

정규식 (Regular Expression)

제11장



Python for Everybody
www.py4e.com



정규식

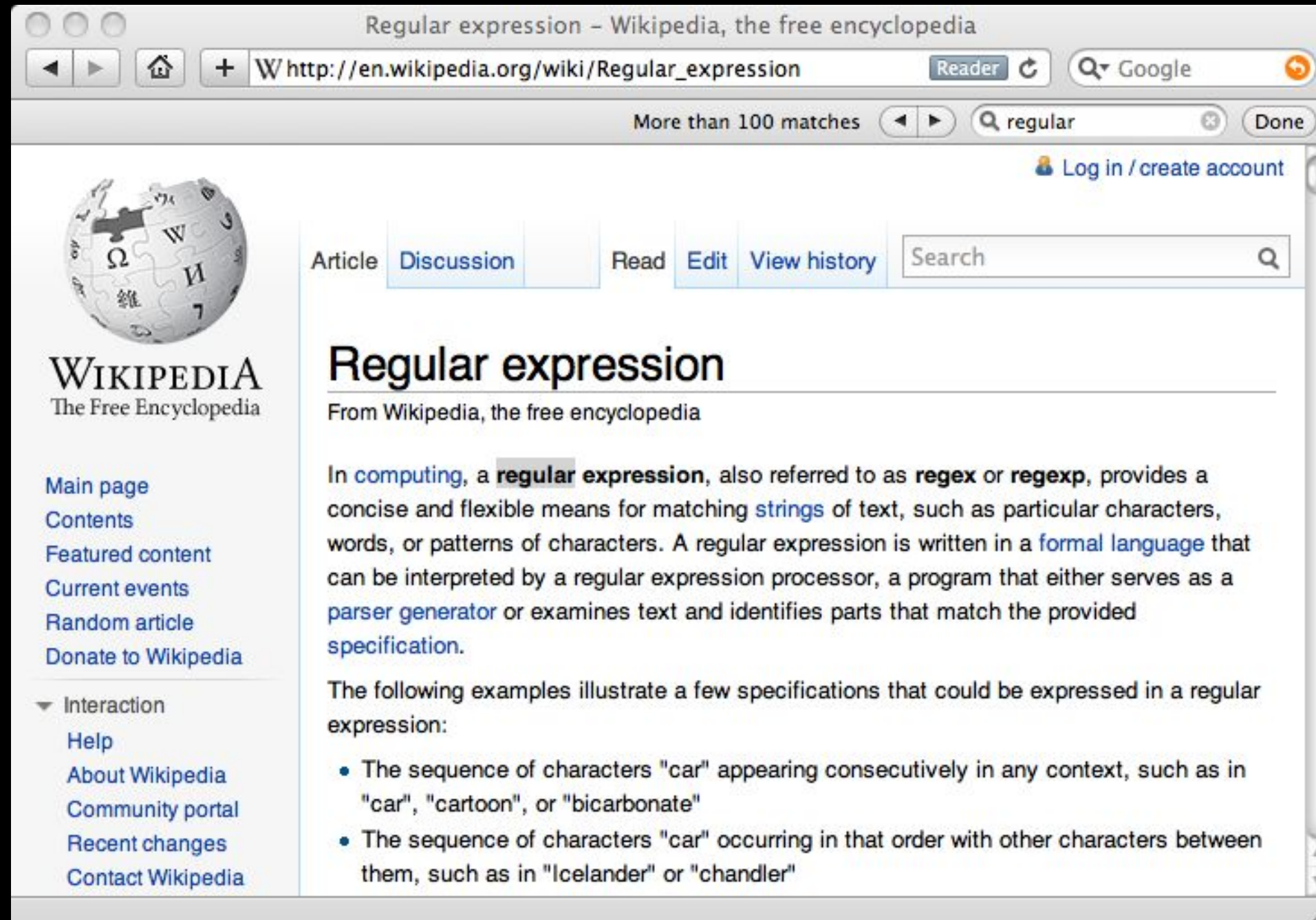
정규식은 텍스트에서 특정 글자나 단어, 패턴 등을 정확하고 유동적으로 표현하는 식입니다. 줄여서 `regex`나 `regexp`라고도 부르는데 정규식 처리기가 해석할 수 있도록 정해진 문법에 따라 사용하여야 합니다.

