

## 첫째 주

- 자바스크립트란?
- 변수
- 변수 타입
- 연산자
- 함수
- BOM, DOM 이란?
- DOM 구조
- DOM 요소 가져오기

# Javascript

## Javascript 란?

javascript는 HTML, CSS 로 이루어진 정적인 페이지를 동적으로 변경해주는 언어입니다.

javascript를 통해 경고창을 띄울수 있고, 서버로 요청을 보내거나, Drag & Drop 기능들을 만들 수 있습니다.

자바스크립트는 다음의 특징을 가지고 있습니다.

- 인터프리터 형식의 언어입니다.
  - 코드를 한줄한줄 읽어서 동작합니다.
  - V8 엔진의 경우 인터프리터의 단점을 개선하기 위해 인터프리터 형식에 컴파일 형식을 같이 사용합니다.
- 자유도가 높고 단순합니다.
  - 문법이 단순해서 배우기 쉽습니다.
  - 하지만 단순해진만큼 코드의 작성방법이 개개인에 따라 많이 달라집니다.
- 다양한 영역에서 활용가능합니다.
  - 자바스크립트를 통해 웹, 앱, 데스크톱앱, 서버 등등 구현이 가능합니다.

# javascript 작성

자바스크립트는 html 내에 **script** 태그 사이에 작성할 수 있습니다.

그러면 브라우저에서 html 파일을 읽을 때 script 태그 사이의 자바스크립트 코드를 실행하게 됩니다.

만약 자바스크립트 코드에 console.log가 호출되었다면 브라우저의 콘솔에 실행 결과가 출력됩니다.

```
<body>
<script>
  console.log('Hello World !!!')
</script>
</body>
```

html 파일 안에 js를 모두 작성하게 된다면 html 파일 자체가 엄청나게 커지고 유지보수 하는데 어려움이 생기게 됩니다.

그래서 대부분 js파일을 따로 만들어 작업하게 되는데, 따로 생성한 js파일 역시 script 태그를 이용해 html에서 불러오는것이 가능합니다.

```
<body>
<script>
  console.log('Hello World !!!')
</script>
<script src="ex1.js"></script>
</body>
```

