

성별을 선택해주세요. \*

☐ 여성

☐ 남성

키를 기입해주세요. \*

단답형 텍스트

체중을 기입해주세요. \*

단답형 텍스트

상품 증정을 위한 휴대폰 번호를 입력해주세요.  
(선택사항) 010-xxxx-xxxx형식으로 기입해주세요.

단답형 텍스트

# Regular Expression

시작, 종료기호

`/regexr/i`

패턴(pattern)      플래그(flag)

# RegExp.prototype.test()

주어진 문자열이 정규 표현식을 만족하는지 판별하고, 그 여부를 true 또는 false로 반환



```
// RegExp.prototype.test()  
var tel = '010-1234-5678팔';  
// 정규 표현식 리터럴로 휴대폰 전화번호 패턴을 정의한다.  
var regexp = /^[0-1]*-?[0-9]*-?[0-9]*$/;  
regexp.test(tel); // false  
  
var tel = '010-1234-5678';  
// 정규 표현식 리터럴로 휴대폰 전화번호 패턴을 정의한다.  
var regexp = /^[0-1]*-?[0-9]*-?[0-9]*$/;  
regexp.test(tel); // true
```

# RegExp.prototype.exec()

주어진 문자열에서 일치 탐색을 수행한 결과를 배열 또는 null로 반환



```
// RegExp.prototype.exec()
var tel = '010-1234-5678팔';
// 정규 표현식 리터럴로 휴대폰 전화번호 패턴을 정의한다.
var regexp = /^[0-1]*-?[0-9]*-?[0-9]*$/;
regexp.exec(tel); // null

var tel = '010-1234-5678';
// 정규 표현식 리터럴로 휴대폰 전화번호 패턴을 정의한다.
var regexp = /^[0-1]*-?[0-9]*-?[0-9]*$/;
regexp.exec(tel); // ["010-1234-5678", index: 0, input: "010-1234-5678", groups: undefined]

var str = 'Is there all there is?';
var regexp = /is/;
regexp.exec(str); // ["is", index: 19, input: "Is there all there is?", groups: undefined]
```

# String.prototype.match()


주어진 문자열이 정규식과 매치되는 부분을 검색하여 배열 또는 null로 반환



```
// String.prototype.match()  
var str = 'Is there all there is?'  
// 정규 표현식 리터럴로 패턴을 정의한다.  
var regExp = /is/;  
str.match(regExp); // ["is", index: 19, input: "Is there all there is?", groups: undefined]  
  
var str = 'Is there all there is?'  
// 정규 표현식 리터럴로 패턴을 정의한다.  
var regExp = /is/g;  
str.match(regExp); // ["is", "is"]
```

# String.prototype.seach()

주어진 문자열이 정규식과 매치되는 것의 인덱스 또는 -1 반환



```
// String.prototype.seach( )
var str = 'Is there all there is?'
// 정규 표현식 리터럴로 패턴을 정의한다.
var regExp = /i/;
// str에서
str.search(regExp); // 19

var str = 'Is there all there is?'
// 정규 표현식 리터럴로 패턴을 정의한다.
var regExp = /t/;
// str에서
str.search(regExp); // 3

var str = 'Is there all there is?'
// 정규 표현식 리터럴로 패턴을 정의한다.
var regExp = /th/;
// str에서
str.search(regExp); // 3
```

# String.prototype.replace()

주어진 문자열에서 정규식과 일치하는 일부 또는 모든 부분이 교체된 새로운 문자열 반환

```

// String.prototype.replace()
var str = 'hello hello';
// 정규 표현식 리터럴로 패턴을 정의한다.
var regexp = /hello/gi;
str.replace(regexp, 'Ho')

var content = '생활코딩 : http://opentutorials.org/course/1 입니다. 네이버 : http://naver.com 입니
다. ';
// 정규 표현식 리터럴로 패턴을 정의한다.
var urlPattern = /\b(?:https?):\/\/[a-z0-9-+&@#\/%?=~_|!:\.,;]*\/gim;
content.replace(urlPattern, function(url){
    return '<a href="' + url + '>' + url + '</a>';
});
// "매쉬업 : <a href=\"https://www.mash-up.it/\">https://www.mash-up.it/</a> 입니다. 네이버 : <a
href=\"http://naver.com\">http://naver.com</a> 입니다. "
```



# String.prototype.split()

주어진 문자열을 구분자를 이용하여 여러 개의 문자열로 나눔



```
// String.prototype.split()  
var str = 'How are you doing?';  
// 정규 표현식 리터럴로 패턴을 정의한다.  
var regExp = /\s/;  
// split의 첫번째 인수로 전달한 패턴을 검색하여 문자열을 구분한 후 분리된 각 문자열로 이루어진 배열을 반환한다.  
str.split(regExp); // ["How", "are", "you", "doing?"]  
  
var str = 'How are you doing?';  
// 정규 표현식 리터럴로 패턴을 정의한다.  
var regExp = /'/;  
// split의 첫번째 인수로 전달한 패턴을 검색하여 문자열을 구분한 후 분리된 각 문자열로 이루어진 배열을 반환한다.  
str.split(regExp); // ["How are you doing?"]
```

# 방대한 양의 표현식...

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.
$\backslash b$	word boundary를 표현하며 문자와 공백사이의 문자를 의미한다.
$\backslash B$	non word boundary를 표현하며 문자와 공백사이가 아닌 문자를 의미한다.
$\backslash d$	digit 를 표현하며 숫자를 의미한다.
$\backslash D$	non digit 를 표현하며 숫자가 아닌 것을 의미한다.
$\backslash s$	space 를 표현하며 공백 문자를 의미한다.
$\backslash S$	non space를 표현하며 공백 문자가 아닌 것을 의미한다.

