Chapter

1

크롤링과 스크레이핑

ParseResult 속성

- urlparse() 를 통해 URL을 튜플의 하위 클래스인 ParseResult 클래스로 변환
- ParseResult의 속성

URL 예시) scheme://username:passwd@netloc:port/path;params?query#fragment

속성	값	설명
scheme	scheme	스키마(http, http, ftp, mailto 등)
username	username	사용자 이름 passwd passwd 비밀번호
hostname	netloc	호스트명
port	port	port번호
netloc	username:passwd@netloc:port	네트워크 위치
path	/path	경로
params	;params	URL 파라미터
query	?query	쿼리스트링
fragment	#fragment	fragement 식별자

출처: https://dololak.tistory.com/254 [코끼리를 냉장고에 넣는 방법]

정규표현식

❖ 정규표현식(Regular expressions)

- 특정한 규칙을 가진 문자열의 집합을 표현하는 데 사용하는 형식 언어
- 복잡한 문자열의 검색과 치환을 위해 사용되며, Python 뿐만 아니라 문자열을 처리하는 모든 곳에서 사용

❖ 메타 문자(Meta characters)

- 메타 문자란 문자가 가진 원래의 의미가 아닌 특별한 용도로 사용되는 문자
- 정규표현식에서 사용되는 메타 문자
  - . ^ $ * + ? ₩ | ( ) { } [ ]

메타문자	설명
[]	정규표현식에서 대괄호 [] 는 대괄호 안에 포함된 문자들 중 하나와 매치
[-]	[] 안의 두 문자에 -를 사용하면 두 문자 사이의 범위
[^]	[] 안에서 ^는 부정
.	.은 줄바꿈 문자인 \n 을 제외한 모든 문자 1개와 매치
[.]	[] 사이에서 .을 사용할 경우 문자 원래의 의미인 마침표
*	반복, * 앞에 오는 문자가 0개 이상 반복되어야 매치
+	+ 앞에 있는 문자가 최소 한 번 이상 반복되어야 매치

정규표현식

메타문자	설명
?	? 앞에 있는 문자가 없거나 하나 있을 때 매치
{m, n}	{m, n} 앞에 있는 문자가 m 번에서 n 번까지 반복될 때 매치된다.
표현식1|표현식2|…	or 의 의미가 적용되어 정규표현식들 중 어느 하나와 매치
^	^의 뒤에 있는 문자로 시작되면 매치 여러 줄의 문자열일 경우 첫 줄만 적용(단, re.MULTILINE 옵션이 적용되면 각 줄의 첫 문자를 검사하여 매치)
$	$의 앞에 있는 문자로 끝나면 매치 여러 줄의 문자열일 경우 마지막 줄만 적용(단, re.MULTILINE 옵션이 적용되면 각 줄의 마지막 문자를 검사하여 매치)

문자 클래스	설명
\d	숫자 [0-9]와 같다.
\D	비숫자 [^0-9]와 같다.
\w	숫자 + 문자 [a-zA-Z0-9]와 같다.
\W	숫자 + 문자가 아닌 것 [^a-zA-Z0-9]와 같다.
\s	공백 [\t\n\r\f\v]와 같다.
\S	비공백 [^ \t\n\r\f\v]와 같다.
\b	단어 경계 (\w 와 \W 의 경계)
\B	비단어 경계

정규표현식

사용 예제	결과
[abc]	'a' # a와 매치 'boy' # b와 매치 'dye' # a, b, c 중 어느 문자와도 매치되지 않음
[a-c] [0-5] [a-zA-Z] [0-9]	[abc]와 같음 [012345]와 같음 모든 알파벳(대소문자) 숫자
[^0-9] [^abc]	숫자를 제외한 문자만 매치 a, b, c를 제외한 모든 문자와 매치
a.b	aab # a와 b 사이의 a는 모든 문자에 포함되므로 매치 a0b # a와 b 사이의 0은 모든 문자에 포함되므로 매치 abc # a와 b 사이에 문자가 없기 때문에 매치되지 않음
a[.]b	a.b # a와 b 사이에 마침표가 있으므로 매치 a0b # a와 b 사이에 마침표가 없으므로 매치 안됨
lo*l	ll # 매치 lol # 매치 looool # 매치 loooooooooooooooooooool # 매치 lbl # 매치 안됨 loooooooooooboooooooool # 매치 안됨

