데이터분석 프로그래밍
텍스트 처리

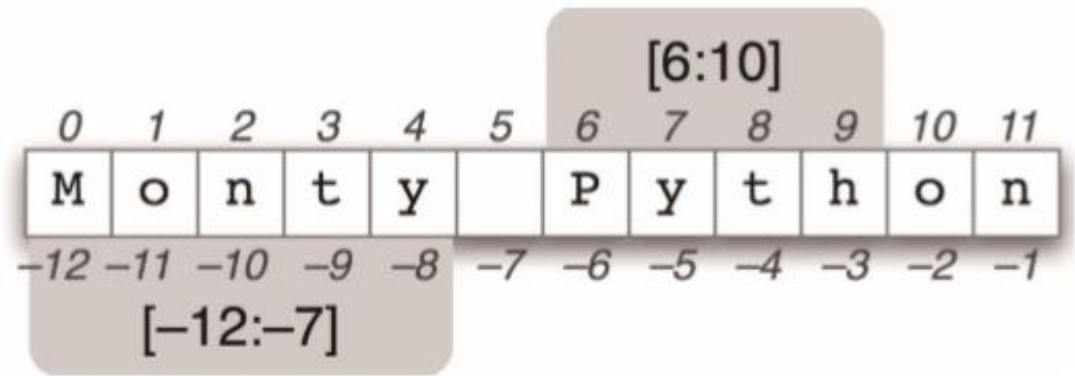
임현기

텍스트 데이터

- 텍스트는 인간이 효율적으로 정보를 교환하는 데에 중요한 수단
- 텍스트 데이터
  - 구조화된 문서: HTML, XML, CSV, JSON 등
  - 구조화되지 않은 문서

- 일반적으로 원천 데이터는 가공된 형태가 아님
  - 따라서 이들을 수정하여 사용해야 함

개별 문자 추출



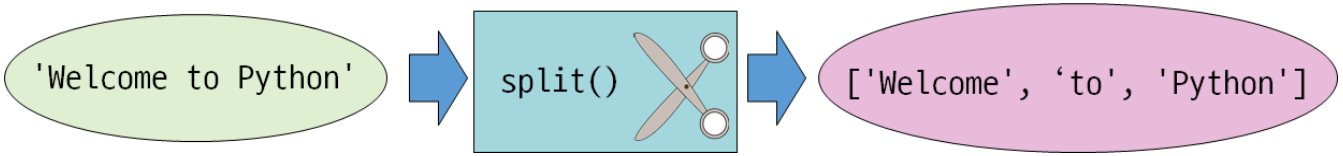

```
>>> s[6:10]
'Pyth'
>>> s[-12:-7]
'Monty'
```

```
>>> t = s[:-2]
>>> t
'Monty Pyth'
```

```
>>> t = s[-2:]          # t는 s의 인덱스 -2에서부터 마지막 원소까지 가져온다
>>> t
'on'
>>> s[:-2] + s[-2:]
'Monty Python'
```

split() 메소드

- 문자열 안에 쉼표나 빈칸 등의 구분자가 있을 때




```
>>> s = 'Welcome to Python'
>>> s.split()
['Welcome', 'to', 'Python']
```

```
>>> s = '2021.8.15'
>>> s.split('.')
['2021', '8', '15']
```

```
>>> s = 'Hello, World!'
>>> s.split(',')          # 쉼표(,)를 구분문자로 사용하므로 W 앞에 공백이 생김
['Hello', ' World!']
```

split() 메소드

- strip(): 공백을 제거하는 메소드

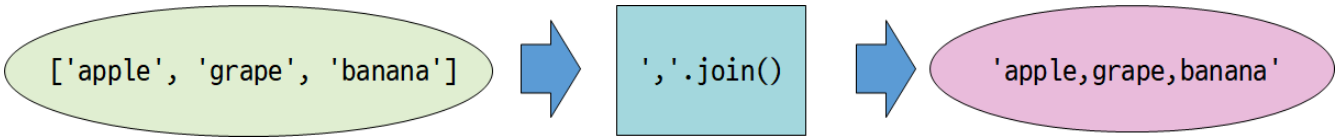
```
>>> s = 'Hello, World!'
>>> s.split(', ')          # W 앞의 공백을 제거함, 하지만 공백이 두 개일 경우 사용불가
['Hello', 'World!']
>>> s = 'Welcome, to,  Python, and ,  bla, bla   '
>>> [x.strip() for x in s.split(',')]
['Welcome', 'to', 'Python', 'and', 'bla', 'bla']
```

- 개별 문자로 모두 분해하려면 list() 메소드

```
>>> list('Hello, World!')
['H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!']
```

문자열 이어붙이기

- join() 메소드




```
>>> ','.join(['apple', 'grape', 'banana'])
'apple,grape,banana'
```

쉼표를 이용하여
세 단어를 연결함.

