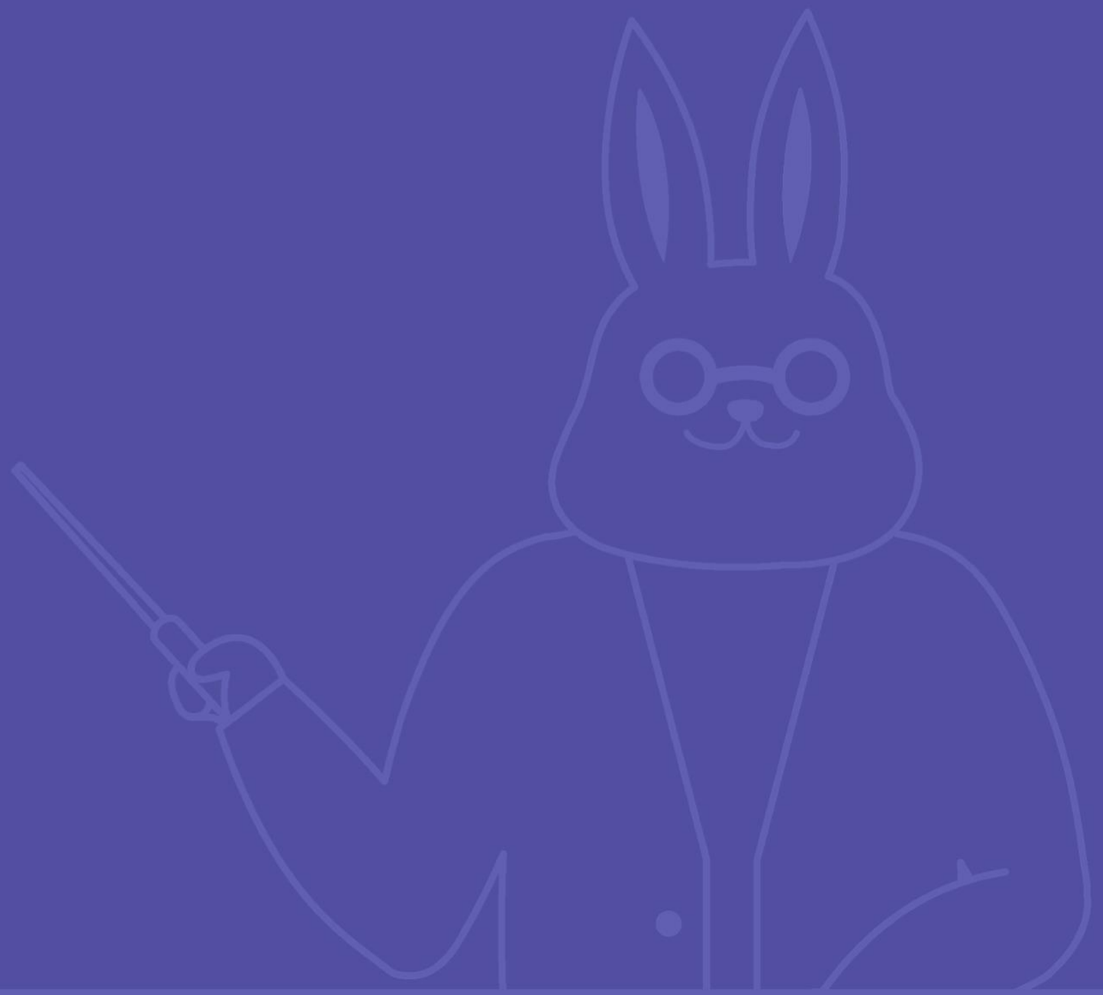
자바스크립트 심화

02 자바스크립트 실행



목차

01

자바스크립트 변수 정의 과정

자바스크립트 엔진

- 자바스크립트 엔진은 자바스크립트 코드를 읽어 실행하는 프로그램이다.
- 작성한 자바스크립트 코드는 자바스크립트 엔진을 통해 파싱되고 실행된다.
- Chrome 브라우저의 경우 V8 엔진을 사용한다.

