

I 4. Python 정규 표현식

1. Python 정규표현식(Regular Expression)

1 Regular Expression

- Text를 검색할때 Ctrl+F로 찾는것을 조금 더 발전시킨 형태
- 글자 자체가 아니라 Pattern을 사용

-정규식 사용 사례

- ① 입력유효성 Check (e-Mail주소 맞는지)
- ② Text에서 특정부분 추출(우편번호등..)
- ③ 특정 Text 바꾸기 (사람->남자 또는 여자)
- ④ Text 쪼개기

2. 자주 사용하는 문자 클래스

[0-9] 또는 [a-zA-Z] 등은 무척 자주 사용하는 정규 표현식.

이렇게 자주 사용하는 정규식은 별도의 표기법으로 표현.

`\d` - 숫자와 매치, [0-9]와 동일한 표현식이다.

`\D` - 숫자가 아닌 것과 매치, [^0-9]와 동일한 표현식이다.

`\s` - whitespace 문자와 매치, [\t\n\r\f\v]와 동일한 표현식이다. 맨 앞의 빈 칸은 공백문자(space)를 의미.

`\S` - whitespace 문자가 아닌 것과 매치, [^\t\n\r\f\v]와 동일한 표현식.

`.` - \n을 제외한 모든 문자 표현식.

`\w` - 문자+숫자(alphanumeric)와 매치, [a-zA-Z0-9_]와 동일한 표현식.

`\W` - 문자+숫자(alphanumeric)가 아닌 문자와 매치, [^a-zA-Z0-9_]와 동일한 표현식.

대문자로 사용된 것은 소문자의 반대임을 추측

3. Dot(.)

정규 표현식의 Dot(.) 메타 문자는 줄바꿈 문자인 \n을 제외한 모든 문자와 매치됨을 의미

예시) a.b

"a + 모든문자 + b" 즉 a와 b라는 문자 사이에 어떤 문자가 들어가도 모두 매치된다는 의미

4. 반복 (*)

예시) ca*t

이 정규식에는 반복을 의미하는 * 메타 문자가 사용.

여기에서 사용한 *은 * 바로 앞에 있는 문자 a가 0부터 무한대로 반복될 수 있다는 의미

1. Python 정규표현식(Regular Expression)

5. 반복 (+) Regular Expression

- 반복을 나타내는 또 다른 메타 문자로 +가 있다. +는 최소 1번 이상 반복될 때 사용한다.
- 즉 *가 반복 횟수 0부터라면 +는 반복 횟수 1부터 사용
- 예시) `ca+t` : "c + a(1번 이상 반복) + t"

6. 반복 ({m,n}, ?)

여기에서 잠깐 생각해 볼 게 있다. 반복 횟수를 3회만 또는 1회부터 3회까지만으로 제한할때 사용

{ } 메타 문자를 사용하면 반복 횟수를 고정할 수 있다. {m, n} 정규식을 사용하면 반복 횟수가 m부터 n까지 매치할 수 있다. 또한 m 또는 n을 생략할 수도 있다. 만약 {3,}처럼 사용하면 반복 횟수가 3 이상인 경우이고 {,3}처럼 사용하면 반복 횟수가 3 이하를 의미

예시1) `ca{2}t` → "c + a(반드시 2번 반복) + t"

-예시2) `ca{2,5}t` → "c + a(2~5회 반복) + t"

7. ?

- 반복은 아니지만 이와 비슷한 개념으로 ? 이 있다. ? 메타문자가 의미하는 것은 {0, 1} 이다.
- 예시) `ab?c` → "a + b(있어도 되고 없어도 된다) + c"

1. Python 정규표현식(Regular Expression)

8. <https://regex101.com>

Regular Expression 입력 → `\b(https?:\w/\w/?)([\w.]{1,2})(\w.[\w]{2,4}){1,2}\b`

test String 입력 →

`http://naver.com`

`https://www.naver.com`

`http://shop.site.co.kr`

<http://www.daum.net>

1) /문자 표현시 Escape를 사용(\)

2) `\b` : 바운더리 표현, word boundary를 표현하며 문자와 공백사이의 문자를 의미

예시) 주로, `\b`를 단어 시작과 마지막에 붙여 정규 표현식에서 찾고자 하는 단어와 완전히 일치시킬 목적으로 사용.

[정규 표현식] `\bcat\b`

[적용문장] The **cat** scattered his food all over the room.