문자열 이어붙이기

```
>>> '-'.join('010.1234.5678'.split('.'))  # .으로 구분된 전화번호를 하이픈으로 고치기
'010-1234-5678'
```

```
>>> '010.1234.5678'.replace('.','-')
'010-1234-5678'
```

```
>>> s = 'hello world'
>>> clist = list(s)
>>> clist
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
>>> ''.join(clist)
'hello world'
```

```
>>> a_string = 'Actions \n\t speak louder than words' # 줄바꿈과 탭이 포함된 문자열
>>> a_string
'Actions \n\t speak louder than words'
>>> print(a_string)
Actions
         speak louder than words
>>> word_list = a_string.split()
>>> word_list
['Actions', 'speak', 'louder', 'than', 'words']
>>> refined_string = ' '.join(word_list)    # a_string에 있는 줄바꿈, 탭문자가 없어진다
>>> print(refined_string)
Actions speak louder than words
```

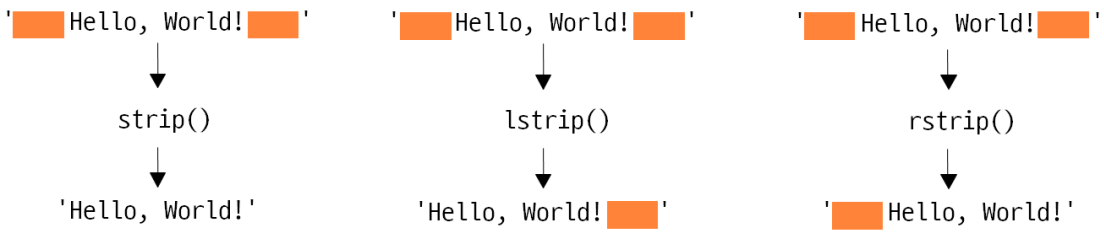
대소문자 변환, 문자열 삭제

```
>>> s = 'Hello, World!'
>>> s.lower()
'hello, world!'
>>> s.upper()
'HELLO, WORLD!'
```

– capitalize(): 첫번째 글자만 대문자로 변환

대소문자 변환, 문자열 삭제

- 공백제거 strip() 메소드




```
>>> s = '   Hello, World!   '
>>> s.strip()          # 왼쪽과 오른쪽의 공백문자를 모두 제거한다
'Hello, World!'
>>> s.lstrip()         # 왼쪽의 공백문자만 제거한다
'Hello, World!   '
>>> s.rstrip()         # 오른쪽의 공백문자만 제거한다
'   Hello, World!'
```

대소문자 변환, 문자열 삭제

- 특정 문자 삭제

```
>>> s = '########this is an example#####'
>>> s.strip('#')       # 문자열의 앞 뒤에 있는 해시문자를 제거한다
'this is an example'
```

```
>>> s = '########this is an example#####'
>>> s.lstrip('#')
'this is an example#####'
>>> s.rstrip('#')
'########this is an example'
>>> s.strip('#').capitalize()    # 샵문자를 제거하고 문장의 첫글자를 대문자로 만든다
'This is an example'
```

대소문자 변환, 문자열 삭제

- find(): 문자열에서 지정된 부분 문자열의 인덱스 반환
  - 지정된 문자를 찾지 못할 경우 -1 반환

```
>>> s = 'www.booksr.co.kr'
>>> s.find('.kr')
13
>>> s.find('x')       # 'x' 문자열이 없을 경우 -1을 반환함
-1
```

문자열 처리 함수

- count(): 부분 문자열이 등장하는 횟수 반환

```
>>> s = 'www.booksr.co.kr'     # 생능출판사의 홈페이지
>>> s.count('.')               # . 이 몇번 나타나는가를 알려준다
3
```

- ord(): 문자의 유니코드 반환
- chr(): 입력된 유니코드 값에 해당하는 문자 반환

```
>>> s = 'www.booksr.co.kr'
>>> ord(max(s))    # s문자열 내에서 유니코드 값이 가장 큰 값의 유니코드 값을 반환
119
>>> ord(min(s))    # s문자열 내에서 유니코드 값이 가장 작은 값의 유니코드 값을 반환
46
>>> chr(119), chr(46)  # 유니코드 값 119, 46에 해당하는 문자를 반환
('w', '.')
```

문자열 처리 함수

- string 모듈
  - 파이썬에서 문자열 처리를 도와주는 모듈
  - ascii_uppercase / asci_lowercase

```
>>> import string
>>> src_str = string.ascii_uppercase
>>> print('src_str =', src_str)
src_str = ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

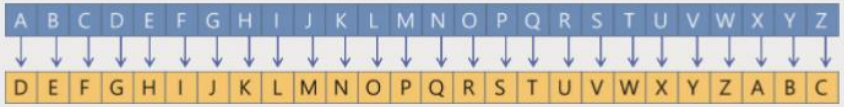
문자열 처리 함수

- 'ABCD…YZ'를 'BCDE…YZA'로 바꾸려면

```
>>> src_str = string.ascii_uppercase
>>> dst_str = src_str[1:] + src_str[:1]
>>> print('dst_str =', dst_str)
dst_str = BCDEFGHIJKLMNOPQRSTUVWXYZA
```