정규표현식

사용 예제	결과
lo+l	ll # 매치 안됨 lol # 매치 looooool # 매치
lo?l	ll # 매치 lol # 매치 lool # 매치 안됨
lo{3, 5}l	lol # 매치 안됨 loool # 매치 loooool # 매치 looooool # 매치 안됨
a|b|c	a # 매치 b # 매치 c # 매치 a b # 매치 a b c # 매치 d # 매치 안됨
^a	a # 매치 aaa # 매치
a$	aa # 매치 baa # 매치

re Python 정규표현식 모듈

❖ Python 에서는 re 모듈을 통해 정규표현식을 사용

```
>>> import re
```

- compile 정규표현식 컴파일
  - 정규표현식을 컴파일하여 변수에 저장한 후 사용

```
>>> 변수이름 = re.compile('정규표현식')
```

  - match: 시작부터 일치하는 패턴 찾기
    - 문자열의 처음 시작부터 검색하여 일치하지 않는 부분이 나올 때까지 찾기

```
>>> p = re.compile('[a-z]+')
>>> p.match('aaaaa')
<_sre.SRE_Match object; span=(0, 5), match='aaaaa'>

>>> p.match('bbbbbbbbb')
<_sre.SRE_Match object; span=(0, 9), match='bbbbbbbbb'>

>>> p.match('1aaaa')
None

>>> p.match('aaa1aaa')
<_sre.SRE_Match object; span=(0, 3), match='aaa'>
```

re Python 정규표현식 모듈

- search: 전체 문자열에서 첫 번째 매치 찾기
  - 문자열 전체에서 검색하여 처음으로 매치되는 문자열을 찾기

```
>>> p = re.compile('[a-z]+')
>>> p.search('aaaaa')
<_sre.SRE_Match object; span=(0, 5), match='aaaaa'>

>>> p.search('11aaaa')
<_sre.SRE_Match object; span=(2, 6), match='aaaa'>

>>> p.search('aaa11aaa')
<_sre.SRE_Match object; span=(0, 3), match='aaa'>

>>> p.search('1aaa11aaa1')
<_sre.SRE_Match object; span=(1, 4), match='aaa'>
```

re Python 정규표현식 모듈

- findall: 모든 매치를 찾아 리스트로 반환
  - 문자열 내에서 일치하는 모든 패턴을 찾아 리스트로 반환

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

>>> p.findall('11aaa')
['aaa']

>>> p.findall('1a1a1a1a1a')
['a', 'a', 'a', 'a', 'a']

>>> p.findall('1aa1aaa1a1aa1aaa')
['aa', 'aaa', 'a', 'aa', 'aaa']
```

- finditer: 모든 매치를 찾아 반복가능 객체로 반환

```
>>> p = re.compile('[a-z]+'))
>>> p.finditer('a1bb1ccc')
<callable_iterator object at 0x7f850c4285f8>
```

re Python 정규표현식 모듈

- finditer: 모든 매치를 찾아 반복가능 객체로 반환

```
>>> p = re.compile('[a-z]+')
>>> p.finditer('a1bb1ccc')
<callable_iterator object at 0x7f850c4285f8>

>>> f_iter = p.finditer('a1bb1ccc')
>>> for i in f_iter:
...         print(i)
<_sre.SRE_Match object; span=(0, 1), match='a'>
<_sre.SRE_Match object; span=(2, 4), match='bb'>
<_sre.SRE_Match object; span=(5, 8), match='ccc'>
```

re Python 정규표현식 모듈

- 매치 객체의 메소드
  - 패턴 객체의 메서드를 통해 리턴된 매치 객체 정보

```
<_sre.SRE_Match object;
        span=(매치 시작지점 인덱스, 매치 끝지점 인덱스),
                                        match='매치된 문자열'>
```

  - 매치 객체는 내부 정보에 접근할 수 있는 네 가지 메서드를 제공

메서드	기능
group()	매치된 문자열 출력
start()	매치 시작지점 인덱스 출력
end()	매치 끝지점 인덱스 출력
span()	(start(), end())를 튜플로 출력

re Python 정규표현식 모듈

- 컴파일 옵션
  - 정규표현식을 컴파일 할 때 옵션을 지정

```
>>> 변수이름 = re.compile('정규표현식', re.옵션)
```

  - DOTALL, S
    - .은 줄바꿈 문자 ₩n 를 제외한 모든 것과 매치
    - 컴파일 할 때 re.DOTALL 또는 re.S 옵션을 넣어주면 ₩n 까지 매치

```
>>> p = re.compile('.') # 옵션 없음
>>> result = p.findall('1a\nbc')
>>> print(result)
['1', 'a', 'b', 'c'] # \n이 매치되지 않음

