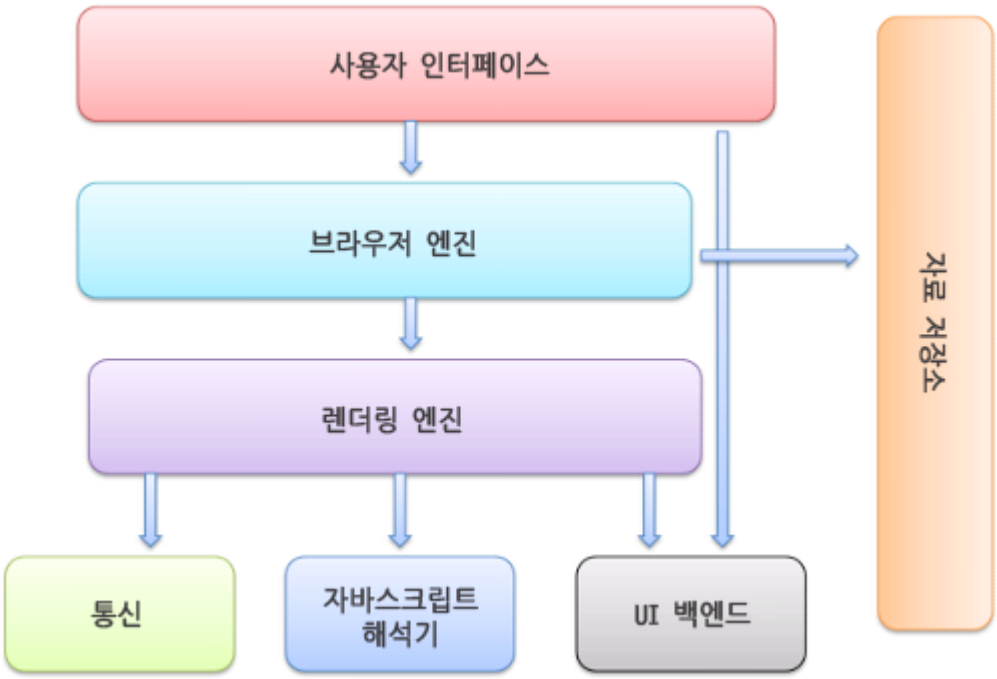
브라우저 동작원리 + JS동작원리

브라우저 정의

웹 브라우저는 동기적으로 HTML+CSS JAVASCRIPT언어를 해석하여 내용을 화면에 보여주는 응용 소프트웨어 입니다.

