

# Javascript

## Chaper1. 자료와 변수

# 기본 용어

- 키워드

- 자바스크립트 자체에서 쓰이는 의미가 있는 단어
- EX) class, await, import 등...

break	else	instanceof	true
case	false	new	try
catch	finally	null	typeof
continue	for	return	var
default	function	switch	void
delete	if	this	while
do	in	throw	with

- 식별자

- 프로그래밍 언어에서 이름을 붙일 때 사용
- 주로 변수명이나 함수명
- 규칙
  - 키워드를 사용하면 안됨
  - 숫자로 시작하면 안됨
  - 특수문자는 \_와 \$만 허용
  - 공백문자 포함 불가능

- 주석

- // 또는 /\* \*/
- 프로그램 사용에 아무런 영향을 미치지 않음

- 출력

- console.log("안녕하세요."); -> 콘솔창에 출력
- document.write("안녕하세요"); -> 브라우저에 출력

var num = 2;

키워드

식별자

자료형

# 기본 자료형

- 문자열(String) 자료형

```
var str = "안녕하세요";
```

- “(작은 따옴표) 혹은 ””(큰따옴표)를 붙이면 문자열
- ‘+’ 기호를 이용해 문자열끼리 합침(문자 연결 연산자)
- var num = ‘1’; 은 문자
- 문자열 안에 따옴표를 사용해야 하려면 ‘작은 따옴표 안에 “큰 따옴표”  
“큰 따옴표 안에 ‘작은 따옴표’”
- 이스케이프 문자(\) 사용시 따옴표를 그대로 사용가능
- \n: 줄바꿈, \t: 탭, \\: 역슬래시(\)

- 숫자(Number) 자료형

```
var num= 100;
```

연산자	설명
+	더하기
-	빼기
*	곱하기
/	나머지
%	나누기

# 기본 자료형

- 불린(Boolean) 자료형

```
var flag= true;
```

연산자	설명
>	왼쪽이 더 크다
<	오른쪽이 더 크다
>=	왼쪽이 더 크거나 같다
<=	오른쪽이 더 크거나 같다
==	양쪽이 같다(값 비교)
!=	양쪽이 다르다(값 비교)
===	양쪽이 같다(값 비교, 자료형 타입 비교)
!==	양쪽이 다르다(값 비교, 자료형 타입 비교)

- Undefined, null, 0, 빈문자열("")을 제외한 모든 것은 True
- Null: 값이 의도 적으로 비어있음, Undefined: 값이 지정되지 않음

```
var name = prompt("당신의 이름은?", "");  
if(name) {  
    // 빈 문자열이 아니면 코드 실행  
}
```



```
var name = prompt("당신의 이름은?", "");  
if(name !== "") {  
    // 빈 문자열이 아니면 코드 실행  
}
```



# 기본 자료형

- 불린(Boolean) 자료형

연산자	설명
&&	논리곱 연산자(AND), <b>두 개 true여야 true</b>
	논리합 연산자(OR), <b>하나만 true여도 true</b>
!	부정(NOT), <b>반대로</b>

- 자료형 검사
  - 숫자, 문자, 불린 같은 자료형 확인

`typeof(자료)`

# 상수와 변수

- 상수
  - 불변의, 값에 이름을 붙이면 값을 수정할 수 없음
  - `const` 로 선언
- 변수
  - 변할 수 있는 수
  - `var`, `let` 으로 선언
- 복합 대입 연산자

연산자	설명	사용 예	의미
<code>+=</code>	기존 변수의 값에 값을 더한다.	<code>a += 1</code>	<code>a = a + 1</code>
<code>-=</code>	기존 변수의 값에 값을 뺀다.	<code>a -= 1</code>	<code>a = a - 1</code>
<code>*=</code>	기존 변수의 값에 값을 곱한다.	<code>a *= 1</code>	<code>a = a * 1</code>
<code>/=</code>	기존 변수의 값에 값을 나눈다.	<code>a /= 1</code>	<code>a = a / 1</code>
<code>%=</code>	기존 변수의 값에 나머지를 구한다.	<code>a %= 1</code>	<code>a = a % 1</code>

- 증감 연산자(1씩 증가 혹은 감소 한다)

연산자	설명
변수++	대입 후 1증가
++변수	1증가 후 대입
변수--	대입 후 1감소
--변수	1감소 후 대입

# 연산자 우선순위

1. ()
2. 단항 연산자(++, --, !)
3. 산술 연산자(\*, /, +, -)
4. 비교 연산자(>, >=, <, <=, ==, ==, !=, !=)
5. 논리 연산자(&&, ||)
6. 복합 대입 연산자(=, +=, -=, \*=, /=, %=)

# 자료형 변환

- 문자열 입력 받는 함수

`prompt(질문, 기본 값)`

- 불린 값 입력 받는 함수

`confirm(질문)`

- 숫자 자료형 변환
  - `Number(자료)`
  - `parseInt(자료)`
- 문자 자료형 변환
  - `String(자료)`
- 문자 자료형 변환
  - `Boolean(자료)`



