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)
```

```
{'suggesting', 'sharp', 'comin', 'Doctor', 'autonomous', 'whispering', 'making',
'time', 'streak', 'bother', 'HEAD', 'depressing', 'word', 'scenes', 'sigh',
'could', "'forgive", 'friend', 'couple', 'Have', 'ye', 'Ahh', 'dine', 'cheesy',
'wounded', 'blood', 'Really', 'split', 'problem', 'follow', 'same', 'names',
'Ow', 'man', 'her', 'own', 'escape', 'Uh', 'mayhem', 'Alright', 'confuse',
'spooky', 'keen', 'suit', 'Most', 'assault', 'quarrel', 'clack', 'MONKS',
'ARMY', 'What', 'jam', 'TIM', 'ptoo', 'n', 'quest', 'curtains', 'More', "'Ere",
'somebody', 'two', 'Consult', 'window-dresser', 'OTHER', 'wide', "'Erbert",
'behaviour', 'Tower', 'parts', 'Aaaaaaaah', 'yourself', 'Other', 'question',
'5', 'sent', 'pansy', 'Idiom', 'brain', 'Jesus', 'Thpppt', 'Chickennn', 'pig',
'SECOND', 'treat', 'busy', 'CHARACTER', 'end', 'us', 'jokes', 'HERBERT', 'Uuh',
'creep', 'Winter', 'worked', 'second', 'Nador', 'None', 'sword', 'aptly',
'Beast', 'GIRLS', 'resting', 'supposed', '22', 'stupid', "'old", 'CHARACTERS',
'joyful', 'Oui', 'Chicken', 'frozen', 'approacheth', 'amazes', 'Welcome',
'carries', 'last', 'Thank', 'Between', 'buy', "'s", 'deeds', 'model', 'why',
'18', 'present', 'undressing', 'forced', 'distributing', 'Yeaah', 'new',
'Nothing', 'laden', 'cope', 'Put', 'headed', 'major', 'unhealthy', '1', 'BLACK',
'certain', 'crone', 'Iiiiives', "'Man", 'Piglet', 'KNIGHT', 'easy', 'feel',
'21', 'aloft', 'rejoicing', 'never', 'path', 'Tale', 'apart', 'Camaaaaaargue',
'send', 'sign', 'pass', 'stew', 'south', 'to', 'Bloody', 'icy', 'VILLAGERS',
'together', 'No', 'master', 'Thppt', 'reared', 'git', 'Oh', 'Dappy', 'relics',
'working', 'any', 'RANDOM', 'twong', 'progress', '15', 'wo', 'birds', 'society',
'BROTHER', 'ungallant', 'prevent', 'SIR', 'also', 'tired', 'bugger-folk',
'keeper', 'father', 'carving', 'pig-dogs', 'spanked', 'Huy', '!', 'tit',
'Three', 'Battle', 'buggering', 'shrubber', 'burned', 'tinder', 'by', 'police',
```

```
'harmless', 'changed', 'safety', 'Hmm', 'formed', 'rabbit', 'door-opening',
'four', 'turned', 'CRAPPER', 'thy', 'purest', 'winter', 'pond', 'penalty',
'chest', 'Two', 'tale', 'He', 'separate', 'feint', 'twin', 'sod', 'point',
'towards', 'Attila', 'leaps', 'spake', 'stand', "'Ecky-ecky-ecky-ecky-pikang-
zoop-boing-goodem-zoo-owli-zhiv", 'My', 'Hoa', 'build', 'previous', 'lie',
'Nine', 'Old', 'asking', 'Bravely', 'Cut', 'stab', 'cave', 'Like', 'right',
'French', 'bad', '7', 'Bravest', 'its', 'hospital', 'really', 'repressed',
'sun', 'with', 'shrubbery', 'ill.', 'Book', 'Hah', 'Yapping', 'ninepence',
'hand', 'out-clever', 'humble', 'looking', 'oui', 'tie', 'sonny', 'packing',
