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)
```

```
{'Which', 'up', 'footwork', 'heads', 'working', 'boys', 'bad', 'Angnor', 'mate',
'carry', 'mystic', 'GUARDS', 'favorite', 'Anthrax', 'pure', "'Til", 'they',
'whoever', 'water', 'Right', 'climes', 'unclog', 'domine', 'sacred', 'goes',
'must', 'but', 'Ooh', 'feint', 'Hooray', 'donaeis', 'gurgle', 'actually', 'wan',
'meant', ']', 'blessing', 'MINSTREL', 'in', 'mistake', 'Aaah', 'taking', 'Augh',
'girl', 'avenged', 'examine', 'Skip', 'obviously', 'door', 'oh', 'pounds',
'Aaauugh', 'anchovies', 'shall', 'spam', 'seems', 'Huy', 'doing', 'four',
'unarmed', 'body', 'centuries', 'brought', 'watery', 'k-niggets', 'perilous',
'stress', 'suggesting', 'fwump', 'committed', 'clllank', 'huge', 'behold',
'conclusions', 'True', 'repressed', 'pray', 'daughter', 'identical', "'is",
'open', 'Walk', 'perpetuates', 'force', 'rest', 'employed', 'music', 'called',
'awhile', 'crash', 'i', 'throughout', 'Yup', 'death', 'beds', 'fallen', 'Very',
'hospital', 'wedding', 'smack', 'Dennis', 'tonight', 'ANIMATOR', 'comin',
'VOICE', 'illustrious', 'French', 'wedlock', 'Silence', 'fellows', 'rescue',
'cheesy', 'line', 'yelling', 'singing', 'without', 'spank', 'Aaagh', 'after',
'Court', 'Anyway', 'glass', 'knights', 'Blue', 'Gable', 'burn', 'chance',
'daring', 'dona', 'setting', 'keep', 'pussy', 'pay', 'Never', 'Nothing',
"d'you", 'Round', 'Dragon', 'CAMERAMAN', 'who', 'wounded', 'Packing', 'Firstly',
'wise', 'Thy', 'Lady', 'Aaaaaaaaah', 'Beyond', 'worried', 'everything', 'waste',
'clunk', 'chickened', 'lads', 'um', 'Whoa', 'carried', 'That', 'trade', 'e',
'sun', 'blanket', 'bats', 'known', 'large', 'My', 'single-handed', 'leave',
'dorsal', 'into', 'farcical', 'liver', 'favor', "'T", '8', 'elderberries',
'search', 'swallows', 'act', 'KNIGHTS', 'Quickly', 'sworn', 'high-pitched',
'someone', 'ROBIN', 'Winter', "e'er", 'thirty-seven', 'DINGO', 'headed',
'accomplished', 'twenty-four', 'Pie', 'ungallant', 'sir', 'mangled', 'you',
```

```
'bite', 'If', 'wings', 'aside', 'least', 'interested', 'sonny', 'is', 'yel',
'Ha', 'crossed', 'Ay', 'Aramaic', 'Recently', 'listen', 'Bors', 'enjoying',
'rodent', 'mad', 'late', 'power', 'beacon', 'donkey-bottom', 'verses',
'kneecaps', 'LEFT', 'underwear', 'Assyria', 'die', 'Nay', 'internal', 'hello',
'splat', 'SIR', 'successful', 'Shall', 'OTHER', 'ever', 'eis', 'CRONE',
'varletesses', 'BORS', 'help', 'afraid', 'Arimathea', 'foot', 'Consult', 'hell',
'16', 'bless', 'third', 'yellow', 'curtains', 'aquatic', 'mashed', 'Rather',
'fifty', 'taunt', 'magne', 'lying', 'Aaaaaah', 'being', 'scimitar', 'than',
'silence', 'hamster', 'dressed', "'Morning", 'Today', 'tackle', 'cart',
'laurels', 'herring', 'smashing', 'For', 'bastards', 'packing', 'maintain',
'Death', 'for', 'north-east', "'ve", 'pansy', 'cadeau', 'starling', 'logically',
'carving', 'PERSON', 'To', 'writing', 'plain', 'again', 'chest', 'keeper', 'a',
'hospitality', 'bride', 'clap', 'each', 'temperate', 'lobbest', 'foe',
'gallantly', 'feet', 'travellers', 'At', 'gra', 'nasty', 'such', 'African',
'will', 'Alright', 'resumes', 'hee', 'The', 'repressing', 'father', 'haste',
'excepting', 'oui', 'derives', 'found', 'limbs', 'strange', 'yours', 'nearly',
'Hiyaah', 'great', 'sneaking', 'Say', 'and', 'Oui', 'two-level', 'passed',
'bridgekeeper', 'mightiest', 'there', 'worked', 'Back', 'WITCH', 'imperialist',
'vary', 'am', 'with', 'Providence', 'off', 'chosen', 'creak', 'required', 'Un',
