정규 표현식(Regular Expression)

정규식(Regular Expression)은 문자열에서 특정 내용을 찾거나 대체 또는 발췌하는데 사용된다.

대표적으로 입력칸에 전화번호나 이메일을 입력하라고 했을때 옳지 않은 값을 입력하면 정규표현식에 의해 필터링되어 걸러져 경고창을 띄우는 화면을 본적이 있을 것이다.

이처럼 반복문과 조건문을 사용해야 할것같은 복잡한 코드도 정규표현식을 이용하면 매우 간단하게 표현할 수 있으며 주로 다음과 같은 상황에서 굉장히 유용하게 사용된다.

- 각각 다른 포맷으로 저장된 엄청나게 많은 전화번호 데이터를 추출해야 할 때
- 사용자가 입력한 이메일, 휴대폰 번호, IP 주소 등이 올바른지 검증하고 싶을 때
- 코드에서 특정 변수의 이름을 치환하고 싶지만, 해당 변수의 이름을 포함하고 있는 함수는 제외하고 싶을 때
- 특정 조건과 위치에 따라서 문자열에 포함된 공백이나 특수문자를 제거하고 싶을 때

JAVASCRIPT

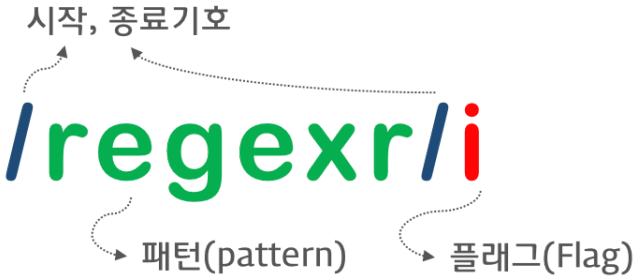
```
// 회원가입 할때 휴대폰번호 양식 검사
// 예를 들어 010-1111-2222 라는 전호번호는
// "숫자3개", "-", "숫자4개", "-", "숫자4개" 로 이루어져 있는데,
const regex = /\d{3}-\d{4}-\d{4}/;
// (\d는 숫자를 의미하고, {} 안의 숫자는 갯수를 의미한다.)
regex.test('010-1111-2222') // true;
regex.test('01-11-22') // false;
```

그러나 정규표현식은 주석이나 공백을 허용하지 않고 여러가지 기호를 혼합하여 사용하기 때문에 **가독성이 좋지 않다는 문제**가 있다는 단점이 있다.

정규식 구성

정규식 구성 코드는 다음과 같다.

슬래쉬 문자 두개 사이로 정규식 기호가 들어가는 형태이다. 뒤의 `i` 는 정규식 플래그이다.



JAVASCRIPT

```
// 리터럴 방식
const regex = /abc/;
// 생성자 방식
const regex = new RegExp("abc");
const regex = new RegExp(/abc/); // 이렇게 해도 됨
```

아래 예제는 자바스크립트 코드지만 **대부분의 언어의 정규식 문법은 비슷**하니, 하나의 언어의 정규식을 잘 익혀두면 다른 언어의 정규식을 익히는데 아주 빠르게 학습이 가능하다.

정규식 메서드

위의 정규표현식을 가지고 이메일이나 전화번호 매칭 필터링을 하기위해선 자바스크립트 정규식 메서드를 이용하여 패턴을 검사하고, 매칭되는 문자열을 추출, 변환한다.

메서드	의미
("문자열").match(/정규표현식/플래그)	"문자열"에서 "정규표현식"에 매칭되는 항목들을 배열로 반환
("문자열").replace(/정규표현식/, "대체문자열")	"정규표현식"에 매칭되는 항목을 "대체문자열"로 변환
("문자열").split(정규표현식)	"문자열"을 "정규표현식"에 매칭되는 항목으로 쪼개어 배열로 반환
(정규표현식).test("문자열")	"문자열"이 "정규표현식"과 매칭되면 true, 아니면 false반환
(정규표현식).exec("문자열")	match메서드와 유사(단, 무조건 첫번째 매칭 결과만 반환)

JAVASCRIPT

```
// 정규표현식을 담은 변수
const regex = /apple/; // apple 이라는 단어가 있는지 필터링
// "문자열"이 "정규표현식"과 매칭되면 true, 아니면 false반환
regex.test("Hello banana and apple hahahaha"); // true
// "문자열"에서 "정규표현식"에 매칭되는 항목들을 배열로 반환
const txt = "Hello banana and apple hahahaha";
txt.match(regex); // ['apple']
// "정규표현식"에 매칭되는 항목을 "대체문자열"로 변환
txt.replace(regex, "watermelon"); // 'Hello banana and watermelon hahahaha'
```

정규식 플래그

정규식 플래그는 정규식을 생성할 때 고급 검색을 위한 전역 옵션을 설정할 수 있도록 지원하는 기능이다.

JAVASCRIPT

```
// flags 에 플래그 문자열이 들어간다.
cosnt flags = 'i';
const regex = new RegExp('abapplec', flags);
// 리터럴로 슬래쉬 문자뒤에 바로 표현이 가능
const regex1 = /apple/i;
const regex2 = /apple/gm;
```

Flag	Meaning	Description
i	Ignore Case	대소문자를 구별하지 않고 검색한다.

g	Global	문자열 내의 모든 패턴을 검색한다.
m	Multi Line	문자열의 행이 바뀌더라도 검색을 계속한다.
s		.(모든 문자 정규식)이 개행 문자 ₩n도 포함하도록
u	unicode	유니코드 전체를 지원
y	sticky	문자 내 특정 위치에서 검색을 진행하는 'sticky' 모드를 활성화

g : 전역 검색

- 전역 검색 플래그가 없는 경우에는 최초 검색 결과 한번만 반환하는 반면,
- 전역 검색 플래그가 있는 경우에는 모든 검색 결과를 배열로 반환

JAVASCRIPT

```
// `a`가 두 개 포함된 문자열
const str = "abcabc";
// `g` 플래그 없이는 최초에 발견된 문자만 반환
str.match(/a/); // ["a", index: 0, input: "abcabc", groups: undefined]
// `g` 플래그가 있으면 모든 결과가 배열로 반환
str.match(/a/g); // (2) ["a", "a"]
```

m : 줄바꿈 검색

- 여러 줄의 문자열에서 필터링 해야 될때 사용된다.
- 뒤에서 배울 입력 시작(^) 앵커나 입력 종료($) 앵커는 전체 문자열이 아닌 **각 줄 별로 대응**되게 만들어졌기 때문에, 만일 여러줄을 검색해야 한다면 m 플래그를 사용한다고 보면 된다

JAVASCRIPT

```
// 줄바꿈이 포함된 3줄 문자열
const str = "Hello World and\nPower Hello?\nPower Overwhelming!!";
/*
Hello World and
Power Hello?
Power Overwhelming!!
*/
// Hello 단어로 시작하는지 검사 (^ 문자는 문장 시작점을 의미)
str.match(/^Hello/); // ["Hello"]
// → 첫번째 줄은 잘 찾음
// Power 단어로 시작하는지 검사 (^ 문자는 문장 시작점을 의미)
str.match(/^Power/); // null
// → 그러나 그 다음 줄은 검색되지 아니함
// 따라서 m 플래그를 통해 개행되는 다음 줄도 검색되게 설정
str.match(/^Power/m); // ['Power']
// 세번째 줄도 검색되게 하고싶으면 g 플래그와 혼합 사용
str.match(/^Power/gm); // ['Power', 'Power']
```

i : 대소문자 구분 없음

- 정규식은 기본적으로 대소문자를 구분 (Case sensitive)
- 대신 i 플래그를 통해 대소문자 구분 하지 않을수 있다.