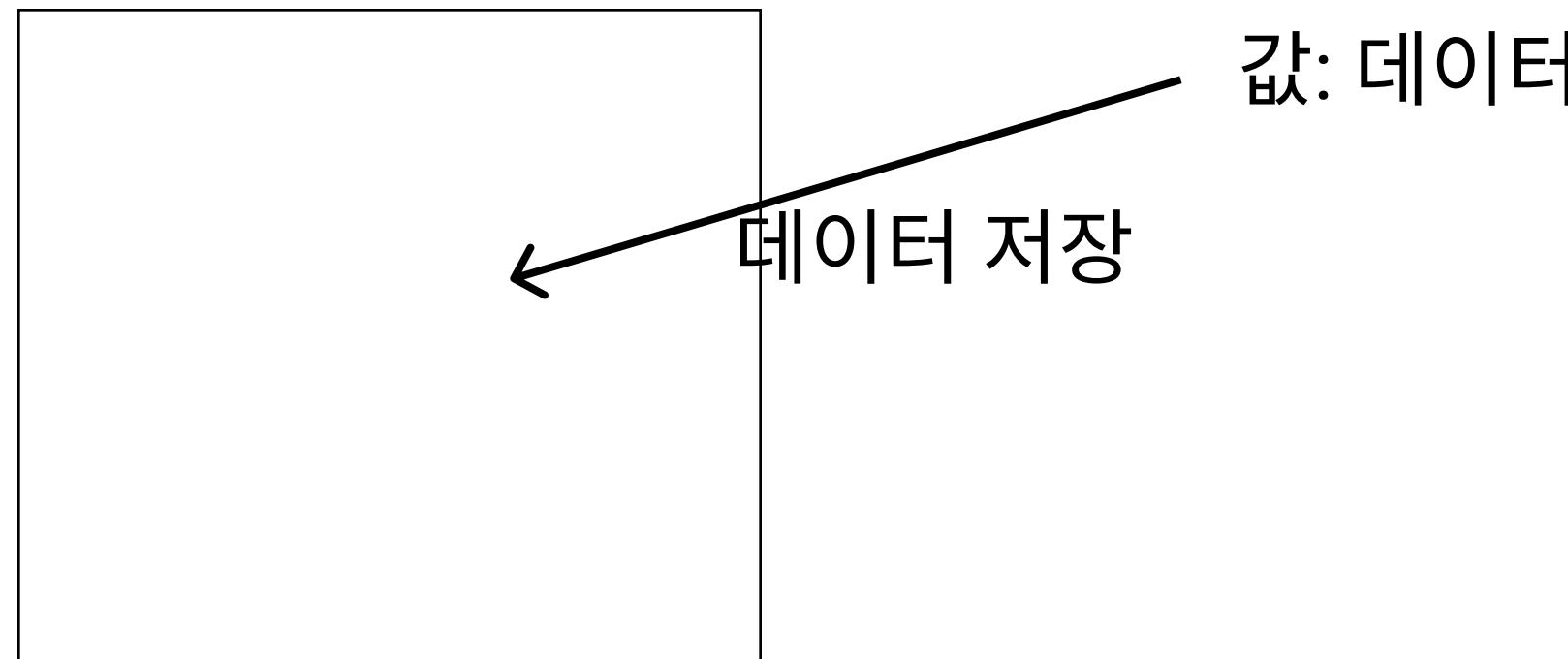
# javascript 변수

## var, let, const

var, let, const는 자바스크립트에서 변수를 선언하기 위한 예약어입니다.

프로그래밍 언어를 통해 변수를 선언한다는 것은 데이터를 저장하기 위해 컴퓨터 메모리를 할당한다는 의미입니다.

데이터 공간 = 변수



변수 선언을 통해 데이터 공간을 확보하고, 확보한 공간에 우리가 저장하고 싶은 데이터를 넣습니다.

이 저장된 데이터는 프로그램이 종료되면 사라집니다.

# javascript 변수

## var, let, const

```
var valueName = 'this is data'
```

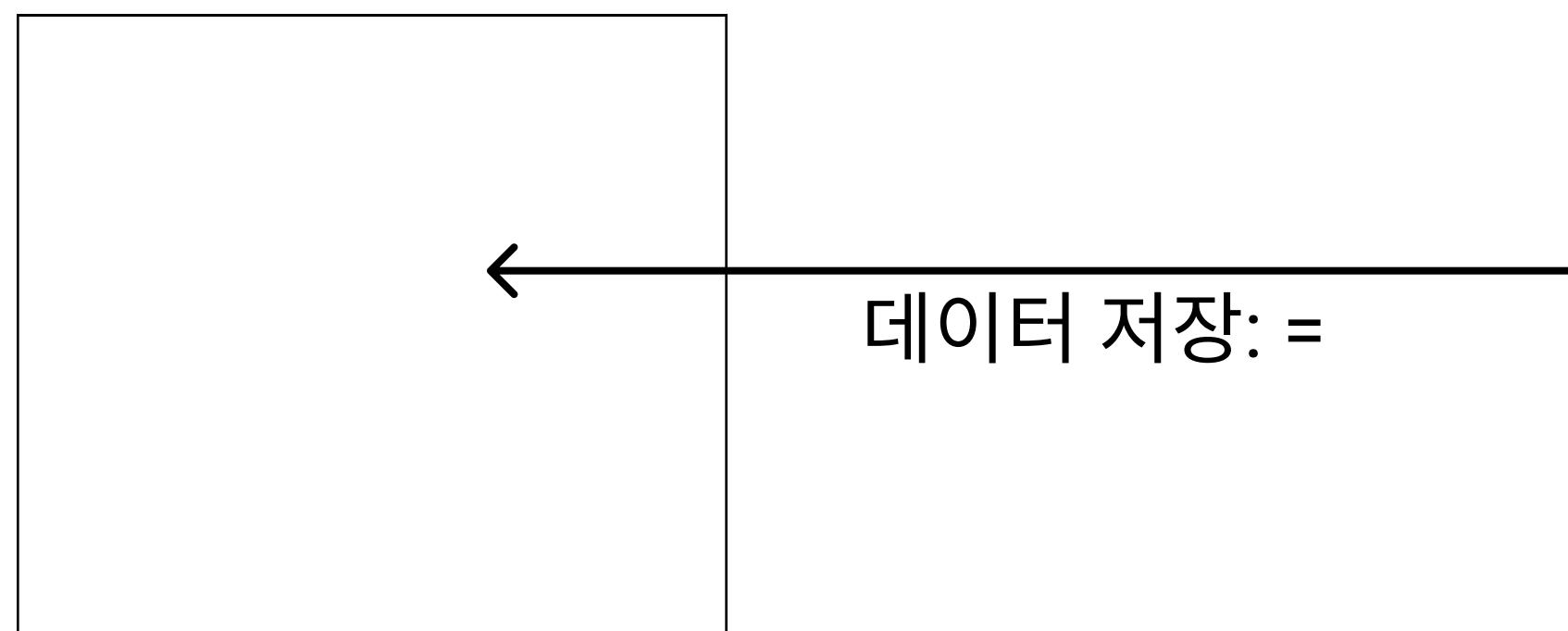
변수 선언은 위와같이 예약어 var를 사용후에 변수명을 입력하시면 됩니다.

