* 예습
5/20

32. String

TABLE OF CONTENTS

표준 빌트인 객체(standard built-in object)인 String은 원시 타입인 문자열을 다룰 때 유용한 프로퍼티와 메소드를 제공한다.

#1. String 생성자 함수

표준 빌트인 객체인 String 객체는 생성자 함수 객체이다. 따라서 new 연산자와 함께 호출하여 String 인스턴스를 생성할 수 있다.

String 생성자 함수에 인수를 전달하지 않고 new 연산자와 함께 호출하면 [[StringData]] 내부 슬롯에 빈 문자열을 할당한 String 래퍼 객체를 생성한다.

JAVASCRIPT

```
const strObj = new String();
console.log(strObj); //
```

> 위 예제를 크롬 브라우저의 개발자 도구에서 실행해보면 [[PrimitiveValue]]라는 프로퍼티가 보인다. 이는 [[StringData]] 내부 슬롯을 가리킨다. ES5에서는 [[StringData]]을 [[PrimitiveValue]]이라 불렸다.

String 생성자 함수에 문자열을 인수로 전달하면 [[StringData]] 내부 슬롯에 인수로 전달받은 문자열을 할당한 String 래퍼 객체를 생성한다.

JAVASCRIPT

```
const strObj = new String('Lee');
console.log(strObj);
//
```

"11.1.2. 문자열과 불변성"에서 살펴보았듯이 문자열은 유사 배열 객체이면서 이터러블이다. 따라서 배열과 유사하게 인덱스를 사용하여 각 문자에 접근할 수 있다.

JAVASCRIPT

```
console.log(strObj[0]); //
```

String 생성자 함수에 문자열이 아닌 값을 인수로 전달하면 전달받은 인수를 문자열로 강제 변환한 후, [[StringData]] 내부 슬롯에 변환한 문자열을 할당한 String 래퍼 객체를 생성한다.

JAVASCRIPT

```
let strObj = new String(123);
console.log(strObj);
//

strObj = new String(null);
```

```
console.log(strObj);
//
```

"9.3. 명시적 타입 변환"에서 살펴보았듯이 new 연산자를 사용하지 않고 String 생성자 함수를 호출하면 String 인스턴스가 아닌 문자열을 반환한다. 이를 이용하여 명시적으로 타입을 변환하기도 한다.

JAVASCRIPT

```
// 숫자 타입 => 문자열 타입
String(1);        // →
String(NaN);      // →
String(Infinity); // →

// 불리언 타입 => 문자열 타입
String(true);  // → "true"
String(false); // → "false"
```

2. length 프로퍼티

length 프로퍼티는 문자열의 문자 개수를 반환한다.

JAVASCRIPT

```
'Hello'.length;     // → 5
'안녕하세요!'.length; // → 6
```

String 레퍼 객체는 배열과 마찬가지로 length 프로퍼티를 갖는다. 그리고 인덱스를 나타내는 숫자를 프로퍼티 키로, 각 문자를 프로퍼티 값으로 가지므로 String 레퍼 객체는 유사 배열 객체이다.

3. String 메소드

String 객체의 모든 메소드는 언제나 새로운 문자열을 반환한다. 문자열은 변경 불가능(immutable)한 원시 값이기 때문이다. 사용 빈도가 높은 메소드에 대해 살펴보도록 하자.

3.1. String.prototype.indexOf

indexOf 메소드는 문자열에서 인수로 전달한 문자열을 검색하여 첫번째 인덱스를 반환한다. 검색에 실패하면 -1을 반환한다.

JAVASCRIPT

```
const str = 'Hello World';

// 문자열 str에서 'l'을 검색하여 첫번째 인덱스를 반환한다.
str.indexOf('l'); // → 2


// 문자열 str에서 'or'을 검색하여 첫번째 인덱스를 반환한다.
str.indexOf('or'); // →

// 문자열 str에서 'x'를 검색하여 첫번째 인덱스를 반환한다.
// 검색에 실패하면 -1을 반환한다.
str.indexOf('x'); // → -1
```

indexOf 메소드의 2번째 인수로 검색을 시작할 인덱스를 전달할 수 있다.

JAVASCRIPT

```
// 문자열 str의 인덱스 3부터 'l'을 검색하여 첫번째 인덱스를 반환한다.
str.indexOf('l', 3); // → 3
```

indexOf 메소드는 문자열에 특정 문자열이 존재하는지 확인할 때 유용하다.

JAVASCRIPT

```
if (str.indexOf('Hello') !== -1) {
  // 문자열 str에 'Hello'가 포함되어 있는 경우에 처리할 내용
}
```

ES6에서 새롭게 도입된 를 사용하면 보다 가독성이 좋다.

