An illustration showing three stylized figures in white and blue robes reaching towards a bright blue sky. They are standing on a green hill with a white path. The background is a light blue sky with a large, faint circular shape.

4차산업 · 지능 정보화 시대의 필수역량 데이터 수집 · 처리 · 분석을 위한

산업융합 빅데이터 분석가 양성과정 (정규 표현식)

2020년 8월

우준식 (주) 시엠아이코리아 센터장

CISA / PMP / DAsP/ 정보시스템감리원

작업상담사 / 경영지도사

Hp) 010-5351-6791 / jswoo100@empas.co.kr

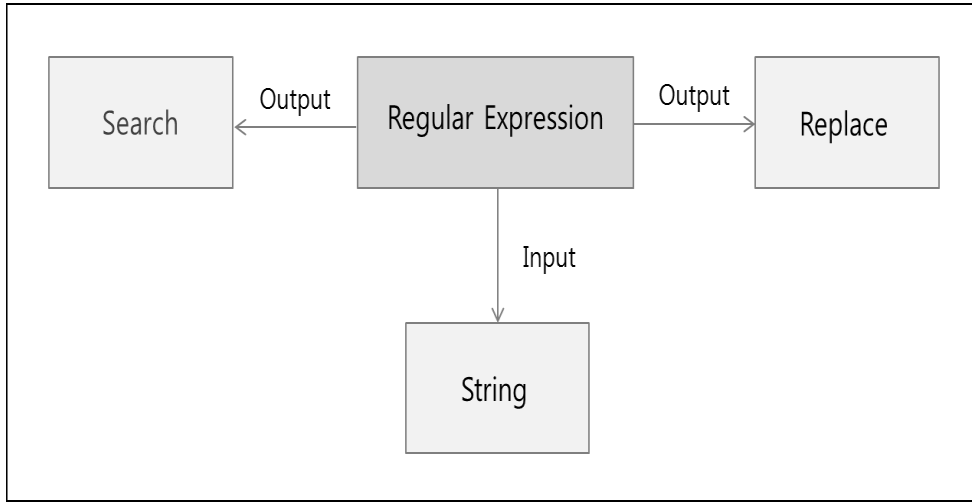


정규 표현식

(Regular Expression, regexp)

- ✓ <https://docs.python.org/3/library/re.html>
- ✓ <http://www.regexper.com/>
- ✓ https://www.w3schools.com/python/python_regex.asp
- ✓ <http://www.nextree.co.kr/p4327/>

◎. 정규표현식



Name
Braund, Mr. Owen Harris
Cumings, Mrs. John Bradley (Florence Briggs Th...
Heikkinen, Miss. Laina
Futrelle, Mrs. Jacques Heath (Lily May Peel)

' ([A-Za-z]+)W.'

- ✓ 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어
- ✓ 복잡한 문자열 패턴을 정의하는 문자 표현 공식
- ✓ 많은 텍스트 편집기와 프로그래밍 언어에서 문자열의 검색과 치환을 위해 지원
- ✓ 검색엔진, 워드 프로세서와 문서 편집기의 찾아 바꾸기 대화상자 등 문자처리 유틸리티에 사용

◎. 정규 표현식 표준 기호 (1/2)

메타 문자	설명	sample
.	•개행 문자 (Wn)을 제외한 모든 문자 1자	• "he..o" -> "hello hewxo heeo"
^	•문자열의 시작	• "^hello" -> "hello python!!!!"
\$	•문자열의 끝	• "world\$" -> "We are the world"
*	•해당 기호 바로 앞 문자가 0번 이상 반복	• "aix*" -> "ai, aix, aixx, bds, qqw"
+	•해당 기호 바로 앞 문자가 1번 이상 반복	• "aix+" -> "ai, aix, aixx, bds, qqw"
?	•해당 기호 바로 앞 문자가 0 또는 1번 반복	• "aix?" -> "ai, aix, aixx, abd, qqw"
[]	•문자의 집합, 기호 안의 여러 문자 중 일치할 경우의 한 문자	• "[c-e]" -> "abcdefghijk"

