

**면접 단골 질문**

**PRE-ONBOARDING**

**이벤트 루프 다시 돌아보기**

**FRONTEND CHALLENGE 25차**

The bottom half of the slide features large, abstract, organic shapes in bright orange and yellow, creating a dynamic and modern background.

# 면접 단골 질문 이벤트 루프 다시 돌아보기

**Week 1-2**

이벤트 루프 자세히 알아보기 1

## 강의 진행 미리보기

Week 1-1

- 강의 OT
- 이벤트 루프 개념
- 비동기, 동기
- 싱글, 멀티 스레드

Week 1-2

- 큐, 스택
- 이벤트루프 큐, 스택

Week 2-1

- callback, promise
- async/await
- 이벤트 루프 고려한 코드

Week 2-2

- 면접 질문 발표해보기




목차

## 3시간 미리보기

1. Queue와 Stack 이해하기
2. 이벤트 루프에서 Queue와 Stack 알아보기
3. 면접관이 놀랄만한 포인트

# 1. Queue, Stack



1. Queue, Stack

이들의 공통점은?

Queue

Array

자료구조

Graph

Stack

Tree

## 1. Queue, Stack

# 자료구조란 무엇인가?

자료구조란 대체 무엇인지 설명해주겠니?



자료구조는 데이터를 효율적으로 저장하고 관리하기 위한 방법이나 형식을 의미합니다. 쉽게 말해, 컴퓨터가 데이터를 잘 저장하고, 필요할 때 빠르게 찾거나 변경할 수 있도록 도와주는 틀이나 구조를 자료구조라고 부릅니다.

## 자료구조의 역할

1. 데이터 저장: 데이터를 체계적으로 저장하여 컴퓨터가 효율적으로 접근할 수 있게 합니다.
2. 데이터 검색 및 처리: 특정 데이터를 빠르게 찾거나, 정렬, 추가, 삭제 등의 작업을 쉽게 수행할 수 있게 도와줍니다.
3. 메모리 관리: 메모리를 효율적으로 사용하여 공간을 절약하고 성능을 향상시킵니다.

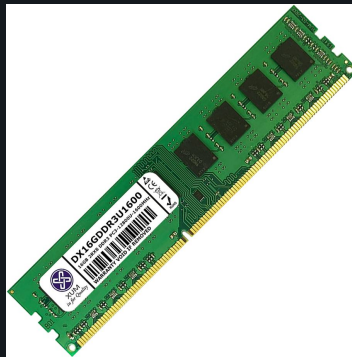
1. Queue, Stack

# 저장 공간 알아보기

## Hard Disk



## Memory





# 1. Queue, Stack

## 저장 공간 알아보기



Hard Disk

가격

저렴하다

용량

많다

성능

느리다

Memory



비싸다

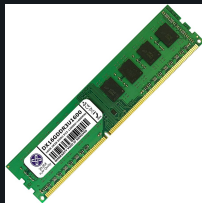
적다

빠르다

1. Queue, Stack

## 저장 공간 알아보기

### Program



1. Queue, Stack

# 저장 공간 알아보기

## Process



1. Queue, Stack

## 저장 공간 알아보기

### Process



용량이 적은 Memory를 효율적으로 사용해야 함



1. Queue, Stack

# 비트와 바이트, 2진법과 16진법

0



1



# 1. Queue, Stack

## 비트와 바이트, 2진법과 16진법

0



1



1 bit

0 | 1

# 1. Queue, Stack

## 비트와 바이트, 2진법과 16진법

2 bit

0

1

00 | 01 | 10 | 11



# 1. Queue, Stack

## 비트와 바이트, 2진법과 16진법

8 bit = 1byte

0

1

00000000 | 00000001

...

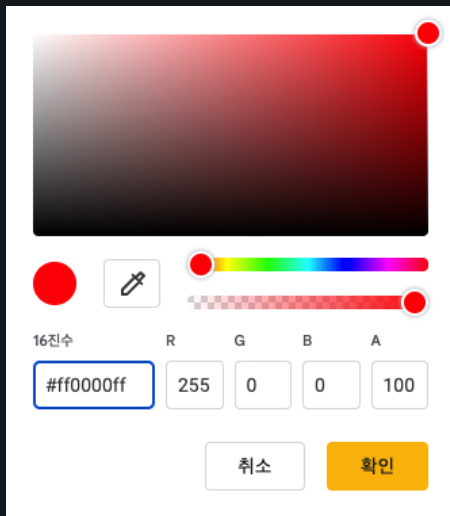
...





## 1. Queue, Stack

# 비트와 바이트, 2진법과 16진법



0xFF0000FF

# 1. Queue, Stack

## 비트와 바이트, 2진법과 16진법

0x FF 00 00 FF

16\*16 = 256

RED

Green

Blue

16진수  
(Hexadecimal)

# 1. Queue, Stack

## 메모리 자세히 알아보기

0x7AF00

0

0

0

0

0

0

0

0x7AF01

0

0

0

0

0

0

0

0x7AF02

0

0

0

0

0

0

0

0x7AF03

0

0

0

0

0

0

0

0x7AF04

0

0

0

0

0

0

0

0x7AF05

0

0

0

0

0

0

0

0x7AF06

0

0

0

0

0

0

0

0x7AF07

0

0

0

0

0

0

0

0x7AF08

0

0

0

0

0

0

0

0x7AF09

0

0

0

0

0

0

0

0x7AF10

0

0

0

0

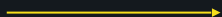
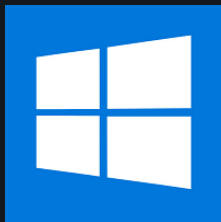
0

0

0

1. Queue, Stack

# 메모리 자세히 알아보기



0x7AF020 부터 0x7AF060 까지 할당해줄게

0x7AF00

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0x7AF01

0x7AF02

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0x7AF03

0x7AF04

0x7AF05

0x7AF06

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0x7AF07

0x7AF08

0x7AF09

0x7AF10


0	0	0	0	0	0	0	0
1	2	3	4	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

## 1. Queue, Stack

# 메모리 자세히 알아보기



Memory



## 1. Queue, Stack


# 여기까지 정리

## 자료구조란?

데이터를 효과적이고 효율적으로 저장하거나 관리하기 위한 방법

## 효율적으로 관리해야 하는 이유?

실제로 프로그램이 실행되는 Memory의 자원은 한정되어 있기 때문



## 1. Queue, Stack

# 여기까지 정리

## 데이터는?

무조건 0과 1로 이루어진 이진법으로 저장, 편의를 위해 16진법으로 표기함

## 메모리의 구조는?

Code, Data, Heap, Stack으로 이루어짐

## 1. Queue, Stack

# Stack 자료구조 알아보기

stack

noun [C]

UK  /stæk/ US  /stæk/

stack noun [C] (PILE)

Add to word list 



a pile of things arranged one on top of another:

- He chose a cartoon from the stack of DVDs on the shelf.