[결과] 우리가 찾으려 했던 것은 정확히 "cat"과 일치하는 단어.

scattered의 cat 앞뒤로는 각각 's'와 't'이므로 공백이 아니어서 일치 대상에서 제외

3) `()` : 괄호 사용 Group을 적절히 활용

4) `.`(dot) 문자 : 와일드카드, `\n` 문자 제외하고 모든 문자 대응

실제로 `.`를 표시하고 싶으면 `\.`으로 사용

5) `^` : 시작 표시

6) `$` : 끝을 표시

7) `https?` -> `http` 나 `https` 둘다 가능

8) `(https?:\w/\w/?)` --> `http` 나 `https://` 전체가 있을수도 없을수도 있음

9) `[\w.]{1,2}` --> 문자+숫자(alphanumeric)와 매치, +는 여러번 발생

10) `{1,2}` 위 9)번이 한번 또는 두번 반복

2. Python에서 정규표현식(Regular Expression) 지원하는 re 모듈

1. 기본개념

- 파이썬은 정규 표현식을 지원하기 위해 re(regular expression의 약어) 모듈을 제공.
- re 모듈은 파이썬을 설치할 때 자동으로 설치되는 기본 라이브러리로 사용 방법은 다음과 같음 .

import re

p=re.compile('ab*') # pattern 을 미리 만들어 둠

2. 정규식을 이용한 문자열 검색

- 1) match() : 문자열의 처음부터 정규식과 매치되는지 조사
- 2) search() : 문자열 전체를 검색하여 정규식과 매치되는지 조사 , 첫번째 일치하는 객체 반환
- 3) findall() : 정규식과 매치되는 모든 문자열(substring)을 **리스트**로 돌려준다.
- 4) finditer() : 정규식과 매치되는 모든 문자열(substring)을 **반복 가능한 객체**로 돌려준다
 - match, search는 정규식과 매치될 때는 match 객체를 돌려주고, 매치되지 않을 때는 None을 돌려준다
- 5) sub() : 바꾸기 기능 , 일치 부분 대체인자로 변경

예시)

```
>>> import re
```

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

match 메서드는 문자열의 처음부터 정규식과 매치되는지 조사한다. 위 패턴에 match 메서드를 수행

```
>>> m = p.match("python")
```

```
>>> print(m)
```

"python" 문자열은 [a-z]+ 정규식에 부합되므로 match 객체를 돌려준다.

```
>>> m = p.match("3 python")
```

```
>>> print(m)
```

None

3 python" 문자열은 처음에 나오는 문자 3이 정규식 [a-z]+에 부합되지 않으므로 None을 돌려준다.

2. Python에서 정규표현식(Regular Expression) 지원하는 re 모듈

예시 계속)

```
>>> import re
```

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

```
>>> result = p.findall("life is too short")
```

```
>>> print(result)
```

['life', 'is', 'too', 'short'] # "life is too short" 문자열의 'life', 'is', 'too', 'short' 단어를 각각 [a-z]+ 정규식과 매치해서 리스트로 돌려준다

3. Python에서 정규표현식(Regular Expression) match 객체

1. match 객체의 메서드

- 1) group() : 매치된 문자열을 돌려준다.
- 2) start() : 매치된 문자열의 시작 위치를 돌려준다.
- 3) end() : 매치된 문자열의 끝 위치를 돌려준다.
- 4) span() : 매치된 문자열의 (시작, 끝)에 해당하는 튜플을 돌려준다

예시) >>> m = p.match("python")

>>> m.group()

'python'

>>> m.start()

0

>>> m.end()

6

>>> m.span()

(0, 6).

4. Python에서 정규표현식 컴파일 옵션1

1. match 객체의 메서드

- 1) DOTALL(S) - . 이 줄바꿈 문자를 포함하여 모든 문자와 매치할 수 있도록 한다.
- 2) IGNORECASE(I) - 대소문자에 관계없이 매치할 수 있도록 한다.
- 3) MULTILINE(M) - 여러줄과 매치할 수 있도록 한다. (^, \$ 메타문자의 사용과 관계가 있는 옵션이다)
- 4) VERBOSE(X) - verbose 모드를 사용할 수 있도록 한다. (정규식을 보기 편하게 만들수 있고 주석등을 사용할 수 있게된다.)
 - 옵션을 사용할 때는 re.DOTALL처럼 전체 옵션 이름을 써도 되고 re.S처럼 약어를 써도 된다.

