31장 RegExp(정규표현식)

정규 표현식이란?

Regular Expression

일정한 패턴을 가진 문자열의 집합을 표현하기 위해 사용하는 형식 언어

- 대부분의 프로그래밍 언어와 코드 에디터에 내장되어 있음
- 문자열을 대상으로 패턴 매칭 기능
- 반복문, 조건문 없이 패턴 정의하고 테스트하는 것으로 문자열 체크 가능
- 주석, 공백을 허용하지 않고 여러 기호가 혼합되어 가독성 좋지 않음

```
> const tel1 = '010-1234-5678';
  const tel2 = '+82-10-1234-5678';

  const telReg = /^\d{3}-\d{4}-\d{4}$/;

  console.log(telReg.test(tel1));
  console.log(telReg.test(tel2));
true
false
```

정규 표현식의 생성

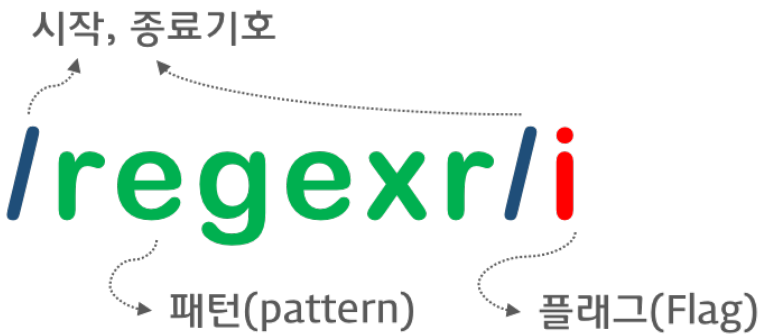
- 정규 표현식 리터럴과 RegExp 생성자 함수 사용
- 패턴과 플래그로 구성됨

```
const target = 'Is this all there is?';

const regexp = /is/i;

regexp.test(target);
true
```

```
> const regexp2 = new RegExp(/is/i);
  regexp2.test(target);
< true
> const regexp3 = new RegExp(/is/, 'i');
  regexp3.test(target);
< true
> const regexp4 = new RegExp('is', 'i');
  regexp4.test(target);
< true
```



정규 표현식의 생성

생성자 함수를 사용하면 변수를 사용해 동적으로 RegExp 객체를 생성할 수 있다

```
> const count = (str, char) => (str.match(new RegExp(char, 'gi')) ?? []).length;

  console.log(count('Is this all there is?', 'is'));
  console.log(count('Is this all there is?', 'x'));
3
0
```

RegExp 메서드

- RegExp.prototype.exec

매칭 결과를 배열로 반환. 매칭 결과가 없는 경우 null.

g 플래그를 지정해도 첫 번째 매칭 결과만 반환

```
> const target = 'Is this all there is?';
  const regExp = /is/;

  regExp.exec(target);
◂ ▶ ['is', index: 5, input: 'Is this all there is?', groups: undefined]

> const target = 'Is this all there is?';
  const regExp = /is/g;

  regExp.exec(target);
◂ ▶ ['is', index: 5, input: 'Is this all there is?', groups: undefined]
```

RegExp 메서드

- RegExp.prototype.test

매칭 결과를 불리언 값으로 반환

```
> const target = 'Is this all there is?';
  const regExp = /is/;

  regExp.test(target);
< true
```

RegExp 메서드

- String.prototype.match

대상 문자열과 인수로 전달받은 정규 표현식과의 매칭 결과를 배열로 반환.

g 플래그 지정되면 모든 매칭 결과 반환

```
> const target = 'Is this all there is?';
  const regExp = /is/;

  target.match(regExp);
◂ ▶ ['is', index: 5, input: 'Is this all there is?', groups: undefined]
> const target = 'Is this all there is?';
  const regExp = /is/g;

  target.match(regExp);
◂ ▶ (2) ['is', 'is']
```

RegExp 플래그

Flag	설명
g	Global - 문자열 내의 모든패턴을 찾습니다.
i	Ignore Case -문자열의 대소문자를 구별하지 않습니다.
m	Multi Line - 문자열이 행이 바뀌어도 찾습니다.

```
> const target = 'Is this all there is?';
  target.match(/is/);
▶ ['is', index: 5, input: 'Is this all there is?', groups: undefined]
> target.match(/is/i);
▶ ['Is', index: 0, input: 'Is this all there is?', groups: undefined]
> target.match(/is/g);
▶ (2) ['is', 'is']
> target.match(/is/ig);
▶ (3) ['Is', 'is', 'is']
```

RegExp 패턴

표현식	의미
^x	문자열의 시작을 표현하며 x 문자로 시작됨을 의미한다.
x$	문자열의 종료를 표현하며 x 문자로 종료됨을 의미한다.
.x	임의의 한 문자의 자리수를 표현하며 문자열이 x 로 끝난다는 것을 의미한다.
x+	반복을 표현하며 x 문자가 한번 이상 반복됨을 의미한다.
x?	존재여부를 표현하며 x 문자가 존재할 수도, 존재하지 않을 수도 있음을 의미한다.
x*	반복여부를 표현하며 x 문자가 0번 또는 그 이상 반복됨을 의미한다.
x|y	or 를 표현하며 x 또는 y 문자가 존재함을 의미한다.
(x)	그룹을 표현하며 x 를 그룹으로 처리함을 의미한다.
(x)(y)	그룹들의 집합을 표현하며 앞에서 부터 순서대로 번호를 부여하여 관리하고 x, y 는 각 그룹의 데이터로 관리된다.
(x)(?:y)	그룹들의 집합에 대한 예외를 표현하며 그룹 집합으로 관리되지 않음을 의미한다.
x{n}	반복을 표현하며 x 문자가 n번 반복됨을 의미한다.
x{n,}	반복을 표현하며 x 문자가 n번 이상 반복됨을 의미한다.
x{n,m}	반복을 표현하며 x 문자가 최소 n번 이상 최대 m 번 이하로 반복됨을 의미한다.

표현식	의미
[xy]	문자 선택을 표현하며 x 와 y 중에 하나를 의미한다.
[^xy]	not 을 표현하며 x 및 y 를 제외한 문자를 의미한다.
[x-z]	range를 표현하며 x ~ z 사이의 문자를 의미한다.
\^	escape 를 표현하며 ^ 를 문자로 사용함을 의미한다.
\b	word boundary를 표현하며 문자와 공백사이의 문자를 의미한다.
\B	non word boundary를 표현하며 문자와 공백사이가 아닌 문자를 의미한다.
\d	digit 를 표현하며 숫자를 의미한다.
\D	non digit 를 표현하며 숫자가 아닌 것을 의미한다.
\s	space 를 표현하며 공백 문자를 의미한다.
\S	non space를 표현하며 공백 문자가 아닌 것을 의미한다.
\t	tab 을 표현하며 탭 문자를 의미한다.
\v	vertical tab을 표현하며 수직 탭(?) 문자를 의미한다.
\w	word 를 표현하며 알파벳 + 숫자 + _ 중의 한 문자임을 의미한다.
\W	non word를 표현하며 알파벳 + 숫자 + _ 가 아닌 문자를 의미한다.

RegExp 패턴

메일 주소 형식 정규식

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