정규 표현식

1. 기호

2. 메소드

3. 문자열 바꾸기

■ 정규 표현식 (Regular Expressions)

- 특정한 규칙을 가진 문자열의 집합을 표현할때 사용하는 언어
- 여러 텍스트 편집기와 프로그래밍 언어에서 지원
- 정규 표현식이 유용하게 사용되는 경우
  - 영어와 숫자로만 작성되어야 하는 회원 아이디 검사
  - 한 글자가 특정 횟수 이상 반복되는 경우 검사
  - 주민등록번호, 이메일, 휴대폰 번호 등의 형식 검사
  - 비밀번호 생성 시 특수문자 포함 여부 검사
- 기본 사용 방법

```
# 정규 표현식 사용을 위한 모듈 import
import re

# 정규 표현식의 규칙 지정
pattern = re.compile('[a]')

# 정규 표현식 규칙에 문자열을 입력하여 확인
print(pattern.search('a'))
```

■ 정규 표현식 (Regular Expressions)

● 정규식에서 사용되는 기호

```
[ ] { } . * + ? ^ $ \ | ( )
```

● 정규식 관련 메소드

메소드	설명
match()	문자열의 **처음부터 정규식과 매치**되는지 확인 매치될 때 match 객체 반환 매치되지 않으면 None 반환
search()	문자열 **전체를 검색하여 정규식과 매치**되는지 확인 매치될 때 match 객체 반환 매치되지 않으면 None 반환
findall()	정규식과 매치되는 모든 문자열을 List로 반환
finditer()	정규식과 매치되는 모든 문자열을 iterator 객체로 반환

■ 정규 표현식 (Regular Expressions)

● [문자]

- 대괄호([ ]) 사이에 문자를 입력
- "or" 의미로 적용
- 두 문자 사이에 하이픈(-)을 사용하면 범위(From - To)를 의미

```
[abc]    # a 또는 b 또는 c
  a      # a가 있으므로 매치
  black  # b 또는 a가 있으므로 매치
  duke   # 일치하는 문자가 없음

[a-z]    # a 부터 z 까지 모든 알파벳 문자
[0-9]    # 0 부터 9까지 모든 숫자
[가-힣]  # 모든 한글
```

■ 정규 표현식 (Regular Expressions)

● [문자]

- 대괄호([ ]) 의 문자로 ^ 가 사용되면 부정(not)을 의미

```
[^abc]          # a 또는 b 또는 c 가 아닌 문자
[^0-9]          # 0 부터 9까지 모든 숫자를 제외한 문자
[^0-3a-dA-D]    # 0 ~ 3, a ~ d, A ~ D 를 제외한 문자
```

- 자주 사용되는 표현식은 별도의 특수문자로 표현 가능

문자	설명
₩d	모든 숫자, [0-9]와 동일
₩D	숫자가 아닌 것, [^0-9]와 동일
₩s	whitespace, [₩t₩n₩r₩f₩v]와 동일
₩S	whitespace가 아닌 것, [^ ₩t₩n₩r₩f₩v]와 동일
₩w	문자 + 숫자 + underscore, [a-zA-Z0-9_...]
₩W	문자 + 숫자가 아닌 것, [^a-zA-Z0-9_...]와 동일

■ 정규 표현식 (Regular Expressions)

● [문자]

- 적용 예

```
import re
pattern = re.compile('[a]')

text = 'a'
print(pattern.search(text))
# True - <re.Match object; span=(0, 1), match='a'>

text = 'ba'
print(pattern.search(text))
# True - <re.Match object; span=(1, 2), match='a'>

text = 'cd'
print(pattern.search(text))
# False - None
```

■ 연습문제

● 사용자 아이디의 유효성 검사하기

- 조건 1) 아이디의 시작 문자는 영어이어야 한다.
- 조건 2) 영어와 숫자로만 작성되어야 한다.

