

07장 파이썬 정규표현식과 XML



정규 표현식과 XML은 한 번
익혀두면 두고두고 써먹을 수
있는 것들



07-1 정규 표현식 살펴보기

주민등록번호의 뒷자리를 * 문자로 변경하시오 1

```
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))
```



07-1 정규 표현식 살펴보기

주민등록번호의 뒷자리를 * 문자로 변경하시오 2

```
import re

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

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



07-2 정규 표현식 시작하기

문자 클래스 []

[abc]

- "a"는 정규식과 일치하는 문자인 "a"가 있으므로 매치
- "before"는 정규식과 일치하는 문자인 "b"가 있으므로 매치
- "dude"는 정규식과 일치하는 문자인 a, b, c 중 어느 하나도 포함하고 있지 않으므로 매치되지 않음



07-2 정규 표현식 시작하기

Dot(.)

```
a.b
```

- "aab"는 가운데 문자 "a"가 모든 문자를 의미하는 `.`과 일치하므로 정규식과 매치.
- "a0b"는 가운데 문자 "0"가 모든 문자를 의미하는 `.`과 일치하므로 정규식과 매치
- "abc"는 "a"문자와 "b"문자 사이에 어떤 문자라도 하나는있어야 하는 이 정규식과 일치하지 않으므로 매치되지 않는다



07-2 정규 표현식 시작하기

반복 (*)

```
ca*t
```

- "ct"는 "a"가 0번 반복되어 매치
- "cat"는 "a"가 0번 이상 반복되어 매치 (1번 반복)
- "caaat"는 "a"가 0번 이상 반복되어 매치 (3번 반복)



07-2 정규 표현식 시작하기

반복 (+)

```
ca+t
```

- "ct"는 "a"가 0번 반복되어 매치되지 않음
- "cat"는 "a"가 1번 이상 반복되어 매치 (1번 반복)
- "caaat"는 "a"가 1번 이상 반복되어 매치 (3번 반복)



07-2 정규 표현식 시작하기

반복 ($\{m,n\}$, ?)

```
ca{2}t
```

- "cat"는 "a"가 1번만 반복되어 매치되지 않음
- "caat"는 "a"가 2번 반복되어 매치



07-2 정규 표현식 시작하기

반복 ($\{m,n\}$, $?$)

```
ca{2,5}t
```

- "cat"는 "a"가 1번만 반복되어 매치되지 않음
- "caat"는 "a"가 2번 반복되어 매치
- "caaaaat"는 "a"가 5번 반복되어 매치



07-2 정규 표현식 시작하기

반복 ($\{m,n\}$, ?)

`ab?c`

- "abc"는 "b"가 1번 사용되어 매치
- "ac"는 "b"가 0번 사용되어 매치



07-2 정규 표현식 시작하기

파이썬에서 정규 표현식을 지원하는 re 모듈

```
>>> import re  
>>> p = re.compile('ab*')
```



07-2 정규 표현식 시작하기

match

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

```
>>> m = p.match("python")
>>> print(m)
<_sre.SRE_Match object at 0x01F3F9F8>
```

```
>>> m = p.match("3 python")
>>> print(m)
None
```



07-2 정규 표현식 시작하기

search

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

```
>>> m = p.search("python")  
>>> print(m)  
<_sre.SRE_Match object at 0x01F3FA68>
```

```
>>> m = p.search("3 python")  
>>> print(m)  
<_sre.SRE_Match object at 0x01F3FA30>
```



07-2 정규 표현식 시작하기

findall

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

```
>>> result = p.findall("life is too short")  
>>> print(result)  
['life', 'is', 'too', 'short']
```



07-2 정규 표현식 시작하기

finditer

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

```
>>> result = p.finditer("life is too short")
>>> print(result)
<callable_iterator object at 0x01F5E390>
>>> for r in result: print(r)
...
<_sre.SRE_Match object at 0x01F3F9F8>
<_sre.SRE_Match object at 0x01F3FAD8>
<_sre.SRE_Match object at 0x01F3FAA0>
<_sre.SRE_Match object at 0x01F3F9F8>
```



07-2 정규 표현식 시작하기

match 객체의 메서드 1

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



07-2 정규 표현식 시작하기

match 객체의 메서드 2

```
>>> m = p.search("3 python")
>>> m.group()
'python'
>>> m.start()
2
>>> m.end()
8
>>> m.span()
(2, 8)
```



07-2 정규 표현식 시작하기

DOTALL, S

```
>>> import re
>>> p = re.compile('a.b')
>>> m = p.match('a\nb')
>>> print(m)
None
```

```
>>> p = re.compile('a.b', re.DOTALL)
>>> m = p.match('a\nb')
>>> print(m)
<_sre.SRE_Match object at 0x01FCF3D8>
```



07-2 정규 표현식 시작하기

IGNORECASE, I

```
>>> p = re.compile('[a-z]', re.I)
>>> p.match('python')
<_sre.SRE_Match object at 0x01FCFA30>
>>> p.match('Python')
<_sre.SRE_Match object at 0x01FCFA68>
>>> p.match('PYTHON')
<_sre.SRE_Match object at 0x01FCF9F8>
```



07-2 정규 표현식 시작하기

MULTILINE, M

```
import re
p = re.compile("^python\s\w+", re.MULTILINE)

data = """python one
life is too short
python two
you need python
python three"""

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

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




07-2 정규 표현식 시작하기

백슬래시 문제

```
\section
```

```
>>> p = re.compile('\\\\section')
```

```
>>> p = re.compile(r'\\section')
```



07-3 강력한 정규 표현식의 세계로

메타문자 |

```
>>> p = re.compile('Crow|Servo')
>>> m = p.match('CrowHello')
>>> print(m)
<_sre.SRE_Match object; span=(0, 4), match='Crow'>
```



