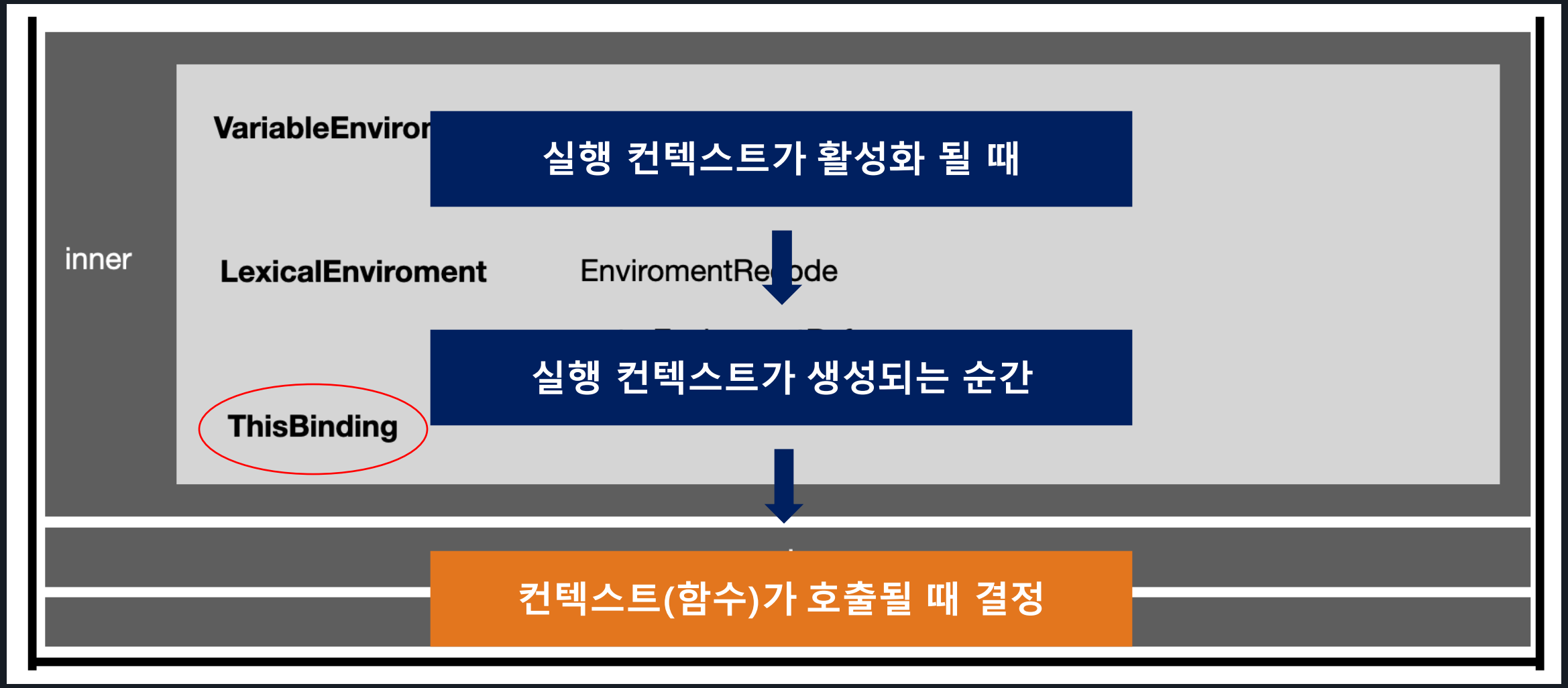
Core JavaScript

3. this

목표

1. 상황별로 this가 어떻게 달라지는지
2. 왜 그렇게 되는지
3. 예상과 다른 대상을 바라보고 있을 경우 그 원인이 무엇인지


VariableEnviror
실행 컨텍스트가 활성화 될 때
inner
LexicalEnviroment
EnviromentRe ode
실행 컨텍스트가 생성되는 순간
ThisBinding
컨텍스트(함수)가 호출될 때 결정

➡ this란 자신을 호출한 객체를 가리키는 자기 참조 변수이다.

this 바인딩이란?