데이터 공간 = 변수

데이터 공간 이름 = valueName

위의 코드를 그림으로 나타내면 다음과 같습니다.

데이터 공간: valueName



데이터: 'this is data'

# javascript 변수

## var, let, const

var 를 통한 변수 선언은 자바스크립트 버전이 낮을 때 사용된 방식입니다.

현재는 let, const 를 통한 변수 선언이 권장됩니다.

let 을 통한 변수선언은 기존의 var와 같이 변수 선언 후 데이터를 자유롭게 변경하고 싶을 때 사용됩니다.

하지만 const의 경우 처음 변수를 선언할 때 데이터를 반드시 저장시켜야 하고, 이후에는 데이터 변경이 불가능합니다.

```
let myValue = 'this is my data  
const myConstValue = 'origin value'  
  
myConstValue = 'new value' // error
```

# javascript 변수 타입

모든 언어의 기본은 데이터 타입을 파악하는데서 시작합니다.

자바스크립트의 데이터 타입은 크게 **기본 타입과 참조 타입**으로 나뉩니다.

기본 타입으로 **number, string, boolean, undefind, null** 이 있으며 이 외의 타입들은 참조 타입으로 구분됩니다.

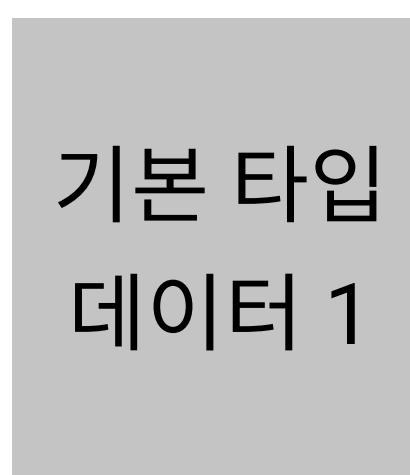
참조 타입에는 **object, array, function** 등이 있습니다.

기본 타입은 데이터 형태 중에서 **가장 단순한 형태**로 그 자체가 하나의 값의 역할을 합니다.

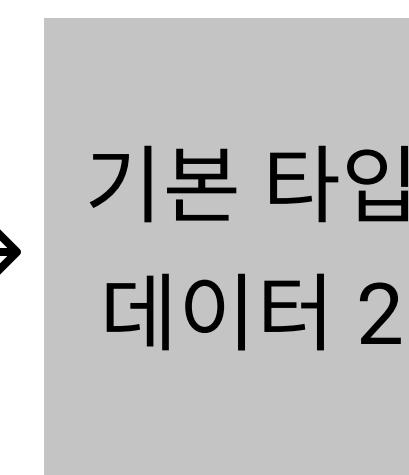
반면 참조 타입의 경우 데이터 자체가 여러 값으로 구성된 **복합값의 표현**되고 참조 타입은 데이터가 저장된 **메모리상의 주소**를 사용하기 때문에 기본 타입과 참조 타입의 데이터 복사 과정이 다르게 나타납니다.

**모든 언어에서 변수에 저장되는 데이터는 메모리에 저장된다는걸 반드시 기억해주세요!**

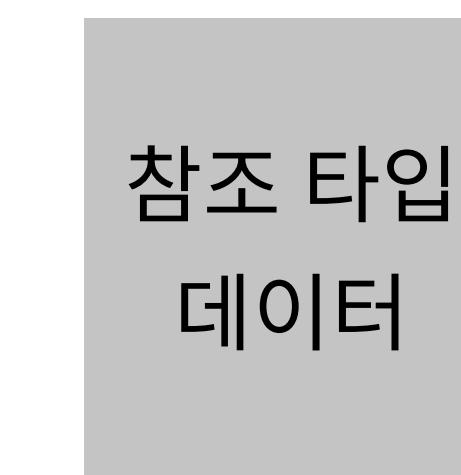
메모리 주소1



메모리 주소2



메모리 주소1



복사

메모리 주소만 복사

# **javascript 변수 타입 - 기본 타입**

## **number, string, boolean**

자바스크립트의 변수에는 저장되는 데이터의 형식에 따라 타입이 지정됩니다.

number 타입의 경우 숫자형 데이터(1, 2, 3...)를 저장할 수 있습니다.

string 타입은 문자열 데이터('abc', 'def'...)를 저장할 수 있습니다.

