

정규 표현식(Regular Expression)

정규 표현식(Regular Expression)

- 일정한 패턴을 가진 문자열을 표현하는 형식.
- 일정한 패턴의 문자열을 검색, 추출, 변경할 때 사용.
- 문자열을 다루는 모든 프로그래밍에 사용되는 공통의 언어.
- 이메일 형식 체크, 전화번호 유형 체크...

모듈 import

```
import re
```

정규 표현식에 사용되는 메타 문자(meta characters)

메타 문자	의미	메타 문자	의미
.	$\backslash n$ 을 제외한 모든 문자	^	시작 문자
?	있거나 없거나	\$	끝 문자
*	0개 이상 반복	{m, n}	m회 이상 n회 이하
+	1개 이상 반복	[]	문자 클래스
$\backslash w$	메타 문자를 일반 문자로	()	그룹 패턴 지정
	논리 OR		

메타 문자 예제

메타 문자	의미	예제
.	$\forall n$ 을 제외한 모든 문자	a.b : Dot(.) 위치에 아무 문자나 하나 올 수 있다. aab, a0b, a!b
?	있거나 없거나	ab?c : "?" 앞에 문자인 "b"가 있거나 없어야 한다. ac, abc
*	0개 이상 반복	ca*t : "*"기호 바로 앞 문자인 a가 0번 이상 출현해야 한다. ct, cat, caaat
+	1개 이상 반복	ca+t : "+"기호 바로 앞 문자인 a가 1번 이상 나와야 한다. cat, caaat
	논리 OR	abc def : abc 또는 def
\backslash	메타 문자를 일반 문자로	$\backslash .$: .문자로 $\backslash ? :$? 문자로

메타 문자 예제

메타 문자	의미	예제
^	시작 문자	^a : a로 시작하면 매치됨 aab, a0b, a!b
\$	끝 문자	a\$: a로 끝나면 매치됨 bba, nana, 00a
{m, n}	m회 이상 n회 이하	go{2, 4}d : o가 2~4번 출현해야 매치됨 good, goood, gooodd
[]	문자 클래스	[abc] : 문자열에 a, b, c 중 하나가 있으면 매치됨 a, boy, day [a-c] : [abc]와 동일함. [a-zA-Z] : 모든 알파벳 [0-9] : 숫자 []안에서 "^"는 반대를 뜻함. [^0-9] : 숫자를 제외한 문자
()	그룹 패턴 지정	(abc)+ : "abc", "abcab", "abcabcab", 등과 매치

문자 클래스의 축약 기호

기존 표현식	축약 표현	설명	사용
[0-9]	\d	숫자	숫자
[^0-9]	\D	숫자가 아닌 것	문자, 특수문자, 화이트스페이스
[\t\n\r\f\v]	\s	whitespace 인 것	스페이스, TAB, 개행
[^\t\n\r\f\v]	\S	whitespace가 아닌 것	문자, 특수문자, 숫자
[a-zA-Z0-9]	\w	문자나 숫자	문자, 숫자
[^a-zA-Z0-9]	\W	문자나 숫자가 아닌 것	

\b: 단어 경계를 나타냅니다. 단어 문자와 비단어 문자 사이의 경계를 찾는 데 사용됩니다.

\B: 단어 경계가 아닌 부분에 대응됩니다.

메타 문자 : . , ?

1. Dot(.) : "." 위치에 "\n" 을 제외한 임의의 한 문자

```
import re
p="a.b"
s="a&b"
s="a9b"
s="a99b"      # None
print(re.search(p,s))
```

2. ? : "?" 기호 앞에 있는 문자가 있거나 없는 패턴.

```
import re
p="ab?c"      # ? 앞에 있는 문자인 "b"가 있거나 없는
s="ac"
s="accc"
p="ab?c"
s="acccd"     #ac가 있으므로 매치됨
print(re.search(p,s))
```

메타 문자 : * , +

3. * : "*" 기호 앞에 있는 문자가 0번 이상 출현하는 패턴.

```
import re
p="ca*t"      # * 앞에 있는 문자이 a가 0번 이상 출현해야 함
s="ct"
s="cat"
s="caaat"
s="caaats"
s="caaagt"    # c와 t 사이에 a외에 다른 문자는 못오므로 None
print(re.search(p,s))
```

4. + : "+" 기호 바로 앞에 있는 문자가 1번 이상 출현해야 함.

```
import re
p="ca+t"
s="cat"
s="caaats"
s="ct"       # a가 없으므로 None
print(re.search(p,s))
```


메타 문자 : | , ^

5. | : or

```
import re
p="cat|dog"
s="cats"
s="dog"
s="cute dog"
print(re.search(p,s))
```

6. ^ : ^ 기호 뒤에 오는 문자로 시작하는지

```
import re
p="^a"
s="ab"
s="a0b"
s="sab"      # a로 시작하지 않으므로 None
print(re.search(p,s))
```