http://en.wikipedia.org/wiki/Regular_expression

정규식

문자열 비교나 처리를 하기 위한 아주 똑똑한
와일드 카드 표현식

http://en.wikipedia.org/wiki/Regular_expression



특히 검색을 효과적으로 할 수 있습니다

정규식의 이해

- 기호로 되어 있어 굉장히 효과적이지만 조금 어려움
- 한 번 배우면 활용할 곳이 많음
- 정규식은 그 자체로 하나의 언어입니다
- 특수(marker) 문자로 이루어진 언어로 문자만을 사용해서 프로그래밍을 하는 개념
- 축약된 '형식 언어'의 한 종류

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!

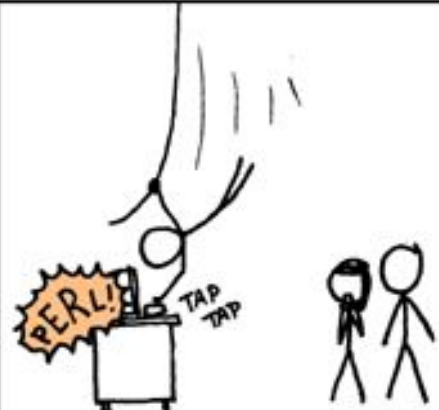


IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR EXPRESSIONS.



<http://xkcd.com/208/>

정규식 훑어보기

<code>^</code>	Matches the beginning of a line
<code>\$</code>	Matches the end of the line
<code>.</code>	Matches any character
<code>\s</code>	Matches whitespace
<code>\S</code>	Matches any non-whitespace character
<code>*</code>	Repeats a character zero or more times
<code>*?</code>	Repeats a character zero or more times (non-greedy)
<code>+</code>	Repeats a character one or more times
<code>+?</code>	Repeats a character one or more times (non-greedy)
<code>[aeiou]</code>	Matches a single character in the listed set
<code>[^XYZ]</code>	Matches a single character not in the listed set
<code>[a-z0-9]</code>	The set of characters can include a range
<code>(</code>	Indicates where string extraction is to start
<code>)</code>	Indicates where string extraction is to end

정규식 모듈

- 정규식을 사용 전 “`import re`” 명령어로 라이브러리를 불러오기
- `re.search()` 를 사용하면 `find()` 메소드를 쓰는 것처럼 정규식에 매칭되는 문자열을 찾을 수 있음
- `re.findall()` 을 사용하면 정규식에 맞는 문자열 추출 가능 (`find()` 와 slicing: `var[5:10]` 을 조합한 것과 유사)

re.search()를 find() 처럼 쓰기

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.find('From:') >= 0:
        print(line)
```

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line) :
        print(line)
```

re.search()를 find() 처럼 쓰기

```
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if line.startswith('From:') :
        print(line)
```

```
import re

hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('^From:', line) :
        print(line)
```

정해진 문자를 추가하여 문자열에
어떻게 매치되는지를 조절

특수 지정 문자

- 점(.) 문자는 어떤 문자가 와도 상관없다는 뜻
- asterisk(*) 문자는 몇 번 와도 상관없다는 뜻

X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain

X로 시작

몇 번이든
올 수 있음

X . * :

아무 문자

내 마음대로 매치 시키기

데이터가 얼마나 잘 정리되어 있는지와 어떻게 사용할지에 따라서 매치를 적절하게 사용

```
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-Plane is behind schedule: two weeks
X-: Very short
```

