

FE:SSION

6주만에 [프론트엔드 개발자](#)로 성장하기 !

WEEK 01 : 2023. 05.02

Lead. 이준규





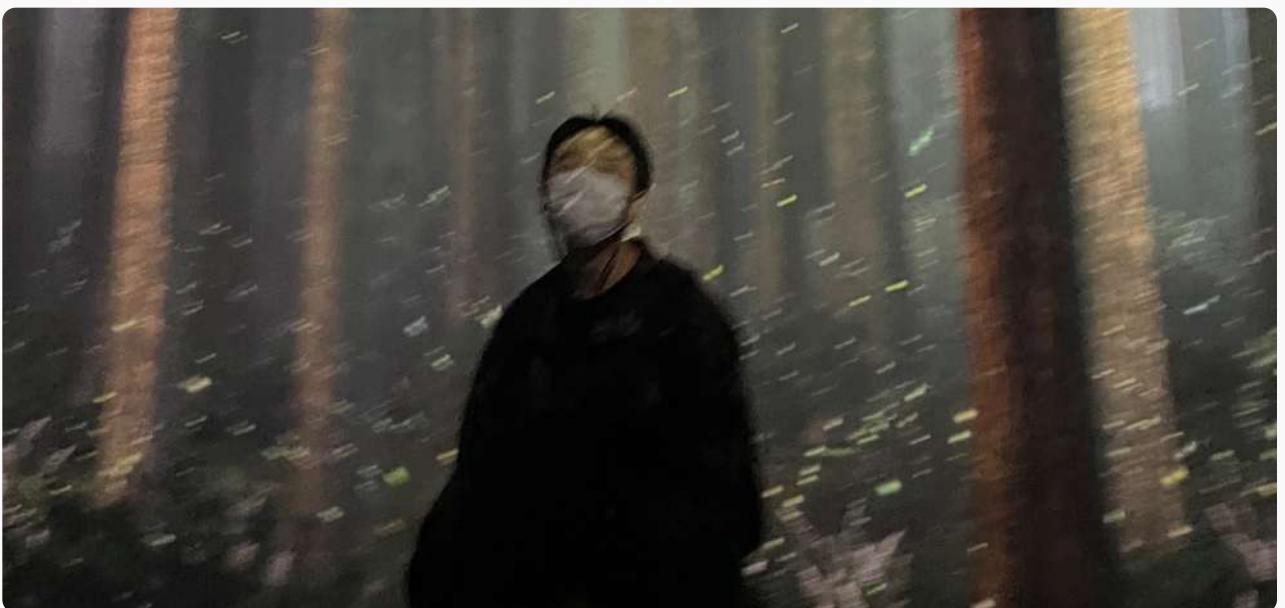
**FESSION
WEEK 01
의욕 강화 및 JS 기초 훈련**

- 01** Getting Started!
- 02** 프론트엔드 개론
- 03** 프론트엔드 주변 개념
- 04** 개발 환경 세팅
- 05** 변수 (변수 / 식별자 / 호이스팅 / 값 할당 / 네이밍 규칙 / ...)
- 06** 연산자 (산술 / 할당 / 비교 / 삼항 / 논리 / ...)
- 07** 제어문 (블록문 / 조건문 / 반복문 / ...)
- 08** Q&A + 과제 공지

6주 동안 함께 성장할,

이준규입니다.

WEB FE & Cross Platform DEVELOPER



한국 IT정책경영학회 학술대회 우수상

2019 유니브 엑스포 우수상

한국전력공사 서포터즈 8기

AI 양재 허브 서포터즈 1기

Nexon MSW Supporters Hackthon

SK Telecom DEVOCEAN YOUNG 2기

AI 양재 허브 우수 서포터즈 상

대학생 연합 IT 동아리 잇츠타임 Web Front-End part member

숭실대학교 멋쟁이사자처럼 10기 Front-End part member

숭실대학교 멋쟁이사자처럼 11기 Front-End part leader

Google Developer Student Clubs 2nd Web/Mobile Member

IT 원스톱 스타트업 SWYG 파트너십

대학생 연합 IT 동아리 잇츠타임 팀 MVP 상

Ablind(플랫폼 개발팀) UI/UX 디자인 및 웹 프론트엔드 개발

건국대학교 캠퍼스타운 2022 K-이노스 캠퍼스타운 우수상

숭실대학교 슈퍼스타 창업경진대회 우수상

Malaysia Multimedia University 해외 교육 프로그램 수료

(주) 블록웨이브랩스 : WEB3.0 스마트 컨트랙 기반 웹 프론트엔드 개발

Adobe Certified Associate Visual Design Specialist (국제인증자격)

데이터분석 준전문가 (ADsP) (국가공인자격)

SQL 개발자 (SQLD) (국가공인자격)

6주 완성, 단기간에 빠르게 성장하고, 해커톤 씹어먹기!

예로부터 숭먼사가 중앙 일짱이었다~ 이말이야...

자바스크립트 기초(1)

자바스크립트라는 언어와 기초 문법

자바스크립트 기초(2)

객체 & 함수와 클래스, 이벤트 핸들링

React + Data Fetching

React 기본과 데이터 패칭

고급 스타일링

가상 클래스 / 선택자 + 애니메이션 / 반응형

나만의 일정 관리 서비스 퍼블리싱

React 기반 일정 관리 서비스 퍼블리싱

나만의 일정 관리 서비스 API 연동

React 기반 일정 관리 서비스 API 연동

FE:SSION 규칙

모르는 부분이 있다면 언제든 자유롭게 공유하기!

파트장은 선생님이 아닌, 조장 정도로 생각해주세요!! (제발)

세션은 가능한 빠지지 말고 참석하기!

여러분의 설문조사를 바탕으로... 정말 꽉꽉 눌러담은 세션이에요.

한 주라도 빠지면 타격이 클지도...

매 주 과제는 필수로!

chatGPT, Blog Driven 등 전부 가능!

어쨌든 무조건 해야해요!

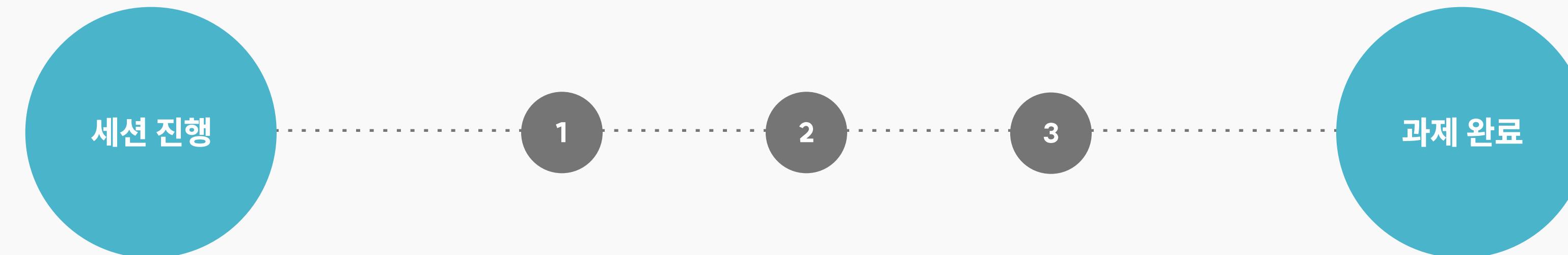
자만하지 않기!

FE:SSION이 진행될수록 빠르게 성장하는 내 모습을 보면, 쉽게 자만할 수 있어요.

하지만 자만은 성장하는 개발자에게 있어서 가장 큰 독!



FE:SSION 과제 플로우



- 1 **매주 고정 과제 + 주차 과제를 직접 완성하기!**
chatGPT나 검색 엔진을 잘 사용하는 것도 훌륭한 능력이니까, 자유롭게 활용해도 좋아요!
- 2 **깃허브에 1차적으로 과제 제출하기!**
다음 세션이 진행되기 전까지, 완성한 과제를 깃허브에 모두 제출해주세요!
이후 일주일 동안 파트장의 코드리뷰가 진행됩니다.
- 3 **코드 리뷰 확인하고, 코드 개선하기!**
해당 주차의 새로운 과제를 진행하던 중, 코드 리뷰 완료 알람을 받으면 꼼꼼히 확인하고 적용해주세요!
지난 과제의 코드를 개선하는 과정에서 복습까지 할 수 있어요.

FE:SSION 고정 과제



1주차 고정과제

당근마켓 랜딩 페이지 클론

당근마켓에 접속했을 때 최초로 접하는 랜딩 페이지를 클론하세요!
페이지 라우팅(연결)을 하거나 실제 로직을 구현하는 것이 아닌,
하나의 페이지를 단순히 HTML과 CSS만으로 클론하면 돼요.

