코어 JS - 2단원

질문거리

1. 왜 eval()은 evil인가? 특징과 문제점에 대해서 설명
2. 실행 컨텍스트의 역할
3. 일반 함수와 화살표 함수의 실행 컨텍스트 차이

02. 실행 컨텍스트

I. 실행 컨텍스트란?

- 실행 컨텍스트: 실행할 코드에 제공할 환경 정보들을 모아놓은 객체
  - 흔히 실행 컨텍스트를 구성하는 방법은 '함수를 실행'하는 것뿐 + ES6 블록()에 의해서 O
- VariableEnvironment: 현재 컨텍스트 내의 식별자들에 대한 정보 + 외부 환경 정보
  - 선언 시점의 LexicalEnvironment의 스냅샷으로, 변경 사항은 반영 X
- LexicalEnvironment: 변경 사항이 실시간으로 반영됨
- ThisBinding: this 식별자가 바라봐야할 대상 객체.

II. VariableEnvironment

- 실행 컨텍스트 생성 시 VariableEnvironment(이하 VE)에 정보를 먼저 담은 다음, 그대로 복사해서 LexicalEnvironment(이하 LE)를 만들고 이후에는 LE를 주로 활용하게 됨
- 내부에는 environmentRecord(이하 eR)와 outer-EnvironmentReference(이하 oER)로 구성돼 있음

III. LexicalEnvironment

- 사전적 환경

1. environmentRecord와 호이스팅

- 현재 컨텍스트와 관련된 코드의 식별자 정보들이 저장됨 / 순서대로 수집
- eR은 현재 실행될 컨텍스트의 대상 코드 내에 **어떤 식별자들이 있는지**에만 관심 O, 각 식별자에 어떤 값이 할당될 것인지는 관심 X

- 호이스팅
  - 변수 - 선언부 / 할당부에서 선언부만 끌어올림
  - 함수 선언은 전체를 끌어올림

- 함수 선언문과 함수 표현식
  - 공통점: 함수를 새롭게 정의할 때 쓰이는 방식
  - 함수 선언문: function 정의부만 존재하고 별도의 할당 명령 X
  - 함수 표현식: 정의한 function을 별도의 변수에 할당하는 것. 보통 익명 함수 표현식을 뜻함.

- 함수 선언문은 전체를 호이스팅 ↔ 함수 표현식은 변수 선언부만 호이스팅
  - 함수를 다른 변수에 값으로써 할당한 것이 곧 함수 표현식
  - 상대적으로 함수 표현식이 안전함

2. 스코프, 스코프 체인, outerEnvironmentReference

- 스코프 체인(scope chain): 식별자의 유효범위를 안에서부터 바깥으로 차례로 검색해 나가는 것
- 이를 가능케 하는 것이 LexicalEnvironment의 두 번째 수집 자료인 outerEnvironmentReference임
- 스코프: 변수의 유효범위
- oER는 해당 함수가 선언된 위치의 LE를 참조함

- 어떤 변수에 접근 시
    - 현재 컨텍스트의 LE 탐색 → oER에 담긴 LE 탐색 → .. → 전역 컨텍스트의 LE까지 탐새갱도 해당 변수를 찾지 못하면 undefined 반환

- 전역 변수: 전역 컨텍스트의 LE에 담긴 변수
- 지역 변수: 그 밖의 함수에 의해 생성된 실행 컨텍스트의 변수들

IV. this

- this에는 실행 컨텍스트를 활성화하는 당시에 지정된 this가 저장됨.
- 함수를 호출하는 방법에 따라 그 값이 달라짐. 저장되지 않은 경우에는 전역 객체가 저장됨.