제7강 Regular Expressions

학습 목차

- 정규식이란?
- 기본 사용법
- 각종 매칭 문법
- 문자 클래스

Regular Expressions

- 텍스트의 패턴을 나타내는 규칙 정의
- 전화번호 010-333-4444

```
\d\d\d-\d\d\d\d-\d\d\d\d
\d{3}-\d{3}-\d{4}
```

기본 사용

```
>> import re
>> hello_re = re.compile(r'hello’)
>> hello_re.search('hello world')
<re.Match object; span=(0, 5), match='hello’>
>> print(hello_re.search('test'))
None
```

기본 사용

```
>>> import re
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('My number is 010-555-4242.')
>>> print('Phone number found: ' + mo.group())
Phone number found: 010-555-4242
```

그룹화 괄호 사용

```
>>> phoneNumRegex = re.compile(r'(₩d₩d₩d)-(₩d₩d₩d-₩d₩d₩d₩d)')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242’
>>> mo.groups()
('415', '555-4242')
>>> areaCode, mainNumber = mo.groups()
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

pipe | - 여러개 그룹을 동시에 매칭

```
>>> heroRegex = re.compile (r'Batman|Tina Fey')
>>> mo1 = heroRegex.search('Batman and Tina Fey.')
>>> mo1.group()
'Batman'
>>> mo2 = heroRegex.search('Tina Fey and Batman.')
>>> mo2.group()
'Tina Fey’
>>> batRegex = re.compile(r'Bat(man|mobile|copter|bat)')
>>> mo = batRegex.search('Batmobile lost a wheel')
>>> mo.group()
'Batmobile'
>>> mo.group(1)
'mobile'
```

? – 없거나 또는 하나 있는 요소에 대한 매칭

```
>>> batRegex = re.compile(r'Bat(wo)?man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'
>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman’

>>> phoneRegex = re.compile(r'(₩d₩d₩d-)?₩d₩d₩d-₩d₩d₩d₩d')
>>> mo1 = phoneRegex.search('My number is 415-555-4242')
>>> mo1.group()
'415-555-4242'
>>> mo2 = phoneRegex.search('My number is 555-4242')
>>> mo2.group()
'555-4242'
```

* - 없거나 또는 여러 번 반복되는 요소에 대한 매칭
+ - 한번 또는 여러 번 반복되는 요소에 대한 매칭

```
>>> batRegex = re.compile(r'Bat(wo)*man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'
>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
>>> mo3 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'

