문자열

따옴표

javascript 문자열은 " 문자로 묶어도 되고, ' 문자로 묶어도 된다.

```
let s1 = "hello";
let s2 = 'world';

console.log(s1, s2);
```

```
"hello" "world"
```

따옴표 문자를 출력하는 방법은 다음과 같다.

```
let s1 = '"hello world"';
let s2 = "'hello world'";
let s3 = "\"hello world\"";
let s4 = '\'hello world\'';

console.log(s1);
console.log(s2);
console.log(s3);
console.log(s4);
```

노란색으로 칠한 부분의 따옴표 문자가 출력된다.

> "hello world"
> 'hello world'
> "hello world"
> 'hello world'

문자열 내부에 " 문자가 포함되어 있다면, 그 문자열은 ' 문자로 묶어야 한다.

`'"hello world"'`

문자열 내부에 ' 문자가 포함되어 있다면, 그 문자열은 " 문자로 묶어야 한다.

`"'hello world'"`

문자열 내부의 " 문자나, ' 문자 앞에 \ 문자를 붙였다면, (escape sequence)
그 문자열을 묶는 문자는 아무거나 선택할 수 있다.

문자열 내부 접근

```
let s = "hello world";

for (let i = 0; i < s.length; ++i)
    console.log(s[i]);
```

배열처럼 사용하면 된다.

```
"h"
"e"
"l"
"l"
"o"
" "
"w"
"o"
"r"
"l"
"d"
```

length 속성을 사용하여, 문자열의 길이를 알 수 있다.
[ ] 연산자를 사용하여 문자열의 문자를 읽을 수 있다.

immutable string

```
let s = "hello world";
console.log(s[0]);

s[0] = 'H';
console.log(s[0]);
```

```
"h"
"h"
```

javascript 문자열은 immutable 하다.
문자열이 생성된 후, 그 내용이 변경될 수 없다.

문자열 메소드들은 원본 문자열을 변경하는 것이 아니고, 새 문자열을 생성해서 리턴한다.

template literal

```
let first = 'Jane';
let last = 'Doe';

let s = `Hello ${first} ${last}!`
console.log(s);
```

` 따옴표를 사용한 문자열에 ${ 표현식 } 형태로 구현하여, 표현식의 값이 문자열에 삽입된다.

키보드 왼쪽 위의 ` 따옴표 문자를 사용해야 한다. (역 따옴표 문자)

string 클래스의 메소드

indexOf 메소드

문자열.indexOf(부분_문자열, 시작_위치)

문자열에서 **부분_문자열**을 찾아서, 찾은 위치(index)를 리턴한다.

시작_위치에서부터 뒤쪽으로(순방향으로) 찾기 시작한다.

시작_위치 파라미터가 생략된 경우 디폴트 값은 0 이다.
찾지 못할 경우 리턴값은 -1 이다.

```
let s = "one two one two";
console.log(s.indexOf("two"));
console.log(s.indexOf("two", 5));
console.log(s.indexOf("TWO"));
```

```
4
12
-1
```

search 메소드

문자열.search(/정규식/)

문자열에서 정규식을 찾아서, 찾은 위치(index)를 리턴한다.
javascript 언어에서 정규식은 /정규식/ 형태이다.

```
let s = "hello World";
console.log(s.search(/[A-Z]/));  // 6
```

정규식(regular expression)

정규식은, 패턴으로 문자열을 찾기 위한 문자열 패턴 표현식이다.

[A-Z] 정규식은 A부터 Z까지 문자 중 아무 한 문자를 의미한다.
즉 대문자 한 개와 일치하는 정규식이다.

slice 메소드

문자열.slice(시작_위치, 끝_위치)

문자열에서 **시작_위치**에서 **끝_위치** 사이의 부분 문자열을 리턴한다.

(시작_위치 <= 위치 < 끝_위치)

원본 문자열은 변하지 않는다. (immutable)

끝_위치 파라미터를 생략한 경우 디폴트 값은 문자열의 길이이다.

시작_위치, 끝_위치 값이 음수이면, 문자열 끝에서 역방향 인덱스이다.