JAVASCRIPT

```
const str = "abcABC";
// 대소문자 a 검색
str.match(/a/gi); // (2) ["a", "A"]
```

정규식 기호 모음

정규식 특정 문자 숫자 매칭 패턴

패턴	의미
a-zA-Z	영어알파벳(-으로 범위 지정)
ㄱ-ㅎ가-힣	한글 문자(-으로 범위 지정)
0-9	숫자(-으로 범위 지정)
.	모든 문자열(숫자, 한글, 영어, 특수기호, 공백 모두) 단, 줄바꿈 X
₩d	숫자
₩D	숫자가 아닌 것
₩w	밑줄 문자를 포함한 영숫자 문자에 대응 [A-Za-z0-9_] 와 동일
₩W	₩w 가 아닌 것
₩s	space 공백
₩S	space 공백이 아닌 것
₩특수기호	특수기호 ₩* ₩^ ₩& ₩! ₩? ...등
₩b	63개 문자(영문 대소문자 52개 + 숫자 10개 + _(underscore))가 아닌 나머지 문자에 일치하는 경계(boundary)
₩B	63개 문자에 일치하는 경계
₩x	16진수 문자에 일치 /₩x61/는 a에 일치
₩0	8진수 문자에 일치 /₩141/은 a에 일치
₩u	유니코드(Unicode) 문자에 일치 /₩u0061/는 a에 일치
₩c	제어(Control) 문자에 일치
₩f	폼 피드(FF, U+000C) 문자에 일치
₩n	줄 바꿈(LF, U+000A) 문자에 일치

₩r	캐리지 리턴(CR, U+000D) 문자에 일치
₩t	탭 (U+0009) 문자에 일치

정규식 검색 기준 패턴

기호	의미
|	OR a|b
[]	괄호안의 문자들 중 하나. or 처리 묶음 보면 된다. /abc/ : "abc"를 포함하는 /[abc]/ : "a" 또는 "b" 또는 "c" 를 포함하는 [다-바] : 다 or 라 or 마 or 바
[^문자]	괄호안의 문자를 제외한 것 [^lgEn] "l" "g" "E" "N" 4개 문자를 제외 ※ 대괄호 안에서 쓰면 제외의 뜻, 대괄호 밖에서 쓰면 시작점 뜻
^문자열	특정 문자열로 시작 (시작점) /^www/
문자열$	특정 문자열로 끝남 (종착점) /com$/

정규식 갯수 반복 패턴

기호	의미
?	없거나 or 최대 한개만 /apple?/
*	없거나 or 있거나 (여러개) /apple*/
+	최소 한개 or 여러개 /apple+/
*?	없거나, 있거나 and 없거나, 최대한개 : 없음 {0}와 동일
+?	최소한개, 있거나 and 없거나, 최대한개 : 한개 {1}와 동일
{n}	n개
{Min,}	최소 Min개 이상
{Min, Max}	최소 Min개 이상, 최대 Max개 이하 {3,5}? == {3}와 동일

정규식 그룹 패턴

기호	의미
()	그룹화 및 캡쳐
(?: 패턴)	그룹화 (캡쳐 X)
(?=)	앞쪽 일치(Lookahead), /ab(?=c)/
(?!)	부정 앞쪽 일치(Negative Lookahead), /ab(?!c)/
(?<=)	뒤쪽 일치(Lookbehind), /(?<=ab)c/
(?<!)	부정 뒤쪽 일치(Negative Lookbehind), /(?<!ab)c/

정규식 그룹 패턴 부분은 꽤나 난이도 있는 정규표현식에 속한다.

어렵고 이해가 잘 안되는 것이 당연하니, 차근차근 알아가보자.

정규식 그룹화

JAVASCRIPT

```
'kokokoko'.match(/ko+/); // "ko"
'kooookoooo'.match(/ko+/); // "koooo"
```

코드를 보면 알수 있듯이, 표현식 ko+는 "o"만 + 를 적용시킨다. ("k" 는 적용안시킴)

그 결과로 "koooo"가 반환되었다.

JAVASCRIPT

```
'kokokoko'.match(/(ko)+/); // "kokokoko", "ko"
'kooookoooo'.match(/(ko)+/); // "ko", "ko"
```

하지만 표현식 (ko)+는 "k"와 "o"를 묶었기(그룹화) 때문에 "ko" 자체를 1회 이상 연속으로 반복되는 문자로 검색하게 된다.

따라서 결과가 "kokokoko"가 반환되었다.

그런데 마지막으로 패턴 ()를 사용한 정규식들의 결과를 잘 보면 일치한 결과가 2개가 나온다.

일부러 한번만 검색되라고, **플래그 g를 사용하지 않았는데** 말이다.

정규식 캡처 기능

패턴 그룹화 ()는 괄호 안에 있는 표현식을 캡처하여 사용한다.

캡처는 일종의 복사본을 생성하는 개념이라고 보면 된다. (복사라는 단어는 이해를 돕기 위해서만 사용하며, 실제 개념과는 다르다)

JAVASCRIPT

```
'kokokoko'.match(/(ko)+/); // "kokokoko", "ko"
```

정규식의 캡처 원리를 알아보자면, 패턴 ()안에 있는 "ko"를 그룹화하여 캡처(복사)한다.

우선 캡처된 표현식은 당장 사용되지 않으며, 그룹화된 "ko"를 패턴 +로 1회 이상 연속으로 반복되는 문자로 검색한다. 그렇게 캡처 외 표현식이 모두 작동하고 난 뒤에 **복사했던(캡처된) 표현식 "ko"가 검색**되는 것이다.

즉, 위의 검색 순서를 정리하자면 다음과 같게 된다.

