Data Preprocessing

Mayank Singh

ACM-IKDD Summer School on Data Science, IIT Gandhinagar, 7th July 2022

NLP Libraries

- NLTK Toolkit (Python)
  https://www.nltk.org
- Spacy (Python)
  https://spacy.io
- Polyglot (Python)
  https://polyglot.readthedocs.io/en/latest/
- Stanford CoreNLP (Java)
  https://stanfordnlp.github.io/CoreNLP/
- Unix Commands

NLP Libraries

- NLTK Toolkit (Python)
  https://www.nltk.org
- Spacy (Python)
  https://spacy.io
- Polyglot (Python)
  https://polyglot.readthedocs.io/en/latest/
- Stanford CoreNLP (Java)
  https://stanfordnlp.github.io/CoreNLP/
- Unix Commands

- Huggingface (Python)
  http://huggingface.co/

We will see all these libraries in action now!

Tokenization

- **Sentence Tokenizer (Sequence of characters -> sentences)**

Input: `It was the best of times, it was the worst of times. It was the age of wisdom, it was the age of foolishness.`

Expected Output:

```
It was the best of times, it was the worst of times.
It was the age of wisdom, it was the age of foolishness.
```

Tokenization

- **Sentence Tokenizer (Sequence of characters -> sentences)**

Input: `It was the best of times, it was the worst of times. It was the age of wisdom, it was the age of foolishness.`

Expected Output:

```
It was the best of times, it was the worst of times.
It was the age of wisdom, it was the age of foolishness.
```

- **Word Tokenizer (Sequence of characters -> words)**

Input: `It was the best of times, it was the worst of times.`

Expected Output: `It, was, the, best, of, times, it, was, the, worst, of, times`

Tokenization

- **Sentence Tokenizer (Sequence of characters -> sentences)**

Input: It was the best of times, it was the worst of times. It was the age of wisdom, it was the age of foolishness.

Expected Output:

It was the best of times, it was the worst of times.
It was the age of wisdom, it was the age of foolishness.

- **Word Tokenizer (Sequence of characters -> words)**

Input: It was the best of times, it was the worst of times.
Expected Output: It, was, the, best, of, times, it, was, the, worst, of, times

- **Subword tokenizer (Sequence of characters -> subwords)**

Input: We would like to embed this extremely short text with an unknown word zozofah!

Expected Output: We, would, like, to, em, ##bed, this, extremely, short, text, with, an, unknown, word, z, ##oz, ##of, ##ah

A Simple Word Tokenization Using Unix commands

Given a text file, output the word tokens and their frequencies

```
tr -sc ’A-Za-z’ ’\n’ < file_name
| sort
| uniq -c
| sort -rn
```

A Simple Word Tokenization Using Unix commands

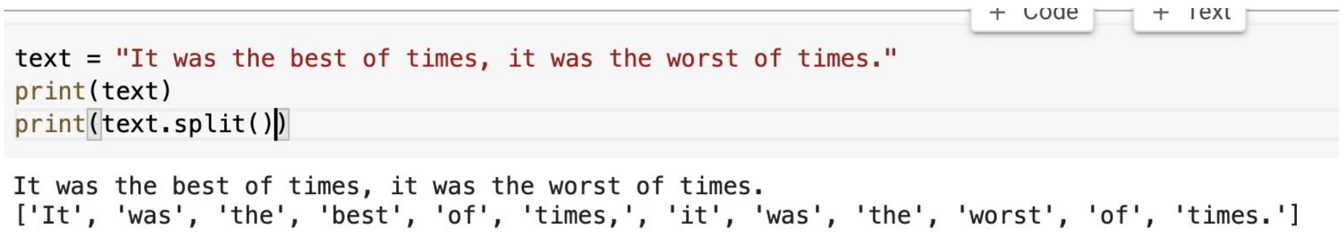
Given a text file, output the word tokens and their frequencies

```
tr -sc ’A-Za-z’ ’\n’ < file_name
| sort
| uniq -c
| sort -rn
```

Explore commands like “sed”, “grep”, etc.

A Simple Word Tokenization Using Python Split Function

Given a string of characters, output the word tokens



```python
text = "It was the best of times, it was the worst of times."
print(text)
print(text.split())
```

```
It was the best of times, it was the worst of times.
['It', 'was', 'the', 'best', 'of', 'times,', 'it', 'was', 'the', 'worst', 'of', 'times.']
```

Explore different delimiters like “?”, “,”, etc.

Challenges With Simple Tokenizers

Common examples

- Finland’s → Finland Finlands Finland’s ?
- What’re, I’m, shouldn’t → What are, I am, should not ?
- San Francisco → one token or two?
- m.p.h. → ??
- State-of-the-art → four tokens or just one?
- Multi-disciplinary → Two tokens or just one?

Challenges With Simple Tokenizers

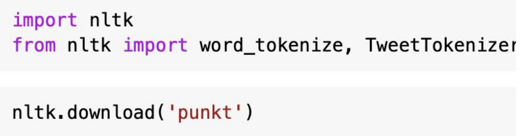
Common examples

- Finland's → Finland Finlands Finland's ?
- What're, I'm, shouldn't → What are, I am, should not ?
- San Francisco → one token or two?
- m.p.h. → ??
- State-of-the-art → four tokens or just one?
- Multi-disciplinary → Two tokens or just one?