문자열의 slice 메소드 사용법은, 배열의 slice 메소드 사용법과 같다.

```
let s = "abcdefgh";
console.log(s.slice(2, 4));  //cd
console.log(s.slice(2));     //cdefgh
console.log(s.slice(-3));    //fgh
```

substring 메소드

substring 메소드는 slice 메소드와 유사하다.
차이점은, substring 메소드는
음수 파라미터를 사용할 수 없다는 점이다.

```
let s = "abcdefgh";
console.log(s.substring(2, 4));          //cd
console.log(s.substring(2));             //cdefgh
console.log(s.substring(s.length - 3));  //fgh
```

substr 메소드

substr 메소드는 slice 메소드와 유사하다.
차이점은, substr 메소드의 두 번째 파라미터는
부분 문자열의 길이라는 점이다.

```
let s = "abcdefgh";
console.log(s.substr(2, 2));  //cd
console.log(s.substr(2));     //cdefgh
console.log(s.substr(-3));    //fgh
```

replace 메소드

문자열.replace(부분_문자열, 치환할_문자열)

replace 메소드는, 문자열에서 **부분_문자열**을 찾아서 **치환할_문자열**로 치환한다.
일치하는 부분 문자열이 여러 개 있어도, 첫 번째 부분 문자열만 치환된다.

부분_문자열에 정규식을 사용할 수 있다.

원본 문자열은 수정되지 않고, 치환된 새 문자열이 만들어져서 리턴된다.

```
let s = "hello world";
console.log(s.replace("o", "O"));  //hellO world
console.log(s);                    //hello world
```

원본 문자열, 즉 s 값은 변경되지 않았음에 주목하자. (immutable)
첫 번째 "o" 부분만 변경되었음에 주목하자.

```
let s = "hello world";
console.log(s.replace(/o/g, "O"));  //hellO wOrld
console.log(s);                     //hello world
```

"o" 부분이 전부 변경되었음에 주목하자.

/정규식/g
g는, 일치하는 부분 문자열들을 전부 치환하라는 옵션이다.

/정규식/i
i는, 문자열을 비교할 때, 대소문자를 무시하라는 옵션이다.

trim 메소드

문자열 앞뒤의 공백을 제거한 새 문자열을 리턴한다.
원본 문자열은 변경되지 않는다.

```
let s = "  hello world  ";

console.log(s.trim());
console.log(s);
```

```
"hello world"
"  hello world  "
```

replaceAll()

이 메소드는 간단하게 문자열에 포함된 모든 공백을 지울 수 있습니다.

```
const str = "  Hello  Wo  rld!  ";
const trimStr = str.replaceAll(' ', '');

console.log(trimStr);
```

```
"HelloWorld!"
```

split 메소드

문자열.split(구분_문자열)

문자열을 구분_문자열을 기준으로 잘라서, 배열을 만들어서 리턴한다.
원본 문자열은 변경되지 않는다.

```
let s = "one two three four";
let a = s.split(" ");

for (let i = 0; i < a.length; ++i)
  console.log(a[i]);
```

```
"one"
"two"
"three"
"four"
```

구분_문자열에 정규식을 사용할 수 있다.

```
let s = "one, two,three  four";
let a = s.split(/[\t ,]+/);

for (let i = 0; i < a.length; ++i)
  console.log(a[i]);
```

```
"one"
"two"
"three"
"four"
```

/[\t ,]+/
탭, 공백, 콤마 문자들의 결합과 일치하는 정규식

```
let s = "hello";
let a = s.split("");

for (let i = 0; i < a.length; ++i)
  console.log(a[i]);
```

```
"h"
"e"
"l"
"l"
"o"
```

구분_문자열이 빈문자열("")이면, 문자 한개씩 쪼개져 만들어진 배열을 리턴한다.

match 메소드

```
let a = "list.jsp?pg=324&sz=15".match(/pg=[0-9]+/)

console.log(a[0]);    // 정규식에 일치하는 문자열
console.log(a.index); // 정규식에 일치하는 문자열의 위치 index
```

```
"pg=324"
9
```

```
let a = "list.jsp?pg=324&sz=15".match(/pg=([0-9]+)/)

console.log(a[0]);    // 정규식에 일치하는 문자열
```

```
console.log(a.index); // 일치하는 문자열의 위치 index
console.log(a[1]);    // 첫번째 괄호에 일치하는 부분
```

```
"pg=324"
9
"324"
```

```
let a = "list.jsp?pg=324&sz=15".match(/pg=([0-9]+)&sz=([0-9]+)/)

console.log(a[0])  // 매칭된 문자열 전체
console.log(a[1])  // 첫째 괄호에 매칭된 부분
console.log(a[2])  // 둘째 괄호에 매칭된 부분
console.log(a.index) // 매칭된 문자열의 위치 index
```

```
"pg=324&sz=15"
"324"
"15"
9
```