

정규 표현식

- 기호
- 메소드
- 문자열 바꾸기 - sub

■ 정규 표현식 (Regular Expressions)

- 특정한 규칙을 가진 문자열의 집합을 표현할 때 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

```
import re # 정규 표현식 사용을 위한 모듈 import  
pattern = re.compile('[a]')  
print(pattern.match('a'))
```

■ 정규 표현식 (Regular Expressions)

● 정규식에서 사용되는 메타 문자 (특별한 용도로 사용되는 문자)

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

● 문자 클래스 (문자 매치)

- 대괄호([]) 사이에 문자 또는 문자열을 입력
- 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

[0-3a-dA-D] # 0 ~ 3, a ~ d, A ~ D

■ 정규 표현식 (Regular Expressions)

● 문자 클래스 (문자 매치)

- 대괄호([]) 내에 ^ 문자를 사용한 경우 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, [a-zA-Z0-9_\-]와 동일
\W	문자 + 숫자가 아닌 것, [^a-zA-Z0-9_\-]와 동일

소문자와 대문자는 서로 반대의 개념

■ 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
아가b # 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 ~ N번 등장하는 경우 매치

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

● Plus(+) : 바로 앞의 문자가 1 ~ N번 등장하는 경우 매치

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

■ 정규 표현식 (Regular Expressions)

- $\{m\}$: 해당 문자가 m 번 등장하는 경우 매치

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

- $\{m, n\}$: 해당 문자가 m 번 ~ n 번 등장하는 경우 매치

```
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 # aab  
b # match  
ab # match  
aab # None
```

연습문제

주민등록번호 형식을 확인하는 정규식 만들기

```
import re

text = '791111-1234567'
regex = '코드 작성'
pattern = re.compile(regex)
res = pattern.match(text)

print(res)
```

○ 조건

- 앞자리 숫자 6개 / 뒷자리 숫자 7개
- 뒷자리의 시작 숫자는 1 ~ 4

791111-1234567 입력

```
<_sre.SRE_Match object; span=(0, 14),  
match='791111-1234567'>
```

791111-2234567 입력

```
<_sre.SRE_Match object; span=(0, 14),  
match='791111-2234567'>
```

791111-4234567 입력

```
<_sre.SRE_Match object; span=(0, 14),  
match='791111-4234567'>
```

791111-5234567 입력

None

연습문제

은행 계좌번호 형식이 맞는지 확인하는 정규식 만들기

```
import re

text = '123-12-123456'
regex = '코드 작성'
pattern = re.compile(regex)
res = pattern.match(text)

print(res)
```

○ 조건

- 숫자 3자리-숫자 2자리-숫자6자리
- 첫번째 3자리 숫자의 시작은 반드시 1
- 두번째 2자리 숫자의 시작은 반드시 1

123-12-123456 입력

```
<_sre.SRE_Match object; span=(0, 13),  
match='123-12-123456'>
```

123-22-123456 입력

None

123-121-123456 입력

None

123-12-654321 입력

```
<_sre.SRE_Match object; span=(0, 13),  
match='123-12-654321'>
```

■ 정규 표현식 (Regular Expressions)

● ^

– 시작

```
pat = re.compile('^abc')
print(pat.match('abc')) # 'a'로 시작하고 bc
print(pat.match('ab'))
print(pat.match('가bc'))
```

```
<re.Match object; span=(0, 3), match='abc'>
None
None
```

– 부정

```
pat = re.compile('[^abc]')
print(pat.match('a'))
print(pat.match('b'))
print(pat.match('c'))
print(pat.match('가')) # a 또는 b 또는 c 아님
```

```
None
None
None
<re.Match object; span=(0, 1), match='가'>
```

■ 정규 표현식 (Regular Expressions)

● \$: 종료

```
pattern = re.compile('₩w+[$]')  
print(pattern.match('Hello.'))  
print(pattern.match('Hi~'))
```

```
<_sre.SRE_Match object; span=(0, 6), match='Hello.'>  
None
```

