

Core JavaScript

CHAPTER 02_ 실행 컨텍스트

Execution context(실행 컨텍스트) 란?

의미부터 보자면

Execution 는 실행(실행 중인 코드or 실행되고 있는 코드)

Context 는 문맥/맥락/환경(코드의 배경이 되는 조건/환경)

합쳐서 코드를 실행하는 데 필요한 배경이 되는 조건,환경 이라고 생각할 수 있으며

자바스크립트 코드가 실행되기 위해 필요한 실행중인 코드와 그 실행 환경에 대한 정보를 담은 객체를 실행 컨텍스트(Execution Context) 라고 할 수 있습니다.

자바스크립트에서 동일한 실행 컨텍스트를 공유하는 4가지

전역공간:

자바스크립트에서 전역공간(global space)은 스크립트의 가장 바깥 영역으로,

모든 스크립트에 공통적으로 존재하는 영역을 말합니다.

함수:

자바스크립트에서 함수(function)는 코드를 분리해서 가독성을 높이고 재사용을 할 수 있도록 작성된 코드 블록입니다.

모듈:

자바스크립트 모듈은 재사용 가능한 코드 블록을 만들기 위한 방법입니다. 모듈은 일반적으로 독립적인 기능을 가진 파일이며, 변수, 함수, 클래스 등을 내보내거나 가져올 수 있습니다.

eval:

eval은 자바스크립트의 내장 함수 중 하나로, 문자열로 표현된 자바스크립트 코드를 실행할 수 있게 합니다.

전역공간:

자바스크립트 코드가 실행되는 순간에 바로 전역 컨텍스트가 실행되고 전체 코드가 끝날때 비로소 끝나기때문에 이를 하나의 거대한 함수 공간이라고 봐도 무방합니다.

모듈:

어디선가 **import** 되는 순간에 그 모듈 내부에 있는 컨텍스트가 실행되고 모듈 코드가 끝났을때 컨텍스트가 종료되기 때문에 모듈도 하나의 함수 공간이라고 생각하면 될거 같습니다.

결국 자바스크립트의 실행 컨텍스트는 함수 단위로 생기며

오직 함수에 의해서만 컨텍스트를 구분할 수 있다는 점입니다.

Execution Context 라는걸 다시 설명해보면 함수를 실행할 때 필요한 환경정보를 담은 객체!

```
var a = 1;
function outer() {
    console.log(a);

    function inner() {
        console.log(a);
        var a = 3;
    }

    inner();

    console.log(a);
}
outer();
console.log(a);
```

실제 코드를 보고 실행 컨텍스트가
원지 감을 잡아봅시다.

```
var a = 1;
function outer() {
  console.log(a);      1

  function inner() {
    console.log(a);    2
    var a = 3;
  }

  inner();

  console.log(a);      3
}
outer();
console.log(a);        4
```

실행 순서는 다음과 같습니다.

1, 2, 3, 4 순서로 출력이 됩니다.



```
var a = 1;
function outer() {
  console.log(a);

  function inner() {
    console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```

```
var a = 1;
function outer() {
  console.log(a);

  function inner() {
    console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```

1


```
var a = 1;  
function outer() {  
    console.log(a);
```

```
    function inner() {
```

```
        ↓ console.log(a);
```

2

```
        var a = 3;
```

```
    }
```

```
    inner();
```

```
    console.log(a);
```

```
}
```

```
outer();
```

```
console.log(a);
```

```
var a = 1;
function outer() {
  console.log(a);


  function inner() {
    console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```



3



```
var a = 1;
function outer() {
  console.log(a);

  function inner() {
    console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```

```
var a = 1;
function outer() {
  console.log(a);

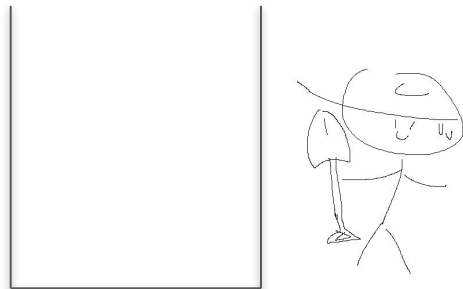
  function inner() {
    console.log(a);
    var a = 3;
  }


  inner();

  console.log(a);
}
outer();
console.log(a);
```