JAVASCRIPT

```
if (str.includes('Hello')) {
  // 문자열 str에 'Hello'가 포함되어 있는 경우에 처리할 내용
```

```
}
```

3.2. String.prototype.includes

ES6에서 새롭게 도입된 includes 메소드는 문자열에 인수로 전달한 문자열이 포함되어 있는지 확인하여 그 결과를 true 또는 false로 반환한다.

JAVASCRIPT

```
const str = 'Hello world';

str.includes('Hello'); // → true
str.includes('');      // → true
str.includes('x');     // → false
str.includes();        //
```

includes 메소드의 2번째 인수로 검색을 시작할 인덱스를 전달할 수 있다.

JAVASCRIPT

```
const str = 'Hello world';

// 문자열 str의 인덱스 3부터 'l'이 포함되어 있는지 확인
str.includes('l', 3); // → true
str.includes('H', 3); // → false
```

3.3. String.prototype.startsWith

ES6에서 새롭게 도입된 startsWith 메소드는 문자열이 인수로 전달한 문자열로 시작되는지 확인하여 그 결과를 true 또는 false로 반환한다.

JAVASCRIPT

```
const str = 'Hello world';

// 문자열 str이 'He'로 시작하는지 확인
str.startsWith('He'); // → true
```

```
// 문자열 str이 'x'로 시작하는지 확인
str.startsWith('x'); // → false
```

startsWith 메소드의 2번째 인수로 검색을 시작할 인덱스를 전달할 수 있다.

JAVASCRIPT

```
// 문자열 str의 인덱스 5부터 시작하는 문자열이 ' '로 시작하는지 확인
str.startsWith(' ', 5); // → true
```

3.4. String.prototype.endsWith

ES6에서 새롭게 도입된 endsWith 메소드는 메소드는 문자열이 인수로 전달한 문자열로 끝나는지 확인하여 그 결과를 true 또는 false로 반환한다.

JAVASCRIPT

```
const str = 'Hello world';

// 문자열 str이 'ld'로 끝나는지 확인
str.endsWith('ld'); // → true
// 문자열 str이 'x'로 끝나는지 확인
str.endsWith('x'); // → false
```

endsWith 메소드의 2번째 인수로 검색할 를 전달할 수 있다.

JAVASCRIPT

```

str.endsWith('lo', 5); //
```

3.5. String.prototype.charAt

charAt 메소드는 인수로 전달한 인덱스에 위치한 문자를 반환한다.

JAVASCRIPT

```
const str = 'Hello';

for (let i = 0; i < str.length; i++) {
  console.log(str.charAt(i)); //
}
```

인덱스는 문자열의 범위, 즉 0 ~ (문자열 길이 - 1) 사이의 정수이어야 한다. 인덱스가 문자열의 범위를 벗어난 정수인 경우, 빈 문자열을 반환한다.

JAVASCRIPT

```
// 인덱스가 문자열의 범위(0 ~ str.length-1)를 벗어난 경우 빈문자열을 반환한다.
str.charAt(5); // → ''
```

charAt 메소드와 유사한 문자열 메소드는 String.prototype.charCodeAt과 String.prototype.codePointsAt이 있다.

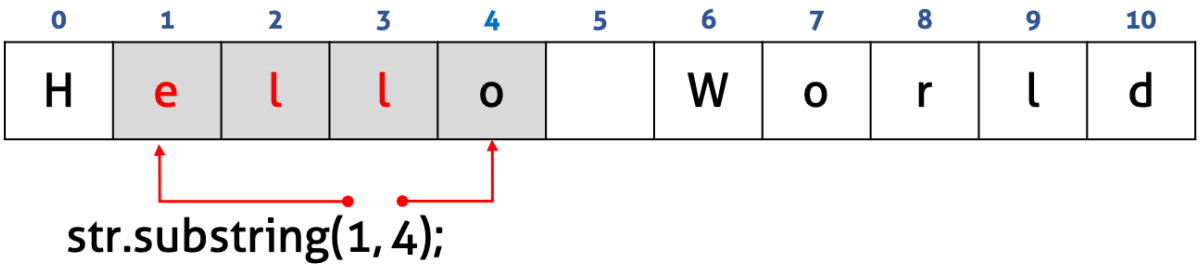
3.6. String.prototype.substring

substring 메소드는 첫번째 인수로 전달한 인덱스에 위치하는 문자부터 두번째 인수로 전달한 인덱스에 위치하는 문자의 바로 이전 문자까지 문자열의 부분 문자열을 반환한다.