boolean 타입은 참, 거짓을 판별하는 데이터(true, false) 형식을 저장합니다.

```
let num = 10;  
let str = 'string';  
let isLoading = false
```

자바스크립트의 경우 타입을 강하게 지정하지 않기 때문에 변수의 데이터 타입 변환이 자유로운편 입니다.

```
let goodValue = 10;  
goodValue = 'this is sentence'  
goodValue = true
```

# javascript 변수 타입 - 기본 타입

## null, undefined

null은 값이 비어있음을 나타내는 데이터입니다. 의도적으로 값이 없음을 명시하기 위해서 사용됩니다.  
undefined는 아직 변수가 정의되지 않았음을 의미합니다. undefined는 타입이자 값을 나타냅니다.

```
let test;  
let empty = null;  
  
console.log(test); //undefined  
console.log(empty); // null
```

null은 의도적으로 사용되는 빈 값이기 때문에 이미 값이 들어가 있는 변수를 초기화 할 때 사용 가능 합니다.

```
let something = 'this is value'  
console.log(something)  
  
something = null  
console.log(something)
```

# javascript 변수 타입 - 참조 타입

자바스크립트에서 객체는 단순히 '이름(key): 값(value)' 형태의 **프로퍼티들을** 저장하는 컨테이너 역할을 합니다.

참조 타입은 기본 타입이 하나의 값만 가지는것과 다르게 **여러개의 프로퍼티들을 포함할 수 있고**  
이 프로퍼티들에는 기본 타입 값들을 포함하거나 다른 객체를 가리킬 수 있습니다.

프로퍼티가 기본 타입을 포함하고 있으면 **멤버 변수**라 부르고  
함수를 포함하고 있으면 **메서드**라고 부릅니다.

객체를 생성하는 방법에는 **Object 생성자 함수 이용, 객체 리터럴을 이용하는 방법, 생성자 함수 이용방법** 이 있는데  
주로 객체 리터럴을 이용하는 방식을 사용합니다.

객체 리터럴 방식은 표기법을 통해 객체를 생성하는 방식인데 {} 를 사용하거나 [] 를 사용해 객체를 생성할 수 있습니다.

# javascript 변수 타입 - 참조 타입

## Array-배열

배열은 자바스크립트의 특별한 객체입니다.

배열을 리터럴 방식으로 생성할 경우 [] 표기법을 사용해 생성할 수 있습니다.

```
let numList = [1, 2, 3, 4, 5, 6, 7]
```

위 numList는 숫자들의 목록을 담습니다.

각각의 요소들에 접근하고 싶으면 **index**를 통해 접근 가능합니다. (index는 0부터 시작됩니다)

배열의 길이를 알고싶으면 **length** 프로퍼티를 이용하면 됩니다.

```
// 2번 index 요소 확인  
numList[2] // 3
```

```
// 배열 길이 확인  
numList.length // 7
```

# javascript 변수 타입 - 참조 타입

다른 언어들과는 다르게 자바스크립트에서의 배열은 요소들끼리 데이터의 타입이 달라도 됩니다.

```
let someList = ['one', 2, true, 4, null, 6, 7]
```

# javascript 변수 타입 - 참조 타입

## Object-객체

객체는 여러 데이터를 하나로 묶어놓은 형태의 데이터 타입입니다.

원래 객체는 다른 프로그래밍 언어에선 현실의 사물을 프로그래밍으로 반영한 것으로 표현이 됩니다.

현실의 사물들은 여러가지 **특징(프로퍼티)**을 가지고있고 **기능(메소드)**도 있지만  
하나의 이름으로 부릅니다.

- 예: 커피자판기
  - 무게: 200kg, 가격: 500만원, 크기: 큼
  - 기능: 아메리카노 만들수 있음, 라떼 만들수 있음, 카푸치노 만들수 있음

자바스크립트의 객체도 그러한 성격을 가지고 있습니다.

# javascript 변수 타입 - 참조 타입

```
let aboutMe = {  
    name: 'KIM',  
    age: 20,  
    height: 190,  
    weight: 100  
}
```

위의 코드는 어떤 사람에 대한 데이터를 하나의 집합으로 표현한 것입니다.  
우리는 이러한 데이터 형태를 자바스크립트에선 **Object** 라고 합니다.

여기서 name: 'KIM', age: 20 ... 들을 **속성(property)**이라고 합니다.  
그리고 속성의 이름에 해당하는 **name, age, height, weight**들을 **key** 라고 불리고  
각 key에 해당하는 값들('KIM', 20, 190, 100)들을 **값(value)**라고 부릅니다.

aboutMe라는 객체는 현재 프로퍼티로만 구성되어 있지만 함수 또한 등록이 가능합니다.  
프로퍼티와 메소드가 포함된 형태의 객체는 이후 객체를 구체적으로 공부할 때 살펴보겠습니다.