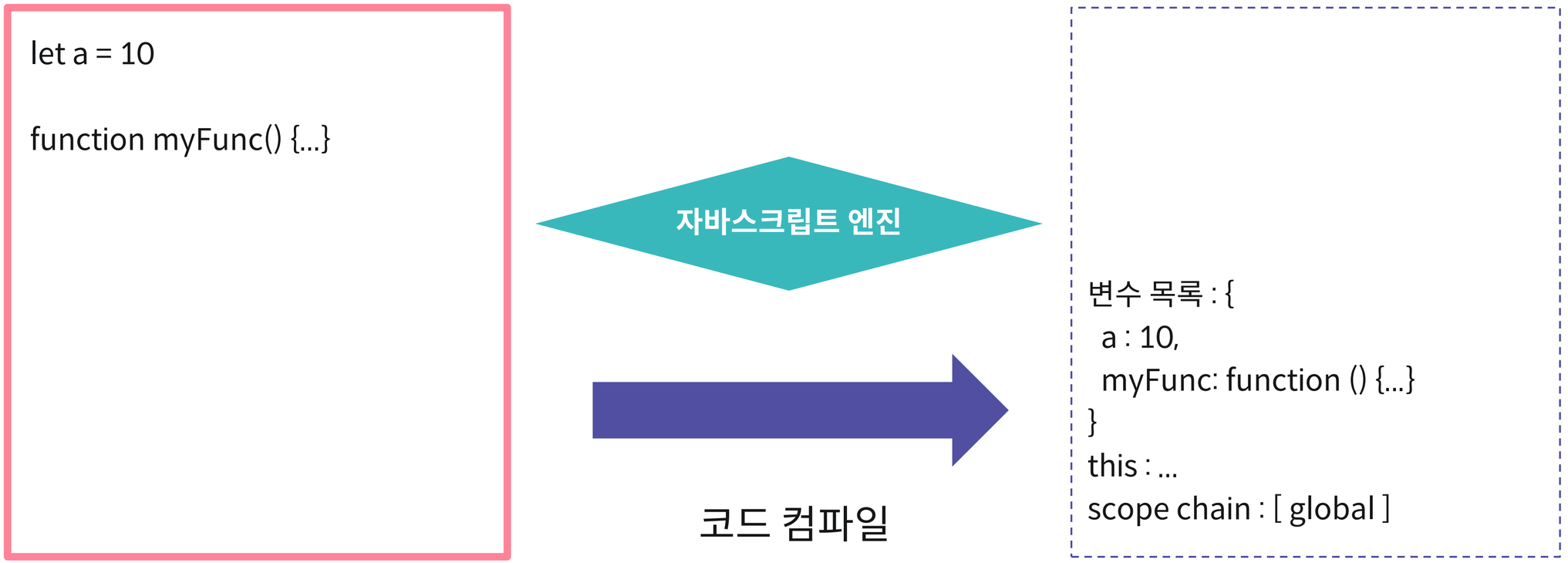
자바스크립트 엔진

- node.js는 브라우저 외의 환경에서 자바스크립트 코드를 실행하도록 하는 프로그램이다.
- node.js는 여러 프로그램으로 구성되며, 자바스크립트 코드를 읽는 프로그램으로 V8을 사용한다.
- 브라우저 환경과 node.js 환경은 같은 자바스크립트 코드를 작성해도 다르게 동작할 수 있다.

자바스크립트 코드 실행

- 자바스크립트 엔진은 코드 실행 전 실행 컨텍스트를 생성한다.
- 실행 컨텍스트는 두 단계를 통해 생성된다.
- 생성 단계에서 자바스크립트 엔진은 변수 선언을 읽는다.
- 실행 단계에서 자바스크립트 엔진은 변수 값을 할당한다.

자바스크립트 코드 실행


let a = 10
function myFunc() {...}
자바스크립트 엔진
코드 컴파일
변수 목록 : {
a : 10,
myFunc: function () {...}
}
this : ...
scope chain : [global]

렉시컬 환경(Lexical Environment)

- 함수의 렉시컬 환경은, 함수가 사용하는 변수들을 둘러싼 환경을 의미한다.
- 특정 변수의 값은 함수의 렉시컬 환경 안에서 찾을 수 있다.
- 렉시컬 환경은 실행 컨텍스트 안에 정의된 Variable Object로 이해할 수 있다.