Flag	의미
$g$	Global 의 표현하며 대상 문자열내에 모든 패턴들을 검색하는 것을 의미한다.
$i$	Ignore case 를 표현하며 대상 문자열에 대해서 대/소문자를 식별하지 않는 것을 의미한다.
$m$	Multi line을 표현하며 대상 문자열이 다중 라인의 문자열인 경우에도 검색하는 것을 의미한다.

# 하지만 외울 필요 없어요!

특정 단어로 시작하는지 검사	/^https?:\w/\w//
특정단어로 끝나는지 검사	/html\$/
숫자로만 이루어진 문자열인지 검사	/^\wd+\$/
아이디로 사용 가능한지 검사 Ex) 알파벳 대소문자 또는 숫자로 시작하고 끝나며, 4 ~ 10자리인지 검사	/^[A-Za-z0-9]{4,10}\$/
메일 주소 형식에 맞는지 검사	/^([\w-]+(?:\w.[\w-]+)*)@((?:(\w-+\.)*\w[\w-]{0,66})\w\.([a-z]{2,6}(?:\w.[a-z]{2})?)?)\$/
핸드폰 번호 형식에 맞는지 검사	/^\wd{3}-\wd{3,4}-\wd{4}\$/
특수 문자 포함 여부 검사	/#[ㄱ-ㅎ ㅏ-ㅣ 가-힣 \w]+/

Regexper <http://www.regexper.com>

Regexr <http://www.regexr.com/>

# 언제 사용할까요?

```
{payload
  ? payload.split(" ").map((word, index) =>
    /[ㄱ-ㅎ|ㅌ-ㅣ|가-힣|\w]+/.test(word) ? (
      <React.Fragment key={index}>
        <Link key={index} to={`/hashtags/${word}`}>
          {word}
        </Link>{" "}
      </React.Fragment>
    ) : (
      <React.Fragment key={index}>{word} </React.Fragment>
    )
  )
: null}
```

# 언제 사용할까요?



```
def preprocessing(review, okt, remove_stopwords = False, stop_words = []):
    review_text = re.sub("[^가-힣ㄱ-ㅎㅏ-ㅣ\\s]", "", review)
    word_review = okt.morphs(review_text, stem=True)

    if remove_stopwords:
        word_review = [token for token in word_review if not token in stop_words]
    return word_review

stop_words = [ '은', '는', '이', '가', '하', '아', '것', '들', '의', '있', '되', '수', '보', '주',
                '등', '한', '을', '를', '으로', '로' ]
okt = Okt()
clean_train_review = []

for review in tqdm(train_data):
    if type(review) == str:
        clean_train_review.append(preprocessing(review, okt, remove_stopwords = True,
                                                stop_words=stop_words))
    else:
        clean_train_review.append([])

train_tokenizer = Tokenizer()
train_tokenizer.fit_on_texts(clean_train_review)
train_sequences = train_tokenizer.texts_to_sequences(clean_train_review)
MAX_SEQUENCE_LENGTH = 30
train_inputs = pad_sequences(train_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post')
```

우리 앞에 차량 경로 같으니 달아 가.  
앞차를 졸졸 따라.  
앞에 있는 차 뒤쫓아.  
앞차 속도대로 운전하면 되겠다.  
앞차 간격 유지하고 붙어 쫓.  
앞차 뒤에만 있으면 돼.  
앞차 경로대로 같이 가.  
앞차 뒤에 붙어서 따라 갈게.  
앞에 차가 가장 방향대로 운전하면 돼.  
지금 여기 앞 있는 차 따라서 주행할 부탁해.  
앞 차 속도 맞춰 운전해 쫓.

```
[['우리', '앞', '에', '차', '랑', '경로', '같다', '달다', '가다'],
 ['앞차', '졸졸', '따르다'],
 ['앞', '에', '있다', '차', '뒤', '쫓다'],
 ['앞차', '속도', '대로', '운전', '하다', '되다'],
 ['앞차', '간격', '유지', '하고', '붙다', '주다'],
 ['앞차', '뒤', '에만', '있다', '돼다'],
 ['앞차', '경로', '대로', '같이', '가다'],
 ['앞차', '뒤', '에', '붙다', '따르다', '갈다'],
 ['앞', '에', '차갑다', '가장', '방향', '대로', '운전', '하다', '돼다'],
```

끗!





