# Javascript

## Chaper2. 제 어 문

# 조건문

- if

```
if(조건식) {  
    //조건식이 참일 때  
}
```

- if else

```
if(조건식) {  
    //조건식이 참일 때  
} else {  
    //조건식이 거짓일 때  
}
```

- if else if

```
if(조건식) {  
    //해당 조건식 일 때  
} else if(조건식) {  
    //해당 조건식 일 때  
} else {  
    //나머지에 해당 할때  
}
```

# 조건문

- switch

```
switch(변수) {  
    case 조건 A:  
        break;  
    case 조건 B:  
        break;  
    case 조건 C:  
        break;  
    default:  
        break  
}
```

→ 생략 가능

- 삼항 연산자

조건식 ? 참일 때 결과 : 거짓 일 때 결과

# 반복문

- 배열

```
var arr = ["hi", 1, "test", 2, 10]
```

[0]	[1]	[2]	[3]	[4]
hi	1	test	2	10

- for in

```
for (const 반복변수 in 배열) {  
  //실행 코드  
}
```

- for of

```
for (const 반복변수 of 배열) {  
  //실행 코드  
}
```

반복 변수에  
요소의 값이 들어간다

- for

```
for (let i=0; i < 반복횟수; i++) {  
  //실행 코드  
}
```

# 반복문

- while

```
while (조건식) {  
    //조건식이 참일 때 실행  
}
```

- do while

```
do {  
    //실행 후 비교  
} while(조건식)
```

- break
  - 조건문이나 반복문을 벗어날 때 사용.
  - Break아래 코드를 다 무시하고 조건문이나 반복문이 끝난다.
- continue
  - 반복문 안의 반복 작업을 멈추고 처음으로 돌아가 다음 작업을 반복.
  - continue아래 코드를 다 무시하고 처음으로 돌아간다.

# Javascript

## Chaper3. 함수

# 함수

- 이름 있는 함수(비익명 함수)

```
function 함수명() {  
    //실행 코드  
}
```

- 이름 없는 함수(익명 함수)



```
const 함수명 = function () {  
    //실행 코드  
}
```

- 콜백(Callback) 함수
  - 매개 변수로 함수를 전달한다
  - `forEach()`, `map()`, `filter()` 함수
- 타이머 함수
  - `setTimeout`, `setInterval`
- 즉시 실행 함수
  - 함수를 선언하자마자 즉시 실행
  - 스코프와 연관있다

```
(function () {  })()
```