예시)

```
>>> import re
>>> p = re.compile('a.b')
>>> m = p.match('aWnb')
>>> print(m)
```

None

정규식이 a.b인 경우 문자열 aWnb는 매치되지 않음을 알 수 있다.

왜냐하면 W는 . 메타 문자와 매치되지 않기 때문이다.

Wn 문자와도 매치되게 하려면 다음과 같이

re.DOTALL 옵션을 사용

예시 DOTALL, S)

```
>>> p = re.compile('a.b', re.DOTALL)
>>> m = p.match('aWnb')
>>> print(m)
```

보통 re.DOTALL 옵션은 여러 줄로 이루어진 문자열에서 Wn에 상관없이 검색할 때 많이 사용

4. Python에서 정규표현식 컴파일 옵션2

2. match 객체의 메서드 IGNORECASE, I

re.IGNORECASE 또는 re.I 옵션은 대소문자 구별 없이 매치를 수행할 때 사용하는 옵션

예시)

```
>>> p = re.compile('[a-z]', re.I)
```

```
>>> m = p.match('python')
```

```
>>> print(m)
```

[a-z] 정규식은 소문자만을 의미하지만 re.I 옵션으로
대소문자 구별 없이 매치

4. Python에서 정규표현식 컴파일 옵션3

3. match 객체의 메서드 MULTILINE, M

re.MULTILINE 또는 re.M 옵션은 조금 후에 설명할 메타 문자인 ^, \$와 연관된 옵션

이 메타 문자에 대해 간단히 설명하자면 ^는 문자열의 처음을 의미하고, \$는 문자열의 마지막을 의미

예를 들어 정규식이 ^python인 경우 문자열의 처음은 항상 python으로 시작해야 매치되고, 만약 정규식이 python\$이라면 문자열의 마지막은 항상 python으로 끝나야 매치된다는 의미

```
import re
p = re.compile("^pythonWsWw+")
data = """python one
life is too short
python two
you need python
python three"""
print(p.findall(data))
```

정규식 ^pythonWsWw+은 python이라는 문자열로 시작하고 그 뒤에 whitespace, 그 뒤에 단어가 와야 한다는 의미이다. 검색할 문자열 data는 여러 줄로 이루어져 있다. 이 스크립트를 실행하면 다음과 같은 결과를 돌려준다.

```
['python one']
```

```
import re
p = re.compile("^pythonWsWw+", re.MULTILINE)
data = """python one
life is too short
python two
you need python
python three"""

print(p.findall(data))
```

re.MULTILINE 옵션으로 인해 ^ 메타 문자가 문자열 전체가 아닌 **각 줄의 처음**이라는 의미를 갖게 되었다. 이 스크립트를 실행하면 다음과 같은 결과가 출력

```
['python one', 'python two', 'python three']
```

4. Python에서 정규표현식 컴파일 옵션4

4. match 객체의 메서드 VERBOSE, X


지금껏 알아본 정규식은 매우 간단하지만 정규식 전문가들이 만든 정규식을 보면 거의 암호수준 정규식을 이해하려면 하나하나 조심스럽게 뜯어보아야만 한다. 이렇게 이해하기 어려운 정규식을 주석 또는 줄 단위로 구분할 수 있다면 얼마나 보기 좋고 이해하기 쉬울까?

방법이 있다.

바로 re.VERBOSE 또는 re.X 옵션을 사용

```
email_regex = re.compile(r'''[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+
                              (\.[a-zA-Z]{2,4}){1,2} ''')
```

위 정규식이 쉽게 이해되는가? 이제 다음 예를 보자
첫 번째와 두 번째 예를 비교해 보면 컴파일된 패턴 객체인 email_regex 는 모두 동일한 역할을 한다.
하지만 정규식이 복잡할 경우 두 번째처럼 주석을 적고 여러 줄로 표현하는 것이 훨씬 가독성이 좋다는 것을 알 수 있다.
re.VERBOSE 옵션을 사용하면 문자열에 사용된 whitespace는 컴파일할 때 제거된다(단 [] 안에 사용한 whitespace는 제외).
그리고 줄 단위로 #기호를 사용하여 주석문을 작성



```
email_regex = re.compile(r'''(
    [a-zA-Z0-9._%+-]+      # username
    @                      # @ symbol
    [a-zA-Z0-9.-]+        # domain name
    (\.[a-zA-Z]{2,4}){1,2} # dot-something
    )''', re.VERBOSE)
```

<https://regex101.com> 참조

<https://regexr.com> 참조