X로 시작

몇 번이든
올 수 있음

X.*:

아무 문자나

내 마음대로 매치 시키기

데이터가 얼마나 잘 정리되어 있는지와 어떻게 사용할지에 따라서 매치를 적절하게 사용

X-Sieve: CMU Sieve 2.3

X-DSPAM-Result: Innocent

X-: Very Short

X-Plane is behind schedule: two weeks

X-로 시작

한 번 이상 반복

X- \ S+ :

공백을 제외한 모든 것이
들어올 수 있음(\S)

매칭과 데이터 추출

- `re.search()` 는 해당 문자열이 대상 정규식을 만족시키는지를 True/False로 리턴
- 만약 매칭된 문자열을 추출하고 싶으면 `re.findall()` 을 사용

`[0-9]+`
↑
0부터 9까지

```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+', x)
>>> print(y)
['2', '19', '42']
```

매칭과 데이터 추출

`re.findall()` 을 사용하면 정규식에 매칭되는 부분 문자열을 모은 리스트를 리턴

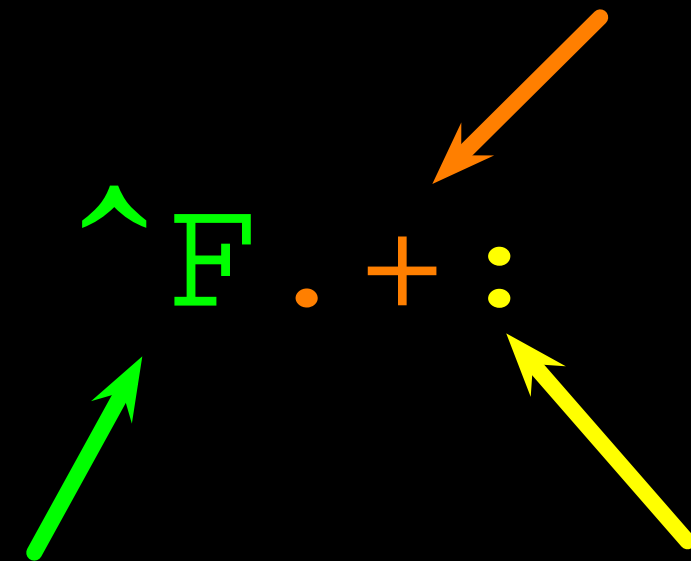
```
>>> import re
>>> x = 'My 2 favorite numbers are 19 and 42'
>>> y = re.findall('[0-9]+', x)
>>> print(y)
['2', '19', '42']
>>> y = re.findall('[AEIOU]+', x)
>>> print(y)
[]
```

주의: 정규식은 욕심쟁이!

repeat characters (* 과 +) 는 가장 길게 매칭되는 경우를 검색

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+:', x)
>>> print(y)
['From: Using the :']
```

하나 이상 반복



왜 'From:' 이 아닐까요?

시작 문자는 F

마지막 문자는 :

짧게도 정규식도 가능

를 붙여주면 욕심을 버릴 수 있음

?가 붙은 반복문은 가장 짧은 경우를 검색

```
>>> import re
>>> x = 'From: Using the : character'
>>> y = re.findall('^F.+?:', x)
>>> print(y)
['From: ']
```

하나 이상 반복 (가장 짧게)

\wedge F . + ? :

시작문자는 F

마지막 문자는 :

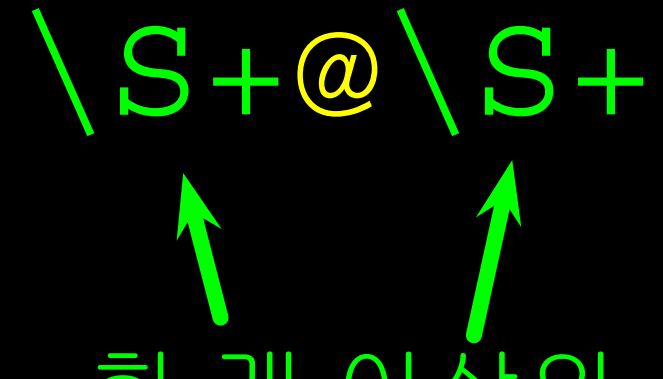
더 섬세한 문자열 추출

`re.findall()`을 사용할 때 괄호 안에 들어가는 정규식을 적절하게 제어하여 문자열을 효과적으로 추출할 수 있음

From `stephen.marquard@uct.ac.za` Sat Jan 5 09:14:16 2008

