git', 'kneecaps', 'an', 'two-level', 'martin', 'tackle', 'particularly',
'brought', 'on', 'indeed', 'build', 'pig', 'hiyaah', 'Alright', 'Hoa', 'if',
'Let', 'straight', 'sword', 'king', 'she', 'happy', 'is', 'Looks', 'hills',
'Aagh', 'dynamite', 'yeah', 'asking', 'waste', 'Our', 'cereals', 'show', 'dona',
'end', 'everyone', 'successful', 'twenty', "'anging", 'N', 'reared', 'taken',
'easily', 'dressed', 'travellers', 'pounds', 'forest', 'She', 'pound', 'ponds',
'Aaaaaaaaah', 'Lucky', 'VILLAGERS', 'drink', 'bloody', 'executive', '17',
'Oooohoohohooo', 'parts', 'then', 'attack', 'creep', 'slash', 'islands',
'influential', 'proved', 'Aaauggh', 'demand', 'town', 'court', 'felt', 'Round',
'CRASH', 'big', 'Hiyah', 'kneeling', 'nick', 'CRAPPER', 'into', 'quite',
'sense', 'Hiyaah', 'Christ', 'Neee-wom', 'Use', 'Ask', 'Stay', 'mangled',
'everything', 'anyway', 'Aggh', 'cart', 'May', 'er', "'Ecky-ecky-ecky-ecky-
pikang-zoop-boing-goodem-zoo-owli-zhiv", 'INSPECTOR', 'cruel', 'Guards',
'resting', 'lie', 'economic', 'nice', 'year', 'bad', 'Psalms', 'live', 'Court',
'bladders', 'Hm', 'done', 'Nine', 'n', 'Shut', 'ye', 'chosen', 'danger',
```

