제4장 정규표현식

정규식표현식 (REGULAR EXPRESSION) 모듈

[반복 메타 문자]

[메타 문자]	[의미]
*	0회 이상 반복
+	1회 이상 반복
?	0회 or 1회
{m}	m회 반복
{m, n}	m회부터 n회까지 반복

[매칭 메타 문자]

[메타 문자]	[의미]
.	줄바꿈 문자를 제외한 모든 문자와 매치됨
^	문자열의 시작과 매치됨
$	문자열의 마지막과 매치됨
[ ]	문자 집합 중 한 문자를 의미
|	또는(or)를 의미
{ }	정규식을 그룹으로 묶음

정규표현식 기호(2)

[이스케이프 기호]

[종류]	[설명]
₩₩	역슬래쉬 문자 자체
₩d	모든 숫자와 매치됨 [0-9]
₩D	숫자가 아닌 문자와 매치됨 [^0-9]
₩s	화이트 스페이스 문자와 매치됨 [₩t₩n₩r₩f₩v]
₩S	화이트 스페이스가 아닌 것과 매치됨 [^ ₩t₩n₩r₩f₩v]
₩w	숫자 또는 문자와 매치됨 [a-zA-Z0-9_]
₩W	숫자 또는 문자가 아닌 것과 매치됨 [^a-zA-Z0-9_]
₩b	단어의 경계를 나타냄. 단어는 영문자 혹은 숫자의 연속 문자열
₩B	단어의 경계가 아님을 나타냄
₩A	문자열의 처음에만 일치
₩Z	문자열의 끝에만 일치

정규표현식 기호(3)

[최소 매칭을 위한 정규식]

[기회]	[의미]
*?	*와 같으나 문자열을 취소로 매치함
+?	+와 같으나 문자열을 취소로 매치함
??	?와 같으나 문자열을 최소로 매치함
{m,n}?	{m,n}과 같으나 문자열을 최소로 매치함

[정규 표현식에서 사용 가능한 플래그]

[플래그]	[내용]
I, IGNORECATE	대, 소문자를 구별하지 않는다
L, LOCATE	₩w, ₩W, ₩b, ₩B를 현재의 로케일에 영향을 받게 한다
M, MULTILINE	^가 문자열의 맨 처음, 각 라인의 맨 처음과 매치 된다 $는 문자열의 맨 끝, 각 라인의 맨 끝과 매치
S, DOTALL	.을 줄바꾸기 문자도 포함하여 매치하게 한다
U, UNICODE	₩w, ₩W, ₩b, ₩B가 유니코드 문자 특성에 의존하게 한다
X, VERBOSE	정규식 안의 공백은 무시된다

정규표현식 기호(4)

[re모듈의 주요 메소드]

[메소드]	[설명]
compile(pattern[, flags])	pattern을 컴파일하여 정규식 객체를 반환
match(pattern, string[,flags])	string의 시작부분부터 pattern이 존재하는지 검사하여 MatchObject 인스턴스를 반환
search(pattern, string[,flags])	string의 전체에 대해서 pattern이 존재하는지 검사하여 MatchObject 인스턴스를 반환
split(pattern, string[, maxplit=0])	pattern을 구분자로 string을 분리하여 리스트로 반환
findall(pattern, string[, flags])	string에서 pattern을 만족하는 문자열을 리스트로 반환
finditer(pattern, string[, flags])	string에서 pattern을 만족하는 문자열을 반복자로 반환
sub(pattern, repl, string[, count=0])	string에서 pattern과 일치하는 부분에 대하여 repl로 교체하여 결과 문자열을 반환
subn(pattern, repl, string[, count=0])	sub와 동일하나, 결과로(결과문자열, 매칭횟수)를 튜플로 반환
escape(string)	영문자 숫자가 아닌 문자들을 백슬래쉬 처리해서 리턴. (임의의 문자열을 정규식 패턴으로 사용할 경우 유용)

정규표현식 기호(5)

[Match 객체]

Match객체는 match(), search()의 수행 결과로 생성되며, 검색된 결과를 효율적으로 처리할 수 있는 기능 제공.