[당근마켓 랜딩 페이지로 이동](#)

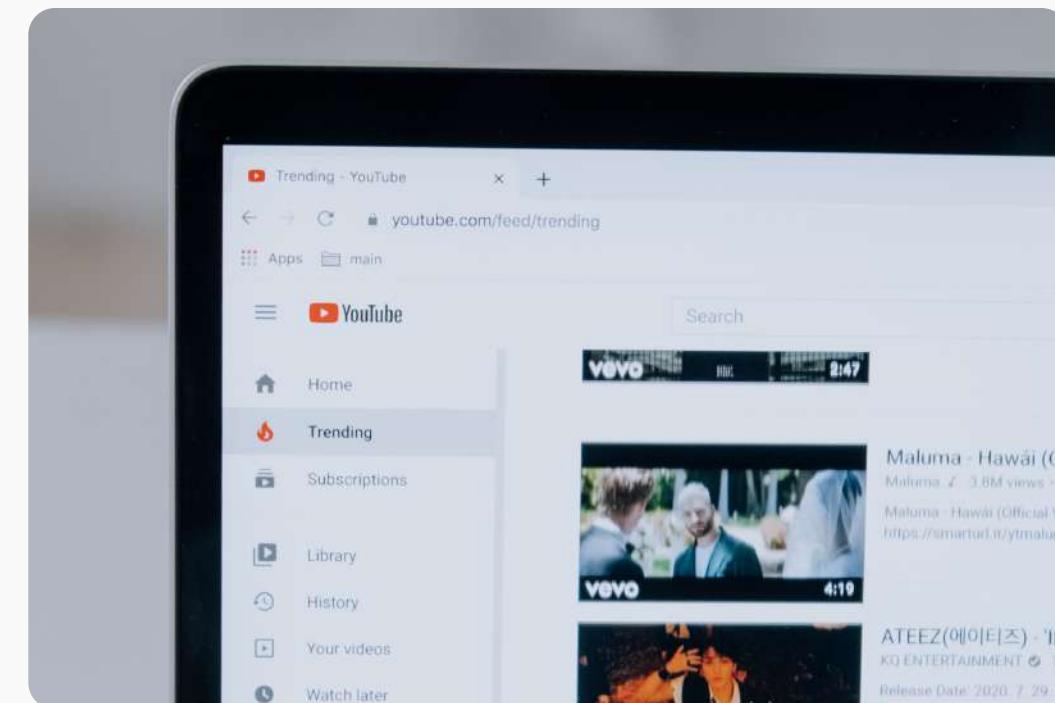


2주차 고정과제

인스타그램 홈 피드 클론

인스타그램에 접속했을 때 최초로 접하는 홈 피드 페이지를 클론하세요!
데이터는 본인이 원하는 데이터를 추출하여 담고,
하나의 페이지를 단순히 HTML과 CSS만으로 클론하면 돼요.

[인스타그램 홈 피드 페이지로 이동](#)



3주차 고정과제

유튜브 랜딩 페이지 클론

유튜브에 접속했을 때 최초로 접하는 홈 피드 페이지를 클론하세요!
영상이 아닌 이미지로 구성하세요!
하나의 페이지를 단순히 HTML과 CSS만으로 클론하면 돼요.

[유튜브 랜딩 페이지로 이동](#)

FESSION이 끝나고, 여러분이 나아가야 할 방향은?

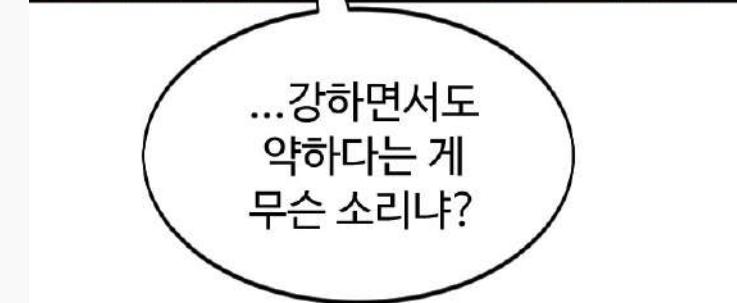
빠르게 세운 탑은 쉽게 무너집니다~ 초심으로 돌아가세요~



...강하면서도
약하다는 게
무슨 소리냐?



...?



그 나이에
배워야 할 것에 비해
너무 많이 배웠어.



많이 배우고 익히면
좋은 거 아니냐?



무학이란
탑을 쌓는 것과
마찬가지야.



그런데 쟤들은
일 층을 채 다 짓기도 전에
이 층, 삼 층을 쌓아 올렸어.



그렇게 쌓아 올려진 탑이
일 층을 완벽하게 지은 탑과
부딪치면 어떻게 될 것 같아?



...무너지겠지.

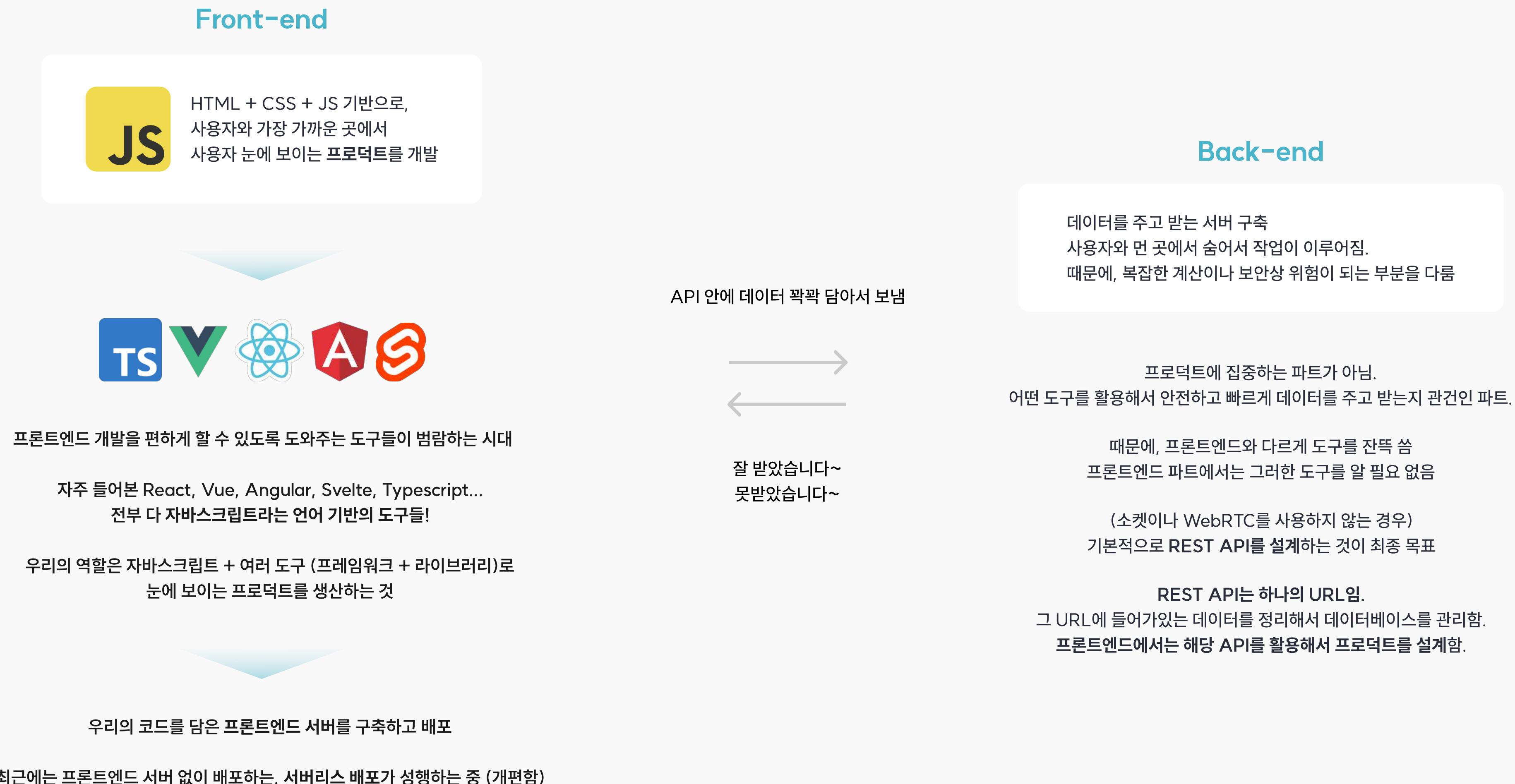
그래.