## javascript 변수 타입 - 참조 타입

```
let aboutMe = { name: 'KIM', age: 20, height: 190, weight: 100 }
```

```
aboutMe.name // 'KIM'  
aboutMe['name'] // 'KIM'
```

객체들의 프로퍼티에 접근하는 방법은 . 을 사용하거나 [] 를 사용해서 해당 속성의 이름을 적으면 됩니다.  
해당 프로퍼티의 값을 바꾸려면 해당 프로퍼티에 접근해서 데이터를 넣으면 됩니다.

```
aboutMe.age = 30
```

객체의 프로퍼티를 동적으로 생성할 수 있습니다.  
아래의 코드는 생성시에 없는 'character' 프로퍼티를 생성하는 코드입니다.

```
aboutMe.character = 'nice'
```

## EX: 변수 선언하기

1. const를 이용해 string형 변수를 선언해보세요  
변수명: myName, 값: 이름
2. let을 이용해 number형 변수를 선언해보세요  
변수명: age, 값: 300
3. let을 이용해 boolean형 변수를 선언해보세요  
변수명: isHappy, 값: true
4. let을 이용해 string형 변수를 선언하고 값을 제거해보세요  
변수명: nextSchedule, 값: study, 값 변경: null

## EX: object 만들어보기

1. myCompany 객체를 리터럴 방식으로 만들어보세요

객체명: myCompany

프로퍼티명: companyName, worker, income

값

companyName: 문자열

worker: 숫자값

income: 숫자값

2. myCompany 객체를 myCompany2 변수에 복사해보세요

3. myCompany2 변수에서 worker 프로퍼티의 값을 30으로 변경해보세요

# **javascript 연산자**

자바스크립트에는 다음과 같은 연산자가 있습니다.

- 산술연산자
- 문자열연산자
- 증감연산자
- 대입연산자
- 비교연산자
- 삼항연산자
- 논리연산자

각각의 연산자에 대해 알아보도록 합시다.

# javascript 연산자

## 산술연산자

산술연산자는 말 그대로 산술 계산을 하는 연산자를 뜻합니다.

기본적인 덧셈, 뺄셈, 곱셈, 나눗셈이 이에 해당합니다.

```
let a = 1 + 2; // 3
let b = 2 - 1; // 1
let c = 3 * 4; // 12
let d = 10 / 4; // 2.5
let e = 10 % 4; // 2
```

다른 사칙연산의 경우 이미 아는것들이지만 %는 처음보실겁니다.

% 는 나눗셈을 한 나머지를 계산해주는 연산자 입니다.

# javascript 연산자

## 문자열연산자

문자열 연산자는 문자열을 연결해주는 연산자입니다.

자바스크립트에서 문자열 연산자로는 + 기호가 사용됩니다.

```
let strA = 'hello' + 'world'  
let strB = 'this number is ' + 10  
let strC = 20 + 'is my age'
```

자바스크립트의 문자열 연산자는 다른 언어와는 다르게 더해지는 값의 형태가 문자열이 아니더라도  
자동으로 문자열 형 변환을 시켜서 연결시킵니다.

# javascript 연산자

## 증감연산자

증감연산자는 숫자에 증감연산자 `++`, `--` 사용하면 1을 더하거나 빼주는 연산자를 말합니다.

증감연산자에는 전위증감연산자, 후위증감연산자가 있는데 증감연산자의 위치에 따른 명칭입니다.

전위증감연산자의 경우 코드의 실행전에 증감연산자가 먼저 동작하고

후위증감연산자의 경우 코드가 실행된 후 증감연산자가 동작합니다.

```
let number = 1
number++
console.log(number) // 2
console.log(number++) // 2
console.log(number) // 3
console.log(++number) // 4
```

# javascript 연산자

## 대입연산자

대입연산자는 앞서서 많이 사용했던 = 연산자를 의미합니다.

변수에 값을 대입할 때 사용되었기 때문에 대입연산자라고 불립니다.

사칙연산과 대입연산자를 줄여서 쓰는 연산자가 있는데 이를 복합대입연산자라고 합니다.

사용은 다음과 같습니다.

```
let number = 1;  
number += 1; // number = number + 1  
number -= 1 ; // number = number - 1  
number *= 10 // number = number * 10  
number /= 10 // number = number / 10
```

# javascript 연산자

## 비교연산자

비교연산자는 데이터의 값을 비교해서 해당 연산자에 따라 참 거짓을 판별해주는 연산자입니다.

