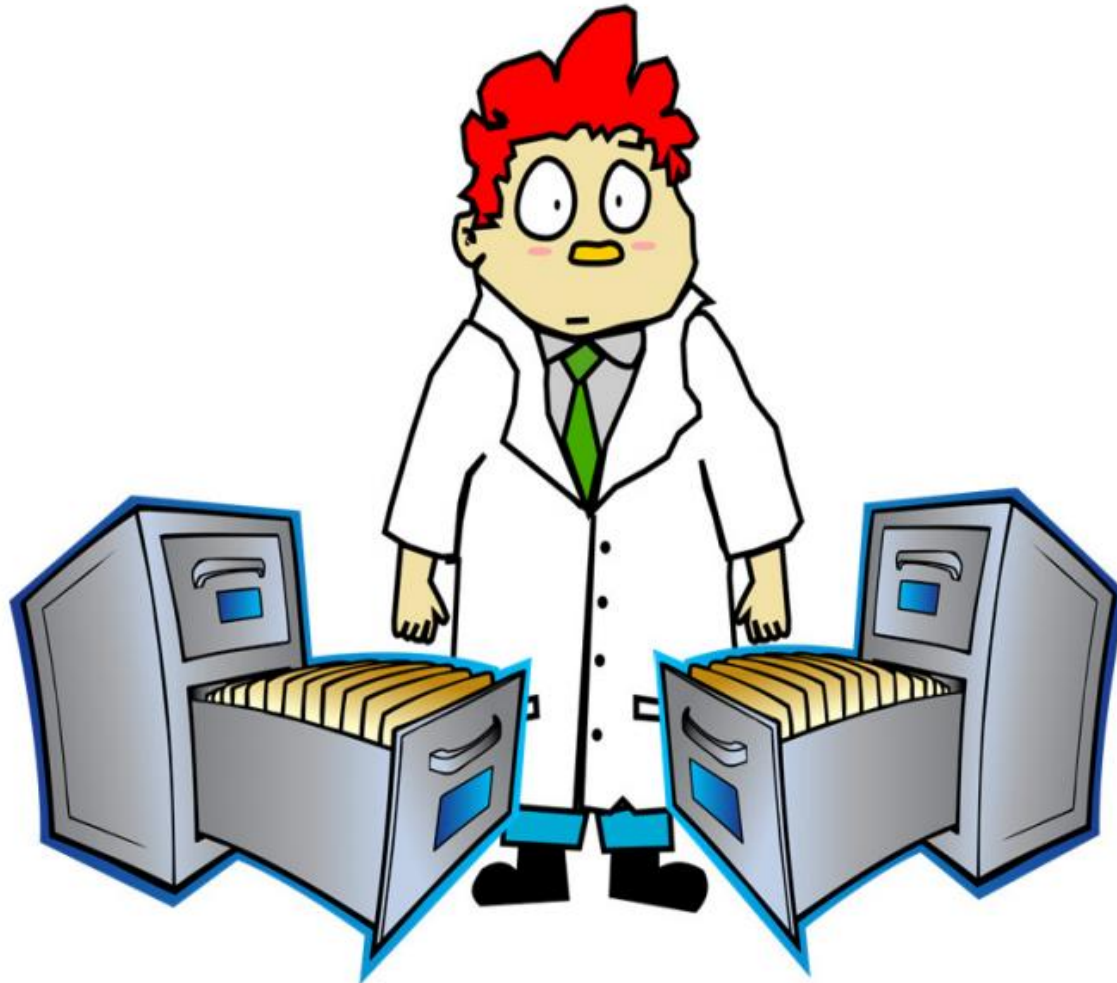


데이터 과학과 인공지능



따라하며 배우는

파이썬과 데이터 과학



9장 텍스트를 처리해보자

이장에서 배울 것들

- 텍스트 처리에 관련된 연산을 살펴 보아요.
- 문자열에 적용할 수 있는 다양한 메소드를 익혀 보아요.
- 이런 메소드를 이용하여 실제 텍스트 데이터를 다루는 연습을 해 보아요.
- 문서의 키워드와 핵심어를 시각적으로 표현하는 워드클라우드를 알아보고 구현해 보아요.
- 텍스트 데이터를 다루는 데에 효율적인 정규식을 이해해 보아요.
- 정규식을 이용하여 강력한 검색 문자열 기능을 직접 만들어 보아요.
- 텍스트 데이터를 다루는 프로그램을 만들 수 있는 기본 역량을 갖추어요.

9.1 텍스트 데이터란 무엇인가

- 텍스트는 인간이 오랫동안 정보를 효율적으로 교환하는 데에 가장 중요한 수단으로 사용해 왔다.
- 텍스트 데이터는 구조화된 문서(HTML, XML, CSV, JSON 파일)와 구조화되지 않은 문서(자연어로 된 텍스트)로 나눌 수 있다.
- 일반적으로 원천 데이터는 가공된 형태가 아니기 때문에 우리는 이들 데이터를 수정하여서 완전한 데이터로 만들어야 한다.