사실 저도 일 층 다 쌓기 전에 100층 까지 쌓은 사람이라.. 할 말은 없지만
솔직히 지금 개발 생태계에선 빠르게 쌓는 것도 어느정도 중요하기 때문에, 너무 기초에만 묶여있는 것도 좋지 않다고 봐요.
저랑은 한 10층까지 빠르게 쌓아 봅시다! 그만큼 쌓으면 '[1층을 다시 쌓아야겠다~](#)'라는 생각이 들거예요.
그때 다시 초심으로 돌아가서, 1층부터 단단히 보수 공사 하면 됩니다! 노베에서 하는 것 보다 훨씬 재밌고, 수월할거예요.



예..에...
빡센거 알지만.....
단기간에 당신들을 프론트엔드 개발자로 성장시
키려면 어쩔수없습니다
화이팅 ㅎㅎ..
그래도 이거 다 따라오면 나름.....나도 어디서 꿀리진 않아..복창가능





우리가 6주 동안 배우게 될 것은, 자바스크립트 언어의 기초 + React

둘 다 HOT한 이유가 있죠, 일단 쉽습니다... 예

장점

입문하기에 너무 쉽다. 진짜 정말 쉽다.

괜히 프론트엔드 개발자 4주 양성~ 이런 광고 있는거 아님. 근데 그거 다 구라임
(FESSION은... 끝난 이후에 여러분이 더 공부해야 찐 개발자 될 수 있다고 미리 말씀드릴게요....)

단점

제대로 활용하기 위해 조금이라도 덥스가 깊어지면,

그 때부터 흰머리 남

타 개발 파트보다 훨씬 지독하고 어렵고... 위험천만한 파트
약한자는 살아남을 수 없습니다..



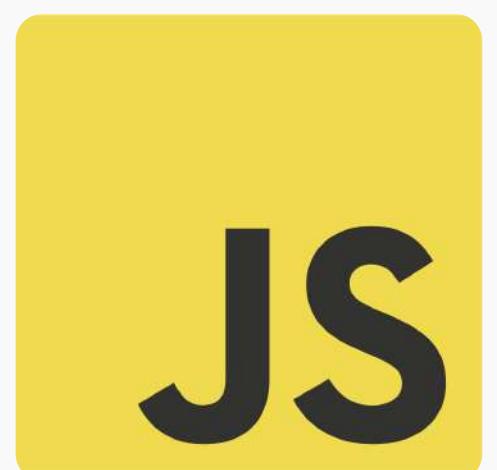
Javascript란?

HTML과 CSS로 구성된 스크립트에 여러가지 로직을 부여할 수 있는 언어.

자바스크립트 === WEB이 아님을 기억해야 한다.

단순히 언어이기 때문에, Backend · ML · App 등에서도 사용된다.

우리는 이 언어로 웹을 만들 뿐!



Javascript

ECMAScript / ES6

자바스크립트는 꽤 오래된 언어임.

Mocha라는 이름으로 시작해서 과도기 시절에 다양한 형태의 Javascript가 사용되었음

가령 1996년에 MS에서는 JScript를 사용하기도 함.

이른 바 웹 춘추 전국시대... 웹 표준화 기관 ECMA에서는 Javascript의 파편화를 막고자 ECMAScript를 만듦

그게 지금의 표준 Javascript가 되었고, 우리가 흔히 말하는 자바스크립트는 ES6(2015) 이상의 버전임

Interpreter

컴파일러(Compiler)는, C언어나 C++등의 High Level 프로그래밍 언어의 전체를 스캔해서 기계어로 변환하는 번역기.

하지만 Javascript는 컴파일러가 아닌 인터프리터를 기반으로 동작함

인터프리터(Interpreter)란, 마찬가지로 High Level 프로그래밍 언어를 기계어로 번역하지만 한 번에 한 문장씩 번역함.

아주아주 요약한 내용이므로, 더 궁금하다면 검색 ㄱㄱ.

요는 Javascript는 인터프리터 기반의 언어이다! 라는 것.

Babel

Babel은 Javascript의 컴파일러(Compiler)이다.

엥 JS는 인터프리터 기반의 언어인데..? 라는 생각이 들었다면 당신은 졸지 않고 있군요 굿.

앞서 언급했듯 JS는 버전이 빠르게 변화하고 있음. ECMAScript 이후 파편화는 없지만, 버전 차이에 따라 문법이나 기능이 달라짐.

여러 버전에서 사용된 JS를 최신 버전으로 변환하는 트랜스컴파일러(Transcompiler)가 바로 바벨(Babel)



Javascript

SPA / CSR / SSR

SPA는 Single Page Application => 말 그대로 하나의 HTML 안에서 모든 페이지를 구성하는 웹

CSR은 Client Side Rendering => 렌더링이란, 우리의 코드를 화면에 그리는 것. CSR은 말 그대로 모든 렌더링을 사용자의 브라우저에서 진행하겠다는 것

SSR은 Server Side Rendering => 말 그대로 렌더링을 (프론트엔드) 서버에서 먼저 진행하겠다는 것

React가 CSR 기반의 CSR 프레임워크였으나... 프론트엔드 생태계의 급변으로 인해 요즘 트렌드는 SSR 기반에 다른 여러 기능을 추가하는...것....
도망가지마세요

V8 Engine

본격적으로 개발을 마치고 브라우저에서 렌더링 시킬 때, 우리의 코드를 실행해주는 엔진이 V8 엔진.

모든 브라우저 안에 탑재 되어있는 오픈소스.

기본적으로 웹에서 작동하는 코드는 Node 환경에서도 잘 작동하나, 브라우저에서만 사용할 수 있는 객체(e.g. window 객체 등)는 사용할 수 없음

Node.js

방금 듣다가 노드는 또 뭐야 했다면 당신은 졸지 않고 있군요 굿.
브라우저는 V8 엔진을 통해서 자바스크립트를 실행함.
브라우저 외부에서 JS를 실행할 수 있는 환경을 만들어주는 것이 바로 Node.js (때문에 이걸로 백엔드 구축 가능)
어이 백엔드로 도망가지마



Node.js

패키지매니저 : npm / yarn

Node 환경에서 제작된 패키지를 안전하게 설치하고 관리해주는 도구임.
언어별로 패키지 매니저가 있으나, Javascript에서는 npm이 가장 많이 쓰이고 그 다음이 yarn. 요즘에는 pnpm도 있음.
웬만하면 yarn 쓰세요 제발 의존성 관리라고 아주 골치 아픈 녀석이 있는데 그걸 잘 해결해주는 고양이임.



TypeScript

MS에서 관리하는 자바스크립트의 슈퍼셋 프로그래밍 언어임
배워보면 알겠지만 자바스크립트는 너무나..너무나 자유로운 언어이기 때문에 오히려 화가 잔뜩 날 때가 있음.
내가 잘못했다면 알려달라고!! 라고 외치는 단점을 해결해줌. 방학 때 스터디로 공부하세요
단점) 한 번 맛보면 자바스크립트로 못돌아감

