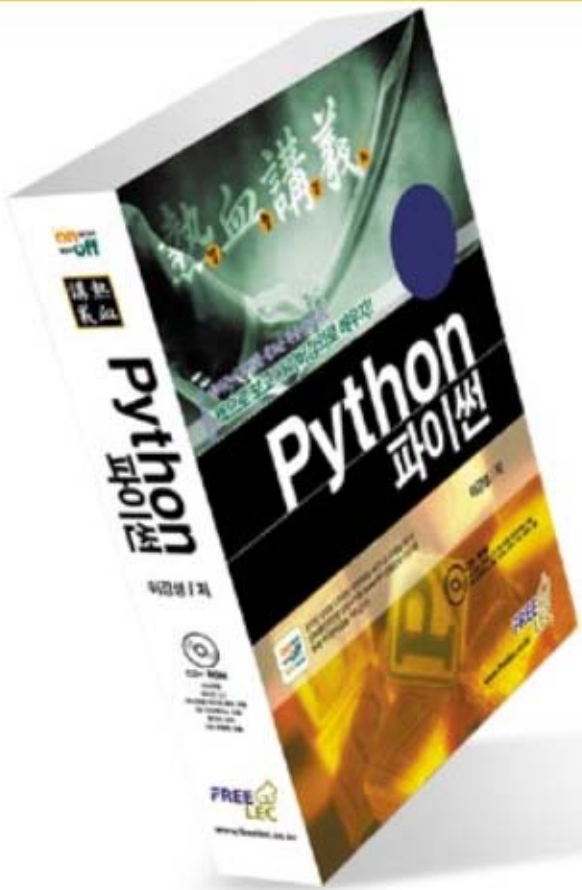
熱血講義

프리렉의 열혈강의 시리즈

Python 파이썬



열혈강의 Python 파이썬

❖제 20 강

주제 : 정규식

이강성(gslee@mail.gwu.ac.kr)

❖강의 목차

1. 개요
2. 메타문자
3. 패턴 매칭
4. 모듈 re의 주요 함수
5. 정규식 객체 사용하기
6. 문자열 대치

정규식이란?

- 문자열 검색
  - 다음 문자열에서 정수/실수를 표현하는 단어만 추출하라
    ‘ 876time a123 3 1.234 .567 892. a.b123’
- 정규식(Regular Expression)이란?
  - 문자열 패턴의 검색, 추출, 대치 규칙을 정의
  - 파이썬 정규식 모듈 - re

메타 문자

- 메타(meta) 문자?
  - 문자 자체보다 문자 패턴을 기술하기 위한 제어문자
- 반복 메타 문자

반복 메타 문자	의미	예
*	0회 이상 반복	ca*t는 ct, cat, caat, caaaaaat등과 매치
+	1회 이상 반복	ca+t는 cat, caaaaaat등과 매치된다.
?	0회 혹은 1회	ca?t는 ct, cat와 매치된다.
{m}	m회 반복	ca{2}는 caa와 매치된다.
{m,n}	m회부터 n회 까지 반복	ca{2,4}는 caat, caaat, caaaat와 매치된다.

메타문자

• 매칭 메타 문자

메타 문자	의미
.	줄바꿈 문자를 제외한 모든 문자와 매치. re.DOTALL 모드로 사용하면 줄 바꿈 문자도 매치된다.
^	1. 문자열의 시작과 매치된다. 2. re.MULTILINE 모드에서는 각 라인의 시작과 매치된다. 3. [ ] 메타 기호 안에서는 반대의 문자열을 취한다. [^5]는 5가 아닌 문자이다.
$	1. 문자열의 마지막과 매치된다. 2. [ ] 메타 기호 안에서는 메타 기호로 사용되지 않고 순수한 $ 문자로 매치된다.
[ ]	선택적 문자 집합을 나타낸다. [abc]는 "a", "b", "c"중 한문자를 의미한다. [a-c]와 같이 쓸 수 있다. -는 범위를 나타낸다. [a-zA-Z0-9]는 영문자 숫자를 의미한다.
|	a | b는 a 또는(or) b의 의미이다.
( )	정규식을 그룹으로 묶는다.

match

- 문자열 매치하기 - match 함수
  - match(*pattern*, *string*[, *flags*])
  - 리턴 : 매치된 경우 Match 객체
    아니면 None

```
>>> re.match('[0-9]', '1234')
<SRE_Match object at 00B06640>
>>> re.match('[0-9]', 'abc')  # 리턴 값이 없다
>>>
```

match

- **More examples**

```
>>> m = re.match('[0-9]', '1234') # 단일 숫자
>>> m.group()
'1'
>>> m = re.match('[0-9]+', '1234') # 1개 이상의 숫자
>>> m.group()
'1234'
>>> m = re.match('[0-9]+', '1234  ') # 뒤에 공백 오는 경우
>>> m.group()
'1234'
>>> re.match('[0-9]+', '    1234  ') # 앞에 공백. 매치 실패
>>> re.match(r'[ \t]*[0-9]+', '   123') # 공백 매치 추가
<_sre.SRE_Match object at 0x00AEEA60>
```