';', 'those', 'soft', 'ptoo', 'behind', 'hidden', 'far', 'tiny', 'One',
'unplugged', 'BRIDE', 'Put', 'design', 'giggle', 'nervous', '--', 'eat',
'scenes', 'throwing', 'much', 'society', 'badger', 'town', 'beyond', 'simple',
'yes', 'many', 'FRENCH', 'stood', 'LOVELY', 'resting', 'Mud', 'U', 'Shut',
'best', '20', 'anyone', 'please', 'ehh', 'masses', 'guiding', 'Gawain', 'then',
'Try', 'depressing', 'pack', 'died', 'nostrils', 'mean', 'sheep', 'Haw', 'She',
'started', 'give', 'ROGER', 'court', 'right', "'Ooooooooh", 'Grenade', 'follow',
'wipers', 'courage', 'Steady', 'pound', 'MIDDLE', 'duty', 'wave', 'ham', 'own',
'GIRLS', 'CONCORDE', 'Action', 'every', 'ones', 'lunged', 'by', 'wishes',
'English', "C'est", 'weight', 'routines', 'witch', 'cry', 'clank', 'aloft',
'ere', 'frozen', 'together', 'Ninepence', 'guards', 'friend', 'buggered',
'only', 'groveling', 'immediately', 'expect', 'science', 'roar', 'what',
'legendary', 'expensive', 'CHARACTERS', 'delirious', 'Waa', 'never', 'Chop',
'Silly', 'Dingo', 'reads', 'Everything', 'Gallahad', 'testicles', 'Every',
'unhealthy', 'officer', 'Summer', 'Umhm', 'eccentric', "'First", 'sequin',
'moistened', 'fourth', 'bint', 'flights', 'had', 'SECOND', 'Where', 'knocked',
'wounding', 'could', 'differences', 'parts', 'noise', 'thwonk', 'MIDGET',
'suffice', 'purely', 'no', 'certainly', 'retold', 'gay', 'Most', 'vote',
'settles', 'fart', 'understanding', 'himself', 'depart', 'consulted', 'Bread',
'Quite', 'alarm', 'va.', 'hoo', 'teeth', 'flint', 'ye', 'wet', 'Hang',
'Crapper', 'worthy', 'NI', 'nothing', 'Bad', 'drilllll', 'or', 'minute',
'Launcelot', 'better', 'Hiyya', 'Hmm', 'stone', 'absolutely', 'turns',
'fooling', 'Thppppt', 'once', 'room', 'Huyah', 'u', 'jump', 'color', 'moment',
'my', 'empty', 'Make', 'near', 'There', 'How', 'demand', 'Armaments', 'knew',
'keen', 'sponge', 'wind', 'bicker', 'make', 'made', 'Psalms', 'round', 'idea',
'tear', 'MAN', 'preserving', 'hmm', 'bi-weekly', 'Are', 'grail-shaped', 'clang',
'face', 'Mercea', 'ninepence', 'tale', 'Olfin', 'set', 'are', 'use', 'bird',
',', 'Speak', 'velocity', 'aaaaaah', 'whinny', '23', "'ni", 'wicked', 'warned',
'unsingable', 'penalty', 'warning', 'persons', 'wants', 'Aaaaugh', 'stab',
```

```
'private', 'brain', 'Eternal', 'trouble', 'test', 'When', 'kill', 'Bristol',
'week', 'weighs', 'whop', 'else', 'fortune', 'bad-tempered', 'SHRUBBER',
'doors', 'soiled', 'bravest', '…', 'just', 'utterly', 'assault', 'its',
'PRINCE', 'swamp', 'says', 'quiet', 'nor', 'entering', 'mayest', 'pass',
'Quiet', 'apart', 'daft', 'ethereal', 'second', 'chickening', 'SENTRY', 'today',
'Explain', 'entrance', 'appease', 'Bravely', 'asking', 'Tower', 'stayed', 'It',
'cartoon', 'Throw', 'Shrubberies', 'heard', 'anarcho-syndicalist', '4',
'Leaving', 'vain', 'oral', 'Oooo', 'build', 'bond', 'outdoors', 'watch', 'ai',
'lonely', 'his', 'tinder', 'us', 'pull', "'aaggggh", 'three', 'Bedevere', 'say',
'King', 'Follow', 'tired', 'good', 'SOLDIER', 'saved', 'Said', 'bleeder',
'Bravest', 'Excalibur', 'get', 'newt', 'dramatic', 'sight', 'dine', 'Apples',
'arms', 'animal', 'wiper', 'longer', ')', 'done', 'temptress', 'purpose',
'disheartened', 'Why', 'sink', 'suit', 'previous', 'period', 'time', 'eyes',
'His', 'Far', 'Saxons', 'impeccable', "'cause", 'MONKS', 'well', 'c', 'answer',
'task', 'bleed', 'Fiends', 'helpful', 'sense', 'run', "'Aauuuuugh", 'wart', 's',
'enough', 'sell', "'Erbert", 'supports', 'samite', 'sing', 'clue', 'chu',