여기 제가 직접 만든 우물이 있습니다
이름은 콜스택 이라고

정했습니다 비어있어요!





```
var a = 1;
function outer() {
  console.log(a);

  function inner() {
    console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```

축하합니다. 최초로 전역 공간에 대한
컨텍스트가 제 콜스택에 생겼네요



```
var a = 1;
function outer() {
  console.log(a);

  function inner() {
    console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```



이제는 **outer**가 제일 맨 위에 있으니까

이 **outer**가 실행중이라는 것을 알 수 있습니다.



```
var a = 1;
function outer() {
  console.log(a);

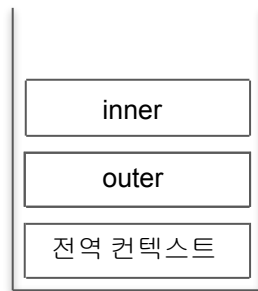
  function inner() {
    ↓ console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```

다시 inner를 호출해서 inner 실행
컨텍스트가 위에 쌓이면

이제 inner가 실행중이라는 걸 알 수
있겠죠.




```
var a = 1;
function outer() {
  console.log(a);

  function inner() {
    console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```





```
var a = 1;
function outer() {
  console.log(a);

  function inner() {
    console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```

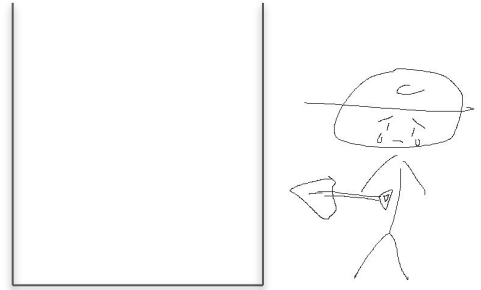


```
var a = 1;
function outer() {
  console.log(a);

  function inner() {
    console.log(a);
    var a = 3;
  }

  inner();

  console.log(a);
}
outer();
console.log(a);
```



inner

outer

전역 컨텍스트

inner

VariableEnvironment	environmentRecord (snapshot) outerEnvironmentReference (snapshot)
LexicalEnvironment	environmentRecord outerEnvironmentReference
ThisBinding	

outer

전역 컨텍스트

inner

VariableEnvironment

environmentRecord (snapshot)

outerEnvironmentReference (snapshot)

LexicalEnvironment

environmentRecord

outerEnvironmentReference

현재 환경과 관련된

ThisBinding

식별자 정보들

outer

전역 컨텍스트

inner

VariableEnvironment

변화 반영 x

environmentRecord (snapshot)

outerEnvironmentReference (snapshot)

LexicalEnvironment

변화 반영 O

environmentRecord

outerEnvironmentReference

식별자 정보 수집

ThisBinding

각 식별자의 '데이터' 추적

outer

전역 컨텍스트

영한 사전

able: 할 수 있는

apple: 사과

arrow: 화살, 화살표

Lexical Environment

어휘적/사전적 환경

영한 사전

실행컨텍스트 A 환경 사전

able: 할 수 있는

내부식별자 a : 현재 값은 undefined이다.

apple: 사과

내부식별자 b : 현재 값은 20이다.

arrow: 화살, 화살표

외부 정보 : D를 참조한다.

Lexical Environment

어휘적/사전적 환경

영한 사전

실행컨텍스트 A 환경 사전

able: 할 수 있는

내부식별자 a : 현재 값은 undefined이다.

apple: 사과

내부식별자 b : 현재 값은 20이다.

arrow: 화살, 화살표

외부 정보 : D를 참조한다.

Lexical Environment

어휘적/사전적 환경

실행컨텍스트를 구성하는 환경 정보들을
모아 사전처럼 구성한 객체.