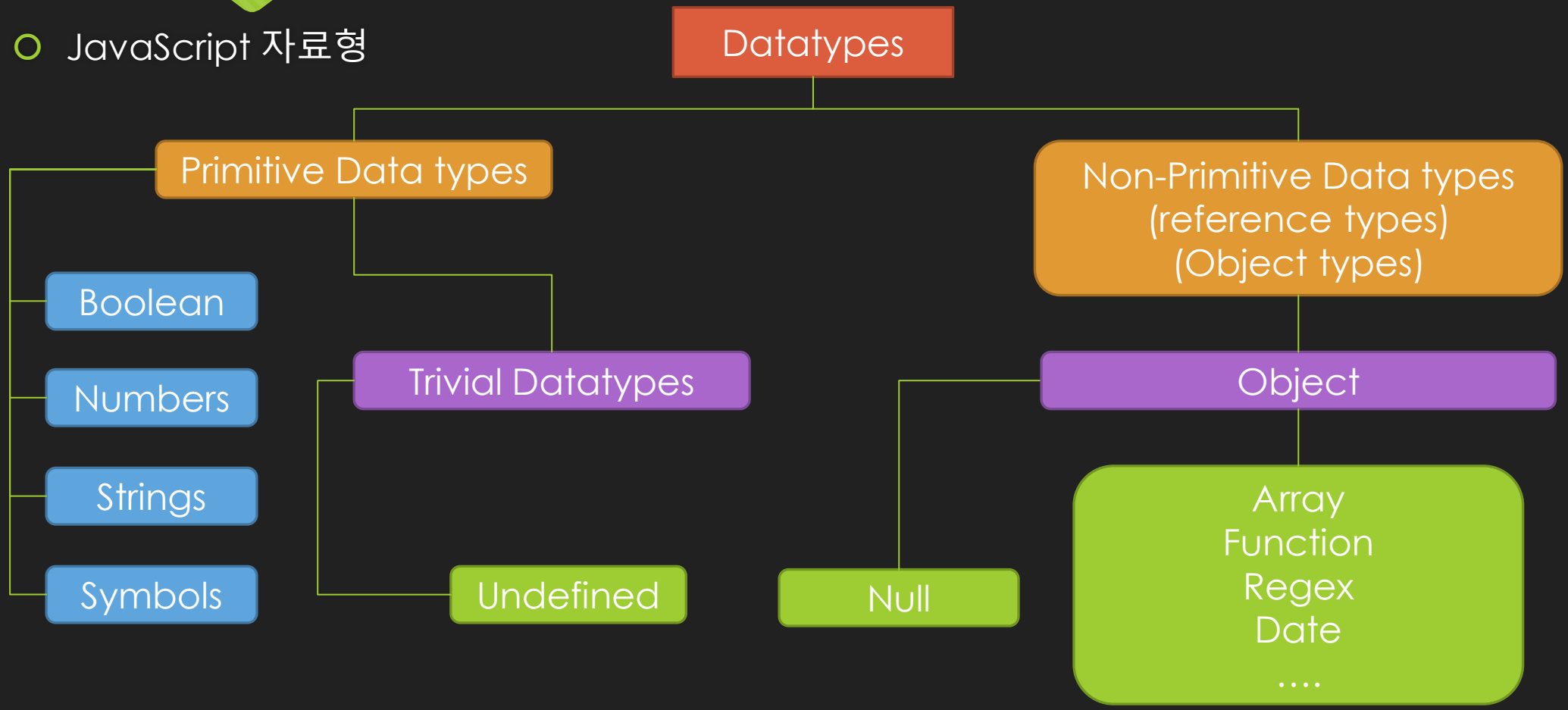
JavaScript

변수와 자료형 – 김근형 강사

자료형

- JavaScript 자료형



자료형

- JavaScript 자료형
  - primitive Types : 자바스크립트에서 기본적으로 주어지는 기본형 타입. 총 6가지를 가리키며 String, Numbers, Boolean, Symbols(ECMA 6 버전에서 추가됨), 과 Null, Undefined 타입이 존재한다.
  - Null과 Undefined는 그 외의 타입으로 따로 분류되기도 한다.
  - Non-primitive Types : 자바스크립트에서 사용자에 의해 임의로 지정된 타입을 이야기 하며 보통은 주소를 갖고있다 하여 Reference Types라 부르기도 하고 Object 객체를 프로토타입으로 가지고 있다고 하여 Object Type이라 부르기도 한다.
  - 여기서 정의되는 모든 타입들은 최 상위 프로토타입으로 Object를 가지고 있다.