>>> batRegex = re.compile(r'Bat(wo)+man')
>>> mo1 = batRegex.search('The Adventures of Batwoman')
>>> mo1.group()
'Batwoman'
>>> mo2 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo2.group()
'Batwowowowoman'
```

{x,y} – 특정 횟수만큼 반복되는 요소에 대한 매칭

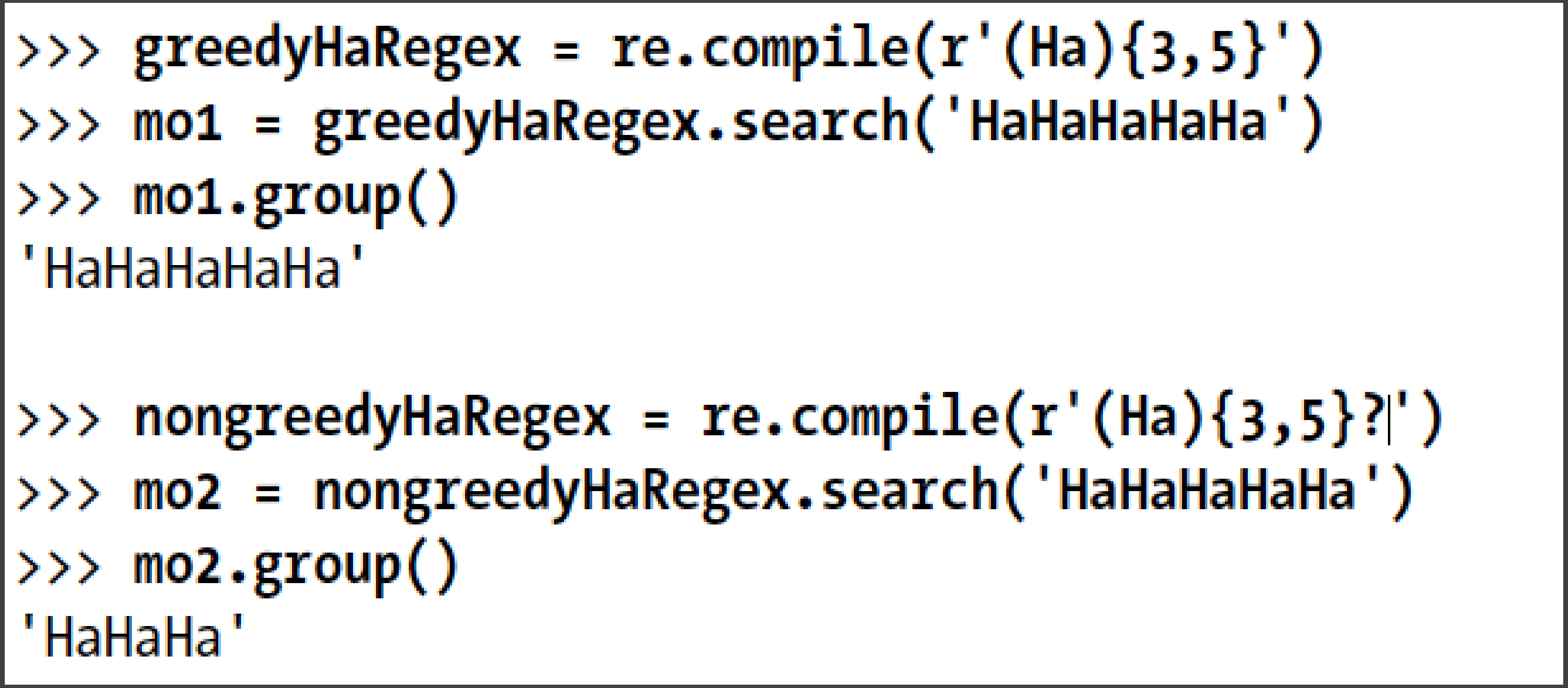
```
>>> haRegex = re.compile(r'(Ha){3}')
