NLP - Python Regular Expressions and More

based on K. Jarmul's *Natural Language Processing Fundamentals*.

Regular expressions are strings with specific syntax, which facilitate pattern matching.

Applications: find web links in documents, remove special characters, etc.

CONTENTS

---

1. INITIALIZE ENVIRONMENT

In [1]:

```
import re # python module for regular expressions
from nltk.tokenize import word_tokenize, regexp_tokenize # NLTK word tokenizer
from matplotlib import pyplot as plt # python charts
```

Let's take a look at some basics.

In [2]:

```
re.match('super','supercalifragilisticexpialidocious') # match pattern, from the b
```

Out[2]:

```
<_sre.SRE_Match object; span=(0, 5), match='super'>
```

In [3]:

```
re.match('super','Supercalifragilisticexpialidocious') # what happens here?
```

In [4]:

```
w_regex = '\w+' # regular expression pattern for "word"
re.match(w_regex,'Hello World!') # matches the first word in the string
```

Out[4]:

```
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
```

---

2. COMMON REGULAR EXPRESSION PATTERNS

- `\w+`: word
- `\d`: digit
- `\s`: space
- `.`: wildcard
- + or `*`: greedy match
- `\W`: not word
- `\D`: not digit
- `\S`: not space
- `[a-z]`: lower case group
- `[A-Z]`: upper case group

In Python, regular expression patterns must be prefixed with an "r" to differentiate between the raw string and the string's interpretation.

---

3. `re` FUNCTIONS

- `split`: split a string on a regular expression
- `findall`: find all patterns in a string
- `search`: search for a pattern
- `match`: match an entire string based on a pattern

Pattern **first**, string **second**.

In [5]:

```
re.split('\s+','Can you do the split?') # splits on the spaces and removes them
```

Out[5]:

```
['Can', 'you', 'do', 'the', 'split?']
```

In [6]:

```
re.split('s+','Can you do the split?') # splits on the "s" and removes it
```

Out[6]:

```
['Can you do the ', 'plit?']
```

In [7]:

```
re.split('\s','Can you do the split?') # splits on the spaces and removes them
```

Out[7]:

```
['Can', 'you', 'do', 'the', 'split?']
```

In [8]:

```
re.split('\w+','Can you do the split?') # splits on the words and removes them
```

Out[8]:

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

In [9]:

```
re.split('\W+','Can you do the split?') # splits on the non-words and removes then
```

Out[9]:

```
['Can', 'you', 'do', 'the', 'split', '']
```

We can also study seriously a silly sentence saved as a string.

In [10]:

```
test_string = 'Oh they built the built the ship Titanic. It was a mistake. It cost
test_string
```

Out[10]:

```
'Oh they built the built the ship Titanic. It was a mistake. It cost m
ore than 1.5 million dollars. Never again!'
```

```
In [11]:
```

```python
sent_ends = r"[.?!]" # these are the characters that could end a sentence in Engli
print(re.split(sent_ends,test_string)) # split the string into sentences
print(len(re.split(sent_ends,test_string))) # how many such sentences are there?
```

```
['Oh they built the built the ship Titanic', ' It was a mistake', ' It
cost more than 1', '5 million dollars', ' Never again', '']
6
```

```
In [12]:
```

```python
cap_words = r"[A-Z]\w+" # Upper case characters
print(re.findall(cap_words,test_string)) # find all the words with an uppercase in
print(len(re.findall(cap_words,test_string))) # how many such words are there?
```

```
['Oh', 'Titanic', 'It', 'It', 'Never']
5
```

```
In [13]:
```

```python
spaces = r"\s+" # spaces
print(re.split(spaces,test_string)) # split on spaces
print(len(re.split(spaces,test_string))) # how many tokens does that yiels?
```

```
['Oh', 'they', 'built', 'the', 'built', 'the', 'ship', 'Titanic.', 'It
', 'was', 'a', 'mistake.', 'It', 'cost', 'more', 'than', '1.5', 'milli
on', 'dollars.', 'Never', 'again!']
21
```

```
In [14]:
```

```python
numbers = r"\d+" # numbers
print(re.findall(numbers,test_string)) # find all the numeric characters
print(len(re.findall(numbers,test_string))) # how many such numerics are there?
```

```
['1', '5']
2
```

The main difference between `search` vs `match` is that `match` tries to match from the beginning while `search` doesn't.