◎. 정규 표현식 표준 기호 (1/2)

메타 문자	설명	sample
{m,n}?	<ul style="list-style-type: none"> 바로 앞 문자의 반복 횟수 중 작은 경우부터 적용 	<ul style="list-style-type: none"> "a{2}" -> "aa" "a{2,}" -> "aa, aaa, aaaa, aaaaaaaaa" "a{2,4}" -> "aa, aaa, aaaa"
	<ul style="list-style-type: none"> 해당 기호의 앞 문자 또는 뒷 문자 or 연산에 해당 	<ul style="list-style-type: none"> "a b c" -> "a, d, e, f", "b,d,e", "c,e,f"

◎. 정규 표현식 표준 기호[2/2]

문자 규칙	설명	sample
W	<ul style="list-style-type: none"> 정규표현식에서 사용하는 문자를 그대로 표현 예) W+, W* 	
Wd	<ul style="list-style-type: none"> 모든 숫자 	<ul style="list-style-type: none"> [0-9]
WD	<ul style="list-style-type: none"> 숫자를 제외한 모든 문자 	<ul style="list-style-type: none"> [^0-9]
Ww	<ul style="list-style-type: none"> 영문 대소문자, 숫자, 밑줄문자 	<ul style="list-style-type: none"> [a-zA-Z0-9]
WW	<ul style="list-style-type: none"> 영문 대소문자, 숫자, 밑줄문자를 제외한 모든것 	<ul style="list-style-type: none"> [^a-zA-Z0-9]
Ws	<ul style="list-style-type: none"> 공백(스페이스), Wt, Wn, Wr, Wf, Wv 	<ul style="list-style-type: none"> [WtWnWrWfWv]
WS	<ul style="list-style-type: none"> 공백을 제외한 white space 문제 	<ul style="list-style-type: none"> [^ WtWnWrWfWv]

◎. 정규표현식 활용 [예]

zetawiki.com/wiki/정규표현식_예시

2.1. 전화번호 [편집]

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

예: 012-3456-7890, 123-4567-8901

`^d{2,3}\-d{3,4}\-d{4}$`

예: 02-1234-5678, 031-234-5678, 123-456-7890

`^01(?:0|1[6-9])\-(?:d{3}|d{4})\-d{4}$`

2.2. 이메일 주소 [편집]

`^[0-9a-zA-Z_\-]+\@[0-9a-zA-Z_\-]+\.$`

`^[0-9a-zA-Z_\-]+\@[0-9a-zA-Z_\-]+\(\.[0-9a-zA-Z_\-]+\)*$`

`^[0-9a-zA-Z_\-]+\@[0-9a-zA-Z_\-]+\(\.[0-9a-zA-Z_\-]+\){1,2}$`

`^[0-9a-zA-Z]([0-9a-zA-Z_\-]*[0-9a-zA-Z_\-_+])*(\.[0-9a-zA-Z]([0-9a-zA-Z_\-]*[0-9a-zA-Z_\-_+])*)+[a-zA-Z]{2,9}$`

2.3. IP 주소 [편집]

`^d{1,3}\.d{1,3}\.d{1,3}\.d{1,3}$`

0.0.0.0 ~ 999.999.999.999^[1]

2.4. 맥 주소 [편집]

`^([0-9a-fA-F]{2}:){5}[0-9a-fA-F]{2}$`

2.5. 주민등록번호 [편집]

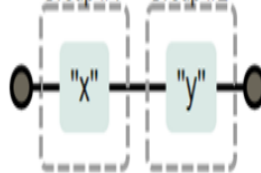
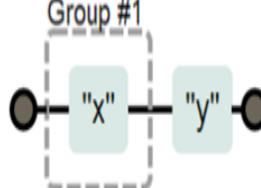
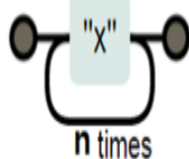
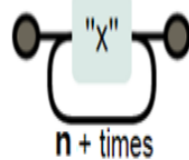
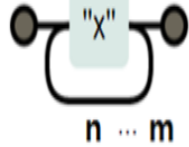
`^\d{6}\-[1-4]\d{6}$`