'dance', 'nibble', 'beautiful', 'afoot', 'dorsal', 'conclusion', 'hidden',
'profane', 'When', 'anywhere', 'Oooh', 'arms', 'Throw', 'laurels', 'two-thirds',
'Roger', 'VOICE', 'guiding', 'excuse', 'knew', 'automatically', 'Nu', 'war',
'MIDDLE', 'ours', 'Himself', 'OLD', 'woosh', 'obviously', 'under', '--', 'told',
'Aaah', 'Joseph', 'BRIDE', 'rhymes', 'twenty-four', 'knights', 'Is',
'horrendous', 'Assyria', 'Lord', 'GOD', 'Firstly', 'Hic', 'internal', 'Aauuugh',
'opera', 'Ah', 'Leaving', 'i', 'took', 'purpose', 'Today', 'whom', '4',
'dictating', 'eccentric', 'leads', 'gon', 'hast', 'lovely', 'goes', 'of', 'On',
'PERSON', 'number', "'Aauuuuugh", 'eight', 'ARTHUR', 'dress', 'types-a', 'foul',
'guests', 'person', 'needs', 'Sir', 'reached', 'retreat', 'it', 'See',
'drillllll', 'Ulk', 'uhh', 'trouble', 'imprisoned', 'cough', 'moment', 'fart',
'mud', 'bringing', 'gravy', 'holy', 'tell', 'CUSTOMER', 'She', 'indeed', 'has',
'dub', 'pure', 'expect', ')', 'ehh', 'groveling', 'further', 'special', 'went',
'farcical', 'merger', 'looked', 'clang', 'count', 'Eh', 'became', 'smashing',
"'is", 'PRISONER', 'migrate', 'GREEN', 'awhile', 'Apples', 'much', 'whoever',
'shows', 'Aaaaaah', 'Ayy', 'van', 'eyes', 'writing', 'Seek', 'some', 'Of',
'Supposing', 'glory', 'sight', 'nearly', 'Great', 'Arimathea', 'There', 'PATSY',
'nineteen-and-a-half', 'clunk', 'lost', 'warned', 'score', 'Thee', 'Bread',
'smack', 'aaugh', 'oooh', 'Gallahad', 'yes', 'Frank', 'understand', 'shall',
'If', 'medical', 'smelt', '3', 'Ho', 'Providence', 'guards', 'Aauuggghhh',
'aunties', 'ju', 'shimmering', 'True', 'act', 'table', 'stop', 'coming',
'Riiight', 'masses', 'outrageous', 'centuries', 'praised', 'nasty', 'certainly',
'brush', 'Haw', 'country', 'look', 'Rheged', 'eats', 'already', 'VILLAGER',
'glad', 'advancing', 'Gable', 'twang', 'awfully', 'purely', 'leg', 'gay', 'Hey',
'beyond', 'supreme', "'Dennis", 'THE', 'Quiet', 'testicles', 'around', 'third',
'oral', '8', 'Lie', 'fourth', 'verses', 'class', "'Morning", 'simple', 'might',
'chu', 'Go', 'Anybody', 'dona', 'too', 'liege', 'carry', 'outdoors', 'ridden',
'maybe', 'Lucky', 'seems', 'Bridge', 'else', 'scales', 'son', 'broken',
'walking', 'earth', 'angels', 'Surely', 'la', 'frontal', 'relax', 'vital',
'Oooo', 'Um', 'arm', 'spam', 'suspenseful', 'INSPECTOR', 'two-level', 'be',
'enemies', 'wicked', 'assist', 'shut', 'plain', 'DEAD', 'Sorry', 'WIFE',
'known', "'m", 'inherent', 'singing', 'quiet', 'bitching', 'Now', 'Brother',
'Listen', 'imperialist', 'art', 'cart', 'wiper', 'weight', 'swallows', ':',
'Hang', 'CARTOON', 'intermission', 'married', 'worry', 'interested', 'eis',
'join', 'With', 'taken', 'um', 'carrying', 'social', 'dear', 'j', 'horse',
