Travel is to make a journey or to have an adventure to
somewhere by bicycle, train, airplane, car, motorcycle, or
boat. It could be an exploration to somewhere new planned
or unplanned to meet new people, new things and new
places. There are different types of adventures waiting
for you to explore.
Q W E R

Text Processing

Regular Expression and thaipynlp

Thanachart Ritbumroong, Ph.D.

Regular Expression

RegEx Module

- Python has a built-in package called re, which can be used to work with Regular Expressions.
- A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.
- RegEx can be used to check if a string contains the specified search pattern.

Match Object

- A Match Object is an object containing information about the search and the result.

- The Match object has properties and methods used to retrieve information about the search, and the result:

  .span() returns a tuple containing the start-, and end positions of the match.
  .string returns the string passed into the function
  .group() returns the part of the string where there was a match

Basic Regular Express

```
import re


#match -
 A Match Object is an object containing information about the search a
nd the result.

print(re.match("ab","ABC"))
```

Basic Regular Express

```
#match

print(re.match("ab","abc"))
```

Basic Regular Express

```
#match.span() returns a tuple containing the start-
, and end positions of the match.

re.match("ab","abc").span()
```

Basic Regular Express

```
#match.string returns the string passed into the function

re.match("ab","abc").string
```

Basic Regular Express

```
#match.group() returns the part of the string where there was a match

re.match("ab","abc").group()
```

Basic Regular Express

```
#match.group() returns the part of the string where there was a match

re.match("ab","abc").group()
```

RegEx Functions

- The re module offers a set of functions that allows us to search a string for a match

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

RegEx Functions

```
#findall() returns a list containing all matches.

pattern = "AIS"
text = '''Manage all your AIS numbers with one login. Manage things
much easier with one login My AIS allows you to access all your AIS nu
mber accounts by logging in only once.
          It is really convenient to manage your AIS postpaid, AIS 1-
2-Call! And Fibre accounts all at one go.'''


re.findall(pattern, text)
```

RegEx Functions

```
#finditer() returns an iterator yielding match objects matching the re
gex pattern.

for match in re.finditer(pattern, text):
  print(f"start index {match.start()}, end index {match.end()}")
```

RegEx Functions

```
#search() function searches the string for a match, and returns a Matc
h object if there is a match. If there is more than one match, only th
e first occurrence of the match will be returned.

re.search(pattern, text)
```

RegEx Functions

```
#split() returns a list where the string has been split at each match.

re.split("!", text)
```

RegEx Functions

```
#sub() replaces the matches with the text of your choice.

re.sub("!", "*", text)
```

RegEx Functions

```
#subn() The re.subn() is similar to re.sub() except it returns a tuple
 of 2 items containing the new string and the number of substitutions
made.

re.subn("!", "*", text)
```

RegEx Functions

```
#compile(pattern) Regular expressions are handled as strings by Python
. However, with compile(), you can computer a regular expression patte
rn into a regular expression object.

pattern = 'AIS'

AIS_pattern = re.compile(pattern)

AIS_pattern.findall(text)
```

Metacharacters

- Metacharacters are characters with a special meaning:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he{2}o"
|	Either or	"falls|stays"

Metacharacters

```
def patternFinder(pattern, text):
  i = 1
  for match in re.finditer(pattern, text):
    print(f"{i}. match {match.group()} - 
 start index {match.start()}, end index {match.end()}")
    i += 1
  if not any(re.finditer(pattern, text)):
    print("No matches")
```

Metacharacters

```
text = '''Manage all your AIS numbers with one login. Manage things mu
ch easier with one login My AIS allows you to access all your AIS numb
er accounts by logging in only once.
          It is really convenient to manage your AIS postpaid, AIS 1-
2-Call! And Fibre accounts all at one go.'''
```

```
def patternFinder(pattern, text):
  i = 1
  for match in re.finditer(pattern, text):
    print(f"{i}. match {match.group()} -
 start index {match.start()}, end index {match.end()}")
    i += 1
  if not any(re.finditer(pattern, text)):
    print("No matches")
```