Node.js 설치

미리 다 설치 해왔죠?
버전 확인만 한 번 해봅시다!

```
› node -v  
v18.14.2
```



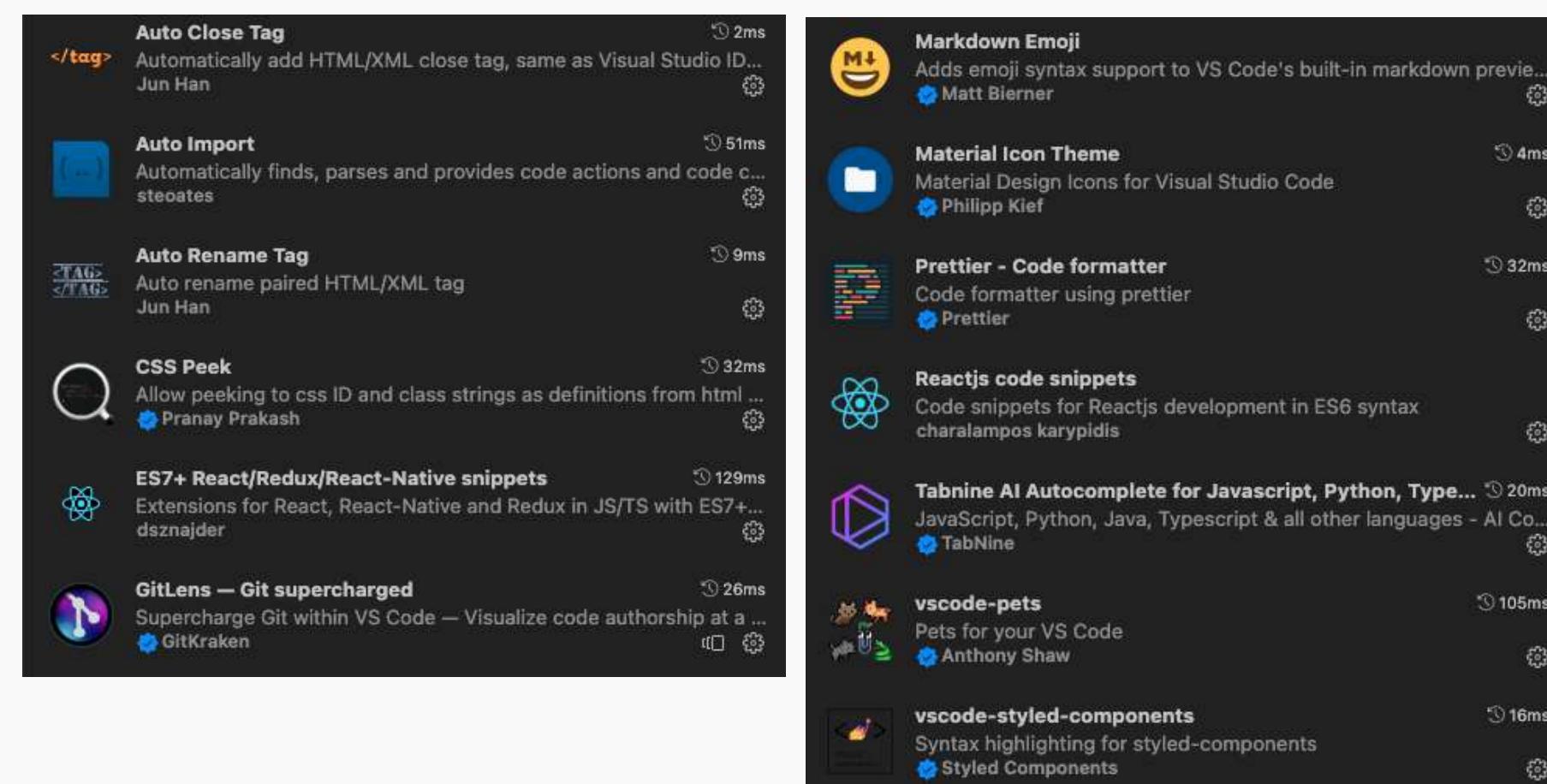
Node.js

사실 지금 쓸 일은 없는데 로고가 귀여우니까 다운 받으세요. 나중에 무조건 쓸 뻔 (yarn 설치까지 해줬는데 npm 쓰기만 해봐 쪽)

```
$ npm install -g yarn
```

필수 익스텐션 설치

프론트엔드 개발하면서 필수적인 익스텐션을 설치해봅시다!



이거로 미쳐갈 쿄 쿄



10분 쉬는시간 그 7

변수 선언

변수를 생성하는 행위를 변수 선언이라고 함

자바스크립트의 변수

변수란, 기본적으로 프로그래밍 언어에서 데이터를 관리하기 위한 핵심 개념

: 어떤 값을 저장하기 위한 공간 자체에 이름을 붙인 것

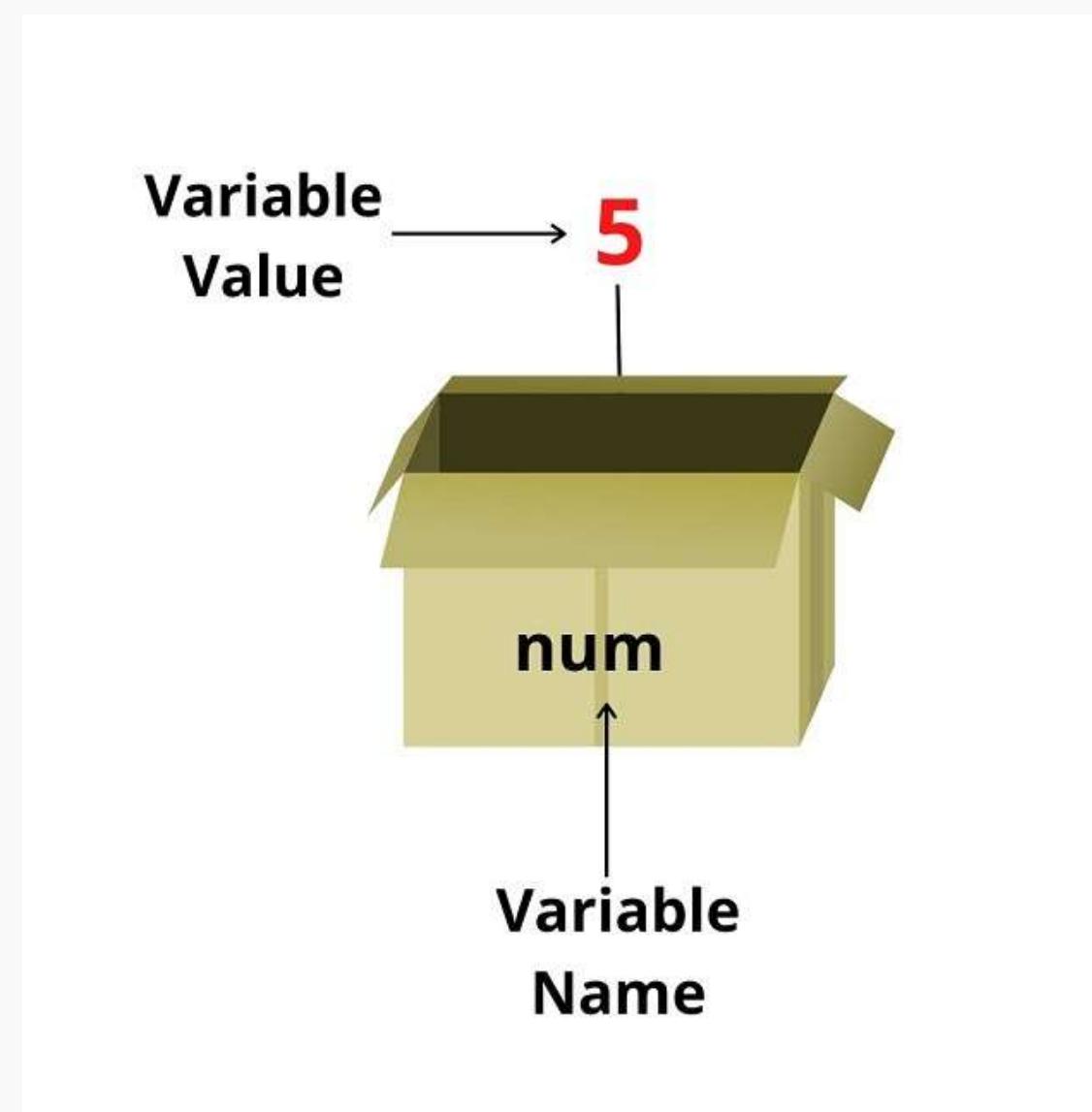
→ 이렇게 붙여진 이름을 **식별자**라고 함.

```
1 //변수 선언
2 const firstName = "준규";
3 let lastName = "이";
4
5 console.log(firstName, lastName);
```

잠깐! `console.log`는 무엇인가요?

: 자바스크립트에서 값을 콘솔창에서 확인하고 싶을 때 사용하는 메서드예요!

```
> node index.js
준규 이
```



const

상수 변수를 선언할 때 사용함. 즉, `const`로 생성된 변수는 불변의 값.

```
//변수 선언
const firstName = "준규";
let lastName = "이";

console.log(firstName, lastName);

//변수 변경
firstName = "중규"; //Error!
console.log(firstName, lastName);
```

TypeError: Assignment to constant variable.
at Object.<anonymous> (/Users/mac/Desktop/Development/likelion/Frontend/Week01/index.js:8:11)

const로 생성된 값은 불변이다!

```
//변수 선언
const firstName = "준규";
let lastName = "이";

console.log(firstName, lastName);

//변수 변경
// firstName = "중규"; //Error!
lastName = "김";
console.log(firstName, lastName);
```

```
> node index.js
준규 이
준규 김
```

let

값을 재할당 할 수 있는 변수

변수 선언의 단계

변수 선언은 런타임(소스코드가 한 줄씩 순차적으로 실행되는 시점)이 아닌, 그 이전 단계에서 먼저 실행됨.

선언 단계 : 변수 이름을 등록해서 자바스크립트 엔진에게 해당 변수의 존재를 알려줌

초기화 단계 : 값을 저장하기 위한 메모리 공간을 확보하고, 암묵적으로 `undefined`를 할당해서 초기화하는 단계

할당 단계 : `undefined`로 초기화 된 변수에 실제 값을 할당하는 단계

→ 뒤에서 타입과 함께 알려드림!

var

`const`와 `let`은 ES6 이후에 등장한 키워드임.

`var`은 `const`와 `let`을 짬뽕해서 사용하던 키워드인데, 공식적으로 사용하는 것을 권장하지 않고 있음. 왜일까?

실험실

변수 선언과 관련된 테스트를 진행해봅시다.