# 함수

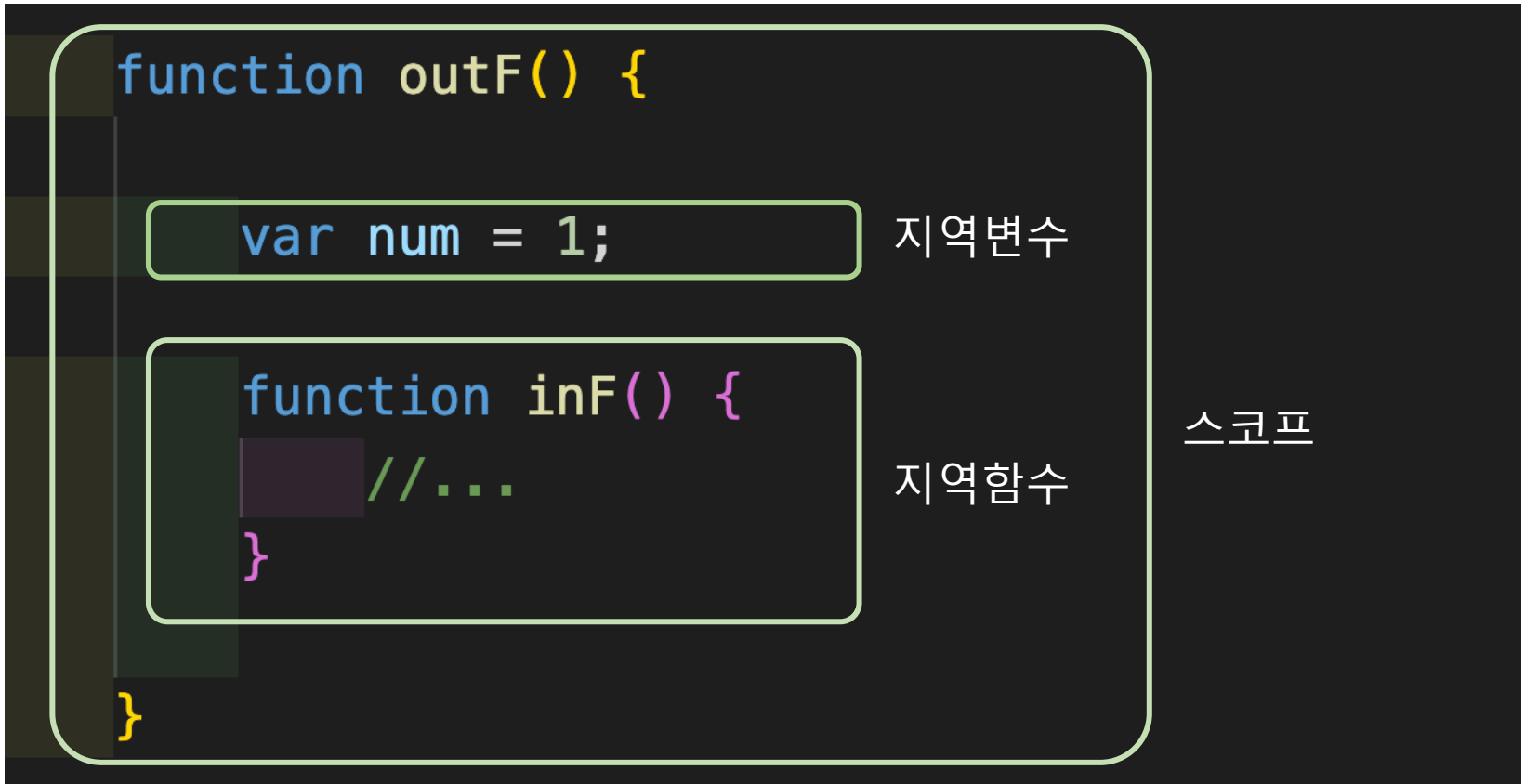
- 호이스팅
  - 비익명 함수와 익명 함수의 차이점

비 익명 함수	익명 함수
<pre>test();  function test() {   //... }</pre> 	<pre>test();  var test = function() {   //... }</pre> 

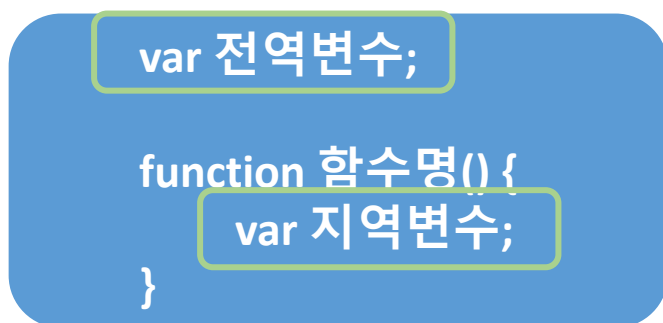


# 함수

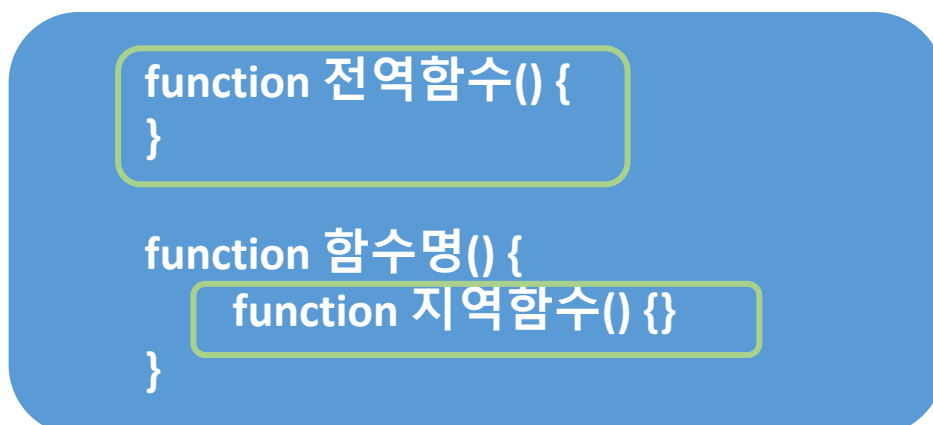
- 스코프
  - 사전적 의미는 범위
  - 변수 또는 함수의 유효 범위



- 전역 변수(함수), 지역 변수(함수)
  - 전역: 자바스크립트 어디서든 사용 할 수 있는 변수(함수)



- 지역: 스코프 영역에서 선언한 변수(함수), 스코프 영역에서만 사용 가능



# 함수

- 지역과 전역을 나누는 이유
  - 충돌을 피하기 위해!

개선 전

```
var num = 100;

// 라이언 개발자
function menu( ) {
    num += 100;
    alert(num);
}
menu();

// 어피치 개발자
function menu( ) {
    alert(num);
}
```



실행 결과는 100

라이언과 어피치가 만든 함수명이 같아서  
라이언이 만든 함수는 제거되고 어피치가  
만든 함수가 실행된다!