메타 문자 : \$, {m, n}

7. \$: \$ 기호 앞에 오는 문자로 끝나는지

```
import re
p="a$"
s="bba"
print(re.search(p,s))
```

8. {m,n} : m~n회 사이 출현

```
import re
p="go{2,4}d"
s="good"
s="goood"
s="goood"
s="goooooood" # "o"가 2~4번만 가능하므로 None
print(re.search(p,s))
```

메타 문자 : [] , ()

9. [] : [] 사이의 문자가 있으면 됨.

```
import re
p="[abc]"
s="have a good day"
s="good"    # a, b, c 문자가 없으므로 None
s="boy"
print(re.search(p,s))
```

10. () : 그룹 지정

```
import re
p="(abc)+"
s="abc"
s="abcabc"
s="1 xabc11"
print(re.search(p,s))
```

문자 시퀀스

\d: 숫자(digit)에 대응됩니다. 0부터 9까지의 모든 숫자를 포함합니다.

\w: 단어 문자(word character)에 대응됩니다. 문자, 숫자, 밑줄(_)을 포함합니다.

\s: 공백 문자에 대응됩니다. 공백, 탭, 개행 문자 등을 포함합니다.

\D: 숫자가 아닌 문자에 대응됩니다.

\W: 단어 문자가 아닌 문자에 대응됩니다.

\S: 공백이 아닌 문자에 대응됩니다.

문자 시퀀스: `\d`, `\w`

11. `[0-9] == \d` : 숫자 지정

```
import re
p="[0-9]"
p=r"\d"           # p="[\d]"와 동일
s="1"
s="111"
s="a111"
s="abc"           # 숫자가 없으므로 None
print(re.search(p,s))
```

12. `[a-zA-Z0-9] == \w` : 문자나 숫자가 있으면 됨.

```
import re
p="[a-zA-Z0-9]"
p=r"\w"           # p="[\w]"와 동일
s="한글입니다"
s="!123!"
s="!@$"           # 문자나 숫자가 없으므로 None
print(re.search(p,s))
```

문자 시퀀스: \s

13. `[\t\n\r\f\v] == \s` : *whitespace* 가 있으면 됨.

```
import re
p=r"\s"                # p="[\t\n\r\f\v]"
s="hi\n!!!"
s="hello\twow!!!"
print(re.search(p,s))
```

문자 시퀀스: [^]

14. [^] : [] 기호 안에 "^"는 반대의 의미

```
import re
p=r"\D"      # [^0-9] == [\D]
s="123"       # 숫자가 아닌 문자가 있어야 하는데 문자가 없으므로 None
s="12a"
print(re.search(p,s))
```

```
p=r"\S"      # [^\t\n\r\f\v] == [\S]
s="\n"       # None
s="\nhi\n"
s=" "        # None
s='a'
print(re.search(p,s))
```

```
p=r"\W"      # [^a-zA-Z0-9] == [\W] : 문자나 숫자가 아닌
s="hello!"
s="안녕"     # 문자나 숫자가 아닌 것이 없으므로 None
print(re.search(p,s))
```

문자 시퀀스: \b

15. \b : 단어 경계

```
import re
p=r"\byou\b"      # you 단어가 있으면 됨
s="are you ready?"
s="your book"      # None
print(re.search(p,s))
```

```
p=r"\byou"  # you로 시작하면 됨
s="it is your book"      # you가 있으면 됨
print(re.search(p,s))
```

```
p=r'\byou\b'      # \b는 한글, 영문, 숫자를 제외한 문자열로 둘러싸인 문자
s='you! book'      # you 주변에 한글, 영문, 숫자가 없으므로 match
s='you&me book'
s='\nyou\tbook'
s='나you우리book'  # you 주변에 한글이 있으므로 None
print(re.search(p,s))
```


re의 주요 메서드들

re모듈의 주요 메서드

- `compile` : 정규식을 컴파일합니다. 이렇게 하면, 같은 패턴을 여러 번 사용할 때 성능이 향상됩니다.
- `search` : 문자열에서 정규식과 일치하는 첫 번째 부분을 찾습니다. 일치하는 항목이 없으면 `None`을 리턴합니다.
- `match` : 문자열의 시작 부분에서 패턴과 일치하는 부분을 찾습니다. 일치하는 항목이 없으면 `None`을 리턴합니다.
- `findall` : 패턴과 매치되는 모든 부분을 찾아 리스트로 반환합니다.
- `finditer` : 패턴과 매치되는 모든 부분을 찾아 이터레이터로 반환합니다.
- `sub(pattern, repl, string, count=0)` : `string`에서 `pattern`과 일치하는 부분을 `replace` 텍스트로 교체합니다. `count` 매개 변수를 사용하여 교체 횟수를 제한할 수 있습니다.
- `split(pattern, string, maxsplit=0)` : `pattern`을 기준으로 `string`을 분리하고, 분리된 문자열 리스트를 반환합니다. `maxsplit` 매개 변수를 사용하여 분할 횟수를 제한할 수 있습니다.