* 힌트) 시작 문자 검사는 match(), 그 외는 search() 사용

```
import re

pattern1 = re.compile('(  ①  )')
pattern2 = re.compile('(  ②  )')

user_id = '(  ③  )'

result1 = pattern1.match(user_id)
result2 = pattern2.search(user_id)

if (  ④  ):
    print('%s <- 가입 가능' % user_id)
else:
    print('%s <- 가입 불가' % user_id)
```

```
12abc가 <- 가입 불가
```

```
12abc <- 가입 불가
```

```
가12abc <- 가입 불가
```

```
a12abc <- 가입 가능
```

■ 정규 표현식 (Regular Expressions)

● Dot(.)

- 줄바꿈 문자(₩n)를 제외한 모든 문자와 매치

```
a.b                 # a + 모든 문자 + b
  abc               # None
  aab               # match
  abbbbbbb          # match
  a가b              # match
  가나다a라b마바사  # match
```

- 대괄호([ ]) 내에 Dot(.)를 사용하면 모든 문자가 아닌 Dot(.) 자체를 의미

```
a[.]b
  a.b  # match
  aab  # None
```

■ 정규 표현식 (Regular Expressions)

● Asterisk(*)

- 바로 앞의 문자가 0 ~ N 번 등장하는 경우 매치

```
a*b                    # a 0개 ~ N개 + b
  b                    # match
  ab                   # match
  aaaaaaaaaabbbbbbb    # match
  acb                  # None
```

● Plus(+)

- 바로 앞의 문자가 1 ~ N 번 등장하는 경우 매치

```
a+b                    # a 1개 ~ N개 + b
  b                    # None
  ab                   # match
  aaaaaaaaaabbbbbbb    # match
  acb                  # None
```

■ 정규 표현식 (Regular Expressions)

● {m}

- 해당 문자가 m번 등장하는 경우 매치

```
a{2}b  # aab
c{3}z  # cccz
```

● {m,n}

- 해당 문자가 m번 ~ n번 등장하는 경우 매치 (띄어쓰기 X)

```
a{1,3}b
  ab      # match
  aab     # match
  aaab    # match
  aaaab   # None
```

● ?

- 해당 문자가 0번 또는 1번 등장하는 경우 매치 ({0,1} 과 같음)

```
a?b
b    # match
ab   # match
aab  # None
```

■ 연습문제

● 주민등록번호 유효성 검사하기

- 조건 1) 앞자리 숫자는 6자리, 뒷자리 숫자는 7자리로 작성되어야 한다.
- 조건 2) 뒷자리의 시작숫자는 1 ~ 4 이어야 한다.

```
import re

pattern = re.compile('(   ①   )')

text = '791111-5234567'

print(text, end=' <- ')

result = pattern.search(text)
if result:
    print('정상')
else:
    print('올바르지 못한 주민등록번호')
```

```
791111-1234567 <- 정상
791111-123456 <- 올바르지 못한 주민등록번호
791111-5234567 <- 올바르지 못한 주민등록번호
```

■ 연습문제

● 은행 계좌번호 형식 검사하기

- 조건 1) 숫자 3자리-숫자2자리-숫자6자리
- 조건 2) 첫번째 3자리 숫자의 시작은 반드시 1 이어야 한다.
- 조건 3) 두번째 2자리 숫자의 시작은 반드시 1 이어야 한다.

```
import re

pattern = re.compile('(  ①  )')

text = '123-12-123456'

print(text, end=' <- ')

result = pattern.search(text)
if result:
    print('확인되었습니다.')
else:
    print('계좌번호 다시 확인해주세요.')
```

```
123-12-123456 <- 확인되었습니다.
```

■ 정규 표현식 (Regular Expressions)

● ^

- 시작

```
import re

pattern = re.compile('^abc')

print(pattern.search('abc')) # 'a' 로 시작하고 bc
print(pattern.search('ab'))
print(pattern.search('가bc'))
```

- 부정

```
import re

pattern = re.compile('[^abc]')

print(pattern.search('a'))
print(pattern.search('b'))
print(pattern.search('c'))
print(pattern.search('가'))  # 'a' 또는 'b' 또는 'c' 아님
```

■ 정규 표현식 (Regular Expressions)