Numbers

- Numbers
  - 숫자를 표현하거나 산술 연산을 하는데 사용되는 데이터 타입
  - 기본적으로 +, -, *, / 등의 산술연산이 가능하며 Math 라는 내장객체를 이용하여 수학함수를 이용한 결과를 얻을 수도 있다.
  - 명세에 따르면 자바스크립트의 Number는 "64비트 형식 IEEE 754 값" 으로 정의 된다($-(2^{53}-1)$ 와 $2^{53}-1$ 사이의 숫자값).
  - 정수만을 표현하기 위한 특별한 자료형은 없다.
  - 또한 8진수나 16진수 형태의 숫자도 표현이 가능하다.

Numbers

- Numbers
  - 기본적으로 자바 스크립트는 정수형의 숫자를 표현할 수 있다.

```
> console.log(1);
  console.log(0);
  console.log(999999999);
  console.log(-12345);
1
0
999999999
-12345
```

  - 또한 실수 형태의 숫자도 표현이 가능하다.

```
> console.log(1.23456);
  console.log(-0.01252);
1.23456
-0.01252
```

Numbers

- Numbers
  - 16진수 표현법 : 숫자 앞에 0x 를 붙여서 해당 숫자가 16진수 임을 나타낼 수 있다.
  - 8진수 표현법 : 숫자 앞에 0을 붙여서 해당 숫자가 8진수 임을 나타낼 수 있다.

```
> console.log(0xff);  // 16 진수 표현법
  console.log(010);  // 8 진수 표현법
255
8
```

Numbers

- 지수 표기법
  - 너무 큰 숫자는 지수 표기법을 사용하여 표현할 수 있다.
  - 이때 e 는 대소문자를 가리지 않는다.
  - e 앞의 숫자를 뒤에 있는 숫자 만큼 10으로 곱한다.
  - 앞의 숫자를 소수로 작성 할 수도 있다.
  - 소수는 e 뒤에 -를 붙여서 표현할 수 있다.

```
> console.log(5e5);  // 지수 표기법
  console.log(5E-5);  // 지수 표기법
500000
0.00005
```

Numbers

- 무한대(Infinity)
  - 자바 스크립트에서는 무한대 값을 표현할 수 있다.
  - 양수 무한대일 경우 Infinity, 음수 무한대일 경우 –Infinity로 출력이 된다.
  - Infinity 값에는 어떤 값을 연산하여도 값은 바뀌지 않는다.
  - Infinity 값을 검사하는 함수로는 isFinite() 라는 함수가 존재한다.

```
> console.log(1/0);
  console.log(-1/0);
  console.log(isFinite(1/0));      // false
  console.log(isFinite(-1/0));     // false
  console.log(isFinite("a"));      // false
  console.log(isFinite(NaN));      // false
  console.log(isFinite(1));        // true
  console.log(Infinity == Infinity);      // true
  console.log(Infinity == Infinity+10);   // true
  console.log(Infinity == Infinity-10);   // true
Infinity
-Infinity
false
false
false
false
true
true
true
true
```

Numbers

- NaN(Nat a Number)
  - 수학 연산을 수행할 수 없을 경우(예: 0을 0으로 나누는) NaN(not-a-number)이 반환된다.
  - NaN은 자기 자신을 포함해 다른 어떤 숫자와도 같은지 비교할 수 없다
  - 어떤 값이 NaN인지 검사할 때는 isNaN() 함수를 사용한다.

```
> console.log(0/0);              // NaN
  console.log(isNaN(0/0));       // true
  console.log(isNaN(NaN));       // true
  console.log(isNaN("a"));       // true
  console.log(isNaN(123));       // false
  console.log(NaN == NaN);       // false
  console.log(NaN == 1);         // false
NaN
true
true
true
false
false
false
```