값이 선언되기 전에 접근했다는 에러가 발생함.

```
1 console.log(firstName, lastName);
2 //변수 선언
3 const firstName = "준규";
4 let lastName = "이";
```

```
> node index.js
/Users/mac/Desktop/Development/likelion/Frontend/Week01/index.js:1
console.log(firstName, lastName);
^

ReferenceError: Cannot access 'firstName' before initialization
    at Object.<anonymous> (/Users/mac/Desktop/Development/likelion/Frontend/Week01/index.js:1:13)
```

```
1 console.log(firstName, lastName);
2 //변수 선언
3 var firstName = "준규";
4 var lastName = "이";
```

```
실행이... 되어버린다?!
Node.js v18.14.2
> node index.js
undefined undefined
```

호이스팅이라는 개념과 연관이 되어있음.

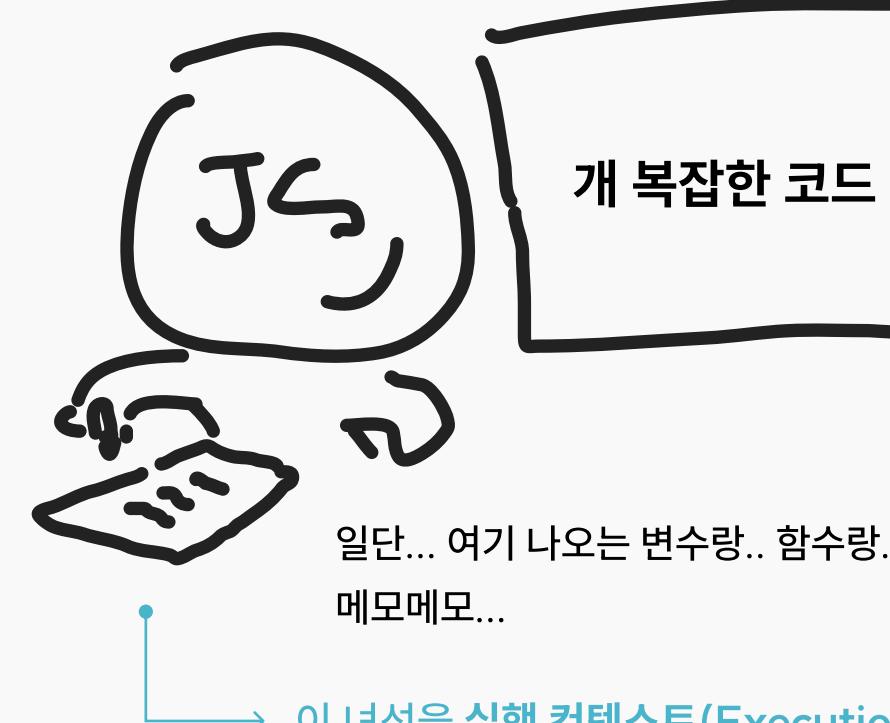
호이스팅

변수나 함수의 선언문이 스코프의 최상단으로 끌어 올려진 것처럼 동작하는 자바스크립트 고유의 특징.

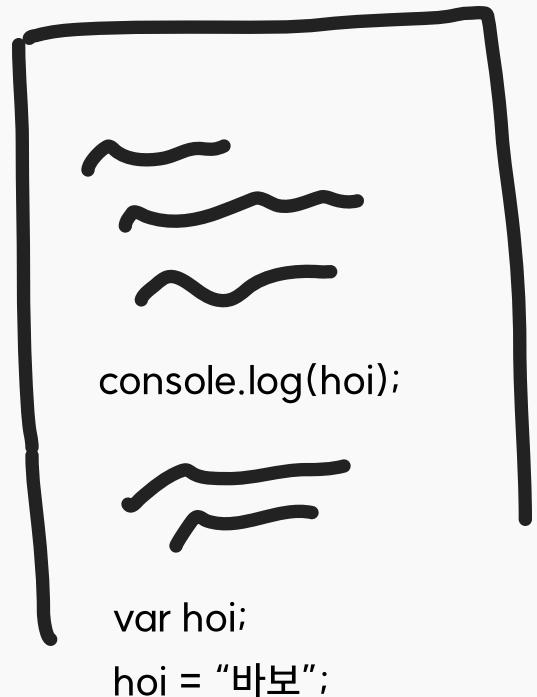
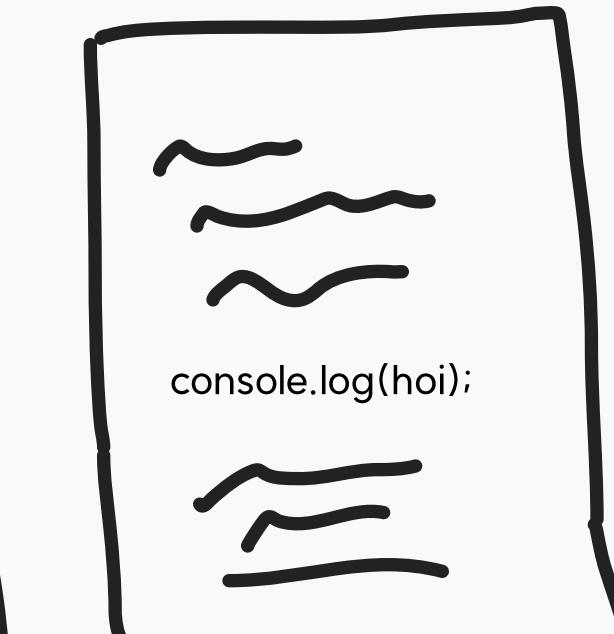
→ 값에 접근할 수 있는 범위! (전역 + 지역)

자, 너무 무서워하지 말고 천천히 생각해봅시다!

자바스크립트 엔진의 동작 과정



오 나 애 암!!!!!
내가 메모해놓은거에 있음!!!!
이 코드에서 나옴!!!!!!1!111111
일단 내 노트에는
`undefined`라고 적혀있음!!



중요한 것은 자바스크립트에서는 코드를 실행할 때 선언된 변수가 모두 위로 끌어올려진 것처럼 동작하는 호이스팅이 일어난다는 것.

하지만, `let const class` 키워드에서는 호이스팅이 일어났음을 인지하고 **호이스팅이 발생하지 않은 것처럼 에러를 발생**해서 알려준다.

그렇다면 `var`을 왜 사용하지 말라는 걸까?

`var` : 선언 단계 + 초기화 단계 동시에 발생해버림. 즉, 선언과 동시에 `undefined`가 할당됨 → 호이스팅 될 때 해당 변수가 `undefined`임이 같이 등록됨.

`let` : 선언 단계만 발생. 값이 아직 초기화되거나 할당되지 않았음을 엔진이 알고 있음.

`const` : 애초에 선언 + 할당이 명시적으로 동시에 발생해야 함. 더불어 `let`과 마찬가지로 호이스팅은 발생하지만 '선언'에 대한 호이스팅만 발생.

이 부분은 자바스크립트의 핵심 개념이지만, 정말 정말 어렵고 난해한 개념이에요.

앞서 말씀드렸듯이 자바스크립트는豁아보기엔 쉬운 언어지만 깊게 공부하기엔 어려운 언어라는 것이 이런 개념들 때문이에요 :(
정말정말 요약했고 뺀거 다 뺀 내용이지만 그럼에도 어려운거 맞고, 이해 못했다고 해서 당신이 바보라는 것이 아님

네이밍 규칙

특수문자를 제외한 문자, 숫자, 언더바, 달러기호 포함 가능

숫자로 시작하면 안됨

예약어를 식별자로 사용할 수 없음 (아래 표)

await	break	case	catch	class	const
continue	debugger	default	delete	do	else
enum	export	extends	false	finally	for
function	if	implements*	import	in	instanceof
interface*	let*	new	null	package*	private*
protected*	public*	return	super	static*	switch
this	throw	TRUE	try	typeof	var
void	while	with	yield*		

리터럴

사람이 이해할 수 있는 문자 또는 약속된 기호를 통해 값을 생성하는 표기법

리터럴	예시	비고
정수 리터럴	100	
부동소수점 리터럴	10.5	
2진수 리터럴	0b01000001	0b로 시작
8진수 리터럴	0o101	ES6에서 도입, 0o로 시작
16진수 리터럴	0x41	ES6에서 도입, 0x로 시작
문자열 리터럴	'Hello' "World"	
불리언 리터럴	true false	
null 리터럴	null	
undefined 리터럴	undefined	
객체 리터럴	{ name: 'Lee', address: 'Seoul' }	
배열 리터럴	[1, 2, 3]	
함수 리터럴	function() {}	
정규 표현식 리터럴	/[A-Z]+/g	

데이터 타입

값은 메모리에 저장 & 참조될 수 있어야함.