>>> p = re.compile('.', re.DOTALL) # re.DOTALL 옵션 추가
>>> result = p.findall('1a\nbc')
>>> print(result)
['1', 'a', '\n', 'b', 'c'] # \n까지 매치
```

re Python 정규표현식 모듈

- IGNORECASE, I
  - re.IGNORECASE 또는 re.I 옵션을 넣어주면 대소문자를 구별하지 않고 매치

```
>>> p = re.compile('[a-z]') # 소문자만 매치
>>> result = p.findall('aAbB')
>>> print(result)
['a', 'b']

>>> p = re.compile('[a-z]', re.IGNORECASE) # re.IGNORECASE 옵션 추가
>>> result = p.findall('aAbB')
>>> print(result)
['a', 'A', 'b', 'B'] # 소문자와 대문자 모두 매치
```

re Python 정규표현식 모듈

- MULTILINE, M
  - re.MULTILINE 또는 re.M 옵션을 넣어주면 여러 줄의 문자열에 ^ 와 $ 를 적용

```
>>> text = '''student-1-name: James
... student-2-name: John
... student-3-name: Jordan
... teacher-1-name: Mike
... student-5-name: John'''
>>> p = re.compile('^student.*') # student로 시작하는 문자열 매치
>>> result = p.findall(text)
>>> print(result)
['student-1-name: James'] # 첫 줄만 매치

>>> p = re.compile('^student.*', re.MULTILINE) # re.MULTILINE 옵션 추가
>>> result = p.findall(text)
>>> print(result)
['student-1-name: James', 'student-2-name: John', 'student-3-name:
Jordan', 'student-5-name: John']
# student로 시작하는 모든 줄이 매치
```

re Python 정규표현식 모듈


```
>>> text = '''student-1-name: James
... student-2-name: John
... student-3-name: Jordan
... teacher-1-name: Mike
... student-5-name: John'''
>>> p = re.compile('.*John$') # John으로 끝나는 문자열 매치
>>> result = p.findall(text)
>>> print(result)
['student-5-name: John'] # 가장 마지막 줄만 매치

>>> p = re.compile('.*John$', re.MULTILINE) # re.MULTILINE 옵션 추가
>>> result = p.findall(text)
>>> print(result)
['student-2-name: John', 'student-5-name: John']
# John으로 끝나는 모든 줄이 매치
```

re Python 정규표현식 모듈

- VERBOSE, X
  - re.VERBOSE 또는 re.X 옵션을 주면 좀 더 가독성 좋게 정규표현식을 작성
  - re.VERBOSE 옵션을 추가하면 정규표현식에 컴파일시 자동으로 제거되는 공백과 코멘트를 추가할 수 있음 (단, [] 안에 입력된 공백문자 제외)
  - 아래의 두 표현식은 동일하게 작동

```
>>> p = re.compile(r'&[#](0[0-7]+|[0-9]+|x[0-9a-fA-F]+);')

>>> p = re.compile(r"""
... &[#]                # Start of a numeric entity reference
...  (
...      0[0-7]+        # Octal form
...    | [0-9]+         # Decimal form
...    | x[0-9a-fA-F]+   # Hexadecimal form
...  )
...  ;                  # Trailing semicolon
... """, re.VERBOSE) # re.VERBOSE 옵션 추가
```

re Python 정규표현식 모듈

❖ Python 정규표현식의 ₩ 문제

- Python에서 정규표현식에 ₩ 을 사용할 때 아래와 같은 문제가 발생

```
>>> text = '\section'
>>> result = re.match('\\section', text)
>>> print(result)
None
```

- ₩s는 공백문자를 뜻하는 메타문자로 인식되기 때문에 ₩section 이라는 문자열을 찾으려면 위해 이스케이프 코드 ₩₩를 사용한 ₩₩section 를 정규표현식에 입력
  - 그러나 Python 정규식 엔진에서 리터럴 규칙에 의해 ₩₩ 가 ₩로 해석되어 전달
  - 따라서 ₩₩ 를 문자 그대로 넘겨주어야 하는데 이를 위해서는 ₩₩₩₩로 입력

```
>>> text = '\section'
>>> result = re.match('\\\\section', text)
>>> print(result)
<_sre.SRE_Match object; span=(0, 8), match='\\section'>
```

re Python 정규표현식 모듈

- 가독성이 심각하게 저하되는데 이를 방지하기 위해 다음과 같이 작성
  - 정규표현식 앞의 r은 raw string

```
>>> text = '\section'
>>> result = re.match(r'\\section', text)
>>> print(result)
<_sre.SRE_Match object; span=(0, 8), match='\\section'>
```