그룹화된 "ko"를 패턴 +로 1회 이상 연속으로 반복하여 검색하여 "kokokoko"를 반환하고

캡처된 "ko"로 검색하여 "ko"를 추가 반환

JAVASCRIPT

```
'123abc'.match(/(\d+)(\w)/); // "123a", "123", "a"
/*
1. 패턴 ()안의 표현식을 순서대로 캡처. \d+, \w
2. 캡처 후 남은 표현식으로 검색.
3. 패턴 \d로 숫자를 검색하되 패턴 +로 1개 이상 연속되는 숫자를 검색 → "123"
4. 다음 패턴 \w는 문자를 검색하니 "a"가 일치
5. 최종적으로 "123a"가 반환.
6. 첫 번째 캡처한 표현식 \d+로 숫자를 재검색하되 패턴 +로 1개 이상 연속되는 숫자를 검색
7. "123"가 일치하여 반환
8. 나머지 캡처한 표현식 \w로 문자를 검색하니 "a"가 일치하여 반환
*/
```

캡처하지 않는 그룹화 (?:)

위에서 살펴봤듯이 뜻하지않은 정규식 그룹화 캡쳐 기능 때문에 쓸데없는 결과값을 얻는 것이 싫다면, 괄호 안에 ?: 문자를 씀으로써 캡쳐를 비활성화 할 수 있다.

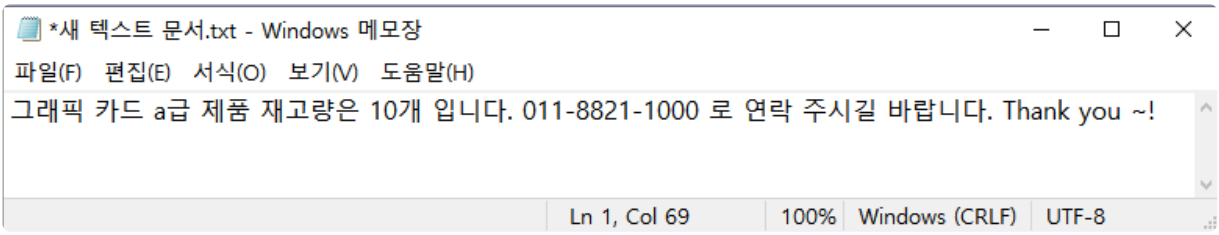
따라서 표현식 캡처를 하지 않기 때문에 "k"와 "o"를 그룹화한 "ko"만으로 검색되게 된다.

JAVASCRIPT

```
// 그룹화 + 캡처
'kokokoko'.match(/(ko)+/); // "kokokoko", "ko"
// 그룹화만
'kokokoko'.match(/(?:ko)+/); // "kokokoko"
```

정규표현식 연습 예제

정규식 문장 연습하기



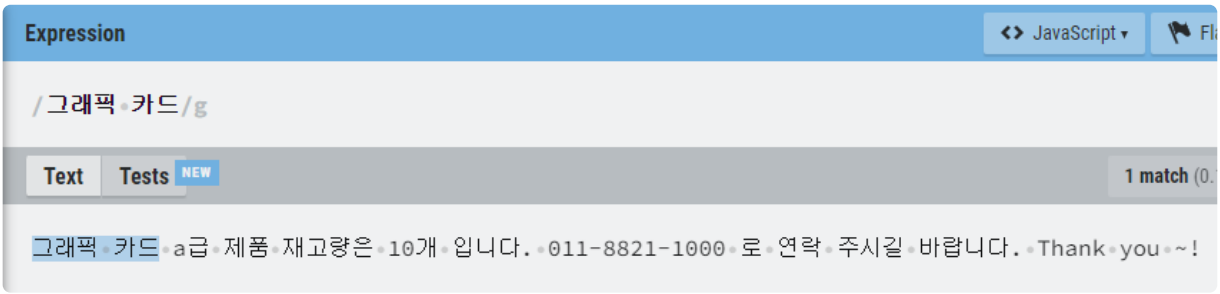
샘플 문자열

/다/ : '다'를 '하나'만 찾는다.

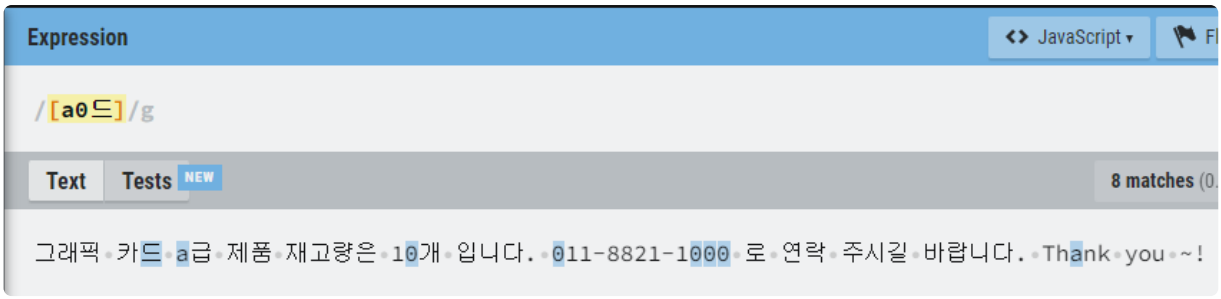
/다/g : '다'를 '모두' 찾는다

* g(global) 플래그를 뒤에 붙인걸 기억하자.

/그래픽 카드/ : '그래픽 카드'를 찾는다.



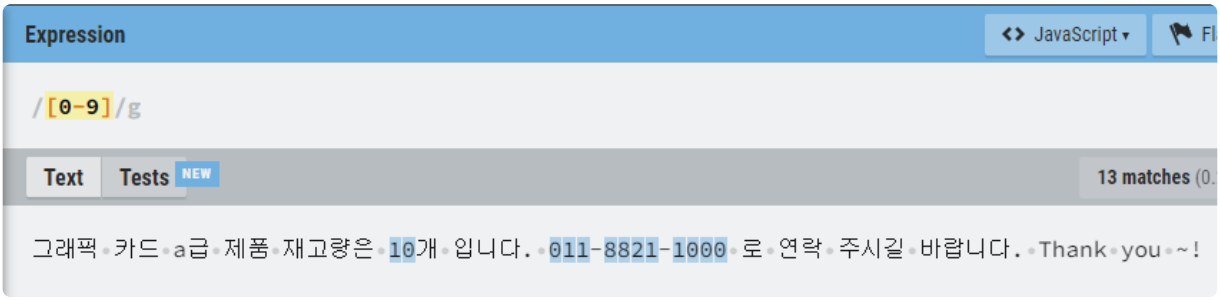
/[a0드]/g : "드", "a", 0 중에 하나라도 포함된 것을 모두 찾는다.