◎ Match객체가 지원하는 메소드와 속성

[메소드]	[속성]
group([group1, ...])	입력받은 인덱스에 해당하는 매칭된 문자열 결과의 부분 집합을 반환합니다. 인덱스가 '0'이거나 입력되지 않은 경우 전체 매칭 문자열을 반환합니다.
groups()	매칭된 결과를 튜플 형태로 반환
groupdict()	이름이 붙여진 매칭 결과를 사전 형태로 반환
start([group])	매칭된 결과 문자열의 시작 인덱스를 반환. (인자로 부분 집합의 번호나 명시된 이름이 전달된 경우, 그에 해당하는 시작 인덱스를 반환)
end([group])	매칭된 결과 문자열의 종료 인덱스를 반환. (인자로 부분 집합의 번호나 명시된 이름이 전달된 경우, 그에 해당하는 종료 인덱스를 반환)
pos	원본 문자열에서 검색을 시작하는 위치입니다.
endpos	원본 문자열에서 검색을 종료하는 위치입니다.
lastindex	매칭된 결과 집합에서 마지막 인덱스 번호를 반환. (일치된 결과가 없는 경우에는 None을 반환)
lastgroup	매칭된 결과 집합에서 마지막으로 일치한 이름을 반환. (정규식의 매칭 조건에 이름이 지정되지 않았거나 일치된 결과가 없는 경우 None 반환)
string	매칭의 대상이 되는 원본 문자열입니다.

정규표현식 기호(6)

[예제] - re 모듈 함수

▶ re.match()와 re.search()의 차이

re.match()의 경우 대상 문자열의 시작부터 검색을 하지만,

re.search()함수는 대상 문자열 전체에 대해서 검색을 수행한다.

예) 아래와 같이 검색의 대상이 되는 문자열에 공백이 있는 경우나 또는

검색 키워드와 일치하는 문자열이 대상 문자열의 중간 이후에 존재하는 경우

re.match()함수는 검색을 못함.

```
>>> import re
>>> bool(re.match('[0-9]*th', '    35th')) # 불린으로 검색 결과 확인
False
>>> bool(re.search('[0-9]*th','    35th'))
True
>>> bool(re.match('ap', 'This is an apple')) # 문자열의 시작부터 검색
False
>>> bool(re.search('ap','This is an apple')) # 문자열 전체에 대해 검색
True
```

▶ re.split() - 대상 문자열을 입력된 패턴을 구분자로 하여 분리

```
>>> # ':', '.', ' '문자를 구분자로 사용
>>> re.split('[:. ]+', 'apple Orange:banana tomato')
['apple', 'Orange', 'banana', 'tomato']
>>> # 패턴에 괄호를 사용하면 해당 분리 문자도 결과 문자열에 포함
>>> re.split('([:. ])+', 'apple Orange:banana tomato')
['apple', ' ', 'Orange', ':', 'banana', ' ', 'tomato']
>>> # maxsplit이 입력된 경우
>>> re.split('[:. ]+', 'apple Orange:banana tomato', 2)
['apple', 'Orange', 'banana tomato']
```

정규표현식 기호(7)

▶ re.findall() - 검색 문자열에서 패턴과 매칭되는 모든 경우를 찾아 리스트로 반환

```
>>> # 매치되는 문자열이 있는 경우
>>> re.findall(r"app\w*", "application orange apple banana")
['application', 'apple']
>>>
>>> # 매치되는 문자열이 없는 경우
>>> re.findall(r"king\w*", "application orange apple banana")
[]
```

▶ re.sub() - 패턴과 일치하는 문자열 변경

```
>>> # 주민등록번호 형식을 변경
>>> re.sub("-", "@", "901225-1234567")
'901225@1234567'
>>> # 필드 구분자를 통일
>>> re.sub(r"[:,|\s]", ", ", "Apple:Orange Banana|Tomato")
'Apple, Orange, Banana, Tomato'
>>> # 문자열의 변경 횟수를 제한
>>> re.sub(r"[:,|\s]", ", ", "Apple:Orange Banana|Tomato", 2)
'Apple, Orange, Banana|Tomato'
```

▷ 또한 변경할 문자열에 대해서 매칭된 문자열을 사용할 수도 있다.

```
>>> # html문장의 일부분 중 연도부분을 이탤릭체로 변경하는 예제
>>> re.sub(r"\b(\d{4}-\d{4})\b", r"<I>\1</I>",\
        "Copyright Derick 1990-2009")
'Copyright Derick <I>1990-2009</I>'
```

정규표현식 기호(8)


re.sub(r"₩b(₩d{4}-₩d{4})₩b", r"<I>₩1</I>", "Copyright Derick 1990-2009")
변경할 문자열에 대해서는 '₩<숫자>' 형태로 사용
매칭시킬 패턴 중 변경할 문자열에 사용할 부분에 대해서 소괄호로 감싸준다.
>>> re.sub(r"\b(?P<year>\d{4}-\d{4})\b", r"<I>\g<year></I>"
, "Copyright Derick 1990-2009")
'Copyright Derick <I>1990-2009</I>'
re.sub(r"₩b(?P<year>₩d{4}-₩d{4})₩b", r"<I>₩g<year></I>", ...)
매칭시킬 패턴에 명시적으로 이름을 지정('(?P<패턴_이름>)')
변경할 문자열에서도 그 이름을 '₩g<패턴_이름>' 형태로 사용

