

11장. 정규 표현식(Regular Expression)



목 차

1

정규 표현식이란

2

정규식에 사용되는 메타 문자

3

문자열 검색 메서드

정규식 – Regular Expression

❖ 정규표현식이란?

- 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어이다. 문자열의 검색과 치환을 지원한다.



▼ 설명서 » 파이썬 표준 라이브러리 » 텍스트 처리 서비스 »

re — 정규식 연산

소스 코드 : [Lib / re.py](#)

이 모듈은 Perl에서 찾은 것과 유사한 정규식 일치 작업을 제공합니다.

검색 할 패턴과 문자열은 모두 `str` 8 비트 문자열 (`bytes`) 뿐만 아니라 유니 코드 문자열과 8 비트 문자열은 혼합 할 수 없습니다. 즉, 유니 코드 문자열과 8 비트 문자열은 혼합 할 수 없습니다. 이와 유사하게 대체를 요청할 때 대체 문자열은 유니 코드 문자열이어야 합니다.

정규식 – Regular Expression

◆ 정규표현식에 사용되는 메타문자

메타문자	설 명
[]	대괄호는 []사이의 문자들과 매치라는 의미를 나타낸다.
-	문자의 범위 를 지정하는 하이픈(-)
.	임의의 한개의 문자 를 나타내는 마침표(Dot)
^	부정을 나타내는 캐럿
*	0번 이상 반복
+	1번 이상 반복
{m}	m은 반복횟수
{n,m}	m은 반복횟수, n은 최소 반복 횟수
()	소괄호는 서브 클래스이다. 그룹을 만들 때 사용

정규식 – Regular Expression

◆ 정규 표현식 실습 – www.regexr.com

```
/[abcde]/g
```

Text Tests NEW

ace
after
before
king
sorry

abcde와 일치

```
/[a-e]/g
```

Text Tests NEW

ace
after
before
king
sorry

abcde와 일치

```
/[^a-e]/g
```

Text Tests NEW

ace
after
before
king
sorry

abcde가 아닐때 일치

```
/[A-F][1-9]/g
```

Text Tests NEW

B8
G7
D6
a2
1A

A1~F9와 일치

정규식 – Regular Expression

- 자주 사용하는 문자 클래스

정규 표현식	설 명
\d	숫자와 매치, [0-9]와 동일한 표현식이다.
\s	space나 Tab처럼 공백을 표현하는 문자와 매치
\w	문자+숫자와 매치, [a-zA-Z0-9]와 동일함