이스케이프 기호

기호	의미
\\	역슬러쉬 문자 자체를 의미한다.
\d	모든 숫자와 매치된다. [0-9]
\D	숫자가 아닌 문자와 매치된다. [^0-9]
\s	화이트스페이스 문자와 매치된다. [\t\n\r\f\v]
\S	화이트스페이스 문자가 아닌 것과 매치된다. [^ \t\n\r\f\v]
\w	숫자 또는 문자와 매치된다. [a-zA-Z0-9_]
\W	숫자 또는 문자가 아닌 것과 매치된다. [^a-zA-Z0-9_]
\b	단어의 경계를 나타낸다. 단어는 영문자 혹은 숫자의 연속 문자열로 가정한다.
\B	\b의 반대로 단어의 경계가 아님을 나타낸다.

```
>>> m = re.match('\s*\d+', '   123')
>>> m.group()
'   123'
>>> m = re.match('\s*(\d+)', '    1234  ')
>>> m.group(1) # 첫 번째 괄호 내의 문자열
'1234'
```

search

- 문자열 매치하기 – search 함수
  - match – 문자열 시작부터 매치
  - search – 부분적인 문자열 매치
  - 형식
    - search(*pattern, string*[, *flags*])

```
>>> re.search('\d+', '   1034  ').group()
'1034'
>>> re.search('\d+', ' asf123 ').group()
'123'
```

정규식

- raw 모드로 정규식 표현하기

```
>>> '\d' #
'\\d'
>>> '\s'
'\\s'
>>> '\b' # 백스페이스로 표현된다
'\x08'
>>> '\\b' # 이중으로 \를 추가해야 한다
'\\b'
>>> r'\b‘    # raw 문자열
'\\b'
```

정규식

- raw 모드로 정규식 표현하기

```
>>> '\\‘   # \ 문자 하나를 표현
'\\'
>>> r'\\'  # \ 문자 두 개
'\\\\'

# 하나의 \를 매치하기 위해 \ 네 개를 써야 한다
>>> re.search('\\\\', ' \ ').group()
'\\'
>>> re.search(r'\\', ' \ ').group()
'\\'
```

정규식

- raw 모드로 정규식 표현하기

```
>>> re.search('\d+', ' !123! ').group()
'123'
>>> re.search('\d+', ' abc123 ').group()
'123'
>>> re.search('\b\d+\b', ' !123! ')
>>> re.search('\b\d+\b', ' abc123 ')
>>> re.search('\\b\d+\\b', ' !123! ').group()
'123'
>>> re.search('\\b\d+\\b', ' abc123 ')
```

정규식

- 탐욕적 (greedy) 매칭
  - 가능한 한 최대한 많이 매칭하는 것.
- 최소(non-greedy) 매칭

기호	의미
*?	*와 같으나 문자열을 최소로 매치한다.
+?	+와 같으나 문자열을 최소로 매치한다.
??	?와 같으나 문자열을 최소로 매치한다.
{m,n}?	{m,n}과 같으나 문자열을 최소로 매치한다.

최소매칭을 위한 정규식

정규식

- 탐욕적 / 최소 매칭 예제

```
>>> s = '<a href="index.html">HERE</a><font size="10">'
>>> re.search(r'href="(.*)"', s).group(1)  # greedy
'index.html">HERE</a><font size="10'

>>> re.search(r'href="(.*?)"', s).group(1)
'index.html'
```

문자열 추출

- 매치된 문자열 추출하기
  - match 객체 메쏘드

메써드	설명
group()	매치된 문자열들을 리턴한다.
start()	매치된 문자열의 시작 위치를 리턴한다.
end()	매치된 문자열의 끝 위치를 리턴한다.
span()	매치된 문자열의 (시작, 끝) 위치 튜플을 리턴한다.

match 객체의 메쏘드

문자열 추출

- 매치된 문자열 추출하기
  - match 객체의 group 관련 메쏘드

group()	매치된 전체 문자열을 리턴한다
group(n)	n번째 그룹 문자열을 리턴한다. 0은 매치된 전체 문자열
groups()	매치된 전체 그룹 문자열을 터플 형식으로 리턴한다

group 메쏘드

문자열 추출

- group 예제

```
>>> m = re.match(r'(\d+)(\w+)(\d+)',
'123abc456')
>>> m.group()
'123abc456'
>>> m.group(0)
'123abc456'
>>> m.group(1)
'123'
>>> m.group(2)
'abc45'
>>> m.group(3)
'6'
```

```
>>> m.groups()
('123', 'abc45', '6')
>>> m.start()
0
>>> m.end()
9
>>> m.span()
(0, 9)
```