Language-specific Issues

- German: Lebensversicherungsgesellschaftsangestellter
  'life insurance company employee'
- Chinese: 莎拉波娃现在居住在美国东南部的佛罗里达。
  莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

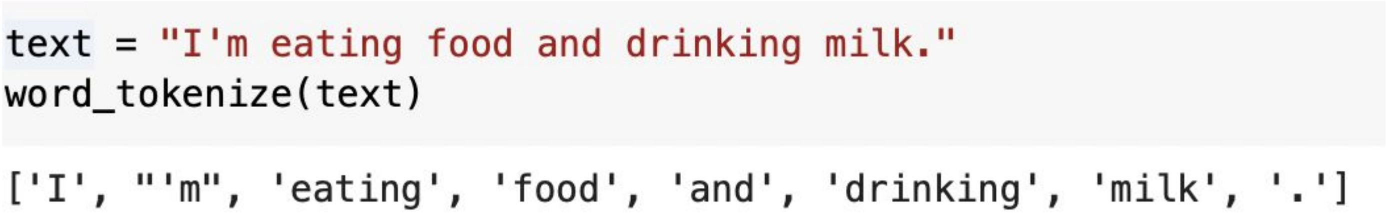
We Need Intelligent Tokenizers



```
import nltk
from nltk import word_tokenize, TweetTokenizer
```

```
nltk.download('punkt')
```

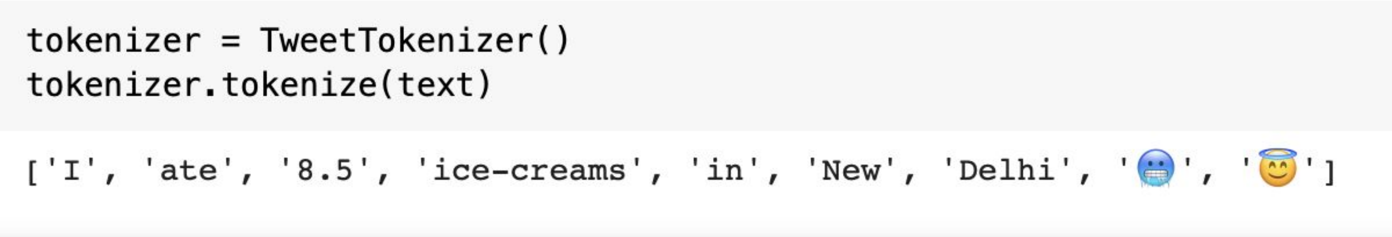
NLTK’s Basic Tokenizer



```
text = "I'm eating food and drinking milk."
word_tokenize(text)
```

```
['I', "'m", 'eating', 'food', 'and', 'drinking', 'milk', '.']
```

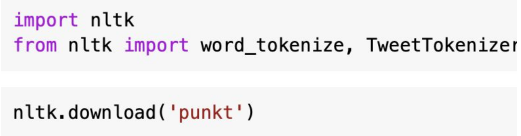
NLTK’s Tweet Tokenizer



```
tokenizer = TweetTokenizer()
tokenizer.tokenize(text)
```

```
['I', 'ate', '8.5', 'ice-creams', 'in', 'New', 'Delhi', '🥶', '😇']
```

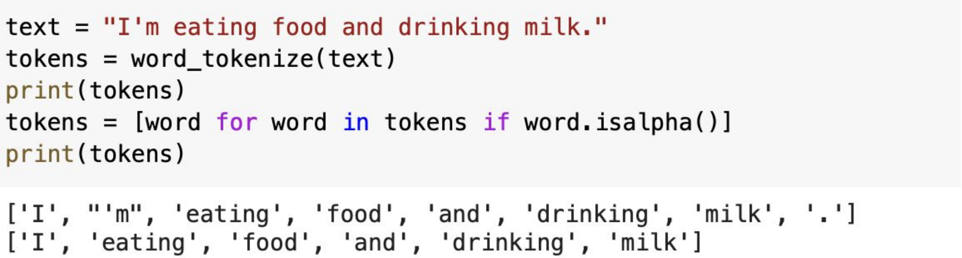
Normalization



```
import nltk
from nltk import word_tokenize, TweetTokenizer
```

```
nltk.download('punkt')
```

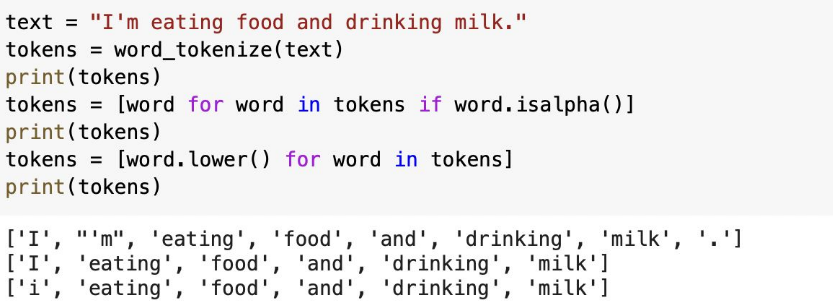
Python’s Punctuation Removal Module



```
text = "I'm eating food and drinking milk."
tokens = word_tokenize(text)
print(tokens)
tokens = [word for word in tokens if word.isalpha()]
print(tokens)
```

```
['I', "'m", 'eating', 'food', 'and', 'drinking', 'milk', '.']
['I', 'eating', 'food', 'and', 'drinking', 'milk']
```

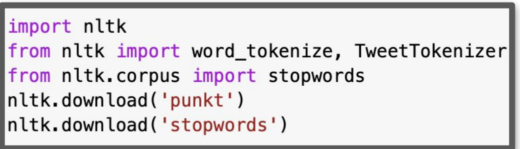
Python’s Lowercasing Module



```
text = "I'm eating food and drinking milk."
tokens = word_tokenize(text)
print(tokens)
tokens = [word for word in tokens if word.isalpha()]
print(tokens)
tokens = [word.lower() for word in tokens]
print(tokens)
```