정규식 – Regular Expression

- 사용 예제

```
/[A-F]-[0-9]-[0-9a-zA-Z]/g
```

Text	Tests	NEW
A-5-5↵ F-9-C↵ G-17-10↵ a-2-9↵		

```
/[A-F]-\d-\w/g
```

Text	Tests	NEW
A-5-5↵ F-9-C↵ G-17-10↵ a-2-9↵		

대문자-숫자-전체문자와 일치

```
/서울..로/g
```

Text	Tests	NEW
서울충무로↵ 서울세종대로↵ 서울을지로↵ 서울노원구↵		

```
/[가-힣]+/g
```

Text	Tests	NEW
학교↵ 회사에 갑니다↵ 지하철4호선↵ 010-1234-5678↵		

한글과 일치

정규식 - Regular Expression

- 사용 예제 - 반복

```
/[a-zA-Z]+[-][0-9]+/g
```

Text	Tests	NEW
G-20		
COVID-19		
window10		
ace-of-base		

영문자-숫자(0~9)와 일치

```
/[0-9]{3}[-][0-9]{2}/g
```

Text	Tests	NEW
123-45		
010-2813		
11-666		
abc-de		

숫자 세 자릿수 - 두 자릿수와 일치

```
/\d{3}[-]\d{2}/g
```

Text	Tests	NEW
123-45		
010-2813		
11-666		
abc-de		

```
/[a-z]{4,8}/g
```

Text	Tests	NEW
cycling		
marathon		
run		
PLANE		

영문소문자 4~8문자와 일치

정규식 – Regular Expression

- 서브 패턴으로 감싸기 – 괄호, ()

```
/2020([A-F])-(...)/g
```

Text	Tests
2015A-fx↵	
2020F-dx↵	
2020f-sx↵	
2015a-fx↵	

- 서브 패턴이 2개

```
/(010|070)-{0,1}[0-9]{4}-{0,1}[0-9]{4}/g
```

Text	Tests
010-2222-3333↵	
070-1234-5678↵	
02-3946-815↵	
01022228888↵	

- 010 또는 070인 숫자 일치
- (하이픈)이 없거나 1개 있으면 일치

정규식을 사용한 문자열 검색

- 정규식을 사용한 문자열 검색 메서드

메서드	설 명
match()	문자열의 처음 부터 정규식과 매치되는지 조사
search()	문자열 전체 를 검색하여 정규식과 매치되는지 조사
findall()	정규식과 매치되는 모든 문자열을 리스트 로 돌려준다
finditer()	정규식과 매치되는 모든 문자열을 반복 가능한 객체 로 돌려줌

정규표현식 지원 – re 모듈

- re 모듈 사용방법

1. `re.compile('[a-z]+')` : `re.compile`을 사용하여 정규 표현식을 컴파일 한다.(바이트 코드로 바꿈)

2. 컴파일된 패턴 객체를 사용하여 문자열 검색을 수행한다.

- `match()` 메서드를 사용한 문자열 검색

```
import re

#compile -> byte 코드로 바꿈
p1 = re.compile("[a-z]+")
m1 = p1.match("afternoon")
print(m1)
```

```
<re.Match object; span=(0, 9), match='afternoon'>
```

정규식을 사용한 문자열 검색

- `search()` 메서드를 사용한 문자열 검색

```
import re

p = re.compile("[a-z]+")
m = p.match("2020 python") # 처음부터 일치해야 찾음
print(m)

s = p.search("2020 python") # 전체를 찾아서 일치하면 됨
print(s)
print(s.group()) # group()은 문자열로 출력
```

None

<re.Match object; span=(5, 11), match='python'>
python

정규식을 사용한 문자열 검색

- `findall()` 을 사용한 문자열 검색 – 결과를 리스트로 반환
 - `re.compile()`을 사용하지 않은 경우

```
import re

str = "Two is too"
m1 = re.findall("T[ow]o", str)
print(m1)

m2 = re.findall("T[ow]o", str, re.IGNORECASE)
print(m2)

m3 = re.findall("t[^w]o", str, re.IGNORECASE)
print(m3)
```

정규표현식

대, 소문자 구분
하지 않음

```
['Two']
['Two', 'too']
['too']
```

정규식을 사용한 문자열 검색

- `findall()` 을 사용한 문자열 검색
 - `re.compile()`은 검색할 내용이 많은 경우 사용하면 좋음

```
import re

str = "123?45yy7890 hi 999 hello"
# compile() 후 findall() - 검색할 내용이 많은 경우(속도 빠름)
pat = re.compile("\d{1,3}")
m = re.findall(pat, str)
print(m)
```

```
# findall()로 검색한 경우
m2 = re.findall('[A-z]', str)
print(m2)
```

```
m3 = re.findall("[1-5]{1,2}", str)
print(m3)
```

```
['123', '45', '789', '0', '999']
['y', 'y', 'h', 'i', 'h', 'e', 'l', 'l', 'o']
['12', '3', '45']
```

정규식을 사용한 문자열 검색

- findall() 을 사용한 문자열 검색

– ‘*’와 ‘+’의 차이

```
import re

# '*'와 '+'의 차이
p = re.compile("ca*t") # 앞 문자가 0번 이상 반복(즉 없어도 찾음)
m1 = re.findall(p, "caat")
print(m1)
m2 = re.findall(p, "ct")
print(m2)

p2 = re.compile("ca+t") # 앞 문자가 1번 이상 반복(즉, 없으면 못찾음)
m3 = re.findall(p2, "caat")
print(m3)
m4 = re.findall(p2, "ct")
print(m4)
```