즉시 실행 함수로 개선

```
// 라이언 개발자
(function () {
    var num = 100;
    function menu() {
        num += 100;
        alert(num);
    }
    menu();
})();

// 어피치 개발자
(function () {
    var num = 100;
    function menu() {
        alert(num);
    }
})();
```

# 블록 변수 let

- 선언된 let 변수는 자신이 속한 중괄호{} 안에서만 사용할 수 있다.

```
function logScope(){  
  let local = 2;  
  if(local){  
    let local = '다른 값';  
    console.log('블록안의 var:', local);  
  }  
  console.log('local', local)  
}  
logScope();
```

# Javascript

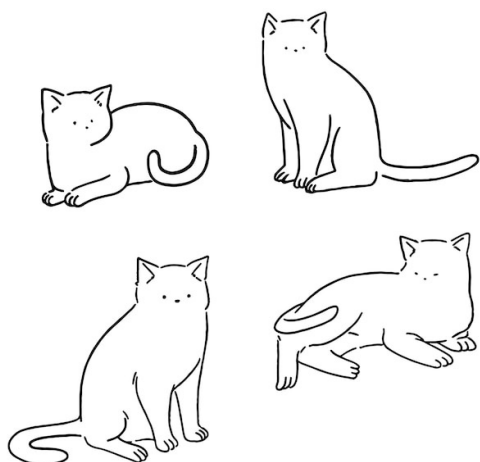
## Chaper4. 객체

# 객체

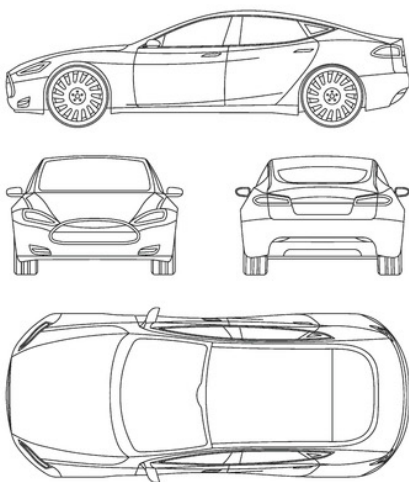
## • 객체란?

- object: 물건, 물체, 존재하는 모든 것
- 사람 사물 동물 모든 것이 될 수 있다.
- 객체 = 속성(스펙) + 기능(함수)

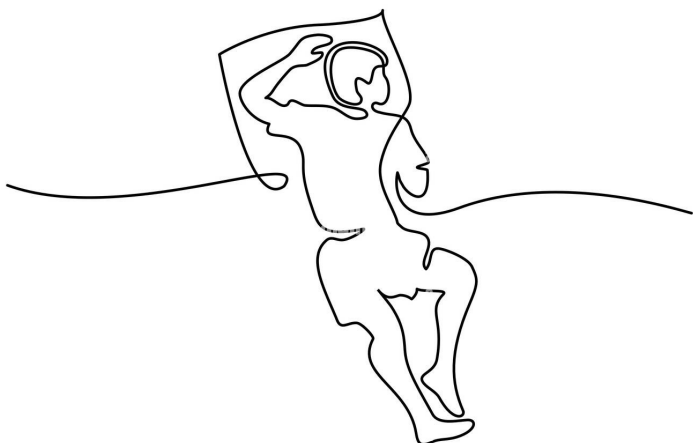
변수로          함수 = 메소드  
정의해요!



속성	함수
이름: 설백 나이: 5살 색: 흰색 종: 브리티쉬	밥먹기() 잠자기() 야옹하기()



속성	함수
이름: K5 색: 검은색 엔진: I5	시동켜기() 달린다() 멈춘다()



속성	함수
????	????

# 객체

- 객체를 만드는 법
  - 리터럴(Literal) 방식

```
// 변수를 선언합니다.  
const pet = {  
  name: '구름',  
  eat: function (food) {  
    alert(this.name + '은/는 ' + food + '을/를 먹습니다.')  
  }  
}  
  
// 메소드를 호출합니다.  
pet.eat('밥')
```

- 객체 생성자 함수 방식: new를 이용해 생성한다

```
function Pet(name) {  
  this.name = name;  
  this.eat = function (food) {  
    alert(this.name + "은/는 " + food + "을/를 먹습니다.");  
  };  
}  
  
const pet = new Pet("구름");  
pet.eat("밥");
```

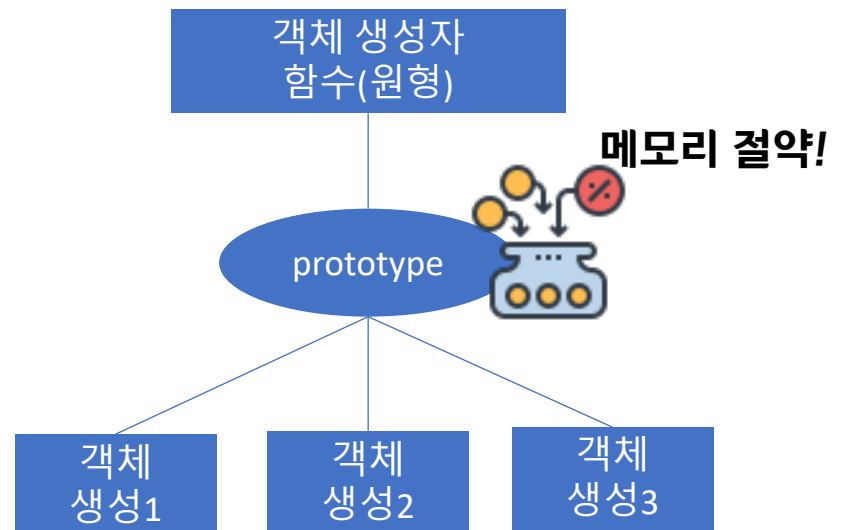
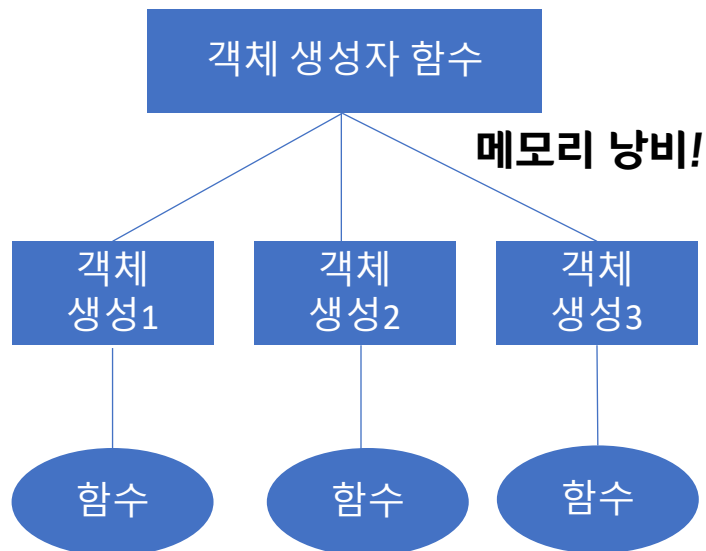
객체 생성