JAVASCRIPT

```
const str = 'Hello World';

// 인덱스 1부터 인덱스 4 이전까지의 부분 문자열을 반환한다.
str.substring(1, 4); // →
```



String.prototype.substring

substring 메소드의 두번째 인수는 생략할 수 있다. 이때 첫번째 인수로 전달한 인덱스에 위치하는 문자부터 마지막 문자까지 부분 문자열을 반환한다.

JAVASCRIPT

```
const str = 'Hello World';

// 인덱스 1부터 마지막 문자까지 부분 문자열을 반환한다.
str.substring(1); // →
```

substring 메소드의 첫번째 인수는 두번째 인수보다 큰 정수이어야 정상이다. 하지만 아래와 같이 인수를 전달하여도 정상 동작한다.

- 첫번째 인수 > 두번째 인수인 경우, 두 인수는 교환된다.
- 인수 < 0 또는 NaN인 경우, 0으로 취급된다.
- 인수 > 문자열의 길이(str.length)인 경우, 인수는 문자열의 길이(str.length)으로 취급된다.

JAVASCRIPT

```
const str = 'Hello World'; // str.length == 11

// 첫번째 인수 > 두번째 인수인 경우, 두 인수는 교환된다.
str.substring(4, 1); // →

// 인수 < 0 또는 NaN인 경우, 0으로 취급된다.
str.substring(-2); // →

// 인수 > 문자열의 길이(str.length)인 경우, 인수는 문자열의 길이(str.length)으로 취급된다.
str.substring(1, 100); //
str.substring(20); // →
```

Stirng.prototype.indexOf 메소드와 함께 사용하면 특정 문자열을 기준으로 앞뒤에 위치한 부분 문자열을 취득할 수 있다.

JAVASCRIPT

```
const str = 'Hello World';

// 스페이스를 기준으로 앞에 있는 부분 문자열 취득
str.substring(0, str.indexOf(' ')); // →
// 스페이스를 기준으로 뒤에 있는 부분 문자열 취득
str.substring(str.indexOf(' ') + 1, str.length); // →
```

3.7. String.prototype.slice

slice 메소드는 substring 메소드와 동일하게 동작한다. 단, slice 메소드에는 음수인 인수를 전달할 수 있다. 음수인 인수를 전달하면 뒤에서부터 시작하여 문자열을 잘라내어 반환한다.

JAVASCRIPT

```
const str = 'hello world';

// substring과 slice 메소드는 동일하게 동작한다.
// 0번째부터 5번째 이전 문자까지 잘라내어 반환
str.substring(0, 5); // → 'hello'
str.slice(0, 5); // → 'hello'

// 인덱스가 2인 문자부터 마지막 문자까지 잘라내어 반환
str.substring(2); // → 'llo world'
str.slice(2); // → 'llo world'

// slice 메소드는 음수인 인수를 전달할 수 있다.
str.substring(-5); // →
// 뒤에서 5자리를 잘라내어 반환한다.
str.slice(-5); // →
```

3.8. String.prototype.toUpperCase

toUpperCase 메소드는 문자열의 모든 문자를 대문자로 변경하여 반환한다.

JAVASCRIPT

```
const str = 'Hello World!';

str.toUpperCase(); // → 'HELLO WORLD!'
```

3.9. String.prototype.toLowerCase

toLowerCase 메소드는 문자열의 모든 문자를 대문자로 변경하여 반환한다.

JAVASCRIPT

```
const str = 'Hello World!';

str.toLowerCase(); // → 'hello world!'
```

3.10. String.prototype.trim

trim 메소드는 문자열 앞뒤에 공백 문자가 있을 경우, 이를 제거한 문자열을 반환한다.

JAVASCRIPT

```
const str = '   foo  ';

str.trim(); // → 'foo'
```

현재 제안 stage 3에 있는 String.prototype.trimStart, String.prototype.trimEnd를 사용하면 문자열 앞 또는 뒤에 공백 문자가 있을 경우, 이를 제거한 문자열을 반환한다.

JAVASCRIPT

```
const str = '   foo  ';

// String.prototype.{trimStart,trimEnd} : Proposal stage 3
str.trimStart(); // → 'foo  '
str.trimEnd();   // → '   foo'
```

String.prototype.replace 메소드에 정규 표현식을 인수로 전달하여 공백 문자를 제거할 수도 있다.

JAVASCRIPT

```
const str = '   foo  ';

// String.prototype.replace
str.replace(/\s/g, '');   // → 'foo'
str.replace(/^\s+/g, ''); // → 'foo  '
str.replace(/\s+$/g, ''); // → '   foo'
```

String.prototype.replace 메소드는 “32.3.12. String.prototype.replace”에서 살펴보도록 하자.