'adversary', 'weapon', 'Two', 'able', 'outdated', 'Once', 'Honestly', 'seemed',
'flesh', 'Since', 'totally', 'Off', 'As', 'Sir', 'master', 'Dramatically',
'feast', 'Hello', 'outwit', 'became', 'rocks', 'dying', 'taunting', 'mine',
'castanets', 'ride', 'tail', 'oo', 'floats', 'running', 'ceremony', 'sister',
'suppose', 'dirty', 'haaa', 'Auuuuuuuugh', ':', 'old', 'INSPECTOR', 'slightly',
'man', 'bunny', 'suspenseful', 'bum', 'going', 'Anybody', 'DENNIS', 'bottoms',
'at', 'vache', 'excuse', 'Uuh', 'somewhere', 'already', 'remembered', 'y',
'capital', 'left', 'hat', 'seem', 'BROTHER', 'dragging', 'about', 'deeds',
'afoot', 'retreat', 'table', 'stretched', 'eisrequiem', 'really', 'full',
'Surely', 'gon', 'Piglet', 'vicious', 'given', 'Your', 'LUCKY', "'aaaah",
'towards', 'cruel', 'Too', 'Be', 'ratified', 'Picture', 'basic', 'silly',
'Over', 'types', 'bugger-folk', 'k-nnnnniggets', 'Eh', 'until', 'enchanter',
'You', 'rhymes', 'scribble', 'trough', 'so-called', 'of', 'uuggggggh', 'order',
'Do', 'Saint', 'thanks', "'round", 'Away', 'vests', 'witness', 'presence',
'bowels', 'banana-shaped', 'ridden', 'Pure', 'bladders', 'fair', 'dictatorship',
'supreme', 'Monsieur', 'union', 'Holy', 'present', "'shrubberies", 'ferocity',
'Mmm', 'five', 'worse', 'streak', 'Heh', 'door-opening', '15', 'Meanwhile',
'night', 'way', 'BEDEVERE', 'took', 'mangy', 'sometimes', 'Galahad', 'nose',
'averting', 'moooooooo', 'etc', 'O', 'CRASH', 'PARTY', 'know', 'Fine', 'window-
dresser', '13', 'Is', 'Guards', 'blood', 'dress', 'plan', 'castle', 'she',
'Concorde', 'CARTOON', 'RIGHT', 'mud', 'sire', '9', 'proceed', 'request',
'AMAZING', 'Cider', 'either', 'pen', 'amazes', 'Pull', 'confuse', 'bangin',
'raised', 'kneeling', 'TIM', 'safety', 'protect', 'Thee', 'an', 'invincible',
'liar', 'looks', 'to', 'any', 'Hm', 'dark', 'Britain', 'PATSY', 'new',
'Aauuugh', 'Between', 'stupid', 'strand', 'requiem', 'tragic', 'deal', 'pointy',
'special', 'HEAD', 'eight', 'Roger', 'women', 'performance', 'N', 'workers',
'young', 'Wait', 'has', 'Did', 'collective', 'auntie', 'bid', 'havin',
"'uuggggggh", 'Look', 'non-migratory', 'decision', 'tart', 'Five', 'small',
'Midget', 'spake', "'To", 'chord', 'violence', 'uuup', 'bridges', 'pestilence',
'formidable', 'Could', 'Wood', 'Ask', 'b', 'year', 'still', 'thonk', 'CUSTOMER',
'main', 'guarded', 'GUARD', 'Ni', 'course', 'knows', 'Grail', 'used', 'hast',
'kills', 'pram', 'terribly', 'Tale', 'shivering', 'dub', 'kick', 'walk',
```

```
'wield', 'discovered', 'see', 'matter', 'miserable', 'threw', 'Build', 'chorus',
'quite', 'g', 'Old', 'sigh', 'MAYNARD', 'manner', 'sloths', 'Churches', 'bows',
'accompanied', 'Robin', 'hiyaah', 'necessary', 'Even', 'grip', 'seen', 'Our',
'remember', 'Must', 'Hallo', 'danger', 'sent', 'sod', 'built', 'armor', 'shut',
'Hold', 'swallow', 'Badon', 'bottom', 'Hey', 'closest', 'Hoa', 'Lancelot',
'dare', 'he', 'said', 'on', 'leaps', 'sweet', 'p', 'apologise', 'smashed',
'Will', 'whether', 'Woa', 'Bedwere', 'along', 'elbows', 'country', 'yourself',
'Alice', 'named', 'defeat', 'majority', 'spooky', 'WINSTON', 'But', 'w',
'knight', 'shrubber', 'forty-three', 'living', 'home', 'head', 'become',
'understand', 'cover', 'minutes', 'Christ', 'Robinson', 'folk', 'Nine',
'considerable', 'Nu', 'tell', 'exciting', 'gouged', 'Man', 'occasion', 'Stop',
'around', 'stop', 'lies', 'path', 'would', 'coconuts', 'king', 'guest',