```
>>> n = src_str.index('A')
>>> print('src_str의 A 인덱스 =', n)
src_str의 A 인덱스 = 0
>>> print('src_str의 A 위치에 있는 dst_str의 문자 =', dst_str[n])
src_str의 A 위치에 있는 dst_str의 문자 = B
```

카이사르의 암호는 로마의 장군인 카이사르가 동맹군들과 소통하기 위해 만든 암호인데, 간단한 치환암호의 일종이다. 이 암호는 암호화하고자 하는 내용을 알파벳 별로 일정한 거리만큼 밀어서 다른 알파벳으로 치환하는 방식이다. 예를 들어 알파벳 A가 있을 경우 알파벳에서 3칸 이동한 알파벳 D로 표기할 수 있다. 이 방식의 경우 다음과 같이 대응되는 치환식을 사용한다.



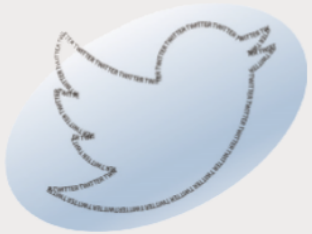
예를 들어 : 'ATTACK ON MIDNIGHT' 이라는 문자는 이 치환표에 의해서 'DWWDFN RQ PLGQLJKW'로 치환된다. 사용자로부터 대문자로 이루어진 문장을 입력 받아서 이와 같은 출력을 수행하는 프로그램을 작성하여라.

원하는 결과

```
문장을 입력하시오: ATTACK ON MIDNIGHT
암호화된 문장 : DWWDFN RQ PLGQLJKW
```

```python
import string

src_str = string.ascii_uppercase
dst_str = src_str[3:] + src_str[:3]

def ciper(a):           # 암호화 코드를 만드는 함수
    idx = src_str.index(a)
    return dst_str[idx]

src = input('문장을 입력하시오: ')
print('암호화된 문장 : ', end='')

for ch in src:
    if ch in src_str:
        print(ciper(ch), end='')
    else:
       print(ch, end='')

print()
```

트위터 메시지에서 추출할 수 있는 가장 기본적인 것은 각 트윗의 단어 수이다. 일반적으로 부정적인 감정은 긍정적인 것보다 적은 양의 단어를 포함한다고 한다. 트윗에서 단어의 개수를 추출하여서 발신자의 감정을 판단해 보기로 했다. split() 함수를 사용하여 트윗에서 단어를 추출하는 일부터 해 보자. 다음과 같은 문자열 t가 주어졌을 때 이 문자열 내의 단어의 개수를 다음과 같이 출력하도록 하여라.



```
t = "There's a reason some people are working to make it harder to vote,
especially for people of color. It's because when we show up, things change."
```

원하는 결과

```
word count: 26
```

```
t = "There's a reason some people are working to make it harder to vote,
especially for people of color. It's because when we show up, things change."
length = len(t.split(' '))
print('word count:', length)
```

도전문제 9.1

원천 데이터를 처리하다 보면 특정 상표나 회사명을 익명 처리해야 하는 경우가 있다. 아래와 같은 트윗 데이터는 KT, SKT, Samsung, LG와 같은 회사명이 나타나고 있다. 이런 데이터에서 회사명은 모두 *로 가려져서 나타나게 변경해 보자.

원천 데이터

```
t = '[ARTICLE] 200820 BLACKPINK Jennie is regarded to have great effect on KT Mystic Red as it was chosen by 50% of those who prebooked for the Samsung Galaxy Note 20 (LG U+ Mystic Pink 30%, SKT Mystic Blue not disclosed) '
```

처리된 결과: print(t)

```
[article] 200820 blackpink jennie is regarded to have great effect on * mystic red as it was chosen by 50% of those who prebooked for the * galaxy note 20 (* u+ mystic pink 30%, * mystic blue not disclosed)
```

문자열의 글자들을 모두 소문자로 바꾸고 구두점들을 제거해 보자. 소문자로 변경하는 이유는 동일한 단어가 중복하여 처리되는 것을 방지할 수 있기 때문이다. 예를 들어 이렇게 하지 않으면 단어 수를 계산하는 동안 'Car'과 'car'는 다른 단어로 간주된다. 이렇게 되면 사실상 동일한 데이터를 서로 다른 것으로 다루게 된다. 이러한 문제를 피하기 위해 문자열 데이터 전체를 소문자나 대문자로 바꾸는 일은 가장 흔한 데이터 사전 준비 과정 중에 하나이다. 다음과 같은 문장 t가 주어졌을 때, 이를 다음과 같은 문장 l로 변환하는 프로그램을 작성하여라.