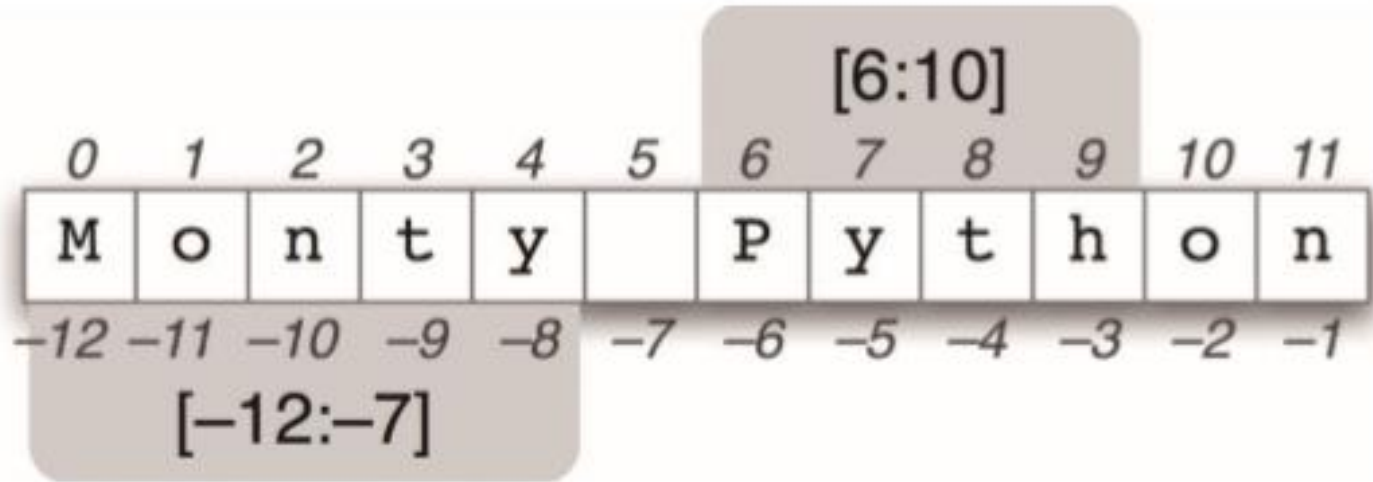
9.1 텍스트 데이터란 무엇인가



- 위의 그림은 wordart.com에서 제공하는 **워드 클라우드** word cloud라는 시각화 기술의 예시이다.
- 파이썬의 기본 라이브러리에서 제공하는 문자열 함수들만 이용하여도 텍스트 데이터를 어느 정도 처리할 수 있지만 BeautifulSoup, csv, json, nltk와 같은 우수한 외부 모듈을 사용하면 비교적 쉽게 텍스트를 처리하고 분석할 수 있다.
- 이 장에서는 텍스트 데이터에 대한 인덱싱과 슬라이싱과 같은 간단한 기능부터 워드 클라우드를 생성하는 라이브러리, 그리고 텍스트 데이터를 다루는 데에 좋은 도구가 되는 **정규식** regular expression 표현을 살펴해보도록 하자.

9.2 문자열에서 개별 문자들을 뽑아보자

- 문자열에서 사용할 수 있는 가장 기본적인 작업은 아마도 문자열 안에 있는 개별 문자들을 추출하는 작업일 것이다



9.2 문자열에서 개별 문자들을 뽑아보자

- 변수 `s`에 'Monty Python'이라는 문자열이 저장되어 있다고 하자.

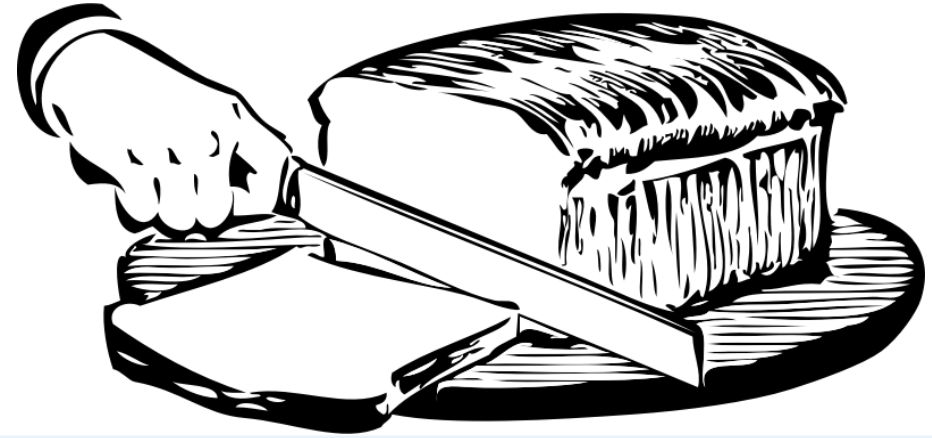
```
>>> s = 'Monty Python'
>>> s[0]
'M'
```

- 위의 예제와 같이 `s[0]`과 같이 인덱스 `0`을 이용해서 문자 'M'에 접근 가능하며, `s[11]`은 마지막 문자 'n'이 된다.

9.2 문자열에서 개별 문자들을 뽑아보자

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

- 문자열의 오른쪽 끝의 두 문자만을 제외하고 슬라이싱하여 복사하려 하면 다음과 같은 표현식을 사용할 수 있다.



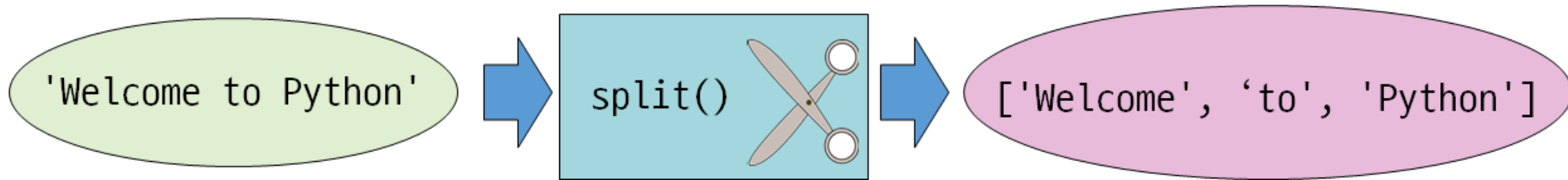
```
>>> t = s[:-2]
>>> t
'Monty Pyth'
```

- 만일 오른쪽 끝에 있는 두 개의 문자를 선택하려면 `s[-2:]`를 사용하면 된다. 그러므로 `s[:-2] + s[-2:]`는 원래 문자열이 된다.

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


9.3 split() 메소드는 문자열을 잘 잘라줘요

- 문자열 안의 단어들이 쉼표나 빈칸 등의 구분자로 분리되어 있다고 하자.



- 예를 들어 다음과 같은 `s` 문자열에 `split()`이라는 메소드를 사용하면 공백을 구분자로 사용하여 하나의 문자열을 3개의 문자열로 분리할 수 있다.

