학습목표

1. 정규표현식(re) 에 대한 이해 및 숙지

   정규표현식 : 특정한 패턴을 찾고, 치환, 제거 하는 기능

- 정규표현식
  - re 모듈을 사용!

```
import re

re.search(찾고싶은 문자열, 대상이 되는 문자열)

m = re.serach(r'[abc]aron','aabcdraron')

m.start() #: 검색된 문자열의 시작부분 위치 출력(1이라면, 실제 위치도 1)
m.end()  #: 검색된 문자열의 끝 부분 위치 출력(6이라면, 실제 위치는 5)
m.group()  #:  검색된 문자열 모두 출력
```

- regular expression
- 특정한 패턴과 일치하는 문자열를 '검색', '치환', '제거' 하는 기능을 지원
- 정규표현식의 도움없이 패턴을 찾는 작업(Rule 기반)은 불완전 하거나, 작업의 cost가 높음
- e.g) 이메일 형식 판별, 전화번호 형식 판별, 숫자로만 이루어진 문자열 등

- **raw string**
  - 문자열 앞에 r이 붙으면 해당 문자열이 구성된 그대로 문자열로 변환

In [3]:

```
a = r'abc\n'
print(a)
```

```
abc\n
```


기본 패턴

중요한 패턴

1. \w : alpha + numeric
2. \W : non (alpha or numeric)
3. \d : numeric
4. \D : non (numeric)
5. ' . ' : 모든 character (@!#$ 포함)
6. [abc] : a or b or c
7. [a.c] : a or . or c
8. [a.c^] : a or . or c or ^
9. [^abc] : not (a or b or c)
10. [a-c] : a부터 c까지
11. [0-9] : 0부터 9까지
12. [a-zA-Z0-9] : 모든 알파벳 및 문자

- a, X, 9 등등 문자 하나하나의 character들은 정확히 해당 문자와 일치
  - e.g) 패턴 test는 test 문자열과 일치
  - 대소문자의 경우 기본적으로 구별하나, 구별하지 않도록 설정 가능
- 몇몇 문자들에 대해서는 예외가 존재하는데, 이들은 틀별한 의미로 사용 됨
  - . ^ $ * + ? { } [ ] \ | ( )
- . (마침표) - 어떤 한개의 character와 일치 (newline(엔터) 제외)
- \w - 문자 character와 일치 [a-zA-Z0-9_]
- \s - 공백문자와 일치
- \t, \n, \r - tab, newline, return
- \d - 숫자 character와 일치 [0-9]
- ^ = 시작, $ = 끝 각각 문자열의 시작과 끝을 의미
- \가 붙으면 스페셜한 의미가 없어짐. 예를들어 \.는 .자체를 의미 \\는 \를 의미
- 자세한 내용은 링크 참조 https://docs.python.org/3/library/re.html (https://docs.python.org/3/library/re.html)

search method

- 첫번째로 패턴을 찾으면 match 객체를 반환
- 패턴을 찾지 못하면 None 반환

In [14]:

```
import re
```

In [51]:

```
m = re.search(r'[a-zA-Z0-9]','!#$@%adfa123abcdefdaf')
m.group()
```

Out[51]:

```
'a'
```

metacharacters (메타 캐릭터)

[ ] 대괄호는 or 를 의미함

[] 문자들의 범위를 나타내기 위해 사용

- [] 내부의 메타 캐릭터는 캐릭터 자체를 나타냄 -> **문자가 특정한 의미를 갖지 않고, 문자 그대로 받아들여야 함 (- 를 제외하고)**
- '-' 는 (~) 를 의미

- **[] 안에는 모두 그냥 문자 그대로 검색 대상**
- 예외) 1. [-] : - 는 문자 범위에 속하는 하나를 찾으라는 것

```
    2. [^  ] : 틸드로 시작한다는 건, 뒤 문자패턴이 아닌 것과 매칭하라는 것. ex) r'[^ab
c]aron' -> aron의 앞에 abc만 아니면 되는 것.
```