malerapaso/iStock/Getty Images  
Plus/Getty Images

쌓아올림, 쌓다



# 1. Queue, Stack

## Stack 자료구조 알아보기

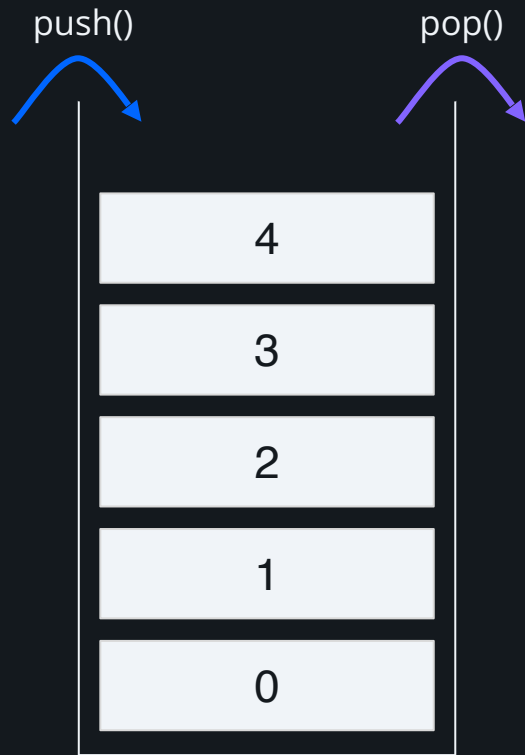
**Last In First Out**

마지막에 들어간 것이 처음 나온다


=

**First In Last Out**

처음 들어간 것이 마지막에 나온다



Stack



## 1. Queue, Stack

# Stack 자료구조 알아보기

구현이 간단하고 빠르다

wanted.co.kr

daum.net

naver.com

History Stack

## 1. Queue, Stack

# Stack 자료구조 알아보기

중간 데이터 접근이 어렵다



Stack

## 1. Queue, Stack

# Queue 자료구조 알아보기

**Last In Last Out**

마지막에 들어간 것이 마지막에 나온다

=

**First In First Out**

처음 들어간 것이 처음 나온다



enqueue()

rear

front



Queue



dequeue()

# 1. Queue, Stack

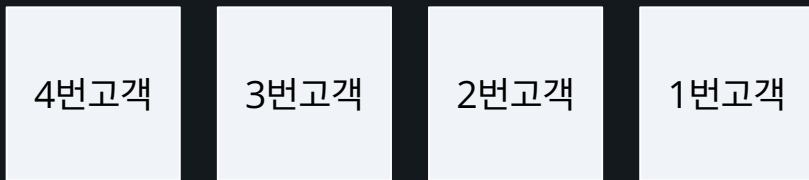
## Queue 자료구조 알아보기



STARBUCKS

rear

front



enqueue()



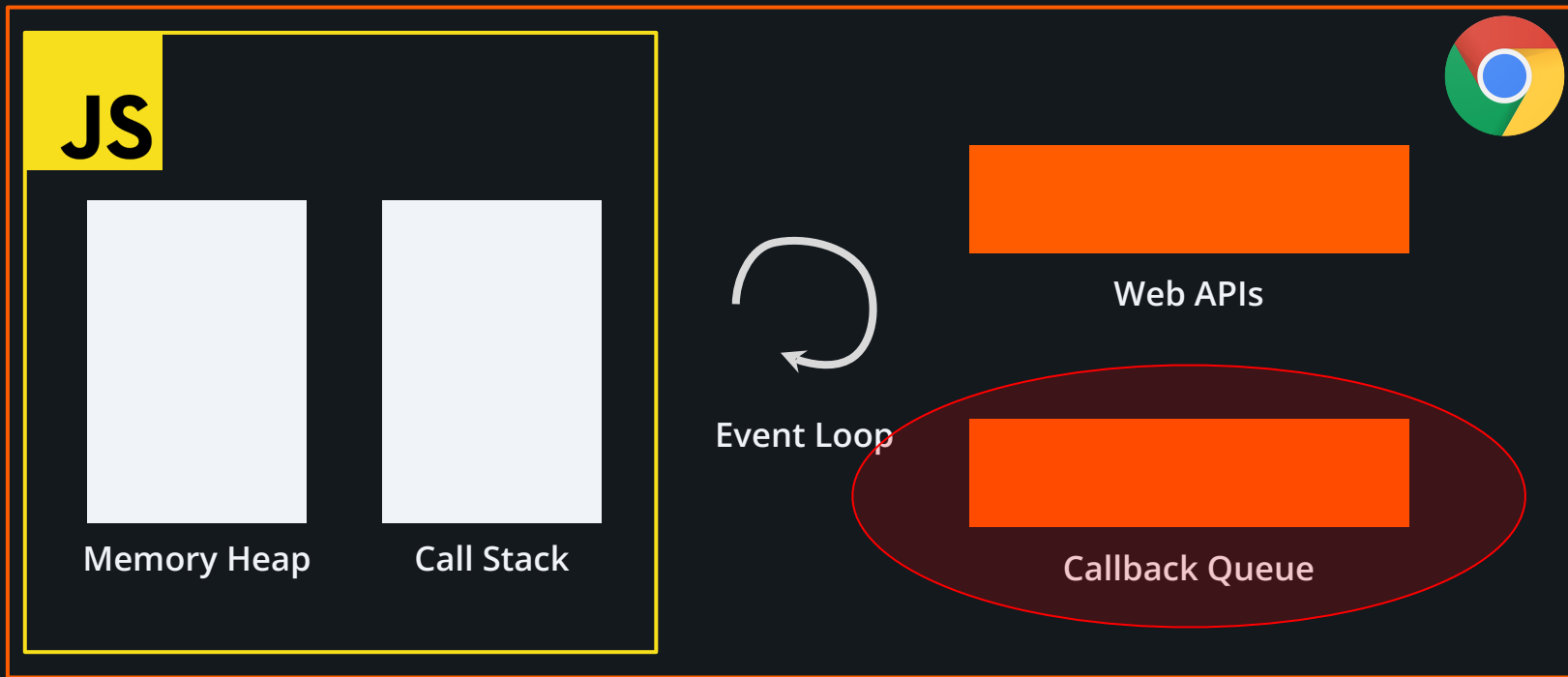
dequeue()

Queue

## 2. Queue, Stack in Event Loop

## 2. Queue, Stack in Event Loop

# Event loop에서 Queue



## 2. Queue, Stack in Event Loop

# Event loop에서 Queue


## Callback Queue

Macro Task Queue

Micro task Queue

Animation Frame Queue





## 2. Queue, Stack in Event Loop

# Event loop에서 Queue

우선순위

Macro Task Queue < Micro task Queue



## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

## Macro Task Queue

**setTimeout**, setInterval, DOM, I/O, Network request, Event handlers

## Micro task Queue

**Promise** callback, mutation observer API, await in async func

## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```

console.log("시작");

setTimeout(()=>{
  console.log("setTimeout 실행");
}, 0);

Promise.resolve().then(()=>{
  console.log("Promise 실행");
});

console.log("종료");

```

## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```

console.log("시작");

setTimeout(()=>{
  console.log("setTimeout 실행");
}, 0);

Promise.resolve().then(()=>{
  console.log("Promise 실행");
});

console.log("종료");