>>> mo1 = haRegex.search('HaHaHa')
>>> mo1.group()
'HaHaHa'
>>> mo2 = haRegex.search('Ha')
>>> mo2 == None
True
```

(Ha){3,5}

((Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha)(Ha))

Greedy vs Non-Greedy

- Greedy : 가장 긴 것에 매칭 (default)
- Non-Greedy : 먼저 발견된 것에 매칭

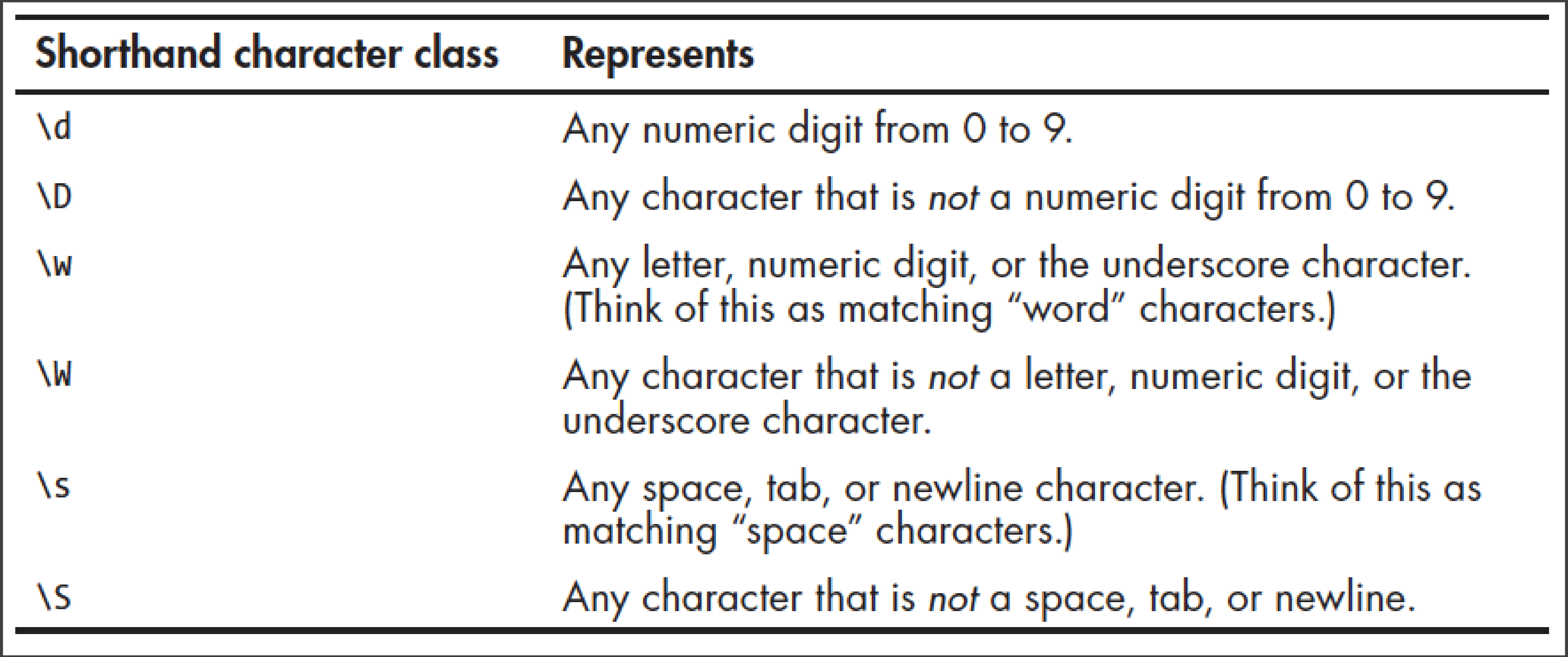