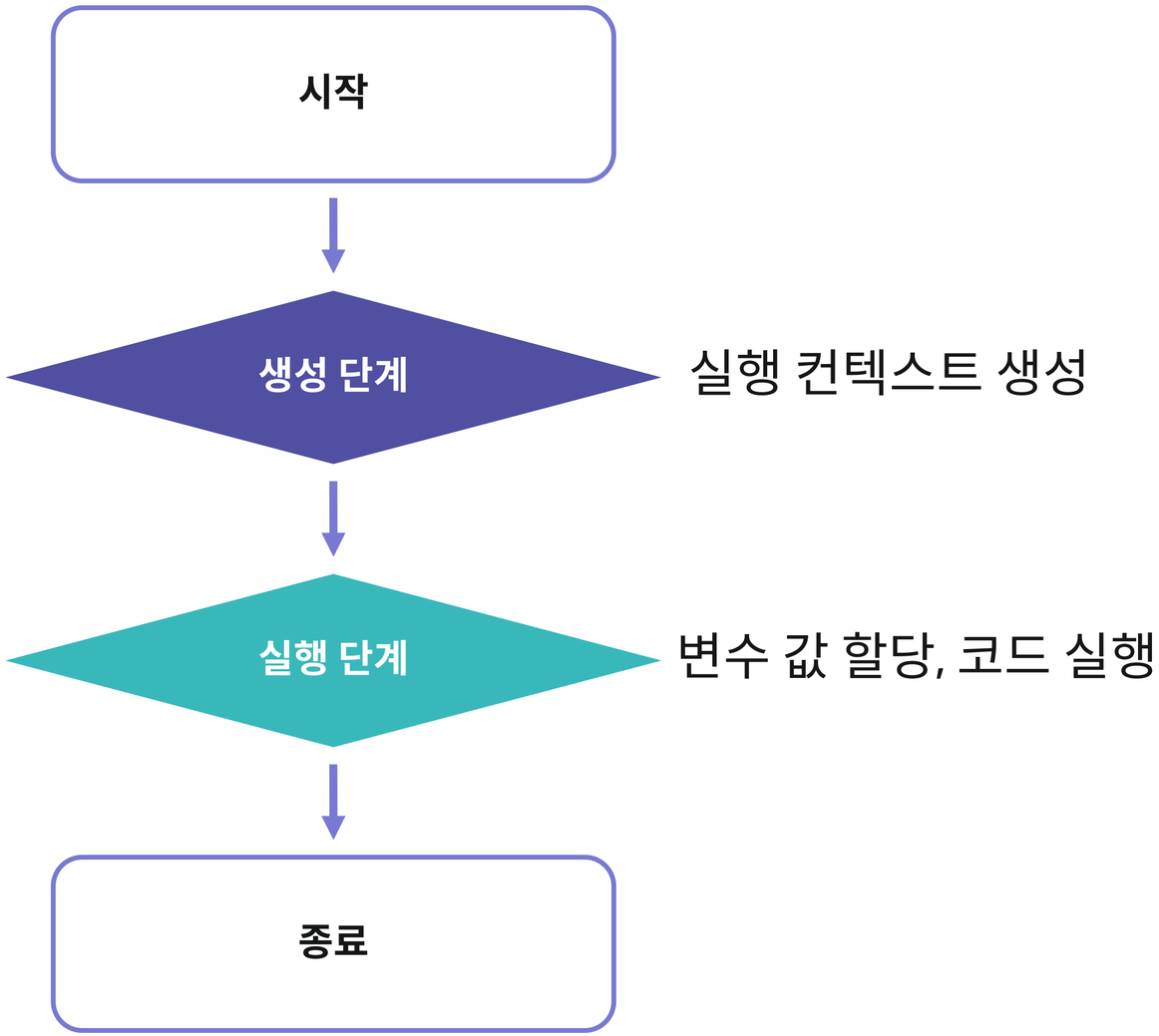
생성 단계에서의 코드 실행

- 자바스크립트 엔진은 생성 단계에서 함수 선언문, 함수 표현식, 변수 등을 읽어 실행 컨텍스트에 저장한다.
- 변수의 경우, 실행 컨텍스트의 렉시컬 환경을 구성한다.
- 함수 선언문 외에 변수는 값이 저장되지 않는다.