```

시작

종료

Promise 실행

← undefined

setTimeout 실행

## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
setTimeout();
```

```
Promise.then()
```

```
console.log("종료")
```

JS

Call Stack



Event Loop

Web APIs

Micro task Queue

Macro task Queue



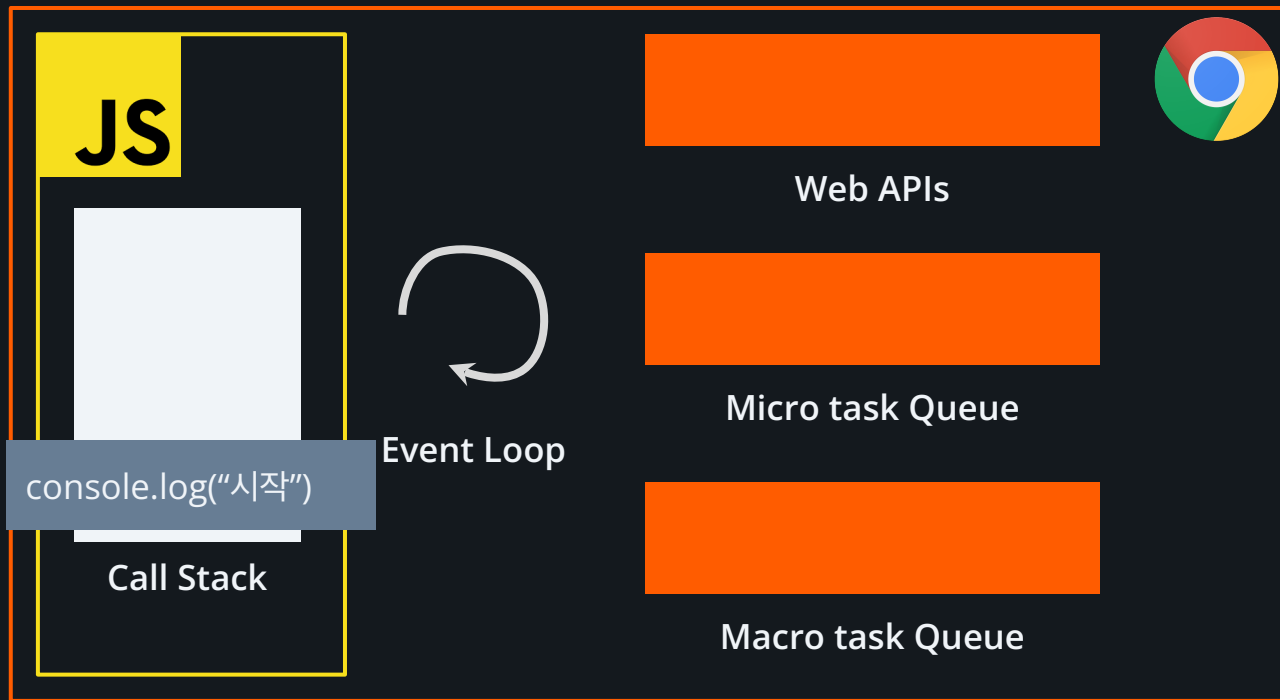
## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```
setTimeout();
```

```
Promise.then()
```

```
console.log("종료")
```



## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
setTimeout();
```

```
Promise.then()
```

```
console.log("종료")
```

시작

JS

Call Stack



Event Loop

Web APIs

Micro task Queue

Macro task Queue



## 2. Queue, Stack in Event Loop

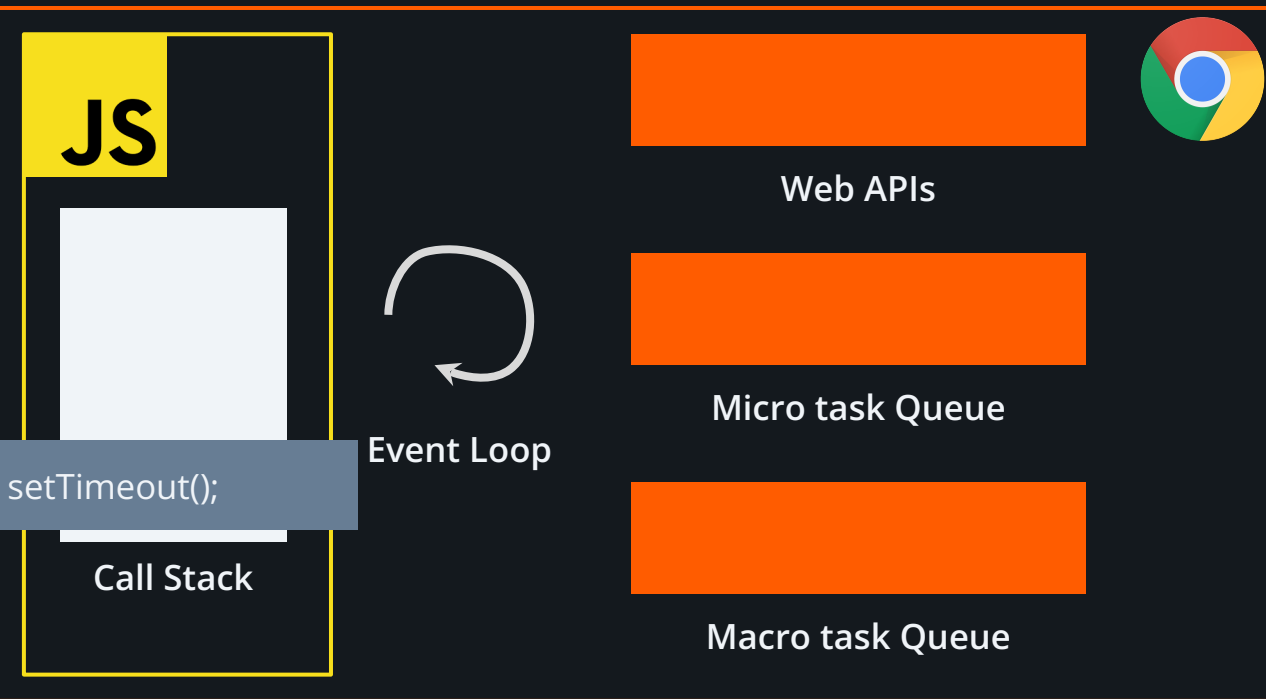
# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
Promise.then()
```

```
console.log("종료")
```

