정규표현식 & 문자열검색

축약표현	[ ]문자클래스	뜻	사용처
\d	[0-9]	숫자를 찾음	숫자
\D	[^0-9]	숫자가 아닌 것을 찾음	텍스트+특수문자+공백문자
\w	[a-zA-Z0-9_]	world 문자+숫자(alphanumeric)를 찾음	스페이스, 탭, 개행(new line)
\W	[^a-zA-Z0-9_]	문자+숫자(alphanumeric)가 아닌 것을 찾음	텍스트+특수문자+숫자
\s	[\t\n\r\f\v]	white space 공백문자인 것을 찾음	텍스트+숫자 = [:blank:]
\S	[^ \t\n\r\f\v]	white space 공백문자가 아닌 것을 찾음	특수문자+공백

\t	탭 Tab	[]	문자 클래스
\n	줄바꿈 (LF)	()	그룹핑, 추출할 패턴 지정
\r	캐리지리턴 (CR)	₩	메타문자를 일반문자로 인식하게함
\f	폼피드	^ $	문자열의 시작과 끝을 의미
\v	수직 탭 Vertical Tab	|	or 조건식을 의미, 다자택일

[xy]	x,y중 하나를 찾습니다.	[:alnum:]	알파벳과 숫자를 찾습니다
[^xy]	x,y를 제외하고 문자 하나를 찾습니다.	[:alpha:]	알파벳을 찾습니다
[x-z]	x~z사이의 문자 중 하나를 찾습니다.	[:digit:]	0~9사이를 찾습니다
\(\)	(소괄호 열기) 소괄호 닫기	[:lower:]	알파벳 소문자를 찾습니다
\\	\ 역슬래쉬	[:upper:]	알파벳 대문자를 찾습니다

.	\n(줄바꿈 문자)를 제외한 모든 문자와 일치 []안에서는 점이 작동하지 않음	a.b	"a+모든문자+b"
		a[.]b	"a+.+b"
? {0,1}	?앞의 표현식이 있어도 되고 없어도 된다	ca?t	ct / cat / ~~caaat~~/
* {0,}	*바로 앞의 문자 0부터 무한대로 반복	ca*t	cat / caaaat / caaaaaaat
+ {1,}	+바로 앞의 문자 1부터 무한대로 반복	ca+t	~~cat~~/ caaaat / caaaaaaat
{ } 메타문자를 사용하면 반복횟수를 고정할 수 있다.		ca{4}t	~~cat~~/ caaaat / ~~caaaaaaat~~
{n,m}	n(0)번부터 m(무한)까지 횟수의 반복 지정	ca{3,}t	반복횟수 3이상 (생략 = 0)
r'정규표현식'의미 : Raw String(순수문자)임을 알려줌		ca{,3}t	반복횟수 3이하
		(c,a){2,3}t	~~cat~~ / cacat / cacacat / ~~....~~

import re	re.함수(pattern, string, flags=0)	re.compile	
re.match()	re.match('a1','list')	문자열 도입에서 패턴a1찾기 (or none)	
re.search()	re.serch('a1','list')	문자열 전체에서 패턴a1찾기 (or none)	
re.findall()	re.findall('a','list')	매치된 모든 문자열을 리스트로 반환	
re.finditer()	re.finditer('a','list')	매치되는 모든 문자열을 반복 가능한 객체로 반환	
re.fullmatch()	re.fullmatch('a','list')	패턴과 문자열이 완벽하게 일치하는지 검사 (or none)	
.split()	dfp['A'].str.split(' - ', 1).str[1]	패턴으로 나누기	
.replace()	.str.replace(' ','')	''을 ''로 바꾸기	
.count()		패턴에 들어가는 문자의 개수를 반환	
.join()	','.join('abcd') => 'a,b,c,d'		

리스트가 순서대로 변수가 되어 수행할 문장으로 들어간다.