기본적으로 자바스크립트에서 데이터 타입은, 값을 할당할 때 암시적으로 추론됨.

구분	데이터 타입	설명
원시타입	숫자 타입	숫자, 정수, 실수 구분 없이 하나의 숫자 타입만 존재
	문자열 타입	문자열
	불리언 타입	논리적 참과 거짓
	undefined 타입	var 키워드로 선언된 변수에 암묵적으로 할당되는 값
	null 타입	값이 없다는 것을 의도적으로 명시할 때 사용하는 값
	심벌 타입	ES6에서 추가된 7번째 타입
객체타입	객체, 함수, 배열 등등	

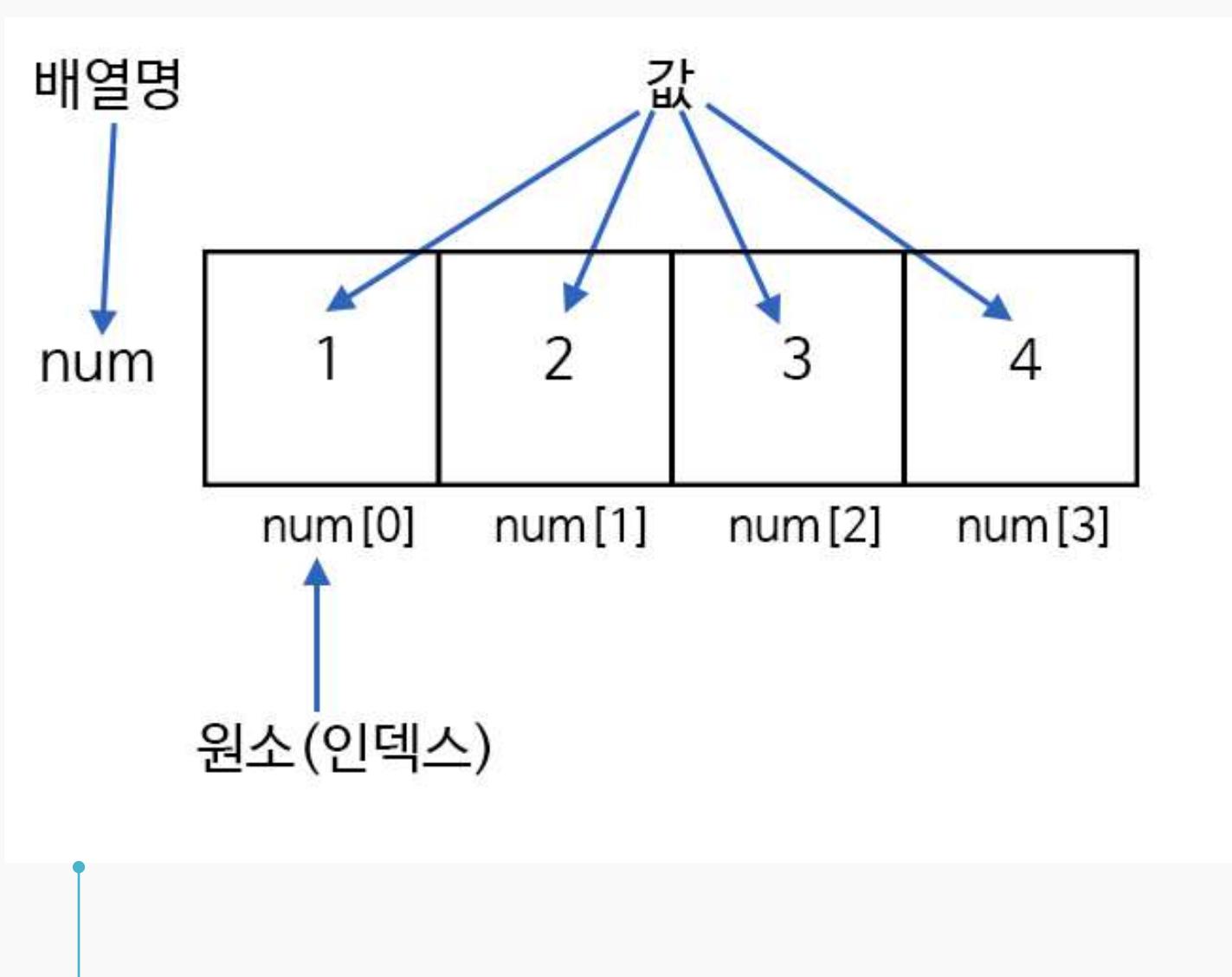
```
// 데이터 타입 테스트
const whatType = "I am string";
console.log(typeof whatType);
```

```
Node.js v18.14.2
> node index.js
string
```

typeof 연산자를 통해서 해당 변수의 타입을 알아낼 수 있음!

데이터 타입 : 배열

자바스크립트는 배열 마저 자유롭다.



배열 만들기 01

빈 배열 만들기

- `[]`로 만들기
- `Array()`로 만들기

```

22 //배열 만들기 (빈 배열)
23 let arr1 = [];
24 let arr2 = new Array();
25
26 arr1[0] = "one";
27 arr1[1] = "two";
28 arr1[2] = "three";
29 console.log(arr1);
30
31 arr2[0] = "one";
32 arr2[1] = "two";
33 arr2[2] = "three";
34 console.log(arr2);
  
```

```

> node index.js
[ 'one', 'two', 'three' ]
[ 'one', 'two', 'three' ]
  
```

배열 만들기 02

초기 값 할당해서 배열 만들기

- `[]` 내부에 값 할당해서 선언
- `Array()` 내부에 값 할당해서 선언

```

//배열 만들기 (초기 값 할당)
let arr1 = ["one", "two", "three"];
let arr2 = new Array("one", "two", "three");

console.log(arr1);
console.log(arr2);
  
```

```

> node index.js
[ 'one', 'two', 'three' ]
[ 'one', 'two', 'three' ]
  
```

배열 만들기 03

배열 크기 지정해서 만들기

```

//배열 크기 지정해서 만들기
let arr1 = [ , , ];
let arr2 = new Array(3);
console.log(arr1.length);
console.log(arr2.length);
  
```

```

> node index.js
3
3
  
```

산술 연산자

이항 / 단항 / 문자열 연결 연산자

```
//이항 산술 연산자
let add = 5 + 2; //7 : 덧셈
let minus = 5 - 2; //3 : 뺄셈
let multiply = 5 * 2; //10 : 곱셈
let divide = 5 / 2; //2.5 : 나눗셈
let mod = 5 % 2; //1 : 나머지
```

```
//단항 산술 연산자
let x = 1;
x++; // x = x + 1 : x가 1 증가
x--; // x = x - 1 : x가 1 감소
+x; // 아무런 효과가 없음 음수 -> 양수의 반전x
-x; // 양수 -> 음수, 음수 -> 양수 변환
```

```
//문자열 연결 연산자
"1" + 2; // "12"
1 + "2"; // "12"

1 + 2; // 3

1 + true; // true가 1이므로, 2
1 + false; // false가 0이므로, 1
1 + null; // null이 0이므로, 1
1 + undefined; // undefined는 숫자로 변환이 안되므로, NaN (Not A Number)
```

할당 연산자

연산자 기준 좌측에 위치한 변수에, 연산과 함께 값을 할당할 수 있는 연산자

Operator	Example	Same As
=	<code>x = y</code>	<code>x = y</code>
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	<code>x %= y</code>	<code>x = x % y</code>

비교 연산자

변수 또는 값 간의 같거나 다름을 결정하는 연산자.

`true`와 `false`를 반환해줌.

연산자	설명
<code>==</code>	<code>equal to</code> 같은지
<code>===</code>	<code>equal value and equal type</code> 값과 타입이 모두 같은지
<code>!=</code>	<code>not equal</code> 같지 않은
<code>!==</code>	<code>not equal value or not equal type</code> 값과 타입이 모두 같지 않은
<code>></code>	<code>greater than</code> 보다 큼
<code><</code>	<code>less than</code> 보다 작은
<code>>=</code>	<code>greater than or equal to</code> 크거나 같은
<code><=</code>	<code>less than or equal to</code> 작거나 같은
<code>?</code>	<code>ternary operator</code> 삼항 연산자

논리 연산자

값을 비교하여 논리 규칙에 따라 true와 false를 반환하는 연산자.