- prototype 방식

```
function Pet(name) {  
  this.name = name;  
}  
  
Pet.prototype.eat = function (food) {  
  alert(this.name + "은/는 " + food + "을/를 먹습니다.");  
};  
  
const pet = new Pet("구름");  
pet.eat("밥");
```

# 객체

- prototype VS 객체 생성자 함수



- Number, String, Date, Math 객체

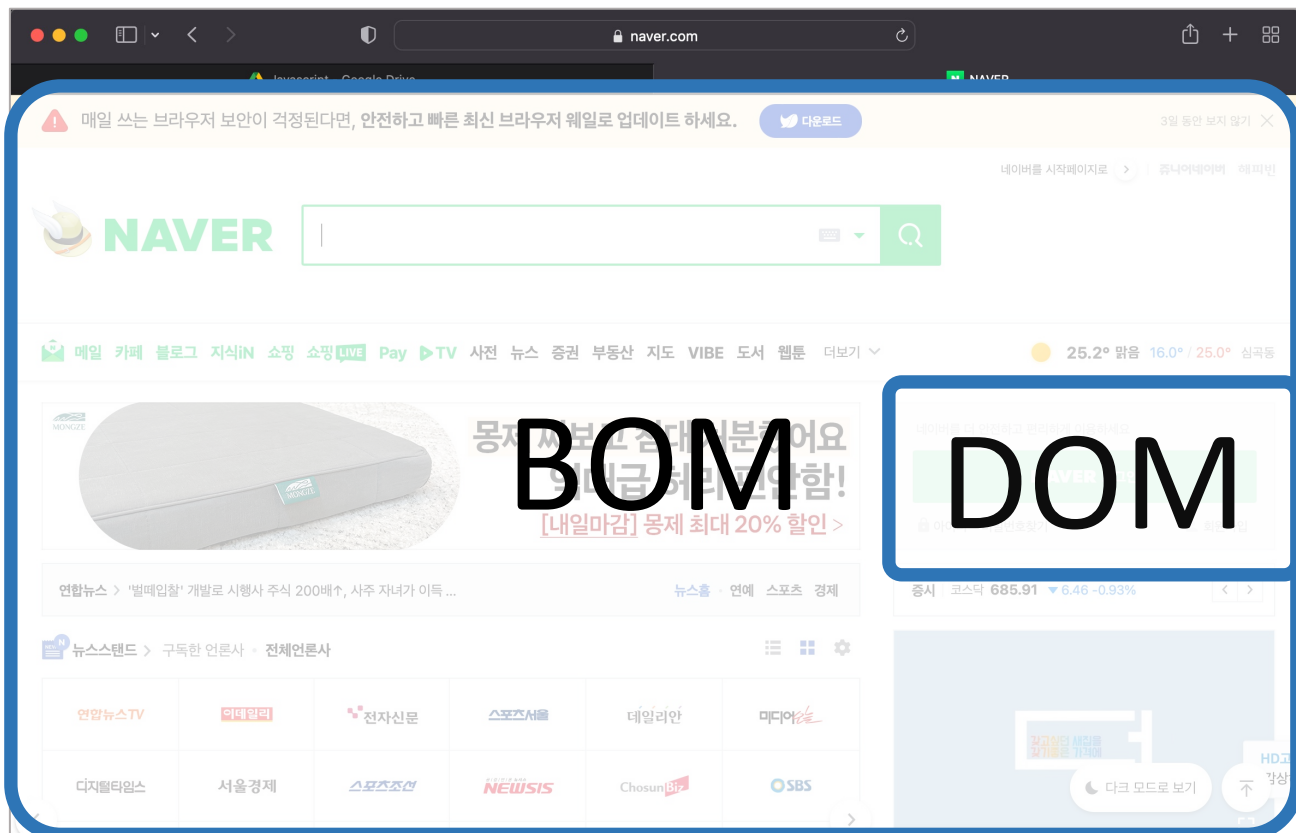
[https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global\\_Objects/Number](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Number)