Bind: 묶다, 가두다, 엮다

바인딩이란 식별자와 값을 연결하는 과정을 의미한다.
예를 들어, 변수 선언은 변수 이름(식별자)과 확보된 메모리 공간의 주소를 바인딩하는 것이다. this 바인딩은 this와 this가 가리킬 객체를 바인딩하는 것이다.

호출하는 방식에 따라 결정되는 this

1. 전역 공간에서 호출할 때
2. 함수로서 호출할 때
3. 메서드로서 호출할 때
4. 콜백 함수에서 호출할 때
5. 생성자 함수로서 호출할 때

호출하는 방식에 따라 결정되는 this

1. 전역 공간에서 호출할 때 ➡ window / global
2. 함수로서 호출할 때
3. 메서드로서 호출할 때
4. 콜백 함수 호출할 때
5. 생성자 함수로서 호출할 때

1. 전역 공간에서

➡ 전역 공간에서의 this는 전역 객체를 가리킨다.

왜? 개념상 전역 컨텍스트를 실행하는 주체가 바로 전역 객체(window, global)이기 때문에

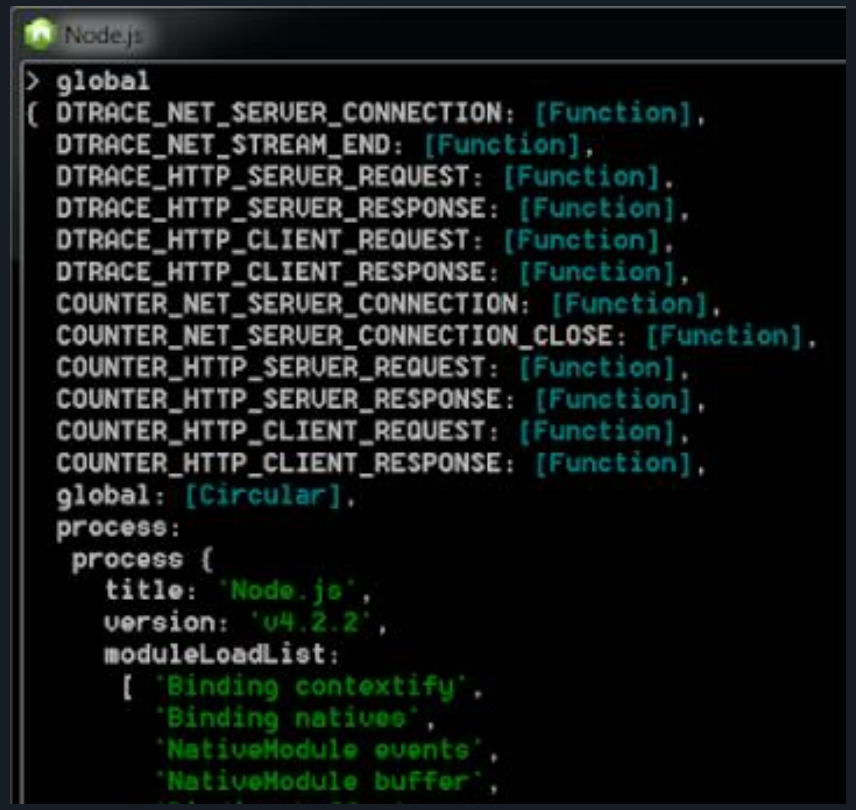
전역 객체는 자바스크립트가 실행되는 환경, 즉 런타임에 따라서 전역 객체의 정보가 달라진다. (브라우저에서 this를 출력하면 window라는 객체가 나오고 node.js에서 this를 출력하면 global이라고 하는 객체가 나온다.)