'discovers', 'owns', 'were', '3', 'met', 'Pin', 'Castle', 'intermission',
'sign', 'sovereign', 'VILLAGERS', 'rrrr', 'Chicken', 'commands', 'kind',
'orangutans', '(', 'put', 'pissing', 'GREEN', 'let', '21', 'shimmering',
'names', 'pulp', 'ready', 'Together', 'ca', 'bravely', 'since', 'kings',
'Cherries', 'Like', 'Tell', 'Hurry', 'spanking', 'tit', "'m", 'continue',
'Attila', 'ha', 'lucky', 'later', 'Aggh', 'trumpets', 'wait', 'killer',
'Battle', 'Am', 'if', 'out', 'scots', 'Ector', 'breadth', 'Eee', 'dunno',
'Mind', 'wayy', 'now', 'weather', 'Britons', 'Thppt', 'brunettes', 'pause',
'Joseph', 'n', 'wo', "'re", 'Not', 'dogma', 'Burn', 'scarper', 'these', 'scene',
'Um', 'Ohh', 'Beast', 'May', 'Aah', 'I', 'couple', 'bringing', 'Knight',
'commune', 'bridge', 'mumble', 'burst', 'thought', 'frontal', 'CART-MASTER',
'our', 'makes', 'FATHER', "'old", 'winter', 'Lake', 'distress', 'carved',
'ways', 'sharp', 'land', 'lord', 'harmless', 'OLD', 'war', 'legs', 'j',
'Defeat', 'wooden', 'God', 'Greetings', 'kicked', 'Nador', 'thump',
'government', 'away', 'Aauuuves', 'ignore', "'Here", 'shit', 'humble', 'Bon',
'north', 'RANDOM', 'Schools', 'Stay', 'distributing', 'oooh', 'number', 'might',
'saying', 'brush', 'bit', 'sample', 'Use', 'even', 'stuffed', 'always', 'Keep',
'guard', 'Well', 'scrape', 'Autumn', 'Thsss', 'economic', 'push', 'business',
'nineteen-and-a-half', 'On', 'GOD', 'Antioch', 'icy', 'Aauuggghhh', 'Or', 'sad',
'In', 'quarrel', 'fled', 'dance', 'under', 'armed', 'acting', 'woods', 'Lead',
'false', 'biggest', 'forget', 'husk', 'un', 'spanked', 'meeting', 'Peril',
'Jesus', 'diaphragm', 'we', 'down', 'sex', 'wide', 'lady', 'WIFE', 'Ho',
'vital', 'real', 'ni', 'affairs', "'sorry", 'grin', 'spoken', 'buy', 'want',
'medieval', 'Get', 'refuse', 'executive', 'agree', 'aptly', 'less', 'rock',
'scholar', '.', 'beautiful', 'howl', 'idiom', 'grovel', 'married', 'OF',
'sword', 'call', 'bits', 'Order', 'Agh', 'GUESTS', 'Wayy', 'example', 'Forgive',
'They', 'Please', 'lad', 'horn', 'Arthur', 'carve', 'Swamp', 'direction',
'advancing', 'GUEST', 'bois', 'Seek', 'shalt', 'may', 'remain', 'inside',
'split', 'la', 'offensive', 'hundred-and-fifty', 'defeator', 'lose',
'exploiting', 'back', 'personally', 'arrows', 'ladies', 'birds', 'social',
'assist', 'entered', 'strangers', 'this', 'Run', 'word', 'carrying', 'worry',
'zone', 'angels', 'bold', 'worst', 'Mine', 'like', 'ENCHANTER', 'Brother',
'praised', 'Prepare', 'pig', 'major', 'relics', 'time-a', 'handle', 'SUN',
'twin', 'Stand', 'told', 'ho', 'point', 'Shrubber', 'cut', 'What', 'leads',
'Uh', 'Neee-wom', 'move', 'grips', 'rope', 'laughing', 'chops', 'fly', 'treat',
'triumphs', 'covered', "'Ni", 'telling', 'THE', 'walking', 'dappy', 'Not-
```

```
appearing-in-this-film', '[', 'find', 'GALAHAD', 'zoosh', 'case', 'tree',
'recover', 'Those', 'Tall', 'England', 'drink', 'warmer', 'No', 'reared',
'profane', 'from', 'twong', 'awaits', 'jam', 'fatal', 'Hic', 'not', 'grenade',
'knees-bent', 'dungeon', 'KING', 'Camaaaaaargue', 'Mother', 'thing',
'approaching', 'tea', 'nice-a', 'risk', 'Hiyah', 'somebody', 'And', 'islands',
'Found', 'been', 'person', 'high', 'rewr', 'hand', 'air-speed', 'aunties',
'pond', 'Hill', 'enter', 'STUNNER', 'purest', 'Hyy', 'traveller', 'lived',
'ugly', 'marrying', 'holy', 'son', 'almost', 'Y', 'me', 'things', 'quests',
'bitching', 'lambs', 'outside', 'other', 'words', 'effect', 'can', 'supposed',
'everyone', 'mayhem', 'We', 'mooo', 'Frank', 'Until', 'Hoo', 'stand', 'Fetchez',