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

9.3 split() 메소드는 문자열을 잘 잘라줘요

- 반면 다음과 같이 마침표로 구분된 연,월,일이 있을 경우 마침표(.)를 split() 메소드의 인자로 주면 마침표 단위로 문자를 구분할 수 있다.

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

- 아래와 같이 쉼표로 'Hello, World!'를 구분해 보도록 하자.

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

- 실행 결과를 보면 'Hello'와 ' World!'로 분리되었음을 알 수 있다. 그러나, W 앞에 공백이 있어서 어색하게 보인다.

9.3 split() 메소드는 문자열을 잘 잘라줘요

- 공백이 두 칸 이상이거나 앞뒤에 공백이 있을 경우에는 아래와 같이 strip()을 이용하여 공백을 제거하는 것이 바람직할 것이다.

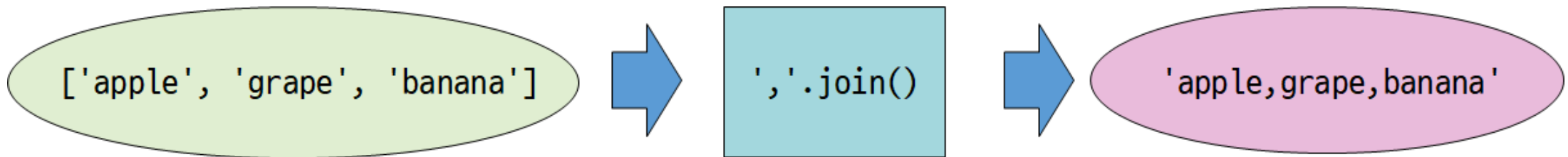
```
>>> s = 'Hello, World!'
>>> s.split(',')          # W 앞의 공백을 제거함, 하지만 공백이 두 개일 경우 사용불가
['Hello', 'World!']
>>> s = 'Welcome, to, Python, and , bla, bla '
>>> [x.strip() for x in s.split(',')]
['Welcome', 'to', 'Python', 'and', 'bla', 'bla']
```

- 만일 문자열을 모두 개별 문자들로 분해하려면 어떻게 하면 될까?
list()를 호출해주면 된다.

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

9.4 문자열을 이어붙이기는 파이썬에겐 쉬운 일

- `split()`가 문자열을 부분 문자열들로 분리하는 함수라면 `join()`은 반대로 부분 문자열들을 모아서 하나의 문자열로 만드는 역할을 하는 함수이다. `join()`을 호출할 때는 접착제 역할을 하는 문자를 지정할 수 있다.



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

쉼표를 이용하여 세
단어를 연결함.

9.4 문자열을 이어붙이는 것은 파이썬한테는 쉬운 일

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

- 위의 예제에서 보면 `join()`은 접착제 문자를 문자열 사이에만 넣고 문자열의 앞이나 뒤에는 넣지 않는 것을 알 수 있다.

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

9.4 문자열을 이어붙이는 것은 파이썬한테는 쉬운 일

- 또한 다음과 같이 `list()` 함수로 분리한 문자들을 모아서 다시 원래의 문자열로 만들때도 `join()`을 사용한다.

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

9.4 문자열을 이어붙이는 것은 파이썬한테는 쉬운 일

- `split()`와 `join()`을 함께 사용하면 문자열 중에서 필요 없는 공백을 제거할 수 있다.

```
>>> 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
```

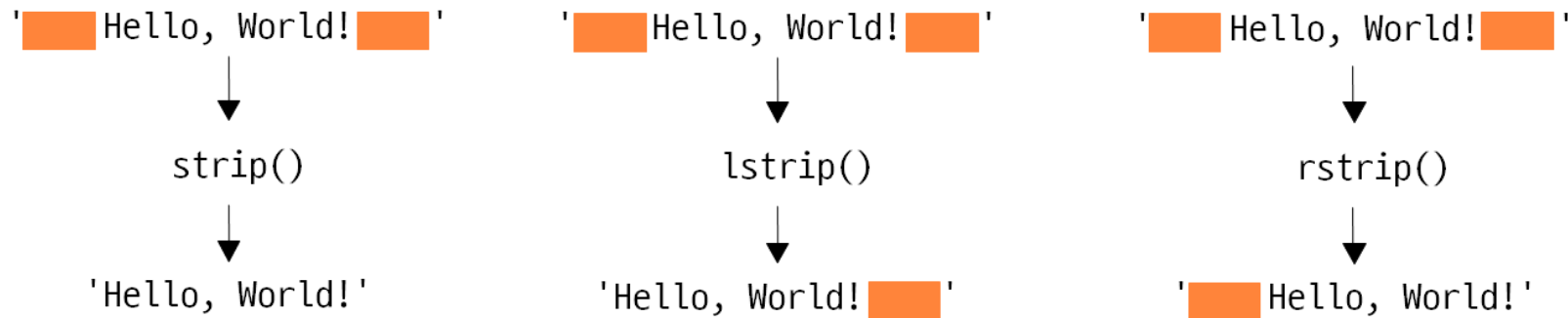
9.5 대문자와 소문자 변환, 그리고 문자열 삭제

- 문자열에서 대문자를 소문자로 변경하는 함수는 `lower()`이고 반대는 대문자로 변경하는 메소드는 `upper()` 함수이다. 첫 번째 문자만 대문자로 변환하는 함수는 `capitalize()`이다.

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


9.5 대문자와 소문자 변환, 그리고 문자열 삭제4

- 문자열 데이터를 처리할 때 문자열에서 원치 않는 공백을 제거하는 작업은 `strip()` 함수가 담당한다.



```
>>> 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!'
```

9.5 대문자와 소문자 변환, 그리고 문자열 삭제

- 만일 문자열의 앞과 뒤에있는 특정한 문자를 삭제하려면 문자를 `strip()`의 인자로 이 문자를 전달한다.