시작





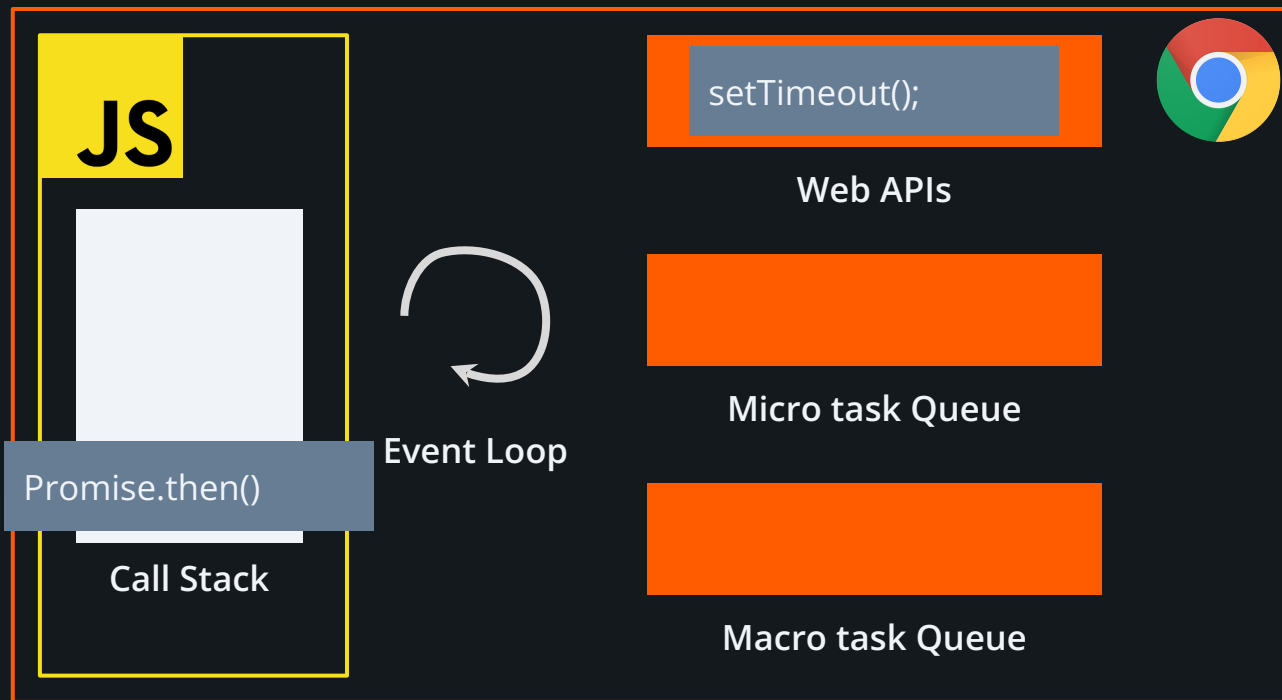
## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
console.log("종료")
```

시작



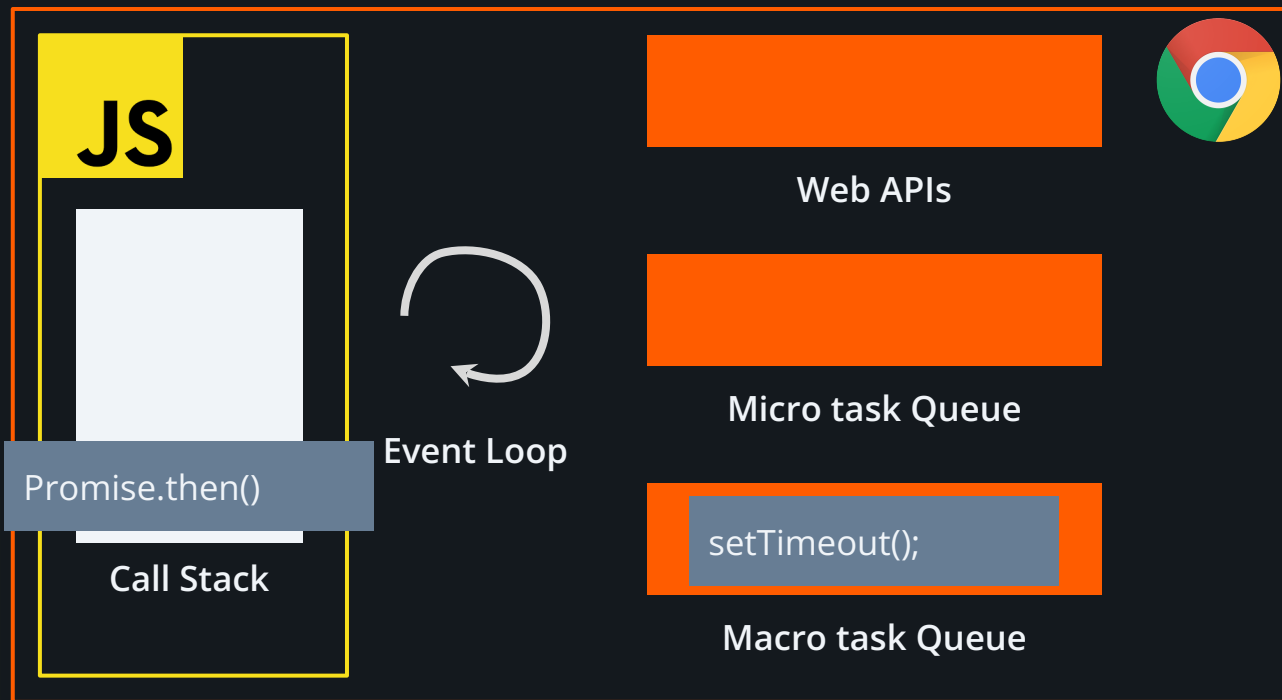
## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
console.log("종료")
```

시작



## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
console.log("종료")
```

시작

JS

Call Stack



Event Loop

Web APIs

Promise.then()

Micro task Queue

setTimeout();

Macro task Queue

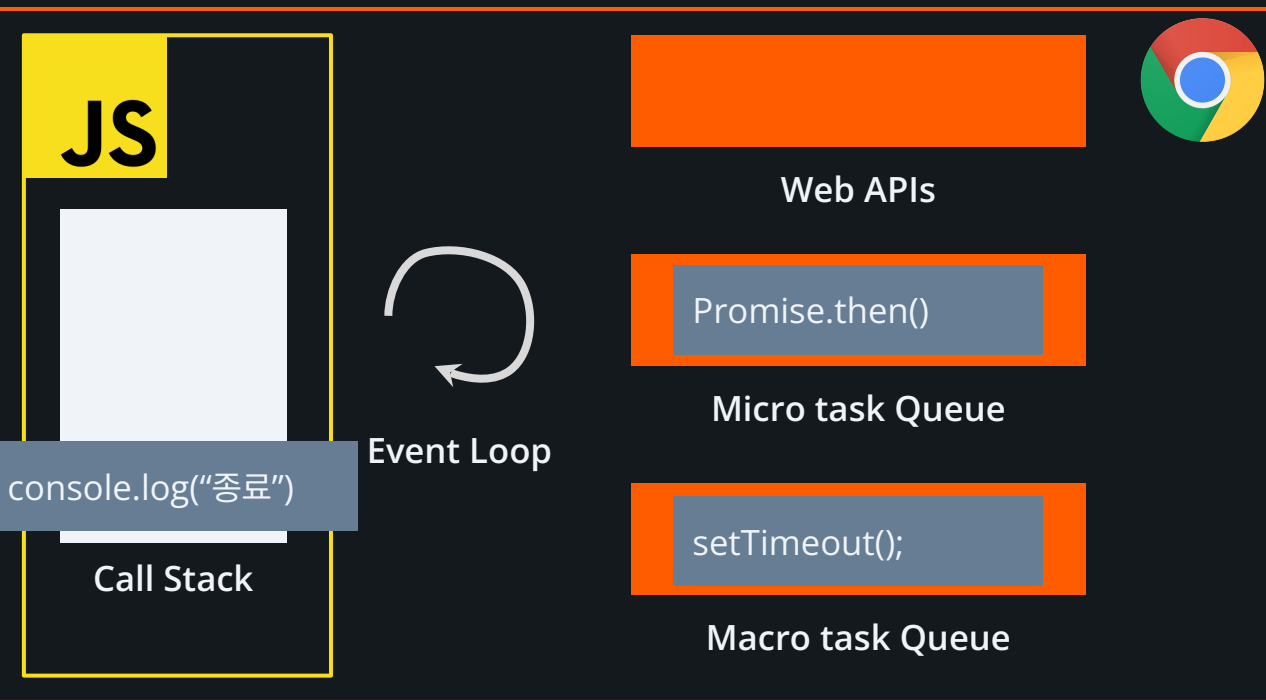


## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```
console.log("시작")
```