4. REGULAR EXPRESSIONS GROUPS ( ) and RANGES [ ] with OR |

- `"[a-zA-Z]+"`: lower and upper case English/French (unaccented) alphabet
- `"[0-9]"`: numbers from 0 to 9 (as digits)
- `"[a-zA-Z'\.\-]+"`: lower and upper case English/French (unaccented) alphabet, `'`, `.` and `-`
- `"(a-z)"`: the characters `a`, `-`, and `z`
- `"(\s+|,)"`: spaces or commas
- `"(\d+|\w+)"`: words or numerics

In [15]:

```
text = 'On the 1st day of xmas, my boat sank.'
numbers_or_words = r"(\d+|\w+)"
spaces_or_commas = r"(\s+|,)"

print(re.findall(numbers_or_words,text))
print(re.findall(spaces_or_commas,text))

# What do you expect to see here?
```

```
['On', 'the', '1', 'st', 'day', 'of', 'xmas', 'my', 'boat', 'sank']
[' ', ' ', ' ', ' ', ' ', ',', ' ', ' ', ' ']
```

In [16]:

```
text = "will something happen after the semi-colon; I don't think so"
print(re.match(r"[a-z -]+",text)) # once it hits the semi-colon, it can't match an
print(re.match(r"[a-z ]+",text)) # once it hits the dash, it can't match any more
print(re.match(r"[a-z]+",text)) # once it hits space, it can't match any more
print(re.match(r"(a-z-)+",text)) # what's happening here?
```

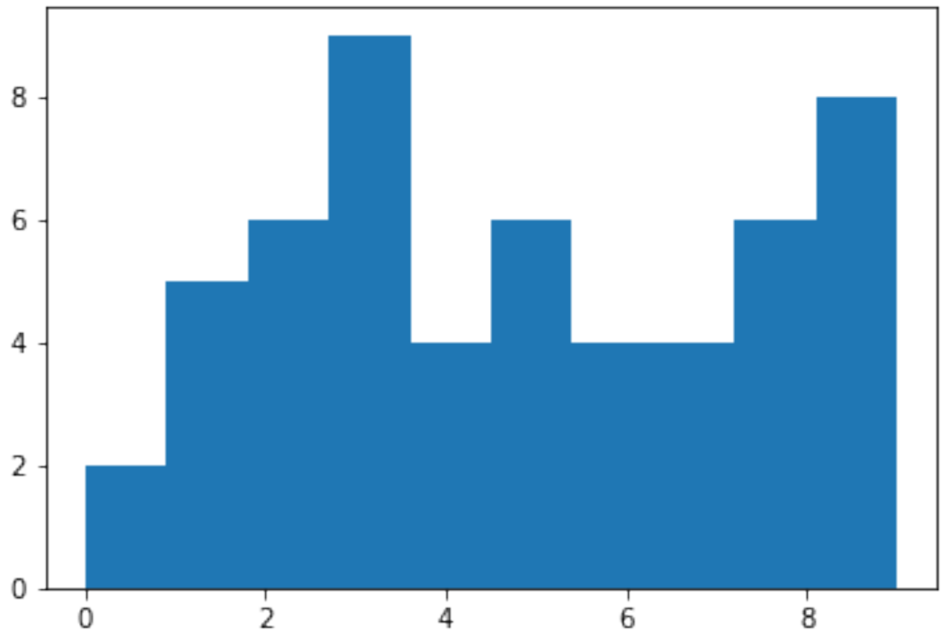
```
<_sre.SRE_Match object; span=(0, 42), match='will something happen aft
er the semi-colon'>
<_sre.SRE_Match object; span=(0, 36), match='will something happen aft
er the semi'>
<_sre.SRE_Match object; span=(0, 4), match='will'>
None
```

5. PYTHON CHARTS

- requires `matplotlib`
- can create histograms, bar charts, scatter plots, etc.

In [17]:

```
# let's create a histogram out of the digits of pi
digits_of_pi=[3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2,3,8,4,6,2,6,4,3,3,8,3,2,7,9,5,0,2,
plt.hist(digits_of_pi)
plt.show()
```



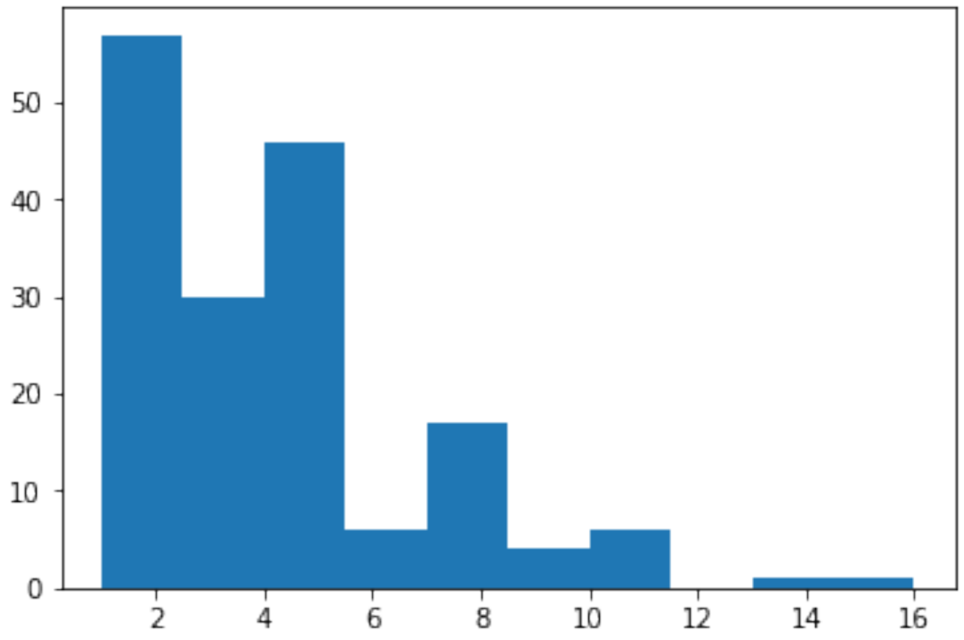
Let's revisit the text string from the NLTK Intro notebook. We'll plot the word length distribution for that text.