3.11. String.prototype.repeat

ES6에서 새롭게 도입된 repeat 메소드는 인수로 전달한 정수만큼 반복해 연결한 새로운 문자열을 반환한다. 인수로 전달한 정수가 0이면 빈 문자열을 반환하고 음수이면 RangeError를 발생시킨다.

JAVASCRIPT

```
const str = 'abc';

str.repeat(0);   // →
str.repeat(1);   // →
str.repeat(2);   // →
str.repeat(2.5); // →
str.repeat(-1);  // →
```

3.12. String.prototype.replace

replace 메소드는 첫번째 인수로 전달한 문자열 또는 정규표현식을 대상 문자열에서 검색하여 두번째 인수로 전달한 문자열로 치환하여 결과가 반영된 새로운 문자열을 반환한다.

JAVASCRIPT

```
const str = 'Hello world';

// str에서 첫번째 인수 'world'를 검색하여 두번째 인수 'Lee'로 치환한다.
str.replace('world', 'Lee'); // → 'Hello Lee'
```

검색된 문자열이 여럿 존재할 경우 첫번째로 검색된 문자열만 치환한다.

JAVASCRIPT

```
const str = 'Hello world world';

str.replace('world', 'Lee'); // → 'Hello Lee world'
```

특수한 교체 패턴을 사용할 수 있다. 예를 들어, $&는 검색된 문자열을 의미한다.

JAVASCRIPT

```
const str = 'Hello world';

// 특수한 교체 패턴을 사용할 수 있다. ($& => 검색된 문자열)
str.replace('world', '<strong>$&</strong>');
```

교체 패턴에 대한 자세한 내용은 https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String/replace 을 참고하기 바란다.

replace 메소드의 첫번째 인수로 정규 표현식을 전달할 수도 있다.

JAVASCRIPT

```
const str = 'Hello Hello';

// /hello/gi은 정규 표현식이다.
// 'hello'를 대소문자를 구별하지 않고 문자열 내의 모든 패턴을 검색한다.
str.replace(/hello/gi, 'Lee'); // → 'Lee Lee'
```

replace 메소드의 두번째 인수로 치환 함수를 전달할 수 있다. 예를 들어 카멜 케이스를 스네이크 케이스로, 스네이크 케이스를 카멜 케이스로 변경하는 함수를 replace 메소드로 구현할 수 있다.

JAVASCRIPT

```
// camelCase ⇒ snake_case
function camelToSnake(camelCase) {
  // /.[A-Z]/g ⇒ 문자와 대문자로 이루어진 문자열 검색
  // 두번째 인수로 치환 함수를 전달할 수 있다.
  return camelCase.replace(/.[A-Z]/g, match ⇒ {
    console.log(match); // 'oW'
    return match[0] + '_' + match[1].toLowerCase();
  });
}

const camelCase = 'helloWorld';
camelToSnake(camelCase); // → 'hello_world'

// snake_case ⇒ camelCase
function snakeToCamel(snakeCase) {
  // /_[a-z]/g ⇒ _와 소문자로 이루어진 문자열 검색
  // 두번째 인수로 치환 함수를 전달할 수 있다.
  return snakeCase.replace(/_[a-z]]/g, match ⇒ {
```

```
    console.log(match); // '_w'
    return match[1].toUpperCase();
  }); // helloWorld
}

const snakeCase = 'hello_world';
snakeToCamel(snakeCase); // → 'helloWorld'
```

3.13. String.prototype.split

첫번째 인수로 전달한 문자열 또는 정규표현식을 대상 문자열에서 검색하여 문자열을 구분한 후 분리된 각 문자열로 이루어진 배열을 반환한다. 원본 문자열은 변경되지 않는다.

인수가 없는 경우, 대상 문자열 전체를 단일 요소로 하는 배열을 반환한다.

JAVASCRIPT

```
/**
 * @param {string | RegExp} [separator] - 구분 대상 문자열 또는 정규표현식
 * @param {number} [limit] - 구분 대상수의 한계를 나타내는 정수
 * @return {string[]}
 */
str.split([separator[, limit]])
```

JAVASCRIPT

```
const str = 'How are you doing?';

// 공백으로 구분(단어로 구분)하여 배열로 반환한다
str.split(' '); // → ["How", "are", "you", "doing?"]

// 정규 표현식
str.split(/\s/); // → ["How", "are", "you", "doing"]

str.split(); // →

str.split(''); // →

// 공백으로 구분하여 배열로 반환한다. 단 요소수는 3개까지만 허용한다
```

```
str.split(' ', 3); // →

// 'o'으로 구분하여 배열로 반환한다.
str.split('o'); // →
```