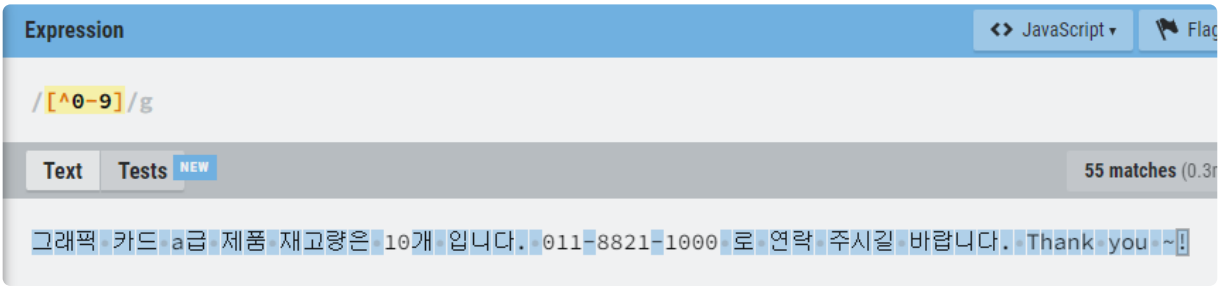
* 대괄호[]는 OR의 기능을 한다.

/[0-9]/g : '숫자0~9'를 모두 찾는다.

⋯› 대나무 빨대 a급 제품은 *10*개 남아있습니다. *010-1111-2222* 로 Call Me~!



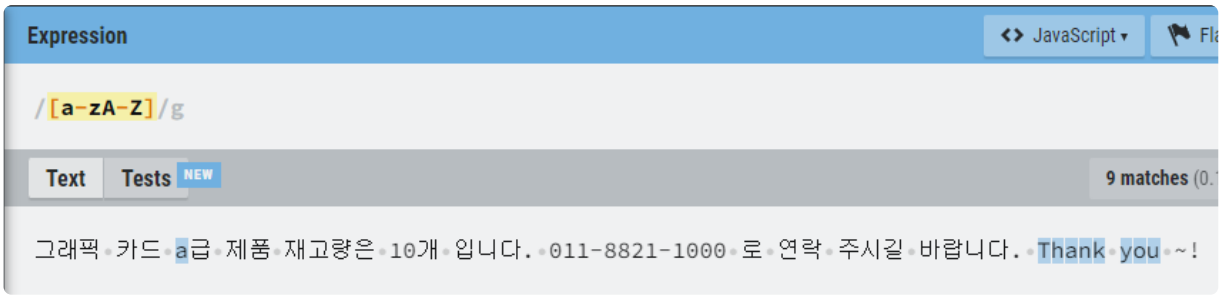
/[^0-9]/g : '숫자0~9'가 아닌 것을 모두 찾는다.



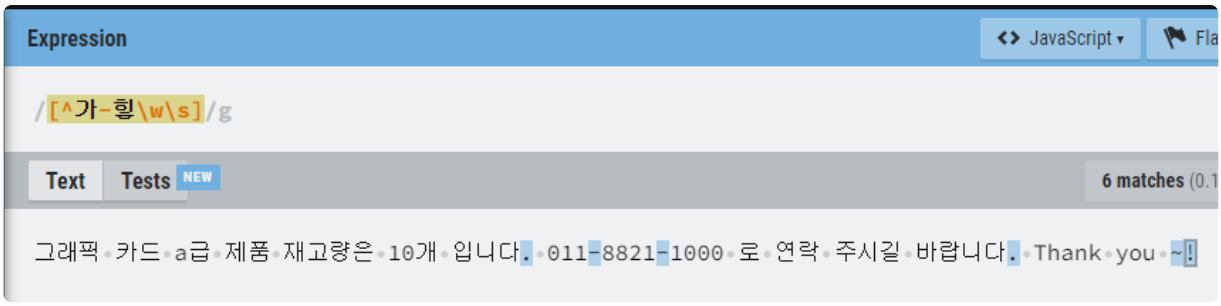
* 대괄호[] 안에서 앞에 ^를 쓰면, 부정(Not)의 기능을 한다.

/[a-zA-Z]/g : '영어알파벳 대문자/소문자'를 모두 찾는다.

⋯› 대나무 빨대 *a*급 제품은 10개 남아있습니다. 010-1111-2222 로 *Call Me*~!

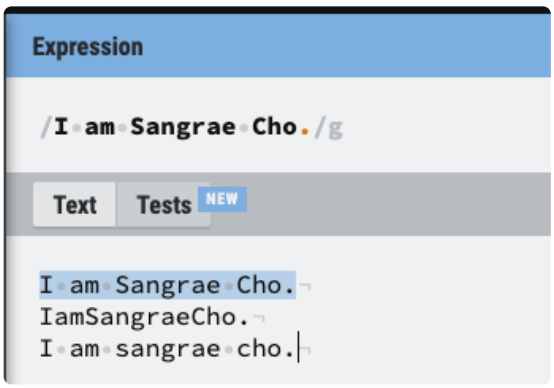


/[^가-힣₩w₩s]/g : 한글, 영문, 숫자, 공백을 제외한 '특수 문자'만 찾는다

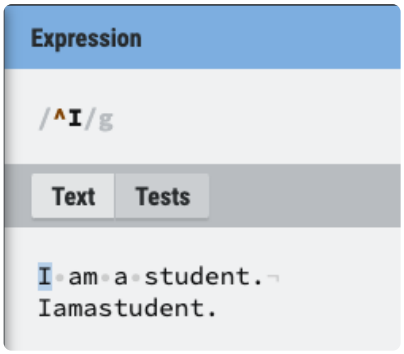


정규식 기호 개별 연습하기

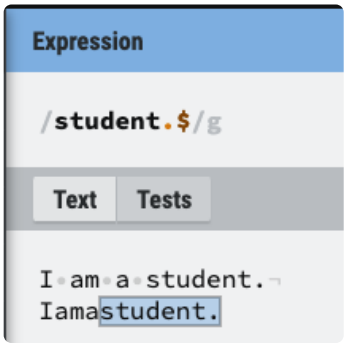
- 기본적으로 문자 있는 그대로 입력하여 선택이 가능
- 정규식은 띄어쓰기까지 구분
- 정규식은 영어 대소문자 구분