'Grail', 'zoosh', 's', 'ENCHANTER', 'newt', 'Shall', 'therefore', 'utterly',
'Almighty', 'using', 'bottoms', 'Our', 'the-not-quite-so-brave-as-Sir-Lancelot',
'Clark', 'who', 'sworn', 'get', 'bed', 'eisrequiem', 'bells', 'They', 'Iesu',
'foot', 'everyone', 'tropical', 'Launcelot', 'pay', 'binding', 'keep', 'left',
```

```
'LAUNCELOT', 'employed', 'nostrils', 'down', 'Which', 'Bring', 'bravely',
'presence', 'SOLDIER', 'Pure', 'weather', 'bladders', 'diaphragm', 'Hiyya',
'Pie', 'idea', 'LOVELY', 'Waa', 'clop', 'BEDEVERE', 'lived', "'First", 'door',
'for', 'weighs', 'brave', 'pen', 'Monsieur', 'Running', 'Saint', 'tea',
'object', '19', 'Christ', 'Lady', 'illustrious', 'wait', 'Hallo', 'daughter',
'swamp', 'other', 'quite', 'best', 'dressing', 'At', 'example', 'bang',
'Cherries', 'baby', 'successful', 'lord', 'Forgive', 'absolutely', 'Clear',
'evil', 'Umm', 'Will', 'creeper', 'Since', "n't", 'blow', 'king', 'Aagh',
'Skip', 'teeth', 'vain', 'Fine', 'lambs', 'spoken', 'passed', 'change', 'cop',
'put', 'Spring', 'haste', 'more', 'Away', 'Ninepence', 'how', 'yellow',
'delirious', 'fortune', 'y', 'kick', 'Forward', 'pweeng', 'vests', 'hills',
'travellers', 'had', 'danger', "'uuggggggh", 'nearer', 'Pin', 'happens',
'apologise', 'push', 'seem', 'siren', 'A', 'expensive', 'someone', 'feathers',
'10', '16', 'underwear', 'day', 'This', 'call', 'himself', 'that', 'Agh',
'donaeis', 'mer', 'coconut', 'honored', 'he', 'band', 'course', 'yeah', 'bint',
'consulted', 'Pull', 'strength', 'helpful', 'where', 'always', 'scrape', 'carp',
'seen', 'Swamp', 'splat', 'Neee-wom', 'ROBIN', 'ratified', 'just', 'quick',
'dogma', 'l', 'defeator', 'autocracy', 'little', '14', 'Halt', 'Princess',
'heeh', 'bet', 'business', 'or', 'p', 'chorus', 'Picture', 'GALAHAD', 'bows',
'you', 'hundred-and-fifty', 'history', 'fled', 'waste', 'search', 'order',
'sweet', "'til", 'explain', 'Thpppppt', 'mooo', 'That', 'hall', 'medieval',
'Shut', 'protect', 'Those', 'Mud', 'proceed', 'shrubberies', 'haaa', '17', '#',
'in', 'considerable', 'since', 'bottom', 'Caerbannog', 'used', 'mumble', 'help',
'slightly', 'Just', 'Galahad', 'foe', 'Ridden', 'Thsss', 'pound', 'even',
'inferior', 'Meanwhile', 'Iiiives', 'felt', 'ANIMATOR', 'GUEST', 'lot',
'bridge', 'Death', 'lies', 'Ooh', 'ca', 'ha', 'must', 'direction', 'Brave',
'Thou', 'request', 'Yes', 'perpetuates', 'wedding', 'although', 'k-nnniggets',
'Come', 'e', 'ruffians', 'outside', 'become', 'tackle', 'sneaking', 'side',
'ROGER', 'hamster', 'scott', 'Fiends', 'Did', 'wayy', 'auuuuuuuugh', 'bats',
'flights', 'Cider', 'violence', 'gone', 'quests', 'pull', 'pussy', 'auntie',
'Pendragon', 'liar', 'strangers', 'seldom', 'risk', 'maintain', 'frighten',
