정규표현식

7장

정규표현식을 쓰는 이유

-> 훨씬 간편하고 직관적으로 코드 작성 가능

```
data = """
park 800905-1049118
kim  700905-1059119
"""


result = []
for line in data.split("\n"):
    word_result = []
    for word in line.split(" "):
        if len(word) == 14 and word[:6].isdigit() and word[7:].isdigit():
            word = word[:6] + "-" + "*******"
        word_result.append(word)
    result.append(" ".join(word_result))
print("\n".join(result))
```

```
import re

data = """
박ㅇㅇ 990905-1049118
김ㅇㅇ 920905-1059119
"""


pat = re.compile("(₩d{6})[-]₩d{7}")
print(pat.sub("₩g<1>-*******", data))
```

```
박ㅇㅇ 990905-*******
김ㅇㅇ 920905-*******
```

01. 정규표현식과 메타 문자

*메타 문자

특별한 용도로 사용되는 문자

예) . ^ $ + ? [] ₩ | ()

**정규표현식

복잡한 문자열을 처리할 때 사용

파이썬뿐만 아니라 문자열을 처리하는

모든 곳에서 사용 가능

1.1> 문자 클래스 [ ]

-> [ ]안에 있는 문자가 문자열에 포함되어 있으면 mach

정규식	문자열	매치 여부
[abc]	a	O
	before	O
	dude	X

** 하이픈(-) 사용하여 범위 표현 가능

예) [abc] => [a-c] [012345] => [0-5]

** ^는 not을 의미함 예) [0-9] = 숫자 / [^0-9] = 숫자가 아닌 문자만 mach

1.2> 자주 사용하는 문자 클래스

-> [0-9], [a-zA-Z] 등은 굉장히 자주 사용

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

** 대문자로 사용된 것은 소문자의 반대 의미를 가짐

** 그 외에도 | ^ $ \A \Z가 있음

|

패턴 안에서 or 연산을 수행할 때 사용

^, $

- ^: 시작
  - ^who : who로 시작하는 것들
- $: 끝
  - end$: end로 끝나는 것들

\A

- 시작점
- cf)^ : multi line

\Z

- 마지막점
- cf)$: multi line

02> . (Dot)

-> 줄바꿈 문자 \n을 제외한 모든 문자와 매치

정규식	문자열	매치 여부
a.b	aab	O
	a0b	O
	abc	X

** a와 b사이에 어떤 문자가 와도 가능

03> 반복

*	앞의 문자가 0번 이상 반복될 때 매치	
+	앞의 문자가 한번 이상 반복될 때 매치	
{m,n}	{m}	반드시 m번 반복될 때만 매치
	{m,n}	앞의 문자가 m번에서 n번 반복될 때 매치
	?	앞의 문자가 없거나 하나 있을 때 매치

- 10*1 -> 11, 1001
- 10+1 -> 11, 1001
- 10{3}1 -> 10001
- 10{3,5}1 -> 11, 101
- 10?1 -> 11, 1001

04> 파이썬에서 정규 표현식을 지원하는 re모듈

-> re모듈을 통해 정규표현식을 사용

```
import re
```

Method	목적
match()	문자열의 처음부터 정규식과 매치되는지 조사한다.
search()	문자열 전체를 검색하여 정규식과 매치되는지 조사한다.
findall()	정규식과 매치되는 모든 문자열(substring)을 리스트로 돌려준다.
finditer()	정규식과 매치되는 모든 문자열(substring)을 반복 가능한 객체로 돌려준다.

04> 파이썬에서 정규 표현식을 지원하는 re모듈

compile 정규표현식 컴파일

`re.compile()` 명령을 통해 정규표현식을 컴파일하여 변수에 저장한 후 사용할 수 있다.

```
변수이름 = re.compile('정규표현식')
```

정규표현식을 컴파일하여 변수에 할당한 후 타입을 확인해보면 `_sre.SRE_Pattern` 이라는 이름의 클래스 객체인 것을 볼 수 있다.

```
p = re.compile('[abc]')

print(type(p))
```

4.1> 패턴 객체의 메서드

match: 시작부터 일치하는 패턴 찾기

문자열의 처음 시작부터 검색하여 일치하지 않는 부분이 나올 때까지 찾는다.

```
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'>
```

search: 전체 문자열에서 첫 번째 매치 찾기

문자열 전체에서 검색하여 처음으로 매치되는 문자열을 찾는다.

```
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'>
```

4.1> 패턴 객체의 메서드

findall: 모든 매치를 찾아 리스트로 반환

문자열 내에서 일치하는 모든 패턴을 찾아 리스트로 반환한다.

```
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.finditer('a1bb1ccc')
<callable_iterator object at 0x7f850c4285f8>
```

`callable_iterator` 라는 객체가 반환되었다. `for` 을 사용하여 하나씩 출력해보자.

```
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'>
```

반복가능 객체는 각 매치의 결과인 매치 객체를 포함하고 있다.

05> mach 객체의 메서드

패턴 객체의 메서드를 통해 리턴된 매치 객체는 아래와 같은 정보를 담고 있다.

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

** 매치 객체는 내부 정보에 접근할 수 있는 4가지 메서드를 제공

method	목적
group()	매치된 문자열을 돌려준다.
start()	매치된 문자열의 시작 위치를 돌려준다.
end()	매치된 문자열의 끝 위치를 돌려준다.
span()	매치된 문자열의 (시작, 끝)에 해당하는 튜플을 돌려준다.

05> mach 객체의 메서드

```
p = re.compile('[a-z]+')
result = p.search('1aaa11aaa1')
print(result)
```

위의 코드를 실행하면 아래의 매치 오브젝트를 얻는다.

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

매치 객체의 메서드를 실행한 결과는 아래와 같다.

```
result.group()
aaa

result.start()
1

result.end()
4

result.span()
(1, 4)
```

06>컴파일 옵션

정규표현식을 컴파일 할 때 옵션을 지정해줄 수 있다.

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

- DOTALL(S) - `.` 이 줄바꿈 문자를 포함하여 모든 문자와 매치할 수 있도록 한다.
- IGNORECASE(I) - 대소문자에 관계없이 매치할 수 있도록 한다.
- MULTILINE(M) - 여러줄과 매치할 수 있도록 한다. (`^`, `$` 메타문자의 사용과 관계가 있는 옵션이다)
- VERBOSE(X) - verbose 모드를 사용할 수 있도록 한다. (정규식을 보기 편하게 만들수 있고 주석등을 사용할 수 있게된다.)

07>백슬래시 문제

-> 정규표현식에 `\` 를 사용할 경우 ₩s 로 인식하는 문제가 발생

```
text = '\section'
result = re.match('\\section', text)
print(result)
```

```
None
```

-> ₩s는 공백문자를 뜻하는 메타문자로

₩section 이라는 문자열을 찾으려면

`\\` 를 사용한 `\\section` 을 입력해야 함

07>백슬래시 문제

```
text = '\section'
result = re.match('\\\\section', text)
print(result)
```

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

가독성이 심각하게 저하되는데 이를 방지하기 위해 다음과 같이 작성한다.

```
text = '\section'
result = re.match(r'\\section', text) # raw string을 뜻하는 r을
print(result)
```

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

정규표현식 앞에 `r` 은 항상 붙여주는 것이 권장된다.