1. 전역 공간에서

➡ **전역 공간에서의 this는 전역 객체를 가리킨다.**

브라우저

Node.js



1. 전역 공간에서

```
> var a = 1
  console.log(a)
  console.log(window.a)
  console.log(this.a)
  1                                   VM653:2
  1                                   VM653:3
  1                                   VM653:4
```

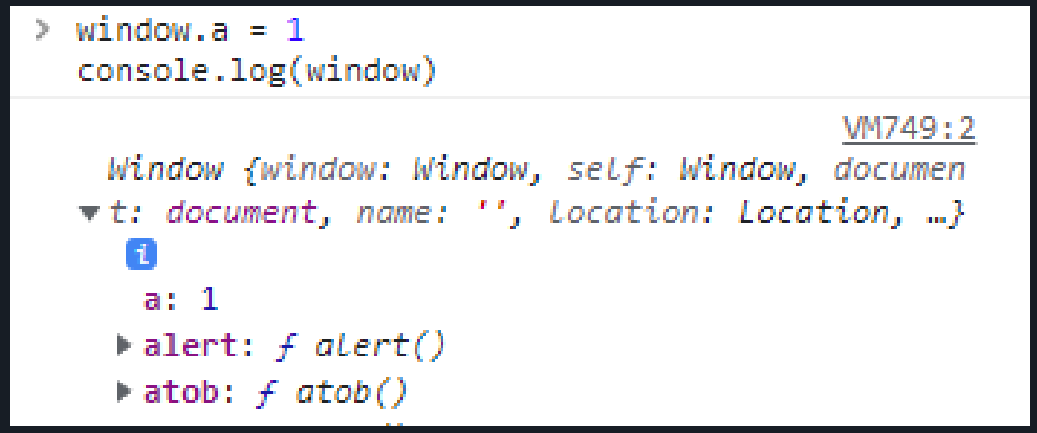
➡ 왜? 자바스크립트의 모든 변수는 특정 객체의 프로퍼티로서 동작하기 때문에

사용자가 var 연산자를 이용해 변수를 선언하면 실제 자바스크립트 엔진은 어떤 객체의 프로퍼티로 인식하는 것이다.

1. 전역 공간에서

```
> var a = 1;
  console.log(window)
                                                    VM1000:2
  Window {window: Window, self: Window, documen
▼ t: document, name: '', location: Location, …}
    i
     a: 1
   ▶ alert: ƒ alert()
   ▶ atob: ƒ atob()
   ▶ blur: ƒ blur()
```

1. 전역 공간에서



➡ 따라서 a를 출력한 값과, window.a를 출력한 값, this.a를 출력한 값이 모두 같다.

차이점? delete 연산자 사용 가능성 및 호이스팅 여부

호출하는 방식에 따라 결정되는 this

1. 전역 공간에서 호출할 때 — window / global
2. 함수로서 호출할 때 ➡ window / global
3. 메서드로서 호출할 때
4. 콜백 함수 호출할 때
5. 생성자 함수로서 호출할 때

2. 함수 호출시

```
> function a() {
      console.log(this);
  }
  a();
```

```
> console.log(window)
                                                    VM1109:1
  Window {window: Window, self: Window, documen
  t: document, name: '', location: Location, …}
    alert: ƒ alert()
    atob: ƒ atob()
    blur: ƒ blur()
    btoa: ƒ btoa()
    cancelAnimationFrame: ƒ cancelAnimationFrame(
    cancelIdleCallback: ƒ cancelIdleCallback()
    captureEvents: ƒ captureEvents()
    chrome: {loadTimes: ƒ, csi: ƒ}
    clearInterval: ƒ clearInterval()
    clearTimeout: ƒ clearTimeout()
    clientInformation: Navigator {vendorSub: '',
    close: ƒ close()
    closed: false
    confirm: ƒ confirm()
    createImageBitmap: ƒ createImageBitmap()
    credentialless: false
    crossOriginIsolated: false
    crypto: Crypto {}
```

2. 함수 호출시

```
function b() {
    function c() {
        console.log(this);
    }
    c();
}
b();
```

※ 설계상의 오류

ES6부터 arrow function

2. 함수 호출시

```
> var d = {
      e: function() {
          function f() {
              console.log(this);
          }
          f();
      }
  }
  d.e();
```

2. 함수 호출시

```
> var d = {
      e: function() {
          var f = () => {
              console.log(this);
          }
          f();
      }
  }
  d.e();
```

```
▼ {e: ƒ}
  ▶ e: ƒ ()
  ▶ [[Prototype]]: Object
```