'Cornwall', 'alive', 'young', 'tragic', 'snore', 'bi-weekly', 'Gawain',
'stuffed', 'eet', 'anyway', 'live', 'fair', 'today', 'dying', 'straight', 'is',
'Grenade', 'meeting', 'him', 'Unfortunately', 'WINSTON', 'sorry', 'thou', 'gra',
'the', 'NARRATOR', 'Victory', 'Four', 'built', "'aaaah", 'string', 'Off',
'power', 'lives', 'would', 'limbs', 'shit', 'something', 'CRONE', 'warmer',
'guided', 'Aaauugh', 'wishes', 'Explain', 'quack', 'pulp', 'through', 'Antioch',
'Stop', 'N', 'Said', 'somewhere', 'living', 'ladies', 'MAN', 'bunny', 'sort',
'training', 'SENTRY', 'exploiting', 'marry', 'clear', 'fwump', 'mangy',
'nervous', 'Quickly', 'forth', "'Course", 'then', 'better', 'alight', 'bad-
tempered', 'science', 'fifty', 'Aaaaaaaaah', 'this', 'taking', 'Concorde',
'avenged', 'they', 'lapin', 'Anarcho-syndicalism', 'dull', 'chance', 'Tell',
'punishment', 'Bedevere', 'sixteen', 'yelling', 'Summer', 'forget', 'Angnor',
'arrows', 'doors', "'To", 'lobbest', 'woods', 'should', 'rope', 'north-east',
'KNIGHTS', 'You', "'T", 'hear', 'Alice', 'ZOOT', 'suffered', 'again', 'eh',
'peasant', 'bold', 'nice-a', 'been', 'kneecaps', 'mayest', 'Aah', 'WITCH',
'silence', 'CAMERAMAN', 'Tim', 'Robin', 'bum', 'mad', 'mile', 'bleeder',
'heads', 'grenade', 'breakfast', 'Patsy', 'Aaaugh', 'Knights', 'stretched',
```

```
'runes', 'persons', 'head', 'Prince', 'Thy', 'Your', 'SHRUBBER', 'large',
'show', 'Hold', 'as', 'samite', 'thump', 'Recently', 'go', 'Beyond', 'bathing',
'Bad', 'their', 'ask', 'answer', 'creak', 'mistake', 'workers', 'name',
'halves', 'clank', 'temptress', 'year', 'things', 'chickened', 'them', 'long',
'favorite', 'wings', 'easily', 'raped', 'bravest', 'take', 'Keep', 'at',
'union', 'ceremony', 'test', 'do', 'uh', 'longer', 'remembered', 'met',
'Mercea', 'Try', 'awaits', 'va.', 'bowels', "'shrubberies", 'please',
'MINSTREL', 'Aauuuves', 'cast', 'scarper', 'Camelot', 'handsome', 'lady',
'Well', 'talk', 'tiny-brained', 'task', 'o', 'high', 'Uugh', 'town', 'tail',
'Table', 'biggest', 'Speak', 'elbows', 'dictatorship', 'Behold', 'kicked',
'along', 'mangled', 'Farewell', 'emperor', 'knock', 'weapon', 'draw', 'Let',
'temptation', 'cost', 'wise', 'swallow', 'worried', 'ethereal', 'wherein',
'pounds', 'warm', 'visually', 'thank', 'Hya', 'wield', 'stay', 'saved',
'FRENCH', 'un', 'thing', 'cry', 'eat', 'accomplished', 'matter', 'death',
'snows', 'requiem', 'European', 'every', 'classes', 'speak', 'rocks', 'sons',
'u', 'starling', 'cereals', 'officer', 'totally', 'lucky', 'sovereign', 'one',
'liver', 'anything', 'armor', 'round', 'buggered', 'strange', 'next', 'king-a',
'on', 'back', 'Therefore', 'our', "'em", 'oh', 'k-nnnnniggets', 'carried',