- JSON 객체
  - 객체처럼 자료를 표현 하는 방식
  - 백엔드에서 데이터를 받을 수 있다(현재 가장 많이 쓰이는 방식)

```
{
  "menu": {
    "id": "file",
    "value": "File",
    "popup": {
      "menuitem": [
        { "value": "New", "onclick": "CreateNewDoc()" },
        { "value": "Open", "onclick": "OpenDoc()" },
        { "value": "Close", "onclick": "CloseDoc()" }
      ]
    }
  }
}
```

# Javascript

## Chaper5

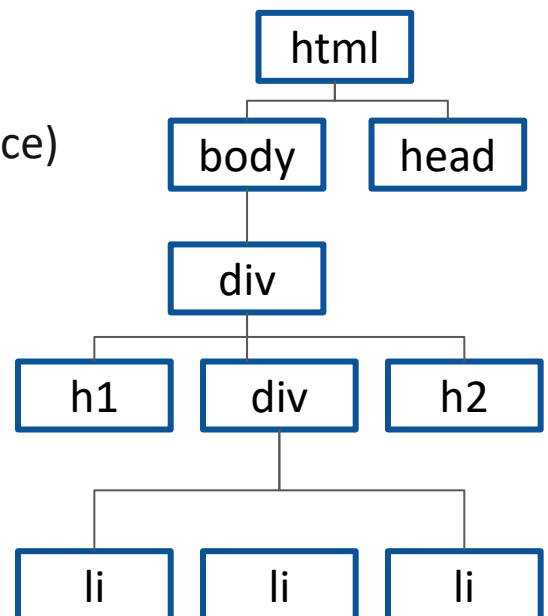




# 문서 객체 모델(DOM)

- DOM이란?

- Document Object Model
- HTML 문서에 접근 하기 위한 인터페이스(interface)
- DOM TREE



- 기본 DOM

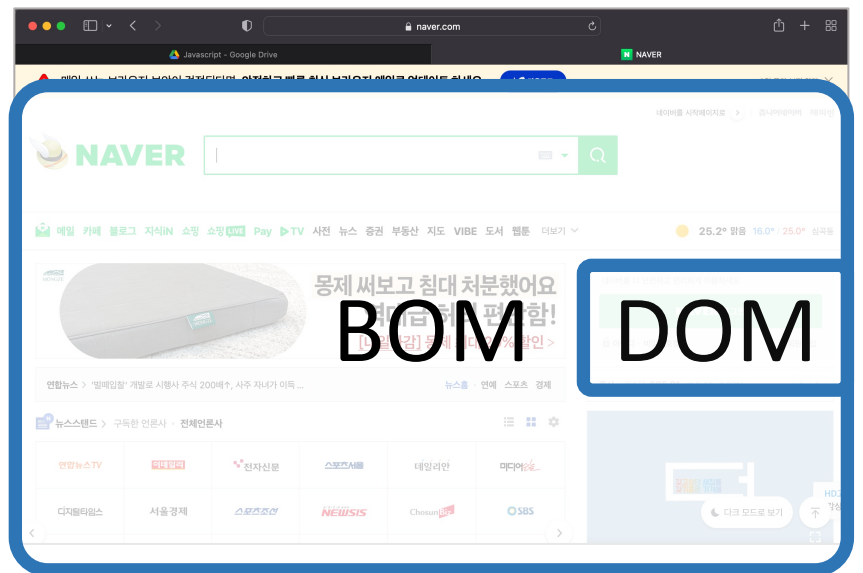
- 페이지 로드
  - DOMContentLoaded
- 요소 선택
  - **querySelector, querySelectorAll**
  - getElementById, getElementsByClassName
- 글자 조작
  - textContent, innerHTML
- 속성 조작
  - setAttribute, getAttribute
- 스타일 조작
  - style
- 태그 조작
  - createElement, appendChild, removeChild
- 이벤트
  - **addEventListener**
  - removeEventListener