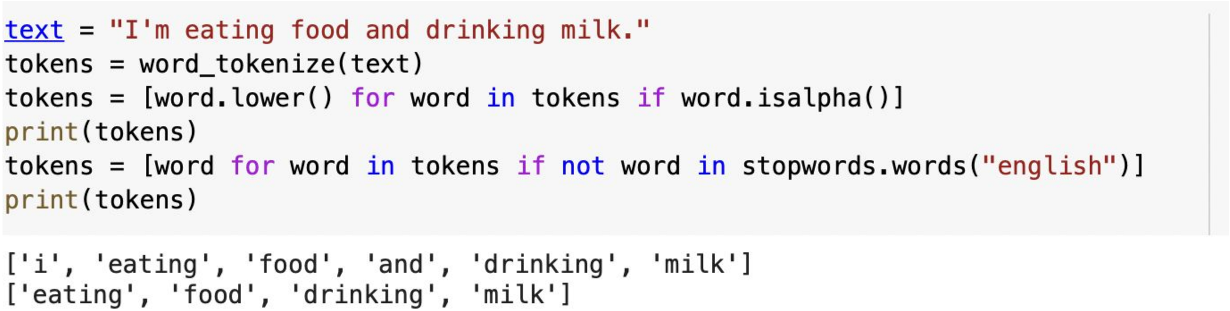
```
['I', "'m", 'eating', 'food', 'and', 'drinking', 'milk', '.']
['I', 'eating', 'food', 'and', 'drinking', 'milk']
['i', 'eating', 'food', 'and', 'drinking', 'milk']
```

Normalization



```
import nltk
from nltk import word_tokenize, TweetTokenizer
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('stopwords')
```

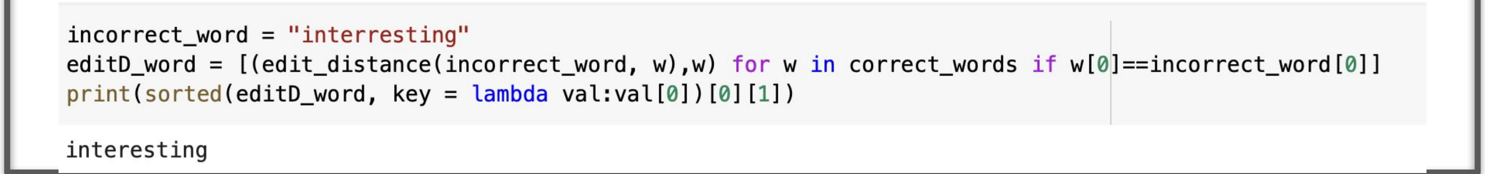
NLTK’s Stopword Removal Module



```
text = "I'm eating food and drinking milk."
tokens = word_tokenize(text)
tokens = [word.lower() for word in tokens if word.isalpha()]
print(tokens)
tokens = [word for word in tokens if not word in stopwords.words("english")]
print(tokens)
```

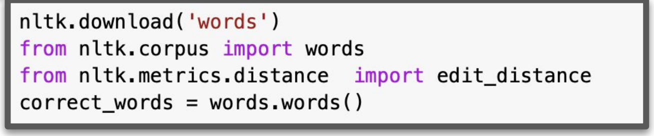
```
['i', 'eating', 'food', 'and', 'drinking', 'milk']
['eating', 'food', 'drinking', 'milk']
```

NLTK’s Spelling Correction Module



```
incorrect_word = "interresting"
editD_word = [(edit_distance(incorrect_word, w),w) for w in correct_words if w[0]==incorrect_word[0]]
print(sorted(editD_word, key = lambda val:val[0])[0][1])
```

```
interesting
```



```
nltk.download('words')
from nltk.corpus import words
from nltk.metrics.distance  import edit_distance
correct_words = words.words()
```

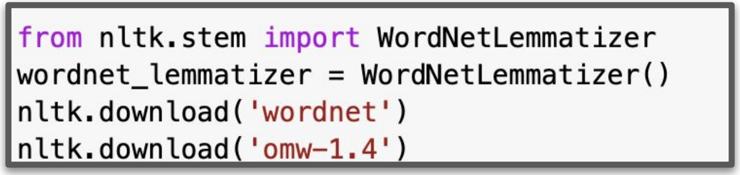
Lemmatization

- Reduce inflections or variant forms to base form:

  am, are, is → be
  car, cars, car’s, cars’ → car

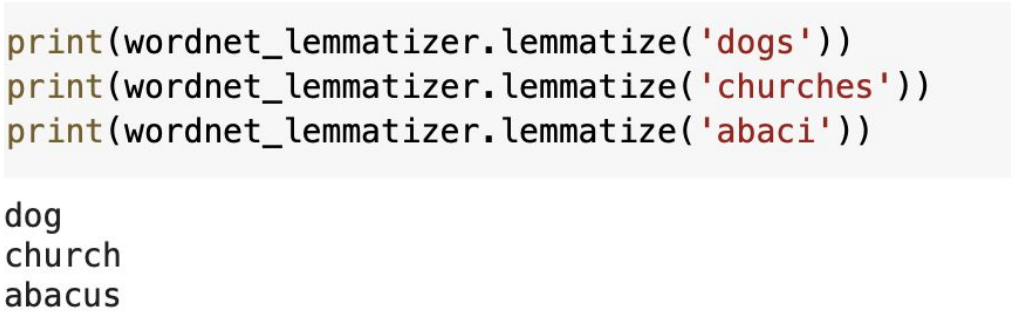
- Have to find the correct dictionary headword form

Lemmatization in Action



```
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
nltk.download('wordnet')
nltk.download('omw-1.4')
```