'Great', 'ran', 'band', 'creeper', 'Charge', 'An', 'lost', '14', 'Bridge',
'bother', 'Brave', '11', 'Allo', 'Lie', 'quick', 'hopeless', 'PRISONER',
'bloody', 'Spring', 'Supreme', 'marry', 'WOMAN', 'forward', 'sort', 'busy',
'Actually', 'Splendid', 'fell', 'Looks', 'learning', 'ZOOT', 'hacked', 'the-not-
quite-so-brave-as-Sir-Lancelot', 'spirit', 'most', 'hall', 'whispering', '10',
'Message', 'basis', "'S", 'NARRATOR', 'little', 'reasonable', 'warm', 'Who',
'convinced', 'Bones', 'cross', 'awaaaaay', 'last', 'think', 'turned', 'bed',
'Ayy', 'alive', 'nobody', 'shrubbery', 'leg', 'biscuits', 'show', 'people',
'Himself', 'DEAD', 'the', 'give-away', 'forced', 'dead', 'riding', 'guests',
'beat', 'Chickennn', 'got', 'shelter', 'emperor', 'valleys', 'completely',
'snows', 'Herbert', 'indefatigable', 'radio', 'stay', 'clad', 'Lord', 'that',
'therefore', 'filth', '7', 'binding', 'your', 'wrong', 'visually', 'beside',
'Yeah', 'ounce', 'smelt', 'Iiiiives', 'crying', 'largest', 'lair', 'nice',
'undressing', 'gentle', 'Iesu', '!', 'Quick', 'Peng', 'crone', 'shows', 'food',
'system', 'legally', 'training', 'baby', 'impersonate', '24', 'Divine', 'Aagh',
'Of', 'keepers', 'glad', 'Four', 'CROWD', 'rabbit', 'buggering', 'joyful',
'PIGLET', 'horse', 'twang', 'suffered', 'doubt', 'boil', 'Cut', 'fire', 'Heee',
'illegitimate-faced', 'very', 'proved', 'finest', 'Oh', 'dad', 'doctors', 'l',
'happy', 'answers', 'tropical', 'men', 'uh', 'VILLAGER', 'Ives', 'lobbed',
'saw', 'tough', 'guided', 'grail', 'hang', 'foul', 'Good', "'ll", 'sank',
"'Ecky-ecky-ecky-ecky-pikang-zoop-boing-goodem-zoo-owli-zhiv", 'duck', 'end',
'certain', '12', 'autonomous', 'scared', 'cop', 'See', 'mile', 'er',
'Therefore', 'count', 'Here', 'surprise', 'Running', 'felt', 'naughty', 'thy',
'Pendragon', 'Maynard', 'taken', 'Sorry', 'lot', 'KNIGHT', 'Then', 'European',
'dictating', 'return', 'lie', 'sixteen', 'draw', 'haw', 'removed', 'Zoot',
'object', 'Unfortunately', 'suddenly', 'Hya', 'na', 'go', 'pweeng', 'place',
'Now', 'length', 'terrible', 'Idiom', 'dangerous', 'it', 'heeh', 'charged',
'join', 'look', 'lapin', 'prevent', 'Winston', 'gravy', 'Erm', 'Gorge',
'rather', 'Yeaah', 'question', 'B', 'thine', 'easy', 'Uther', 'Supposing',
'looking', 'compared', 'change', 'Patsy', '?', 'nearer', 'Hand', 'police',
'dull', 'did', 'two-thirds', 'merger', 'git', 'twenty', 'strongest', 'clear',
'servant', "'forgive", 'art', 'clop', 'eats', 'types-a', 'how', 'inherent',
'should', 'nightfall', 'LAUNCELOT', 'all', 'nine', 'middle', 'earthquakes',
'trusty', 'peasant', 'Yes', 'whom', 'anyway', 'string', 'hear', '..', 'OFFICER',
'him', 'Behold', 'carries', 'kingdom', 'biters', 'need', 'Farewell', 'snap',
'Ow', 'Would', 'frighten', "'em", 'boom', 'Thou', 'Bloody', 'Torment', 'which',
'was', 'quest', 'o', 'arm', 'against', 'tops', 'Uugh', 'Twenty-one', 'Knights',
'19', 'pig-dogs', 'creep', 'minstrels', 'mercy', 'Ridden', 'side', 'times',
```

```
'progress', 'Iiiives', 'sniff', 'allowed', 'runes', 'cave', 'Caerbannog', 'ju',
'gave', 'mandate', 'her', 'sorry', 'shrubberies', 'Tim', 'Dappy', 'Amen',
'having', 'wherein', 'luck', 'try', 'further', 'squeak', 'Chapter', 'scales',
'Halt', 'HERBERT', 'Three', 'yeah', 'higher', 'bosom', 'feathers', 'Father',
'formed', 'Open', 'burned', 'note', 'soon', 'glory', 'ARTHUR', 'does', 'come',
'flight', 'looney', 'because', 'seek', 'Black', 'maybe', 'ARMY', 'finds', 'W',