문자열 추출

- group 예제

```
>>> m = re.match('(a(b(c))(d))',
'abcd')
>>> m.group(0)
'abcd'
>>> m.group(1)
'abcd'
>>> m.group(2)
'bc'
>>> m.group(3)
'c'
>>> m.group(4)
'd'
```

그룹 이름

- 그룹에 이름 사용하기
  - 사용 패턴 : (?P<name>...)
  - name : 그룹 이름, ... : 일반 정규식

```
>>> m = re.match(r'(?P<var>[_a-zA-
Z]\w*)\s*=\s*(?P<num>\d+)', 'a = 123')
>>> m.group('var')
'a'
>>> m.group('num')
'123'
>>> m.group()
'a = 123'
```

re 모듈 주요 함수

메써드	내용
compile(pattern[, flags])	pattern을 컴파일 하여 정규식 객체를 리턴한다(다음절 참조).
search(pattern, string[, flags])	string을 검사해서 pattern에 맞는 문자열 부분이 있는가를 찾는다.
match(pattern, string[, flags])	string의 시작부터 pattern에 맞는지 검사한다.
split(pattern, string[, maxsplit = 0])	string을 pattern을 기준으로 분리한다.
findall(pattern, string)	string에서 pattern을 만족하는 모든 문자열을 추출한다.
sub(pattern, repl, string[, count = 0])	string에서 pattern을 repl로 대치한다. (20-5절 참조)
subn(pattern, repl, string[, count = 0])	sub와 동일하나 대치횟수도 함께 전달된다. (20-5절 참조)
escape(string)	영문자 숫자가 아닌 문자들을 백슬러쉬 처리해서 리턴한다. 임의의 문자열을 정규식 패턴으로 사용할 경우에 유용하다.

- **re. split(*pattern, string*[, *maxsplit = 0*])**

```
>>> re.split('\W+', 'applt, orange and spam.')  #
영문자,숫자,언더라인 이외의 1개 이상의 문자를 기준으로 분리한다

['applt', 'orange', 'and', 'spam', '']

>>> re.split('(\W+)', 'applt, orange and spam.') # 패턴에 괄호가
있으면 괄호에 포함된 문자열도 함께 포함된다.

['applt', ', ', 'orange', ' ', 'and', ' ', 'spam', '.', '']

>>> re.split('\W+', 'applt, orange and spam.', 1) # maxsplit
인수가 주어졌을 경우 최대 분리 횟수를 나타낸다.

['applt', 'orange and spam.']

>>> re.split('\W+', 'applt, orange and spam.', 2)

['applt', 'orange', 'and spam.']

>>> re.split('\W+', 'applt, orange and spam.', 3)

['applt', 'orange', 'and', 'spam.']
```

- **findall(*pattern, string*)**

```
>>> re.findall(r'[_a-zA-Z]\w*', '123 abc 123 def')
['abc', 'def']
>>> s = '''
<a href="link1.html">link1</a>
<a href="link2.html">link2</a>
<a href="link3.html">link3</a>'''
>>> re.findall(r'href="(.*?)"', s)
['link1.html', 'link2.html', 'link3.html']
```

정규식 객체 사용하기

- **compile(*pattern*[, *flags*])**
  - 정규식을 컴파일해서 정규식 객체를 리턴
  - 정규식의 반복적 적용에 유용
  - 정규식에 플래그 함께 사용 가능

```
>>> p = re.compile(r'([_a-zA-Z]\w*)\s*=\s*(\d+)')
>>> m = p.match('a = 123')
>>> m.groups()
('a', '123')
>>> m.group(1)
'a'
>>> m.group(2)
'123'
```

정규식 객체 사용하기

- **compile 함수에 사용 가능한 플래그**

플래그	내용
I, IGNORECASE	대소문자를 구별하지 않고 매치한다.
L, LOCALE	\w, \W, \b, \B를 현재의 로케일에 영향을 받는다.
M, MULTILINE	^가 문자열의 맨 처음, 그리고 각 라인의 맨 처음과 매치된다. $는 문자열의 맨 끝, 그리고 각 라인의 끝에 매치된다. 기본 값은 ^는 문자열의 맨 처음만, $는 문자열의 맨 마지막만 매치된다.
S, DOTALL	.를 줄바꾸기 문자(\n)도 포함하여 매치하게 한다.
U, UNICODE	\w, \W, \b, \B가 유니코드 문자에 맞게 처리된다.
X, VERBOSE	보기 좋은 정규식을 쓸 수 있게 해준다. 정규식 안의 공백은 무시된다. 공백을 사용하려면 백스러쉬 문자로 표현해야 한다. 정규식 안에 #문자를 쓰면 그 주석으로 그 이후의 모든 문자는 무시된다.