- DOM API

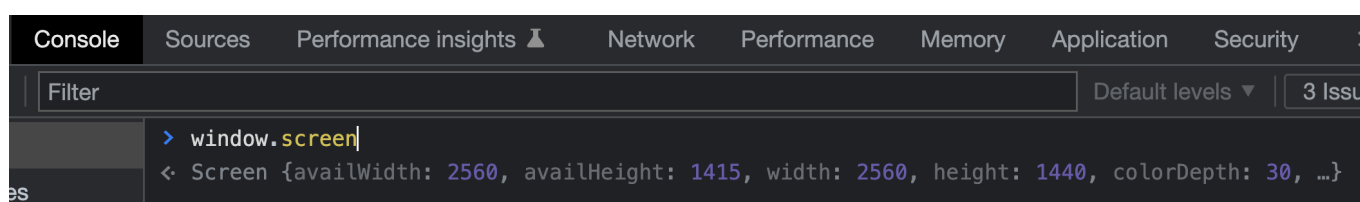
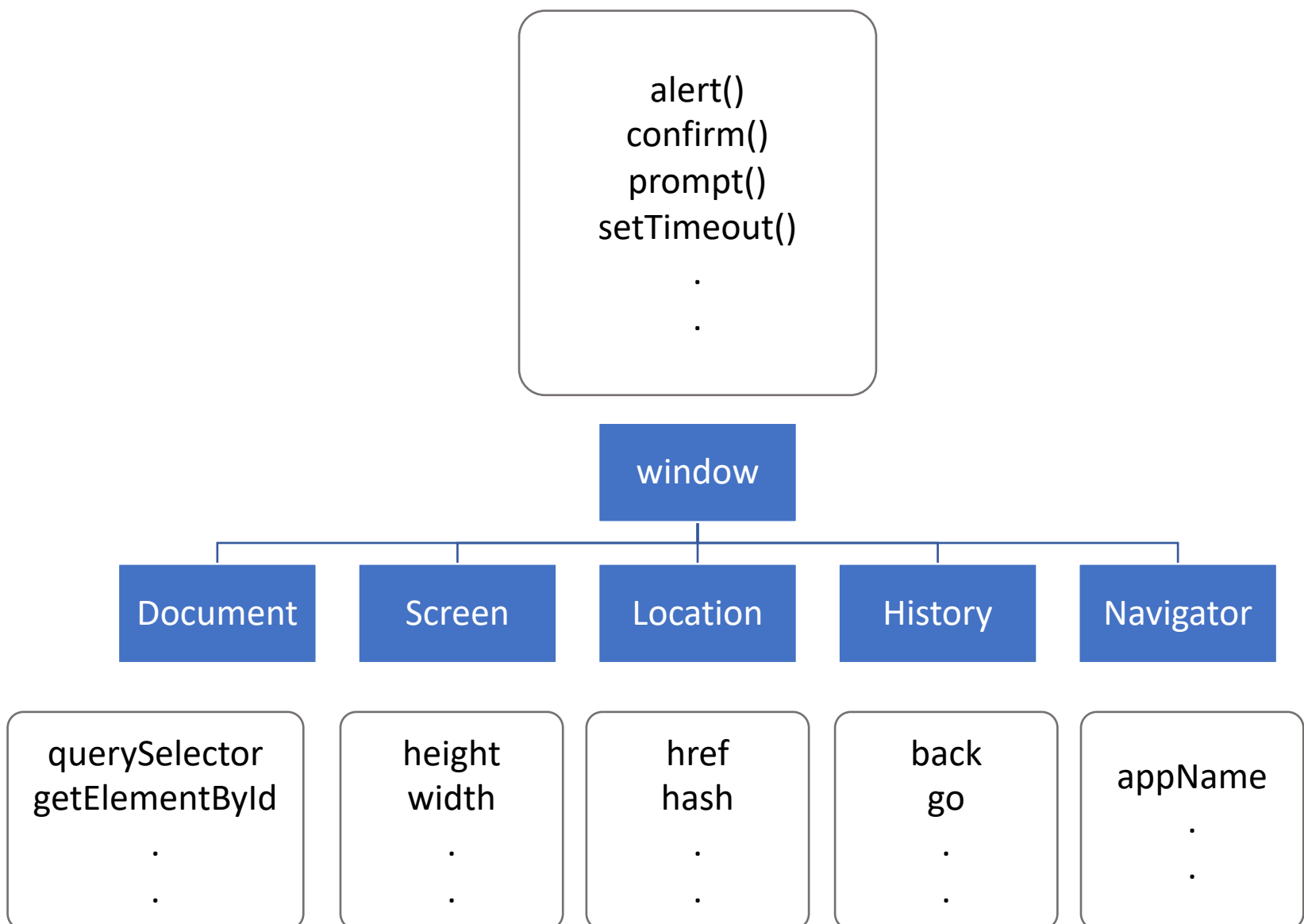
- [https://www.w3schools.com/jsref/dom\\_obj\\_document.asp](https://www.w3schools.com/jsref/dom_obj_document.asp)
- <https://developer.mozilla.org/ko/docs/Web/API/Document>