String

- 문자형
  - 문자열 자료형은 자바스크립트 내의 텍스트를 나타낸다
  - 문자열은 유니코드 문자의 나열로서 모든 문자열은 내부적으로 유니코드로 구성돼 있다.
  - 보통 자바스크립트 문자열은 문자열 리터럴을 이용해 생성하는데, 작은따옴표(')나 큰따옴표(") 안에 텍스트가 들어 있으면 문자열이 될 수 있다.

```
> console.log("Hello world.");
  console.log('12345');
  console.log("It's great!");
Hello world.
12345
It's great!
```

String

- 문자형
  - String 전역 객체를 직접적으로 이용해 문자열을 생성할 수 있다.

```
> // 원시 문자열
  var str1 = String("12345");

  // String 객체
  var str2 = new String("Hello world.");

  // valueOf 메서드를 이용해 String 객체를 그에 상응하는 원시 문자열로 변환
  var str3 = str2.valueOf();

  console.log(typeof str1);   // string
  console.log(typeof str2);   // Object
  console.log(typeof str3);   // string
string                                                    VM48:10
object                                                    VM48:11
string                                                    VM48:12
```

String

- Escape 문자
  - 자바스크립트 문자열 안에서 특수 문자를 사용하려면 백슬래시()를 문자 앞에 둘 필요가 있는데, 이를 이스케이프(escape)라고 한다.

```
console.log("first line \n second line \n third line");   // 개행
console.log("\f");                  // 폼 피드
console.log("\r");                  // 캐리지 리턴
console.log("\t");                  // 탭
console.log("\v");                  // 수직 탭
console.log("\'");                  // 작은따옴표
console.log("\"");                  // 큰따옴표
console.log("c:\\temp");            // 백슬래시 문자(\)
console.log("\b");                  // 백스페이스
// 유니코드 문자는 4자리 16진수값으로 지정한다.
// 예를 들어, \u00A9는 저작권 기호를 나타내는 유니코드 문자다.
console.log("\u00A9");
```

String

- Escape 문자

```
<script type="text/javascript">
    console.log("첫 번째 줄, \
     두 번째 줄, \
      세 번째 줄");
    console.log("첫 번째 줄, \n\
     두 번째 줄, \n\
      세 번째 줄");
</script>
```

```
첫 번째 줄,         두 번째 줄,         세 번째 줄
첫 번째 줄,
        두 번째 줄,
          세 번째 줄
```

String

- String.length
  - 문자열 길이를 출력하는 속성
  - 유니코드 문자를 기준으로 문자열의 길이를 출력한다.

```
>       // 개행 이스케이프는 한 문자에 해당
        console.log("hello\n".length);   // 6
        // 세 개의 백슬래시
        console.log("\\\\\\".length);   // 3
        // 두 개의 유니코드 문자
        console.log("\u00A9\u00A9".length);   // 2
6
3
2
```

Boolean

- Boolean
  - 논리적인 요소를 나타내고자 할 경우 쓰이는 타입
  - 값은 true 아니면 false
  - 실제 Boolean 타입은 비교/논리 연산의 결과 값으로 자바 스크립트 제어 구조에 많이 사용한다.

```
> console.log(true)
  true
< undefined
> console.log(false)
  false
< undefined
```

Boolean

- Boolean 연산
  - 불린 값이 숫자 연산에 사용되면 참은 숫자 1로, 거짓은 숫자 0으로 변환된다.
  - 불린 값이 문자열 연산에 사용되면 참은 "true"라는 문자열로, 거짓은 "false"라는 문자열로 변환된다.
  - 불린 값이 예상되는 곳에 숫자가 사용되면 해당 숫자가 거짓으로 변환되는 0이나 NaN이 아닌 이상 참으로 변환된다.
  - 불린 값이 예상되는 곳에 문자열이 사용되면 거짓으로 변환되는 빈 문자열인 경우를 제외하고 해당 문자열은 참으로 변환된다.
  - *null*과 *undefined* 값은 거짓으로 변환되며 널이 아닌 객체나 배열, 함수는 참으로 변환된다.

