

01

변수 선언과 메모리 구조





학습내용

- 01 프로그램의 실행 원리
- 02 메모리 구조
- 03 변수 선언





학습목표

- 프로그램의 실행 과정을 이해하고, JavaScript 코드가 실행되는 흐름을 설명할 수 있다.
- JavaScript에서 변수와 데이터가 메모리에 어떻게 저장되고 처리되는지 설명할 수 있다.
- var, let, const 키워드를 사용하여 변수를 선언하고, 각각의 특징과 차이를 구분할 수 있다.

지난 주차 복습

1/2

JavaScript의 특징

- 웹 브라우저에서 동작하는 **유일한 프로그래밍 언어**
- 개발자가 별도의 컴파일 작업을 수행하지 않는 **인터프리터 언어**
 - 대부분의 모던 JavaScript 엔진은 인터프리터와 컴파일러의 장점을 결합해 비교적 처리 속도가 느린 인터프리터의 단점 해결
- **명령형, 함수형, 프로토타입 기반 객체지향 프로그래밍**을 지원하는 멀티 패러다임 프로그래밍 언어

지난 주차 복습

2/2

JavaScript의 특징

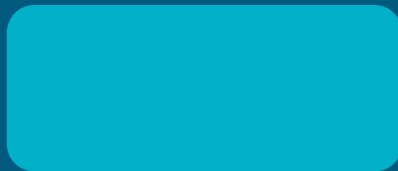
- 클래스 기반이 아닌 **프로토타입 기반의 객체지향 언어**

참고[프로토타입이란?](#)

객체 간의 상속이 아닌
다른 객체를 참조하는
방식

JavaScript

ECMAScript





생각 해보기

Q

ECMAScript는 해마다 새로운 버전을 출시하고 있다.
그 중 가장 혁신적이었다고 생각되는 버전(예 ES6)을
선택해 그 이유를 생각해 봅니다.
그리고 자신의 생각을 게시판에 올려주세요.