'luck', 'Be', 'up', '13', 'plan', 'now', 'twenty', 'all', 'nick', 'burst', '24',
'going', 'there', 'me', 'song', 'castanets', 'earthquakes', 'ride', 'give',
'Hee', 'terribly', "'Here", 'islands', 'from', 'crying', 'chickening',
'actually', 'was', 'a', 'bits', 'want', 'music', 'mystic', 'sheep', 'your',
'completely', 'Too', 'velocity', 'face', 'self-perpetuating', 'telling',
'Shrubber', 'an', 'snap', 'Everything', 'havin', 'words', 'flint',
'Shrubberies', 'suddenly', 'hat', 'Oooohoohohooo', 'we', 'ignore', 'breath',
'Zoot', 'so', 'clllank', 'most', 'Hiyah', 'legally', 'cover', 'SCENE', 'Round',
'she', 'yours', 'removed', 'blondes', 'DENNIS', 'impeccable', 'Hyy', 'pointy',
'Yup', 'says', 'sacrifice', 'MIDGET', 'PARTY', 'learning', 'B', 'outwit',
'fallen', 'wave', 'discovers', 'badger', 'daring', 'bois', 'sex', 'radio',
'temperate', 'near', 'have', 'gave', 'As', 'high-pitched', 'only', 'flesh',
'appease', 'see', 'largest', 'week', 'Yeah', 'Open', 'mac', 'continue', "'re",
'food', 'trough', 'immediately', 'deal', 'CART-MASTER', 'Honestly',
'discovered', 'Mine', 'Bors', 'dramatic', 'Looks', 'Nay', 'Whoa', 'duck',
'sponge', 'terrible', 'least', 'Hello', 'shivering', 'leave', 'thwonk', 'said',
'Autumn', 'adversary', 'grips', 'Remove', 'silly', 'unclog', 'walk', 'Chop',
'Heh', 'saying', 'Holy', 'bastards', 'defeat', 'yet', 'made', 'WOMAN',
'Bedwere', 'fine', "'Ni", 'note', 'bless', 'sir', 'Eternal', 'Use', 'Ay',
'held', 'what', 'thud', 'done', '.', 'night', 'pestilence', 'AMAZING',
'unladen', 'Ask', 'three', 'footwork', 'awaaaaay', 'CRASH', "C'est", 'Prepare',
'sacred', 'fell', 'Huyah', 'refuse', 'let', 'cartoon', "'ll", 'kneeling',
'perilous', 'beacon', 'house', 'moistened', 'thonk', 'traveller', 'ratios',
'fold', 'Charge', 'carved', 'bite', 'Hurry', 'place', 'Winston', 'wounding',
'hee', 'people', 'noise', 'MAYNARD', 'vouchsafed', 'animator', 'gouged',
'whinny', 'Mmm', 'offensive', 'Does', 'nose', 'glass', 'questions', 'Wood',
'unplugged', 'Ages', 'those', 'Far', 'mercy', 'decision', 'bastard', 'Rather',
'Eee', 'feast', 'An', 'empty', 'Greetings', 'executive', 'because', 'chord',
'wrong', 'sank', 'impersonate', 'pram', 'Peng', 'finest', 'na', 'Am', 'vache',
'seemed', 'without', 'Black', 'knows', 'brought', 'valor', 'air-speed', 'later',
```

```
'grail-shaped', 'worst', 'manner', 'open', 'elderberries', 'Bones', 'Must',
'nothing', 'accompanied', 'convinced', 'roar', 'Fetchez', 'dirty', 'mine',
'All', 'attend', 'bloody', 'ponds', 'OF', 'lobbed', 'come', 'warning',
'Maynard', 'minstrels', 'Huh', 'beat', 'whether', 'upon', 'duty', 'nice',
'Guards', 'Would', 'suffice', 'alarm', 'Never', 'rest', 'Splendid', 'Who', 'if',