```
'singing', 'delirious', 'Splendid', 'Those', 'shimmering', 'major', 'trouble',
'requiem', 'things', 'plan', 'gon', 'appease', 'Which', 'always', 'land',
'pulp', 'Speak', 'suffice', 'grips', 'bleed', 'Tim', 'get', 'unarmed',
'valiant', 'killed', 'mile', 'alight', 'Ives', 'ran', 'seemed', 'moistened',
'16', "'Erbert", 'started', 'oo', 'MAN', 'frozen', 'hoo', 'crash', 'collective',
'Keep', 'cadeau', 'clunk', 'certain', 'really', 'starling', 'thy', '..', 'ride',
'Honestly', 'Too', 'not', 'Recently', 'Eh', 'Exactly', 'Fetchez', 'ooh',
'squeak', 'after', '…', 'FRENCH', 'two', 'pen', 'we', 'bother', 'meeting',
'enjoying', 'floats', 'taking', 'CAMERAMAN', 'chu', 'our', 'foul', 'walking',
'lost', 'Y', 'Almighty', 'bells', 'Ay', 'um', 'allowed', 'discovers', "'em",
'headed', 'afraid', 'shrubbery', 'Seek', 'approacheth', 'marrying', 'Armaments',
'flight', '3', 'ignore', "'sorry", 'vain', 'therefore', 'four', 'friend',
'grenade', '18', 'GUARD', 'medical', 'Hya', 'upon', 'mate', 'depart', 'Where',
'blanket', 'Not', 'velocity', 'j', 'hee', 'fooling', 'Roger', 'Or', 'defeator',
'nibble', 'There', 'freedom', 'hmm', 'discovered', 'blessing', 'acting', 'like',
'path', 'carrying', 'averting', 'aunties', 'perilous', 'hamster', 'In',
'Therefore', 'horse', 'prevent', 'ethereal', 'Heh', 'cope', 'Nador', 'Aauuuves',
'up', 'individually', 'dragging', 'new', 'tonight', 'tragic', 'lovely',
'required', 'pussy', 'creak', 'radio', "'forgive", '14', 'were', 'farcical',
'autocracy', 'herring', 'single-handed', 'hand', 'wicked', 'DIRECTOR', 'did',
'whop', 'magne', 'moment', 'be', 'trumpets', 'afoot', 'Wait', 'elbows',
'absolutely', 'Bones', 'illustrious', 'Four', 'having', 'Excalibur',
'punishment', 'the-not-quite-so-brave-as-Sir-Lancelot', 'with', 'throughout',
'bows', 'horrendous', 'sniff', 'simple', 'scribble', 'understand', 'Never',
'real', 'able', 'laughing', 'warning', "d'you", 'sharp', 'Nay', 'impeccable',
'Aaaah', 'Bad', 'are', 'The', 'Fiends', 'Until', 'noise', 'test', 'worthy',
'Look', 'praised', 'now', 'while', 'PATSY', 'tap-dancing', 'what', 'Go', 'ZOOT',
'Ahh', 'blondes', 'Loimbard', 'carp', 'want', 'wounding', 'hopeless', 'soiled',
'killer', 'Mercea', 'Dingo', 'Attila', 'beautiful', 'does', 'said', 'weather',
'business', 'Leaving', 'adversary', 'later', 'Gorge', 'giggle', 'dine', 'sight',
'split', 'Aauuggghhh', 'middle', 'since', 'purpose', 'eet', 'imprisoned',
'Follow', 'quests', 'outrageous', 'B', 'HEAD', 'too', "'Dennis", 'place',
'Explain', 'scots', 'been', 'Fine', 'raised', 'Chickennn', 'Ector', 'warmer',
'me', 'You', 'Cherries', 'Iiiives', 'Peng', "'Ere", 'Jesus', 'settles', 'blow',
'trade', "'round", 'saw', 'rather', 'hall', 'animal', 'ratios', "n't", 'bottom',
'dance', 'going', 'France', 'worked', 'scott', 'guiding', 'Thsss', 'Message',
'hacked', 'pram', 'eh', 'beyond', 'yes', 'servant', 'cartoon', 'nothing',
'vote', 'armor', 'Back', 'almost', 'WOMAN', 'a', 'Quick', 'doors', 'Ni',
'already', 'broken', 'ways', 'that', 'mangy', 'None', 'something', 'defeat',
'bed-wetting', 'words', 'tear', 'Dappy', 'Tale', 'way', 'kind', 'Most', 'face',
'lives', 'maintain', "'til", 'cut', 'cry', 'move', 'reasonable', 'performance',
'my', 'properly', 'Said', 'tinder', 'relax', 'Put', 'foot', 'mother', 'oh',
"'re", 'mac', 'first', 'act', 'took', 'uuggggggh', 'honored', 'jam', 'tracts',
'ham', 'goes', 'holy', 'avenged', 'music', 'gurgle', 'HERBERT', 'ceremony',
'domine', 'somewhere', 'Behold', 'DINGO', 'knees-bent', 'search', 'types',
'personally', 'soft', 'death', 'riding', 'tie', 'eyes', 'RIGHT', 'MIDGET',
'nostrils', 'bond', 'from', 'Thursday', 'Yes', 'kill', 'More', 'strongest',
'roar', 'Actually', 'minute', 'less', 'near', 'learning', 'dad', 'rope', '15',