```
>>> greedyHaRegex = re.compile(r'(Ha){3,5}')
>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'

>>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?')
>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

findall() – 매칭되는 모든 문자열을 찾음.

```
>>> phoneNumRegex = re.compile(r'₩d₩d₩d-₩d₩d₩d-₩d₩d₩d₩d') # has no gro
ups
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
['415-555-9999', '212-555-0000’]

>>> phoneNumRegex = re.compile(r'(₩d₩d₩d)-(₩d₩d₩d)-(₩d₩d₩d₩d)') # has gro
ups
>>> phoneNumRegex.findall('Cell: 415-555-9999 Work: 212-555-0000')
[('415', '555', '1122'), ('212', '555', '0000')]
```

Character Class

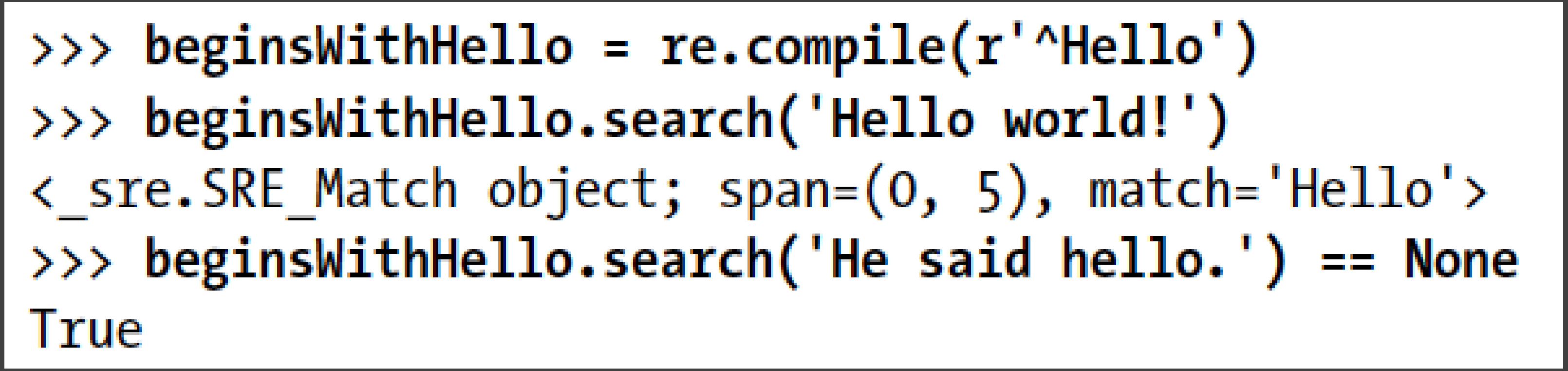


Shorthand character class	Represents
`\d`	Any numeric digit from 0 to 9.
`\D`	Any character that is *not* a numeric digit from 0 to 9.
`\w`	Any letter, numeric digit, or the underscore character. (Think of this as matching "word" characters.)
`\W`	Any character that is *not* a letter, numeric digit, or the underscore character.
`\s`	Any space, tab, or newline character. (Think of this as matching "space" characters.)
`\S`	Any character that is *not* a space, tab, or newline.

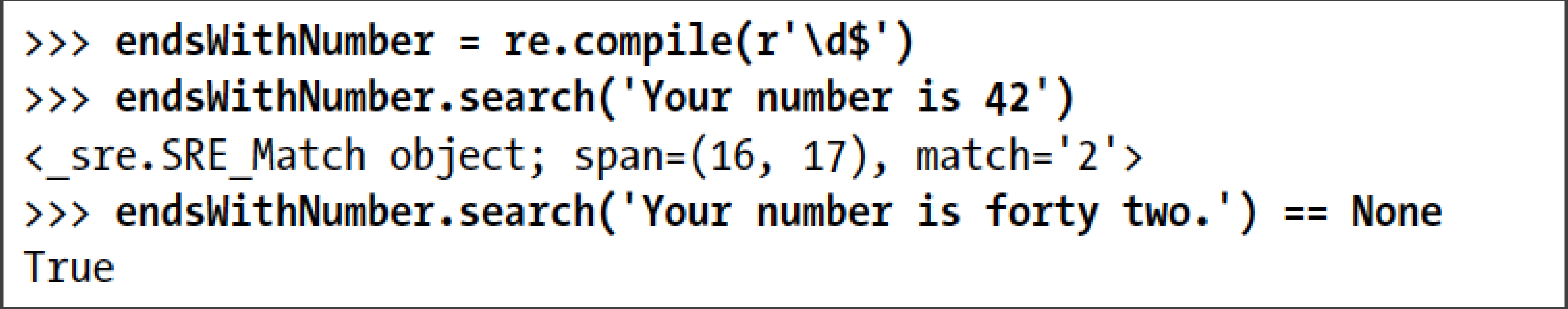
[] - 문자 클래스 사용자 지정

```
(a|b|c|d|e|f|g)
[abcdefg]
[a-g]
[aeiouAEIOU]
[a-zA-Z0-9]
[^aeiouAEIOU]
```