시작



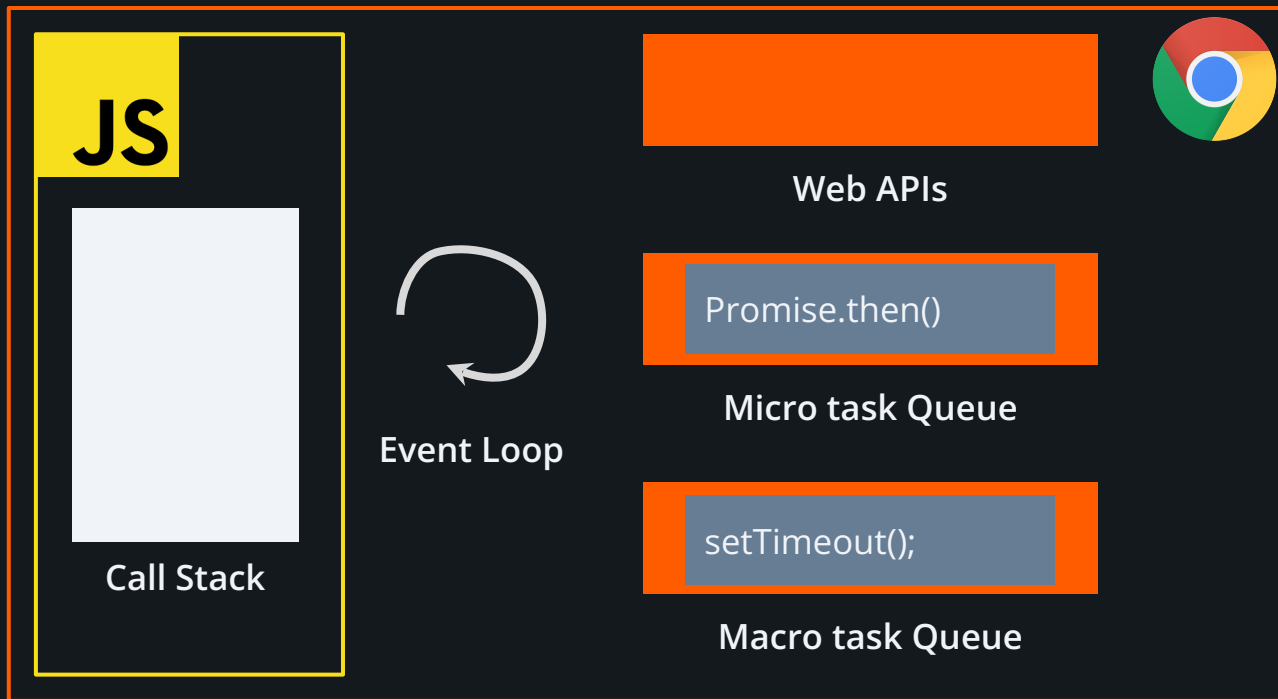
## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
console.log("종료")
```

시작  
종료



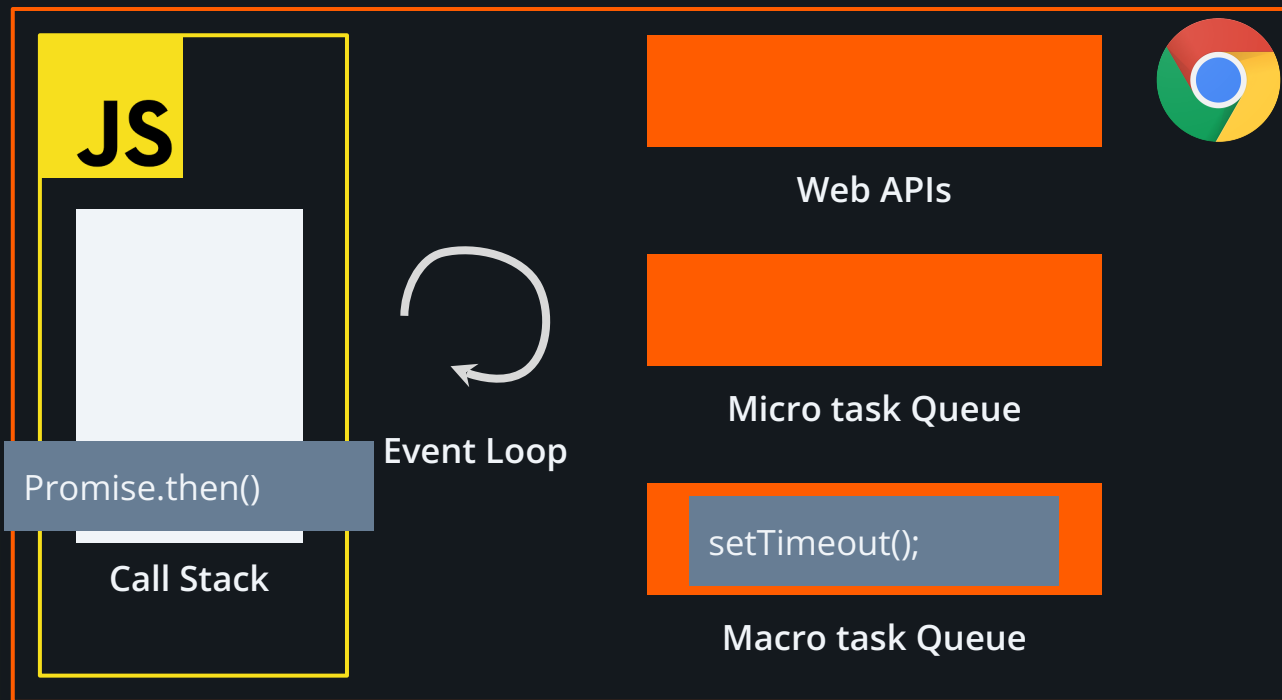
## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
console.log("종료")
```