'#', 'Clear', "'anging", 'Chaste', 'cough', 'bones', 'Aaaah', 'happens', '2',
'Riiight', 'anywhere', 'witches', 'one', 'More', 'earth', 'temptation',
'Welcome', 'forth', 'HISTORIAN', 'hills', 'passing', 'Prince', 'carp', 'tracts',
'throat', 'woman', 'anything', 'sacrifice', 'bastard', 'killed', 'knock',
'forest', 'Remove', 'Let', 'eet', 'heroic', 'auuuuuuuugh', 'live', 'coming',
'particular', 'leap', "'Ere", 'so', 'strategy', 'arrange', 'looked', 'classes',
'Princess', 'another', 'sure', 'vouchsafed', 'where', '17', 'strength', 'fruit',
'conclusion', 'awfully', 'Ah', 'easily', 'although', 'alight', 'Thank', 'bonk',
'general', 'nibble', 'strewn', 'All', 'liege', 'fought', 'Help', 'life', 'van',
'Other', 'language', 'regulations', 'something', 'mortally', 'Oooohoohohooo',
"n't", 'fine', 'Camelot', 'making', 'evil', 'HEADS', 'raped', 'rode', 'cost',
'None', 'Yay', 'king-a', 'SCENE', 'why', 'Ewing', 'properly', 'Go', 'enemies',
'here', 'Clark', 'autocracy', 'Exactly', 'Ahh', 'bet', 'summon', 'some',
'imprisoned', 'quack', 'Enchanter', 'valor', 'cereals', 'migrate', 'through',
'particularly', 'Rheged', 'Ulk', 'handsome', 'sawwwww', 'woosh', 'south', 'out-
clever', 'thou', 'Hah', 'their', 'individually', 'command', 'punishment',
'snuff', 'Bring', 'scott', 'attend', 'fight', 'dynamite', 'Cornwall', 'ask',
'By', 'k-nnniggets', '22', 'baaaa', 'whose', 'uhh', 'same', 'Book', 'Doctor',
'ponds', 'be', "'Man", 'heh', 'straight', 'Guy', '18', 'With', 'Yeaaah', 'cast',
'inferior', 'miss', 'horrendous', 'bang', 'returns', 'Victory', 'long', 'over',
'France', 'song', 'more', 'them', 'Quoi', 'as', 'So', 'held', 'Table', "'",
'splash', 'medical', 'blondes', 'Excuse', 'aaugh', 'animator', 'BRIDGEKEEPER',
'outrageous', 'This', 'automatically', 'nick', 'awaaay', 'Thpppt', 'class',
'rejoicing', 'feel', 'breakfast', 'Aaauggh', 'rich', 'clack', 'ratios', 'relax',
'problems', 'fold', 'talk', 'Thpppppt', 'de', 'stew', 'A', 'reached',
'questions', "'d", 'tiny-brained', 'influential', 'asks', 'escape', 'name',
'signifying', 'separate', 'changed', 'lives', 'pimples', 'Ages', 'gone',
'Loimbard', 'cope', 'heart', 'chastity', 'needs', 'plover', 'Come', 'coconut',
'decided', 'Aaaaaaaah', "'til", 'approacheth', 'ruffians', 'Really', 'siren',
'honored', 'Thursday', 'mother', 'jokes', 'argue', 'eh', 'mer', 'bells', 'Does',
'explain', 'wound', '5', 'ooh', "'it", 'creature', 'Shh', 'when', 'Almighty',
'speak', 'seldom', 'headoff', 'upon', 'indeed', 'voluntarily', 'ALL', 'Anarcho-
syndicalism', 'Perhaps', 'first', "'Course", '1', 'CHARACTER', '6', 'ill.',
'went', 'behaviour', 'Forward', 'next', 'accent', 'breath', 'between', 'while',
'two', 'bring', 'using', "'Dennis", 'gained', 'day', 'house', 'take', 'swords',
'unladen', 'blow', 'sons', 'model', 'do', 'work', 'ordinary', 'thud', 'dancing',
'broken', 'CRAPPER', 'history', 'Listen', 'Yapping', 'problem', 'wonderful',
'wood', 'getting', 'send', 'bed-wetting', 'have', 'tie', 'halves', 'too', 'yet',
'scratch', 'dressing', 'Oooh', 'dear', 'counting', "'s", 'tap-dancing',
'lovely', 'BLACK', 'stops', 'Uhh', 'PRINCESS', 'chanting', 'He', 'laden',
'electric', 'opera', 'also', 'ours', 'Umm', 'Just', 'Lucky', 'Huh', 'Practice',
'DIRECTOR', 'snore', 'martin', 'valiant', 'Aaaugh', 'welcome', 'brave', 'mac',
```

```
'bathing', 'Hee', 'Dis-mount', 'self-perpetuating', 'score', 'slash', 'big',
'Have', 'thank', 'attack', 'peril', 'freedom'}
```

▶ **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?