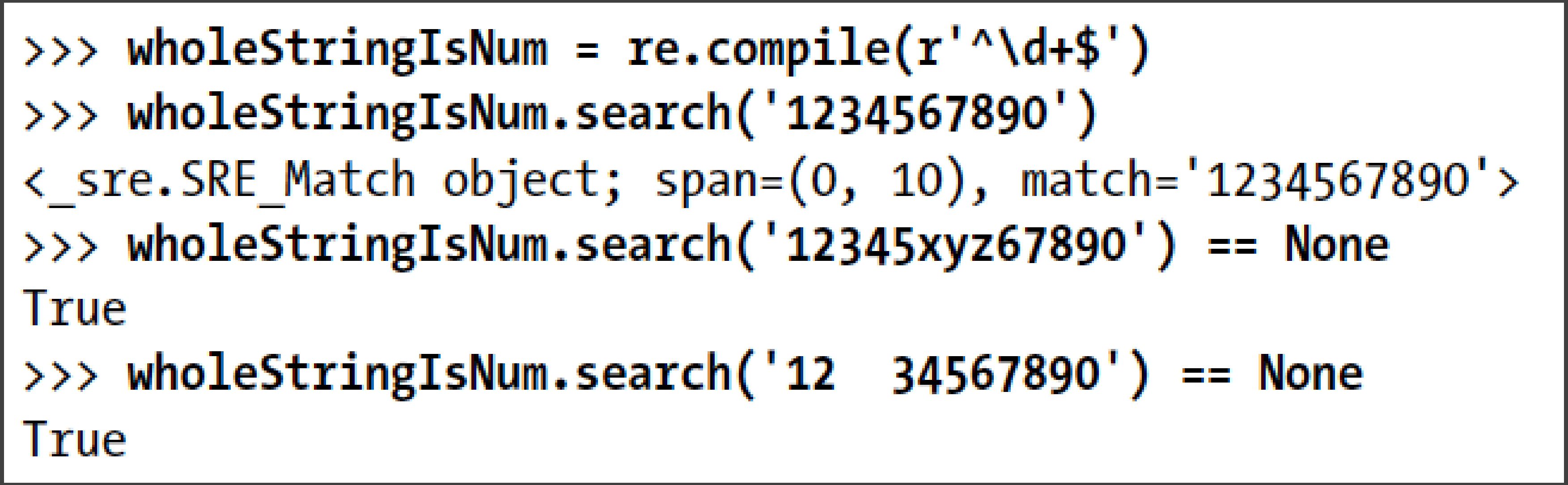
^ - 문자열의 시작을 매칭, $ - 문자열의 마지막을 매칭



```
>>> beginsWithHello = re.compile(r'^Hello')
>>> beginsWithHello.search('Hello world!')
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>> beginsWithHello.search('He said hello.') == None
True
```

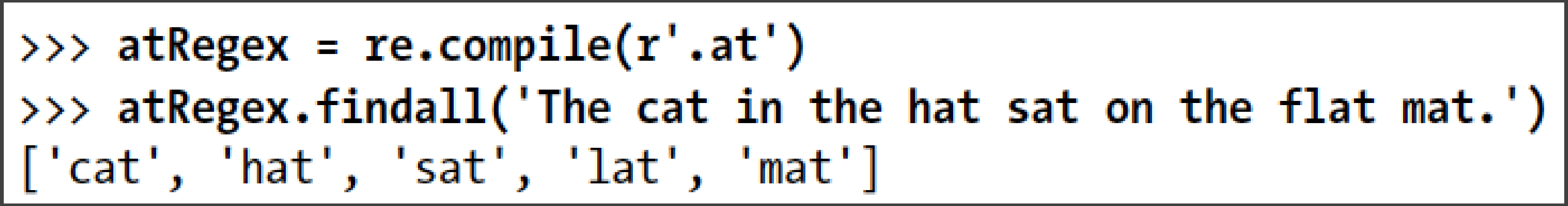


```
>>> endsWithNumber = re.compile(r'\d$')
>>> endsWithNumber.search('Your number is 42')
<_sre.SRE_Match object; span=(16, 17), match='2'>
>>> endsWithNumber.search('Your number is forty two.') == None
True
```