시작  
종료



## 2. Queue, Stack in Event Loop

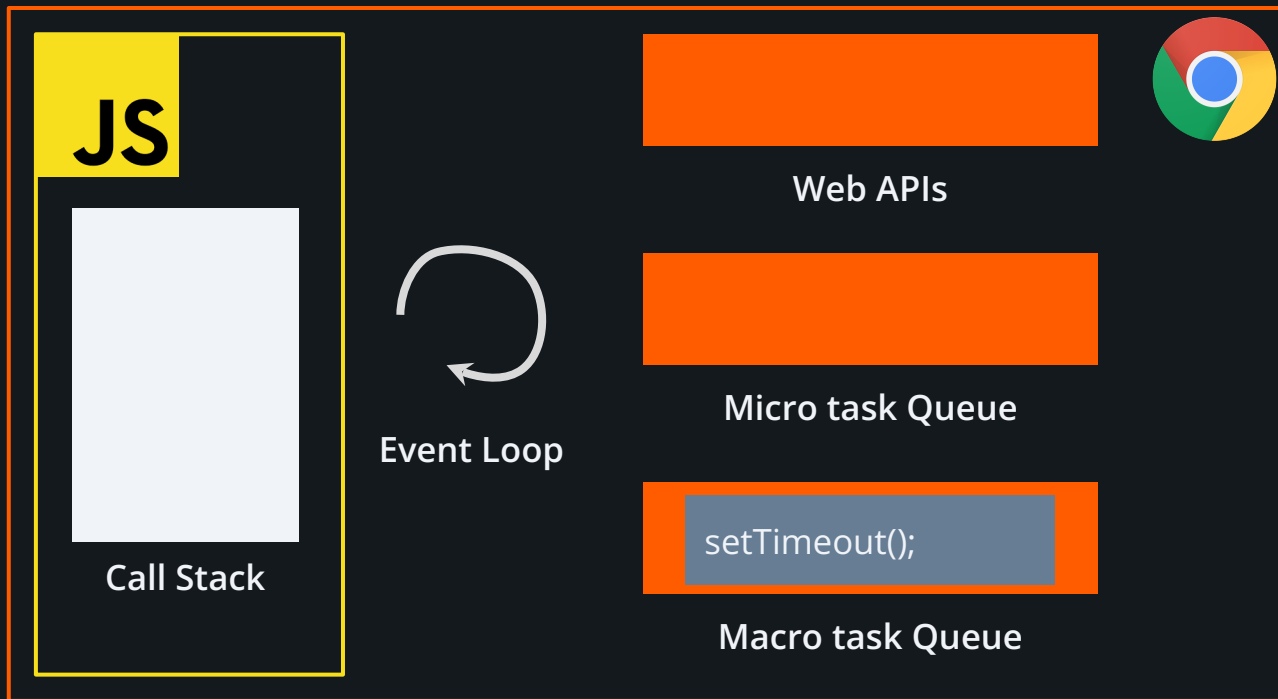
# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
Promise.then()
```

```
console.log("종료")
```

시작  
종료  
Promise 실행



## 2. Queue, Stack in Event Loop

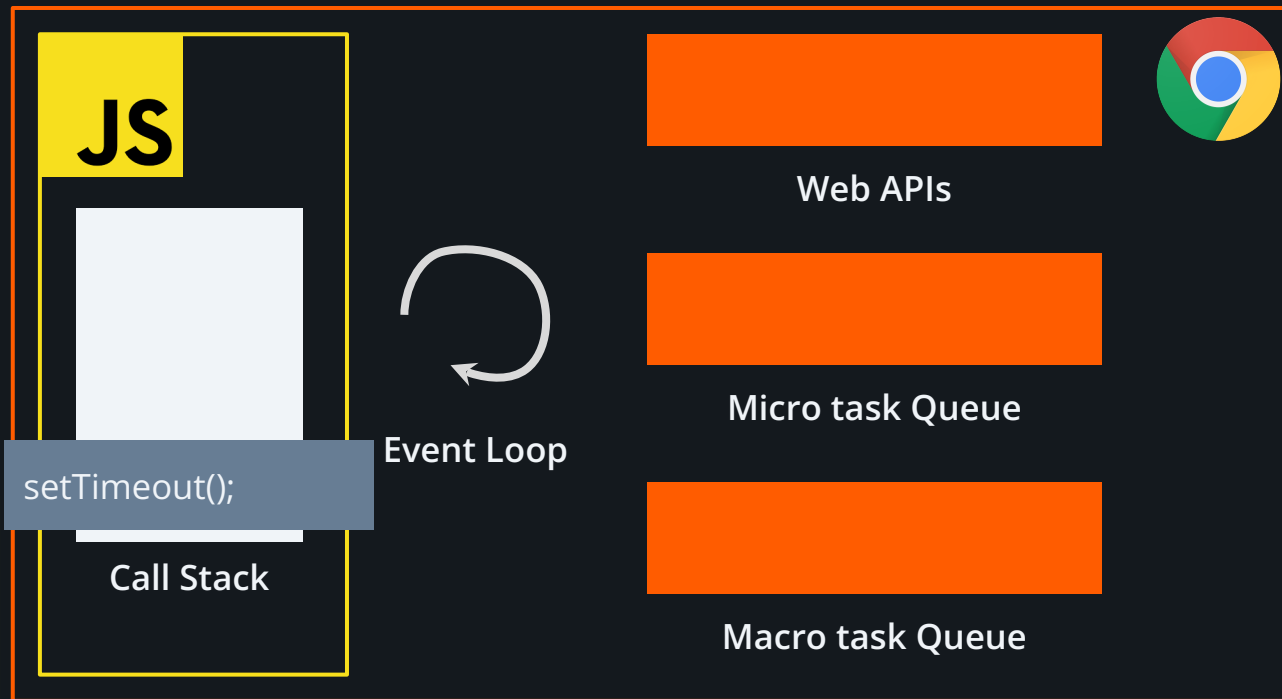
# Task Queue, Micro Task Queue

```
console.log("시작")
```

```
Promise.then()
```

```
console.log("종료")
```

시작  
종료  
Promise 실행





## 2. Queue, Stack in Event Loop

# Task Queue, Micro Task Queue

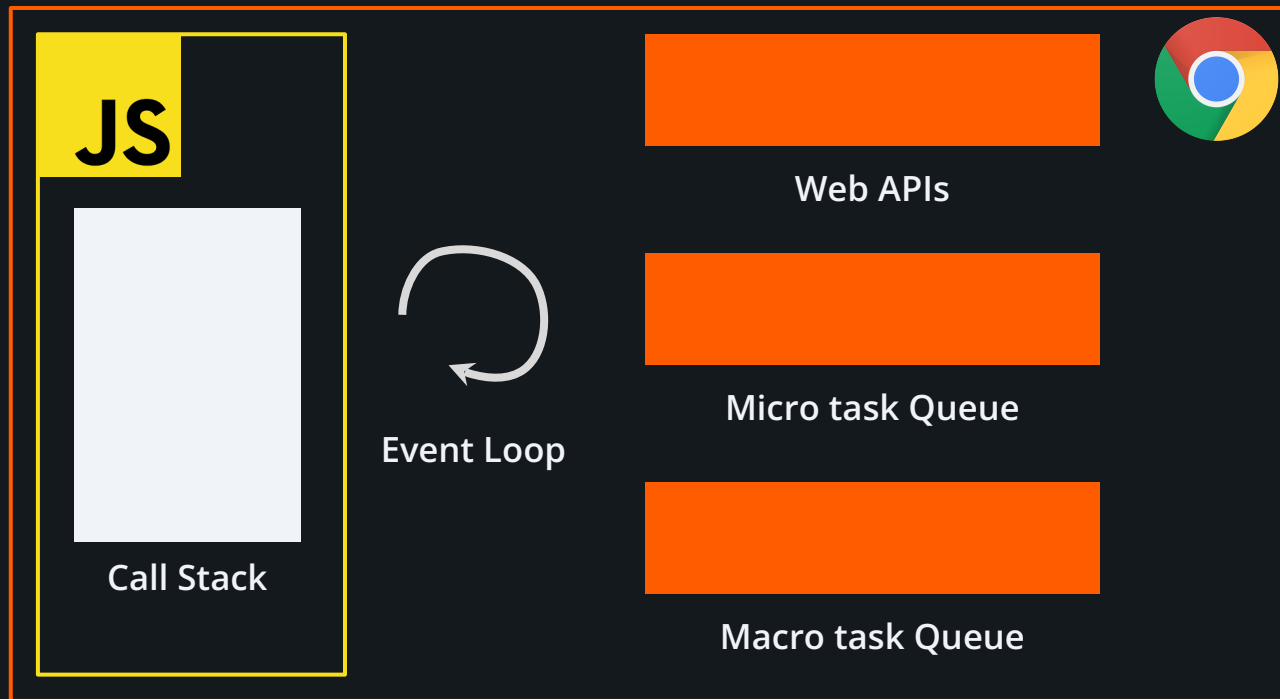
```
console.log("시작")
```