```
>>> y = re.findall('\S+@\S+', x)
>>> print(y)
['stephen.marquard@uct.ac.za']
```

`\S+@\S+`



한 개 이상의
빈 칸이 아닌
문자!

더 섬세한 문자열 추출

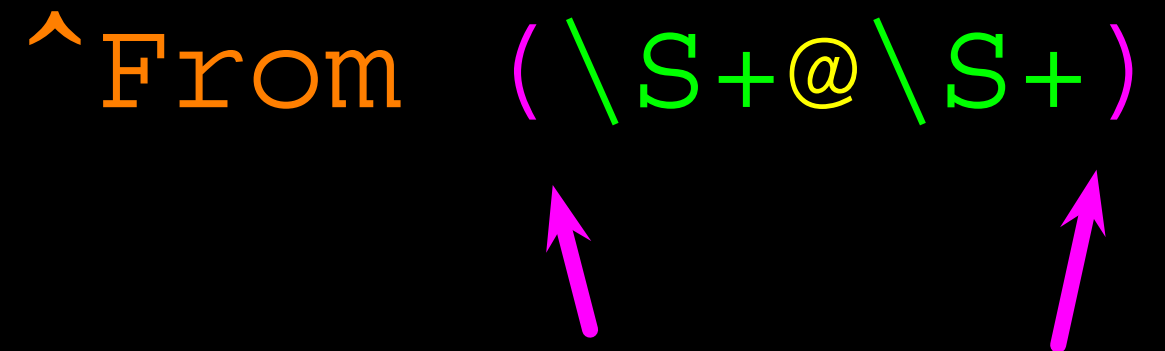
괄호는 매칭에 비포함

추출될 문자열의 시작 지점과 끝 지점을 지정하는 역할

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> y = re.findall('\S+@\S+', x)
>>> print(y)
['stephen.marquard@uct.ac.za']
>>> y = re.findall('^From (\S+@\S+)', x)
>>> print(y)
['stephen.marquard@uct.ac.za']
```

^From (\S+@\S+)

A diagram illustrating the regex pattern `^From (\S+@\S+)`. The text is color-coded: `^From` is orange, `(` is purple, `\S+@` is green, `\S+` is green, and `)` is purple. Two purple arrows point upwards to the opening and closing parentheses of the capturing group, highlighting that the parentheses themselves are not part of the extracted string.

문자열 처리 예시...

21 31
↓ ↓
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ', atpos)
>>> print(sppos)
31
>>> host = data[atpos+1 : sppos]
>>> print(host)
uct.ac.za
```

find와 slicing으로
hostname만 추출

중복 split 패턴

문자열을 split한 이후, 다시 일부분을 split 해야하는 경우가 있음

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
words = line.split()  
email = words[1]  
pieces = email.split('@')  
print(pieces[1])
```

```
stephen.marquard@uct.ac.za  
['stephen.marquard', 'uct.ac.za']  
'uct.ac.za'
```

정규식으로 쓰는 중복 split 패턴

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('@([^\s]*)', lin)
print(y)
```

```
['uct.ac.za']
```

'@([^\s]*)'

at(@) 표시가 나올 때까지 string을 탐색

정규식으로 쓴 중복 split 패턴

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('@([ ^ ]*)', lin)
print(y)
```

```
['uct.ac.za']
```