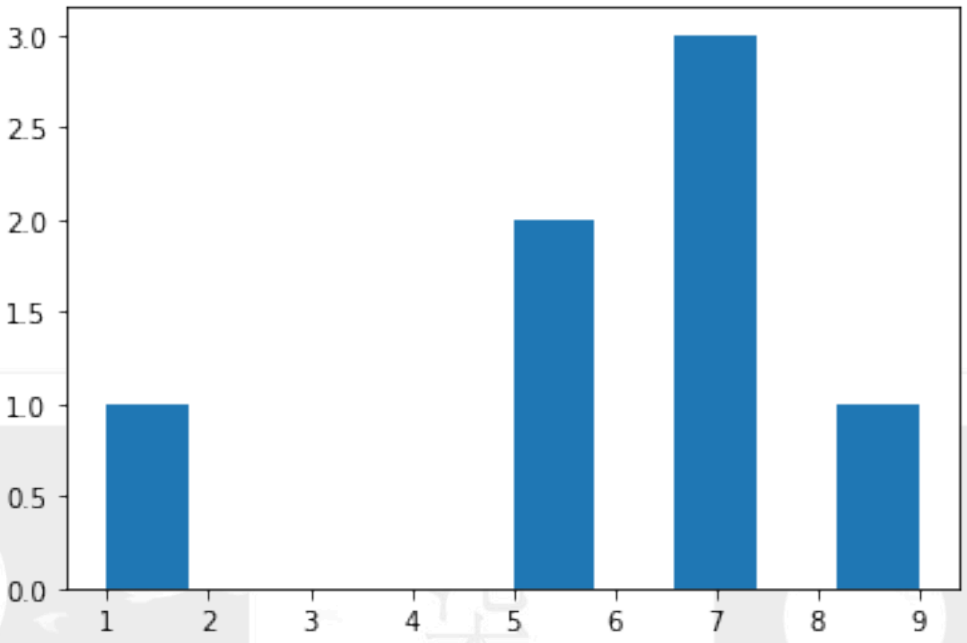
- It is a charting library used by many open-source Python projects.
- It has straightforward functionality with lots of options:
  - *histograms*
  - *bar charts*
  - *line charts*
  - *scatter plots*
- And also, it has advanced functionality like 3D graphs and animations!

4.2 Code of plotting a histogram with matplotlib:

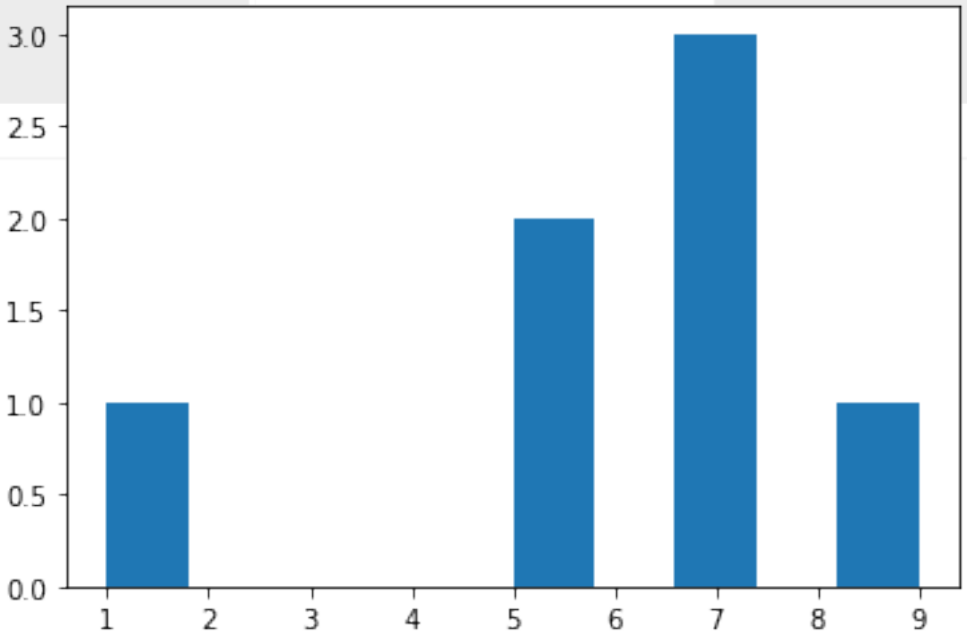