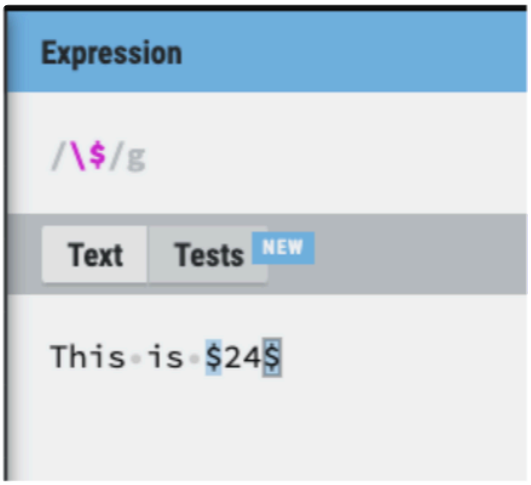
- **^(캐럿)** : ^ 뒤에 나오는 문자로 시작되는 소스

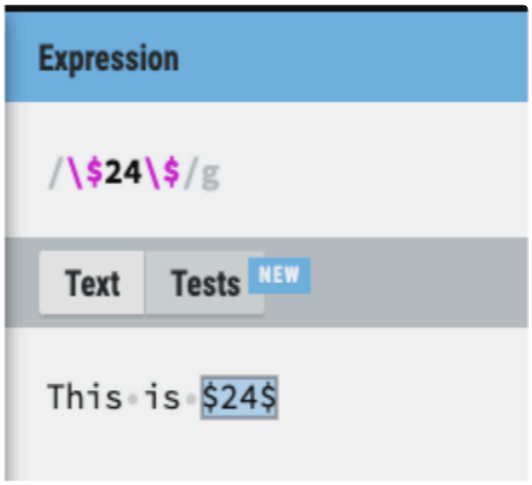


- **$** : $ 앞에 나오는 문자로 끝나는 소스

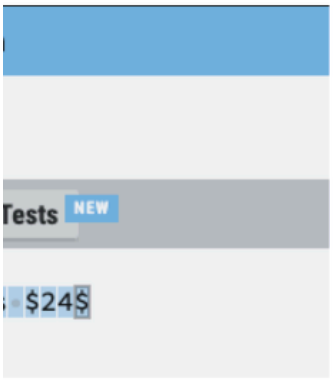


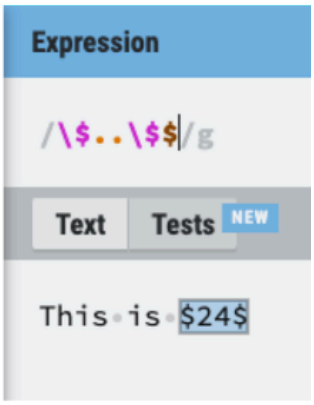
- **₩(백슬래쉬)** : 이스캐이핑 → ^, $ 처럼 특수한 기능을 가진 문자를 일반 문자로



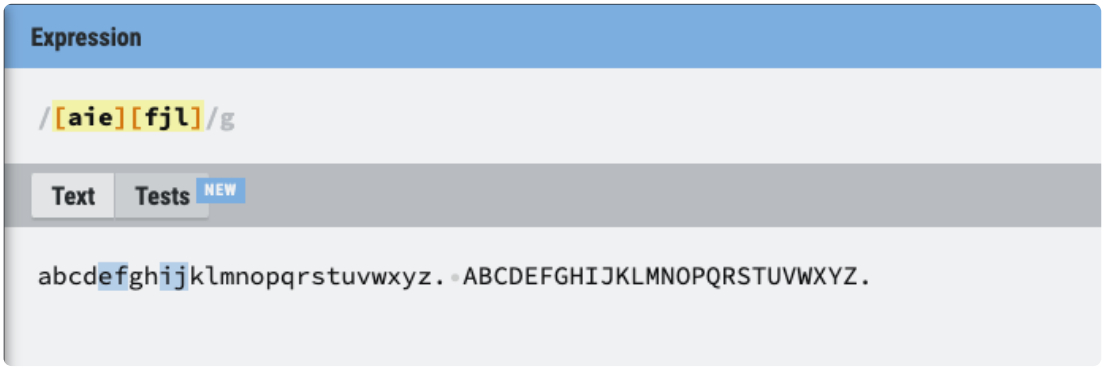


- **.(점)** : 문자, 공백, 특수문자 등의 모든 텍스트 .(점)의 개수로 여러단위 선택 가능

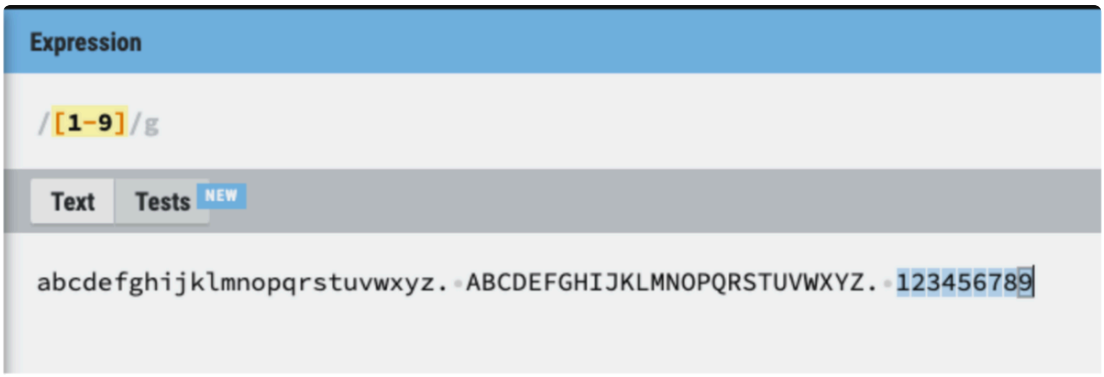


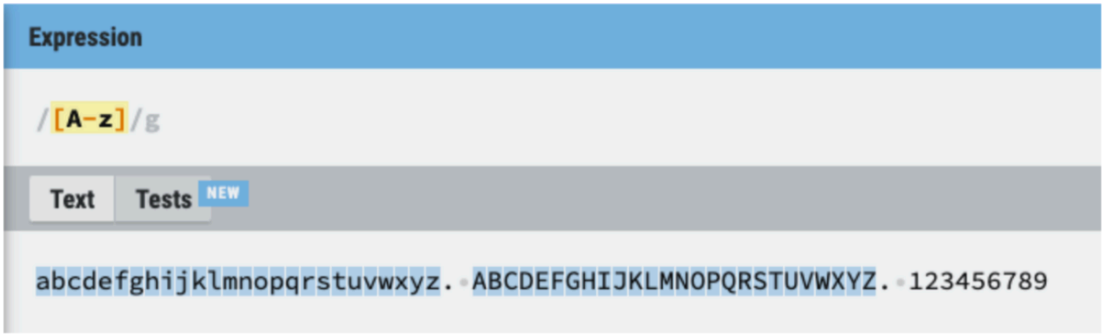


- **[] (Square Braket)** : 안에 들어가는 문자 단위 하나하나 (or)