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

- `rstrip()`과 `lstrip()`에도 특정한 문자를 인자로 넣어줄 수 있다.

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

9.5 대문자와 소문자 변환, 그리고 문자열 삭제

- `find()` 메소드는 문자열에서 지정된 부분 문자열을 찾아서 그 인덱스를 반환한다. 지정된 문자를 찾지 못했을 경우에는 `-1`을 반환한다. 문자열 중에서 관심 있는 부분을 찾을 때 `find()` 함수를 사용하면 좋다.

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

9.6 다양한 문자열 처리 함수와 string 모듈

- 문자열에서 `count()` 메소드는 문자열 중에서 부분 문자열이 등장하는 횟수를 반환한다.

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

- 위에서 언급한 메소드 말고 파이썬 내장함수도 텍스트 데이터에 적용할 수 있다. `len()` 함수는 문자열의 길이를 반환한다.

9.6 다양한 문자열 처리 함수와 string 모듈

- 그리고 `max()`, `min()` 함수를 사용하여 가장 큰 문자, 작은 문자를 얻을 수 있다. 크다 작다의 기준은 유니코드 코드 값을 기준으로 한다.
- `ord()` 함수는 문자에 대한 유니코드 값을 얻을 수 있으며, `chr()` 값은 입력된 유니코드 값에 해당하는 문자를 반환하는 역할을 한다.

```
>>> 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', '.')
```

9.6 다양한 문자열 처리 함수와 string 모듈

- 파이썬에는 문자열 처리를 도와주는 string이라는 모듈이 있으며 이 모듈에 있는 `ascii_uppercase`는 알파벳 대문자를, `ascii_lowercase`는 알파벳 소문자들을 포함하고 있다.

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

9.6 다양한 문자열 처리 함수와 string 모듈

- 만일 'ABCDE..YZ' 까지의 문자열을 한 글자씩 이동시켜서 'BCDEF...ZA'와 같은 문자열을 만들기 위해서는 어떤 방법을 사용해야 할까?
- 이 경우 다음과 같이 간단한 슬라이싱과 덧셈 연산이면 충분하다.

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

9.6 다양한 문자열 처리 함수와 string 모듈

- 문자 A가 src_str의 몇 번째에 있는가는 다음과 같은 index() 메소드로 조회할 수 있으며 반대로 이 인덱스를 사용하여 dst_str을 조회하면 src_str의 'A' 위치에 대응하는 dst_str의 문자가 'B'라는 것을 확인할 수 있다.

```
>>> 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
```


LAB⁹-1 카이사르 암호를 만들어 보자

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.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

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



왔노라
보았노라
이겼노라!

LAB⁹-1 카이사르 암호를 만들어 보자

```
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()
```

LAB⁹⁻² 트위터 메시지 처리의 단어 추출

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 추출하여 발신자의 감정을 판단해 보기로 했다. `split()` 함수를 사용하여 트윗에서 단어를 추출하는 일부터 해 보자. 다음과 같은 문자열 `t`가 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
```



LAB⁹⁻² 트위터 메시지 처리의 단어 추출

```
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)
```

LAB⁹⁻² 트위터 메시지 처리의 단어 추출



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)
```

LAB⁹⁻³ 트위터 메시지의 대문자, 소문자 변환

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 1로 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

```
>>> 1
'it's not the right time to conduct exams. my demand in bold and capital.
no exams in covid!!!'
```


LAB⁹⁻³ 트위터 메시지의 대문자, 소문자 변환

```
>>> 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!!!'
```

LAB⁹-3 트위터 메시지의 대문자, 소문자 변환



도전문제 9.2

트윗 데이터에서 대문자나 느낌표가 많이 나타나는 것은 글쓰는 사람의 감정이 흥분하거나 분노한 상태임을 나타내는 경우가 많다. 주어진 원천 트윗 데이터에서 대문자와 느낌표가 몇 번 사용되었는지 계산하는 코드를 작성해 보라. 느낌표가 몇 개인지 헤아리는 것은 `count()` 함수를 사용하면 쉽게 할 수 있다. 대문자는 어떻게 셀 수 있을까? 원 트윗을 `list()`를 이용하여 하나 하나의 문자로 분리하여 리스트를 만들 수 있다. 그리고 이 리스트의 각 항목 문자가 `ch`라고 할 때, `ch.isupper()`를 호출하면 대문자인 경우 `True`가 반환된다.

느낌표 갯수 : 3

대문자 갯수 : 46

LAB⁹-4 1회용 패스워드를 만들어 보자

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

9.7 정보를 한눈에 보여주는 워드 클라우드

- **워드 클라우드** word cloud는 각 단어의 크기가 빈도 또는 중요성을 나타내는 텍스트 데이터 시각화 기술이다.
- 파이썬에서 워드 클라우드를 생성하려면 matplotlib, pandas, wordcloud 모듈이 있어야 한다. matplotlib, pandas는 이미 설치되어 있고 우리는 wordcloud 모듈만 새로 설치하면 된다.
- 워드 클라우드를 생성하기 위해서는 우선 원천 데이터의 역할을 수행할 텍스트를 준비해야 한다.
- 이러한 텍스트를 준비하기 위한 방법으로 위키백과의 내용을 가져오는 wikipedia 모듈이 있다. 이를 사용하기 위해서는 pip install wikipedia 명령을 통해 wikipedia 모듈을 먼저 설치하기만 하면 된다.

matplotlib, pandas가 설치되어 있는 경우 아래 명령으로 모듈 설치
pip install wikipedia wordcloud

9.7 정보를 한눈에 보여주는 워드 클라우드

- 그러면 아래와 같이 `wikipedia.page(title)`이라고 하여 `title`을 제목으로 하는 위키백과 페이지를 얻을 수 있다. 이 페이지의 텍스트 데이터를 얻고 싶으면 해당 페이지의 `content`를 사용하면 된다.

```
import wikipedia

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

- 이렇게 텍스트 데이터가 준비되면, 이 데이터를 이용하여 워드 클라우드 이미지를 생성한다.

9.7 정보를 한눈에 보여주는 워드 클라우드

- 설치를 해야 한다면 `pip install wordcloud` 명령으로 쉽게 설치할 수 있을 것이다.
- 아래와 같이 가로와 세로 크기를 클래스의 생성자에 넘겨 주어 이미지의 크기를 정하고, `generate()` 함수를 불러 워드 클라우드를 만들 재료가 될 텍스트 데이터를 인자로 넘겨준다.
- 인자들 중에서 `width`는 워드 클라우드 이미지의 너비이고 `height`는 높이를 픽셀단위로 표현한 것이다.