종류로는 ==, !=, >, <, >=, <= 가 있으며 순서대로

같다, 다르다, 크다, 작다, 크거나 같다, 작거나 같다 를 의미합니다.

```
let betweenA = 1;  
let betweenB = 2;  
  
betweenA == betweenB // false  
betweenA != betweenB // true  
betweenA > betweenB // false  
betweenA < betweenB // true  
betweenA >= betweenB // false  
betweenA <= betweenB // true
```

# **javascript 연산자**

## **삼항연산자**

삼항연산자는 조건문의 줄임 형태입니다.

앞에 제시한 조건이 참인지 거짓인지에 따라 값을 결정해주는 연산자 입니다.

문법의 형태는 다음과 같습니다.

조건 ? 참 : 거짓

```
let number = 10  
let truefalse = number > 10 ? 20 : 1
```

# javascript 연산자

## 논리연산자

논리연산자는 `&&`, `||`, `!` 연산자가 있습니다.

순서대로 AND, OR, NOT 연산을 하는데요

`&&(AND)` 연산자는 연산자 앞 뒤의 값이 참일경우 `true`, 하나라도 거짓일 경우 `false`를 반환합니다.

`||(OR)` 연산자는 연산자의 앞 뒤에 하나라도 참일경우 `true`, 둘 다 거짓일 경우 `false`를 반환합니다.

`!(NOT)` 연산자는 참을 거짓으로, 거짓을 참으로 변환하는 연산자입니다.

```
let numberA = 10;
let numberB = 20;

(numberA == numberB) && (numberA != numberB) // false
(numberA == numberB) || (numberA != numberB) // true
!numberA // false
!!numberA // true
```

javascript에는 거짓에 해당하는 값이 있습니다. `false`, `''`, `0`, `NaN`, `undefined`, `null` 이 있죠.

그 이외의 값들은 `true`에 해당하기 때문에 위의 `!numberA` 는 `false`가 됩니다 (`numberA` 가 boolean type에선 `true` 임).

## EX: 연산결과 예상하기

산술연산자

`10 + 20`

`34 % 10`

증감연산자

```
let number = 1
```

```
number++
```

```
console.log('number: ', number);
```

```
console.log('number: ', -number);
```

```
console.log('number: ', number);
```

```
console.log('number: ', number++);
```

```
console.log('number: ', number);
```

문자열 연산자

`' str test ' + 3`

`3 + 4 + ' str test '`

`' str test ' + 3 + 4`

대입 연산자

```
let number = 1
```

```
number += 3
```

```
console.log('number: ', number);
```

```
number *= 4
```

```
console.log('number: ', number);
```

```
number %= 10
```

```
console.log('number: ', number);
```

## EX: 연산결과 예상하기

논리 연산자

비교 연산자

```
let betweenA = 20  
let betweenB = 30
```

```
console.log(betweenA == betweenB)  
console.log(betweenA != betweenB)  
console.log(betweenA > betweenB)  
console.log(betweenA < betweenB)
```

삼항 연산자

```
let test = 20 > 30 ? 'AAA' : 'BBB'
```

```
console.log('test: ', test);
```

```
test = 20 > 30  
? 'AAA'  
: 40 < 50  
? 'CCC' : 'DDD'
```

```
console.log('test: ', test);
```

```
let numberA = 0
```

```
let numberB = 20
```

```
let numberC = 30
```

```
// numberB
```

```
console.log('numberB: ', !numberB);
```

```
// numberA
```

```
console.log('numberA: ', !numberA);
```

```
// numberA && numberB
```

```
console.log('numberA && numberB: ', !!  
numberA && !!numberB);
```

```
// numberA || numberB
```

```
console.log('numberA || numberB: ', !!  
numberA || !!numberB);
```

# **javascript 함수**

자바스크립트에는 다른 언어들과 마찬가지로 함수가 있습니다.

함수는 특정 로직들을 하나의 기능으로 만들어놓은것이라 생각하시면 됩니다.

함수를 사용하는 이유는 다음과 같습니다

- 코드의 중복제거 및 재사용
- 유지보수 용이성
- 코드 가독성 증가

함수는 로직을 미리 만들어 놓고 필요한 곳에서 가져다 쓰면 되기 때문에 계속해서 반복되는 기능들을 매번 구현할 필요가 없습니다. 그래서 코드의 중복성이 사라지고 가독성이 증가하게 됩니다.