```

'Waa', 'worried', 'bunny', 'beat', 'necessary', 'baby', 'women', 'Well',
'mercy', 'Umhm', 'miss', 'Chop', 'van', 'majority', 'la', 'leaps', 'clang',
'string', 'put', 'returns', 'still', 'chest', 'flesh', 'Beyond', 'Gallahad',
'Haw', 'sod', 'k-niggets', 'ratified', 'pass', 'also', 'considerable', 'fire',
'mooo', 'heart', 'undressing', 'whose', 'wrong', 'castanets', 'wounded', 'used',
'Battle', 'fatal', 'air-speed', 'their', 'suddenly', 'U', 'shrubberies', 'bi-
weekly', 'glory', 'oooh', '11', 'arms', 'AMAZING', 'Do', 'Excuse', 'Twenty-one',
'If', 'scarper', 'wo', 'sneaking', 'let', 'i', 'brunettes', 'crying', 'Help',
'three', 'buggering', 'We', 'MINSTREL', 'crossed', 'boys', 'bones', 'Bring',
'line', 'although', 'Uther', 'Pull', 'terribly', 'looked', 'Father', 'Is',
'voluntarily', 'warned', 'Aaagh', 'note', 'swamp', 'why', 'in', 'continue',
'person', 'awaaaaay', 'there', 'telling', 'hello', 'strange', 'clap', 'covered',
'WITCH', 'many', 'excepting', 'bravely', 'Shrubberies', 'given', 'social',
'strand', "'aaaah", 'Could', 'making', 'sort', 'Robinson', 'Am', 'Clark',
'seems', 'grip', 'strategy', 'Get', "'shrubberies", 'French', 'examine', 'Yup',
'siren', 'Camelot', 'lose', 'King', 'carving', 'Joseph', 'whispering',
'temptress', 'sun', 'relics', 'Ayy', 'brave', 'Umm', 'push', 'taunt', 'fallen',
'huge', 'Huy', 'behaviour', 'evil', 'filth', 'howl', 'of', 'wiper', 'burned',
'tough', 'His', 'terrible', 'Bon', 'luck', 'streak', 'small', 'purely',
'buggered', 'country', 'bridge', 'five', 'lobbed', 'men', 'knock',
'varletesses', 'basic', 'shrubber', 'GIRLS', 'immediately', 'for', 'curtains',
'Heee', 'should', 'spirit', 'gay', 'hear', 'ready', 'decided', 'lobbest', 'An',
'the', 'comin', 'At', 'glad', 'Herbert', 'opera', 'found', 'awaits',
"'uuggggggh", 'his', "'ve", 'private', 'Idiom', 'beside', 'Mud', 'turned', 'de',
'between', 'question', 'migrate', 'True', 'stew', 'invincible', 'sink', 'sank',
'wings', 'Peril', 'eisrequiem', 'scratch', 'Allo', 'strewn', 'give-away',
'Mine', 'worst', 'without', 'PIGLET', 'bowels', 'Launcelot', '20', 'room',
'Hoo', 'along', 'earthquakes', 'kicked', 'ladies', 'thanks', 'where', 'getting',
'Will', 'marry', 'suppose', 'strangers', 'under', 'Aaaaaah', 'assist', 'sworn',
'change', 'clear', 'stone', 'shalt', 'ferocity', 'Bedwere', 'quiet', 'reached',
'minstrels', 'removed', 'lads', 'completely', 'Forward', 'son', 'against',
'whinny', 'indefatigable', 'chastity', 'carve', 'dress', 'Pie', 'help',
'yourself', 'sheep', 'door-opening', 'living', 'wan', 'score', 'Make',
'dramatic', 'Surely', 'A', 'had', 'sample', 'names', "'T", 'Oui', 'wedlock',
'laurels', 'Angnor', 'SUN', 'Defeat', 'OLD', 'Ages', 'such', 'feet', 'Bravely',
'behind', 'jokes', 'actually', 'entering', 'called', 'SENTRY', 'thwonk', 'seek',
'twenty-four', 'Oooh', 'no', 'castle', 'haw', 'Aauuugh', "'Course", 'uuup',
'dear', 'auntie', 'make', 'STUNNER', 'supposed', 'Shrubber', 'Hic', 'better',
'wield', 'Maynard', 'named', 'Holy', 'Did', 'Concorde', 'try', 'second', 'task',
'distress', 'Britain', 'order', 'agree', 'sometimes', '23', 'problem',
'another', '!', 'would', 'hospital', 'kills', 'twong', 'footwork', 'coming',
'Prince', 'finds', 'halves', 'creeper', 'e', 'Between', 'Wayy', 'sent', 'bits',
'9', 'decision', 'hast', 'direction', 'stand', 'ENCHANTER', 'FATHER', 'boil',
'look', 'Quoi', 'Run', 'CROWD', 'bicker', 'arrange', 'Firstly', 'coconut',
'climes', 'command', 'auuuuuuuugh', 'ca', 'so-called', 'buy', '.', 'Apples',
'fell', 'course', 'do', 'house', 'nightfall', 'clop', 'us', 'limbs', 'Torment',
'clad', 'feathers', 'earth', 'class', 'longer', 'ridden', 'threw', 'depressing',
'stops', 'clue', 'DENNIS', 'hat', 'lonely', 'Yeaah', 'rabbit', 'heh', 'Ow',