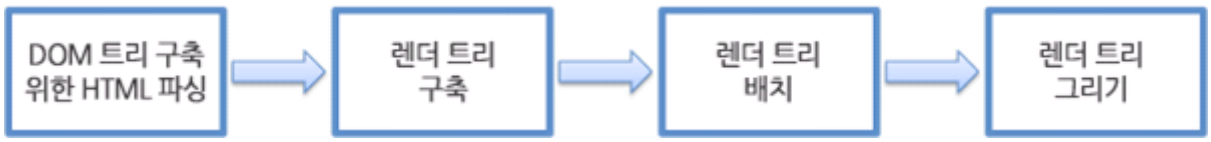
브라우저 구조



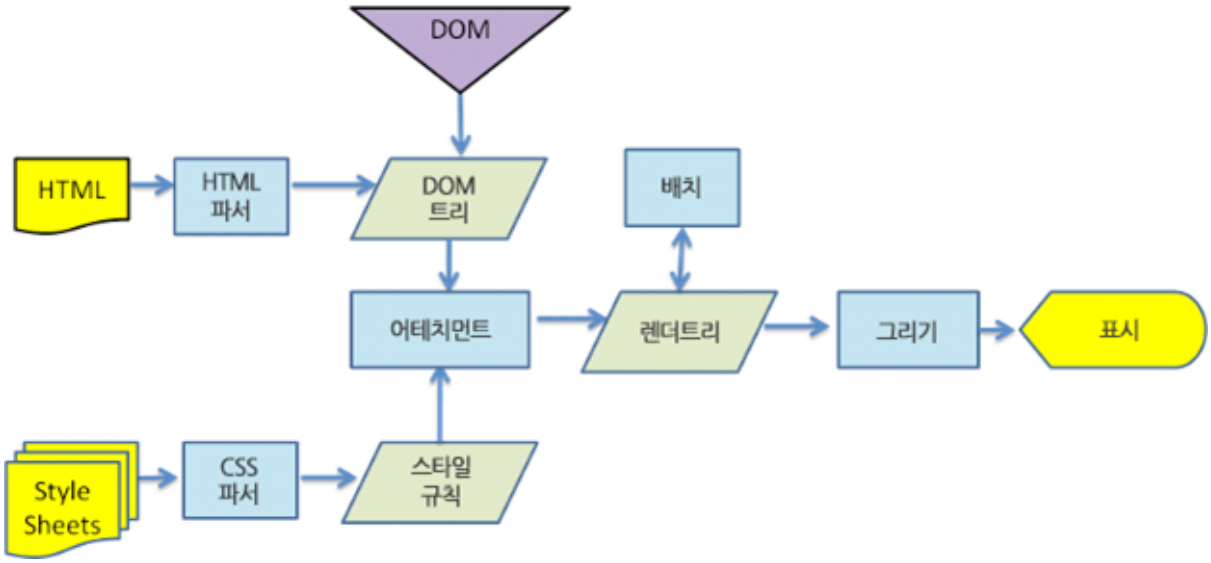
렌더링 엔진 종류

```
-moz-border-radius: 1em; // 파이어폭스 브라우저에 적용
-ms-border-radius: 2em; // 익스플로어에 적용, 보통 생략
-o-border-radius: 3em; // 오페라에 적용
-webkit-border-radius: 4em; // 구글, 사파리 브라우저에 적용
```

렌더링 엔진 동작 과정

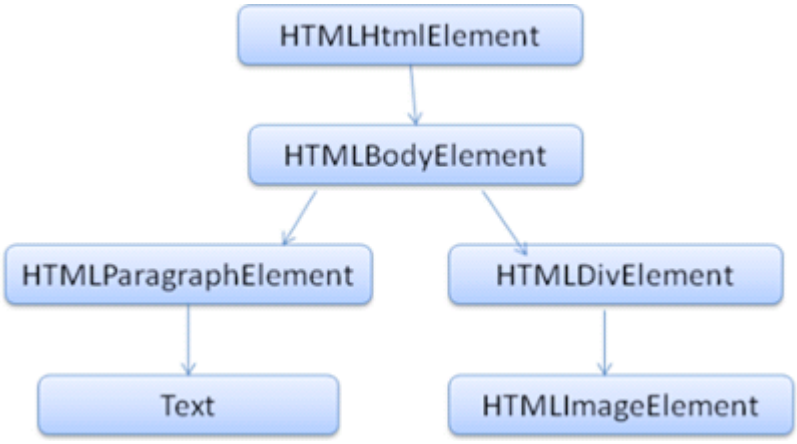


1. HTML 문서 파싱 DOM 트리 구축
2. CSS파일과 스타일요소 파싱
    a. https://velog.io/@elrion018/브라우저-작동방식-탐구-HTML-파서Parser-구현하기
3. DOM 트리 + 2의 결과물 = 렌더트리 구축
4. 렌더 트리 각 노드에 대해 화면 상에서 배치할 곳 결정
5. UI 백앤드에서 렌더트리의 각 노드 그림