07-3 강력한 정규 표현식의 세계로

메타문자 ^

```
>>> print(re.search('^Life', 'Life is too short'))  
<_sre.SRE_Match object at 0x01FCF3D8>  
>>> print(re.search('^Life', 'My Life'))  
None
```




07-3 강력한 정규 표현식의 세계로

메타문자 \$

```
>>> print(re.search('short$', 'Life is too short'))
<_sre.SRE_Match object at 0x01F6F3D8>
>>> print(re.search('short$', 'Life is too short, you need
python'))
None
```



07-3 강력한 정규 표현식의 세계로

메타문자 `\b`

```
>>> p = re.compile(r'\bclass\b')
>>> print(p.search('no class at all'))
<_sre.SRE_Match object at 0x01F6F3D8>
```

```
>>> print(p.search('the declassified algorithm'))
None
```

```
>>> print(p.search('one subclass is'))
None
```



07-3 강력한 정규 표현식의 세계로

그룹핑 1

`(ABC)+`

```
>>> p = re.compile('(ABC)+')
>>> m = p.search('ABCABCABC OK?')
>>> print(m)
<_sre.SRE_Match object at 0x01F7B320>
>>> print(m.group())
ABCABCABC
```



07-3 강력한 정규 표현식의 세계로

그룹핑 2

```
>>> p = re.compile(r"(\w+)\s+\d+[-]\d+[-]\d+")
>>> m = p.search("park 010-1234-1234")
>>> print(m.group(1))
park
```



07-3 강력한 정규 표현식의 세계로

그룹핑된 문자열 재참조하기

```
>>> p = re.compile(r'(\b\w+)\s+\1')
>>> p.search('Paris in the the spring').group()
'the the'
```



07-3 강력한 정규 표현식의 세계로

그룹핑된 문자열에 이름 붙이기

```
(?P<name>\w+)\s+((\d+)[-]\d+[-]\d+)
```

```
>>> p = re.compile(r"(?P<name>\w+)\s+((\d+)[-]\d+[-]\d+)")
>>> m = p.search("park 010-1234-1234")
>>> print(m.group("name"))
park
```

```
>>> p = re.compile(r'(?P<word>\b\w+)\s+(?P=word)')
>>> p.search('Paris in the the spring').group()
'the the'
```



07-3 강력한 정규 표현식의 세계로

긍정형 전방 탐색

```
>>> p = re.compile(".*(?=)")
>>> m = p.search("http://google.com")
>>> print(m.group())
http
```



07-3 강력한 정규 표현식의 세계로

부정형 전방 탐색

```
.*[.](^[b].??.?|.^[a]??.?|..?[^t]?)$
```

```
.*[.](?!bat$).*
```

```
.*[.](?!bat$|exe$).*
```




07-3 강력한 정규 표현식의 세계로

문자열 바꾸기

```
>>> p = re.compile('(blue|white|red)')
>>> p.sub('colour', 'blue socks and red shoes')
'colour socks and colour shoes'
```

```
>>> p.sub('colour', 'blue socks and red shoes', count=1)
'colour socks and red shoes'
```



07-3 강력한 정규 표현식의 세계로

Greedy vs Non-Greedy

```
>>> s = '<html><head><title>Title</title>'  
>>> len(s)  
32  
>>> print(re.match('<.*>', s).span())  
(0, 32)  
>>> print(re.match('<.*>', s).group())  
<html><head><title>Title</title>
```

```
>>> print(re.match('<.*?>', s).group())  
<html>
```



07-4 파이썬으로 XML 처리하기

XML 문서 생성하기 1

```
<note date="20120104">  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>
```



07-4 파이썬으로 XML 처리하기

XML 문서 생성하기 2

```
from xml.etree.ElementTree import Element, dump

note = Element("note")
to = Element("to")
to.text = "Tove"

note.append(to)
dump(note)
```

```
<note><to>Tove</to></note>
```



07-4 파이썬으로 XML 처리하기

XML 문서 생성하기 3

```
from xml.etree.ElementTree import Element, SubElement, dump

note = Element("note")
to = Element("to")
to.text = "Tove"

note.append(to)
SubElement(note, "from").text = "Jani"
```

```
<note><to>Tove</to><from>Jani</from></note>
```



07-4 파이썬으로 XML 처리하기

XML 문서 생성하기 4

```
from xml.etree.ElementTree import Element, SubElement, dump

note = Element("note")
to = Element("to")
to.text = "Tove"

note.append(to)
SubElement(note, "from").text = "Jani"
note.attrib["date"] = "20120104"

dump(note)
```

```
<note date="20120104"><to>Tove</to><from>Jani</from></note>
```



07-4 파이썬으로 XML 처리하기

파일에 쓰기(write) 수행하기

```
from xml.etree.ElementTree import ElementTree  
ElementTree(note).write("note.xml")
```



07-4 파이썬으로 XML 처리하기

XML문서 파싱하기

```
from xml.etree.ElementTree import parse  
tree = parse("note.xml")  
note = tree.getroot()
```




07-4 파이썬으로 XML 처리하기

애트리뷰트 값 읽기

```
print note.get("date")  
print note.get("foo", "default")  
print note.keys()  
print note.items()
```

```
20120104  
default  
['date']  
[('date', '20120104')]
```



07-4 파이썬으로 XML 처리하기

XML 태그 접근하기

```
from_tag = note.find("from")  
from_tags = note.findall("from")  
from_text = note.findtext("from")
```

```
childs = note.getiterator()  
childs = note.getchildren()
```

```
for parent in tree.getiterator():  
    for child in parent:  
        ... work on parent/child tuple
```