● | : 두 패턴 중 하나 (or)

```
pattern = re.compile('^Hello|.*Bye.$|.*Good.*')  
print(pattern.match('Hello~'))  
print(pattern.match('Bye.'))  
print(pattern.match('Hi~ Good End'))
```

```
<_sre.SRE_Match object; span=(0, 5), match='Hello'>  
<_sre.SRE_Match object; span=(0, 4), match='Bye.'>  
<_sre.SRE_Match object; span=(0, 12), match='Hi~ Good End'>
```

연습문제

문자열의 내용이 특정 형식에 맞는지 확인하는 정규식 만들기

```
import re

text = '안녕하세요. 저는 OOO에서 태어났고... 열심히 하겠습니다. 감사합니다.'
regex = '코드 작성'
pattern = re.compile(regex)
res = pattern.match(text)

print(res)
```

- 조건
 - '안녕하세요' 라는 문자열로 시작
 - 내용 중 '열심히' 라는 문자열 반드시 포함
 - '감사합니다.' 라는 문자열로 종료

■ 정규 표현식 (Regular Expressions)

● ()

– 정규식 그룹화 – index

```
regex = '\d{3}-\d{4}-\d{4}'  
pattern = re.compile(regex)  
  
pattern.match('010-8478-8181')
```

<_sre.SRE_Match object; span=(0, 13), match='010-8478-8181'>

```
regex = '(\d{3})-(\d{4})-(\d{4})'  
pattern = re.compile(regex)  
  
result = pattern.match('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)

● (?P<그룹명>)

– 정규식 그룹화 – name

```
regex = '\d{3}-\d{4}-\d{4}'  
pattern = re.compile(regex)  
  
pattern.match('010-8478-8181')
```

<_sre.SRE_Match object; span=(0, 13), match='010-8478-8181'>

```
regex = '(?P<a>\d{3})-(?P<b>\d{4})-(?P<c>\d{4})'  
pattern = re.compile(regex)  
  
result = pattern.match('010-8478-8181')  
print(result.groupdict())  
print(result.group())  
print(result.group('a'))  
print(result.group('b'))  
print(result.group('c'))
```

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

010-8478-8181

010

8478

8181

연습문제

문자열의 내용 중 이름과 전화번호를 찾아내는 정규식 만들기

```
import re

text = '이름:꼬렙, 전화번호:1234'
regex = '코드 작성'
pattern = re.compile(regex)
res = pattern.match(text)

print(res.group(1))
print(res.group(2))
```

이름:꼬렙, 전화번호:1234 입력

꼬렙
1234

이름:파이썬, 전화번호:2324 입력

파이썬
2324

■ 정규 표현식 (Regular Expressions)

● 정규식 관련 메소드

메소드	설명
match()	문자열의 처음부터 정규식과 매치되는지 확인 매치될 때 match 객체 반환 매치되지 않으면 None 반환
search()	문자열 전체를 검색하여 정규식과 매치되는 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)

● match()

- 문자열의 처음부터 정규식과 매치되는지 확인

```
>>> import re
>>> p = re.compile('[a-z]+')

>>> print(p.match('python'))
<_sre.SRE_Match object; span=(0, 6), match='python'>

>>> print(p.match('1python'))
None

>>> print(p.match('1python'))
None

>>> print(p.match('가python'))
None
```

■ 정규 표현식 (Regular Expressions)

● match()

– 영어 소문자 2자로 시작되고 숫자 6자리 검색

```
regex = '^[a-z]{2}\d{6}$'
pattern = re.compile(regex)

print(pattern.match('12345678'))
print(pattern.match('ab12345678'))
print(pattern.match('ab123456'))
```

None

None

<_sre.SRE_Match object; span=(0, 8), match='ab123456'>

– 특수문자 사용 확인

```
regex = '[a-zA-Z0-9]*[^a-zA-Z0-9]+[a-zA-Z0-9]*'
pattern = re.compile(regex)

print(pattern.match('가123456'))
print(pattern.match('123&456'))
print(pattern.match('123456가'))
```

<_sre.SRE_Match object; span=(0, 7), match='가123456'>

<_sre.SRE_Match object; span=(0, 7), match='123&456'>

<_sre.SRE_Match object; span=(0, 7), match='123456가'>

연습문제

숫자 2자리 + 영어 소문자 2자리 + 숫자 4자리로 구성되었는지 확인하는 정규식 만들기