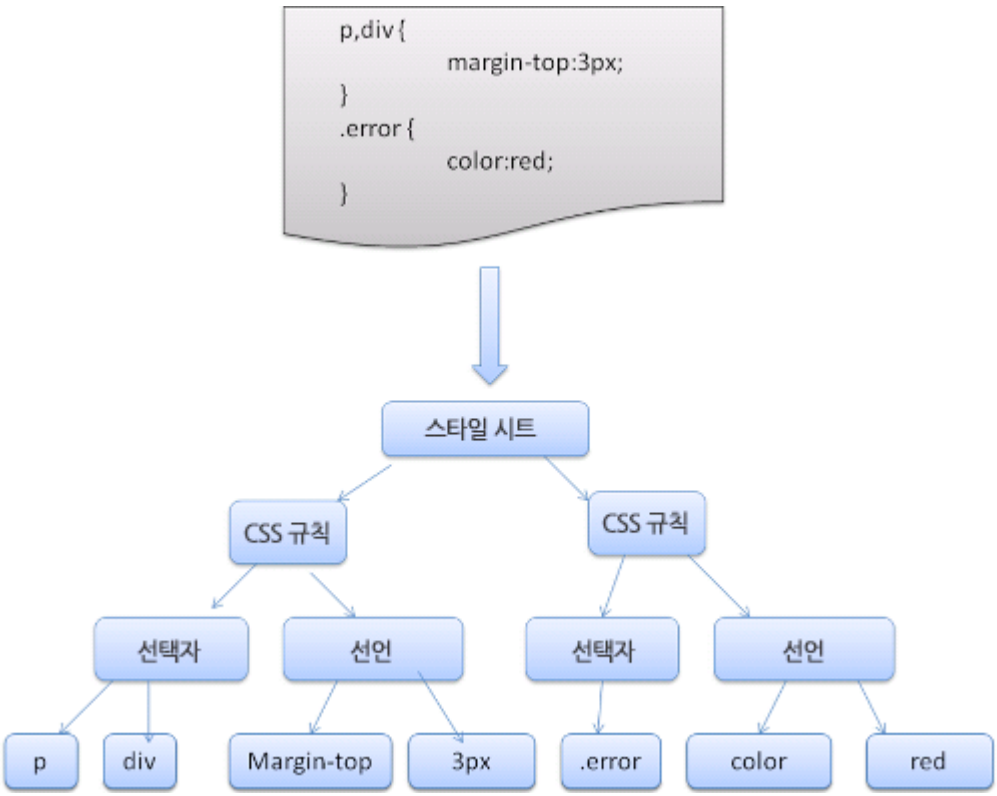


1. HTML 문서 파싱 DOM 트리 구축

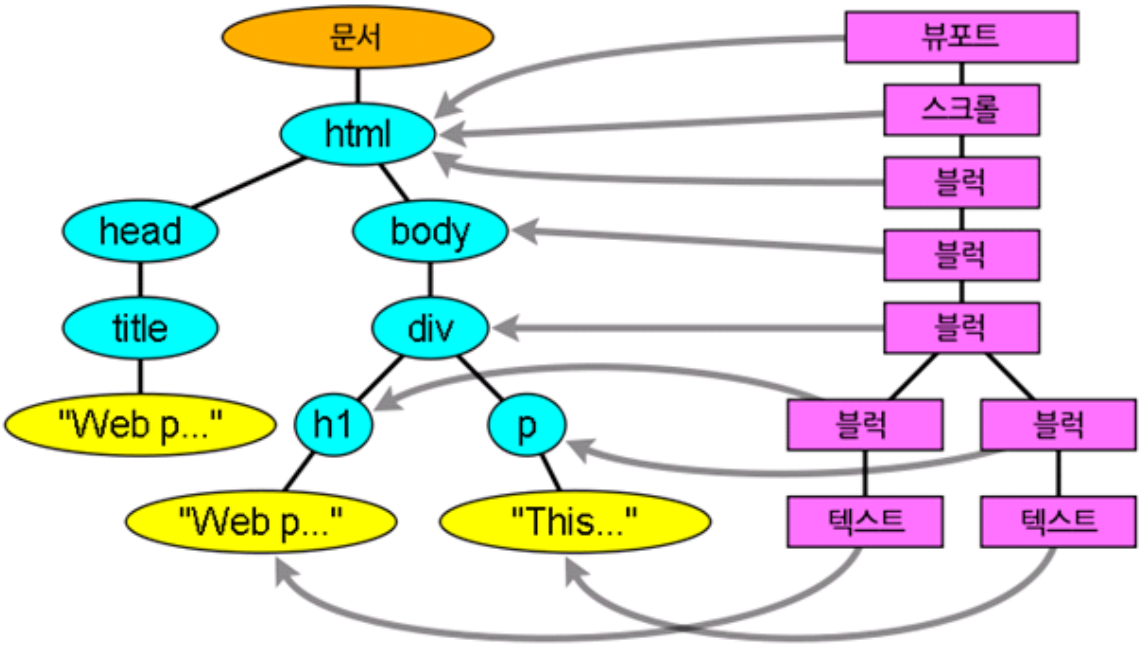
```
<html>
  <body>
   <p>Hello World</p>
   <div><img src="example.png" /></div>
  </body>
</html>
```