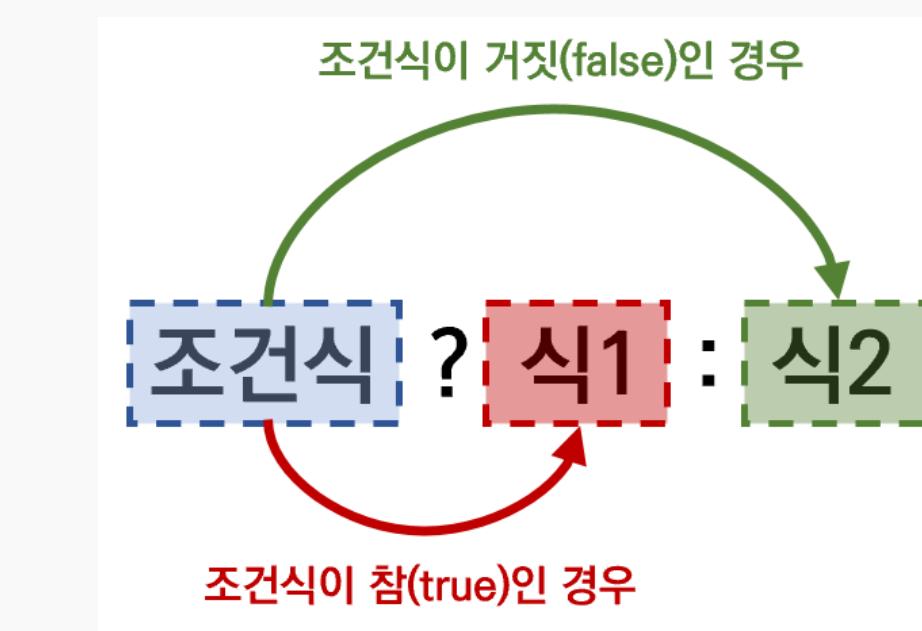
AND, OR, NOT이 존재함

연산자	설명
&&	logical and
	logical or
!	logical not

Conjunction		Disjunction	
A	B	A && B	A B
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

삼항 조건 연산자

if ~ else ~ 의 조건문을 뛰어난 가독성으로 사용할 수 있는 연산자



typeof / instanceof

연산자	설명
typeof	변수의 타입을 반환한다
instanceof	object가 해당 object 타입이면 true를 반환한다

typeof의 경우, 문자열 비교를 통해서 true / false 도출이 가능하다!
(e.g. typeof 3; // === 'number')

instanceof의 경우, 특정 객체가 특정 클래스 출신인지 확인할 수 있는 연산자이다.

타입 변환 : 암묵적 타입 변환

다른 타입의 값 사이에서 연산이 일어날 때,
자바스크립트 스스로 타입을 판단해서 결정하는 변환 방법.

```
number + number // number
number + string // string
string + string // string
string + boolean // string
number + boolean // number
50 + 50; //100
100 + "점"; //"100점"
"100" + "점"; //"100점"
"10" + false; //"100"
99 + true; //100
```

```
// 다른 연산자(-,*,/,%)
string * number // number
string * string // number
number * number // number
string * boolean //number
number * boolean //number
"2" * false; //0
2 * true; //2
```

타입 변환 : 명시적 타입 변환

표준 빌트인 생성자 함수 (String, Number, Boolean)를 통해서 변환하는 방법

```
let trans = 100; //Number
Object(trans); //100
console.log(typeof trans); //Number
toString(trans); //"100"
console.log(typeof trans); //String
Boolean(trans); //true
console.log(typeof trans); //Boolean
```

숫자로 변환

Number() / parseInt() / parseFloat()

```
const falsy1 = null;
Number(falsy1); // 0;

const falsy2 = '';
Number(falsy2); // 0;

const falsy3 = false;
Number(falsy3); // 0;

const truthy1 = [];
Number(truthy1); // 0;

const truthy2 = true;
Number(truthy2); // 1;

const truthy3 = {};
Number({}); // NaN;
```

문자로 변환

Number() / parseInt() / parseFloat()

```
String(123); //"123"
String(123.456); //"123.456"
```

```
var trans = 100;
trans.toString(); //"100"
trans.toString(2); //"1100100"
trans.toString(8); //"144"
var boolT = true;
var boolF = false;
boolT.toString(); //"true"
boolF.toString(); //"false"
```

Boolean으로 변환

Boolean()

```
Boolean(100); //true
Boolean("1"); //true
Boolean(true); //true
Boolean(Object); //true
Boolean([]); //true
Boolean(0); //false
Boolean(NaN); //false
Boolean(null); //false
Boolean(undefined); //false
Boolean(); //false

const numb1 = 0;
Boolean(numb1); // false
!numb1; // false
!numb1; // true
```

타입 갖고 놀아보기!

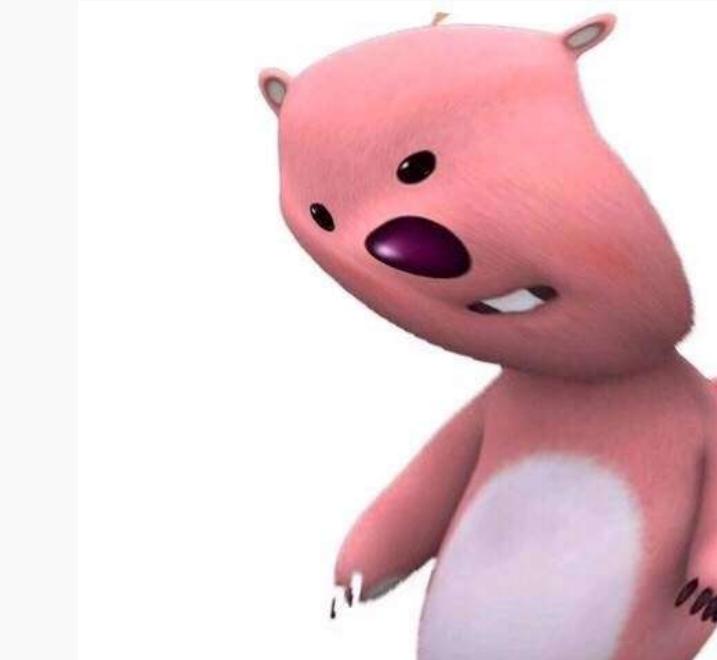
가이드라인에 따라, 변수에 내 정보를 할당해서 배열에 담아주세요!
이후 백틱(`)을 활용하여 본인의 소개문을 작성해주세요!

```
function solution() {
    let info = [];
    let myName; //본인의 이름을 담아주세요
    let myAge; //본인의 나이를 담아주세요
    let department; //본인의 학과를 담아주세요
    let sId; //본인의 학번을 담아주세요

    /**
     * info 배열에 순서대로 값을 넣어주세요!
     * [이름, 나이, 학과, 학번]
     */

    let introduce; //배열을 이용해서 본인의 소개문을 작성해주세요!
    return introduce;
}

console.log(solution());
```



백틱이란, 자바스크립트에서 문자열과 변수를 조합할 때 유용하게 사용할 수 있는 표기법이다.
키보드 상에서 상단 1번 왼쪽 물결 표시있는 부분에 위치하고 있음. (맥북은 영어일때만 나옴)

```
const num1 = 10;
const num2 = 20;

console.log(` ${num1} + ${num2} = ${num1 + num2} 입니다.`);
// "10 + 20 = 30 입니다."
```

요로코롬,
백틱 내부에서 변수를 갖고오고 싶을 때
\${} 안에 변수를 담아준다.



잘 쉬는 것도 중요한 일이랬어

블록문

제어문에서 기본이 되는 코드 블록!

기본적으로 중괄호로 묶여있는 하나의 묶음을 블록이라 부른다.

```
{
  let block = "hi~";
}
```

- 자바스크립트에서 블록문은 하나의 실행 단위!
- 제어문을 사용하거나 / 함수를 정의할 때 사용하는 것이 일반적임.

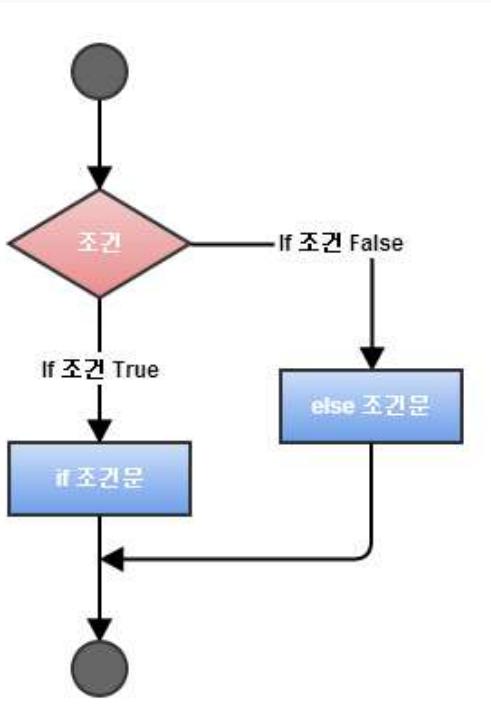
조건문

주어진 조건식의 평가 결과에 따라서 코드 블록의 실행을 결정하는 제어문.