- e.g)
- [\w] : alpha or numeric
- [abck] : a or b or c or k
- [abc.^] : a or b or c or . or ^
- [a-d] : -와 함께 사용되면 해당 문자 사이의 범위에 속하는 문자 중 하나
- [0-9] : 모든 숫자
- [a-z] : 모든 소문자
- [A-Z] : 모든 대문자
- [a-zA-Z0-9] : 모든 알파벳 문자 및 숫자
- [^0-9] : ^가 맨 앞에 사용 되는 경우 해당 문자 패턴이 아닌 것과 매칭

In [50]:

```
re.search(r'[abc]aron','daron')
```

\ : 캐릭터 자체로 만듦

\d 와 \D 의 차이 : '숫자'와 '숫자가 아닌 문자'

1. 다른 문자와 함께 사용되어 특수한 의미를 지님
   - \d : 숫자를 [0-9]와 동일
   - \D : 숫자가 아닌 문자 [^0-9]와 동일
   - \s : 공백 문자(띄어쓰기, 탭, 엔터 등)
   - \S : 공백이 아닌 문자
   - \w : 알파벳대소문자, 숫자 [0-9a-zA-Z]와 동일
   - \W : non alpha-numeric 문자 [^0-9a-zA-Z]와 동일
2. 메타 캐릭터가 캐릭터 자체를 표현하도록 할 경우 사용
   - \. , \\

In [70]:

```
re.search('\Sand','apple and banana')
```


In [71]:

```
re.search('\sand','apple and banana')
```

Out[71]:

```
<re.Match object; span=(5, 9), match=' and'>
```

.

- 모든 문자를 의미

반복패턴

- 패턴 뒤에 위치하는 *, +, ?는 해당 패턴이 반복적으로 존재하는지 검사
  - '+' -> 1번 이상의 패턴이 발생
  - '*' -> 0번 이상의 패턴이 발생
  - '?' -> 0 혹은 1번의 패턴이 발생
- 반복을 패턴의 경우 greedy하게 검색 함, 즉 가능한 많은 부분이 매칭되도록 함
  - e.g) a[bcd]*b 패턴을 abcbdccb에서 검색하는 경우
    - ab, abcb, abcbdccb 전부 가능 하지만 최대한 많은 부분이 매칭된 abcbdccb가 검색된 패턴

In [52]:

```
re.search(r'pi+g','pg')
```

In [53]:

```
re.search(r'pi*g','pg')
```

Out[53]:

```
<re.Match object; span=(0, 2), match='pg'>
```

In [55]:

```
re.search(r'https?','httpk://www.')
```

Out[55]:

```
<re.Match object; span=(0, 4), match='http'>
```

^, $

- ^ 문자열의 맨 앞부터 일치하는 경우 검색
- $ 문자열의 맨 뒤부터 일치하는 경우 검색

In [56]:

```
re.search(r'b\w+a','cabana')
```

Out[56]:

```
<re.Match object; span=(2, 6), match='bana'>
```

In [57]:

```
re.search(r'^b\w+a','cabana')
```

In [59]:

```
re.search(r'b\w+a$','cabana')
```

Out[59]:

```
<re.Match object; span=(2, 6), match='bana'>
```

In [61]:

```
re.search(r'b\w+a$','cabanap')
```

grouping

- ()을 사용하여 그루핑
- 매칭 결과를 각 그룹별로 분리 가능
- 패턴 명시 할 때, 각 그룹을 괄호() 안에 넣어 분리하여 사용

```
# ID 와 domain을 그룹지어서 따로 보고 싶다. 그럼 ( ) 로 그룹지어주면 됨
m = re.search('(\w+)@(.+)','test@gmail.com')
print(m.group(1))
print(m.group(2))
print(m.group(0))
```