NLTK’s Lemmatization Module



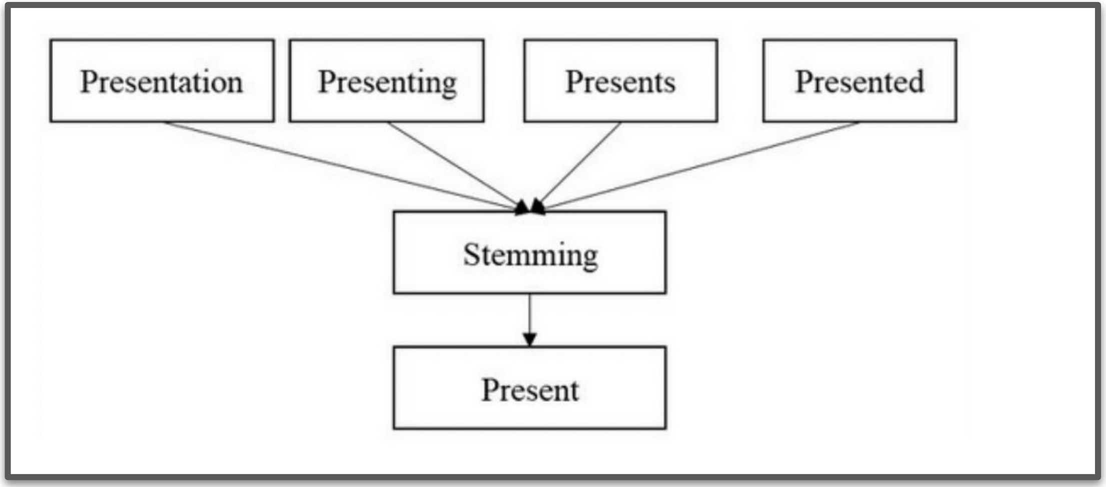
```
print(wordnet_lemmatizer.lemmatize('dogs'))
print(wordnet_lemmatizer.lemmatize('churches'))
print(wordnet_lemmatizer.lemmatize('abaci'))
```

```
dog
church
abacus
```

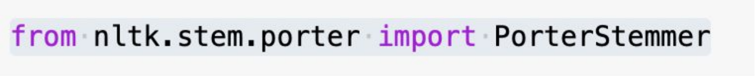
Stemming

- **Reducing terms to their stems**

- **Crude chopping of affixes**

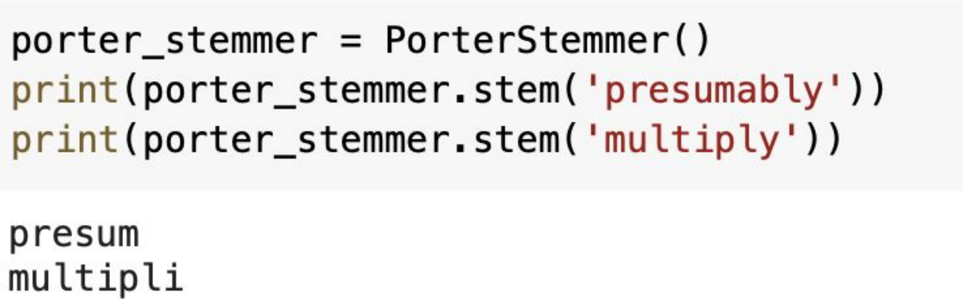


Stemming in Action



```python
from nltk.stem.porter import PorterStemmer
```

NLTK’s Stemming Module



```python
porter_stemmer = PorterStemmer()
print(porter_stemmer.stem('presumably'))
print(porter_stemmer.stem('multiply'))
```

```
presum
multipli
```

