세상의 속도를
따라잡고 싶다면

점프 투 파이썬

박응용 지음(위키독스 운영자)

이지스퍼블리싱

08

정규 표현식

08-1 정규 표현식 살펴보기

08-2 정규 표현식 시작하기

08-1 정규 표현식 살펴보기

▪ 정규 표현식은 왜 필요한가?

- **정규 표현식**(regular expressions)
  - 복잡한 문자열을 처리할 때 사용하는 기법
  - 파이썬만의 고유 문법이 아니라 문자열을 처리하는 모든 곳에서 사용

- 정규 표현식을 사용하면 좋은 예

```
주민등록번호를 포함하고 있는 텍스트가 있다. 이 텍스트에 포함된 모든 주민등록번호의 뒷자리를
* 문자로 변경해 보자.
```

08-1 정규 표현식 살펴보기

■ 정규 표현식은 왜 필요한가?

- 정규식을 사용하지 않는 풀이법
  1) 전체 텍스트를 공백 문자로 나눈다(split).
  2) 나뉜 단어가 주민등록번호 형식인지 조사한다.
  3) 단어가 주민등록번호 형식이라면 뒷자리를 *로 변환한다.
  4) 나뉜 단어를 다시 조립한다.

```
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))
```

전체 텍스트

공백 문자마다 나누기

나눈 단어 조립하기

실행 결과

```
park 800905-*******
kim  700905-*******
```

08-1 정규 표현식 살펴보기

▪ 정규 표현식은 왜 필요한가?

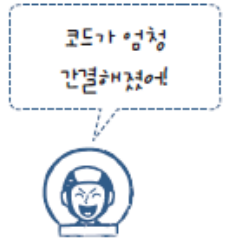
- 정규식을 사용한 풀이법
  - 정규 표현식을 사용하면 코드가 간결해짐

- 찾으려는 문자열 또는 바꾸어야 할 문자열의 규칙이 매우 복잡하다면 정규식의 효용은 더 커짐

```
import re  <-- 정규 표현식을 사용하기 위한 re 모듈

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

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



실행 결과

```
park 800905-*******
kim  700905-*******
```

08-2 정규 표현식 시작하기

정규 표현식의 기초, 메타 문자

- **메타 문자(meta characters)**
  - 원래 그 문자가 가진 뜻이 아닌 특별한 의미를 가진 문자

- 정규 표현식에 사용하는 메타 문자

```
. ^ $ * + ? {} [] \ | ()
```

  - 정규 표현식에 메타 문자를 사용하면 특별한 의미를 갖게 됨

08-2 정규 표현식 시작하기

▪ 정규 표현식의 기초, 메타 문자

- 문자 클래스(character class)
  - 메타 문자 : [ ]
  - 의미 : [ ] 사이의 문자들과 매치
  - [ ] 사이에는 어떤 문자도 들어갈 수 있음

- 예) [abc] : a, b, c 중 한 개의 문자와 매치

정규식	문자열	매치 여부	설명
[abc]	a	O	"a"는 정규식과 일치하는 문자인 "a"가 있으므로 매치된다.
	before	O	"before"는 정규식과 일치하는 문자인 "b"가 있으므로 매치된다.
	dude	X	"dude"는 정규식과 일치하는 문자인 a, b, c 중 어느 하나도 포함하고 있지 않으므로 매치되지 않는다.

08-2 정규 표현식 시작하기

▪ 정규 표현식의 기초, 메타 문자

- 문자 클래스(character class)
  - [ ] 안의 두 문자 사이에 하이픈(-)을 사용하면 두 문자 사이의 범위를 의미
  - 예) [a-c] : [abc]와 동일한 의미
  - 예) [0-5] : [012345]와 동일한 의미

  - 하이픈(-)을 사용한 문자 클래스의 사용 예)

    - [a-zA-Z]: 모든 알파벳
    - [0-9]: 모든 숫자

  - 단, 문자 클래스 안에 ^ 메타 문자를 사용할 경우에는 반대(not)라는 의미를 가짐!
    - 예) [^0-9] : 숫자가 아닌 문자만 매치됨

08-2 정규 표현식 시작하기

▪ 정규 표현식의 기초, 메타 문자

- .[dot] 문자 클래스 - ₩n을 제외한 모든 문자
  - 줄바꿈 문자인 ₩n을 제외한 모든 문자와 매치됨을 의미
  - 예)