2. 함수 호출시

함수 vs 메서드

➡ 독립성

함수	메서드
그 자체로 독립적인 기능 수행	자신을 호출한 대상 객체에 관한 동작 수행

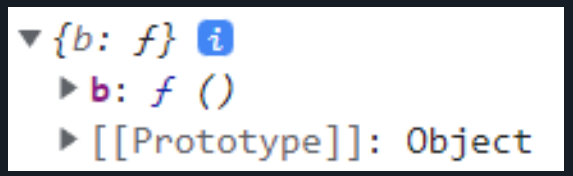
➡ 자바스크립트에서 이를 구분하는 거의 유일한 기능이 this

호출하는 방식에 따라 결정되는 this

1. 전역 공간에서 호출할 때 — window / global
2. 함수로서 호출할 때 — window / global
3. 메서드로서 호출할 때 ➡ 메서드 호출 주체(메서드명 앞)
4. 콜백 함수 호출할 때
5. 생성자 함수로서 호출할 때

3. 메서드 호출시

```
var func = function () {
    console.log(this);
};
func(); // Window {...}

var a = {
    b: func
};
a.b();
```



b함수를 a객체의 '메서드'로서 호출했다.

3. 메서드 호출시

```
> function b() {
      function c() {
          console.log(this);
      }
      c();
  }
  b();
```

```
var a = {
    b: function() {
        console.log(this);
    }
}
a.b();
```

3. 메서드 호출시

```
var a = {
    b: {
        c: function() {
            console.log(this);
        }
    }
}
a.b.c();
```

```
▼{c: f} i
  ▶c: f ()
  ▶[[Prototype]]: Object
```

3. 메서드 호출시

```
obj.func();
obj['func']();

person.info.getName();
person.info['getName']();
person['info'].getName();
person['info']['getName']();
```

3. 메서드 호출시

```
var a = 10; // 전역 변수
var obj = {
    a: 20,
    b: function() {
        console.log(this.a); // 20

        function c() {
            console.log(this.a);
        }                       // 10
        c();
    }
}
obj.b();
```

3. 메서드 호출시

```
var a = 10; // 전역 변수
var obj = {
    a: 20,
    b: function() {
        console.log(this.a); // 20

        function c() {
            console.log(this.a);
        }                       // 10
        c();
    }
}
obj.b();
```

this === obj 같게 하려면?

3. 메서드 호출시

```
var a = 10; // 전역 변수
var obj = {
    a: 20,
    b: function() {
        var self = this;
        console.log(this.a);

        function c() {
            console.log(self.a);
        }
        c(); // 20
    }
}
obj.b(); // 20
```

_this, that 등

ES6 부터

```
var a = 10; // 전역 변수
var obj = {
    a: 20,
    b: function() {
        console.log(this.a); // 20

        const c = () => {
            console.log(this.a);
        }                    // 20
        c();
    }
}
obj.b();
```

호출하는 방식에 따라 결정되는 this

1. 전역 공간에서 호출할 때	window / global
2. 함수로서 호출할 때	window / global
3. 메서드로서 호출할 때	메서드 호출 주체(메서드명 앞)
4. 콜백 함수 호출할 때 ➡	기본적으로는 함수 내부에서와 동일
5. 생성자 함수로서 호출할 때	

4. 콜백함수 호출시

콜백 함수란?

➡ 함수 a의 제어권을 다른 함수(또는 메서드) b에게 넘겨주는 경우에 함수 a를 콜백함수라고 한다.

➡ 다른 코드(함수)의 인수로 넘겨주는 함수

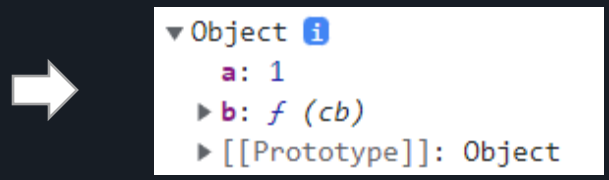
콜백 함수를 넘겨받은 코드는 이 콜백 함수를 필요에 따라 적절한 시점에 실행할 것이다.