'strategy', '6', 'Or', 'Run', 'Man', 'squeak', 'tough', 'grail', 'sequin',
'outdated', 'Saxons', 'legendary', 'Look', 'beside', 'enjoying', '(', 'five',
'period', 'PRINCESS', 'gurgle', 'sink', 'haw', 'fatal', 'am', 'Quite', 'dead',
'whose', 'tear', 'wind', 'Not-appearing-in-this-film', 'proved', 'Quick', 'bid',
'ugly', 'animal', 'particular', "'aaggggh", 'sing', 'bangin', 'strand',
'dragging', 'government', 'sawwwww', 'Defeat', 'thine', 'Psalms', 'nine',
'soft', 'try', 'ALL', 'snuff', 'allowed', "'round", 'scratch', 'Then',
'affairs', 'Lancelot', 'majority', 'sire', 'herring', 'far', 'forward', 'room',
'supports', 'nor', 'daft', 'lair', 'reads', 'anarcho-syndicalist', 'lunged',
'excepting', 'NI', 'Silence', 'closest', 'kind', 'need', 'influential',
'regulations', 'RIGHT', 'use', 'well', 'bones', 'after', 'chastity', 'Even',
'hiyaah', 'running', 'W', 'times', 'died', 'length', 'like', 'rewr', 'acting',
'spank', 'giggle', '[', 'magne', 'away', 'Ha', 'over', 'make', 'Stand',
'Britons', 'carve', 'His', 'ni', 'Until', 'here', 'zone', 'Aaaah', 'How',
'servant', 'Here', 'doctors', 'threw', 'demand', 'ounce', 'knees-bent', 'hello',
'taunting', 'rode', 'entrance', 'men', 'ever', 'Bristol', 'electric',
'blessing', 'Aaaaugh', 'sniff', '..', 'Wayy', 'single-handed', 'asks', 'than',
'design', 'give-away', 'did', 'Mind', 'Amen', 'peril', 'Heee', 'courage',
'late', 'sure', 'gained', 'ordinary', 'economic', 'chanting', 'heroic',
'mightiest', 'bridgekeeper', 'sad', 'horn', 'trusty', 'k-niggets', ']',
'African', 'understanding', 'Once', 'fight', 'behold', 'force', 'line',
'Practice', 'For', 'dappy', 'killer', 'Arthur', 'soiled', 'particularly',
'effect', 'hoo', 'forty-three', 'Olfin', 'guarded', 'scimitar', 'legs', 'clad',
'Mother', 'heh', 'girl', 'kings', 'castle', 'lose', 'smashed', 'cruel', 'think',
'routines', 'wipers', 'mandate', 'Lead', 'Over', 'clue', 'derives', 'Shh',
'command', 'does', 'another', '23', 'Please', "'", 'witch', 'handle', 'still',
'beds', 'Order', 'owns', 'recover', 'time-a', 'small', 'started',
'indefatigable', 'forest', 'Midget', 'Stay', 'splash', 'basic', 'performance',
'Tall', 'folk', 'soon', 'killed', 'aquatic', 'Dramatically', 'Excuse', 'drink',
'Steady', 'old', 'logically', 'happy', "'Ooooooh", 'freedom', 'body',
'preserving', 'strewn', 'averting', 'until', 'stayed', 'reasonable', 'yel',
'turns', 'CROWD', 'rich', 'Why', 'no', 'life', 'anchovies', 'mashed', 'main',
'wants', 'commune', 'enough', 'O', 'w', 'vote', 'So', 'strongest', 'unsingable',
'GUESTS', 'hacked', 'chosen', 'g', 'vicious', 'nightfall', 'Lake', 'riding',
'Erm', 'King', "'ve", 'system', 'It', 'Aramaic', 'are', 'c', 'hang',
'personally', 'non-migratory', 'middle', 'anyone', 'bed-wetting', 'basis',