```
a.b
```

    - a와 b라는 문자 사이에 어떤 문자가 들어가도 모두 매치된다는 의미

```
"a + 모든_문자 + b"
```

08-2 정규 표현식 시작하기

▪ 정규 표현식의 기초, 메타 문자

▪ .[dot] 문자 클래스 - ₩n을 제외한 모든 문자

▪ 정규식 a.b 살펴보기

정규식	문자열	매치 여부	설명
a.b	aab	O	"aab"는 가운데 문자 "a"가 모든 문자를 의미하는 .과 일치하므로 정규식과 매치된다.
	a0b	O	"a0b"는 가운데 문자 "0"가 모든 문자를 의미하는 .과 일치하므로 정규식과 매치된다.
	abc	X	"abc"는 "a"문자와 "b"문자 사이에 어떤 문자라도 하나는 있어야 하는 이 정규식과 일치하지 않으므로 매치되지 않는다.

08-2 정규 표현식 시작하기

▪ 정규 표현식의 기초, 메타 문자

- .[dot] 문자 클래스 - ₩n을 제외한 모든 문자
  - 혼동하기 쉬운 예) 문자 클래스 [ ] 안에 dot(.) 문자가 있는 경우

```
a[.]b
```

  - dot(.) 문자는 메타 문자가 아니라 '.' 문자 그대로를 의미함!

```
"a + . + b"
```

  - "a.b" 문자열은 매치됨
  - "a0b" 문자열과는 매치되지 않음

08-2 정규 표현식 시작하기

정규 표현식의 기초, 메타 문자

- *** 문자**
  - 반복을 의미하는 * 메타 문자
  - 0부터 무한대까지 반복될 때 사용
    - 실제로는 메모리 제한으로 무한대가 아닌 2억 개 정도만 가능
  - 예)

```
ca*t
```

    - * 바로 앞에 있는 문자 a가 0부터 무한대까지 반복될 수 있다는 의미

08-2 정규 표현식 시작하기

정규 표현식의 기초, 메타 문자

* 문자

- 정규식 ca*t 살펴보기

정규식	문자열	매치 여부	설명
ca*t	ct	Yes	"a"가 0번 반복되어 매치
ca*t	cat	Yes	"a"가 0번 이상 반복되어 매치(1번 반복)
ca*t	caaat	Yes	"a"가 0번 이상 반복되어 매치(3번 반복)

08-2 정규 표현식 시작하기

정규 표현식의 기초, 메타 문자

+ 문자

- 반복을 의미하는 + 메타 문자
- 최소 1번 이상 반복될 때 사용
- *가 반복 횟수가 0부터라면 +는 반복 횟수가 1부터

- 예)