4. 콜백함수 호출시

```
var callback = function() {
    console.dir(this); // Window
};

var obj = {
    a: 1,
    b: function(cb) {
        cb();
    }
};
obj.b(callback);
```

4. 콜백함수 호출시

```
var callback = function() {
    console.dir(this);
};

var obj = {
    a: 1,
    b: function(cb) {
        cb.call(this);
    }
};
obj.b(callback);
```



정리 : 콜백함수 내부에서의 this는 콜백함수 자체(callback)에서 어떻게 할 수 있는 것이 아니라 콜백함수를 넘겨받는 대상이 매개변수로 넘겨받은 콜백함를 어떻게 처리하느냐에 this가 다르게 나타난다.

결론 : 그때그때 다르다.

4. 콜백함수 호출시

대표적인 콜백 함수:

1. setTimeout() 함수
2. forEach() 함수
3. addEventListener()

4. 콜백함수 호출시

대표적인 콜백 함수:

1. setTimeout() 함수
2. forEach() 함수
3. addEventListener()

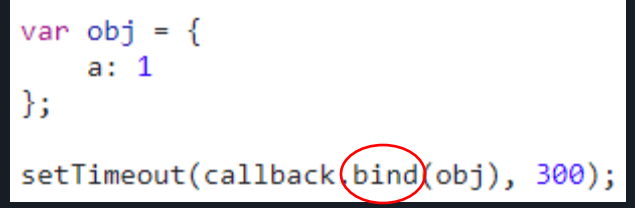
```
var callback = function() {
    console.dir(this); // Window
};

setTimeout(callback, 300);
```

➡ setTimeout은 내부에서 콜백 함수를 호출할 때 따로 this를 지정하지 않는다.

4. 콜백함수 호출시

대표적인 콜백 함수:

1. setTimeout() 함수
2. forEach() 함수
3. addEventListener()



```
var callback = function() {
    console.dir(this); // Object (obj)
};

var obj = {
    a: 1
};

setTimeout(callback.bind(obj), 300);
```

➡ setTimeout은 내부에서 콜백 함수를 호출할 때 따로 this를 지정하지 않는다.

4. 콜백함수 호출시

대표적인 콜백 함수:

1. setTimeout() 함수
2. forEach() 함수
3. addEventListener()

```
var callback = function(x) {
    console.log(this, x); // Window
};

[1, 2, 3, 4, 5].forEach((x) =>
callback(x));
```

➡ forEach도 내부에서 콜백 함수를 호출할 때 따로 this를 지정하지 않는다.

4. 콜백함수 호출시

대표적인 콜백 함수:

1. **setTimeout() 함수**
2. **forEach() 함수**
3. **addEventListener()**

```
document.body.innerHTML += '<div id="a"> 클릭하세요
</div>';

document.getElementById('a').addEventListener('click',
function() {
    console.dir(this);
});
```

```
▼div#a
    accessKey: ""
    align: ""
    ariaAtomic: null
```

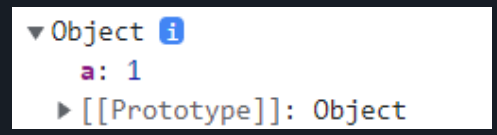
- 만약 addEventListener 함수가 이벤트 발생시 별도로 this를 정의해놓은게 없다면 전역객체가 나와야 된다. 하지만, addEventListener는 콜백함수를 처리할 때, this가 이벤트가 발생한 그 타겟 대상(돔) 엘리먼트로 하도록 정의가 되어있다.

4. 콜백함수 호출시

대표적인 콜백 함수:

1. setTimeout() 함수
2. forEach() 함수
3. addEventListener()

```
document.body.innerHTML += '<div id="a"> 클릭하세요
</div>';

var obj = {a: 1};

document.getElementById('a').addEventListener('click
', function() {
    console.dir(this);
}.bind(obj));
```



4. 콜백함수 호출시

이벤트 수신기 내부의 this 값

비슷한 요소 다수의 이벤트를 모두 처리할 수 있는 범용 수신기를 정의하는 경우, 부착된 요소의 참조를 가져와야 하는 상황이 종종 발생합니다.