Boolean

- Boolean 연산

```
console.log(0 + true);  // 1
console.log(0 + false);  // 0
console.log("" + true); // "true"
console.log("" + false); // "false"
console.log(!!(1));     // true
console.log(!!(-1));     // true
console.log(!!(0));     // false
console.log(!!(NaN));     // false
console.log(!!"abc");     // true
console.log(!!"false");     // true
console.log(!!"");     // false
console.log(!!null);     // false
console.log(!!undefined);     // false
console.log(!![]);     // true
console.log(!!{});     // true
```

```
1
0
true
false
true
true
false
false
true
true
false
false
false
true
true
```

Null, Undefined

- Undefined
  - 기본적으로 값이 할당되지 않은 변수는 undefined 타입이며, undefined 타입은 변수 자체의 값 또한 undefined 이다.
  - 정의되지 않은 것, 초기화되어 있지 않거나 존재하지 않는 객체의 프로퍼티 및 존재하지 않는 배열의 원소 값에 접근하려고 할 때 얻어지는 변수의 값이다.
- Null
  - 명시적으로 값이 비어 있음을 나타내는데 사용
  - "아무것도 참조하고 있지 않다"라는 의미가 담겨 있으며 주로 객체를 담을 변수를 초기화할 때 많이 사용한다.
  - 하지만 null 역시 undefined와 마찬가지로 값 이며 데이터 타입이다.
  - 분명한 차이점은 undefined는 변수를 선언만 하더라도 할당되지만 null은 변수를 선언한 후에 null로 값을 바꾼다.
- 둘의 가장 큰 차이점은 null은 object 타입이고 undefined는 undefined 타입이다.