```
setTimeout();
```

```
Promise.then()
```

```
console.log("종료")
```

시작  
종료  
Promise 실행  
setTimeout 실행





2. Queue, Stack in Event Loop

## Task Queue, Micro Task Queue

### Micro task Queue

1번 수행할 때 큐를 모두 비운다

### Macro Task Queue

1번 수행할 때 1개의 Task만 비운다

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

## Call Stack





## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

## Call Stack

프로그램이 실행되며 함수 호출에 대한 정보를 추적, 관리하는 역할

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

Call Stack

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

A();

Call Stack

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

**B();**

**A();**

Call Stack

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

```
C();
```

```
B();
```

```
A();
```

Call Stack



## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

```
console.log("C")
```

```
C();
```

```
B();
```

```
A();
```

Call Stack

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

```
C();
```

```
B();
```

```
A();
```

Call Stack

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

**A();**

**B();**

**A();**

Call Stack

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

```
console.log("B")
```

```
B();
```

```
A();
```

Call Stack

## 2. Queue, Stack in Event Loop

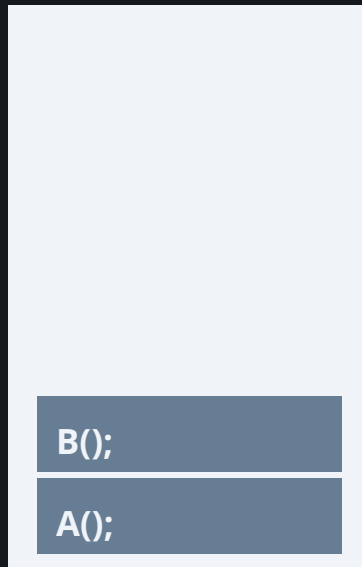
# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

**A();**



Call Stack

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

A();

Call Stack

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

```
console.log("A");
```

```
A();
```

Call Stack

## 2. Queue, Stack in Event Loop

# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```

A();

Call Stack



## 2. Queue, Stack in Event Loop

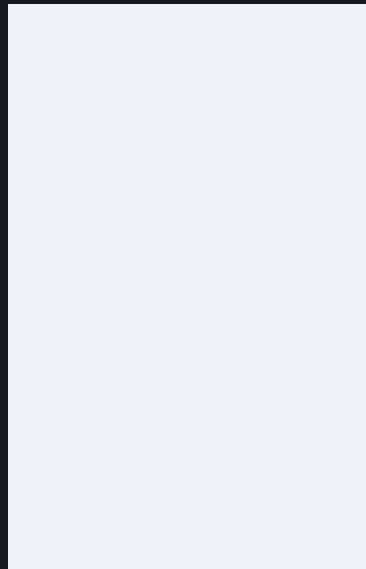
# Event loop에서 Stack

```
function A () {  
  B();  
  console.log("A");  
}
```

```
function B () {  
  C();  
  console.log("B");  
}
```

```
function C () {  
  console.log("C");  
}
```

```
A();
```



Call Stack

2. Queue, Stack in Event Loop

## Event loop에서 Stack

### Call Stack



2. Queue, Stack in Event Loop

## Event loop에서 Stack

# Execution Context





2. Queue, Stack in Event Loop

# JavaScript의 Execution Context

## Call Stack

프로그램이 실행되며 함수 호출에 대한 정보를 추적, 관리하는 역할


# JavaScript의 Execution Context

Execution

실행

Context

맥락



2. Queue, Stack in Event Loop

## JavaScript의 Execution Context

**맥락** 脈(줄기 맥) 絡(이을 락)

사물이 서로 이어져 있는 관계

## JavaScript의 Execution Context

부서진 벽을 보고

= 보수할 **벽돌**을 가져와라

떨어지는 벽돌을 보고

= **벽돌**을 피해라

마리오를 보고

= **벽돌**을 부숴라



2. Queue, Stack in Event Loop

# JavaScript의 Execution Context

😓 : 아 너무 배고프다

😱 : 밥먹을래? 빵먹을래? 과일먹을래?

😓 : 아 너무 배고프다

😈 : 스크립트 1세트 추가



## 2. Queue, Stack in Event Loop

# JavaScript의 Execution Context



```
const 변수_1 = "Hello";
```

```
function 함수( ) {
```

```
    const 변수_1 = "World";
```

```
    console.log(변수_1);
```

```
}
```



# JavaScript의 Execution Context



```
const 변수_1 = "Hello";
```

```
function 함수( ) {
```

```
    const 변수_1 = "World";
```

```
    console.log(변수_1);
```

```
}
```



변수\_1 이라는 변수가 2개나 있는데 어떤걸 보여줘야 하지..?

2. Queue, Stack in Event Loop

# JavaScript의 Execution Context

Lexical



어휘적

Scope



범위

# JavaScript의 Execution Context