```
import re

text = '12ab1234'
regex = '코드 작성'
pattern = re.compile(regex)
res = pattern.match(text)

print(res)
```

12ab1234 입력

```
<_sre.SRE_Match object; span=(0, 8),  
match='12ab1234'>
```

12aba1234 입력

```
None
```

121ab1234 입력

```
None
```

98zx9876 입력

```
<_sre.SRE_Match object; span=(0, 8),  
match='98zx9876'>
```

연습문제

영어와 숫자로만 구성되어 있는지 확인하는 정규식 만들기

```
import re

text = 'abcd1234'
regex = '코드 작성'
pattern = re.compile(regex)
res = pattern.match(text)

print(res)
```

abcd1234 입력

```
<_sre.SRE_Match object; span=(0, 8),  
match='abcd1234'>
```

abcd가1234 입력

None

a 입력

```
<_sre.SRE_Match object; span=(0, 1),  
match='a'>
```

공백 입력

None

■ 정규 표현식 (Regular Expressions)

● search()

- 문자열 전체를 검색하여 정규식과 매치되는지 확인

```
>>> import re
>>> p = re.compile('[a-z]+')

>>> print(p.search('python'))
<_sre.SRE_Match object; span=(0, 6), match='python'>

>>> print(p.search(' python'))
<_sre.SRE_Match object; span=(1, 7), match='python'>

>>> print(p.search('1python'))
<_sre.SRE_Match object; span=(1, 7), match='python'>

>>> print(p.search('가python'))
<_sre.SRE_Match object; span=(1, 7), match='python'>
```

■ 정규 표현식 (Regular Expressions)

● search()

– 전화번호 찾아내기

```
text="기타 사항은 전화번호 : 02-1234-1678로 연락 주세요"
pattern = re.compile("#d{2,}-#d{3,}-#d{4}")
res = pattern.search(text)
print(res)

tel = res.group()
print(tel)
```

```
<_sre.SRE_Match object; span=(14, 26), match='02-1234-1678'>
02-1234-1678
```

– 이메일 찾아내기

```
text="제 이메일 주소는 ggoreb@ggoreb.com 입니다."
pattern = re.compile('##w+@##w+[.][a-z]{2,}')
res = pattern.search(text)
email = res.group()
print(email)
```

```
ggoreb@ggoreb.com
```

연습문제

문자열 중 사업자등록번호만 확인하는 정규식 만들기

```
import re

text = '대표자 : 꼬렘   사업자등록번호 : 100-99-23242324   대표전화 : 051-1234-5678'
regex = '코드 작성'
pattern = re.compile(regex)
res = pattern.search(text)

print(res)
```

- 조건
 - 숫자 3자리-숫자 2자리-숫자8자리

연습문제

문자열 중 전화번호 확인하는 정규식 만들기

```
import re
```

```
text = '대표자 : 꼬렙  사업자등록번호 : 100-99-23242324  대표전화 : 051-1234-5678'
```

```
regex = '코드 작성'
```

```
pattern = re.compile(regex)
```

```
res = pattern.search(text)
```

```
print(res.group())
```

○ 조건

– 숫자 2~3자리-숫자4자리-숫자4자리

051-1234-5678 입력

051-1234-5678

02-1234-5678 입력

02-1234-5678

■ 정규 표현식 (Regular Expressions)

● findall()

- 정규식과 매치되는 모든 문자열을 List로 반환

```
>>> import re
>>> p = re.compile('[a-z]+')

>>> result = p.findall('Life is too short, You need Python')
>>> result
['ife', 'is', 'too', 'short', 'ou', 'need', 'ython']
```

- 모든 숫자 List로 반환

```
>>> import re

>>> p = re.compile('[^~]') # 하이픈(-)이 아닌 1개 문자

>>> result = p.findall('010-8478-8181')
>>> result
['0', '1', '0', '8', '4', '7', '8', '8', '1', '8', '1']