```
from wordcloud import WordCloud
```

```
# 워드 클라우드를 생성하기 위해 위의 코드를 삽입할 것
```

```
wordcloud = WordCloud(width = 2000, height = 1500).generate(text)
```

9.7 정보를 한눈에 보여주는 워드 클라우드

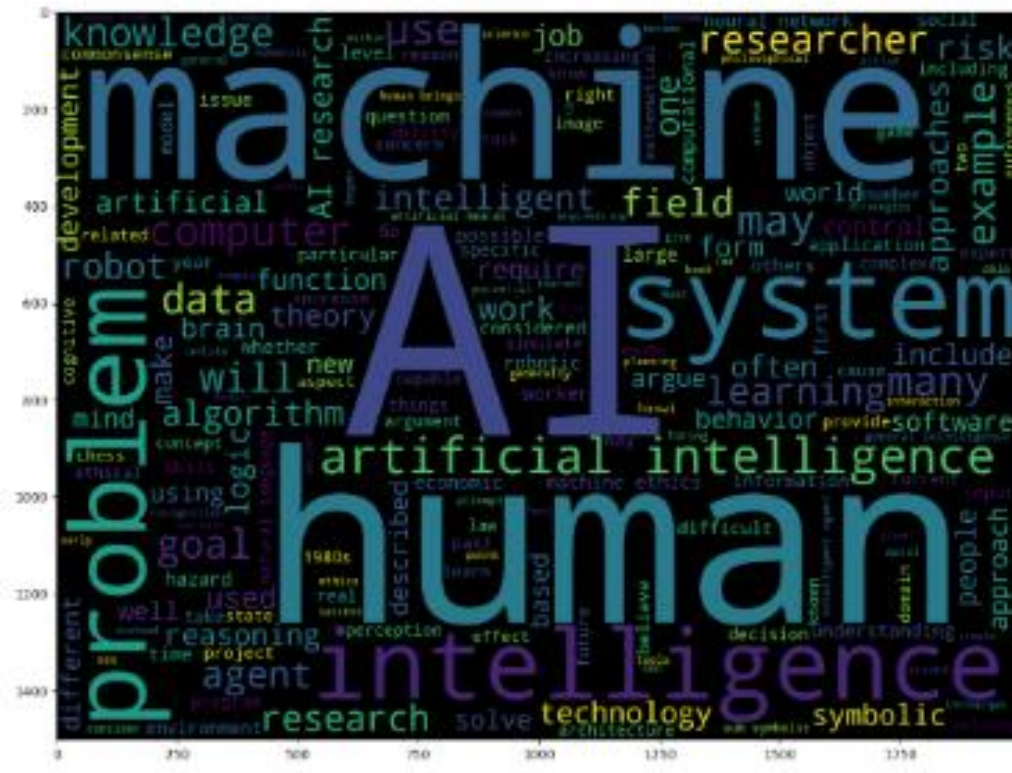
- 이제 이 워드 클라우드 이미지를 화면에 그리면 된다. 맷플롯립의 이미지 그리기 함수인 `imshow()`를 이용하면 쉽게 그릴수 있다.

```
import matplotlib.pyplot as plt

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

9.7 정보를 한눈에 보여주는 워드 클라우드

- 그림과 같이 워드 클라우드가 잘 그려졌다. 그런데, 많은 텍스트 데이터에는 자주 쓰이지만, 특별히 중요한 의미를 갖지 않는 단어들이 있다.
- 이러한 단어를 자연어 처리에서는 **중지어**stop word라고 한다.