또한 함수 사용에 있어서 버그나 기능 개선이 필요한 경우, 해당 함수에 접근해서 수정하면 되기 때문에 유지보수가 쉬워집니다.

# javascript 함수

## 함수 선언

함수 선언 문법은 다음과 같습니다.

```
function 함수이름( 매개변수 ) {  
    ...실행로직  
    return 반환값  
}
```

함수는 자바스크립트의 `function`이라는 예약어를 통해 만들 수 있습니다.

`function` 뒤에 함수이름을 작성해서 함수를 만들 수 있습니다.

매개변수와 반환값은 천천히 알아보고 먼저 가장 간단한 형태의 함수를 만들어 보도록 합시다.

# javascript 함수

## 매개변수와 반환값이 없는 함수

```
function logger() {  
    console.log('this is my first function !!!')  
}  
  
logger()  
logger()  
logger()
```

위의 함수는 매개변수와 반환값이 없는 함수입니다.

단순하게 함수의 로직으로on `console.log`를 찍어주는 기능을 가지고 있습니다.

`function` 뒤에 `logger`라고 함수의 이름을 명시해주고 함수를 만들었습니다.

그리고 불러서 사용할 땐 밑의 코드처럼 함수명과 소괄호를 통해 사용하면 됩니다.

# javascript 함수

## 매개변수는 있지만 반환값이 없는 함수

```
function customLogger(logText) {  
    console.log('logText is :: ', logText)  
}
```

```
function sum(a, b) {  
    const res = a + b;  
    console.log('sum :: ', res)  
}
```

```
customLogger('my log text')  
sum(1, 2)
```

매개변수를 받는 함수 `customLogger` 와 `sum` 함수를 만들어봤습니다.

`customLogger`의 경우 매개변수를 하나 받고 `sum` 함수는 2개를 받고 있습니다.

매개변수를 함수에서 받아서 쓸 수 있게되면서 함수를 조금 더 유용하게 사용할 수 있습니다.

만약 `sum`에 매개변수 `a,b`를 받을 수 없다면 우리는 `sum` 이라는 함수를 더해야할 값들이 바뀔때마다 새로 함수를 만들어야겠지만 이렇게 매개변수를 받음으로써 여러가지 덧셈의 경우를 매개변수로 받아서 처리할 수 있습니다.

# javascript 함수

## 매개변수와 반환값이 있는 함수

```
function sum(a, b) {  
    const res = a + b;  
  
    return res  
}  
  
const sumResA = sum(1, 2)  
const sumResB = sum(3, 4)
```

함수의 반환값이 있는 경우는 `return` 예약어를 통해 값을 반환시킵니다.

위의 `sum` 함수의 경우 `a,b`를 입력받아 더한 결과를 반환해주는 로직을 가진 함수입니다.

이렇게 함수 내부에서 처리한 결과값을 외부에서 사용 가능하게 함으로써 반환값이 필요한 곳에 사용 가능합니다.

## EX: 함수 만들어보기

함수명: pagination

매개변수: page, limit

반환값: page, limit를 가진 object

로직

매개변수로 입력받은 page, limit를 각각 console.log로 찍어보고

객체로 page, limit를 반환시킴

# BOM, DOM 이란?

BOM은 Browser Object Model의 약자입니다.

BOM은 웹 브라우저의 기능들을 추상화 하여 프로그래밍적으로 제어할 수 있도록 제공하는 수단입니다.

BOM은 window라는 객체를 최상위 객체로 가지고 있으며 그 하위에는 document, screen, location, history, navigator 등등 브라우저를 제어하기 위한 객체들을 제공하고 있습니다.