```
t = "It's Not The Right Time To Conduct Exams. MY DEMAND IN BOLD AND CAPITAL. NO EXAMS IN COVID!!!"
```

원하는 결과

```
>>> l
'it's not the right time to conduct exams. my demand in bold and capital. no exams in covid!!!'
```

```
>>> t = "It's Not The Right Time To Conduct Exams. MY DEMAND IN BOLD AND
CAPITAL. NO EXAMS IN COVID!!!"
>>> l = t.lower()
>>> l
'it's not the right time to conduct exams. my demand in bold and capital.
no exams in covid!!!'
```

일회용 암호(OTP)는 컴퓨터 또는 디지털 장치에서 하나의 로그인 세션 또는 트랜잭션에만 유효한 암호이다. OTP는 인터넷 뱅킹, 온라인 거래 등과 같은 거의 모든 서비스에 사용된다. 일반적으로 다음과 같은 6자리의 숫자 조합이다.

파이썬에서 random() 함수는 난수 생성 함수로서 임의의 OTP를 생성하는 데 사용할 수 있다. 파이썬을 사용하여 다음과 같이 OTP 번호를 생성해보자. 우리 프로그램은 몇 자리의 비밀번호를 원하는지 물어본 뒤에 입력된 자릿수를 가진 비밀번호를 생성한다.

원하는 결과

```
몇 자리의 비밀번호를 원하십니까? 10
9370598010
몇 자리의 비밀번호를 원하십니까? 6
332443
```

```
import random

n_digits = int(input('몇 자리의 비밀번호를 원하십니까? '))

otp = ''
for i in range(n_digits) :
      otp += str(random.randrange(0, 10))

print(otp)
```

도전문제 9.1

1회용 패스워드에는 숫자만이 아닌 영문자 대문자와 소문자도 포함될 수 있다. 위의 문제를 수정하여 영문자와 숫자가 포함된 패스워드를 생성하여라.

워드 클라우드


BIG
DATA
INFORMATION
LARGE
STORAGE
SETS
MILLION
SOFTWARE
SAN
NAS
SCIENCE
FUTURE
GOVERNMENT
VOLUME
PROCESS
TECHNOLOGIES
MANAGEMENT
DATABASES
PROCESSING
SIZE
EXABYTES
PETABYTES
ANALYSIS
RESEARCH
LEARNING
WORLD
INTERNET
SEARCH
BASED
DEVELOPMENT
TECHNOLOGY
BILLION
NEW
PARALLEL
INITIATIVE
TERABYTES
TIME
SOCIAL
YEARS
NETWORKS
SYSTEMS
AMOUNT
APPLICATIONS
ANALYTICS

워드 클라우드

- 단어의 빈도 또는 중요성을 크기로 나타내는 텍스트 데이터 시각화 기술
- 파이썬에서 워드 클라우드를 생성하려면 matplotlib, pandas, wordcloud 모듈이 있어야
  - pip install wordcloud

- 워드 클라우드 생성을 위해서는 데이터의 역할을 수행할 텍스트가 필요

워드 클라우드

- wikipedia.page(title)이라고 하여 title을 제목으로 하는 위키백과 페이지를 얻을 수 있음

```
import wikipedia

# 위키백과 사전의 컨텐츠 제목을 명시해 준다
wiki = wikipedia.page('Artificial intelligence')
# 이 페이지의 텍스트 컨텐츠를 추출하도록 한다
text = wiki.content
```

  - 텍스트 데이터가 준비되면, 이 데이터를 이용하여 워드 클라우드 이미지를 생성

워드 클라우드

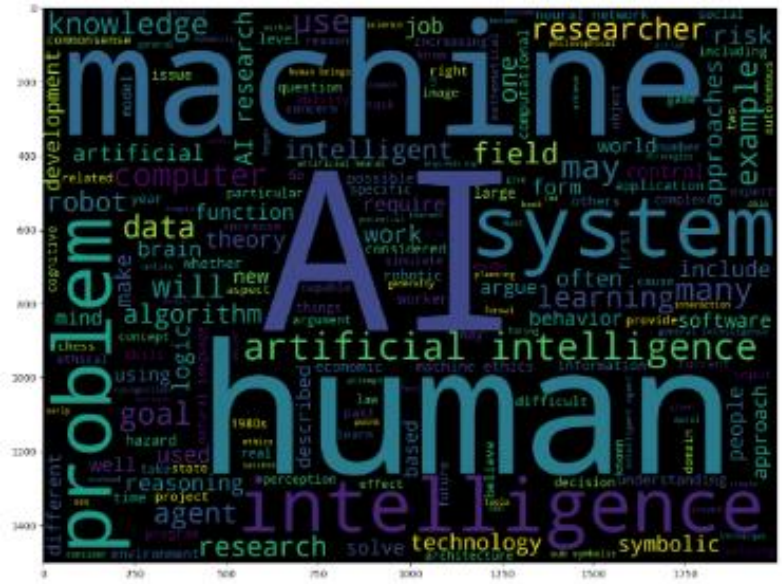
- 가로와 세로 크기를 클래스의 생성자에 넘겨 주어 이미지의 크기를 정하고, generate() 함수를 불러 워드 클라우드를 만들 재료가 될 텍스트 데이터를 인자로 넘김
- 인자들 중에서 width는 워드 클라우드 이미지의 너비이고 height는 높이를 픽셀단위로 표현

```
from wordcloud import WordCloud

# 워드 클라우드를 생성하기 위해 위의 코드를 삽입할 것
wordcloud = WordCloud(width = 2000, height = 1500).generate(text)
```

워드 클라우드

- 이 워드 클라우드 이미지를 matplotlib의 imshow()를 이용하여 그림