→ 불리언 값으로 평가될 수 있는 표현식

```
let num = 2;
let kind;

if (num > 0) {
  kind = "양수";
} else if (num < 0) {
  kind = "음수";
} else {
  kind = "zero";
}
```



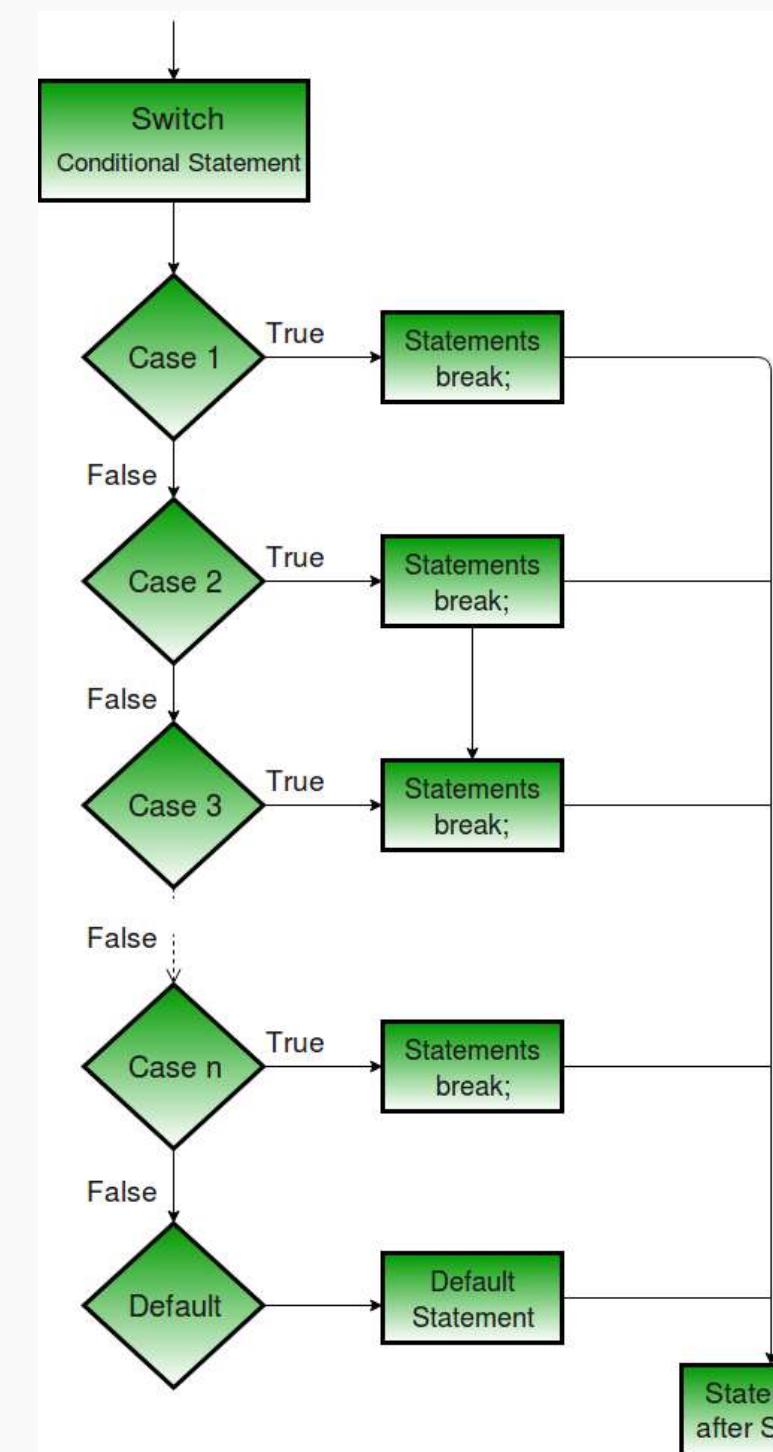
조건문은 여러번 중첩되거나 / 조건식이 복잡해질수록
가독성이 급격하게 떨어진다는 특징이 있음.

Switch문

조건문의 가독성을 향상한 버전.

주어진 표현식을 평가

-> 그 값과 일치하는 표현식을 갖는 case문으로 실행 흐름을 옮김



```
const Test = 80;
switch(Test) {
  case 100 : // case : 조건
    console.log('100점 통과');
    break; // 멈추기
  case 90 :
    console.log('90점 통과');
    break;
  case 80 :
    console.log('80점 통과'); // 조건값과 동일한 case, 미 구문을 실행한다.
    break;
  default : // 조건 외의 값 else와 가능이 동일
    console.log('80점 미만 탈락입니다.');
    break;
}
```

반복문

말 그대로 코드 블록을 반복하는 제어문.

조건식의 평가가 참일 때만 반복을 이어나간다.

while 문

조건식의 결과를 바탕으로,
코드 블록을 반복함

초기화

```
while(조건식) {
    증감문
    //코드를 작성하는 곳
}
```

for 문

조건식의 결과를 바탕으로,
증감식을 통해 코드 블록을 반복함

```
②true
⑤true ④i=1
①i=0 ⑧false ⑦i=2
for (var i = 0; i < 2; i++) {
    console.log(i); ③i=0 ⑥i=1
}
```

```
for(초기화 조건식 증감문) {
    //코드를 작성하는 곳
}
```

break

코드 블록 중간에,
반복을 아예 중지하고 해당 블록을 탈출

```
for (init; condition; update) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

```
while (condition) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

continue

해당 반복을 중지하고
다음 회차의 반복으로 넘어감

```
for (init; condition; update) {
    // code
    if (condition to break) {
        continue;
    }
    // code
}
```

```
while (condition) {
    // code
    if (condition to break) {
        continue;
    }
    // code
}
```

for of 문

열거할 수 있는 이터러블 (배열, Map, Set등) 순회



과제 샤워

forEach

배열 순회 전용 메서드

do while

선수행 블록이 첨가된 while문.
수행 이후 코드블록 반복.

Array.map

배열 순회 전용 메서드

Array.filter

배열 순회 전용 메서드

해당 내용을 공부하고 요약해서
마크다운으로 정리하기

깃허브에 업로드하기

다음 세션 시작 전에 해당 내용 공유할 예정

반복문 갖고 놀아보기!

가이드라인에 따라, 변수에 내 TMI를 할당해서 배열에 담아주세요!
이후 백틱 / 연산자 / 반복문을 활용해서 TMI를 출력해주세요!

```
/**  
 * 내 소개문을 보다 기계적으로 보여주는 함수  
 */  
function solution() {  
    let info = [];  
    //info 배열에 본인의 TMI 다섯개 적기  
    /**  
     * 반복문을 활용해서 출력하기  
     * 단, TMI와 TMI 사이에는 줄 바꿈이 있어야한다.  
     * e.g.  
     * 오늘은 집에 누워있었다.  
     * 유튜브에 빠져 살았다.  
     */  
}  
  
solution();
```





1주차 고정과제

당근마켓 랜딩 페이지 클론

당근마켓에 접속했을 때 최초로 접하는 랜딩 페이지를 클론하세요!
페이지 라우팅(연결)을 하거나 실제 로직을 구현하는 것이 아닌,
하나의 페이지를 단순히 HTML과 CSS만으로 클론하면 돼요.

[당근마켓 랜딩 페이지로 이동](#)



주차 과제

반복문 학습 + 문서화

이전 페이지에서 제시한 자바스크립트의 반복문 내용을 학습하고,
마크다운 형식으로 정리해서 깃허브에 업로드합니다.
2주차 세션 시간에 정리한 내용을 공유할 예정이에요.

깃허브 FE:SSION 레포지토리 clone

```
git clone https://github.com/likelion-ssu/2023_FE-SSION.git
```

*master 브랜치에서, 새로운 브랜치 생성

```
git checkout master  
git checkout -b week01/[본인이름]  
e.g. git checkout -b week01/junkyu
```

Week01 디렉토리 하위에 본인의 이름으로 디렉토리 생성 후, 과제 파일 작성

```
git checkout master  
git checkout -b week01/[본인이름]  
e.g. git checkout -b week01/junkyu
```

과제 작성 중간 중간 커밋 (add / fix / remove)

```
git add .  
git commit -m "add: [커밋 내용 요약]"  
e.g. git commit -m "add: study about repeat for JS"
```

과제 작성이 완료되면, push 후 Pull Request 업로드

```
git push origin [현재 본인의 브랜치 이름]  
FE:SSION 레포지토리의 Pull Requests 탭에서 New pull request 클릭  
[본인 브랜치] -> [master] 설정 후 내용 작성해서 Pull Request 업로드  
e.g. git push origin week01/junkyu
```



최강 프론트 화이팅!!!!!!