In [18]:

```
text = re.sub(r'\n', ' ', '''
"Bravo, Jos!" said Mr. Sedley; on hearing the bantering of which well-known voice,
Jos   instantly relapsed into an alarmed silence, and quickly took his departure.
He did not lie awake all night thinking whether or not he was in love with Miss Sh
the passion of love never interfered with the appetite or the slumber of Mr. Josep
Sedley; but he thought to himself how delightful it would be to hear such songs as
those after Cutcherry—what a distinguee girl she was—how she could speak French be
than the Governor-General's lady herself—and what a sensation she would make at th
Calcutta balls. "It's evident the poor devil's in love with me," thought he. "She
just as rich as most of the girls who come out to India. I might go farther, and f
worse, egad!" And in these meditations he fell asleep.
''')
```

In [19]:

```
words = word_tokenize(text) # tokenizes the text into words
word_lengths = [len(w) for w in words] # goes through all the words and compute th
plt.hist(word_lengths)
plt.show() # is the appearance of the plot surprising?
```



6. WORD COUNTS AND PRE-PROCESSING

Frequent words in a text are perhaps more significant. Bag of words functionality in Python is provided by the module `collections`.

Start by initializing the environment, import *Meet the Elements* by **They Might Be Giants**, and pre-process the text.

In [20]:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from collections import Counter

stopwords = nltk.corpus.stopwords.words('english')
wordnet_lemmatizer = WordNetLemmatizer()
```

In [21]:

```
meet_the_elements_TMBG = """Iron is a metal, you see it every day
Oxygen, eventually, will make it rust away
Carbon in its ordinary form is coal
Crush it together, and diamonds are born
Come on come on and meet the elements
May I introduce you to our friends, the elements?
Like a box of paints that are mixed to make every shade
They either combine to make a chemical compound or stand alone as they are
Neon's a gas that lights up the sign for a pizza place
The coins that you pay with are copper, nickel, and zinc
Silicon and oxygen make concrete bricks and glass
Now add some gold and silver for some pizza place class
Come on come on and meet the elements
I think you should check out the ones they call the elements
Like a box of paints that are mixed to make every shade
They either combine to make a chemical compound or stand alone as they are
Team up with other elements making compounds when they combine
Or make up a simple element formed out of atoms of the one kind
Balloons are full of helium, and so is every star
Stars are mostly hydrogen, which may someday fill your car
Hey, who let in all these elephants?
Did you know that elephants are made of elements?
Elephants are mostly made of four elements
And every living thing is mostly made of four elements
Plants, bugs, birds, fish, bacteria and men
Are mostly carbon, hydrogen, nitrogen and oxygen
Come on come on and meet the elements
You and I are complicated, but we're made of elements
Like a box of paints that are mixed to make every shade
They either combine to make a chemical compound or stand alone as they are
Team up with other elements making compounds when they combine
Or make up a simple element formed out of atoms of the one kind
Come on come on and meet the elements
Check out the ones they call the elements
Like a box of paints that are mixed to make every shade
They either combine to make a chemical compound or stand alone as they are""" # ""
```

In [22]:

```
meet_the_elements = word_tokenize(meet_the_elements_TMBG) # tokenizes the text alo
meet_the_elements = [t.lower() for t in meet_the_elements if t.isalpha() if t not 
     # converts into lowercase
     # retains alpha characters
     # removes english stopwords
meet_the_elements = [wordnet_lemmatizer.lemmatize(t) for t in meet_the_elements] #
counters=Counter(meet_the_elements) # provides the word count
counters.most_common(10) # shows 10 most common terms
```

Out[22]:

```
[('element', 15),
 ('make', 12),
 ('come', 8),
 ('every', 7),
 ('compound', 6),
 ('combine', 6),
 ('meet', 4),
 ('alone', 4),
 ('one', 4),
 ('made', 4)]
```