2.6. 태그 [편집]

https://zetawiki.com/wiki/정규표현식_예시

◎. 정규표현식 표현방법(1/3)

정규표현식	표현	설명
$\wedge x$		문자열이 x로 시작합니다.
$x\$$		문자열이 x로 끝납니다.
$.x$		임의의 한 문자를 표현합니다. (x가 마지막으로 끝납니다.)
x^+		x가 1번이상 반복합니다.
$x^?$		x가 존재하거나 존재하지 않습니다.
x^*		x가 0번이상 반복합니다.
$x y$		x 또는 y를 찾습니다. (or연산자를 의미합니다.)
(x)		()안의 내용을 캡처하며, 그룹화 합니다.

$(x)(y)$		그룹화 할 때, 자동으로 앞에서부터 1번부터 그룹 번호를 부여해서 캡처합니다. 결과값에 그룹화한 Data가 배열 형식으로 그룹번호 순서대로 들어갑니다.
$(x)(?:y)$		캡처하지 않는 그룹을 생성할 경우 ?:를 사용합니다. 결과값 배열에 캡처하지 않는 그룹은 들어가지 않습니다.
$x\{n\}$		x를 n번 반복한 문자를 찾습니다.
$x\{n,\}$		x를 n번이상 반복한 문자를 찾습니다.
$x\{n,m\}$		x를 n번이상 m번이하 반복한 문자를 찾습니다.

◎. 정규표현식 표현방법(2/3)

정규표현식	표현	설명
[xy]	One of: 	xy중 하나를 찾습니다.
[^xy]	None of: 	xy를 제외하고 문자 하나를 찾습니다. (문자 클래스 내의 ^는 not을 의미합니다.)
[x-z]	One of: 	x~z 사이의 문자중 하나를 찾습니다.
\w^		^(특수문자)를 식에 문자 자체로 포함합니다. (escape)
\wb		문자와 공백사이의 문자를 찾습니다.
\WB		문자와 공백사이가 아닌 값을 찾습니다.

\d		숫자를 찾습니다.
\D		숫자가 아닌 값을 찾습니다.
\s		공백문자를 찾습니다.
\S		공백이 아닌 문자를 찾습니다.
\t		Tab 문자를 찾습니다.
\v		Vertical Tab 문자를 찾습니다.
\w		알파벳 + 숫자 + _ 를 찾습니다.
\W		알파벳 + 숫자 + _ 을 제외한 모든 문자를 찾습니다.

◎. 정규표현식 표현방법(5/3)

정규표현식	표현	설명
[alnum:]		알파벳과 숫자를 찾습니다.
[alpha:]		알파벳을 찾습니다.
[digit:]		0~9사이를 찾습니다.
[lower:]		알파벳 소문자를 찾습니다.
[upper:]		알파벳 대문자를 찾습니다.
[blank:]		탭과 공백문자를 찾습니다.

REGEXPER
You thought you only had two problems...

- Changelog
- Documentation
- Source on GitLab

`\d{6} \- [1-4]\d{6}`

Display

[Download SVG](#) // [Download PNG](#) // [Permalink](#)

The diagram illustrates the Non-deterministic Finite Automaton (NFA) for the regular expression `\d{6} \- [1-4]\d{6}`. It consists of three main components connected in sequence: 1. A green box labeled 'digit' with a self-loop arrow and the text '5 times' below it. 2. A light blue box containing the hyphen character '-'. 3. Another green box labeled 'digit' with a self-loop arrow and the text '5 times' below it. Above the hyphen box is a grey box labeled 'One of:' which contains two sub-boxes, one with '1' and one with '4', indicating a choice between these two characters.