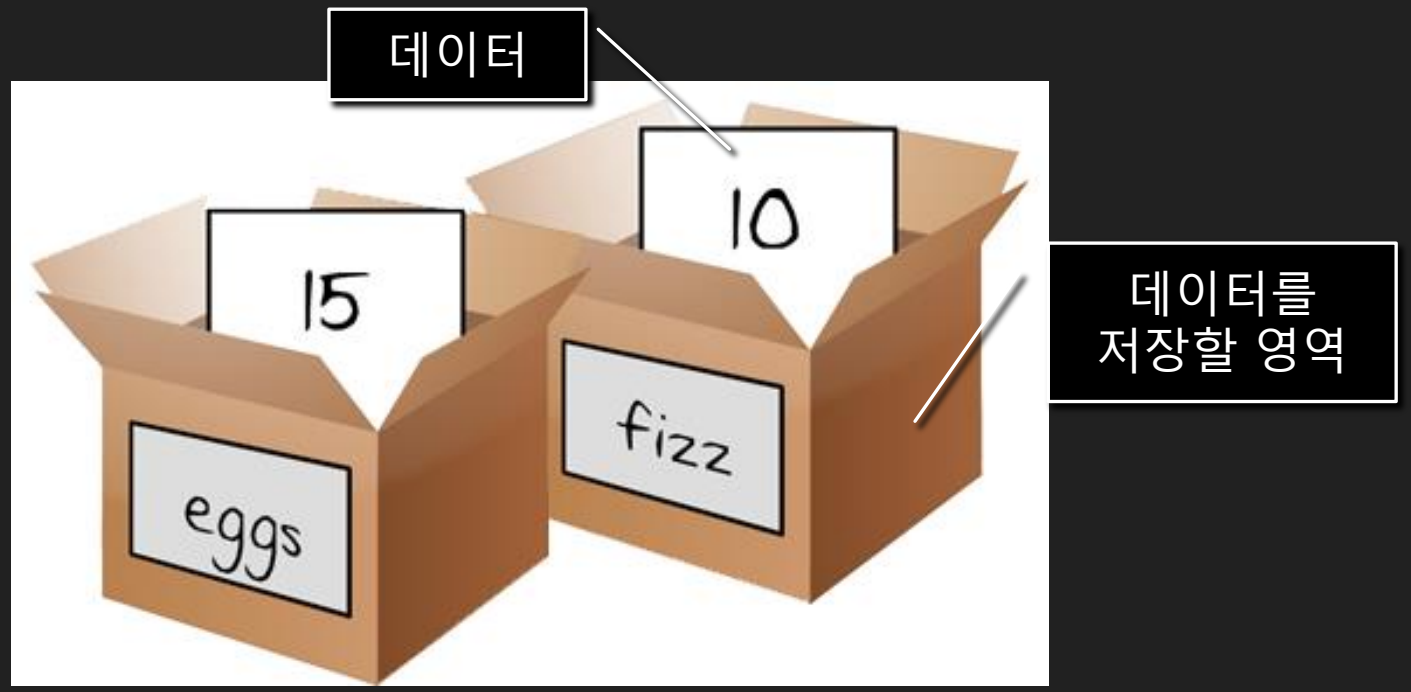
Null, Undefined

- Null, Undefined 예제

```
console.log(typeof null)
object
undefined
console.log(typeof undefined)
undefined
undefined
```

변수

- 변수



변수

- 변수
  - "데이터를 담아두는 그릇“
  - 변수는 반드시 한 개의 값을 담을 수 있으며 다른 값이 삽입될 경우 해당 변수에 들어있던 이전 값은 소멸한다.
  - 변수는 기본형 타입 외에 참조형 타입의 데이터들을 담을 수 있다.
  - 변수의 타입에는 var, let, const가 존재한다.
  - 선언방법은 아래와 같다.

var/let/const 변수이름 = **데이터** ;

세미콜론 :
문장의 끝

변수

- 변수 명 작성 시 주의사항
  - 숫자로 시작할 수 없음 => var 1na (x)
  - 변수 명 첫글자로는 $,_, 영문자만 가능 => var ^num=10 (x)
  - 대소문자 구분을 해야 한다 => var name ≠ var Name
  - 변수는 대문자가 아닌 소문자 명사형으로 시작할 것 => var Name (x) var name(o)
  - 변하지 않는 환경변수의 값을 담는 상수 변수는 모두 대문자로 표기 => var ADMIN_ID
  - 여러 단어가 조합되는 경우 낙타 표기법(camelcase)으로 작성한다. => var userName
  - 이미 정의된 예약어는 사용 불가. => var break (x) var for(x)

변수

- var
  - ECMA2015버전 이전에 기본적으로 사용되는 자바 스크립트 변수
  - 기본형 뿐만이 아니라 참조형에 대한 값을 넣을 수 있다.
  - 가장 초창기에 사용되었던 변수인 만큼 가장 많이 사용되었던 변수형으로써 현재는 let과 const로 많이 대체 되고 있다.

```
> var a = 1;
  var b = "hello";
  var c = true;
  console.log(a);
  console.log(b);
  console.log(c);
1
hello
true
```

```
> var a = 1;
  console.log(a);
  var a = 2;  // 같은 이름으로 재 선언 되면 위의 값을 덮어씌운다.
  console.log(a);
  a = 3;  // 기존의 a 변수에 값을 넣어 관리한다.
  console.log(a);
1
2
3
```

변수

- let
  - ECMA2015 버전에서 생겨난 변수형
  - var와 비슷한 기능을 가지며 최근 let이라는 변수형을 더 많이 사용하고 있다.

```
let a = 1;
let b = "hello";
let c = true;
console.log(a);
console.log(b);
console.log(c);
```

```
1
hello
true
```

```
let a = 1;
console.log(a);
//let a = 2;    // 같은 이름으로 재 선언 불가
console.log(a);
a = 3;  // 기존의 a 변수에 값을 넣어 관리한다.
console.log(a);
```

변수

- const
  - ECMA2015 버전에서 생겨난 변수형
  - 상수형이라고 부르며 한번 값이 선언되면 다른 값으로 변환되지 않는다.

```
const a = 1;
const b = "hello";
const c = true;
console.log(a);
console.log(b);
console.log(c);
```

```
const a = 1;
console.log(a);
//let a = 2;    // 같은 이름으로 재 선언 불가
console.log(a);
// a = 3;    // 상수형은 기존의 값을 다른 값으로 대체 불가.
console.log(a);
```