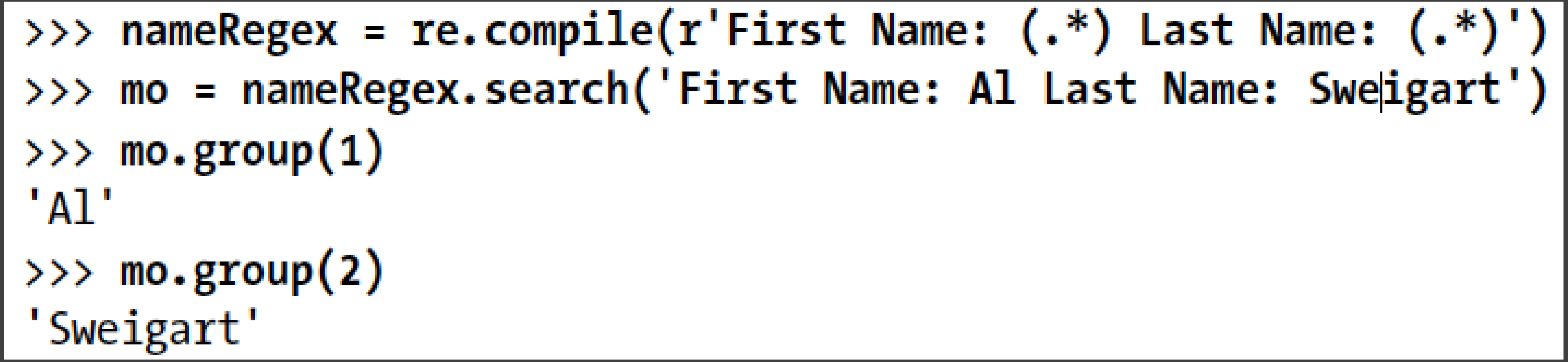
```
>>> wholeStringIsNum = re.compile(r'^\d+$')
>>> wholeStringIsNum.search('1234567890')
<_sre.SRE_Match object; span=(0, 10), match='1234567890'>
>>> wholeStringIsNum.search('12345xyz67890') == None
True
>>> wholeStringIsNum.search('12  34567890') == None
True
```

. - new line 을 제외한 모든 문자와 매칭



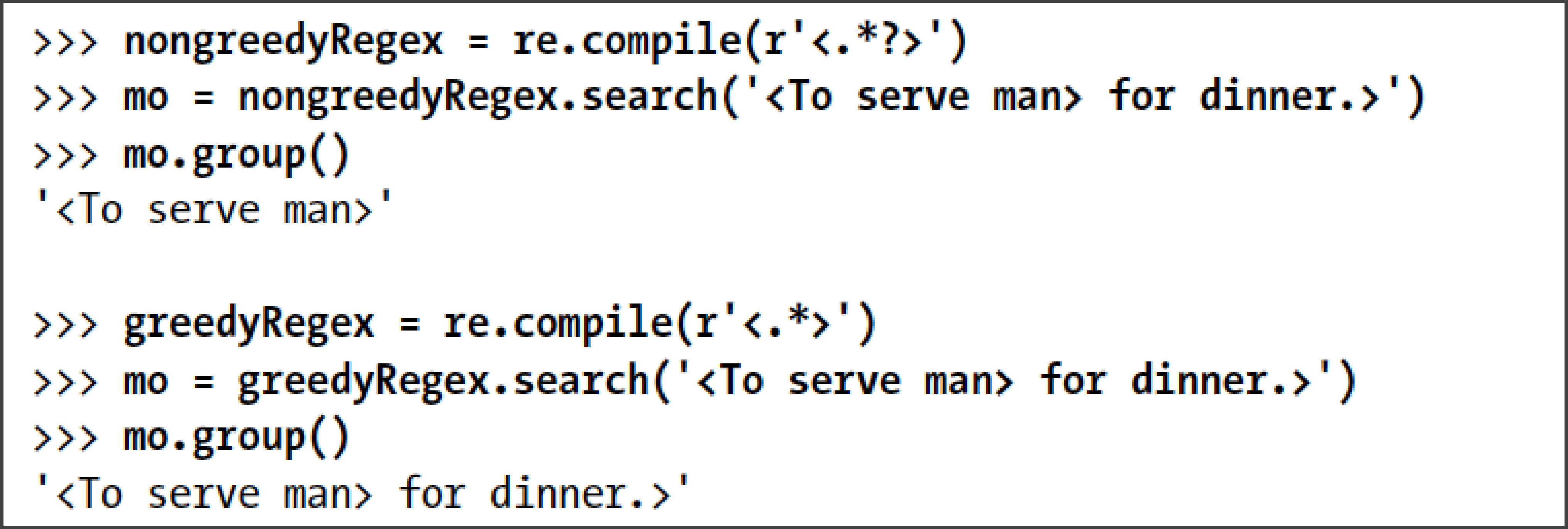
```
>>> atRegex = re.compile(r'.at')
>>> atRegex.findall('The cat in the hat sat on the flat mat.')
['cat', 'hat', 'sat', 'lat', 'mat']
```