[40]:
```
from matplotlib import pyplot as plt

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()
```

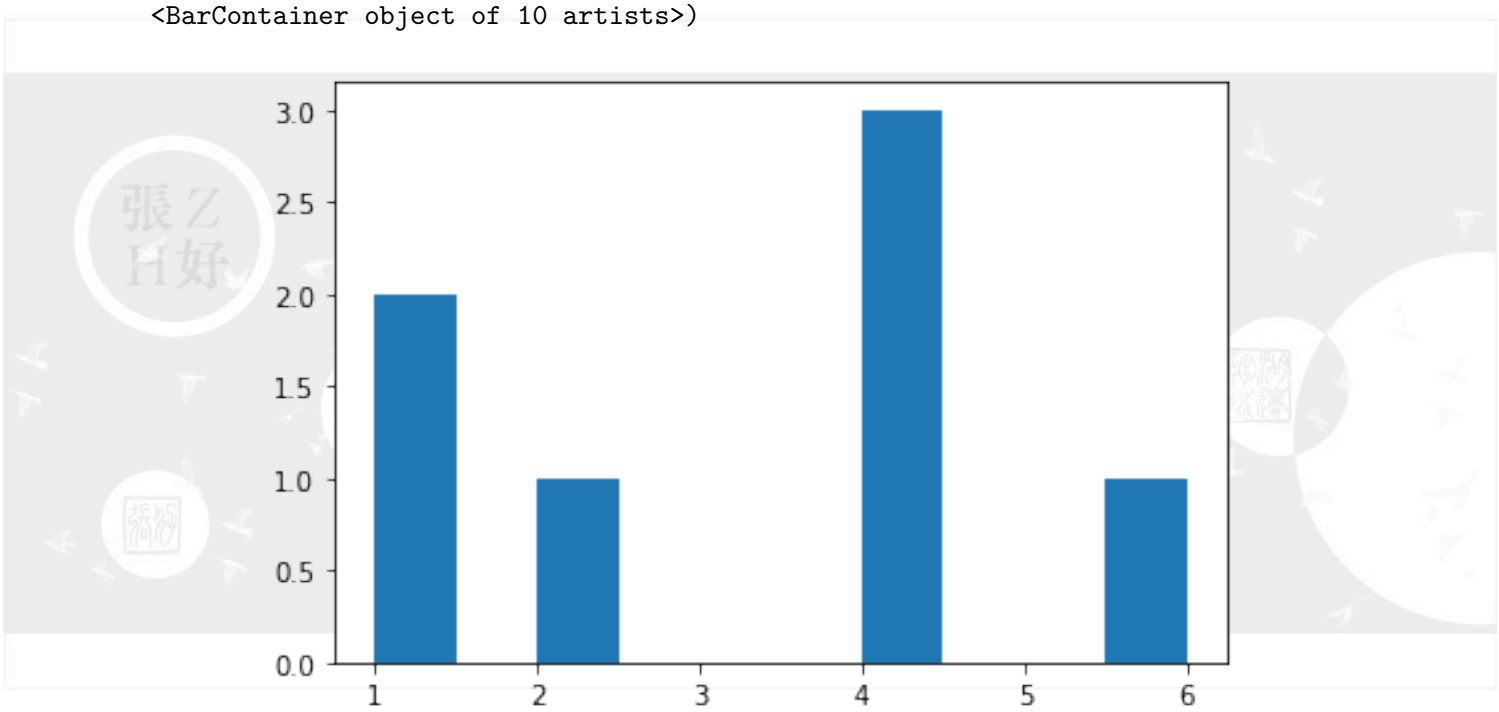


4.3 Code of combining NLP data extraction with plotting:

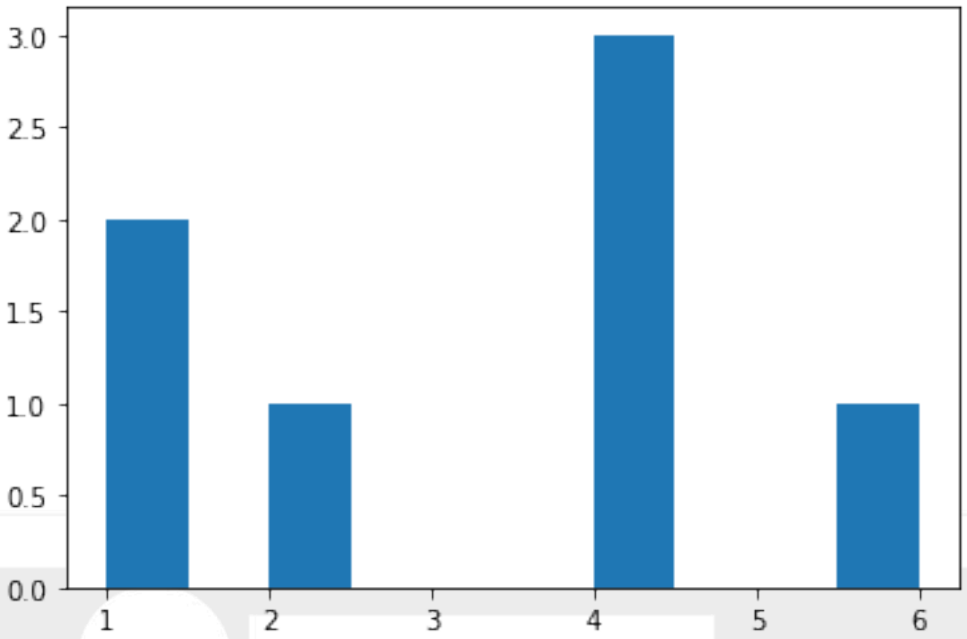
[42]:
```
from matplotlib import pyplot as plt
from nltk.tokenize import word_tokenize

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()
```