변수

- undefined와 변수의 값 선언
  - 값 없이 선언되는 let 혹은 var 변수의 경우 undefind의 값을 가진다.
  - const는 무조건 선언될 때 값을 넣어야 하기 때문에 값 없이 선언하는 것은 불가능하다.

```
let a;
console.log(a);
a = 3;
console.log(a);
//const b; // 값 없이 선언 불가능
const b = 4;
console.log(b);
```

변수

- 변수 명명 시 유의사항
  - 숫자로 시작할 수 없음 => var 1na (x)
  - 변수 명 첫글자로는 $,_, 영문자만 가능 => var ^num=10 (x)
  - 대소문자 구분을 해야 한다 => var name ≠ var Name
  - 변수는 대문자가 아닌 소문자 명사형으로 시작할 것 => var Name (x) var name(o)
  - 변하지 않는 환경변수의 값을 담는 상수 변수는 모두 대문자로 표기 => var ADMIN_ID
  - 여러 단어가 조합되는 경우 낙타 표기법(camelcase)으로 작성한다. => var userName
  - 이미 정의된 예약어는 사용 불가. => var break (x) var for(x)

Template literals

- 템플릿 리터럴(Template literals)
  - ECMAScript에서 새로운 문자열 선언 방식이 등장한다.
  - BackTick(`) 이라는 개념이 등장하면서 실제 문자열을 이전보다 자유롭게 선언할 수 있는 방법이 등장
  - 그 중 하나가 템플릿 리터럴이며 템플릿 리터럴은 기존에 결합 연산자를 통해 문자열과 변수를 더했던 방식과는 달리 '${변수명}' 을 문자열 안에 입력하여 변수를 문자열 안에 삽입하는 방식이다.
  - 이런 기능을 통해 기존 문자열에 불필요한 연산자를 제거할 수 있고 좀 더 직관적으로 문자열에 삽입된 변수를 볼 수 있다는 장점이 생겼다.

Template literals

- Template
  - 템플릿 리터럴(Template Literal)은 문자열 처리를 위한 템플릿을 제공한다.

```
[구문]
`문자열`
`문자열 ${expression} 문자열`
tag `문자열 ${expression} 문자열`
```

  - 여기서 사용하는 ` 는 역따옴표(Backtick)이다.
  - 역따옴표 안에 문자열과 표현식을 작성할 수 있다.
  - 따라서 표현식에 따라 출력되는 결과가 다르며 문자열과 표현식의 결과를 묶어 문자열로 표현하는 것이 템플릿 리터럴이다.

Template literals

- 템플릿 리터럴

```
console.log("1:", `123ABC가나다`);

console.log("2:", '라인 1\n라인 2');
console.log("3:", `첫 번째 줄
두 번째 줄`);

let one = 1, two = 2;
console.log("4:", `${one} + ${two}는 ${one + two}이다`);
```

```
1: 123ABC가나다
2: 라인 1
라인 2
3: 첫 번째 줄
                두 번째 줄
4: 1 + 2는 3이다
```

```
var phone1 = '010';
var phone2 = 1234;
var phone3 = 5678;

console.log("제 전화번호는 "+phone1+"-"+phone2+"-"+phone3+" 입니다.");
console.log(`제 전화번호는 ${phone1}-${phone2}-${phone3} 입니다.`);
```

```
제 전화번호는 010-1234-5678 입니다.
제 전화번호는 010-1234-5678 입니다.
```