● $

- 종료

```
import re

pattern = re.compile('\w+[.]$')

print(pattern.search('Hello.'))  # 문자 1개 이상 + 마지막 .
print(pattern.search('Hi~'))
```

● |

- 두 패턴 중 하나 (or)

```
import re

pattern = re.compile('^Hello|.*Bye.$|.*Good.*')

print(pattern.search('Hello~'))         # 시작 Hello
print(pattern.search('Bye.'))           # 마지막 Bye.
print(pattern.search('Hi~ Good End'))   # Good 포함
```

■ 연습문제

● 문자열이 형식에 맞는지 검사하기

- 조건 1) '안녕하세요' 문자로 시작
- 조건 2) 내용 중 '열심히' 문자 포함
- 조건 3) '감사합니다.' 문자로 종료

* 힌트) 줄바꿈 문자(₩n)는 . 에 해당되지 않음

```
import re

pattern = re.compile('(  ①  )')

text = '''
안녕하세요. 저는 ooo에서 태어났고...
열심히 하겠습니다. 감사합니다.
'''

# 코드 작성

print(pattern.search(text))
```

■ 정규 표현식 (Regular Expressions)

● ( )

- 정규식 그룹화 (index)

```
import re

pattern = re.compile('(\d{3})-(\d{4})-(\d{4})')

result = pattern.search('010-8478-8181')
print(result.group())
print(result.group(0))
print(result.group(1))
print(result.group(2))
print(result.group(3))
```

```
010-8478-8181
010-8478-8181
010
8478
8181
```

■ 정규 표현식 (Regular Expressions)

● ( )

- 정규식 그룹화 (name)

```
import re

pattern = re.compile('(?P<a>\d{3})-(?P<b>\d{4})-(?P<c>\d{4})')

result = pattern.search('010-8478-8181')
print(result.groupdict())  # Dictionary 형태로 접근 가능
print(result.group())
print(result.group(0))
print(result.group('a'))   # result.group(1)
print(result.group('b'))   # result.group(2)
print(result.group('c'))   # result.group(3)
```

```
{'a': '010', 'b': '8478', 'c': '8181'}
010-8478-8181
010-8478-8181
010
8478
8181
```

■ 연습문제

● 문자열의 내용 중 이름과 전화번호 찾아내기

```
import re

pattern = re.compile(
    '(  ①  ) '
)

text = '''
사용자 정보, 이름: 꼬렙, 전화번호 : 1234, 이메일 : seorab@naver.com
'''

# 코드 작성
```

```
이름: 꼬렙
전화번호 : 1234
```

■ 연습문제

● 문자열의 내용 중 이메일 찾아내기

```
import re

pattern = re.compile(
    '(  ①  ) '
)

text = '''
사용자 정보, 이름: 꼬렙, 전화번호 : 1234, 이메일 : seorab@naver.com
'''

# 코드 작성
```

```
이메일 : seorab@naver.com
```

■ 정규 표현식 (Regular Expressions)

● findall()

- 정규식과 매치되는 모든 문자열을 List로 반환

```
pattern = re.compile('[a-z]+')

result = pattern.findall('Life is too short, You need Python')
print(result)
```

```
['ife', 'is', 'too', 'short', 'ou', 'need', 'ython']
```

- 모든 숫자 List로 반환

```
pattern = re.compile('[^-]')  # 하이픈(-)이 아닌 1개 문자
result = pattern.findall('010-8478-8181')
print(result)

pattern = re.compile('[^-]+')  # 하이픈(-)이 아닌 1개 이상의 문자
result = pattern.findall('010-8478-8181')
print(result)
```

```
['0', '1', '0', '8', '4', '7', '8', '8', '1', '8', '1']
['010', '8478', '8181']
```

■ 정규 표현식 (Regular Expressions)

● findall()

- 에러 번호 찾아내기

```
import re
pattern = re.compile('에러\s*(\d+)')
result = pattern.findall(
    '에러 404 : 페이지 없음\n 에러 500 : 서버 오류')
print(result)
```

```
['404', '500']
```