01

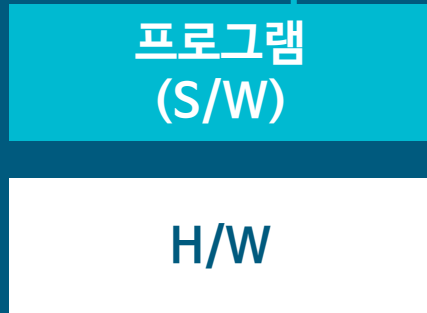


프로그램의 실행 원리



초기 계층화된 모델

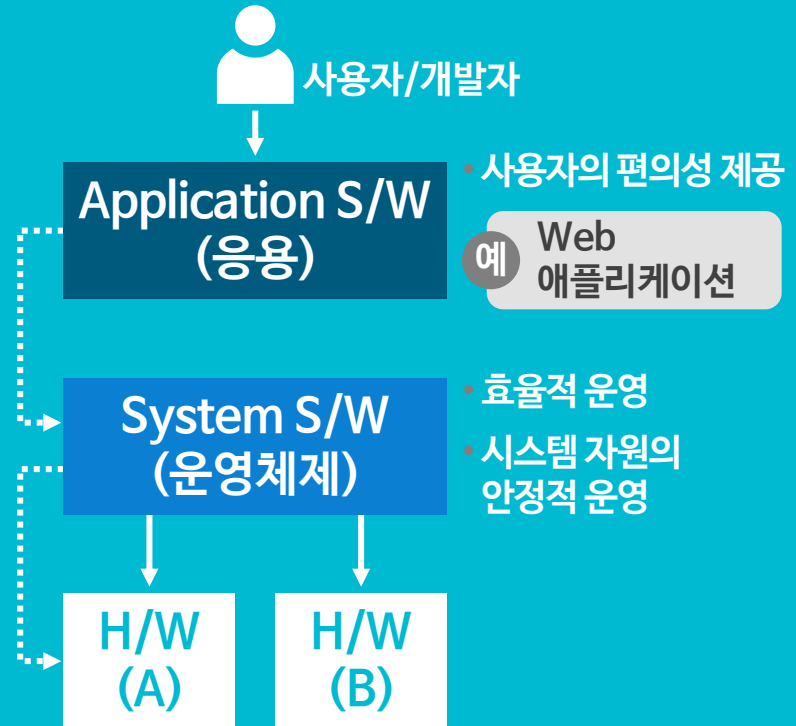
특정 작업을 수행하도록
H/W에 동작을 지시하는
명령어들의 집합



실제 동작을 수행

S/W의
역할 분리

현재 계층화된 모델



1) 컴퓨터의 구성요소



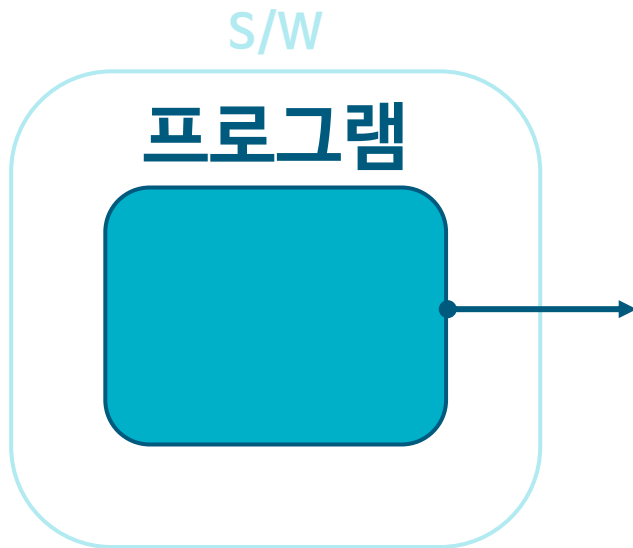
S/W

프로그램



컴퓨터 시스템에서 동작하거나
시스템을 제어하는 **모든 프로그램**,
데이터, 문서, 프로시저 등을
포함하는 **총체적인 개념**

1) 컴퓨터의 구성요소



- S/W의 일종
- 특정 작업을 수행하도록 H/W에 동작을 지시하는 명령어 (Instructions)들의 집합

Q

프로그램을 잘 작성하려면 어떻게 해야 할까?