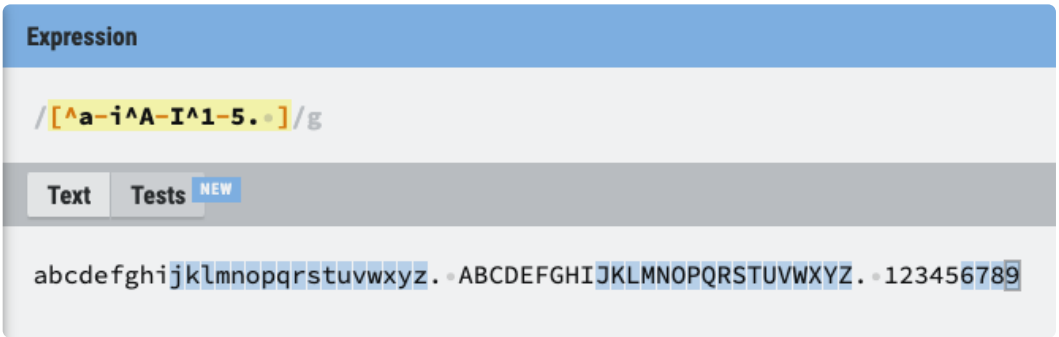


- [1-9] / [a-z] / [A-z] 와 같이 범위 설정이 가능

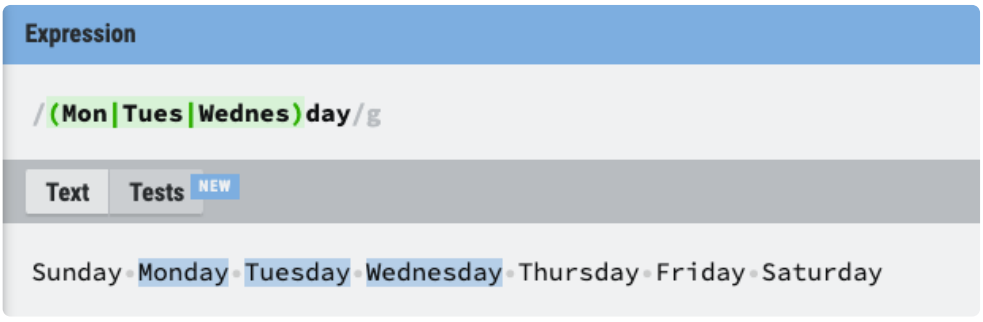




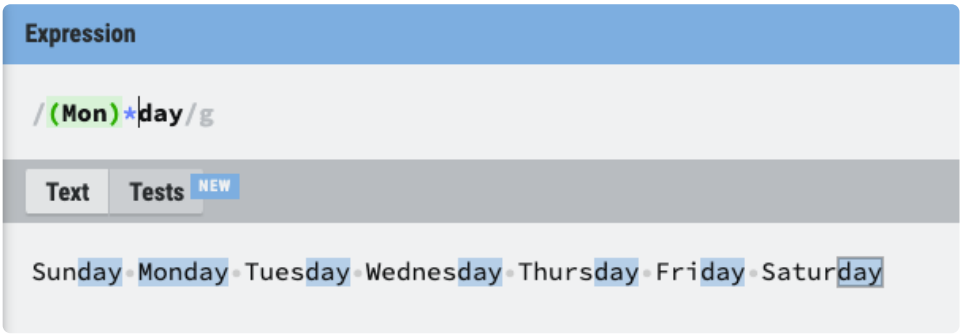
- [^x-z] 와 같이 ^(캐럿)을 붙여서 부정(제외)의 뜻



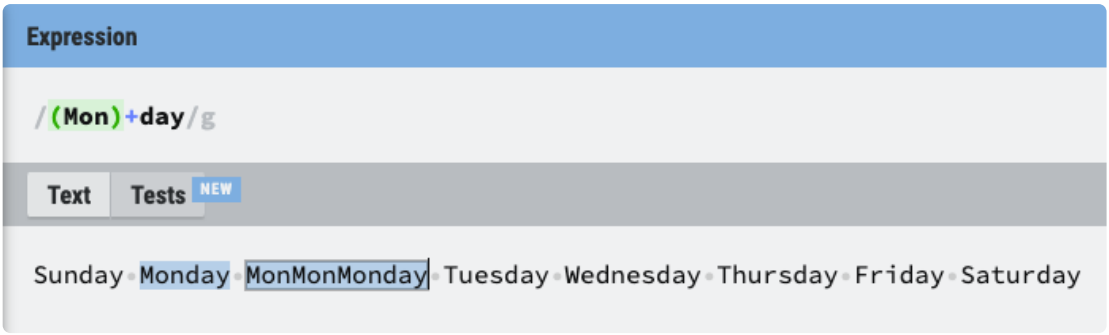
- (문자|문자|문자) : () 안에 들어간 문자를 선택 |(파이프라인)을 이용해 나눌 수 있다.



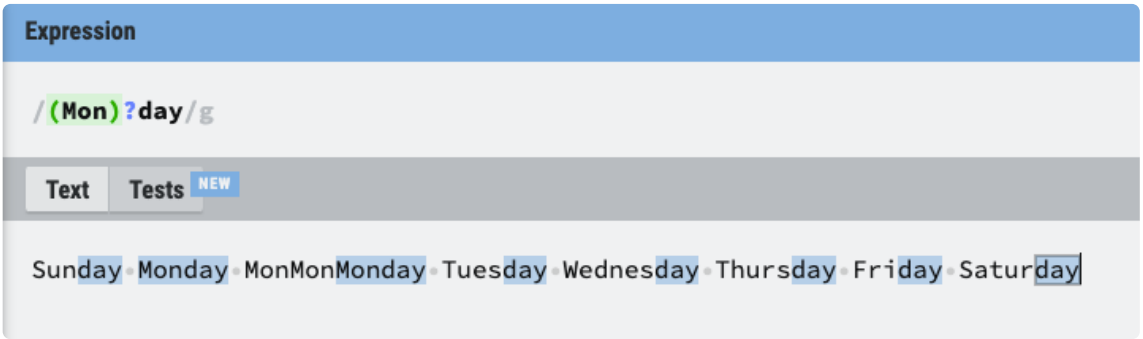
- `*` : * 앞에 입력된 문자가 있거나 없을 경우



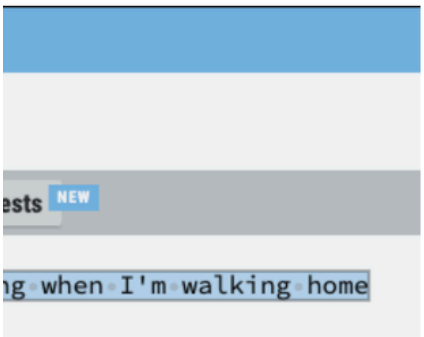
- `+` : +앞에 입력된 문자가 하나에서 여러개 (최소 하나)