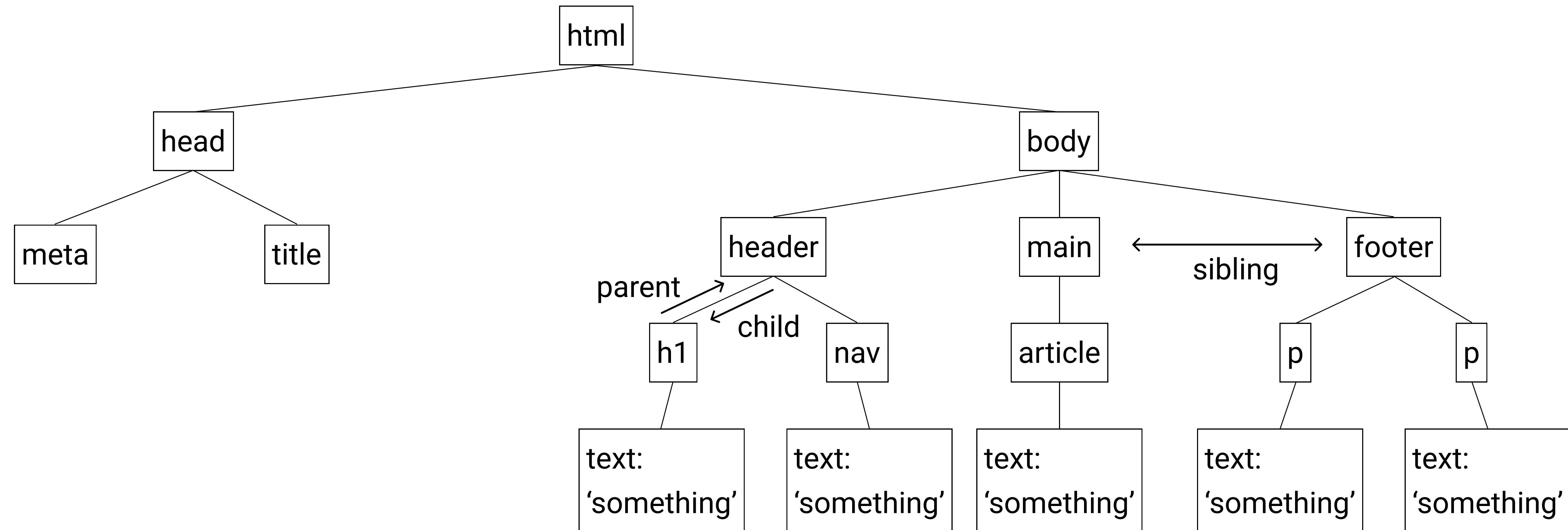
DOM은 Document Object Model의 약자입니다. 한글로 번역하면 문서 객체 모델이 되는데

여기서 문서 객체란 <html> <body> 같은 html문서의 태그들을 javascript로 접근할 수 있는 객체형태(Object)로 만들어진것을 문서 객체 모델이라 합니다.

개발자는 javascript를 통해 BOM, DOM 에 접근함으로써 브라우저나, 화면을 제어할 수 있습니다.

# DOM의 구조

개발자가 HTML을 작성해서 브라우저로 해당 파일을 읽으면  
브라우저에서 HTML을 해석하고 DOM을 생성합니다.  
여기서 DOM은 다음과 같이 Tree 구조를 가지고 있습니다.



# DOM의 구조

Tree 구조는 Node(노드)로 이루어진 자료구조를 말합니다.

Tree 자료구조는 다음과 같은 특징을 가지고 있습니다.

- 트리는 하나의 루트 노드를 가집니다.
- 루트노드는 0개 이상의 자식 노드를 가지고 있습니다.
- 자식노드 또한 0개 이상의 자식 노드를 가지고 있으며, 그 하위로 반복적인 구조를 가집니다.
- 각각의 노드는 서로 연관된 (edge)간선으로 구성되어 있습니다.

html에서 노드는 tag 노드와 text 노드 등등이 있는데, 보통 tag노드는 요소라고 부릅니다.

# DOM 요소 가져오기

이제 작업할 요소를 가져오는 방법 알아봅시다.

javascript document 객체를 사용해서 DOM tree에 있는 html 요소를 가져올 수 있습니다.

document객체에서 dom을 가져오는 방법으로는 다음 함수를 사용하면 됩니다.

- `document.getElementById()` : ID값으로 요소를 가져옵니다.
- `document.getElementsByClassName()` : class명으로 요소를 가져옵니다.
- `document.getElementsByTagName()` : Tag명으로 요소를 가져옵니다.
- `document.querySelector()` : selector rule을 이용해서 요소를 가져옵니다. 가장 첫번째로 해당하는 1개 요소만 가져옵니다.
- `document.querySelectorAll()` : selector rule을 이용해서 요소를 가져옵니다. 해당하는 요소를 전부 가져옵니다.

```
const fromId = document.getElementById('elementId');
const fromClass = document.getElementsByClassName('element-class-name')
const fromTag = getElementsByTagName('li')
```

```
const fromSelector = document.querySelector('.some-class-name')
const fromSelectorAll = document.querySelectorAll('ul li')
```

## EX: 요소 가져오기

1. getElementsByTagName, getElementsByClassName, getElementById 함수들을  
querySelector, querySelectorAll로 바꿔보기
2. some-list의 두번째 li를 가상 선택자를 활용해서 가져오기
3. h3 태그를 가져오기
4. some-item class가 들어간 요소를 가져오기

# javascript minimap

## javascript core 문법

- 변수
- 연산자
- 함수

## DOM 다루기

- 요소 가져오기