for 변수 **in** 리스트[] 또는 튜브() 또는 문자열 :
    수행할 문장1

flags= 패턴변경자		
	g	global 문자 내의 모든 패턴 찾음
	i	Ignore Case 대소문자 구별않음
	s	.에 (\n)도 포함. 줄바꿈도 문자취급
	m	Multi Line 행이 바뀌어도 찾음
	x	공백문자 무시

(공통) 시퀀스연산

연산	결과
`x in s`	s 의 항목 중 하나가 x 와 같으면 True, 그렇지 않으면 False
`x not in s`	s 의 항목 중 하나가 x 와 같으면 False, 그렇지 않으면 True
`s + t`	s 와 t 의 이어 붙이기
`s * n`	s 를 그 자신에 n 번 더하는 것과 같습니다
`n * s`	
`s[i]`	s 의 i 번째 항목, 0에서 시작합니다
`s[i:j]`	s 의 i 에서 j 까지의 슬라이스
`s[i:j:k]`	s 의 i 에서 j 까지 스텝 k 의 슬라이스
`len(s)`	s 의 길이
`min(s)`	s 의 가장 작은 항목
`max(s)`	s 의 가장 큰 항목
`s.index(x[, i[, j]])`	(인덱스 i 또는 그 이후에, 인덱스 j 전에 등장하는) s 의 첫 번째 x 의 인덱스
`s.count(x)`	s 등장하는 x 의 총수

(가변) 시퀀스연산

연산	결과
`s[i] = x`	s 의 항목 i 를 x 로 대체합니다
`s[i:j] = t`	i 에서 j 까지의 s 슬라이스가 이터러블 t 의 내용으로 대체됩니다
`del s[i:j]`	s[i:j] = [] 와 같습니다
`s[i:j:k] = t`	s[i:j:k] 의 항목들이 t 의 항목들로 대체됩니다 (t 는 교체할 슬라이스와 길이가 같아야 합니다)
`del s[i:j:k]`	리스트에서 s[i:j:k] 의 항목들을 제거합니다
`s.append(x)`	시퀀스의 끝에 x 를 추가합니다 (s[len(s):len(s)] = [x] 와 같습니다)
`s.clear()`	s 에서 모든 항목을 제거 (del s[:] 와 같습니다) (i의 기본값은 -1, 마지막항목제거되면서 반환)
`s.copy()`	s 의 얕은 복사본을 만듭니다 (s[:] 와 같습니다)
`s.extend(t) 또는 s += t`	t 의 내용으로 s 를 확장합니다 (대부분 s[len(s):len(s)] = t 와 같습니다)
`s *= n`	내용이 n 번 반복되도록 s 를 갱신합니다
`s.insert(i, x)`	x 를 s 의 i 로 주어진 인덱스에 삽입합니다 (s[i:i] = [x] 와 같습니다)
`s.pop() or s.pop(i)`	i 에 있는 항목을 꺼냄과 동시에 s 에서 제거합니다
`s.remove(x)`	s[i] 가 x 와 같은 첫 번째 항목을 s 에서 제거합니다
`s.reverse()`	제자리에서 s 의 항목들의 순서를 뒤집습니다

printf 스타일 문자열 포매팅

(1) 변환 플래그 문자

`'#'`	값 변환에 “대체 형식” (아래에 정의되어있습니다) 을 사용합니다.	`'-'`	변환된 값은 왼쪽으로 정렬됩니다 (둘 다 주어지면 '0' 변환보다 우선 합니다).
`'0'`	변환은 숫자 값의 경우 0으로 채웁니다.	`' '`	(스페이스) 부호 있는 변환 때문에 만들어진 양수 앞에 빈 칸을 남겨둡니다 (음수면 빈 문자열입니다).
		`'+'`	부호 문자 ('+' or '-') 가 변환 앞에 놓입니다 (' ' 플래그에 우선합니다).