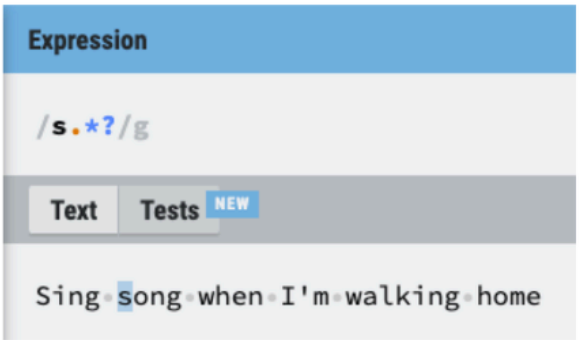


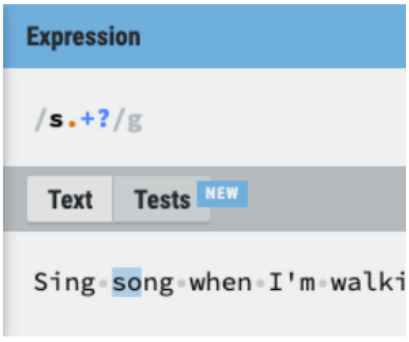
- `?` : ?앞에 입력된 문자가 없거나 하나인 경우 (최대 하나)



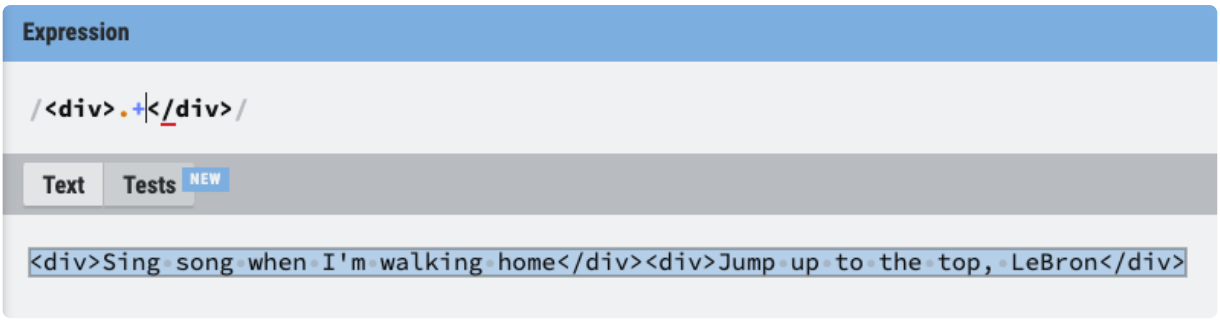
- 수량자에 ?를 붙여주어 수량자의 최소 숫자로 고정이 가능하다. (*? - 0개, +? - 1개)





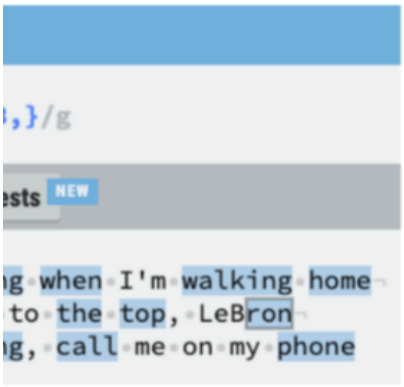


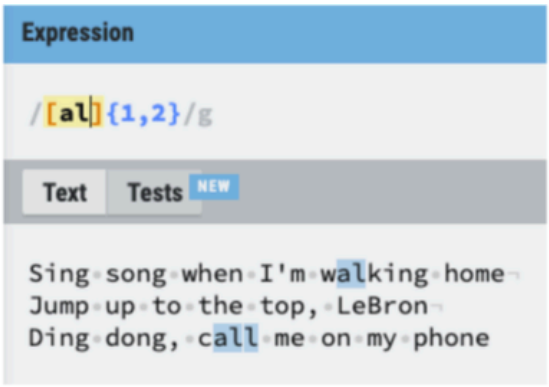
- Greedy : 해당하는 모든 텍스트를 선택

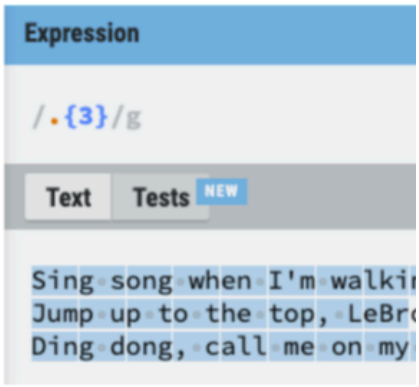


- Non-Greedy (Lazy) : 해당하는 부분의 텍스트를 선택

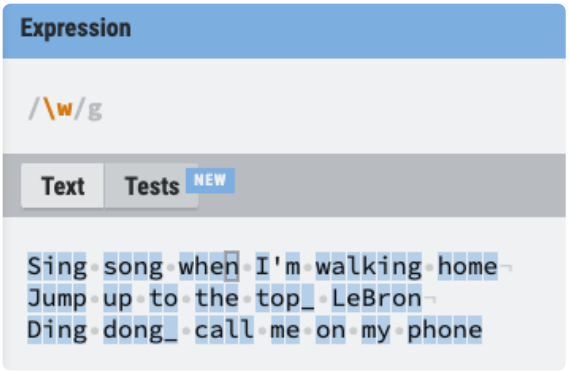
- {} : 숫자로 단위를 표현하여 선택
  - {n, m} 앞에 입력된 문자가 n개 이상 m개 미만
  - {n,} 뒤의 인자를 비워두면 n개 이상을 의미



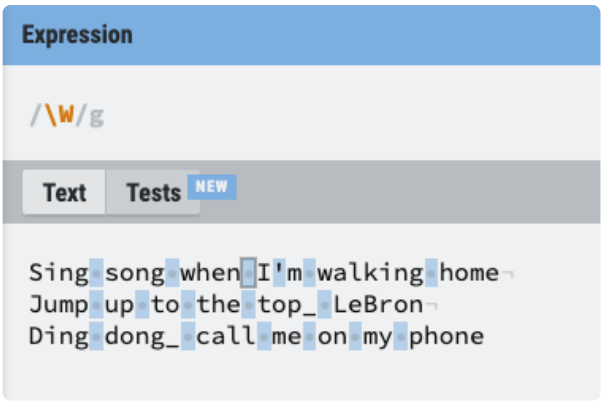




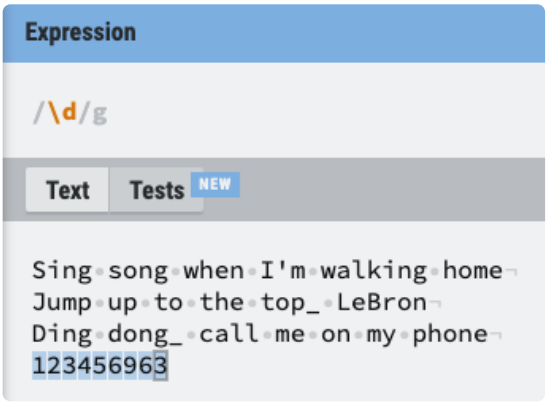
- ₩w : 모든 문자 [A-z0-9] 와 _(언더바) 까지 포함



- ₩W : 대문자 W는 w와 정반대의 의미



- ₩d : 숫자를 의미 (₩D는 반대의 의미)