>>> p = re.compile('[^~]+') # 하이픈(-)이 아닌 1개 이상의 문자

>>> result = p.findall('010-8478-8181')
>>> result
['010', '8478', '8181']
```

■ 정규 표현식 (Regular Expressions)

● findall()

– 에러 번호 찾아내기

```
text = "에러 404 : 페이지 없음\n 에러 500 : 서버 오류"  
regex = '에러\s(\d+)'  
pattern = re.compile(regex)  
res = pattern.findall(text)  
print(res)
```

['404', '500']

– 이름 찾아내기

```
text = "이름:홍길동, 이름      : 김길동, 이름 :      최길동"  
regex = '이름\s*:\s*(\w+)'  
pattern = re.compile(regex)  
res = pattern.findall(text)  
print(res)
```

['홍길동', '김길동', '최길동']

■ 정규 표현식 (Regular Expressions)

● finditer()

- 정규식과 매치되는 모든 문자열을 iterator 객체로 반환

```
>>> p = re.compile('[a-z]+')
>>> result = p.finditer('Life is too short, You need Python')
>>> result
<callable_iterator object at 0x00000147BD44BE80>
>>> for r in result:
    print(r)

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

연습문제

여러 기호를 제외하고 검색어만 확인하는 정규식 만들기

```
import re

text = '#서울, #강남, #맛집, #주꾸미'
regex = '코드 작성'
pattern = re.compile(regex)
res = pattern.findall(text)

print(res)
```

○ 조건

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

#서울, #강남, #맛집, #주꾸미 입력

['서울', '강남', '맛집', '주꾸미']

#서울, #강남, 입력

['서울', '강남']

연습문제

com과 net으로 끝나는 이메일 주소를 찾는 정규식 만들기

```
import re

text = '''주로 사용하는 이메일 주소는 a@ggoreb.com 이고,
가끔 사용하는 이메일 주소는 b@ggoreb.info 입니다.
이번에 새로 만든 c@ggoreb.net도 있습니다.'''
regex = '코드 작성'
pattern = re.compile(regex)

it = pattern.finditer(text)

for i in it:
    print(i.group())
```

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

■ 매칭되는 문자열 바꾸기 (sub)

● 특정 문자열을 찾은 후 원하는 문자열로 변경

```
import re

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

print(res)
```

GNU is Not OS, OS is not OS

● 변경 횟수 지정

```
import re

text = 'GNU is Not Unix, Unix is not Linux'
pattern = re.compile('Unix|Linux')
change = 'OS'
res = pattern.sub(change, text, 2)

print(res)
```

GNU is Not OS, OS is not Linux

■ 매칭되는 문자열 바꾸기 (sub)

● 특정 문자열을 찾은 후 원하는 문자열로 변경

```
import re

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

print(res)
```

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

● 지정 부분 제외 후 문자열 변경 (그룹 이용)

```
import re

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

print(res)
```

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

■ 매칭되는 문자열 바꾸기 (sub)

● 특정 문자열을 찾은 후 원하는 문자열로 변경

```
import re

text = '791111-1234567'
pattern = re.compile('#d{6}-#d{7}')
change = '*****'
res = pattern.sub(change, text)

print(res)
```

● 지정 부분 제외 후 문자열 변경 (그룹 이용)

```
import re

text = '791111-1234567'
pattern = re.compile('(##d{6})-#d{7}')
change = '#g<1>-*****'
res = pattern.sub(change, text)

print(res)
```

791111-*****

연습문제

전화번호 중 중간자리 숫자 마킹 처리

```
import re

text = '010-8478-8181'
regex = '코드 작성'
pattern = re.compile(regex)
change = 'Wg<1>-****-Wg<2>'
res = pattern.sub(change, text)

print(res)
```

010-****-8181

연습문제

이메일의 도메인 앞부분 마킹 처리

```
import re

user_list = ['ggoreb@naver.com', 'seorab@gmail.com', 'human@ggoreb.com']
regex = '코드 작성'
pattern = re.compile(regex)
change = 'Wg<1>*****Wg<2>'

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

```
ggoreb@*****.com
seorab@*****.com
human@*****.com
```