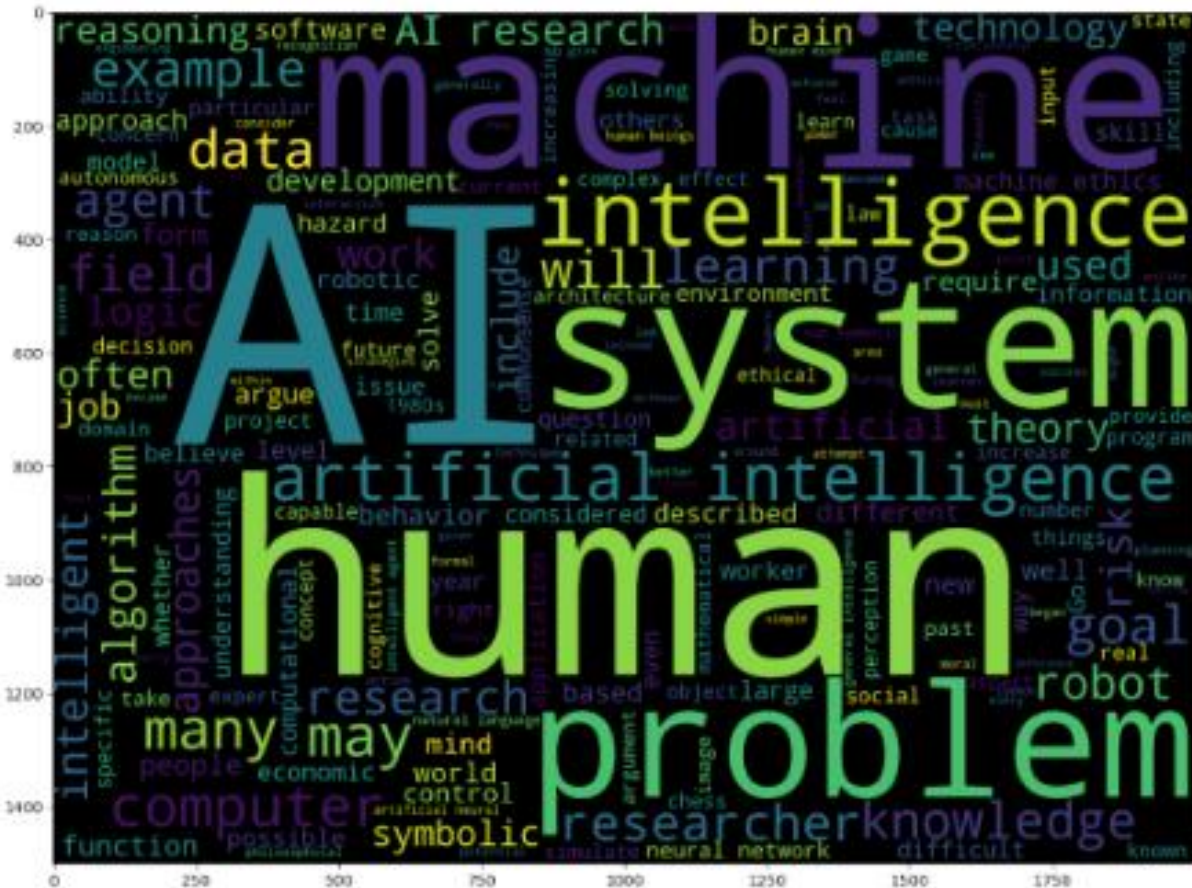
9.7 정보를 한눈에 보여주는 워드 클라우드

- 워드 클라우드를 만들 때 이러한 중지어를 제외하는 방법이 있다.
- 중지어 리스트는 wordcloud 모듈의 STOPWORDS에 정리되어 있다. STOPWORDS는 **집합set** 데이터로 정의되어 있으며, 우리가 중지어를 제거하고 싶으면 다음과 같이 사용하면 된다.

```
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)
```


9.7 정보를 한눈에 보여주는 워드 클라우드

- 이제 다음과 같이 이러한 중지어가 없어진 워드 클라우드를 얻을 수 있을 것이다.

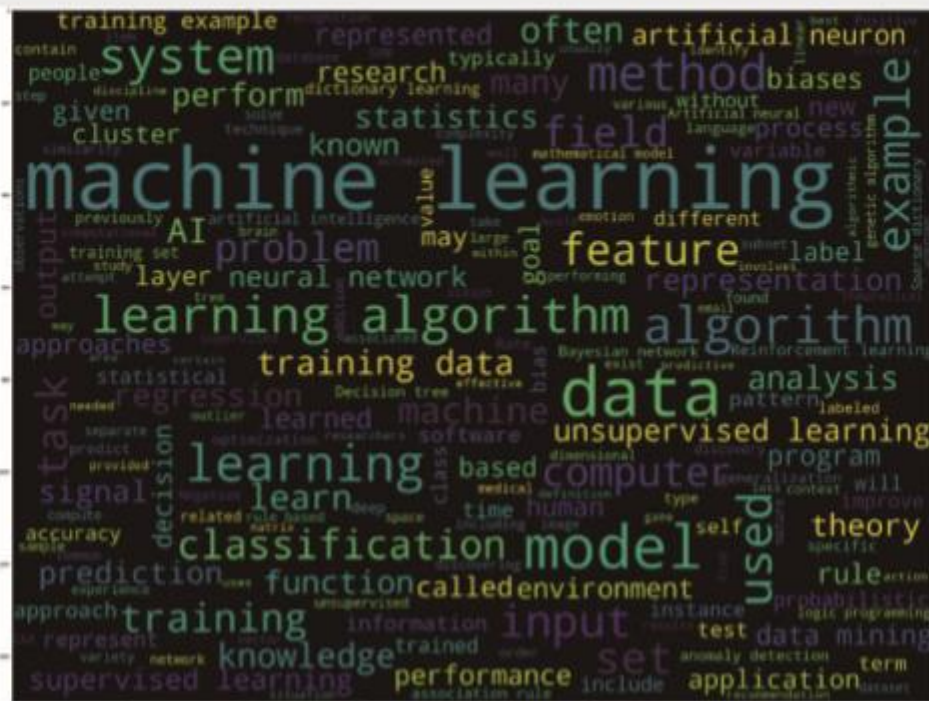


9.7 정보를 한눈에 보여주는 워드 클라우드



도전문제 9.4

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



힌트 : Python은 Python(programming langage)를 제목으로 사용

9.8 규칙을 이용해서 문자를 추출하는 정규식과 메타 문자

- **정규식** **regular expression**이란 특정한 규칙을 가지고 있는 문자열들을 표현하는데 사용되는 규칙을 가진 언어이다.
- 우선 간단한 예제를 이용해서 정규식을 익혀보도록 하자.



9.8 규칙을 이용해서 문자를 추출하는 정규식과 메타 문자

파이썬에서 정규식을 사용하려면 re 모듈을 포함시켜야 한다.

```
>>> 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
```

위의 결과를 보면 txt1에는 대문자 L로 시작하는 Life가 있기 때문에 첫 print()문의 결과를 보면, 일치한다고 나타난다.

- 이제 <_sre.SRE_Match object;..> 라는 다소 난해한 출력결과를 다시 한번 살펴보자.

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

- 이 출력을 해석하기 위해서 다음과 같이 re.search() 의 결과값을 match라는 변수에 할당하고 이 변수의 group() start(), end(), span() 함수를 호출해 보자.

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

- `group()` 메소드는 정규표현식의 검색 결과가 여러 개일 경우 묶음을 위해서 필요하며 `start()`와 `end()`는 정규표현식을 통해 일치하는 문자의 시작과 끝 인덱스를 나타낸다. 마지막으로 `span()`은 일치하는 구간을 슬라이싱하기 위한 인덱스를 나타낸다.