정규식을 사용한 문자열 검색

- findall() 을 사용한 문자열 검색

- '*'와 '+'의 차이

```
import re

# 점(.)은 임의의 문자 , 괄호()는 서브클래스
# - 태그를 제외하고 괄호 안의 내용 매칭
str = "abcd<hr>Thank you"
pat1 = re.compile("<(.*)>")
m1 = re.findall(pat1, str)
print(m1)

# 태그 포함하여 괄호 안의 내용 매칭
pat2 = re.compile("<(.*)>")
m2 = re.findall(pat2, str)
print(m2)
```

```
['hr']
['<hr>']
```


정규식을 사용한 문자열 검색

- `finditer()`을 사용한 문자열 검색 – 결과를 객체로 반환

```
import re

str = "123?45yy7890 hi 999 Hello"

m1 = re.findall("\d{1,3}", str)
print(m1)

m2 = re.finditer("[A-Za-z]+", str)
#print(m2)
for i in m2:
    print(i)
```

```
['123', '45', '789', '0', '999']
<re.Match object; span=(6, 8), match='yy'>
<re.Match object; span=(13, 15), match='hi'>
<re.Match object; span=(20, 25), match='Hello'>
```

정규식을 사용한 문자열 검색

➤ match, search 객체의 메서드

match와 search 메서드를 수행한 결과로 돌려준 객체의 정보를 알 수 있는 메서드

메서드	목적
group()	매치된 문자열을 돌려준다.
start()	매치된 문자열의 시작위치를 돌려준다.
end()	매치된 문자열의 끝위치를 돌려준다
span()	매치된 문자열의 (시작, 끝)에 해당하는 튜플을 돌려준다.

정규식을 사용한 문자열 검색

➤ match, search 객체의 메서드

```
import re
p = re.compile('[a-z]+')
m=p.match("hello")

print(m.group())
print(m.start())
print(m.end())
print(m.span())
```

```
hello
0
5
(0, 5)
```

```
import re

p = re.compile("[a-z]+")
m = p.search("2020 hello")
print(m.group())
print(m.start())
print(m.end())
print(m.span())
```

```
C:\WINDOWS\system32\cmd.exe
hello
5
10
(5, 10)
```

그룹핑(Grouping)

- 그룹핑(Grouping)

문자열 중에서 특정 부분의 문자열만 뽑아내고 싶을 때 사용한다.

group(인덱스)	설 명
group(0)	매치된 전체 문자열
group(1)	첫 번째 그룹에 해당하는 문자열
group(2)	두 번째 그룹에 해당하는 문자열
group(n)	n 번째 그룹에 해당하는 문자열

그루핑(Grouping)

- 이름과 전화번호를 분리해서 추출하기

```
import re

p = re.compile("(\w+)\s+(\d+[-]\d+[-]\d+)")
m = p.search("jang 010-1119-1004")
print(m.group(1))
print(m.group(2))
```

C:\WINDOWS\system32\cmd.exe

jang
010-1119-1004

()가 그룹을 만든다.

그룹핑(Grouping)

- 그룹핑된 문자열에 이름 붙이기

(?P<그룹이름>)

```
p2 = re.compile("(?P<name>\w+)\s+(?P<phone>\d+[-]\d+[-]\d+)" )
m2 = p2.search("park 010-1234-5678")
print(m2.group("name"))
print(m2.group("phone"))
```

그루핑(Grouping)

- 문자열 바꾸기 – **sub 메서드**

sub 메서드를 사용하면 정규식과 매치되는 부분을 다른 문자로 바꿀 수 있다.

```
>>> import re
>>> p = re.compile('blue|red')
>>> s = p.sub('color', 'blue socks and red shoes')
>>> s
'color socks and color shoes'
```

참조 구문 사용하기 - sub(\g <그룹 이름>)

```
import re

p = re.compile("(?P<name>\w+)\s+(?P<phone>\d+[-]\d+[-]\d+)"
s = p.sub("\g<phone> \g<name>", "park 010-1111-2222")
print(s)
```

그루핑(Grouping)

- 문자열 바꾸기

참조 번호 사용하기 - sub(\g <1>)

```
import re

data = '''
kim 871212-1234567
lee 770707-2345678
'''

#jumin = re.compile("(\\d{6})[-]\\d{7}")
p = re.compile("(\\d+)[-]\\d+")
print(p.sub("\\g<1>-*****", data))
```

```
kim 871212-*****
lee 770707-*****
```