```
ca+t
```

  - + 바로 앞에 있는 문자 a가 1부터 무한대까지 반복될 수 있다는 의미

```
"c + a가_1번_이상_반복 + t"
```

08-2 정규 표현식 시작하기

정규 표현식의 기초, 메타 문자

+ 문자

- 정규식 ca+t 살펴보기

정규식	문자열	매치 여부	설명
ca+t	ct	No	"a"가 0번 반복되어 매치되지 않음.
ca+t	cat	Yes	"a"가 1번 이상 반복되어 매치(1번 반복)
ca+t	caaat	Yes	"a"가 1번 이상 반복되어 매치(3번 반복)

08-2 정규 표현식 시작하기

정규 표현식의 기초, 메타 문자

{ } 문자와 ? 문자

- { } 메타 문자는 반복 횟수를 고정하는 의미
  - {m, n} 정규식 사용 시 반복 횟수가 m부터 n까지인 문자와 매치할 수 있음
  - m 또는 n은 생략 가능
    - 예) {3,} : 반복 횟수가 3 이상인 경우
    - 예) {,3} : 반복 횟수가 3 이하인 경우
    - 생략된 m은 0과 동일, 생략된 n은 무한대(2억 개 미만)를 의미함

08-2 정규 표현식 시작하기

▪ 정규 표현식의 기초, 메타 문자

▪ { } 문자와 ? 문자

1. {m}

- 정규식에 대한 매치 여부

정규식	문자열	매치 여부	설명
ca{2}t	cat	No	"a"가 1번만 반복되어 매치되지 않음.
ca{2}t	caat	Yes	"a"가 2번 반복되어 매치

08-2 정규 표현식 시작하기

▪ 정규 표현식의 기초, 메타 문자

▪ { } 문자와 ? 문자

2. {m, n}

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

→

"c + a를_2~5회_반복 + t"

- 정규식에 대한 매치 여부

정규식	문자열	매치 여부	설명
ca{2,5}t	cat	No	"a"가 1번만 반복되어 매치되지 않음.
ca{2,5}t	caat	Yes	"a"가 2번 반복되어 매치
ca{2,5}t	caaaaat	Yes	"a"가 5번 반복되어 매치

08-2 정규 표현식 시작하기

▪ 정규 표현식의 기초, 메타 문자

▪ { } 문자와 ? 문자

3. ?
   - {0, 1}을 의미하는 메타 문자

   - 정규식에 대한 매치 여부

정규식	문자열	매치 여부	설명
ab?c	abc	Yes	"b"가 1번 사용되어 매치
ab?c	ac	Yes	"b"가 0번 사용되어 매치

08-2 정규 표현식 시작하기

▪ 파이썬에서 정규 표현식을 지원하는 re 모듈

- re(regular expression) 모듈
  - 파이썬을 설치할 때 자동으로 설치되는 표준 라이브러리
  - 사용 방법

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

    - re.compile을 사용하여 정규 표현식(ab*)을 컴파일
    - re.compile의 리턴값을 객체 p(컴파일된 패턴 객체)에 할당해 그 이후의 작업을 수행

08-2 정규 표현식 시작하기

▪ 정규식을 이용한 문자열 검색

▪ 컴파일된 패턴 객체가 제공하는 4가지 메서드

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

▪ 패턴 예

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

08-2 정규 표현식 시작하기

▪ 정규식을 이용한 문자열 검색

▪ match

- 문자열의 처음부터 정규식과 매치되는지 조사

```
>>> m = p.match("python")
>>> print(m)
<re.Match object; span=(0, 6), match='python'>
```

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

- 파이썬 정규식 프로그램의 작성 흐름

```
p = re.compile(정규_표현식)
m = p.match('조사할 문자열')
if m:
    print('Match found: ', m.group())
else:
    print('No match')
```

08-2 정규 표현식 시작하기

정규식을 이용한 문자열 검색

search

- match 메서드를 수행했을 때와 동일하게 매치됨

```
>>> m = p.search("python")
>>> print(m)
<re.Match object; span=(0, 6), match='python'>
```

```
>>> m = p.search("3 python")
>>> print(m)
<re.Match object; span=(2, 8), match='python'>
```

- match 메서드와 search 메서드는 문자열의 처음부터 검색할지의 여부에 따라 다르게 사용해야 함

08-2 정규 표현식 시작하기

정규식을 이용한 문자열 검색

findall

- 패턴과 매치되는 모든 값을 찾아 리스트로 리턴

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

finditer

- Findall과 동일하지만, 그 결과로 반복 가능한 객체(iterator object)를 리턴