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

- 정규 표현식의 함수 `search(탐색문자열, 원본문자열)`의 결과를 잘 살펴보자. 원본문자열에서 찾고자 하는 탐색문자열을 찾을 수는 있으나 `txt2`의 탐색과 같이 소문자 `l`로 시작하는 `life`는 찾지 못하고 `Life`만 찾는 문제점이 있다. 이제 다음과 같은 방법으로 다시 시도해 보자

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

- 이 경우 txt2의 뒷부분에 'life'가 있기 때문에 span(23, 27) 결과를 통해 'life'가 있는 곳의 인덱스를 제대로 찾아 표시한다.
- 이 때 사용된 세로로 된 작대기 모양의 연산자 |는 'Life|life'를 찾으라는 의미가 아니고 **Life 또는 life 문자열을 찾으라**는 의미로 해석된다.

- Life나 life는 첫 글자만 다르고 나머지 글자들이 같기 때문에 다음과 같은 표현식을 사용해도 동일한 결과를 얻을 수 있다.

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

- 앞의 예제와 마찬가지로 [Ll]ife는 Life혹은 life를 의미하는데 **큰괄호 []는 문자 선택의 범위를 표현하는 문자**이다. 예를 들어 [0-9]의 경우 0부터 9까지의 모든 숫자 문자를 의미한다.
- 이와 같이 정규표현식에서 **[], -, |**와 같은 특별한 의미를 가지는 문자들을 사용하여 검색하거나 교체하기도 한다. 이러한 특수한 용도의 문자를 **메타문자meta character**라고 한다.

- 이제 정규표현식에서 첫 문자를 의미하는 문자인 ^에 대해서 살펴보자.

```
>>> 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
```

- 위의 결과를 살펴보면 검색시에 ^Life를 이용하여 txt3을 검색할 경우, 메타문자 ^에 의하여 문장의 첫 단어가 Life인 문장만을 검색하므로 **문장 중간에 Life가 있음에도 불구하고 None을 출력하는 것을 볼 수 있다.**



- 이제 다음의 문자열들에 대해서 위의 정규식을 적용해 보자.

```
>>> 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'>
```

- 표의 두 번째에 있는 \$ 기호를 사용하면 간단하게 마지막 단어가 life인 문자열을 찾을 수 있다.

9.9 메타 문자를 좀 더 상세하게 알아보자

- 우리는 필수적인 최소한의 기능을 중심으로 살펴보자. 정규식에서 사용되는 주요 **메타문자**의 의미를 요약하면 다음과 같다.
- 메타 문자 중에서 가장 중요한 문자는 점(.)과 별표(*)이다. 점은 어떤 문자가 와도 상관없고 별표는 몇 번 반복되어도 상관없다는 것을 의미한다.

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

9.9 메타 문자를 좀 더 상세하게 알아보자

- 점(.) 메타문자를 먼저 살펴보자.
- 이 문자는 임의의 문자 한 개를 의미하는 메타 문자이다.
- 예를 들어 ABA, ABBA, ABBBA라는 문자가 있을 경우 이 문자들 중에서 A..A라는 조건을 만족하는 문자는 스웨덴 출신의 4인조 밴드이름인 ABBA이다.



```
>>> 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
```

9.9 메타 문자를 좀 더 상세하게 알아보자

- 메타 문자 중에서 매우 강력한 문자가 바로 별표(*)이다. 이 문자는 직전에 있는 임의의 패턴과 1회 이상 반복되는 문자와 매치된다.

```
>>> 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*', 'CABBBBBBA')         # 'ABBBBBB'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(1, 7), match='ABBBBBB'>
```

9.9 메타 문자를 좀 더 상세하게 알아보자

- 다음으로 널리 사용되는 메타 문자 `?`에 대하여 알아보자. 이 문자는 문자열에서 찾기 위해 설정된 문자와 처음 매칭된 경우에 해당한다.

```
>>> 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'>

>>> re.search('AB?', 'ABBABB')
<re.Match object; span=(0, 2), match='AB'>
```

- 앞서 살펴본 `*`와 그 결과가 비슷해 보이지만 `CABBA`에서 조건에 맞는 문자가 `AB`이며 `CABBBBBA` 문자 역시 조건에 맞는 문자가 `AB`라는 점에서 큰 차이점이 있다.

9.9 메타 문자를 좀 더 상세하게 알아보자

- 메타문자 `+`는 직전에 있는 임의의 패턴을 1회 또는 그 이상의 수로 가급적 많이 반복하는 패턴에 대해서 매치되는 표현식이다.
- 따라서 `AB+`는 `A`와는 매치되지 않으며 `AB`, `ABB`, `ABBB`, `CABBA`와 같은 패턴의 문자열과 일치한다.

```
>>> 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+', 'CABBBBBBA')   # 'ABBBBBB'라는 문자열이 조건에 맞음
<_sre.SRE_Match object; span=(1, 7), match='ABBBBBB'>
```


9.9 메타 문자를 좀 더 상세하게 알아보자

- 다음으로 검색 명령인 `findall()`에 대하여 알아보도록 하자.
- 정규 표현식의 `findall()`을 사용하면 정규식을 만족하는 모든 문자열들을 추출할 수 있다.

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


9.10 정규식을 활용해서 멋지게 검색을 하자

- UN의 세계 인권 선언문은 <https://www.un.org/en/universal-declaration-human-rights/>를 방문하면 텍스트를 얻을 수 있다.
- 이 파일 역시 이 책의 github 주소로 접속하여 다운받을 수 있다. 이 텍스트를 d:\data 폴더에 UNDHR.txt로 저장하자(주의: 이 폴더 위치가 변경될 경우 코드도 수정해야 한다).
- 제일 먼저 (숫자) 형태의 항은 문장의 처음에 나타나야 한다.
- 다음으로 숫자가 와야 하는데, 이는 $[0-9]^+$ 또는 $\backslash d^+$ 라고 표시하면 된다.
- 이상과 같은 작업을 위해서는 모든 라인을 읽은 뒤에, `rstrip()`을 이용하여 오른쪽에 있는 모든 공백 문자를 제거하는 것이 필요하다.