원하는 산출물을 출력시키기 위해
H/W를 어떻게, 어떤 순서로 나열하여 원하는 결과를 낼 수
있을지를 결정하는 것이 핵심

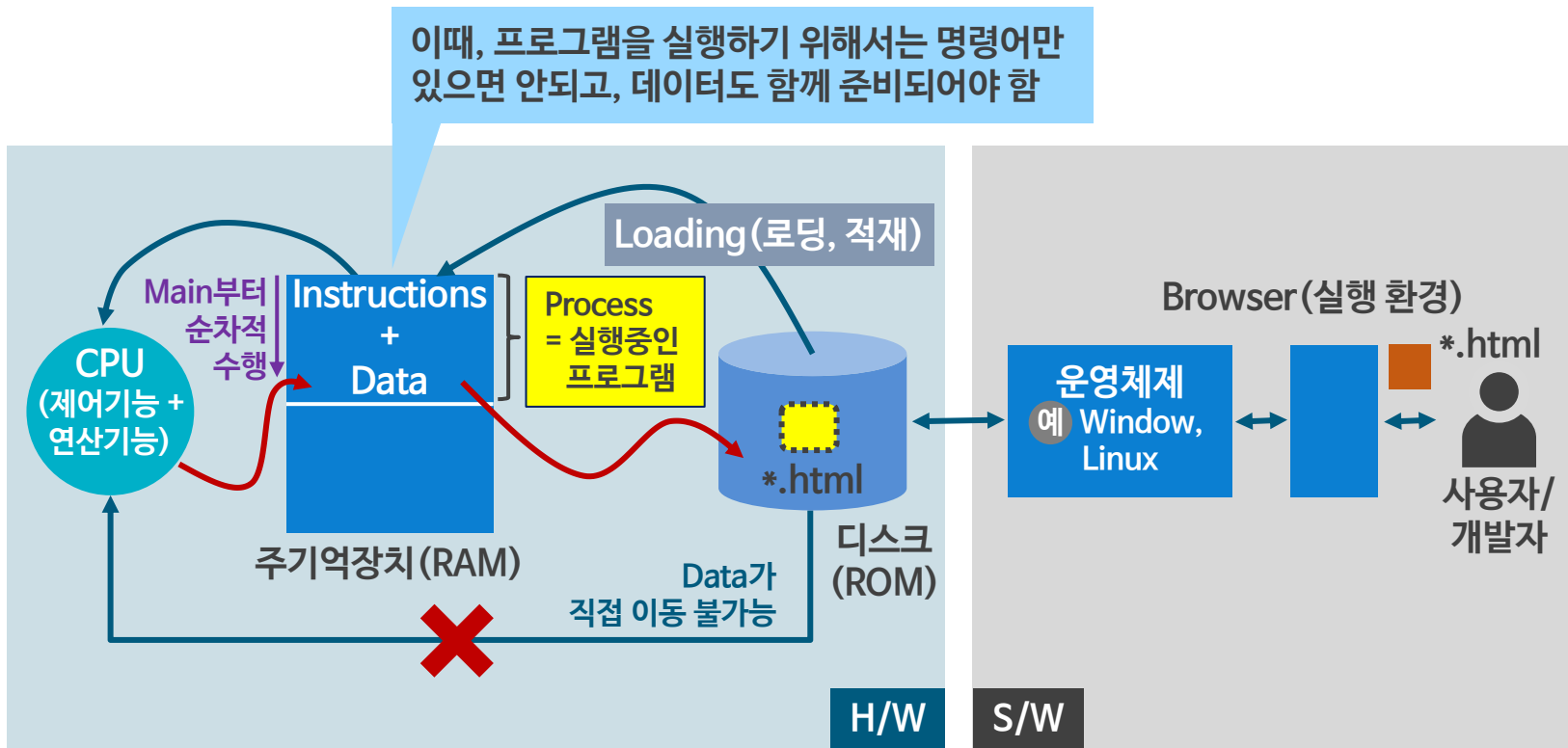


“

H/W에 대한 이해 필요

”

● H/W의 구성 및 프로그래밍 실행 원리



● 프로그램의 구성

프로그램

원하는 산출물을 출력시키기 위해 **특정 작업을 수행하도록 H/W에 동작을 지시하는 명령어 (Instructions)들의 집합**

실행 가능한 프로그램

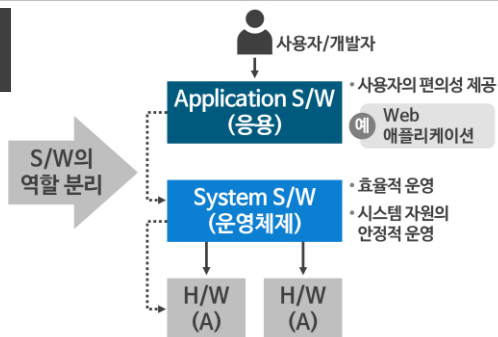
=

명령어들

+

데이터

현재 계층화된 모델



3) 프로그램 작성을 위한 개발자의 준비사항

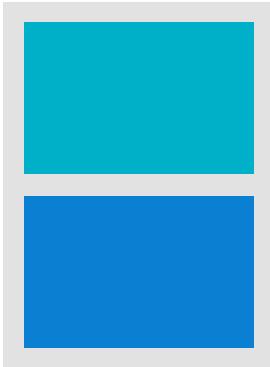


01 데이터 준비하기

➔ 변수

02 원하는 산출물을 내기 위해 필요로 하는 명령어들을 순차적으로 나열하기

➔ 연산자(연산기능), 기타 입출력 명령들(제어기능)