# 브라우저 객체 모델(BOM)

- BOM이란?
  - Browser Object Model
  - 객체 브라우저에 내장된 객체



- 계층 구조

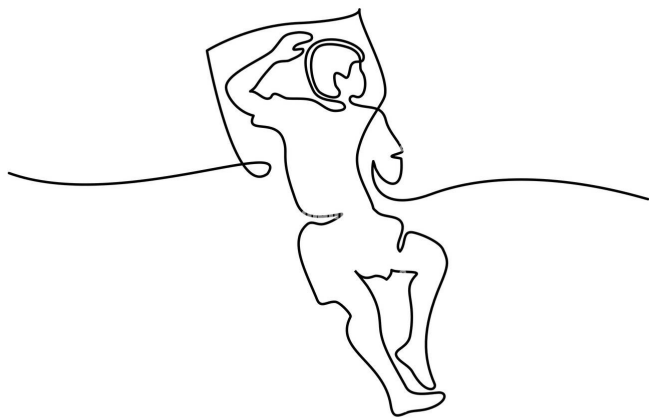


# Javascript

## Chaper6. 클래스

# 클래스

- 객체지향 프로그래밍 언어들은 클래스(class) 라는 문법으로 객체를 효율적이고 안전하게 만들어 줌.
- 객체를 만들기 위한 설계도 혹은 틀.
- 추상화
  - 프로그램에 필요한 요소만 사용해서 객체를 표현하는 것
  - ex) 병원에서 사용하는 업무 프로그램을 만들려면 의사, 간호사, 예약기록 등의 정보만 필요하고 그 중에서 환자 정보를 사용한다면 환자 이름, 생년월일 등만 알면 된다.



이름, 생년월일, 나이, 취미,  
특기, 키, 얼굴둘레, 입의 크기,  
코의 크기, 몸무게....



환자 이름, 생년월일, 성별, 나이,

- 객체? 인스턴스?
  - 거의 같은 말이라고 보면 됨
  - 객체: 구현할 대상(구현 전)
  - 인스턴스: 구현된 구체적 실체(구현 후)

# 클래스

- 클래스

```
class 클래스명 {  
}
```

- 생성자 함수

```
class 클래스명 {  
    constructor() {  
        //생성자 코드  
    }  
}
```

- 메소드

```
class 클래스명 {  
    constructor() {  
        //생성자 코드  
    }  
  
    메소드() {  
        //...  
    }  
}
```

# 클래스

- 상속

```
class 클래스A {  
    constructor() {  
        //생성자 코드  
    }  
  
    메소드() {  
        //...  
    }  
}
```

```
class 클래스B extends 클래스 A {  
    constructor() {  
        super();  
    }  
}
```



클래스A(부모)



클래스B(자식)



- 자식은 부모 것을 가져다 쓴다.
- B클래스는 A클래스에 있는 속성과 메소드를 상속받아 사용한다.
- super()로 부모의 생성자 함수를 호출한다.

# 클래스

- private

```
class 클래스A {  
    #속성명  
  
    constructor() {  
        //생성자 코드  
    }  
  
    메소드() {  
        //...  
    }  
}
```

- getter, setter
  - private로 정의한 속성에 접근 가능하게 만듦.
- static(정적 속성, 정적 메소드)
  - 인스턴스(객체)를 만들지 않고 클래스 이름으로 바로 접근이 가능하다.

클래스 이름.속성  
클래스 이름.메소드()

# 클래스 ETC

- static, prototype, non-prototype 차이

static(정적) 메소드	prototype 메소드	non prototype 메소드
그냥 공유	prototype에서 메모리 공유	메모리 공유X
캡슐화X	캡슐화O	캡슐화O
인스턴스를 생성하지 않고 클래스에서 바로 실행	인스턴스를 생성한 후 실행	인스턴스를 생성한 후 실행
Date.now()	new Date()	new Date()

- 그냥 함수 vs 정적 함수

function	static function
클래스에 포함되지 않음 관리가 어려움	클래스에 가깝게 포함됨 관리가 쉬움



# Javascript

Chaper7. 예외처리


# 예외처리

- 오류의 종류
  - 구문 오류: 괄호의 짝을 맞추지 않던가 문자열을 열었는데 닫지 않았을 때 발생하는 오류
  - 예외(런타임 오류): 실행 중 발생하는 오류
- 기본 예외 처리
  - if문을 사용해 예외가 발생하지 않게 처리
- 고급 예외 처리
  - try ~ catch ~ finally

```
try {  
  
    //예외가 발생할 가능성이 있는 코드  
  
} catch (exception) {  
  
    //예외 발생시 실행 할 코드  
  
} finally {  
  
    //무조건 실행 할 코드  
  
}
```

# 예외처리

- 예외 객체

```
try {  
    //예외가 발생할 가능성이 있는 코드  
} catch (exception) {  예외객체  
    //예외 발생시 실행 할 코드  
}  
finally {  
    //무조건 실행 할 코드  
}
```

- exception.name: 예외이름
- exception.message: 예외 메시지

- 예외 강제처리 발생

```
//단순하게 예외를 발생시킴  
throw 문자열  
  
//좀 더 자세하게 예외를 발생시킴  
throw new Error(문자열)
```

# ES6 주요 문법

- var, const, let
  - arrow function(화살표 함수)
  - map, filter 표기
  - template literal(백틱 사용)
    - 백틱(`) -> 작은 따옴표 x
  - default parameter
  - array and object destructuring
    - 구조화된 배열 또는 객체를 destructuring(비구조화, 파괴) 하여 개별적인 변수에 할당
  - promise
  - promise, fetch
  - async, await
  - rest spread
  - class
- 
- 자바스크립트 버전 별 브라우저 지원 모음
    - <https://kangax.github.io/compat-table/es6/>

# ETC 심화

- 동기, 비동기, 콜백지옥
- 얕은 복사, 깊은 복사
- 커링(currying)
- ajax, fetch, axios 차이