- 이름 찾아내기

```
import re
pattern = re.compile('이름\s*:\s*(\w+)')
result = pattern.findall(
    '이름:홍길동, 이름     : 김길동, 이름 :          최길동')
print(result)
```

```
['홍길동', '김길동', '최길동']
```

■ 정규 표현식 (Regular Expressions)

● findall()

- 특정 문자열 등장 횟수 확인하기

```
with open('harry_potter.txt') as file:
    text = file.read()

import re

pattern = re.compile('(Harry Potter)', re.I)  # 대소문자 무시
len(pattern.findall(text))
```

```
28
Harry Potter Harry Potter Harry Potter Harry Potter Harry Potter Harry Potter
Harry Potter Harry Potter Harry Potter Harry Potter Harry Potter Harry Potter
Harry Potter Harry Potter Harry Potter Harry Potter Harry Potter Harry Potter
Harry Potter Harry Potter Harry Potter Harry Potter HARRY POTTER
Harry Potter Harry Potter Harry Potter Harry Potter Harry Potter Harry Potter
Harry Potter Harry Potter Harry Potter
```

■ 연습문제

● 여러 기호를 제외하고 검색어만 확인하기

- 조건 1) 검색어는 #기호 다음부터 시작
- 조건 2) 여러 개의 검색어가 입력될 때는 comma(,)와 공백으로 구분
- 조건 3) 검색어 구분자 중 공백은 한개 이상이 입력될 수 있음

```
import re

pattern = re.compile('(  ①  )')

text = '#서울, #강남, #맛집, #주꾸미'

res = pattern.findall(text)
print(res)
```

```
['서울', '강남', '맛집', '주꾸미']
```

■ 연습문제

● 이름 등장 횟수 확인하기

```
import requests
import re

res = requests.get(
    'http://ggoreb.com/quiz/운수좋은날.txt')
res.encoding = None
text = res.text

# 코드 작성
```

```
49
['김 첨지', '김 첨지', '김 첨지', '김 첨지', '김  첨지', '김    첨지', '김 첨 지', '김 첨지', '김
첨지', '김 첨   지', '김첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김
첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨
지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨
지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨지', '김 첨
지', '김 첨지', '김 첨지', '김 첨지', '김 첨지']
```

■ 정규 표현식 (Regular Expressions)

● finditer()

- 정규식과 매치되는 모든 문자열을 iterator 객체로 반환

```
import re

pattern = re.compile('[a-z]+')

result = pattern.finditer(
    'Life is too short, You need Python')

for r in result:
    print(r, r.group())
```

```
<re.Match object; span=(1, 4), match='ife'> ife
<re.Match object; span=(5, 7), match='is'> is
<re.Match object; span=(8, 11), match='too'> too
<re.Match object; span=(12, 17), match='short'> short
<re.Match object; span=(20, 22), match='ou'> ou
<re.Match object; span=(23, 27), match='need'> need
<re.Match object; span=(29, 34), match='ython'> ython
```

■ 정규 표현식 (Regular Expressions)

● finditer()

- com 또는 net으로 끝나는 이메일 주소 찾기 (전체 표현)

```
import re
                                    ①              ②
pattern = re.compile('\w+@\w+[.]com|\w+@\w+[.]net')

text = '''
주로 사용하는 이메일 주소는 a@ggoreb.com 이고,
가끔 사용하는 이메일 주소는 b@ggoreb.info 입니다.
이번에 새로 만든 c@ggoreb.net도 있습니다.
'''

result = pattern.findall(text)
for r in result:
    print(r)
```

```
a@ggoreb.com
c@ggoreb.net
```

```
result = pattern.finditer(text)
for r in result:
    print(r.group())
```

```
a@ggoreb.com
c@ggoreb.net
```

■ 정규 표현식 (Regular Expressions)

● finditer()