```
'banana-shaped', 'scholar', 'answers', 'Huyah', 'PRINCESS', 'Chaste',
'perpetuates', 'passing', 'Forgive', 'GALAHAD', 'police', '22'}
```

▶ **Package pre-loading:**

[20]:
```
import re
```

▶ **Regex (re.search()) practice:**

[21]:
```
# 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
```

[22]:
```
# 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]'>
```

[23]:
```
# 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:'>
##############################################
##                                          ##
##  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":

[24]:
```
import re

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

[24]:
```
['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()`:

[25]:
```
import re

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

[25]:
```
<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:**

[26]:
```
from nltk.tokenize import regexp_tokenize
```

▶ **Data pre-loading:**

[27]:
```
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:**

[28]:
```
pattern = pattern1
print(regexp_tokenize(string, pattern))
```

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

[29]:
```
pattern = pattern2
print(regexp_tokenize(string, pattern))
```

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

[30]:
```
pattern = pattern3
print(regexp_tokenize(string, pattern))
```

```
[]
```

[31]:
```
pattern = pattern4
print(regexp_tokenize(string, pattern))
```

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

3.6 Practice exercises for advanced tokenization with NLTK and regex:

▶ **Data pre-loading:**

[32]:
```
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:**

[33]:
```
# Import the necessary modules
from nltk.tokenize import TweetTokenizer
```

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

[34]:
```
# 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']
```

[35]:
```
# 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']
```

[36]:
```
# 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:

[37]:
```
from nltk.tokenize import word_tokenize
```

▶ Data re-pre-loading:

[38]:
```
german_text = 'Wann gehen wir Pizza essen?  Und fährst du mit Über? '
```

▶ Non-ascii tokenization practice:

[39]:
```
# 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']
[' ', ' ']
```

```
##############################################
##                                          ##
##  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?

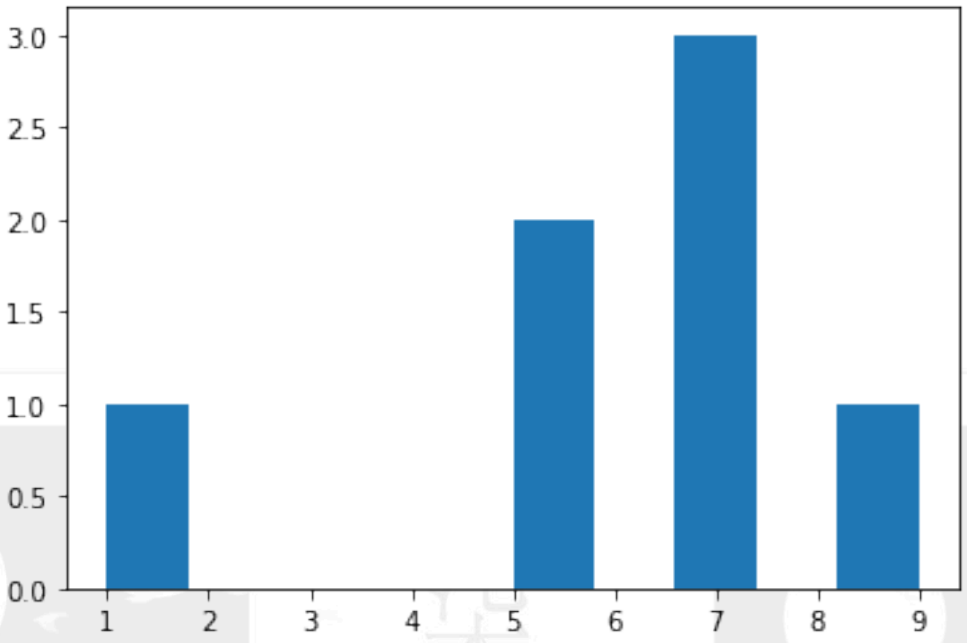
- It is a charting library used by many open-source Python projects.
- It has straightforward functionality with lots of options:
  - *histograms*
  - *bar charts*
  - *line charts*
  - *scatter plots*
- And also, it has advanced functionality like 3D graphs and animations!