정규식 객체 사용하기

- 예제

```
>>> import re
>>> p = re.compile('the', re.I)
>>> p.findall('The cat was hungry. They were scared
because of the cat')
['The', 'The', 'the']
```

정규식 객체의 주요 메쏘드

메써드	내용
search(string[, pos[, endpos]])	string을 검사해서 문자열 부분을 찾는다. pos, endpos는 string의 찾을 문자열의 범위를 지정한다.
match(string[, pos[, endpos]])	string의 처음부터 정규식 패턴에 맞는지 검사한다.
split(string[, maxsplit = 0])	re.split와 기능상 동일하다.
findall(string)	re.findall과 기능상 동일하다.
sub(repl, string[, count = 0])	re.sub와 기능상 동일하다.
subn(repl, string[, count = 0])	re.subn과 기능상 동일하다.

- **search(*string*[, *pos*[, *endpos*]])**

```
>>> t = 'My ham, my egg, my spam'
>>> p = re.compile('(my)', re.I)
>>> pos = 0
>>> while 1:
        m = p.search(t, pos)
        if m:
                print m.group(1)
                pos = m.end()
        else: break
My
my
my
```

문자열 대치

- **sub(*repl, string*[, *count = 0*])**
  - **count를 설정하면 대치 횟수 조절 가능**

```
>>> re.sub(r'[.,:;]', '', 'a:b;c, d.')  # 특수문자 없애기
'abc d'
>>> re.sub(r'\W', '', 'a:b;c, d.')   # 영문자, 숫자,
                                     # 언더라인만 제외하고 모두 없애기
'abcd'
```

문자열 대치

- 대치할 문자열에 매치된 문자열 다시 사용하기
  - {\n}, {g\<n>} : 그룹 n의 문자열로 대치한다.

```
>>> p = re.compile('section{ ( [^}]* ) }', re.VERBOSE)
# VERBOSE를 쓴 것은 공백을 무시하기 위해서이다.

>>> p.sub(r'subsection{\1}','section{First}
section{second}')

'subsection{First} subsection{second}'
```

- 매치된 문자열 다시 사용하기
  - {\g<name>} : 그룹 n의 문자열로 대치한다.

```
>>> p = re.compile('section{ (?P<name> [^}]* ) }', re.VERBOSE)
>>> p.sub(r'subsection{\g<name>}','section{First}')
'subsection{First}'
```

문자열 대치

- 대치 문자열을 함수로 처리하기
  - sub의 첫 인수로 대치 함수 지정

```
>>> def hexrepl( match ):
        value = int( match.group() )
        return hex(value)

>>> p = re.compile(r'\d+')
>>> p.sub(hexrepl, 'Call 65490 for printing, 49152 for user code.')
'Call 0xffd2 for printing, 0xc000 for user code.'
```

예제 : 주민등록 번호 추출

- 주민등록번호
  - 6자리숫자 – 7자리 숫자
  - 6자리숫자7자리숫자

```
>>> p = re.compile('(\d{6})-?(\d{7})')
>>> p.search('123456-1234567').groups()
('123456', '1234567')
>>> p.search('1234561234567').groups()
('123456', '1234567')
```

예제:HTML에서 URL 추출

- HTML문서에서 <a href=..>에 의한 링크 추출

```
p = re.compile('''
href=([^'"]\S+?)[\s>] |     # href=http://www.python.or.kr
href="([^"]*?)" |           # href="http://www.python.or.kr"
href='([^']*?)'             # href='http://www.python.or.kr'
''', re.I | re.X)
```

```
import re

s = '''<a href="http://www.daum.net">daum</a>
<a href='http://www.naver.com'>naver</a>
<a href=http://www.chosun.com>chosun</a>
<a href=http://www.chosun.com class>chosun</a>
<a href="http://job.daum.net/ " target="new">
<a href="http://go.daum.net/bin/go.cgi?relative=1&url=/Mail-
bin/login_f.cgi%3Ferror%3Dlogin" class="tls0">
'''
p =
re.compile('''href=([^'"]\S+?)[\s>]|href="([^"]*?)"|href='([^']*?)\'''',
re.I)
pos = 0
while 1:
        match = p.search(s, pos)
        if match:
                url = match.groups()
                pos = match.end()
                print filter(None, url)[0]
        else:
                break
```

- 출력결과

```
http://www.daum.net
http://www.naver.com
http://www.chosun.com
http://www.chosun.com
http://job.daum.net/
http://go.daum.net/bin/go.cgi?relative=1&url=/Mail-bin/login_f.cgi%3Ferror%3Dlogin
```