.* - 모든 문자열에 매칭



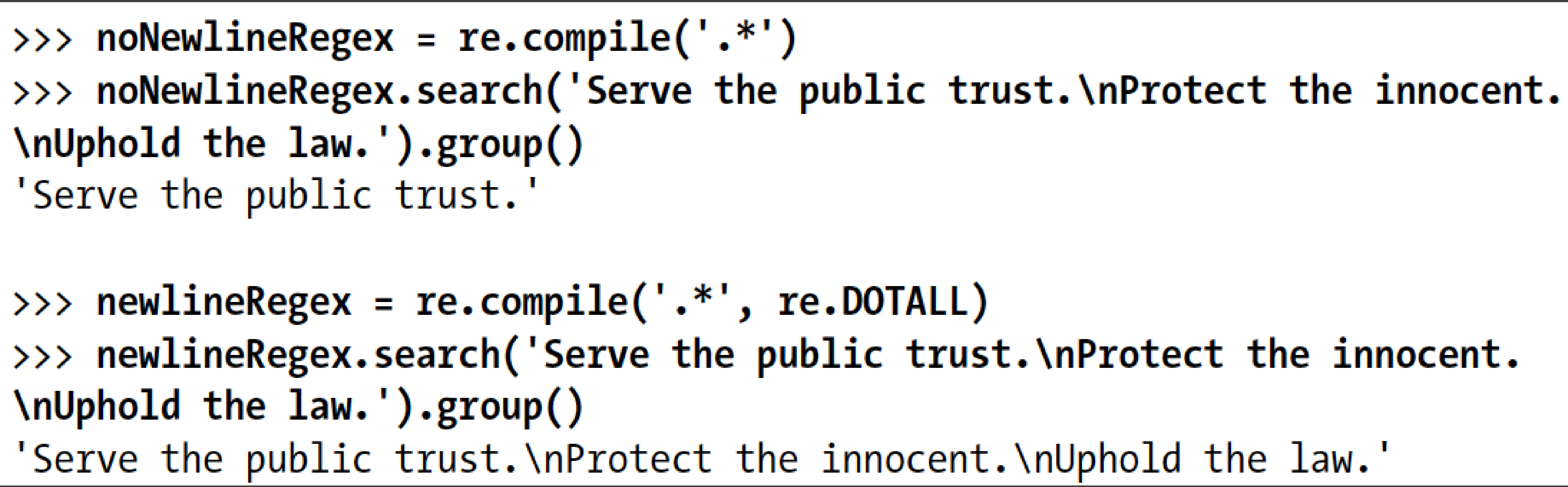
```
>>> nameRegex = re.compile(r'First Name: (.*) Last Name: (.*)')
>>> mo = nameRegex.search('First Name: Al Last Name: Sweigart')
>>> mo.group(1)
'Al'
>>> mo.group(2)
'Sweigart'
```