데이터 → $x = 10$ // x, y에 데이터를 담아 준비
 $y = 0$

명령어 → $y = x + 10$ // 데이터를 활용한 명령어



02

메모리 구조



1) 메모리의 구조

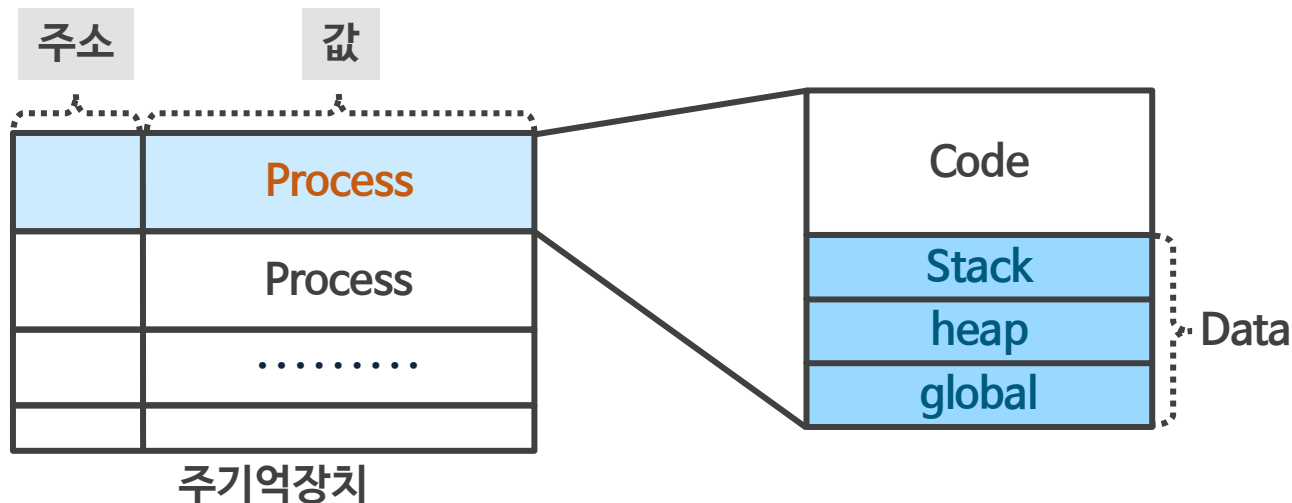


명령어와 데이터를 분리하여 저장

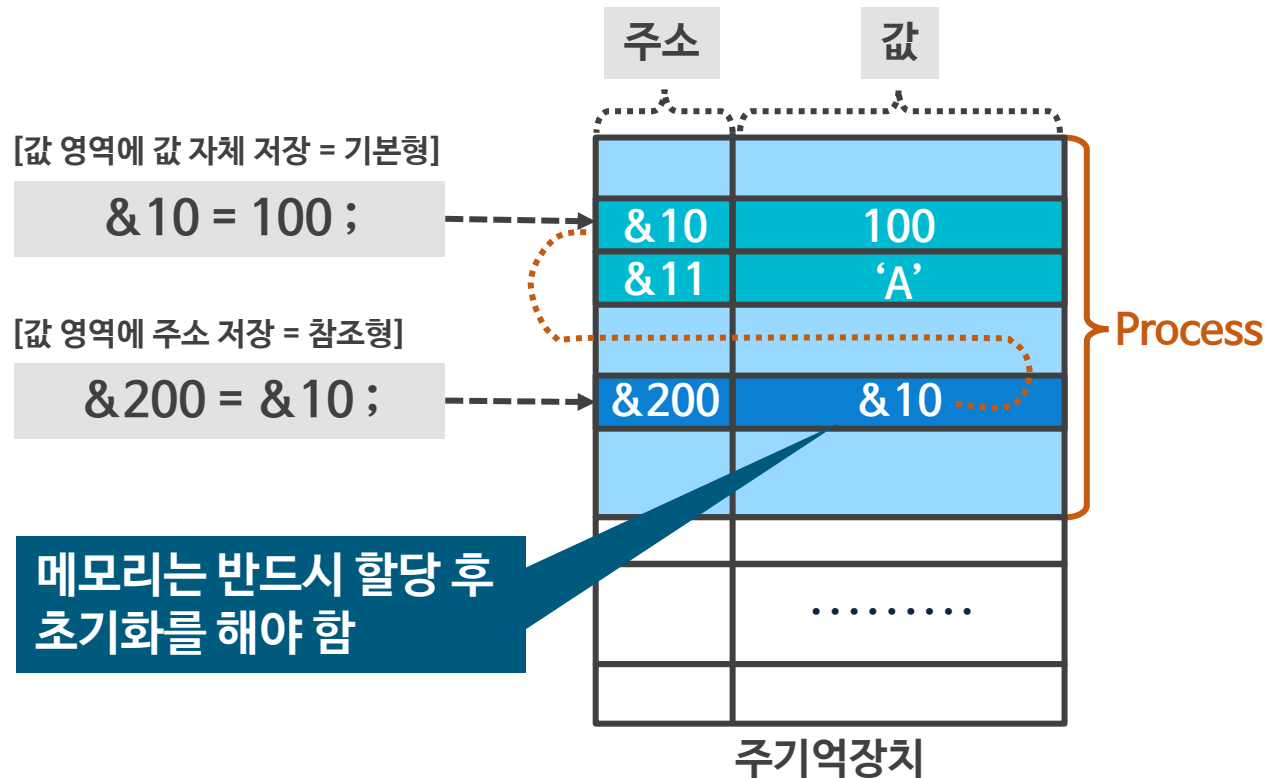


프로그램 작성 순서

- 데이터를 먼저 준비한 후, 데이터에 적용할 명령어 작성



● Data의 저장 원리





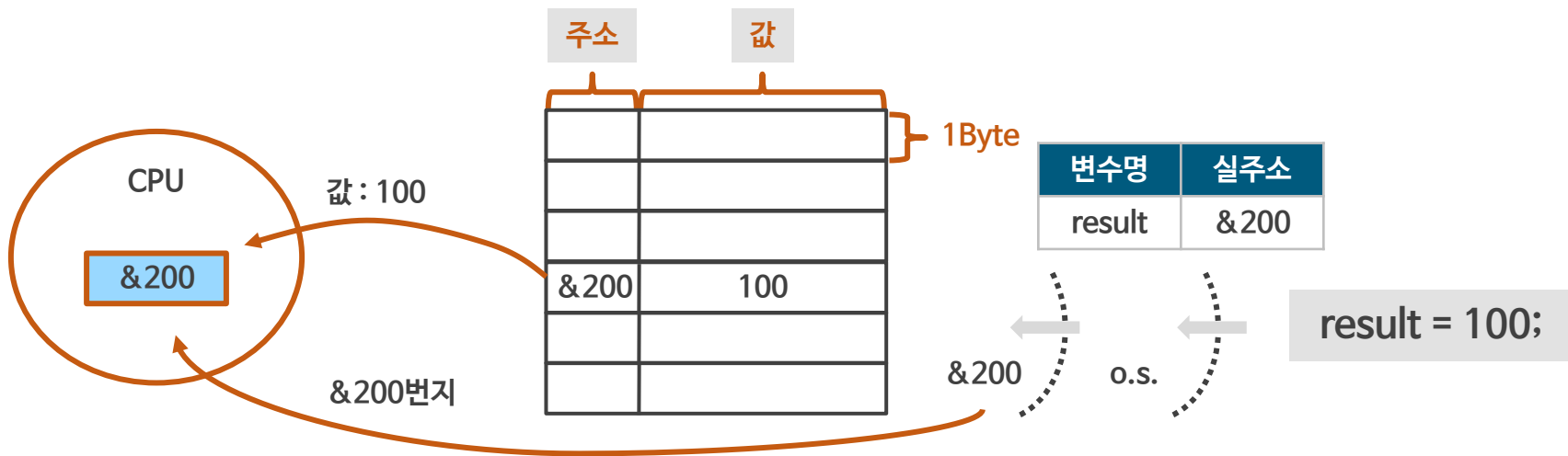
03

변수 선언



변수

- 데이터 준비의 의미
- 값을 저장하기 위한 메모리 공간을 확보하는 것
- 변수 이름과 확보된 메모리 공간의 주소를 연결해서 값을 저장할 수 있게 준비하는 것



2) 변수의 선언



자료형을 선언하지 않아도 되며,
변수를 선언할 때는 **var, let, const** 키워드를 사용함

항목	var	let	const
출시 버전	ES5(기존)	ES6(2015)	ES6(2015)
재선언	가능	불가능	불가능
재할당	가능	가능	불가능
스코프	함수 스코프	블록 스코프	블록 스코프
호이스팅	호이스팅 (초기화 Undefined)	호이스팅되나 TDZ 존재	호이스팅되나 TDZ 존재
초기화 필요 여부	선택사항	선택사항	선언 시 반드시 초기화