2. CSSOM(Css Object Model)을 생성



3. 렌더 트리 (DOM + CSSOM)을 생성



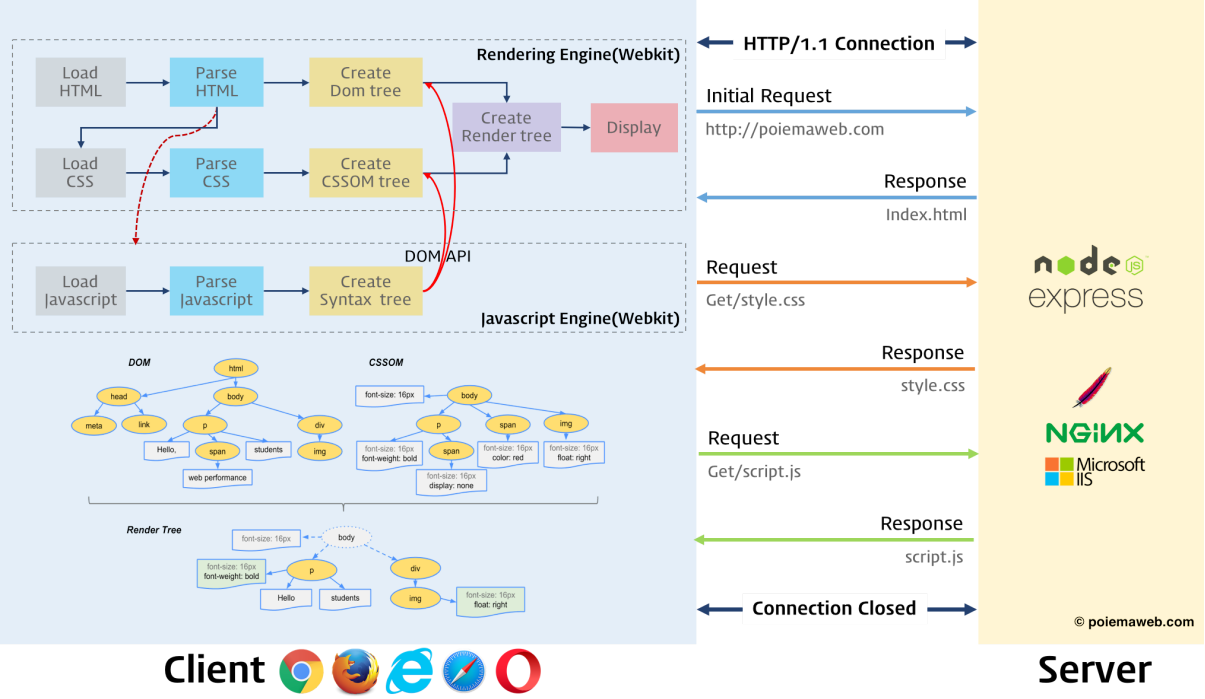
4. 렌더 트리를 배치 한다.