```
const 변수_1 = "Hello";
```

```
function 함수( ) {
```

```
    const 변수_1 = "World";
```

```
    console.log(변수_1);
```

```
}
```



코드(어휘) 상으로 함수() 범위 내에 있는 변수\_1을 사용하자

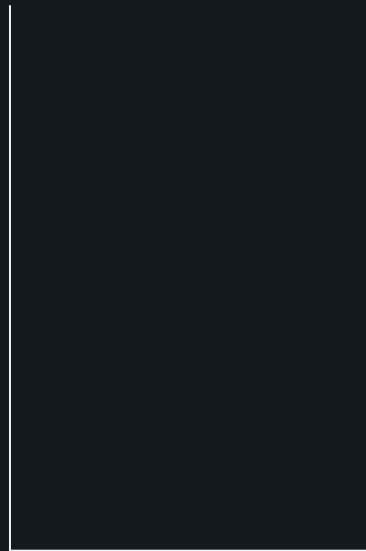


2. Queue, Stack in Event Loop

# JavaScript의 Execution Context



Call Stack



Call Stack

2. Queue, Stack in Event Loop

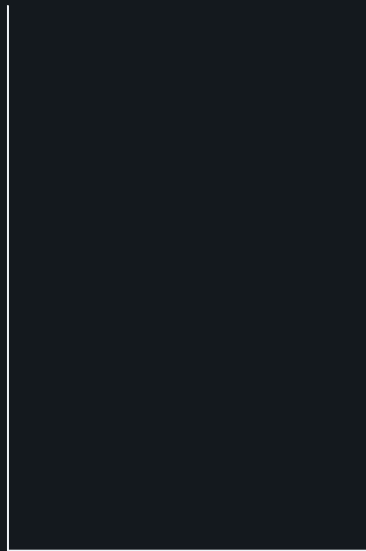
# JavaScript의 Execution Context




Call Stack

= Execution Context Stack

실행 컨텍스트를 쌓는다



Execution Context Stack



2. Queue, Stack in Event Loop

# JavaScript의 Execution Context

코드의 종류

Global Code

Function Code



2. Queue, Stack in Event Loop

## JavaScript의 Execution Context

### Global Code

전역에 존재하는 코드

단, 정의되어 있는 함수나 클래스의 내부 코드는 포함되지 않음



## 2. Queue, Stack in Event Loop

# JavaScript의 Execution Context



```
let 전역변수_1 = "Hello";  
let 전역변수_2 = "World";
```

```
function 함수(){  
  const 지역변수_1 = "hi";  
  
  console.log("함수");  
}
```



2. Queue, Stack in Event Loop

## JavaScript의 Execution Context

### Function Code

함수 내부에 존재하는 코드

단, 함수 내부에 중첩된 함수나 클래스의 내부 코드는 포함되지 않음

## 2. Queue, Stack in Event Loop

# JavaScript의 Execution Context




```
let 전역변수_1 = "Hello";
let 전역변수_2 = "World";

function 함수_1(){
  const 지역변수_1 = "hi";

  function 함수_2(){
    const 지역변수_2 = "Hello";
    console.log("함수_2");
  }

  console.log("함수_1");
}
```



2. Queue, Stack in Event Loop

## JavaScript의 Execution Context

## JavaScript Engine 동작 방식

1. 소스코드 평가

2. 소스코드 실행



2. Queue, Stack in Event Loop

## JavaScript의 Execution Context

### 소스코드 평가

Execution Context를 생성하고, 코드의 시작부터 끝까지 확인 함

이 때 변수나 함수의 선언문 만 먼저 실행하여 식별자를 Execution Context에 저장함

## 2. Queue, Stack in Event Loop

# JavaScript의 Execution Context

## 소스코드 평가

이름 = undefined

Execution Context



```
var 이름;
```

코드 선언문만 실행

```
이름 = "황주현";
```



2. Queue, Stack in Event Loop

# JavaScript의 Execution Context

## 소스코드 실행

선언문을 제외한 나머지 코드를 실행

## 2. Queue, Stack in Event Loop

# JavaScript의 Execution Context

## 소스코드 실행

이름 = undefined

Execution Context



```
var 이름;
```

```
이름 = "황주현";
```



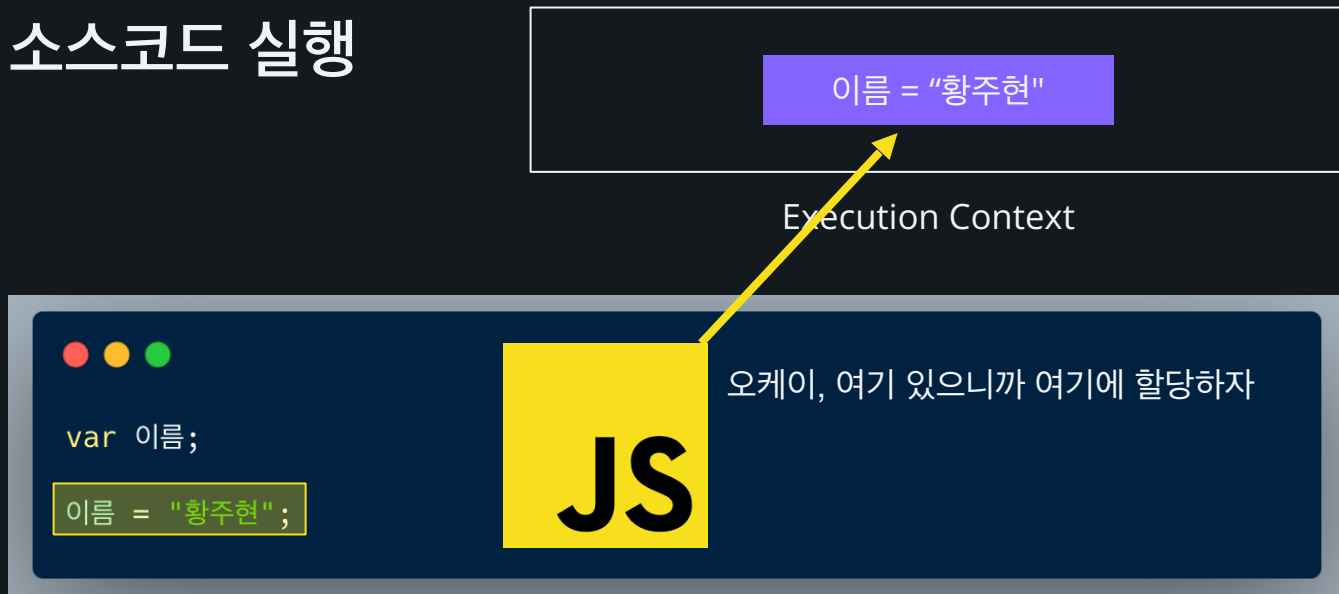
이름이라는 변수에 값을 할당 하고 있네  
근데 이 식별자가 선언 되어 있나?



## 2. Queue, Stack in Event Loop

# JavaScript의 Execution Context

## 소스코드 실행



## 2. Queue, Stack in Event Loop

# JavaScript의 Execution Context



```
function 함수_1( ){
  const 지역변수 = "안녕하세요";
  함수_2( );
}

function 함수_2( ){
  const 지역변수 = "반갑습니다";
}

함수_1( );
```

지역변수:반갑습니다

함수\_2 Execution Context


지역변수:안녕하세요

함수\_1 Execution Context

Global Execution Context

Execution Context Stack

### 3. 면접관이 놀랄만한 포인트



3. 면접관이 놀랄만한 포인트

## 자료구조의 필요성

자료구조가 무엇을 의미하는지

그것이 왜 필요한지



3. 면접관이 놀랄만한 포인트

## Event Loop의 Callback Queue

Callback Queue의 종류를 아는 것

Callback Queue가 왜 나누어졌는지 아는 것



3. 면접관이 놀랄만한 포인트

## Event Loop의 Call Stack

Execution Context와 Call Stack의 관계를 아는 것

JavaScript Engine의 동작 방식을 이해 하고 있는 것

Lexical Scope, Scope Chain에 대해 이해 하고 있는 것

고생하셨습니다.