```
import matplotlib.pyplot as plt

plt.figure(figsize=(40, 30))
# 화면에 이미지를 그려준다
plt.imshow(wordcloud)
plt.show()
```

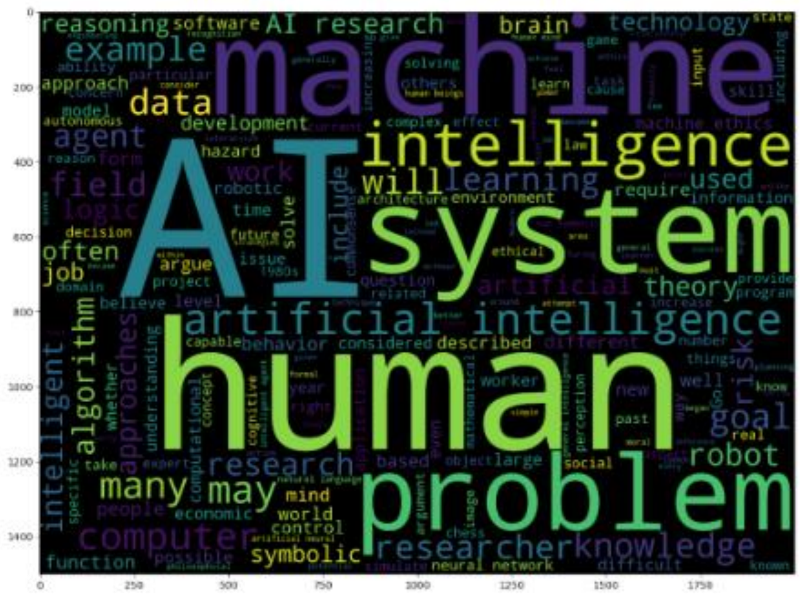


워드 클라우드

- 중지어(stop word)
  - 많은 텍스트 데이터에 쓰이지만, 특별히 중요한 의미를 가지지 않는 단어들
  - wordcloud 모듈의 STOPWORDS에 정리되어 있음
  - 중지어를 추가하고 싶으면 다음과 같이 새로운 집합과 합집합을 구해서 사용

```
from wordcloud import WordCloud, STOPWORDS
# 중지어가 제외된 워드 클라우드를 만들자
s_words = STOPWORDS.union( {'one', 'using', 'first', 'two', 'make', 'use'} )
wordcloud = WordCloud(width = 2000, height = 1500,
                      stopwords = s_words).generate(text)
```

워드 클라우드

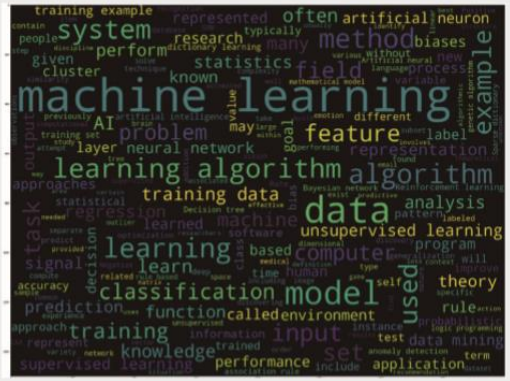


reasoning
software
AI research
brain
technology
machine
example
approach
data
development
agent
intelligence
AI
system
will
learning
field
often
job
argue
theory
artificial intelligence
human
algorithm
intelligent
approaches
goal
robot
research
many
may
problem
computer
researcher
knowledge
symbolic

워드 클라우드

도전문제 9.4

이제 다음과 같이 Python과 Machine Learning을 이용하여 워드 클라우드를 각각 만들어 보아라.



정규식과 메타문자

- 정규표현식(regular expression)
  - 특정한 규칙을 가지고 있는 문자열들을 표현하는데 사용되는 규칙을 가진 언어

- 파이썬에서 정규식을 사용하려면 re 모듈을 포함

```
>>> import re
>>> txt1 = 'Life is too short, you need python.'
>>> txt2 = 'The best moments of my life.'
>>> print(re.search('Life', txt1))    # 문장 안에 Life가 있는가 검사함
<_sre.SRE_Match object; span=(0, 4), match='Life'>
>>> print(re.search('Life', txt2))    # 문장 안에 Life가 있는가 검사함
None
```

  - ‘Life’가 있는지 검사

정규식과 메타문자

- 검사 결과를 해석하기 위해 re.search()의 결과 값을 match라는 변수에 할당 후 group, start, end, span 함수를 사용
  - group

```
>>> match = re.search('Life', txt1)
>>> match.group()      # 검색의 결과가 여러개의 그룹으로 나타날 경우에 필요함
'Life'
```

정규식과 메타문자

- group() 예제 코드

```
[ ]  1 import re
     2
     3 text = "Please call 010-2345-1234."
     4
     5 # group() 기능을 테스트하는 코드, 2개의 그룹이 생성된다
     6 regex = re.compile('(₩d{3})-(₩d{4}-₩d{4})')
     7
     8 match_obj = regex.search(text)
```

```
[ ]  1 print(match_obj.group())
```

```
010-2345-1234
```

```
[ ]  1 # 1 그룹
     2 print(match_obj.group(1))
```

```
010
```

```
     1 # 2 그룹
     2 print(match_obj.group(2))
```

```
2345-1234
```

정규식과 메타문자

- group() 메소드는 정규표현식의 검색 결과가 여러 개일 경우 묶음을 위해서 필요
  - start()와 end()는 일치하는 문자의 시작과 끝 인덱스를 나타냄
  - span()은 일치하는 구간을 슬라이싱하기 위한 인덱스