'got', 'north', 'were', '9', 'while', 'throughout', 'when', 'depart', 'grovel',
'tonight', 'into', 'lads', 'doubt', 'types', "'d", 'rock', 'Badon', 'burn',
'Peril', '12', 'attack', 'Are', 'able', 'which', 'leap', 'uuup', 'wan',
'afraid', 'coconuts', 'hospitality', 'keepers', '2', 'fly', 'gallantly', 'Allo',
'way', 'LUCKY', 'Chaste', 'etc', 'Silly', 'fought', 'many', 'ai', 'wet',
'distress', 'climes', 'court', 'necessary', 'bride', 'DINGO', 'Father',
'brunettes', 'answers', 'May', 'Aggh', 'Burn', 'varletesses', 'bridges', ',',
```

```
'full', 'Ewing', 'rodent', 'orangutans', 'having', 'Armaments', 'wonderful',
'found', 'But', 'Packing', 'disheartened', 'triumphs', 'woman', 'meant', 'set',
'shalt', 'Castle', 'swords', 'Court', "d'you", 'Thursday', 'Do', 'Excalibur',
'thanks', 'voluntarily', 'Bon', 'Ector', 'biters', 'Good', 'pause', 'In',
'remember', 'To', 'b', 'blanket', 'clap', 'lad', 'say', 'collective', 'Uhh',
'capital', 'out', 'summon', 'approaching', 'listen', 'worthy', 'his', 'sell',
'HEADS', 'howl', 'repressing', 'kills', 'Aaauggh', 'case', 'Perhaps', "'ni",
'higher', 'dare', 'watch', 'Found', 'counting', 'heart', 'Knight', 'flight',
'wedlock', 'mortally', 'wound', "'sorry", 'God', 'wood', 'Britain', 'resumes',
'fruit', 'er', 'commands', 'laughing', 'getting', 'spanking', 'between', 'boom',
'makes', 'Dis-mount', 'compared', 'scene', 'LEFT', 'Every', 'Loimbard',
'enchanter', 'sloths', 'Robinson', 'knocked', 'SUN', 'sense', 'exciting',
'dangerous', 'argue', 'gentle', 'naughty', 'shelter', 'Actually', 'wooden',
'such', 'tracts', 'marrying', 'bonk', 'mean', 'lying', 'either', 'mooooooo',
'Hooray', 'grip', 'kill', 'arrange', 'The', 'decided', 'examine', 'Hill',
'Divine', 'and', 'fellows', 'first', 'not', 'scribble', 'ho', 'witness',
'sometimes', 'Build', 'Blue', 'hmm', 'doing', 'Ohh', 'dunno', 'Where', 'aside',
'thirty-seven', '11', 'chops', 'rescue', 'Twenty-one', 'know', 'die',
'everything', 'Follow', 'Enchanter', 'BRIDGEKEEPER', 'Hm', 'armed', 'dad', 'U',
'slash', 'English', 'taunt', 'minutes', 'Hand', 'setting', 'Five', 'Ni',
'witches', 'ferocity', 'dynamite', 'bird', 'almost', 'language', 'dressed',
'Herbert', '20', 'Right', 'bicker', 'these', 'plover', 'tap-dancing', 'Crapper',
'Make', 'properly', "'S", 'GUARD', 'By', 'bond', 'minute', 'Ives', 'filth',
'Woa', 'Chapter', 'miserable', 'water', 'finds', 'entering', 'Message',
'conclusions', 'Torment', 'Very', 'Auuuuuuuugh', 'dancing', 'Thppppt', 'return',
'pimples', 'covered', 'looks', 'home', 'Yay', 'work', 'Un', 'husk', 'We',
'will', 'Yeaaah', 'identical', "'Til", 'pray', 'saw', 'against', 'seek',
'uuggggggh', 'OFFICER', 'fire', 'rather', 'Dingo', 'trade', 'land', 'welcome',