.* 에 대한 Greedy VS Non-Greedy



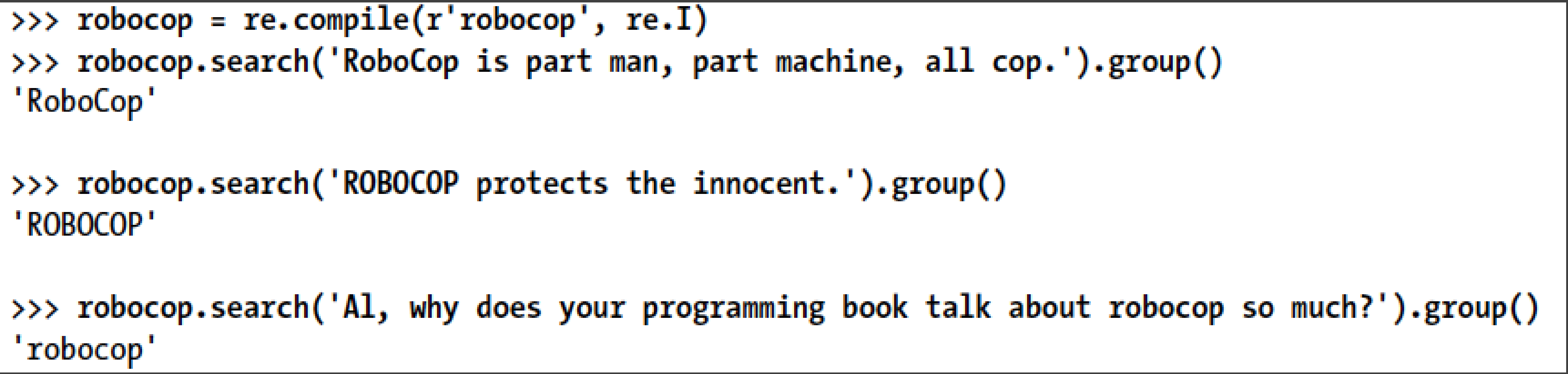
```
>>> nongreedyRegex = re.compile(r'<.*?>')
>>> mo = nongreedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man>'

>>> greedyRegex = re.compile(r'<.*>')
>>> mo = greedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man> for dinner.>'
```

new line /n 매칭 – DOTALL



```
>>> noNewlineRegex = re.compile('.*')
>>> noNewlineRegex.search('Serve the public trust.\nProtect the innocent.
\nUphold the law.').group()
'Serve the public trust.'

>>> newlineRegex = re.compile('.*', re.DOTALL)
>>> newlineRegex.search('Serve the public trust.\nProtect the innocent.
\nUphold the law.').group()
'Serve the public trust.\nProtect the innocent.\nUphold the law.'
```

대소문자 무시 - I



```
>>> robocop = re.compile(r'robocop', re.I)
>>> robocop.search('RoboCop is part man, part machine, all cop.').group()
'RoboCop'

>>> robocop.search('ROBOCOP protects the innocent.').group()
'ROBOCOP'

>>> robocop.search('Al, why does your programming book talk about robocop so much?').group()
'robocop'
```

sub() – 문자열 교체

```
>>> namesRegex = re.compile(r'Agent \w+')
>>> namesRegex.sub('CENSORED', 'Agent Alice gave the secret documents to Agent Bob.')
'CENSORED gave the secret documents to CENSORED.'
```

```
>>> agentNamesRegex = re.compile(r'Agent (\w)\w*')
>>> agentNamesRegex.sub(r'\1****', 'Agent Alice told Agent Carol that Agent
Eve knew Agent Bob was a double agent.')
A**** told C**** that E**** knew B**** was a double agent.'
```

복잡한 정규식의 쉬운 표시 방법 - VERBOSE

```
phoneRegex = re.compile(r'((\d{3}|\(\d{3}\))?(\s|-|\.)?\d{3}(\s|-|\.)\d{4}
(\s*(ext|x|ext.)\s*\d{2,5})?)')
```

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?            # area code
    (\s|-|\.)?                    # separator
    \d{3}                         # first 3 digits
    (\s|-|\.)                     # separator
    \d{4}                         # last 4 digits
    (\s*(ext|x|ext.)\s*\d{2,5})?  # extension
    )''', re.VERBOSE)
```

옵션의 동시 선택

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL | re.VERBOSE)
```

정리

?

*

+

{n,m}

^spam

spam$

.

₩d, ₩w, ₩s

[abc]

[^abc]