```
>>> match.start()
0
>>> match.end()
4
>>> match.span()
(0, 4)
>>> txt1[0:4]     # span (0, 4) 값을 이용해서 인덱싱을 하자
'Life'
```

정규식과 메타문자

```
>>> import re
>>> txt1 = 'Life is too short, you need python.'
>>> txt2 = 'The best moments of my life.'
>>> print(re.search('Life', txt1))    # 문장 안에 Life가 있는가 검사함
<_sre.SRE_Match object; span=(0, 4), match='Life'>
>>> print(re.search('Life', txt2))    # 문장 안에 Life가 있는가 검사함
None
```

```
>>> print(re.search('Life|life', txt2))    # 문장 안에 Life또는 life가 있는가 검사함
<_sre.SRE_Match object; span=(23, 27), match='life'>
```

```
>>> print(re.search('[Ll]ife', txt2))     # 문장 안에 Life 혹은 life가 있는가 검사함
<_sre.SRE_Match object; span=(23, 27), match='life'>
```

- [Ll]ife는 문자 선택의 범위를 표현
- [], -, |와 같은 특별한 의미를 가지는 문자를 메타문자라고 함

정규식과 메타문자

- 메타문자^
  - 첫 단어가 Life인 문장만을 검색

```
>>> txt1 = 'Life is too short, you need python'
>>> txt2 = 'The best moments of my life'
>>> txt3 = "My Life My Choice."
>>> print(re.search('^Life', txt1))    # 제일 첫 단어로 Life가 있는가 검사함
<_sre.SRE_Match object; span=(0, 4), match='Life'>
>>> print(re.search('^Life', txt2))    # 제일 첫 단어로 Life가 있는가 검사함
None
>>> print(re.search('^Life', txt3))    # 제일 첫 단어로 Life가 있는가 검사함
None
```

- 메타문자 $
  - 끝 문자 검색

```
>>> txt1 = 'Who are you to judge the life I live'
>>> txt2 = 'The best moments of my life'
>>> print(re.search('life$', txt1)) # life가 마지막 단어로 포함되어 있는가 검사
None
>>> print(re.search('life$', txt2)) # life가 마지막 단어로 포함되어 있는가 검사
<_sre.SRE_Match object; span=(23, 27), match='life'>
```

메타문자

식	기능	설명
^	시작	문자열의 시작을 의미함
$	끝	문자열의 끝을 의미함
.	문자	한 개의 문자
\d	숫자	한 개의 숫자
\w	문자와 숫자	한 개의 문자나 숫자
\s	공백문자	공백 문자 (스페이스, 탭, 줄바꿈 등)
\S	공백제외 문자	공백 문자를 제외한 모든 문자가 될 수 있음
*	반복	앞 문자가 0번 이상 반복
+	반복	앞 문자가 1번 이상 반복
[abc]	문자 선택 범위	a, b, c 가운데 하나의 문자
[^abc]	문자 제외 범위	a, b, c가 아닌 어떤 문자
|	또는	| 앞의 문자 또는 뒤의 문자를 의미함

메타문자

- .
  - 임의의 문자 한 개를 의미
  - ABA, ABBA, ABBBA 중 A..A 조건을 만족하는 문자는 ABBA

```
>>> re.search('A..A', 'ABA')              # 조건에 맞지 않음
>>> re.search('A..A', 'ABBA')             # 조건에 맞음
<_sre.SRE_Match object; span=(0, 4), match='ABBA'>
>>> re.search('A..A', 'ABBBA')            # 조건에 맞지 않음
```

메타문자

- *
  - 직전에 있는 임의의 패턴과 0회 이상 반복되는 문자와 매치 됨

```
>>> re.search('AB*', 'A')                # 'A'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(0, 1), match='A'>
>>> re.search('AB*', 'AA')               # 'A'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(0, 1), match='A'>
>>> re.search('AB*', 'J-HOPE')           # 조건에 맞지 않음
>>> re.search('AB*', 'X-MAN')            # 'A'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(3, 4), match='A'>
>>> re.search('AB*', 'CABBA')            # 'ABB'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(1, 4), match='ABB'>
>>> re.search('AB*', 'CABBBBBA')         # 'ABBBBB'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(1, 7), match='ABBBBB'>
```

메타문자

- ?
  - 직전에 있는 임의의 문자를 0회 또는 1회 반복한 패턴

```
>>> re.search('AB?', 'A')                # 'A'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(0, 1), match='A'>
>>> re.search('AB?', 'AA')               # 'A'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(0, 1), match='A'>
>>> re.search('AB?', 'J-HOPE')           # 조건에 맞지 않음
>>> re.search('AB?', 'X-MAN')            # 'A'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(3, 4), match='A'>
>>> re.search('AB?', 'CABBA')            # 'AB'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(1, 3), match='AB'>
>>> re.search('AB?', 'CABBBBBA')         # 'AB'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(1, 3), match='AB'>
```

  - *과의 차이: CABBA, CABBBBBA에서 조건에 맞는 문자가 AB

메타문자

- +
  - 직전에 있는 임의의 패턴을 1회 또는 그 이상의 수로 많이 반복된 패턴에 대해 매치