In [63]:

```
m = re.search('\w+@.+','test@gmail.com')
m
```

Out[63]:

```
<re.Match object; span=(0, 14), match='test@gmail.com'>
```

In [64]:

```
m = re.search('\w+@.+','test@gmail.com')
m.group()
```

Out[64]:

```
'test@gmail.com'
```


In [69]:

```
# ID 와 domain을 그룹지어서 따로 보고 싶다. 그럼 ( ) 로 그룹지어주면 됨

m = re.search('(\w+)@(.+)','test@gmail.com')
print(m.group(1))
print(m.group(2))
print(m.group(0))
```

```
test
gmail.com
test@gmail.com
```

{} : 3번반복만 or 최소 3번~최대5번만

- *, +, ?을 사용하여 반복적인 패턴을 찾는 것이 가능하나, **반복의 횟수 제한은 불가**
- 패턴뒤에 위치하는 중괄호{}에 숫자를 명시하면 해당 숫자 만큼의 반복인 경우에만 매칭
- {4} - 4번 반복
- {3,4} - 3 ~ 4번 반복

In [2]:

```
import re
# 만약 pi나오고 i가 3번만 나오고 g 나오는 걸 검출하고 싶다면?
# re.search(r'pi+g','piiiiig') 로는 검출할 수 없다.
re.search(r'pi+g','piiiiig')
```

Out[2]:

```
<re.Match object; span=(0, 7), match='piiiiig'>
```

In [6]:

```
# 따라서 중괄호 { 3 }  을 사용
re.search(r'pi{3}g','piiiig')
re.search(r'pi{3}g','piiig')
```

Out[6]:

```
<re.Match object; span=(0, 5), match='piiig'>
```

In [7]:

```
# 최소 3번에서 최대 5번 반복을 검출하고 싶다면? {3,5}
re.search(r'pi{3,5}g','piiiig')
```

Out[7]:

```
<re.Match object; span=(0, 6), match='piiiig'>
```

미니멈 매칭(non-greedy way) ----> *? +?

- 기본적으로 *, +, ?를 사용하면 greedy(맥시멈 매칭)하게 동작함
- *?, +?을 이용하여 해당 기능을 구현

In [8]:

```
re.search(r'<.*>','<html>hahaha<html>')
```

Out[8]:

```
<re.Match object; span=(0, 18), match='<html>hahaha<html>'>
```

In [9]:

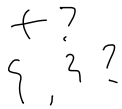
```
re.search(r'<.*?>','<html>hahaha<html>')
```

Out[9]:

```
<re.Match object; span=(0, 6), match='<html>'>
```

{}?

- {m,n}의 경우 m번 에서 n번 반복하나 greedy하게 동작
- {m,n}?로 사용하면 non-greedy하게 동작. 즉, 최소 m번만 매칭하면 만족



In [11]:

```
# 최소 3 ~ 최대 6이지만, 논그리디이기 때문에 {,}? 최소 기준으로 찾아줌
re.search(r'a{3,6}?','aaaaaa')
```

Out[11]:

```
<re.Match object; span=(0, 3), match='aaa'>
```

match : 시작부터 맞아야 한다

- search와 유사하나, 주어진 문자열의 시작부터 비교하여 패턴이 있는지 확인
- 시작부터 해당 패턴이 존재하지 않다면 None 반환

In [13]:

```
re.match(r'\d\d\d','my number is 123')
```

In [14]:

```
re.match(r'\d\d\d','123 number is my number')
```

Out[14]:

```
<re.Match object; span=(0, 3), match='123'>
```

findall

- search가 최초로 매칭되는 패턴만 반환한다면, findall은 매칭되는 전체의 패턴을 반환
- 매칭되는 모든 결과를 리스트 형태로 반환

In [26]:

```
re.findall(r'[\w]+','test@gmail.com test22@gmail.com test test icecream test33@gmail.com')
```

Out[26]:

```
['test',
 'gmail',
 'com',
 'test22',
 'gmail',
 'com',
 'test',
 'test',
 'icecream',
 'test33',
 'gmail',
 'com']
```