9.10 정규식을 활용해서 멋지게 검색을 하자

```
import re

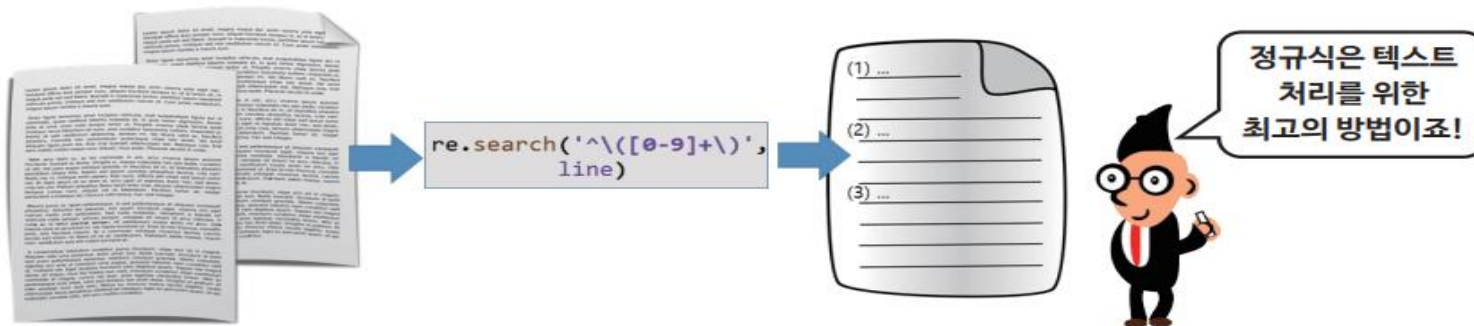
f = open('d:/data/UNDHR.txt')      # 현재 폴더에 있는 파일을 읽어온다.

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.

...



LAB⁹-5 학사 코드 추출하기에도전하자

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']
```

LAB⁹⁻⁵ 학사 코드 추출하기에 도전하자

```
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)
```

LAB⁹⁻⁵ 학사 코드 추출하기에 도전하자



도전문제 9.5

- 1) 위의 해답 코드의 문제점은 무엇일까? 과목 이름에 'Python Part1'과 같이 숫자가 포함되어 있는 경우 어떻게 될지 생각해 보자. 이런 문제를 해결할 수 있는 방법은 무엇일까?
- 2) 위의 코드를 수정하여 문자열의 길이가 3이고 모두 대문자인 수업요약 코드 COM, MAT, ENG를 추출하여라.
['COM', 'MAT', 'ENG']

LAB⁹⁻⁶ 이메일 주소를 분석해 보자

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']
```

LAB⁹⁻⁶ 이메일 주소를 분석해 보자

```
import re

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



도전문제 9.6

위의 코드를 수정하여 이메일 아이디와 도메인 주소를 구분하여 다음과 같이 출력하여라.

추출된 아이디 : abc , 도메인 : facebook.com

추출된 아이디 : bbc , 도메인 : google.com

LAB⁹⁻⁷ 패스워드 검사 프로그램을 만들자

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 임포트 하여 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
```


LAB⁹⁻⁷ 패스워드 검사 프로그램을 만들자

```
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
```

9.11 정규식에서 특정 문자를 대체하는 함수 : sub()

- 파이썬 정규식에는 검색하는 함수인 `search()`, `findall()`와 함께 다음과 같이 특정한 문자열을 다른 문자열로 대체하는 `sub()` 함수도 존재한다.

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

- 만일 특정한 수에 대해서 이 숫자를 외부에 비공개로 하기 위하여 별표문자를 이용하여 숨기고자 할 경우에는 다음과 같이 사용하면 된다.

```
>>> 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 * * *'
```

9.11 정규식에서 특정 문자를 대체하는 함수 : sub()

- 이제 이 행운의 번호를 해시함수를 사용해서 암호화 한 다음 공개를 해 보도록 하자.

```
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
```

- 위의 코드를 살펴보면 `hash_by_mult_and_modulo(m)` 함수가 매개변수 `m`을 정수로 만들어 23,435를 곱한 다음 973으로 나누어 나머지를 반환하는 기능이 있음을 알 수 있다.

LAB⁹⁻⁸ 트윗 메시지를 깔끔하게 정제하자

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

LAB⁹⁻⁸ 트윗 메시지를 깔끔하게 정제하자

```
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)
```

LAB⁹⁻⁸ 트윗 메시지를 깔끔하게 정제하자



잠깐 - 구글 검색에서도 쉽게 사용가능한 정규표현식

정규표현식 혹은 정규식은 특정한 규칙을 가진 문자열의 집합을 표현하는데 매우 유용한 형식언어이다. 이 정규표현식은 대부분의 프로그래밍 언어에서 별도의 라이브러리를 통해서 제공하고 있다. 또한 검색엔진에서도 널리 사용되는데 구글 검색시 "gray dog", "gray fox", "red dog", "red fox"를 키워드로 검색하고 싶을 경우 "(gray|red) (dog|fox)"와 같은 방법을 사용하면 동일한 검색이 이루어진다.

이와 같이 정규표현식을 사용하면 간략하면서도 강력한 문자열 검색과 조회, 대체까지 가능하다.



summary

핵심 정리



- 텍스트 정보는 사람이 읽고 이해할 수 있는 정보이다.
- 많은 양의 정보가 텍스트 데이터로 관리되고 유통된다.
- 파이썬은 텍스트 정보를 다룰 수 있는 다양한 함수를 제공하고 있다.
- 텍스트 데이터를 다루는 데에 정규식은 매우 유용한 도구가 된다.
- 정규식을 이용하여 문자열의 검색과 대치를 강력한 패턴 검색 형식으로 수행할 수 있다.

따라하며 배우는

파이썬과 데이터 과학



Questions?