```
>>> result = p.finditer("life is too short")
>>> print(result)
<callable_iterator object at 0x01F5E390>
>>> for r in result: print(r)
...
<re.Match object; span=(0, 4), match='life'>
<re.Match object; span=(5, 7), match='is'>
<re.Match object; span=(8, 11), match='too'>
<re.Match object; span=(12, 17), match='short'>
```

08-2 정규 표현식 시작하기

match 객체의 메서드

- p.match, p.search 또는 p.finditer 메서드에 의해 리턴된 매치 객체(match object)를 의미
- match 객체의 메서드의 종류

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

```
>>> m = p.match("python")
>>> m.group()
'python'
>>> m.start()
0
>>> m.end()
6
>>> m.span()
(0, 6)
```

08-2 정규 표현식 시작하기

▪ 컴파일 옵션

- 정규식을 컴파일할 때 사용할 수 있는 옵션

옵션 이름	약어	설명
DOTALL	S	.(dot)이 줄바꿈 문자를 포함해 모든 문자와 매치될 수 있게 한다.
IGNORECASE	I	대소문자에 관계없이 매치될 수 있게 한다.
MULTILINE	M	여러 줄과 매치될 수 있게 한다. ^, $ 메타 문자 사용과 관계 있는 옵션이다.
VERBOSE	X	verbose 모드를 사용할 수 있게 한다. 정규식을 보기 편하게 만들 수 있고 주석 등을 사용할 수 있게 된다.

- 옵션을 사용할 때는 re.DOTALL처럼 전체 옵션 이름을 쓰거나 re.S처럼 약어 사용도 가능

08-2 정규 표현식 시작하기

컴파일 옵션

DOTALL, S

- 메타 문자를 줄바꿈 문자(₩n)와도 매치되게 할 때 사용하는 옵션

```
>>> p = re.compile('a.b', re.DOTALL)
>>> m = p.match('a\nb')
>>> print(m)
<re.Match object; span=(0, 3), match='a\nb'>
```

- re.DOTALL 옵션은 여러 줄로 이루어진 문자열에서 줄바꿈 문자에 상관없이 검색할 때 많이 사용함

08-2 정규 표현식 시작하기

컴파일 옵션

IGNORECASE, I

- 대소문자 구별 없이 매치를 수행할 때 사용하는 옵션

```
>>> p = re.compile('[a-z]+', re.I)
>>> p.match('python')
<re.Match object; span=(0, 6), match='python'>
>>> p.match('Python')
<re.Match object; span=(0, 6), match='Python'>
>>> p.match('PYTHON')
<re.Match object; span=(0, 6), match='PYTHON'>
```

- [a-z]+ 정규식은 소문자만을 의미하지만, re.I 옵션으로 대소문자 구별 없이 매치됨

08-2 정규 표현식 시작하기

컴파일 옵션

MULTILINE, M

- 메타 문자 ^(문자열의 처음), $(문자열의 마지막)와 연관된 옵션
- ^, $ 메타 문자를 각 줄마다 전체 적용시키고 싶을 때 사용하는 옵션

```
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))
```

실행 결과

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

08-2 정규 표현식 시작하기

▪ 컴파일 옵션

▪ VERBOSE, X

- 이해하기 어려운 정규식을 주석 또는 줄 단위로 구분할 때 사용하는 옵션

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

```
charref = 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)
```

- 문자열에 사용된 화이트스페이스는 컴파일할 때 제거됨
  (단, [ ] 안에 사용한 경우는 제외)

08-2 정규 표현식 시작하기

역슬래시 문제

- 정규 표현식을 파이썬에서 사용할 때 역슬래시(₩)를 여러 번 복잡하게 사용하는 경우 발생

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

- raw string 표현법
  - 정규식 문자열 앞에 r 문자를 삽입해 역슬래시 2개 대신 1개만 써도 2개를 쓴 것과 동일한 효과

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

감사합니다.