```
>>> re.search('AB+', 'A')                # 조건에 맞지 않음
>>> re.search('AB+', 'AA')               # 조건에 맞지 않음
>>> re.search('AB+', 'J-HOPE')           # 조건에 맞지 않음
>>> re.search('AB+', 'X-MAN')            # 조건에 맞지 않음
>>> re.search('AB+', 'CABBA')            # 'ABB'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(1, 4), match='ABB'>
>>> re.search('AB+', 'CABBBBBA')         # 'ABBBBB'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(1, 7), match='ABBBBB'>
```

  - AB+는 A와 매치되지 않고, AB, ABB, ABBB, CABBA와 같은 패턴의 문자열과 일치

메타문자

- findall()
  - 정규식을 만족하는 모든 문자열들을 추출

```
>>> txt3 = 'My life my life my life in the sunshine'
>>> re.findall('[Mm]y', txt3)
['My', 'my', 'my']
```

정규식을 활용한 검색

- UN의 세계 인권 선언문
  - https://www.un.org/en/universal-declaration-human-rights/
  - UNDHR.txt로 저장

UNDHR - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

Preamble

Whereas recognition of the inherent dignity and of the equal and inalienable rights of all members of the human family is the foundation of freedom, justice and peace in the world,

Whereas disregard and contempt for human rights have resulted in barbarous acts which have outraged the conscience of mankind, and the advent of a world in which human beings shall enjoy freedom of speech and belief and freedom from fear and want has been proclaimed as the highest aspiration of the common people,

Whereas it is essential, if man is not to be compelled to have recourse, as a last resort, to rebellion against tyranny and oppression, that human rights should be protected by the rule of law,

Whereas it is essential to promote the development of friendly relations between nations,

Whereas the peoples of the United Nations have in the Charter reaffirmed their faith in fundamental human rights, in the dignity and worth of the human person and in the equal rights of men and women and have determined to promote social progress and better standards of life in larger freedom,

Whereas Member States have pledged themselves to achieve, in co-operation with the United Nations, the promotion of universal respect for and observance of human rights and fundamental freedoms,

Whereas a common understanding of these rights and freedoms is of the greatest importance for the full realization of this pledge,

Now, Therefore THE GENERAL ASSEMBLY proclaims THIS UNIVERSAL DECLARATION OF HUMAN RIGHTS as a common standard of achievement for all peoples and all nations, to the end that every individual and every organ of society, keeping this Declaration constantly in mind, shall strive by teaching and education to promote respect for these rights and freedoms and by progressive measures, national and international, to

Ln 2, Col 377 100% Windows (CRLF) UTF-8

UNDHR - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

promote respect for these rights and freedoms and by progressive measures, national and international, to secure their universal and effective recognition and observance, both among the peoples of Member States themselves and among the peoples of territories under their jurisdiction.

Article 1.

All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.

Article 2.

Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, colour, sex, language, religion, political or other opinion, national or social origin, property, birth or other status. Furthermore, no distinction shall be made on the basis of the political, jurisdictional or international status of the country or territory to which a person belongs, whether it be independent, trust, non-self-governing or under any other limitation of sovereignty.

Article 3.

Everyone has the right to life, liberty and security of person.

Article 4.

No one shall be held in slavery or servitude; slavery and the slave trade shall be prohibited in all their forms.

Article 5.

No one shall be subjected to torture or to cruel, inhuman or degrading treatment or punishment.

Ln 2, Col 377 | 100% | Windows (CRLF) | UTF-8

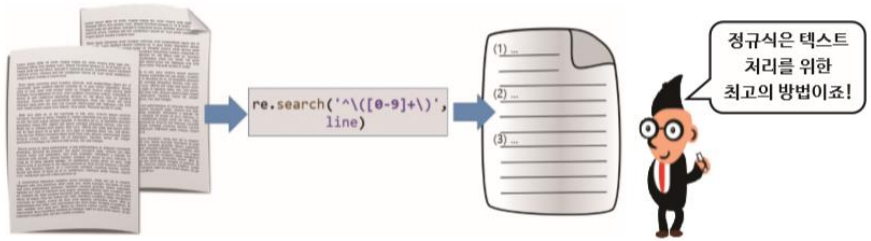
정규식을 활용한 검색

- 숫자 형태의 항으로 이루어진 문장들을 추출
  - [0-9]+ 또는 ₩d+로 표시
  - 모든 라인을 읽은 뒤에 rstrip()을 이용하여 오른쪽에 있는 모든 공백 문자 제거

정규식을 활용한 검색

```
import re

f = open('d:/data/UNDHR.txt')      # 현재 폴더에 있는 파일을 읽어온다.

for line in f:
    line = line.rstrip()
    if re.search('^\([0-9]+\)', line) :
        print(line)
```

```
(1) Everyone charged with a penal offence has the right to be presumed innocent
until proved guilty according to law in a public trial at which he has had all the
guarantees necessary for his defence.
(2) No one shall be held guilty of any penal offence on account of any act or
omission which did not constitute a penal offence, under national or international
law, at the time when it was committed. Nor shall a heavier penalty be imposed than
the one that was applicable at the time the penal offence was committed.
...
```



다음과 같은 수강 교과목 코드와 약칭으로된 문자열이 있다고 가정하자.