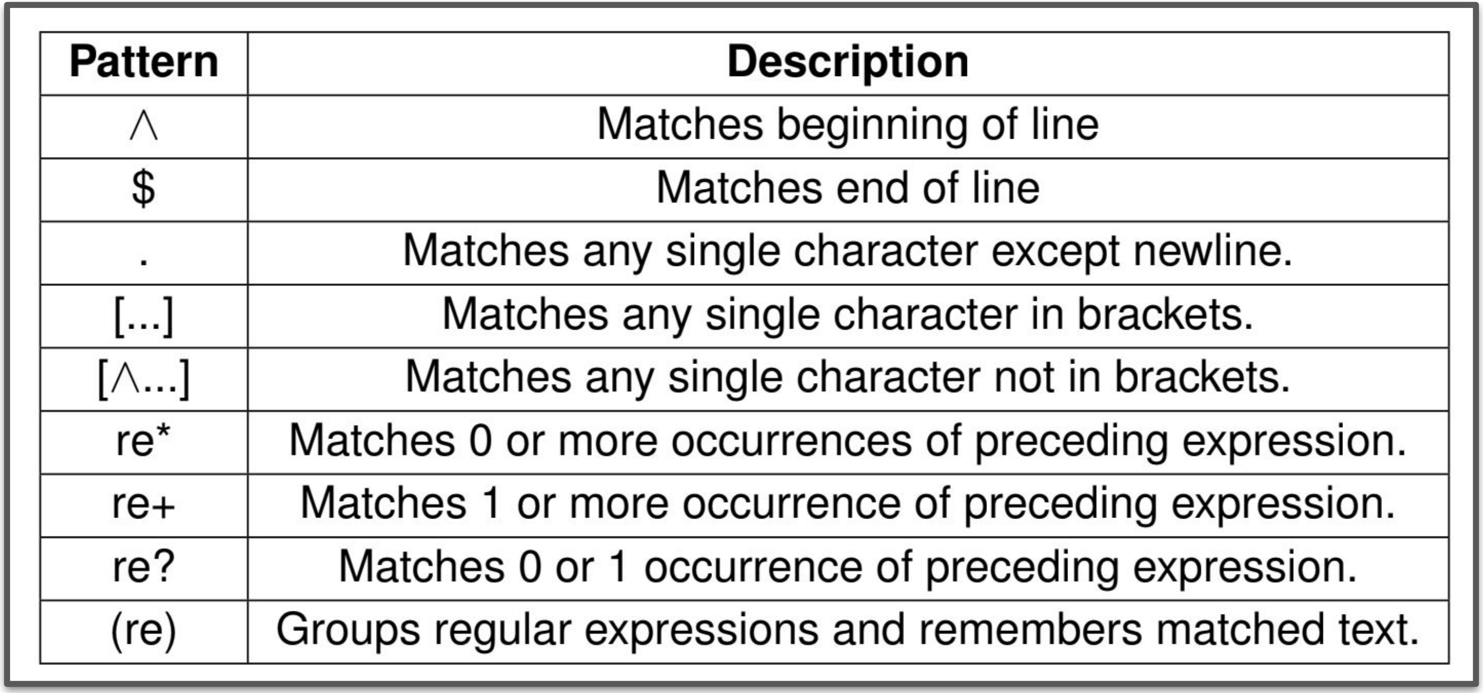
Regular Expressions

A regular expression is a special sequence of characters that helps you **match or find** other strings or sets of strings.

- Regular expressions are widely used in UNIX world.
- How to use them? Depends on different implementations.
- In python, re module provides full support for regular expressions.

Regular Expression Patterns

Except for control characters, (+ ? . * ∧ $ ( ) [ ] |), all characters match themselves.

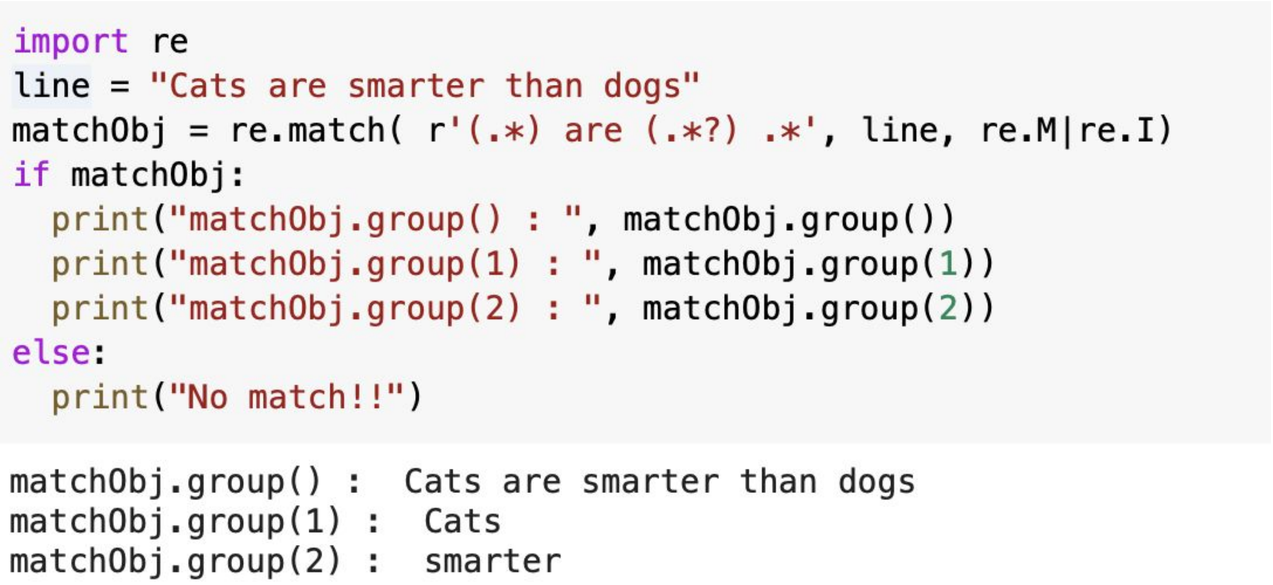


Pattern	Description
∧	Matches beginning of line
$	Matches end of line
.	Matches any single character except newline.
[...]	Matches any single character in brackets.
[∧...]	Matches any single character not in brackets.
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.
(re)	Groups regular expressions and remembers matched text.