`\d{6} \- [1-4]\d{6}`

Python »

English

3.8.5

Documentation » The Python Standard Library » Text Processing Services »

Table of Contents

re — Regular expression operations

Regular Expression Syntax

Module Contents

Regular Expression Objects

Match Objects

Regular Expression Examples

Checking for a Pair

Simulating scanf()

search() vs. match()

Making a Phonebook

Text Munging

Finding all Adverbs

Finding all Adverbs and their Positions

Raw String Notation

Writing a Tokenizer

Previous topic

string — Common string operations

Next topic

difflib — Helpers for computing deltas

This Page

Report a Bug

Show Source

re — Regular expression operations

Source code: [Lib/re.py](#)

This module provides regular expression matching operations similar to those found in Perl.

Both patterns and strings to be searched can be Unicode strings (`str`) as well as 8-bit strings (`bytes`). However, Unicode strings and 8-bit strings cannot be mixed: that is, you cannot match a Unicode string with a byte pattern or vice-versa; similarly, when asking for a substitution, the replacement string must be of the same type as both the pattern and the search string.

Regular expressions use the backslash character (`'\'`) to indicate special forms or to allow special characters to be used without invoking their special meaning. This collides with Python's usage of the same character for the same purpose in string literals; for example, to match a literal backslash, one might have to write `'\\'` as the pattern string, because the regular expression must be `\\`, and each backslash must be expressed as `\\` inside a regular Python string literal. Also, please note that any invalid escape sequences in Python's usage of the backslash in string literals now generate a `DeprecationWarning` and in the future this will become a `SyntaxError`. This behaviour will happen even if it is a valid escape sequence for a regular expression.

The solution is to use Python's raw string notation for regular expression patterns; backslashes are not handled in any special way in a string literal prefixed with `'r'`. So `r'\n'` is a two-character string containing `'\'` and `'n'`, while `"\n"` is a one-character string containing a newline. Usually patterns will be expressed in Python code using this raw string notation.

It is important to note that most regular expression operations are available as module-level functions and methods on [compiled regular expressions](#). The functions are shortcuts that don't require you to compile a regex object first, but miss some fine-tuning parameters.

See also:

The third-party [regex](#) module, which has an API compatible with the standard library `re` module, but offers additional functionality and a more thorough Unicode support.

Regular Expression Syntax

A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

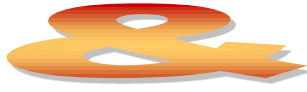
Regular expressions can be concatenated to form new regular expressions; if *A* and *B* are both regular expressions, then *AB* is also a regular expression. In general, if a string *s* matches *A* and another string *t*

◎. re 모듈 함수

모듈	설명
re.compile()	<ul style="list-style-type: none"> • 정규표현식을 컴파일 하는 함수, 파이썬에 전해주는 역할 • 찾고자 하는 패턴이 빈번한 경우에 미리 컴파일해 놓고 사용(속도와 편의성)
re.search()	<ul style="list-style-type: none"> • 문자열 전체에 대해서 정규표현식과 매치되는지 검색
re.match()	<ul style="list-style-type: none"> • 문자열의 처음이 정규표현식과 매치되는지 검색
re.split()	<ul style="list-style-type: none"> • 정규 표현식을 기준으로 문자열을 분리하여 리스트로 리턴
re.findall()	<ul style="list-style-type: none"> • 문자열에서 정규 표현식과 매치되는 모든 경우의 문자열을 찾아서 리스트로 리턴 • 매치되는 문자열이 없다면 빈 리스트가 리턴
re.finditer()	<ul style="list-style-type: none"> • 문자열에서 정규 표현식과 매치되는 모든 경우의 문자열에 대한 이터레이터 객체를 리턴
re.sub()	<ul style="list-style-type: none"> • 문자열에서 정규 표현식과 일치하는 부분에 대해서 다른 문자열로 대체

◎. 꿈은 이루어 진다.





우준식 [주]시엠아이코리아 아카데미 센터장

직업상담사 / 경영지도사(인사관리) / 인적자원개발사

PMP / CISA / 재난관리사 / 정보시스템감리원

“IT.Career · HRD 전문가”

연락처 : [hp] 010-5351-6791, jswoo100@empas.com

블로그 : <http://blog.naver.com/jswoo100>

SNS : [twitter ID]->@goodjob21

[facebook]->jswoo100@empas.com