어느 공간에 위치해야 할지 각 객체들에게 위치와 크기를 결정

5. 렌더 트리를 그림

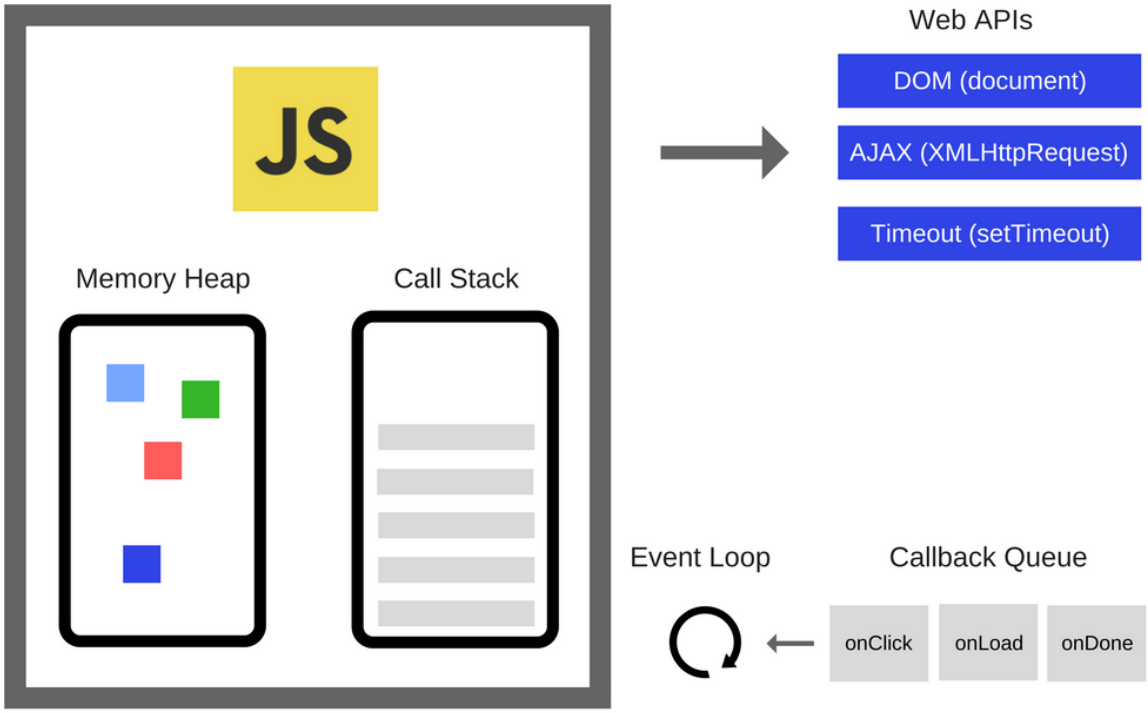
렌더트리의 각 객체를 화면의 픽셀 값으로 나타냄

브라우저 동작 플로우



JS 동작원리

- **Memory Heap** : 메모리 할당이 일어나는 곳
- **Call Stack** : 코드 실행에 따라 호출 스택이 쌓이는 곳



호출 스택

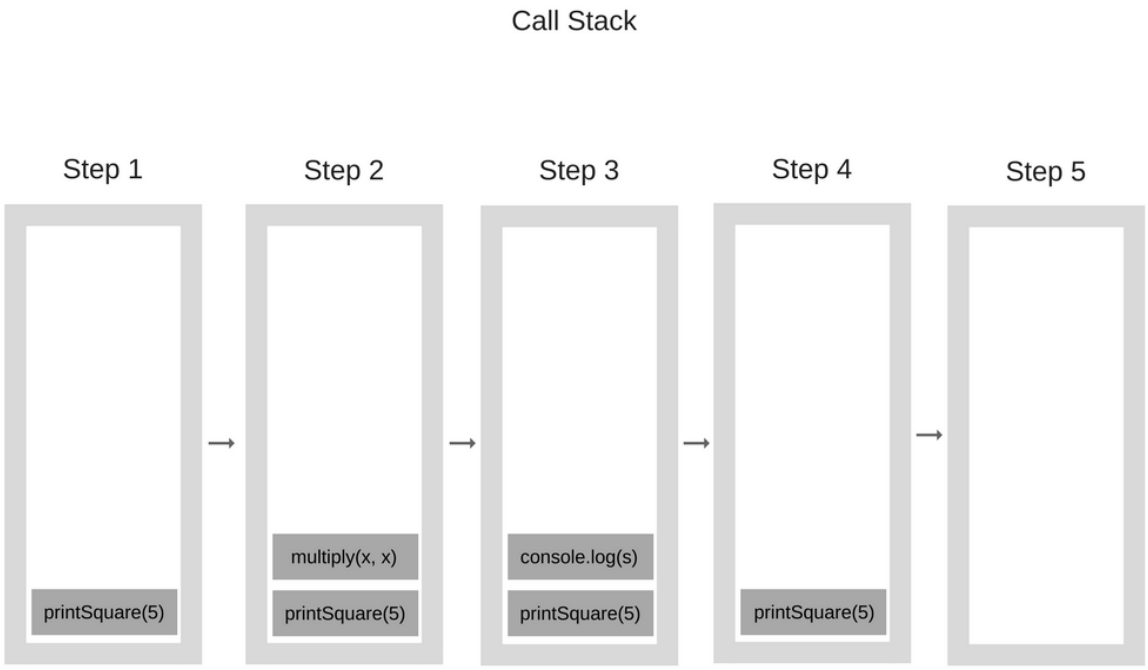
자바스크립트는 기본적으로 싱글 쓰레드 기반 언어입니다. 호출 스택이 하나라는 소리죠. 따라서 한 번에 한 작업만 처리할 수 있습니다.

호출 스택은 기본적으로 우리가 프로그램 상에서 어디에 있는지를 기록하는 자료구조입니다.

만약 함수를 실행하면(실행 커서가 함수 안에 있으면), 해당 함수는 호출 스택의 가장 상단에 위치하는 거죠. 함수의 실행이 끝날 때(리턴 값을 돌려줄 때), 해당 함수를 호출 스택에서 제거합니다. 그게 스택의 역할입니다.

```
function multiply(x, y) {
    return x * y;
}
function printSquare(x) {
    var s = multiply(x, x);