Source: https://www.tutorialspoint.com/python/python_reg_expressions.htm

Regular Expressions in Action

Python’s RE Module

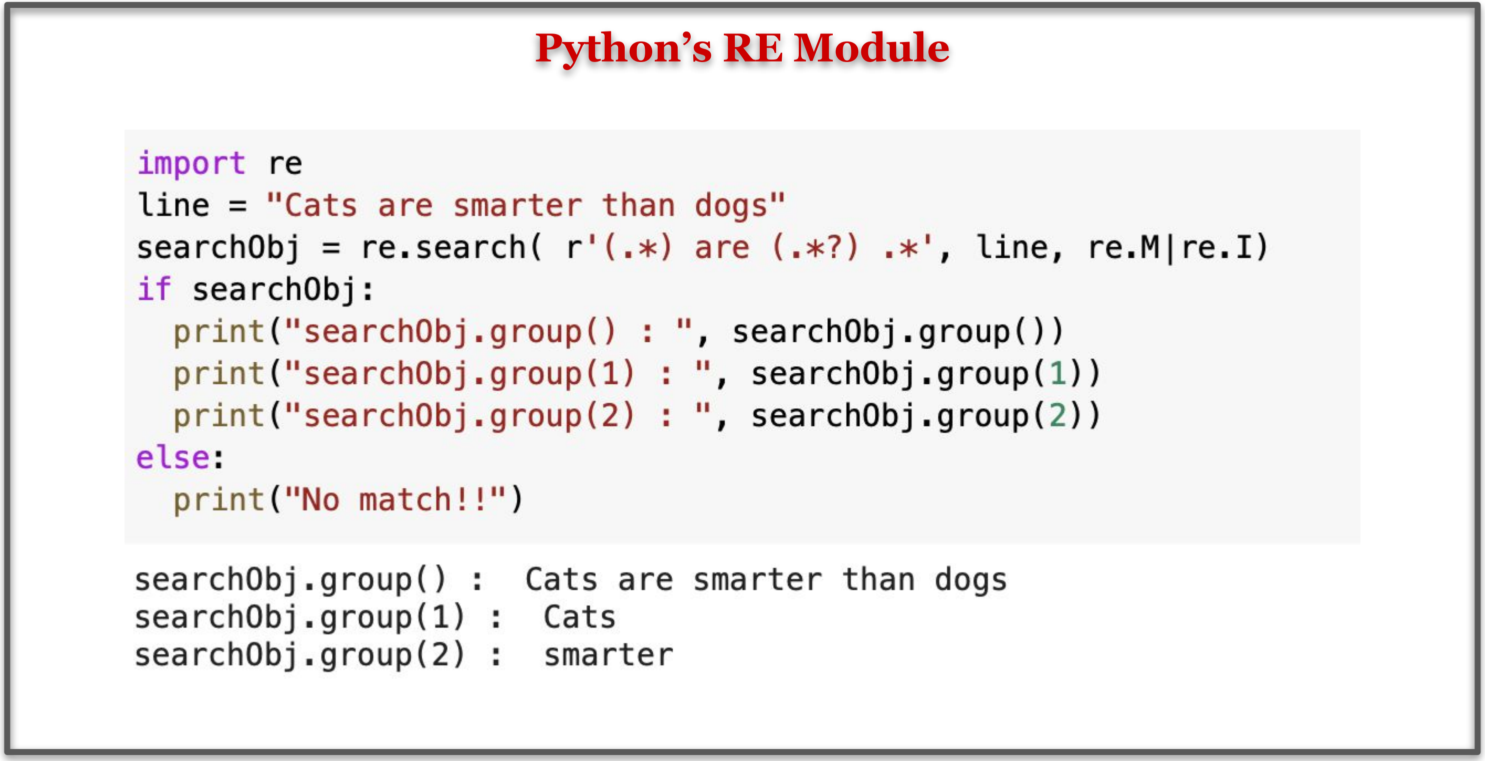


```
import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
  print("matchObj.group() : ", matchObj.group())
  print("matchObj.group(1) : ", matchObj.group(1))
  print("matchObj.group(2) : ", matchObj.group(2))
else:
  print("No match!!")
```

```
matchObj.group() :  Cats are smarter than dogs
matchObj.group(1) :  Cats
matchObj.group(2) :  smarter
```

Match checks for a match only at the beginning of the string

Regular Expressions in Action



Python’s RE Module

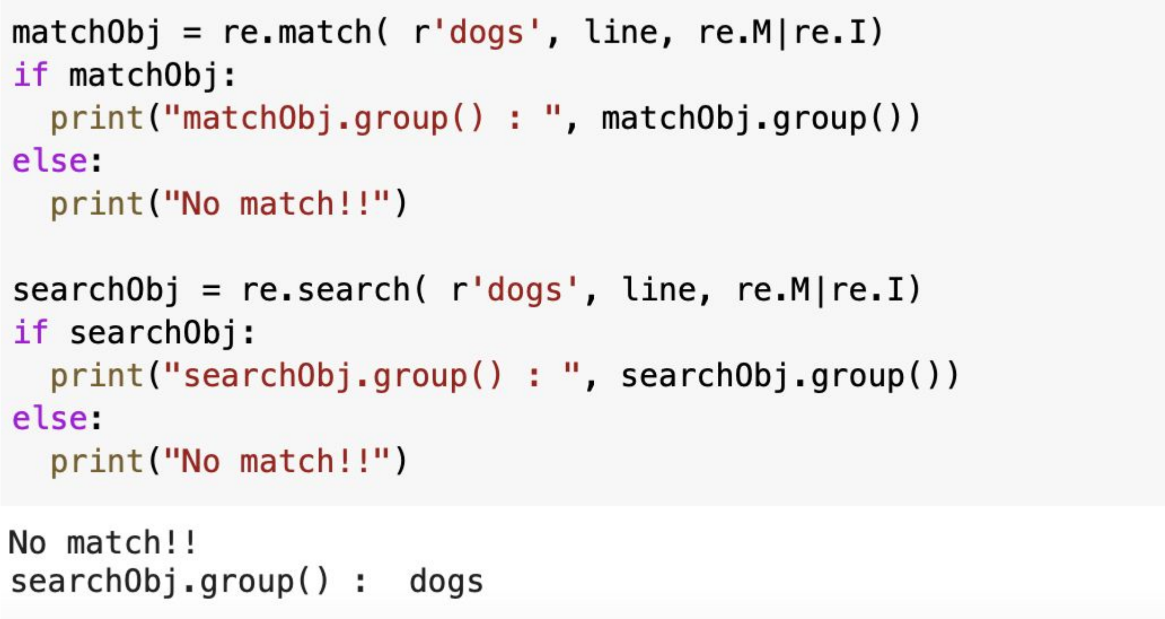
```python
import re
line = "Cats are smarter than dogs"
searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)
if searchObj:
  print("searchObj.group() : ", searchObj.group())
  print("searchObj.group(1) : ", searchObj.group(1))
  print("searchObj.group(2) : ", searchObj.group(2))
else:
  print("No match!!")
```

```
searchObj.group() :  Cats are smarter than dogs
searchObj.group(1) :  Cats
searchObj.group(2) :  smarter
```