- ₩b(바운더리) : 단어의 단위로 선택 (₩B는 반대의 의미)

- 전/후방 탐색 (?=g) : 특정 문자 앞까지만 조회 (모든 문자열 3개를 선택할 건데 g앞에 붙은 것만 조회)

정규표현식 샘플 코드

정규식 실무 예제

특정 단어로 끝나는지 검사

JAVASCRIPT

```
const fileName = 'index.html';
// 'html'로 끝나는지 검사
// $ : 문자열의 끝을 의미한다.
const regexr = /html$/;
```

숫자로만 이루어져 있는지 검사

JAVASCRIPT

```
const targetStr = '12345';
// 모두 숫자인지 검사
// [] 바깥의 ^는 문자열의 처음을 의미한다.
const regexr = /^\d+$/;
```

아이디 사용 검사

- 알파벳 대소문자 또는 숫자로 시작하고 끝나며 4 ~10자리인지 검사

JAVASCRIPT

```
const id = 'abc123';
// 알파벳 대소문자 또는 숫자로 시작하고 끝나며 4 ~10자리인지 검사
// {4,10}: 4 ~ 10자리
const regexr = /^[A-Za-z0-9]{4,10}$/;
```

핸드폰 번호 형식

JAVASCRIPT

```
const cellphone = '010-1234-5678';
const regexr = /^\d{3}-\d{3,4}-\d{4}$/;
```

웹사이트 주소 형식

- http:// 나 https://로 시작하고, 알파벳, 어더스코어(_), 하이픈(-), dot(.)으로 이루어져 있는 정규식

JAVASCRIPT

```
const text =
`http://dogumaster.com http://google.com 010-1111-2222 02-333-7777 curryyou@aaa.com`;
text.match(/https?:\/\/[\w\-\.]+/g); // ["http://dogumaster.com", "http://google.com"]
/*
1) http ⇒ 로 시작하고,
2) s? ⇒ 다음에 s는 없거나, 있고,
3) \/\/ => 다음에 특수기호 // 가 오고
4) [\w\-\.]+ ⇒ \w(영문자, 언더스코어), 하이픈, 쩜 으로 이루어진 문자열이 한개 이상(+) 있다.
5) g ⇒ 매칭되는걸 모두 다 찾는다.(플래그)
*/
```

전화번호 형식

- 유선번호라면 02-111-2222 형식이고, 핸드폰번호라면 010-1111-2222 형식을 모두 포함하는 정규식 (숫자의 갯수가 다름)

JAVASCRIPT

```
const text =
`http://dogumaster.com http://google.com 010-1111-2222 02-333-7777 curryyou@aaa.com`;
text.match(/\d{2,3}-\d{3,4}-\d{4}/g); // [ '010-1111-2222', '02-333-7777' ]
/*
1) \d{2,3} ⇒ 숫자 2~3개로 시작하고,
2) \- => 다음에 하이픈(-)이 오고
3) \d{3, 4} ⇒ 다음에 숫자가 3~4개 오고,
4) \- => 다음에 하이픈(-)이 오고,
5) \d{4} ⇒ 다음에 숫자가 4개 온다.
6) g ⇒ 매칭되는걸 모두 다 찾는다(플래그)
*/
```

이메일주소 형식

- xxx@xxxx.com 등의 형식

JAVASCRIPT

```
const text = `http://dogumaster.com http://google.com 010-1111-2222 02-333-7777 curryyou@aaa.com`;
text.match(/[\w\-\.]+\@[\w\-\.]+/g); // [ 'curryyou@aaa.com' ]
```

JAVASCRIPT

```
// 좀더 엄격한 검사가 필요하다면, 상황에 맞게 수정해서 사용면 된다.
const email = 'ungmo2@gmail.com';
```

```
const regexr = /^[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*@[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*\.[a-zA-Z]{2,3}$/;
```

특수기호 정규표현식

JAVASCRIPT

```
// 모든 특수기호를 나열
const regex = /\[\]\{\}\/\(\)\.\?\<\>!@#$%^&*/g
// 문자와 숫자가 아닌것을 매칭
const regex = /[^a-zA-Z0-9가-힣ㄱ-ㅎ]/g
```

이밖의 정규표현식 모음

JAVASCRIPT

```
/* 전화번호 */
var localPhone = /^(0(2¦3[1-3]¦4[1-4]¦5[1-5]¦6[1-4]))(\d{3,4})(\d{4})$/;
var cellPhone = /^(?:(010\d{4})¦(01[1¦6¦7¦8¦9]-\d{3,4}))(\d{4})$/;
/* 숫자 형식 */
var number = /[0-9]/;
var unsignedInt  = /^[1-9][0-9]*$/;
var notNumber = /[^(0-9)]/gi;
/* 문자 형식 */
var korea_cv = /[ㄱ-ㅎ¦ㅏ-ㅣ]/;
var korea = /[가-힣]/;
var koreaName = /[가-힣]/;
var english = /[a-z ¦ A-Z]/;
/* 특문 */
var special_char = /[\{\}\[\]\/?.,;:¦\)*~`!^\-+<>@\#$%&\\\=\(\'\"]/;
var comma_char = /,/g;
var blank = /[\s]/g;
/* 아이디 / 비밀번호 */
var id_check = /^[a-z ¦ A-Z]{3,6}[0-9]{3,6}$/;
var password =/^.*(?=.{6,20})(?=.*[0-9])(?=.*[a-zA-Z]).*$/;
/* 이메일 형식 */
var email =/([\w-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)¦(([\w-]+\.)+))([a-zA-Z]{2,4}¦[0-9]{1,3})(\]?)$/;
/* 도메인 형식 */
var domain_all =/([\w-\.]+)@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.)
¦(([\w-]+\.)+))([a-zA-Z]{2,4}¦[0-9]{1,3})(\]?)$/;
var domain_include = /^((http(s?))\:\/\/)([0-9a-zA-Z\-]+\.)+[a-zA-Z]{2,6}(\:[0-9]+)?(\/\S*)?$/;
var domain_exclude = /^[^((http(s?))\:\/\/)]([0-9a-zA-Z\-]+\.)+[a-zA-Z]{2,6}(\:[0-9]+)?(\/\S*)?$/;
/* 영문 한글만 */
var ko_en_num_charactor = /^[가-힣a-zA-Z0-9]*$/;
var ko_en_charactor = /^[가-힣a-zA-Z]*$/;
/* 자동차 번호판 */
var car = /^[0-9]{2}[\s]*[가-힣]{1}[\s]*[0-9]{4}$/;
```