```

```
    console.log(s);
}
printSquare(5);
```



호출 스택의 각 단계를 스택 프레임(Stack Frame)이라고 합니다.

그리고 보통 예외가 발생했을 때 콘솔 로그 상에서 나타나는 스택 트레이스(Stack Trace)가 오류가 발생하기까지의 스택 트레이스들로 구성됩니다. 간단히 말해서 에러가 났을 때의 호출 스택의 단계를 의미하는 거죠.

```
function foo() {
    throw new Error('SessionStack will help you resolve crashes :)');
}
function bar() {
    foo();
}
function start() {
    bar();
}
start();
```

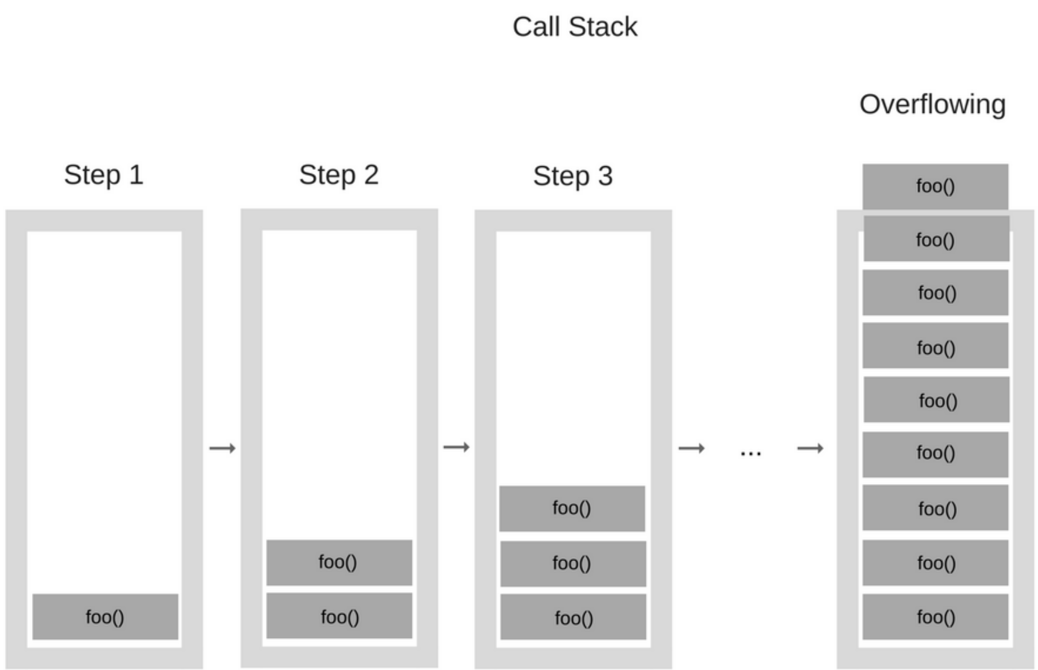
```
Uncaught Error: SessionStack will help you resolve crashes :) foo.js:2
      at foo (foo.js:2)
      at bar (foo.js:6)
      at start (foo.js:10)
      at foo.js:13
```