match, fullmatch

```
### match(p,s) : s가 p로 시작하는지 검사
p='a'
s="abc"
s="bcd"      # a로 시작하지 않으므로 None
print(re.match(p,s))
```

```
### fullmatch(p,s) : 문자열 전체가 패턴과 일치하는지 검사
p="abc"
s="abc"
s="abc def"  # 정확히 abc만 있어야 하므로 None
print(re.fullmatch(p,s))
```

```
### match vs fullmatch
p=r"\d{2,3}-\d{3,4}-\d{4}"
tel1="02-1234-12345"
print(re.match(p,tel1))
print(re.fullmatch(p,tel1))
```

search

search(p,s) : search는 포함하고 있는지 검사

p='a'

s="abc"

s="bcdaa"

s="bcd" *# a가 포함되어 있지 않으므로 None*

print(re.search(p,s))

match vs search

p="me"

s="help me!!!"

print(re.match(p,s)) *# me로 시작되지 않으므로 None*

print(re.search(p,s))

findall

findall : 패턴과 매치되는 모든 것을 찾아 리스트로 반환

```
p="apple"
```

```
s="apple banana pineapple april"
```

```
print(re.findall(p,s))
```

findall 응용(이메일주소 찾기)

```
p= r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
```

```
s = ""
```

이메일 목록:

```
john.doe@example.com
```

```
jane_doe123@company.org
```

```
info@website.net
```

```
""
```

```
print(re.findall(p,s))
```

단순 문자열이 아닌 패턴으로 찾기

```
p=r'[가-힣a-zA-Z]{3,}' # 한글과 알파벳 문자중 3자 이상인 것
```

```
s='abc def*efg 문자열 1234 패턴으로 분리'
```

```
print(re.findall(p,s))
```

sub

```
### sub : 패턴과 매치되는 것을 바꾸기
p="apple"
s="apple banana pineapple april"
print(re.sub(p,"애플",s))
# sub 응용(sub로 공백삭제하기)
word = 'I am home now. \n\nI am with my cat. \n\n'
# print(word)
# p="\n| "
p=r'\s+'
print(re.sub(p,' ',word))
```

split, finditer

split : 지정한 패턴으로 문자열을 분리해서 리스트로 반환

```
p = ' '  
s = "I am with my cat."  
print(re.split(p,s))  
# maxsplit로 분할 횟수 지정  
print(re.split(p,s,2))
```

finditer : 지정한 패턴을 찾아 이터레이터 객체를 반환

```
p = r'\d+\s'  
s = '1234 5678 1313'  
result=re.finditer(p, s)  
print(re.finditer(p, s))  
print([x.group() for x in result])
```

응용(이메일 주소 체크)

```
import re
### 이메일 주소 체크
email='happy!@seoul.na'
p=r'[a-zA-Z0-9+ -\_.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]{2,}'
if re.fullmatch(p,email):
    print(f"{email}은 유효합니다.")
else:
    print("이메일 형식에 맞지 않습니다.")
```


응용(전화번호 체크)

```
import re

### 전화번호 체크
tel="02-1234-5678"
p=r'\d{2,3}-\d{3,4}-\d{4}'
if re.fullmatch(p,tel):
    print(f"{tel}은 유효합니다.")
else:
    print("연락처 형식에 맞지 않습니다.")
```

응용(비밀번호 체크)

```
import re
```

```
### 비밀번호 체크 :
```

```
# 최소 8자, 최소 하나의 문자, 하나의 숫자 및 하나의 특수 문자
```

```
pwd="pass#word1"
```

```
p=r"^(?=.*[A-Za-z])(?=.*\d)(?=.*[@!%*#?&])[A-Za-z\d@!%*#?&]{8,}$"
```

```
if re.fullmatch(p,pwd):
```

```
    print(f"{pwd}은 유효합니다.")
```

```
else:
```

```
    print("비밀번호 형식에 맞지 않습니다.")
```

응용(friends 대본의 대화 내용 추출)

대본 대사 추출

<https://fangj.github.io/friends/> 에서 101 Monica Gets A Roommate 다운로드

```
import re
```

파일 읽고 대화 추출

```
f=open('friends101.txt')
```

```
script=f.read()
```

```
print(script)
```

```
monica_script=re.findall(r'Monica:.*',script)
```

```
# monica_script=re.findall(r'Monica:.*',script)
```

```
f.close()
```

대화 저장

```
f=open('monica.txt','w')
```

```
for monica in monica_script:
```

```
    f.write(monica+'\n')
```

```
f.close()
```