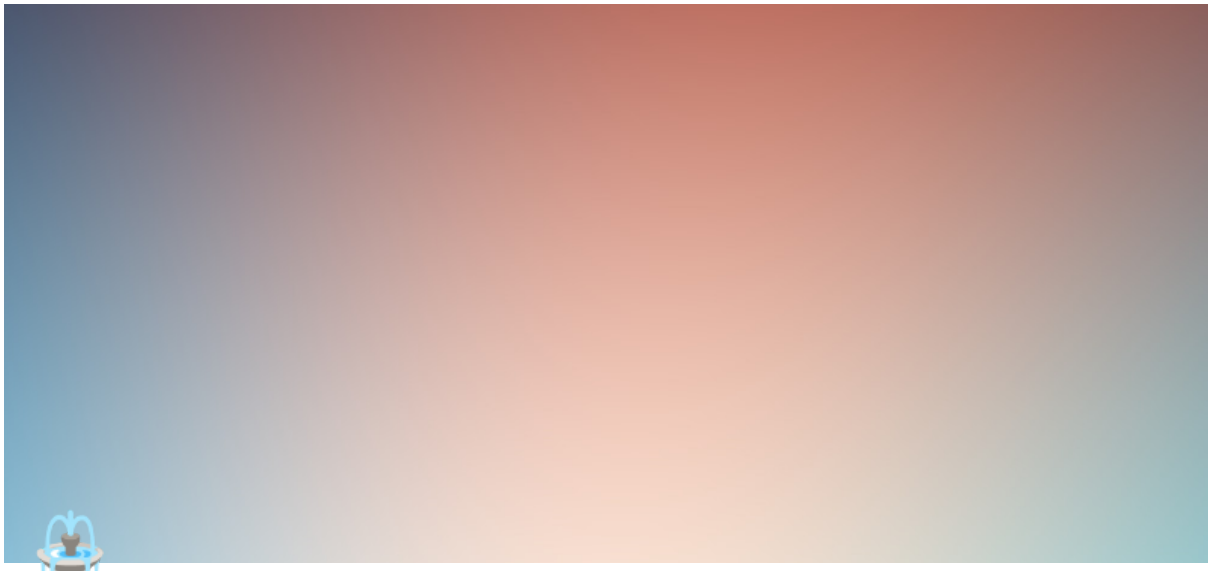

스코프

- 스코프란?
  - 식별자 접근 규칙에 따른 유효 범위
  - JS가 어떻게 변수를 식별하는지에 대한 규칙이다.
- 스코프는 보통 두 가지 방식으로 작동한다.
  - 렉시컬 스코프(Lexical Scope)
  - 동적 스코프(Dynamic Scope)
- JS에서는 렉시컬 스코프, Perl이나 Bash에서는 동적 스코프를 따른다.

렉시컬 스코프

- 렉시컬 스코프란 렉싱 타임 (Lexing Time)에 정의되는 스코프이다.
  - 프로그래머가 코드를 짤 때 변수와 스코프 블록을 어디서 작성하는가에 기초해 확정된다.

```
var number = 1;

function a() {
  var number = 10;
  b();
}
```

```
function b() {
  console.log(number);
}

a(); // 1
b(); // 1
```

- 컴파일러는 토크나이징 (혹은 렉싱)이라 불리는 작업을 시작한다.
  - 이 과정에서 소스 코드 문자열을 분석하여 토큰에 의미를 부여한다.
- 10, 1이 아니라 1, 1이 출력된 이유는 함수의 호출이 아닌 선언된 위치에 따라 상위 스코프가 결정되었기 때문이다.

함수 기반 스코프

- 새로운 함수가 생성될 때마다 새로운 스코프가 발생한다.

```
function a() {
  const secret = '12345';
}
secret; // ReferenceError
```

- 함수 `a()` 은 함수 스코프, `secret`은 전역 스코프를 가지고 있다.
- 스코프가 다르기 때문에 접근이 불가능하다.

var

- 함수 단위 스코프
- 호이스팅에 영향을 받는다
- 런타임에 값이 정의 된다

블록 기반 스코프

- 블록이 생성될 때마다 새로운 스코프가 형성된다.

```
function loop() {
  for (let i = 0; i < 5; i++) {
    console.log(i);
  }
  console.log('final', i);
}
loop(); /* ReferenceError: i is not defined */
```

- 위의 코드에서 i를 var로 선언하면 final이 5로 출력이 되지만, let으로 선언했기 때문에 블록 스코프 밖에서 접근이 불가능하다.

let

- 블록 단위 스코프

const

- 블록 단위 스코프
- 선언된 값 고정

스코프 체이닝

- 변수나 함수를 스코프 내부에서 찾지 못하면 바깥의 스코프를 체이닝해가면서 탐색하게 된다.
- 중첩 스코프를 탐사할 때 사용되는 간단한 규칙은 다음과 같다.
  - 엔진은 현재 스코프에서 변수를 찾기 시작하고, 찾지 못하면 한 단계씩 올라간다.
  - 최상위 글로벌 스코프에 도달하면 변수를 찾았든, 못 찾았든 검색을 멈춘다.

```
function foo(a) {
    console.log(a + b);
    b = a;
}
foo(2); // ReferenceError
```

- 검색을 통해서 찾지 못한 변수는 `ReferenceError`를 발생한다.
- `strict mode`로 동작하지 않는다면 글로벌 스코프는 엔진이 검색하는 이름을 가진 새로운 변수를 생성해서 엔진에게 넘겨준다.