Metacharacters

```
pattern = "AIS"

patternFinder(pattern, text)
```

Metacharacters

```
#[] A set of characters

pattern = "[AEIOU]"

patternFinder(pattern, text)
```

Metacharacters

```
#. Any character (except newline character)

pattern = "[AEIOU].."

patternFinder(pattern, text)
```

Metacharacters

```
#^ Starts with

pattern = "^AIS"

patternFinder(pattern, text)
```

Metacharacters

```
#^ Starts with

pattern = "^Manage"

patternFinder(pattern, text)
```

Metacharacters

```
#$ Ends with

pattern = "AIS$"

patternFinder(pattern, text)
```

Metacharacters

```
#* Zero or more occurrences

pattern = "the*"
text = "they thou their thief thee"

patternFinder(pattern, text)
```

Metacharacters

```
#+ one or more occurrences

pattern = "the+"
text = "they thou their thief thee"

patternFinder(pattern, text)
```

Metacharacters

```
#? zero or one occurrences

pattern = "the?"
text = "they thou their thief thee"

patternFinder(pattern, text)
```

Metacharacters

```
#{} Exactly the specified number of occurrences

pattern = "the{2}"
text = "they thou their thief thee"

patternFinder(pattern, text)
```

Metacharacters

```
#| Either or

pattern = "ei|ef"
text = "they thou their thief thee"

patternFinder(pattern, text)
```

Special Sequences

- A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Special Sequences

```
text = '''Manage all your AIS numbers with one login. Manage things mu
ch easier with one login My AIS allows you to access all your AIS numb
er accounts by logging in only once.
          It is really convenient to manage your AIS postpaid, AIS 1-
2-Call! And Fibre accounts all at one go.’‘’

#\A Returns a match if the specified characters are at the beginning o
f the string

pattern = "\AManage"

patternFinder(pattern, text)
```

Special Sequences

```
#\b Returns a match where the specified characters are at the beginnin
g or at the end of a word (the "r" in the beginning is making sure tha
t the string is being treated as a "raw string")

pattern = r"\bAIS"

patternFinder(pattern, text)
```

Special Sequences

```
#\B Returns a match where the specified characters are at the beginnin
g or at the end of a word (the "r" in the beginning is making sure tha
t the string is being treated as a "raw string")

pattern = r"\BAIS"

patternFinder(pattern, text)
```

Special Sequences

```
#\d Returns a match where the string contains digits (numbers from 0-
9)
#\D Returns a match where the string DOES NOT contain digits

pattern = "\d"

patternFinder(pattern, text)
```

Special Sequences

```
#\s Returns a match where the string contains a white space character
#\S Returns a match where the string DOES NOT contain a white space ch
aracter

pattern = "\s"

patternFinder(pattern, text)
```

Special Sequences

```
#\w Returns a match where the string contains any word characters (cha
racters from a to Z, digits from 0-9, and the underscore _ character)
#\W Returns a match where the string DOES NOT contain any word charact
ers

pattern = "\s\w{10}\s"

patternFinder(pattern, text)
```

Special Sequences

```
#\Z Returns a match if the specified characters are at the end of the
string

pattern = "g..\Z"

patternFinder(pattern, text)
```

Special Sequences

```
#\Z Returns a match if the specified characters are at the end of the
string

pattern = "g..\Z"

patternFinder(pattern, text)
```

Sets

- A set is a set of characters inside a pair of square brackets [] with a special meaning:

Set	Description
[arn]	Returns a match where one of the specified characters (a, r, or n) are present
[a-n]	Returns a match for any lower case character, alphabetically between a and n
[^arn]	Returns a match for any character EXCEPT a, r, and n
[0123]	Returns a match where any of the specified digits (0, 1, 2, or 3) are present
[0-9]	Returns a match for any digit between 0 and 9
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string

