15장 let, const 키워드와 블록 레벨 스코프

15.1 var 키워드로 선언한 변수의 문제점

- 0. 머릿말
  - ES5까지 변수를 선언할 수 있는 유일한 방법은 var 키워드
- 1. 변수 중복 선언 허용
  - 기존에 변수가 있다.
    - 초기화 문 (값 할당 문)이 없을 경우 → 변수 선언문 무시 → 기존 값 유지
    - 초기화 문 (값 할당 문)이 있을 경우 → Var 키워드가 없는 것처럼 동작 → 기존 값 변경
- 2. 함수 레벨 스코프
  - var 키워드로 선언한 변수는 오로지 함수의 코드 블록만을 지역 스코프로 인정
  - 전역 변수를 남발할 가능성을 높인다.
- 3. 변수 호이스팅
  - 변수 선언문이 스코프의 선두로 끌어 올려진 것처럼 동작
    - 선언문 이전에 참조 할 수 있다.
    - 단, 할당문 이전에 참조하면 언제나 undefined 반환
  - 변수 선언문 이전에 변수를 참조하는 것은 호이스팅에 의해 에러를 발생시키진 않지만 프로그램 흐름상 맞지 않고, 가독성을 떨어뜨려 오류 발생시킬 여지를 남김 → 비추천

15.2 let 키워드

- 1. 변수 중복 선언 금지
  - var키워드에서는 에러 발생 X
  - let과 const는 에러 발생 O
- 2. 블록 레벨 스코프
  - 모든 코드 블록 { } (함수, if, for, while, try / catch 등)을 지역 스코프로 인정
- 3. 변수 호이스팅
  - let키워드로 선언한 변수는 변수 호이스팅이 발생하지 않는 것처럼 동작 → let 키워드로 선언한 변수를 선언문 이전에 참조하면 참조 에러가 발생한다
    - var키워드는 런타임 이전에 js엔진에 의해 암묵적으로 "선언 단계"와 "초기화 단계"가 한번에 진행 → 초기화단계때 undefined를 암묵적 초기화
    - let키워드는 "선언단계", "초기화 단계" 분리되어 진행되기때문 → 초기화 단계는 변수 선언문에 도달했을 때 실행
  - 일시적 사각지대 (TDZ = Temporal Dead Zone) → 스코프의 시작 지점부터 초기화 시작 지점까지 변수를 참조할 수 없는 구간 → 예제
  - 호이스팅이 발생하지 않는 것처럼 보이지만 그렇지 않다. → 예제 → let 키워드로 선언한 변수의 경우 호이스팅이 발생하지 않는다면 위 예제는 전역 변수 foo의 값을 출력해야 한다. 하지만 let 키워드로 선언한 변수도 여전히 호이스팅이 발생하기 때문에 참조 에러가 발생하는 것이다.
  - 자바스크립트 ES6에서 도입된 let, const 를 포함해서 모든 선언 (var, let, const, function, function*, class 등)을 호이스팅 한다. → 단, let, const, class를 사용한 선언문은 호이스팅이 발생하지 않는 것처럼 동작
- 4. 전역 객체와 let
  - Let 키워드로 선언한 전역 변수는 전역 객체(window)의 프로퍼티가 아니다. → Let 전역변수는 보이지 않는 개념적인 블록(전역 렉시컬 환경의 선언전 환경 레코드) 내에 존재 → 실행 컨텍스트와 관련

15.3 const 키워드

- 0. 머릿말
  - 상수를 선언하기 위해 사용 → But, 반드시 상수만을 위해 사용 X
  - const 키워드 특징은 let 키워드와 대부분 동일 → But, 다른 차이점도 있다. 그 중심으로 살펴보자
- 1. 선언과 초기화
  - const 키워드로 선언한 변수는 반드시 선언과 동시에 초기화해야 한다. → const foo; // SyntaxError
  - let과 마찬가지로 블록 레벨 스코프, 변수 호이스팅이 발생하지 않는 것처럼 동작
- 2. 재할당 금지
  - let이나 var와 달리 재할당 금지 → TypeError 발생
- 3. 상수
  - 원시 값을 할당할 경우 변수 값 변경 X → 상수를 표현하는데 사용
    - 상태유지, 가독성
    - 유지보수의 편의
  - 상수 이름은 대문자로 선언해 상수임을 명확히 나타냄 → 여러 단어로 이루어진 경우 언더스코어(_)로 구분
- 4. const 키워드와 객체
  - const 키워드로 선언된 변수에 객체를 할당한 경우 값 변경 가능 → 재할당 없이도 직접 변경이 가능하기 때문
  - const 키워드는 재할당을 금지할 뿐 "불변"을 의미하진 않는다.
    - 객체의 프로퍼티 동적 생성, 수정, 삭제를 통해 객체 변경 가능
    - 대신 변수에 할당된 참조 값은 변경 X

15.4 var vs. let vs. const

- 변수 선언에 기본적으로 const를 사용하고 let은 재할당이 필요한 경우에 한정해 사용
- const는 의도치 않은 재할당을 방지해서 안전
- ES6를 사용한다면 var 사용 X
- 재할당 필요한 경우 let => 변수의 스코프는 최대한 좁게
- 읽기 전용으로 사용하는 원시 값, 객체에는 const 사용
- 변수를 선언할 때는 일단 const 키워드 사용 → 추후에 재할당이 필요한 경우 let으로 변경해도 충분