In [24]:

```
re.findall(r'\w+@[\w.]+','test@gmail.com test22@gmail.com test test icecream test33@gmail.com')
```

Out[24]:

```
['test@gmail.com', 'test22@gmail.com', 'test33@gmail.com']
```

In [28]:

```
re.findall(r'[\w]+@[\w.]+','test@gmail.com test22@gmail.com test test icecream test33@gmail.com')
```

Out[28]:

```
['test@gmail.com', 'test22@gmail.com', 'test33@gmail.com']
```

sub : substitute / 치환

- 주어진 문자열에서 일치하는 모든 패턴을 replace
- 그 결과를 문자열로 다시 반환함
- 두번째 인자는 특정 문자열이 될 수도 있고, 함수가 될 수 도 있음
- **count가 0인 경우는 전체를, 1이상이면 해당 숫자만큼 치환 됨**

In [30]:

```
re.sub(r'[\w]+@[\w.]+','great','test@gmail.com test22@gmail.com test test icecream test33@gmail.com')
```

Out[30]:

```
'great great test test icecream great'
```

In [31]:

```
re.sub(r'[\w]+@[\w.]+','great','test@gmail.com test22@gmail.com test test icecre
am test33@gmail.com',count = 1)
```

Out[31]:

```
'great test22@gmail.com test test icecream test33@gmail.com'
```

compile

- 동일한 정규표현식을 매번 다시 쓰기 번거로움을 해결
- compile로 해당표현식을 re.RegexObject 객체로 저장하여 사용가능

In [36]:

```
email_reg = re.compile(r'[\w]+@[\w.]+') #이렇게 re.compile(정규표현식) 을 해놓으면

re.search(r'정규표현식','서칭 대상 문자열') #이렇게 하지 않아도 된다

#re패키지를 새로 쓰지 않아도 됨. 이미 컴파일 과정에서 녹였기 때문에 re.compile('정규표현식')
'컴파일명'.search('서칭 대상 문자열')  #이렇게 하면 됨
```

```
---------------------------------------------------------------------------
-------
AttributeError                            Traceback (most recent cal
l last)
<ipython-input-36-17c1e97dfbfa> in <module>
      4
      5 #re패키지를 새로 쓰지 않아도 됨. 이미 컴파일 과정에서 녹였기 때문에 re.compi
le('정규표현식')
----> 6 '컴파일명'.search('서칭 대상 문자열')  #이렇게 하면 됨

AttributeError: 'str' object has no attribute 'search'
```

In [38]:

```
email_reg = re.compile(r'[\w]+@[\w.]+')

email_reg.search('test@gmail.com')
email_reg.findall('test@gmail.com')
```