' @ ([^] *) '

Blank가 아닌 Character와 Match 몇 개든 제한 없음

정규식으로 쓴 중복 split 패턴

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('@([ ^ ]*)', lin)
print(y)
```

```
['uct.ac.za']
```

'@([^]*)'

blank가 아닌 문자들을 추출

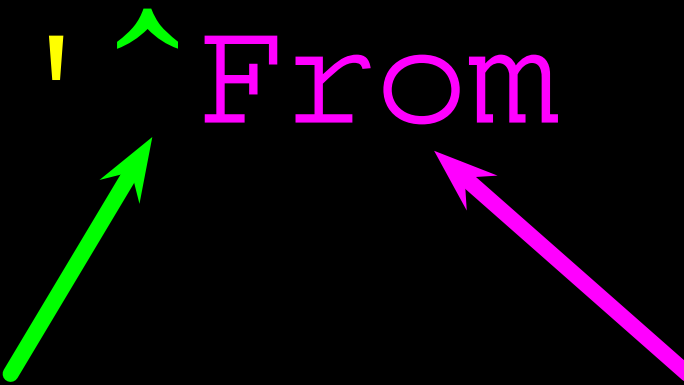
업그레이드된 정규식

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
import re  
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'  
y = re.findall('^From .*@([ ^]*)', lin)  
print(y)
```

```
['uct.ac.za']
```

'^From .*@([^]*)'

A diagram illustrating the components of the regular expression pattern. A green arrow points from the '^' character to the text '문자열의 시작 점을 찾는데,'. A purple arrow points from the word 'From' to the text '으로 시작하는 것을 검색'.

문자열의 시작 점을 찾는데, 'From' 으로 시작하는 것을 검색

업그레이드된 정규식

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('^From .*@([ ^]*)', lin)
print(y)
```

```
['uct.ac.za']
```

'^From . * @ ([^] *) '

다른 캐릭터는 모두 통과, @표시를 검색

업그레이드된 정규식

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('^From .*@([ ^]*)', lin)
print(y)
```

```
['uct.ac.za']
```

'^From .*@([^]*)'

추출 시작



업그레이드된 정규식

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('^From .*@([ ^]*)', lin)
print(y)
```

['uct.ac.za']

'^From .*@([^]*+)'

공백이 아닌 문자열

1개 이상이면 다 해당

업그레이드된 정규식

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

```
import re
lin = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
y = re.findall('^From .*@([ ^]*)', lin)
print(y)
```

```
['uct.ac.za']
```

```
'^From .*@([ ^]+)'
```

추출 종료



Spam 감지

```
import re
hand = open('mbox-short.txt')
numlist = list()
for line in hand:
    line = line.rstrip()
    stuff = re.findall('^X-DSPAM-Confidence: ([0-9.]+)', line)
    if len(stuff) != 1 : continue
    num = float(stuff[0])
    numlist.append(num)
print('Maximum:', max(numlist))
```

X-DSPAM-Confidence: 0.8475

python ds.py

Maximum: 0.9907

Escape Character

지정된 특수 문자를 그냥 문자 그대로 그대로
사용하고 싶을 경우 대부분 '\'를 붙이면 됨

```
>>> import re
>>> x = 'We just received $10.00 for cookies.'
>>> y = re.findall('\$[0-9.]+', x)
>>> print(y)
['$10.00']
```

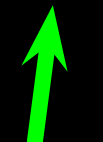
한 개 이상



\\$ [0-9.] +



\\$ = 그냥 \$



숫자 또는 점

요약

- 정규식은 조금 복잡하지만, 문자열을 처리하는데 매우 효과적
- 정규식은 특수문자로 의도를 함축적으로 표현



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

Contributor:

- Seung-June Lee (plusjune@gmail.com)
- Connect Foundation

Translator:

- Yang Incheol (inchyangv@gmail.com)
- Jeungmin Oh (tangza@gmail.com)