Search checks for a match anywhere in the string

Regular Expressions in Action

Python’s RE Module



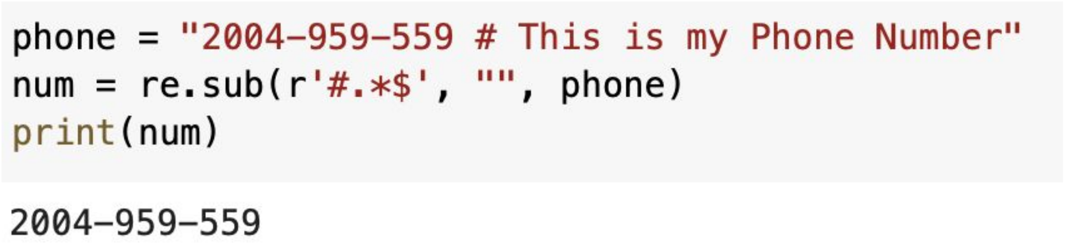
```
matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
  print("matchObj.group() : ", matchObj.group())
else:
  print("No match!!")

searchObj = re.search( r'dogs', line, re.M|re.I)
if searchObj:
  print("searchObj.group() : ", searchObj.group())
else:
  print("No match!!")
```

```
No match!!
searchObj.group() :  dogs
```

Match vs Search

Regular Expressions in Action

Python's RE Module



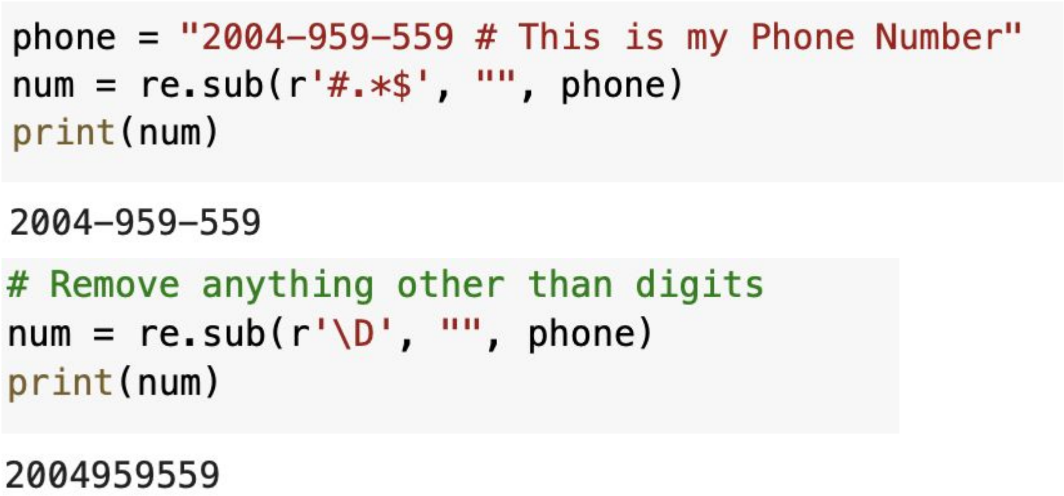
```
phone = "2004-959-559 # This is my Phone Number"
num = re.sub(r'#.*$', "", phone)
print(num)
```

```
2004-959-559
```

Replace functionality

Regular Expressions in Action

Python’s RE Module



```
phone = "2004-959-559 # This is my Phone Number"
num = re.sub(r'#.*$', "", phone)
print(num)
```

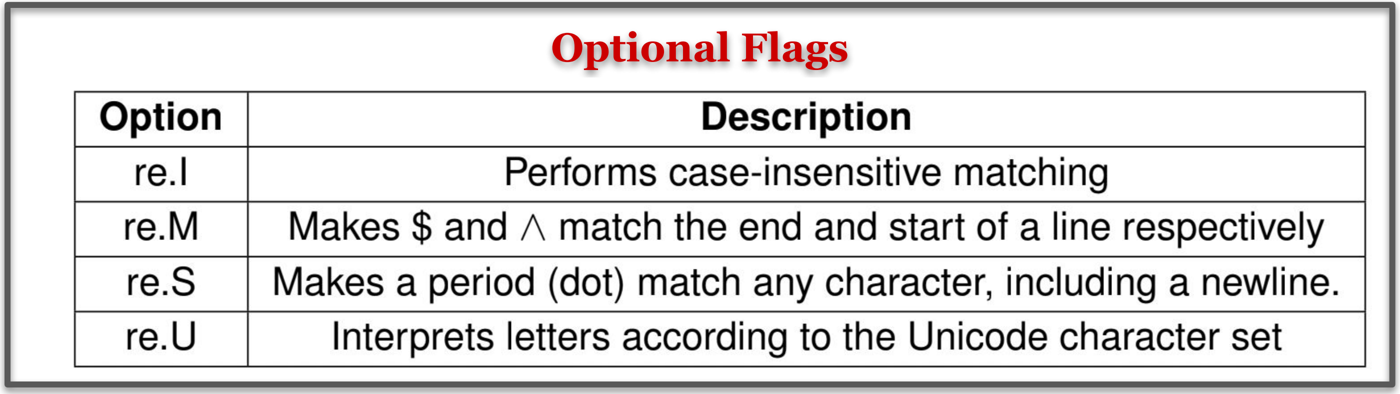
```
2004-959-559
```

```
# Remove anything other than digits
num = re.sub(r'\D', "", phone)
print(num)
```

```
2004959559
```