'England', 'breadth', 'valleys', 'dark', 'charged', 'settles', 'ham', 'fooling',
'knight', 'PRINCE', 'false', 'cut', 'Uther', 'worse', 'Get', 'wart', 'DIRECTOR',
'enter', 'required', 'so-called', 'headoff', 'illegitimate-faced', 'can',
'boys', 'favor', 'vary', 'behind', 'inside', 'thought', 'mate', 'France',
'grin', 'HISTORIAN', 'nobody', 'Together', 'guard', 'Anthrax', 'I', 'ones',
'my', 'tiny', 'named', 'KING', 'Could', 'scholar', 'biscuits', 'GUARDS',
'differences', 'entered', 'miss', 'great', 'awaaay', 'once', 'very', 'STUNNER',
'off', 'private', 'Guy', 'less', "'cause", 'Say', 'bring', 'bosom', 'being',
'banana-shaped', 'spirit', 'but', 'ere', 'hell', 'sample', 'throwing', '?',
'formidable', 'rrrr', 'feet', '…', 'ran', 'pissing', 'creature', 'Walk',
'domine', 'FATHER', 'Supreme', 'cadeau', 'women', 'given', 'passing', 'Quoi',
'each', 'And', 'valiant', 'stress', 'run', 'watery', 'Y', 'good', 'Aaagh',
'unarmed', 'crossed', 'retold', 'stone', 'signifying', 'de', 'tops', ';',
'lonely', 'trumpets', 'BORS', 'occasion', 'move', 'throat', 'find', 'crash',
'looney', 'whop', 'CONCORDE', 'general', 'kingdom', 'problems', 'cross',
'about', 'pack', 'accent', 'jump', 'scots', 'martin', 'remain', 'bleed',
'agree', 'suppose', 'Back', 'boil', 'real', 'hopeless', 'Gorge', 'aaaaaah',
'heard', 'stood', 'ready', 'baaaa', 'scared', 'ways', 'Churches', 'may',
'dungeon', 'returns', 'ooh', 'Hoo', 'committed', 'called', 'Hiyaah', 'Help',
'tart', 'Dennis', 'Not', 'individually', 'big', 'Dragon', 'Action', 'Augh',
```

```
'color', 'Wait', 'tree', "'it", 'huge', 'Exactly', 'invincible', 'Umhm',
'raised', 'Schools', 'bit', 'stops', 'Anyway', 'floats', 'PIGLET', 'sister',
'idiom', "'anging", 'surprise', 'donkey-bottom', 'mother', 'oo', "e'er",
'guest', 'One'}
```

▶ **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

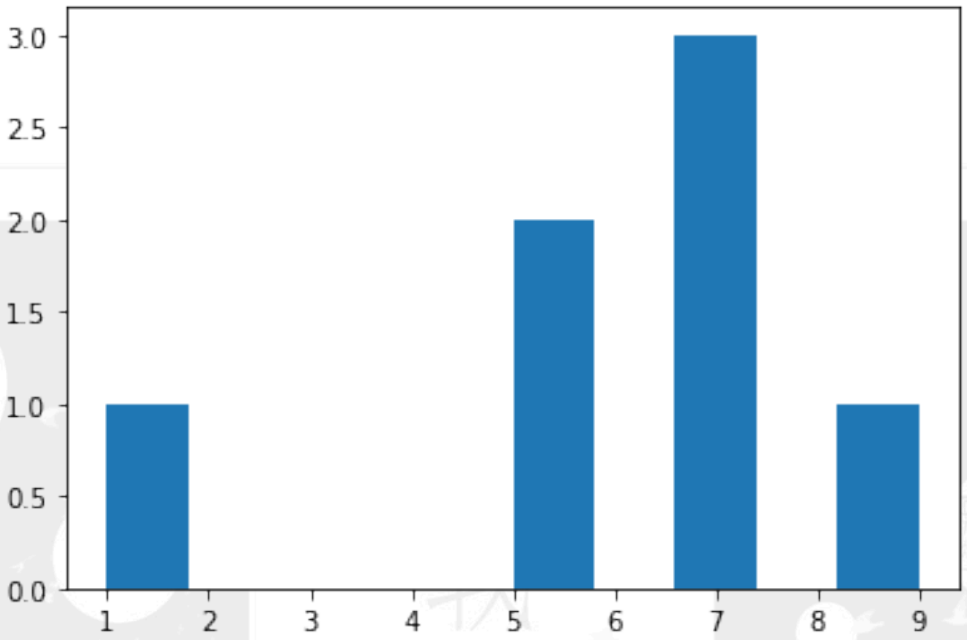
4.1 Why is it in need to get started with matplotlib?

- 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!

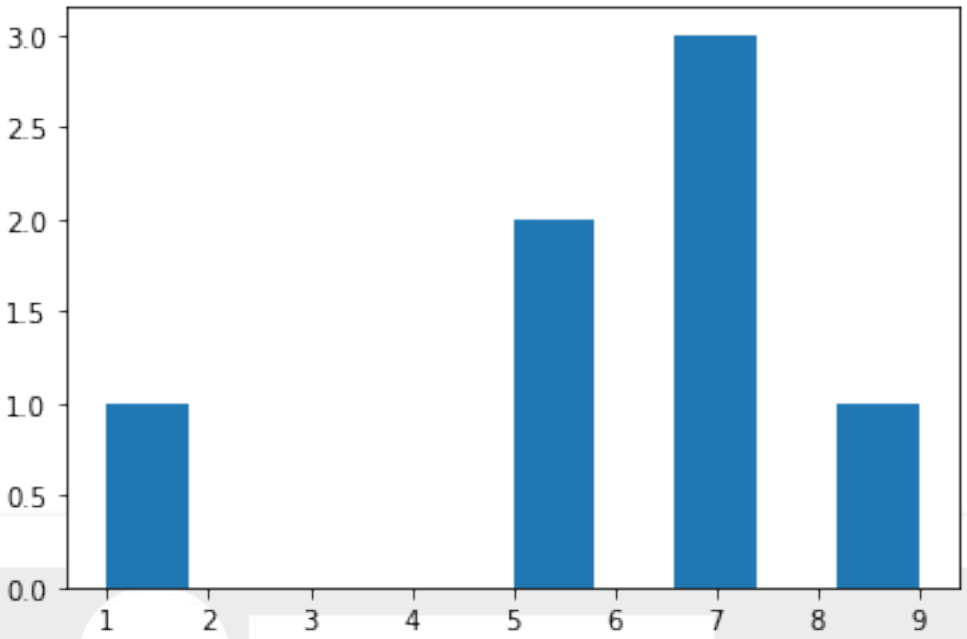
4.2 Code of plotting a histogram with matplotlib:

[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>)
```



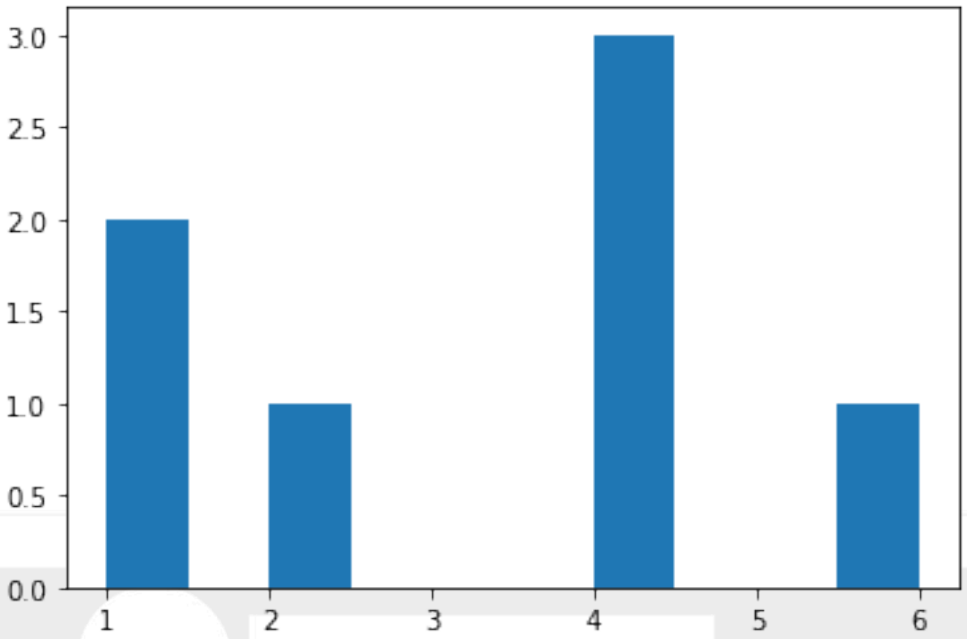
[41]:
```
plt.hist([1, 5, 5, 7, 7, 7, 9])
plt.show()
```



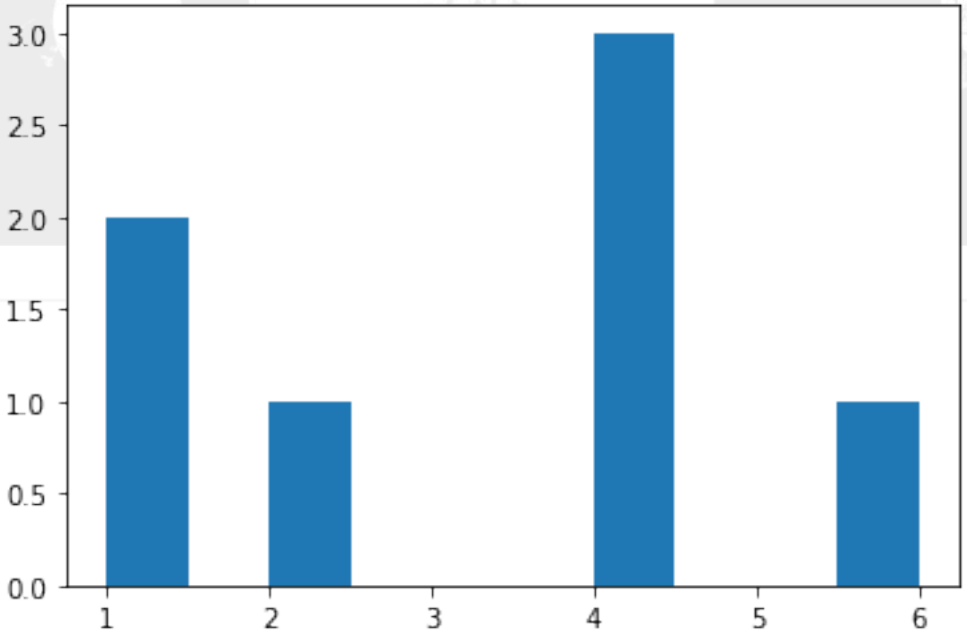
4.3 Code of combining NLP data extraction with plotting:

[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:**

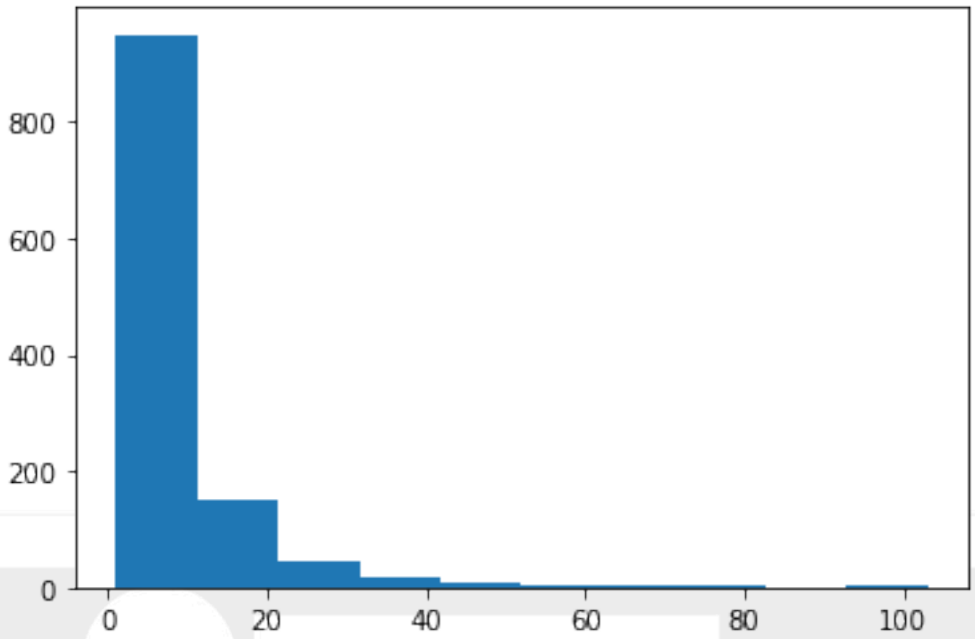
[44]:
```
import re
from matplotlib import pyplot as plt
from nltk.tokenize import regexp_tokenize
```

▶ **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()
```


800
600
400
200
0
0
20
40
60
80
100