`addEventListener()`를 사용해 요소에 수신기를 부착하게 되면 수신기 내부의 `this` 값은 대상 요소를 가리키게 되며, 이는 수신기가 매개변수로 받게 되는 이벤트 객체의 `currentTarget` 속성과 같습니다.

```
my_element.addEventListener('click', function (e) {
  console.log(this.className)           // my_element의 className 기록
  console.log(e.currentTarget === this) // `true` 기록
})
```

다만 화살표 함수는 스스로의 `this` 맥락을 가지지 않는다는 점을 기억해야 합니다.

```
my_element.addEventListener('click', (e) => {
  console.log(this.className)           // 경고: `this`가 `my_element`가 아님
  console.log(e.currentTarget === this) // `false` 기록
})
```

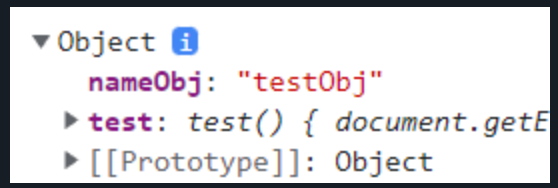
4. 콜백함수 호출시

```
> document.body.innerHTML += '<div id="a"> 클릭
</div>';

document.getElementById('a').addEventListener
("click", () => {
    console.dir(this);
});
```

```
▼ Window
  ▶ alert: ƒ alert()
  ▶ atob: ƒ atob()
```

4. 콜백함수 호출시

```
document.body.innerHTML += '<div id="a"> 클릭 </div>';

const myObj = {
    nameObj: "testObj",
    test() {
        document.getElementById('a').addEventListener("click", () => {
            console.dir(this);
        });
    }
}
myObj.test();
```



4. 콜백함수 호출시

정리

1. 기본적으로는 함수의 this와 같다. (대부분 전역 객체를 가리킨다.)
2. 제어권을 가진 함수가 콜백의 this를 지정해둔 경우도 있다. Ex) addEventListener
3. addEventListener 같은 경우에는 함수로서 this를 호출하면 메서드 앞 엘리먼트를 가리키고 화살표 함수를 사용하면 오히려 전역 객체를 가리킨다.
4. this를 바인딩해서 직접 지정해줄수 있다.
5. 콜백 함수에서의 this는 '무조건 이거다!'라고 정의할 수 없다.

this는 호출하는 방식에 따라 동적으로 결정

1. 전역 공간에서	window / global (전역 객체)
2. 함수 호출시	window / global (전역 객체)
3. 메서드 호출시	메서드 호출 주체 (메서드명 앞)
4. Callback 함수 호출시	기본적으로는 함수내부에서와 동일
5. 생성자 함수 호출시	인스턴스 객체 자신

5. 생성자 함수 호출시

생성자 함수란?

➡ New 연산자와 함께 호출하여 객체(인스턴스)를 생성하는 함수를 말한다.

생성자 함수	인스턴스	
Person	오지원 우혜리	개성, 성격 등..
이름, 나이, 성별 등..	임다솜 김율이	
	성재윤 장예지	

➡ '생성자'는 구체적인 인스턴스를 만들기 위한 일종의 틀이다.

5. 생성자 함수 호출시

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

const jiwon = new Person('Jiwon', 27);
console.log(jiwon);
```

5. 생성자 함수 호출시

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

const jiwon = Person('Jiwon', 27);
console.log(jiwon); // undefined
```

5. 생성자 함수 호출시

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

const jiwon = Person('Jiwon', 27);
console.log(window);
```

```
Window {window: Window, self: Window, document:
document, name: 'Jiwon', location: Location, …}
▶ Person: ƒ Person(name, age)
  age: 27
```

5. 생성자 함수 호출시

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

const jiwon = new Person('Jiwon', 27);
console.log(jiwon);
```

```
▼Person {name: 'Jiwon', age: 27}
    age: 27
    name: "Jiwon"
  ▶ [[Prototype]]: Object
```