Out[38]:

```
['test@gmail.com']
```

연습문제

- 아래 뉴스에서 이메일 주소를 추출해 보세요
- 다음중 올바른 (http, https) 웹페이지만 찾으시오

In [39]:

```
import requests
from bs4 import BeautifulSoup
# 위의 두 모듈이 없는 경우에는 pip install requests bs4 실행

def get_news_content(url):
    response = requests.get(url)
    content = response.text

    soup = BeautifulSoup(content, 'html5lib')

    div = soup.find('div', attrs = {'id' : 'harmonyContainer'})

    content = ''
    for paragraph in div.find_all('p'):
        content += paragraph.get_text()

    return content

news1 = get_news_content('https://news.v.daum.net/v/20190617073049838')
print(news1)
```

```
(로스앤젤레스=연합뉴스) 옥철 특파원 = 팀 쿡 애플 최고경영자(CEO)가 16일(현지시간) 실리
콘밸리 앞마당 격인 미국 서부 명문 스탠퍼드대학 학위수여식에서 테크기업들을 향해 쓴소리를 쏟
아냈다.쿡은 이날 연설에서 실리콘밸리 테크기업들은 자신들이 만든 혼란에 대한 책임을 질 필요
가 있다고 경고했다.근래 IT 업계의 가장 큰 이슈인 개인정보 침해, 사생활 보호 문제를 콕 집
어 라이벌인 구글, 페이스북 등 IT 공룡을 겨냥한 발언이라는 해석이 나왔다.쿡은 "최근 실리콘
밸리 산업은 고귀한 혁신과는 점점 더 거리가 멀어지는 것으로 알려져 있다. 책임을 받아들이지
않고도 신뢰를 얻을 수 있다는 그런 믿음 말이다"라고 꼬집었다.개인정보 유출 사건으로 미 의회
청문회에 줄줄이 불려 나간 경쟁사 CEO들을 향해 일침을 가한 것으로 보인다.그는 또 실리콘밸
리에서 희대의 사기극을 연출한 바이오벤처 스타트업 테라노스(Theranos)를 직격했다.쿡은
"피 한 방울로 거짓된 기적을 만들 수 있다고 믿었느냐"면서 "이런 식으로 혼돈의 공장을 만든
다면 그 책임에서 절대 벗어날 수 없다"라고 비난했다.테라노스는 손가락 끝을 찔러 극미량의 혈
액 샘플만 있으면 각종 의학정보 분석은 물론 거의 모든 질병 진단이 가능한 바이오헬스 기술을
개발했다고 속여 월가 큰손들로부터 거액의 투자를 유치했다가 해당 기술이 사기인 것으로 드러나
청산한 기업이다.쿡은 애플의 경우 프라이버시(사생활) 보호에 초점을 맞춘 새로운 제품 기능들
로 경쟁사들에 맞서고 있다며 자사의 데이터 보호 정책을 은근히 홍보하기도 했다.oakchul@yn
a.co.kr
```

In [43]:

```
import re

re.search(r'[\w-]+@[\w.]+',news1)
```

Out[43]:

```
<re.Match object; span=(774, 791), match='oakchul@yna.co.kr'>
```

In [60]:

```
webs = ['http://www.test.co.kr',
        'https://www.test1.com',
        'http://www.test.com',
        'ftp://www.test.com',
        'http:://www.test.com',
       'htp://www.test.com',
       'http://www.google.com',
       'https://www.homepage.com.']
web = ' '.join(webs)
web
```

Out[60]:

```
'http://www.test.co.kr https://www.test1.com http://www.test.com ft
p://www.test.com http:://www.test.com htp://www.test.com http://www.
google.com https://www.homepage.com.'
```

In [61]:

```
# 오답!
# 마지막이 '.'으로 끝나면 안됨
re.findall(r'https*://[\w.]+',web)
```

Out[61]:

```
['http://www.test.co.kr',
 'https://www.test1.com',
 'http://www.test.com',
 'http://www.google.com',
 'https://www.homepage.com.']
```

In [82]:

```
# 끝이 numeric이나 alpha로 끝나야 함
#re.findall(r'https*://[\w.]+\w+$',web)  ->  이렇게 하면, web(공백으로 이어진 문자열)은 제
대로 분할 검색 안된다.
for i in webs:
    if re.search(r'https*://[\w.]+\w+$',i):
        print(i)
```

```
http://www.test.co.kr
https://www.test1.com
http://www.test.com
http://www.google.com
```

In [83]:

```
# 선생님 해답
reg_name = re.compile(r'https*://[\w.]+\w+$')

# web리스트의 항목별로 올바른지 아닌지 True, False로 반환하는 리스트 만들어 볼 것임
# 각 리스트의 항목을 True or False 로 변환시키기 위해, map(key,대상변수) 과 lambda를 사용
# lambda는 입력을 받아, 판별식(true or false)로 출력해내도록 함!
list(map(lambda x:reg_name.search(x)!=None,webs))
```

Out[83]:

```
[True, True, True, False, False, False, True, False]
```


In [ ]:

```

```