※ 호이스팅 : 코드에서 변수를 선언한 위치와 상관없이, 자바스크립트가 실행되기 전에 변수 선언을 코드의 맨 위로 미리 끌어올리는 현상

※ TDZ(Temporal Dead Zone) : 선언 전에 접근하면 에러가 발생하는 영역(let, const만 해당)

2) 변수의 선언



● 예시

var

함수 스코프

블록 밖에서도 사용 가능

let

블록 스코프

블록 안에서만 사용 가능

```
function scopeTest() {  
  if (true) { //블록  
    var x = 1;  
    let y = 2;  
  }  
  console.log("var x:", x); // 1 (함수 스코프)  
  // console.log("let y:", y); // 에러! (블록 스코프)  
}  
scopeTest();
```

- 호이스팅 (Hoisting)이란?

```
console.log(x); // undefined  
var x = 5;
```



자바스크립트는
위 코드를 다음처럼 해석함

```
var x; //선언만 끌어올려지고 값은 undefined  
  
console.log(x); // undefined  
x = 5;
```

→ var x가 자동으로 코드 맨 위로 옮겨진 것처럼 동작함

이게 바로 ‘호이스팅 = 선언 끌어올림’이라고 하는 이유임

● 호이스팅 (Hoisting)이란?

```
console.log(x); // Reference Error : Cannot access 'x' before initialization  
let x = 5;
```

- let/const도 선언 자체는 호이스팅되지만, 초기화 전까지 TDZ(잠깐 접근하면 안 되는 시간, 준비가 안 된 상태)에 묶여 접근할 수 없음
- ReferenceError가 발생하며, 이는 var의 “암묵적 undefined”와 큰 차이를 만듦

let·const 변수의 호이스팅 - “TDZ(Temporal Dead Zone)”에 갇힘

● 예시

{ 변수를 선언할 때는 var, let, const 키워드를 사용함 }

```
// var 예시
var x = 10;
var x = 20;    // 재선언 가능
console.log(x); // 20

// let 예시
let y = 5;
// let y = 8;    // ✗ 재선언 불가 ➡ 에러
y = 8;          // 재할당은 가능
console.log(y); // 8

// const 예시
const z = 15;
// z = 20;        // ✗ 재할당 불가 ➡ 에러
// const z = 30;   // ✗ 재선언도 불가
console.log(z);   // 15
```


02주. 데이터 준비와 연산 명령어로 기본 프로그램 만들기

02 자료형