- com 또는 net으로 끝나는 이메일 주소 찾기 (일부분 표현)

```
import re
                                    ①        ②
pattern = re.compile('\w+@\w+[.]com|net')

text = '''
주로 사용하는 이메일 주소는 a@ggoreb.com 이고,
가끔 사용하는 이메일 주소는 b@ggoreb.info 입니다.
이번에 새로 만든 c@ggoreb.net도 있습니다.
'''

result = pattern.findall(text)
for r in result:
    print(r)
```

```
a@ggoreb.com
net
```

```
result = pattern.finditer(text)
for r in result:
    print(r.group())
```

```
a@ggoreb.com
net
```

■ 정규 표현식 (Regular Expressions)

● finditer()

- com 또는 net으로 끝나는 이메일 주소 찾기 (그룹화)

```
import re

                                        ①   ②
pattern = re.compile('\w+@\w+[.](com|net)')

text = '''
주로 사용하는 이메일 주소는 a@ggoreb.com 이고,
가끔 사용하는 이메일 주소는 b@ggoreb.info 입니다.
이번에 새로 만든 c@ggoreb.net도 있습니다.
'''

result = pattern.findall(text)
for r in result:
    print(r)
```

```
com
net
```

```
result = pattern.finditer(text)
for r in result:
    print(r.group())
```

```
a@ggoreb.com
c@ggoreb.net
```

■ 매칭되는 문자열 바꾸기 (sub)

● 특정 문자열을 찾은 후 원하는 문자열로 변경

```
import re
pattern = re.compile('Unix|Linux')
text = 'GNU is Not Unix, Unix is not Linux'
change = 'OS'

result = pattern.sub(change, text)
print(result)
```

```
GNU is Not OS, OS is not OS
```

● 변경 횟수 지정

```
import re
pattern = re.compile('Unix|Linux')
text = 'GNU is Not Unix, Unix is not Linux'
change = 'OS'

result = pattern.sub(change, text, 2)
print(result)
```

```
GNU is Not OS, OS is not Linux
```

■ 매칭되는 문자열 바꾸기 (sub)

● 특정 문자열을 찾은 후 전체 변경

```
import re
pattern = re.compile('\w+@\w+[.]\w+')
text = '1번 a@ggoreb.com, 2번 주소 b@ggoreb.info 3번 - c@ggoreb.net'
change = 'ggoreb.co.kr'

result = pattern.sub(change, text)
print(result)
```

```
1번 ggoreb.co.kr, 2번 주소 ggoreb.co.kr 3번 - ggoreb.co.kr
```

● 그룹으로 지정된 부분은 남겨놓고 나머지 문자열을 변경

```
import re
pattern = re.compile('(\w+@)\w+[.]\w+')
text = '1번 a@ggoreb.com, 2번 주소 b@ggoreb.info 3번 - c@ggoreb.net'
change = '\g<1>ggoreb.co.kr'

result = pattern.sub(change, text)
print(result)
```

```
1번 a@ggoreb.co.kr, 2번 주소 b@ggoreb.co.kr 3번 - c@ggoreb.co.kr
```

■ 매칭되는 문자열 바꾸기 (sub)

● 특정 문자열을 찾은 후 전체 변경

```
import re
pattern = re.compile('\d{6}-\d{7}')
text = '791111-1234567'
change = '*******'

result = pattern.sub(change, text)
print(result)
```

```
*******
```

● 그룹으로 지정된 부분은 남겨놓고 나머지 문자열을 변경

```
import re
pattern = re.compile('(\d{6}-)\d{7}')
text = '791111-1234567'
change = '\g<1>*******'

result = pattern.sub(change, text)
print(result)
```

```
791111-*******
```

■ 연습문제

● 전화번호 중간자리 숫자 마킹 처리

```
import re
pattern = re.compile('(  ①  )')
text = '010-8478-8181'
change = '(  ②  )'

res = pattern.sub(change, text)
print(res)
```

```
010-****-8181
```

■ 연습문제

● 이메일의 도메인 앞부분 마킹 처리

```
import re
pattern = re.compile('(  ①  )')
user_list = ['ggoreb@naver.com', 'seorab@gmail.com',
'human@ggoreb.net']
change = '(  ②  )'

for user in user_list:
    res = pattern.sub(change, user)
    print(res)
```

```
ggoreb@*****com
seorab@*****com
human@*****net
```