(2) 변환 유형

`'d'`	부호 있는 정수 십진 표기.	`‘f’`	부동 소수점 십진수 형식.
`'i'`	부호 있는 정수 십진 표기.	`‘F’`	부동 소수점 십진수 형식.
`'o'`	부호 있는 8진수 값.	`‘g’`	부동 소수점 형식. 지수가 -4보다 작거나 정밀도 보다 작지 않으면 소문자 지수형식을 사용하고, 그렇지 않으면 십진수 형식을 사용합니다.
`‘u’`	쓸데없는 유형 – 'd' 와 같습니다.	`‘G’`	부동 소수점 형식. 지수가 -4보다 작거나 정밀도 보다 작지 않으면 대문자 지수형식을 사용하고, 그렇지 않으면 십진수 형식을 사용합니다.
`'x'`	부호 있는 16진수 (소문자).	`‘c’`	단일 바이트 (정수 또는 길이 1인 바이너리 시퀀스를 허용합니다).
`'X'`	부호 있는 16진수 (대문자).	`'b'`	바이너리 시퀀스 (버퍼 프로토콜 을 따르거나 __bytes__() 가 있는 모든 객체).
`'e'`	부동 소수점 지수 형식 (소문자).	`'s'`	's' 는 'b' 의 별칭이고 파이썬 2/3에서만 사용되어야 합니다.
`'E'`	부동 소수점 지수 형식 (대문자).	`'a'`	Bytes (converts any Python object using repr(obj).encode('ascii', 'backslashreplace')).
`‘f’`	부동 소수점 십진수 형식.	`'r'`	'r' 는 'a' 의 별칭이고 파이썬 2/3에서만 사용되어야 합니다.
`‘F’`	부동 소수점 십진수 형식.	`'%'`	인자는 변환되지 않고, 결과에 '%' 문자가 표시됩니다.

데이터타입		
자료형	**설명**	
1 숫자형	**int정수, float실수, complex**	**Number**
ㄱ 문자열 자료형	'Hello', "123" **string(object)**	**Text**
[ ] 리스트 자료형	[ ], [1, 3, 4, 7, 8] 수정가능	**Sequence**
( ) 튜플 자료형	( ), (1, 2, 3) 읽기전용수정불가	**Binary**
{ } 딕셔너리 자료형	dict({'name' : 'MH', 'birth': '1130'})	**Mapping**
set() 집합자료형	set([1,2,3]) set("hello")	**Set**
Bloolean	True False	**Bloolean**

비교연산자, 논리연산자, 비트연산자, 산술연산자

비교연산자 (Comparison Operators)	
연산	**결과**
<	엄격히 작다
<=	작거나 같다
>	엄격히 크다
>=	크거나 같다
==	같다
!=	같지 않다
is	객체 아이덴티티
is not	부정된 객체 아이덴티티

논리연산자 (Logical Operators)	
연산	**결과**
x or y	x가 거짓이면 y, 그렇지 않으면 x
x and y	x*가 거짓이면 *x, 그렇지않으면 y
not x	x가 거짓이면 True, 아니면 False

정수형에 대한 비트연산자 (Bitwise Operators)	
x | y	x와 y의 비트별 or
x ^ y	x와 y의 비트별 배타적 or (제외or)
x & y	x와 y의 비트별 and
x << n	x를 n비트만큼 왼쪽으로 시프트
~x	x의 비트 반전

산술연산자 (Arithmetic Operators)			
연산	**결과**	**연산**	**결과**
x + y	x와 y의 합	**int(x)**	정수로 변환된 x
x – y	x와 y의 차	**float(x)**	실수로 변환된 x
x * y	x와 y의 곱	**complex(re, im)**	실수부 re와 허수부 im으로 구성된 복소수. (기본값 0)
x / y	x와 y의 몫	**c.conjugate( )**	복소수 c의 켤레
x // y	x와 y의 정수로 내림한 몫	**divmod(x, y)**	x와 y의 정수로 내림한 몫
x % y	x / y의 의 나머지	**pow(x, y)**	x의 y거듭제곱
- x	음의 x	**x ** y**	x의 y거듭제곱
+ x	x 그대로	**math.floor(x)**	가장 큰 integral이 x
abs(x)	x 의 절댓값 또는 크기	**math.ceil(x)**	가장 작은 integral이 x