```
101 COM PythonProgramming
102 MAT LinearAlgebra
103 ENG ComputerEnglish
```

위의 텍스트는 '[수강 번호] [수강 코드] [과목 이름]' 형식으로 되어 있다. 위의 텍스트에서 수강 번호만을 다음과 같이 리스트 형식으로 추출해보자.

원하는 결과

```
['101', '102', '103']
```

```
import re

# 멀티라인 텍스트는 세 개의 따옴표를 사용하여 표현한다
text = '''101 COM PythonProgramming
102 MAT LinearAlgebra
103 ENG ComputerEnglish'''

s = re.findall('\d+', text)
print(s)
```

어떤 텍스트 데이터에 이메일 주소가 포함되어 있다고 하자. 예를 들어 수신된 이메일을 간략히 보고하는 아래와 같은 텍스트 데이터가 있다고 하자. 아래 예에서는 두 개의 전자우편 주소가 포함되어 있다. abc@facebook.com와 bbc@google.com이라는 이메일 주소가 그것이다. 이러한 이메일 주소만을 추출하고 싶다. 정규표현식을 이용하여 이를 추출해 보라.

```
txt = 'abc@facebook.com와 bbc@google.com에서 이메일이 도착하였습니다.'
```

원하는 결과

```
추출된 이메일 : ['abc@facebook.com', 'bbc@google.com']
```

```
import re

txt = 'abc@facebook.com와 bbc@google.com에서 이메일이 도착하였습니다.'
output = re.findall('\S+@[a-z.]+', txt)
print('추출된 이메일 :', output)
```

요즘 해킹을 막기 위하여 패스워드가 아주 복잡해지고 있다. 사용자가 입력한 패스워드를 검증하는 프로그램을 작성해보자. 사용자가 패스워드를 입력하도록 하고 이 패스워드가 조건에 맞지 않을 경우 다시 입력하도록 while 문을 사용하여라. 패스워드의 조건은 다음과 같으며 re 모듈을 임포트하여 정규식을 이용하도록 하자.

1. 최소 8글자라야 한다.
2. 적어도 하나의 영문자 대문자 및 소문자를 포함해야 한다.
3. 적어도 하나의 숫자를 포함해야 한다.
4. 다음에 나타난 특수문자[_, @, $,!]중 하나를 반드시 포함해야 한다.

원하는 결과

```
패스워드를 입력하세요 : qwerty1234
유효하지 않은 패스워드!
패스워드를 입력하세요 : 1abc@AB@!
유효한 패스워드
```

```
import re

while True:
    password = input("패스워드를 입력하세요 : ");
    if len(password)<8 or not re.search("[a-z]", password) or \
        not re.search("[A-Z]", password) or \
        not re.search("[0-9]", password) or not re.search("[_@$!]", password):
        print("유효하지 않은 패스워드!")
    else:
        print("유효한 패스워드")
        break
```

문자 대체

- sub()
  - 특정한 문자열을 다른 문자열로 대체

```
>>> import re
>>> s = 'I like BTS!'
>>> re.sub('BTS', 'Black Pink', s)
'I like Black Pink!'
```

- 특정한 수를 숨기고자 할 때

```
>>> s = 'My lucky number 2 7 99'
>>> re.sub('[0-9]+', '*', s)      # 숫자만 찾아서 *으로 바꿈
'My lucky number * * *'
>>> re.sub('\d+', '*', s)         # 숫자만 찾아서 *으로 바꿈
'My lucky number * * *'
```

문자 대체

- 특정 번호를 해시함수로 암호화 한 후에 공개

```
import re
s = 'My lucky number 2 7 99'

def hash_by_mult_and_modulo(m):
    n = int(m.group())                  # 매칭된 문자열을 가져와서 정수로 변환
    return str(n * 23435 % 973)         # 숫자에 해시함수를 적용하고 문자열로 만들어 반환

print(re.sub('[0-9]+', hash_by_mult_and_modulo, s))
```

```
My lucky number 166 581 433
```

파이썬의 정규표현식을 사용하여 트윗 메시지에서 사용자 메시지만을 추려보자. 즉 특수 문자나 URL, 해쉬 태그, 이메일 주소, RT, CC는 삭제한다

원하는 결과

```
트윗을 입력하시오: Good Morning! RT @PythonUser I like Python #Python
Good Morning!   I like Python
```

```
import re
tweet = input('트윗을 입력하시오: ')
tweet = re.sub('RT', '', tweet)     # RT 문자열을 삭제
tweet = re.sub('#\S+', '', tweet)  # 해시(#)다음에 나타나는 문자열을 삭제
tweet = re.sub('@\S+', '', tweet)  # 앳사인(@)다음에 나타나는 문자열을 삭제
print(tweet)
```