호출 스택이 최대 크기가 되면 **“스택 날려 버리기”** 가 일어납니다.

이는 반복문 코드를 광범위하게 테스트하지 않고 실행했을 때 자주 발생합니다. 아래 코드를 보시면

```
function foo() {
    foo();
}
foo();
```

엔진에서 이 코드를 실행할 때, foo()에 의해서 foo 함수가 호출됩니다. 그런데 여기서 foo 함수가 반복적으로 자신을 다시 호출하는 재귀 호출을 수행합니다. 그러면 매번 실행할 때마다 호출 스택에 foo() 가 쌓이게 됩니다. 아래와 같이 말이죠.



그러다가 특정 시점에 함수 호출 횟수가 호출 스택(Call Stack)의 최대 허용치를 넘게 되면 브라우저가 아래와 같은 에러를 발생시킵니다.

```
▶Uncaught RangeError: Maximum call stack size exceeded
```

싱글 스레드 기반 코딩은 멀티 스레드 환경에서 제기되는 복잡한 문제나 시나리오를 고민하지 않아도 되기 때문에 상당히 쉽습니다. 예를 들면, 데드락 같은 게 있겠죠.

허나 싱글 스레드에서 코드를 실행하는 건 상당히 제약이 많습니다. 한 개의 호출 스택을 갖고 있는 자바스크립트의 실행이 느려지면 어떻게 될까요?

동시성(Concurrency) & 이벤트 루프(Event Loop)

호출 스택에 처리 시간이 어마어마하게 오래 걸리는 함수가 있으면 무슨 일이 발생할까요? 예를 들어, 브라우저에서 자바스크립트로 매우 복잡한 이미지 프로세싱 작업을 한다고 합시다.

이게 대체 어때서? 라고 의문이 생길지도 모르지만, 여기서 문제는 호출 스택에서 해당 함수가 실행되는 동안 브라우저는 아무 작업도 못하고 대기 상태가 된다는 겁니다. 이 말뜻은 **브라우저는 페이지를 그리지도 못하고, 어느 코드도 실행을 못한다는 거죠. 말 그대로 그냥 가만히 있는 겁니다.** 만약 매끄럽고 자연스러운 화면 UI를 가진 앱을 원한다면 위 경우는 문제가 됩니다.

문제는 이뿐만이 아닙니다. 브라우저가 호출 스택의 정말 많은 작업들을 처리하다 보면 화면이 아마 오랫동안 응답하지 않게 됩니다. 이 경우에 대부분의 브라우저가 아래와 같은 에러를 띄우면서 페이지를 종료할 건지 물어봅니다.

자 그렇다면 어떻게 페이지 렌더링 동작을 방해하지 않고 브라우저의 응답도 끊지 않으면서 연산량이 많은 코드를 실행할 수 있을까요?

정답은 바로 **비동기 콜백** 입니다.