Sets

```
#[arn] Returns a match where one of the specified characters (a, r, or
 n) are present
#[^arn] Returns a match for any character EXCEPT a, r, and n

pattern = "[ABCDEFGH]"

patternFinder(pattern, text)
```

Sets

```
#[a-n] Returns a match for any lower case character,
alphabetically between a and n

pattern = "[A-H]"

patternFinder(pattern, text)
```

Sets

```
#[0123] Returns a match where any of the specified digits (0, 1, 2, or
 3) are present

pattern = "[12345]"

patternFinder(pattern, text)
```

Sets

```
#[0-9] Returns a match for any digit between 0 and 9

pattern = "[1-5]"

patternFinder(pattern, text)
```

Sets

```
#[0-5][0-9] Returns a match for any two-digit numbers from 00 and 59

pattern = "[0-9][0-9]"

patternFinder(pattern, text)
```

Sets

```
#[a-zA-
Z] Returns a match for any character alphabetically between a and z,
lower case OR upper case

pattern = "\A[a-zA-Z]+\s"

patternFinder(pattern, text)
```

pythainlp

pythainlp Module

- PyThaiNLP is a Python library for Thai Natural Language Processing.

  https://pythainlp.github.io/

Basic pythainlp

```
import pythainlp

#check for Thai character

pythainlp.util.isthai("เอไอเอส")
```

Basic pythainlp

```
#sorting according to Thai dictionary

words = ["กิน","กัน","ก่อน"]

pythainlp.util.collate(words)
```

Word Tokenization

```
text = "ครอบคลุมทุกไลฟ์สไตล์ ทั้ง กินดื่ม ฟู้ดเดลิเวอรี ช้อปปิ้ง สุขภาพ และการแพทย์“

#maximum matching algorithm (default)

print(pythainlp.word_tokenize(text, engine="newmm"))

#logest algorithm

print(pythainlp.word_tokenize(text, engine="longest"))
```

Word Tokenization

●●●●

```
from pythainlp.corpus.common import thai_words
from pythainlp.util import Trie

new_words = {"ฟู้ดเดลิเวอรี"}

words = new_words.union(thai_words())

custom_dictionary_trie = Trie(words)

print(pythainlp.word_tokenize(text, custom_dict=custom_dictionary_trie
))
```

Word Tokenization

```
#syllable tokenization

from pythainlp.tokenize import syllable_tokenize

text = "เอไอเอส"

syllable_tokenize(text)
```

Word Tokenization

```
#romanize

from pythainlp.transliterate import romanize

text = "เอไอเอส"

romanize(text)
```

Word Tokenization

```
#Soundex

from pythainlp.soundex import lk82, metasound, udom83

print(lk82("รถ") == lk82("รด"))
print(metasound("รถ") == metasound("รด"))
print(udom83("รถ") == udom83("รด"))
```

Word Tokenization

```
texts = ["รถ","รด"]
for text in texts:
    print(
        "{} - lk82: {} - udom83: {} - metasound: {}".format(
            text, lk82(text), udom83(text), metasound(text)
        )
    )
```

Word Tokenization

```
#spellchecking

from pythainlp import spell

spell("อนุญาติ")
```

Word Tokenization

```
#Part-of-speech tagging

from pythainlp.tag import pos_tag, pos_tag_sents

pos_tag(["ฉัน","ชอบ","กิน","ขนม"])
```

Word Tokenization

```
#Named-entity tagging

from pythainlp.tag.named_entity import ThaiNameTagger

ner = ThaiNameTagger()
ner.get_ner("ฉันเจอเขาที่ตึกชินวัตรเมื่อวานนี้")
```

Word Tokenization

```
#word normalization

from pythainlp.util import normalize

text = "แปลก"#เเปลก

normalize(text) == "แปลก"#แปลก
```

Word Tokenization

```
#word vector

import pythainlp.word_vector

pythainlp.word_vector.similarity("ลูกชาย","บุตรชาย")
```