Template literals

- 여러 줄 문자열
  - 여러 줄 문자열을 사용할 경우 보통은 이스케이프 문자인 '\n'을 사이에 넣어서 표현하였다.
  - 하지만 이 방식은 기존의 글을 그대로 옮겨 붙여넣기에는 다른 전처리를 해줘야 만 했다.
  - 이런 전처리 없이 삽입하기 위해 Backtick(‘)을 문자열 기본 값으로 삽입하여 넣는 방식이 ECMA6 이후부터 나오기 시작하였다.
  - 문제는 백틱을 통해 사용한 여러 줄 문자열은 탭 공백도 인식을 하기 때문에 주의해야 한다.

```
console.log("동해물과 백두산이 마르고 닳도록\n"+
"하느님이 보우하사 우리 나라만세\n"+
"무궁화 삼천리 화려 강산\n"+
"대한사람 대한으로 길이 보전하세");
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리 나라만세
무궁화 삼천리 화려 강산
대한사람 대한으로 길이 보전하세
```

```
console.log(
`동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리 나라만세
무궁화 삼천리 화려 강산
대한사람 대한으로 길이 보전하세`);
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리 나라만세
무궁화 삼천리 화려 강산
대한사람 대한으로 길이 보전하세
```

```
console.log(
        `동해물과 백두산이 마르고 닳도록
        하느님이 보우하사 우리 나라만세
        무궁화 삼천리 화려 강산
        대한사람 대한으로 길이 보전하세`);
```

```
동해물과 백두산이 마르고 닳도록
        하느님이 보우하사 우리 나라만세
        무궁화 삼천리 화려 강산
        대한사람 대한으로 길이 보전하세
```

Template literals

- 표현식 삽입법
  - 만약 연산의 결과를 템플릿 리터럴에 선언하고 싶다면 아래와 같이 할 수 있다.

```
var a = 5;
var b = 10;
console.log(`Fifteen is ${a + b} and
not ${2 * a + b}.`);
// "Fifteen is 15 and
// not 20."
```

Tagged Template

- tagged template
  - 템플릿 앞에 tag를 작성한 형태를 태그드(tagged) 템플릿이라고 한다..

```
[구문]
tag `문자열 ${expression} 문자열`
```

  - tag 위치에 호출할 함수 이름을 작성한다.
  - 함수를 호출하기 전에 템플릿에서 문자열과 표현식을 분리하고 이를 파라미터 값으로 넘겨준다.
  - 함수 이름이 작성된 템플릿을 태그드 템플릿이라고 하고 호출되는 함수를 태그 함수라고 한다.

Tagged Template

- tagged template

```
let one = 1, two = 2;
function tagFunction(text, value) {
  console.log("1:", text[0]);
  console.log("2:", value);
  console.log("3:", text[1]);
  console.log("4:", typeof text[1]);
}
tagFunction `1+2=${one + two}`;
```

```
1: 1+2=
2: 3
3:
4: string
```

Tagged Template

- tagged template
  - 다수의 파라미터 형태

```
let one = 1, two = 2;
function tagFunction(text, plus, minus) {
  console.log(text[0], plus, text[1]);
  console.log(minus, text[2], text[3]);
}
tagFunction `1+2=${one + two}이고 1-2=${one - two}이다`;
```

```
1+2= 3 이고 1-2=
-1 "이다" undefined
```

Tagged Template

- String.raw
  - String.raw는 템플릿의 표현식은 변환하지만 특수 문자와 유니코드는 문자열로 인식한다.

```
let one = 1, two = 2;
console.log("1:", String.raw`1+2=${one + two}`);

console.log("2:", `줄 바꿈-1\n줄 바꿈-2`);
console.log("3:", String.raw`줄 바꿈-1\n줄 바꿈-2`);

console.log("4:", `Unicode \u0031\u0032`);
console.log("5:", String.raw`Unicode \u0031\u0032`);
```

```
1: 1+2=3
2: 줄 바꿈-1
줄 바꿈-2
3: 줄 바꿈-1\n줄 바꿈-2
4: Unicode 12
5: Unicode \u0031\u0032
```

Tagged Template

- String.raw() : 문자열 전개, 조합
  - 첫번째 파라미터의 raw 프로퍼티 값인 문자열을 문자 하나씩 전개하면서 두 번째 이후의 파라미터를 조합하여 반환한다.

구분	타입	데이터(값)
형태		String.raw()
파라미터	Object	{raw : "문자열" } 형태
	Any	조합할 값, 콤마로 구분하여 다수 작성 가능
반환	String	실행 결과

Tagged Template

- String.raw() : 문자열 전개, 조합

```
let one = 1, two = 2;
let result = String.raw({raw: "ABCDE"}, one, two, 3);
console.log(result);
```

A1B2C3DE