Replace functionality

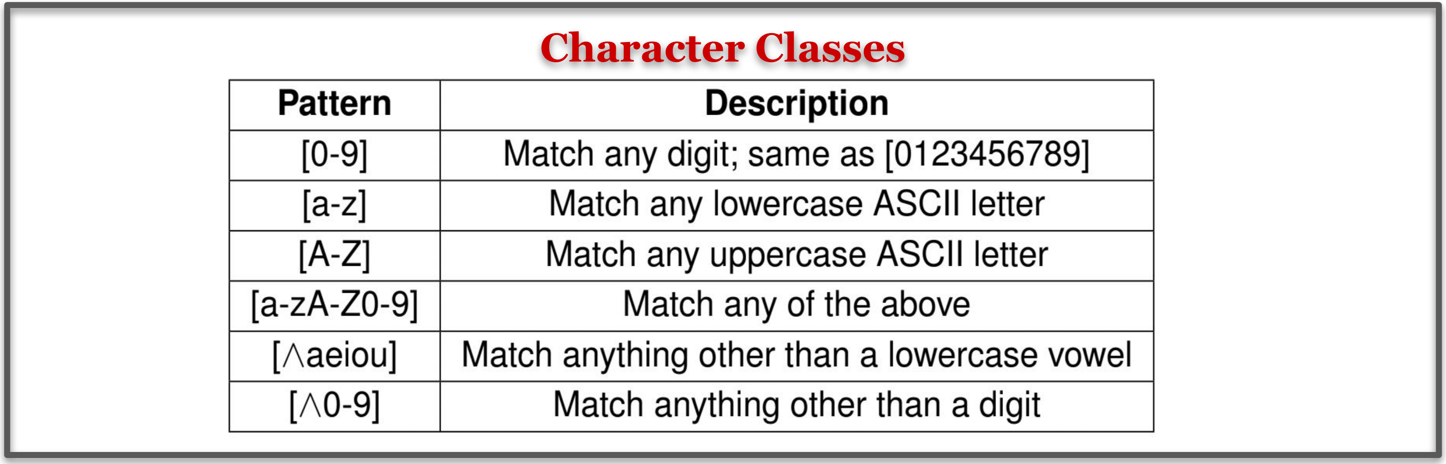
Regular Expressions in Action



Optional Flags

Option	Description
re.I	Performs case-insensitive matching
re.M	Makes $ and ∧ match the end and start of a line respectively
re.S	Makes a period (dot) match any character, including a newline.
re.U	Interprets letters according to the Unicode character set

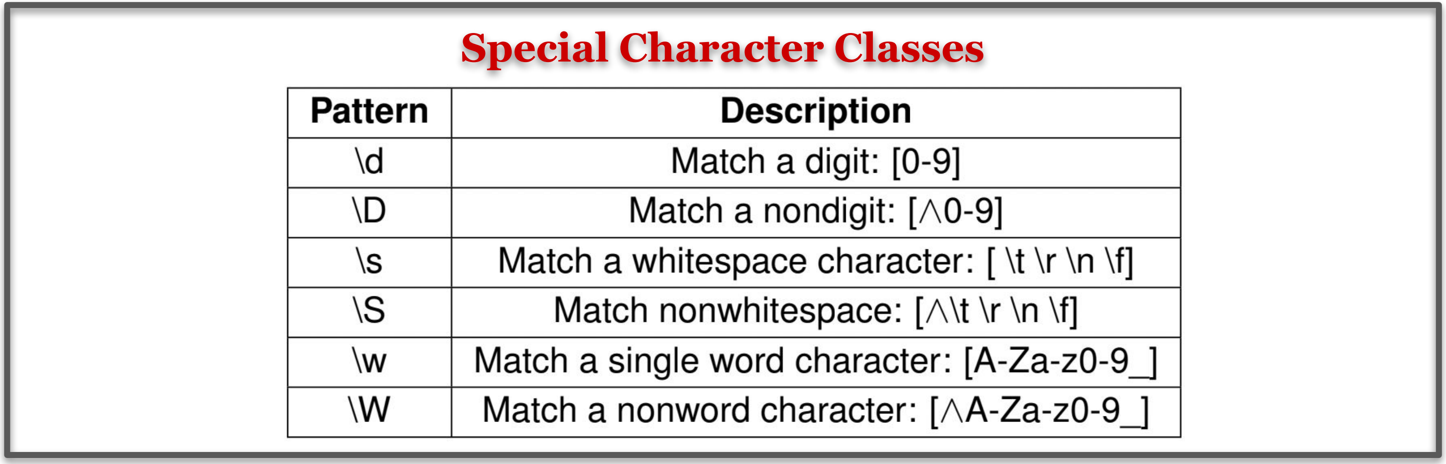
Regular Expressions in Action



Character Classes

Pattern	Description
[0-9]	Match any digit; same as [0123456789]
[a-z]	Match any lowercase ASCII letter
[A-Z]	Match any uppercase ASCII letter
[a-zA-Z0-9]	Match any of the above
[∧aeiou]	Match anything other than a lowercase vowel
[∧0-9]	Match anything other than a digit

Regular Expressions in Action



Special Character Classes

Pattern	Description
\d	Match a digit: [0-9]
\D	Match a nondigit: [∧0-9]
\s	Match a whitespace character: [\t \r \n \f]
\S	Match nonwhitespace: [∧\t \r \n \f]
\w	Match a single word character: [A-Za-z0-9_]
\W	Match a nonword character: [∧A-Za-z0-9_]

Email: singh.mayank@iitgn.ac.in
Webpage: https://mayank4490.github.io/