4.2 Code of plotting a histogram with matplotlib:

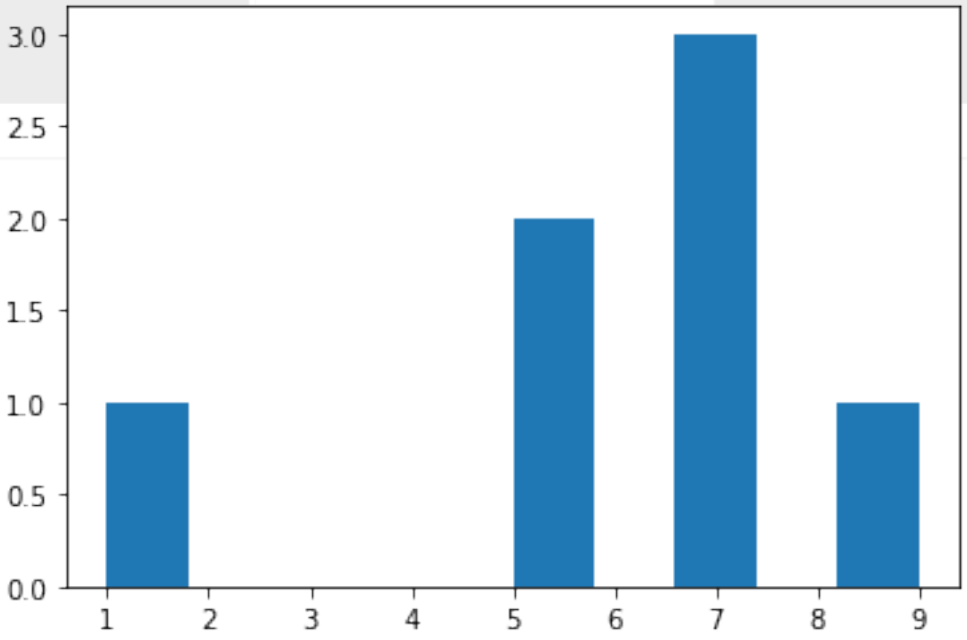
[40]:
```
from matplotlib import pyplot as plt

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

[40]:
```
(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>)
```



[41]:
```
plt.hist([1, 5, 5, 7, 7, 7, 9])
plt.show()
```

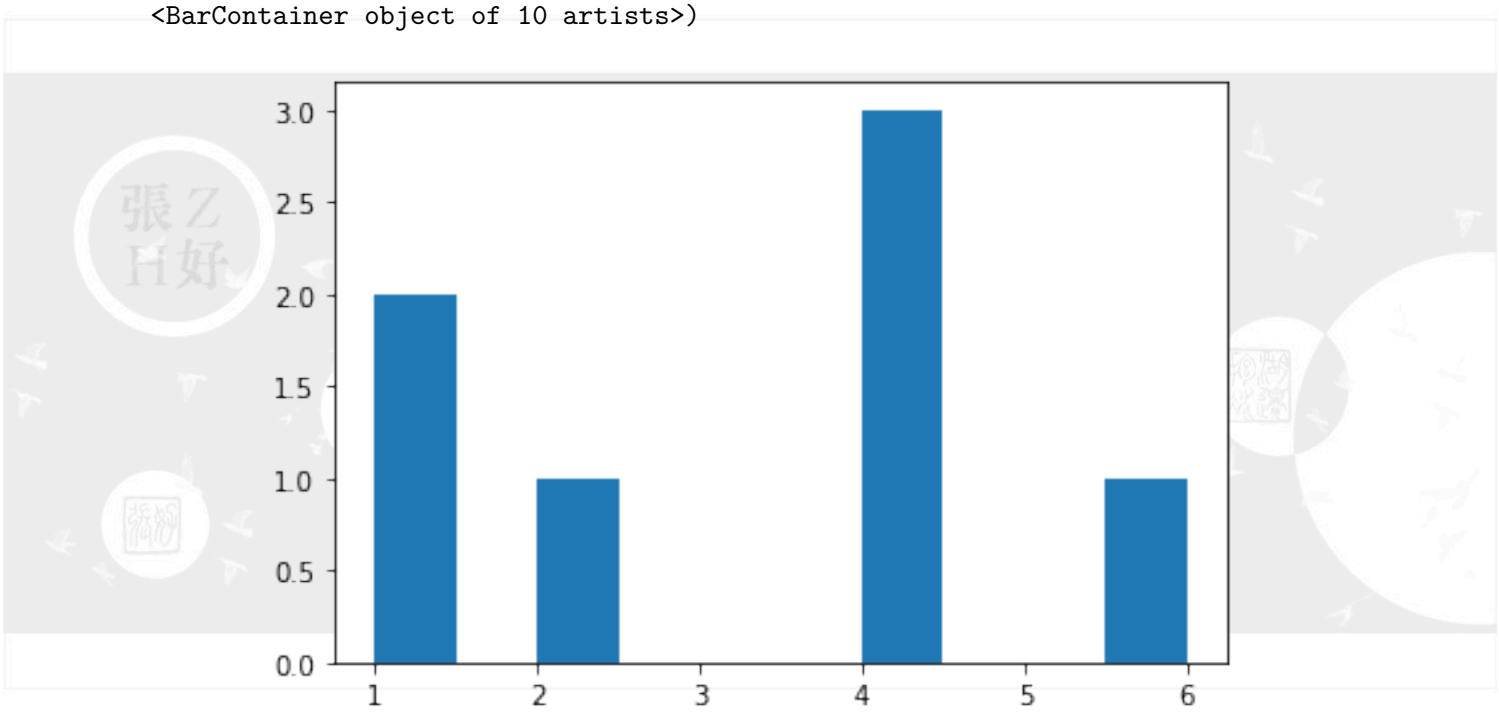


4.3 Code of combining NLP data extraction with plotting:

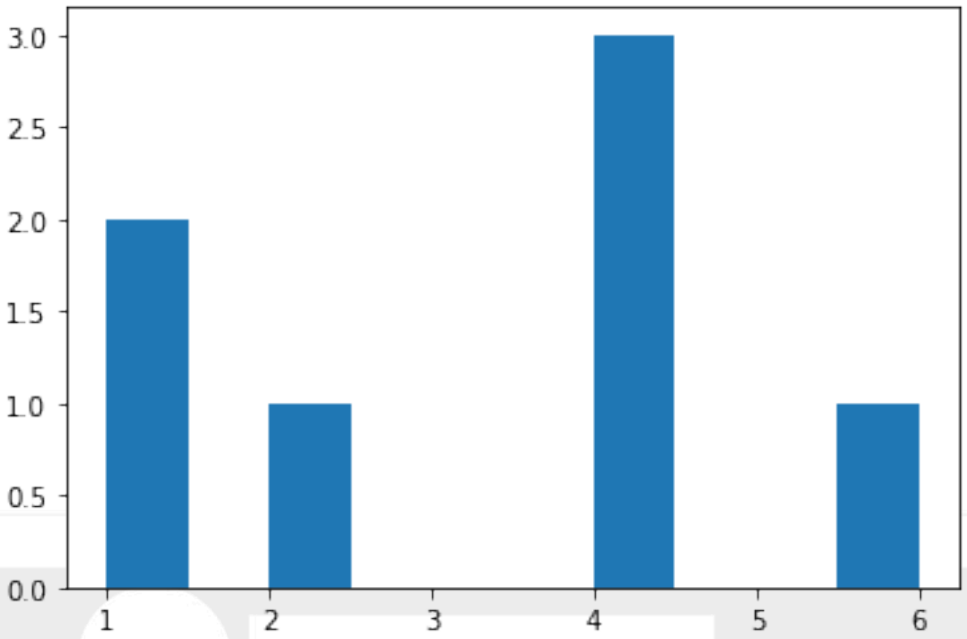
[42]:
```
from matplotlib import pyplot as plt
from nltk.tokenize import word_tokenize

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

[42]:
```
(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>)
```



[43]:
```
plt.hist(word_lengths)
plt.show()
```



4.4 Practice exercises for charting word length with NLTK:

▶ **Package pre-loading:**

[44]:
```
import re
from matplotlib import pyplot as plt
from nltk.tokenize import regexp_tokenize
```

▶ **Data pre-loading:**

[45]:
```
holy_grail = open("ref2. Monty Python and the Holy Grail.txt").read()
```

▶ **Charting practice:**

[46]:
```
# 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()
```