정규표현식 기호(9)

[예제] - 정규 표현식 객체

▶ 동일한 패턴을 연속적으로 검색하는 경우, 정규식을 컴파일하여 정규표현식 객체를 생성.

```
>>> c = re.compile(r"app\w*")
>>> # 정규식을 분석하여 객체에 저장
>>> c = re.compile(r"app\w*")
>>> c.findall("application orange apple banana") # 분석없이 검색
['application', 'apple']
```

▶ re.IGNORECASE 플래그로 대소문자 구분하지 않고 매칭 작업 수행

```
>>> s = 'Apple is a big company and apple is very delicious.'
>>> c = re.compile('apple', re.I)  # 대소문자 구분하지 않음
>>> c.findall(s)
['Apple', 'apple']
```

▶ re.MULTILINE 플래그를 설정하여 빈 라인을 제외하고 라인별로 분리

```
>>> s = """window
unix
linux
solaris"""
>>> c = re.compile('^.+') # 첫 라인만을 매칭
>>> c.findall(s)
['window']
>>> c = re.compile('^.+', re.M) # MULTILINE 설정
>>> c.findall(s)
['window', 'unix', 'linux', 'solaris']
```

정규표현식 기호(10)

[예제] - Match 객체

▶ 일반적인 형식의 전화번호를 인식하여 Match 객체가 지원하는 메소드 분석.

```
>>> # 전화번호를 인식하는 정규식 객체 생성
>>> telChecker = re.compile(r"(\d{2,3})-(\d{3,4})-(\d{4})")
>>> m = telChecker.match("02-123-4567")
>>> m.groups() # 매칭된 문자열 집합을 튜플로 반환
('02', '123', '4567')
>>> m.group()  # 매칭된 전체 문자열을 반환
'02-123-4567'
>>> m.group(1) # 첫 번째로 매칭된 문자열
'02'
>>> m.group(2,3) # 두 번째와 세 번째 매칭된 문자열을 튜플로 반환
('123', '4567')
>>> m.start()  # 매칭된 전체 문자열의 시작 인덱스
0
>>> m.end()    # 매칭된 전체 문자열의 종료 인덱스
11
>>> m.start(2) # 두번째 매칭된 문자열("123")의 시작 인덱스
3
>>> m.end(2)   # 두번째 매칭된 문자열("123")의 종료 인덱스
6
>>> m.string[m.start(2):m.end(3)] # 지역번호를 제외한 전화번호 출력
'123-4567'
```

▶ 정규식 작성시 '(?<이름>..)' 형식으로 매칭 결과에 대해 이름을 부여하고,
groupdict() 메서드를 이용하여 사전 형태로 이름과 검색된 문자쌍을 얻음.

```
>>> # 매칭 결과에 대해서 이름을 부여
>>> c = re.compile(r"(?P<area>\d+)-(?P<exchange>\d+)-(?P<user>\d+)")
>>> m = c.match("02-123-4567")
>>> m.group("area")  # 'area'의 이름에 해당하는 문자열
'02'
>>> m.start("user")  # 'user'의 이름에 해당하는 시작 인덱스
7
```

정규표현식 기호(11)

로우 문자열 표기법 (Raw string notation)

이스케이프 문자열을 표현하기 위하여 '₩'(백슬래쉬)문자를 사용하기 때문에, 문자 '₩'를 정규표현식으로 표현하기 위해서는 '₩₩₩₩'로, 일반 문자열에선느 '₩₩'로 표현해야 합니다. 그래서 '₩apple'이란 문자열을 검색하기 위해서는 아래와 같이 매우 복잡한 형식으로 표현해야 합니다.

```
>>> bool(re.search("\\\\\w+", "\\apple"))
True
```

로우 문자열 표기법은 문자열 앞에 'r'을 더한 것으로, ₩(백슬래쉬) 문자를 이스케이프 문자열로 처리하지 않고 일반 문자와 동일하게 처리합니다. 이렇게 함으로써 정규표현식 및 문자열에서 '₩'를 간단하게 표현할 수 있습니다. 일반적으로 정규표현식에 사용되는 문자열에서는 이러한 편리함 때문에 많이 사용합니다.

```
>>> bool(re.search("\\\\\w+", "\\apple")) # 로 표기법 적용 안된 경우
True
>>> bool(re.search(r"\\\w+", r"\apple"))  # 로 표기법이 적용 된 경우
True
```