실행 단계에서의 코드 실행

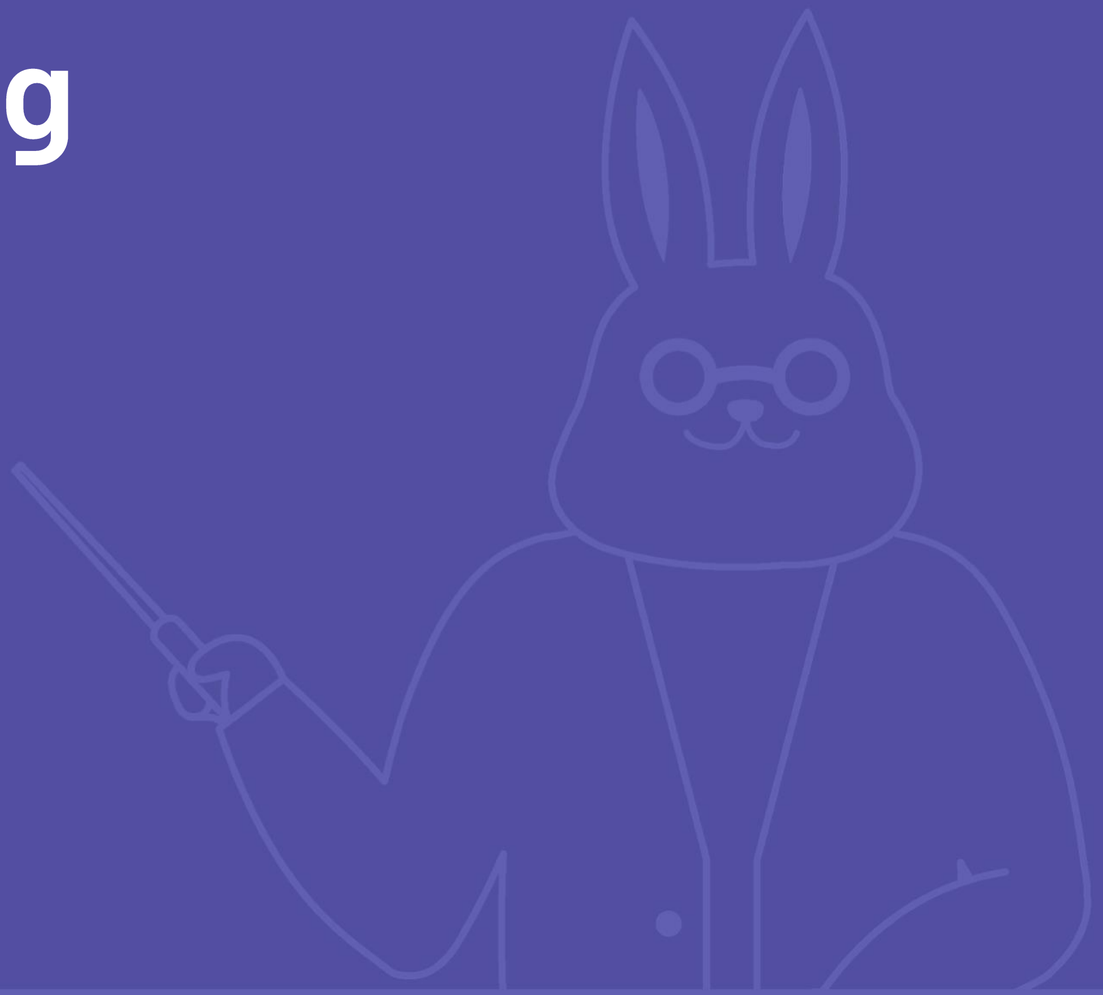
- 자바스크립트 엔진은 변수에 값을 할당하는 구문을 만나면 실행 컨텍스트에 값을 저장한다.
- 그 외 코드를 한 줄씩 읽어 나가며 실행한다.

자바스크립트 코드 실행



02

자바스크립트 Hoisting



코드 실행 시 변수 처리

- 자바스크립트 엔진이 코드를 읽으면, 생성 단계에서 실행 컨텍스트를 생성한다.
- 이때 함수 선언문은 실행 단계에서 함수 전체가 실행 컨텍스트에 저장된다.
- var 변수는 저장 시 undefined로 초기화된다.
- let, const는 초기화되지 않는다.

Hoisting

code

```
console.log(callMe())
// undefined

var x = 10

console.log(callMe()) // 10

function callMe() {
  return x
}
```

- Hoisting은 변수가 선언된 시점보다 앞에서 사용되는 현상이다.
- 이는 var 변수가 생성 단계에서 undefined로 초기화되는 것이 원인이다.
- 함수는 생성 단계에서 함수 전체가 저장되므로 뒤에서 선언되어도 호출이 가능하다.

Hoisting

code

```
// ReferenceError: Cannot access 'a'
// before initialization
console.log(callMe())

let x = 10

console.log(callMe()) // 10

function callMe() {
  return x
}
```

- let, const 변수는 생성 단계에서 초기화되지 않는다.
- 선언문 이전에 접근 시 ReferenceError가 발생한다.
- 이 경계를 Temporal Dead Zone(TDZ)라 한다.
- 따라서 let, const는 hoisting이 발생하지 않는다.

var, let, const

- var, let, const 모두 변수를 선언하는 키워드.
- var, let은 변수에 재할당이 가능하지만, const는 재할당이 불가능하다.
- var은 함수 스코프, let과 const는 블록 스코프 변수이다.

var, let, const

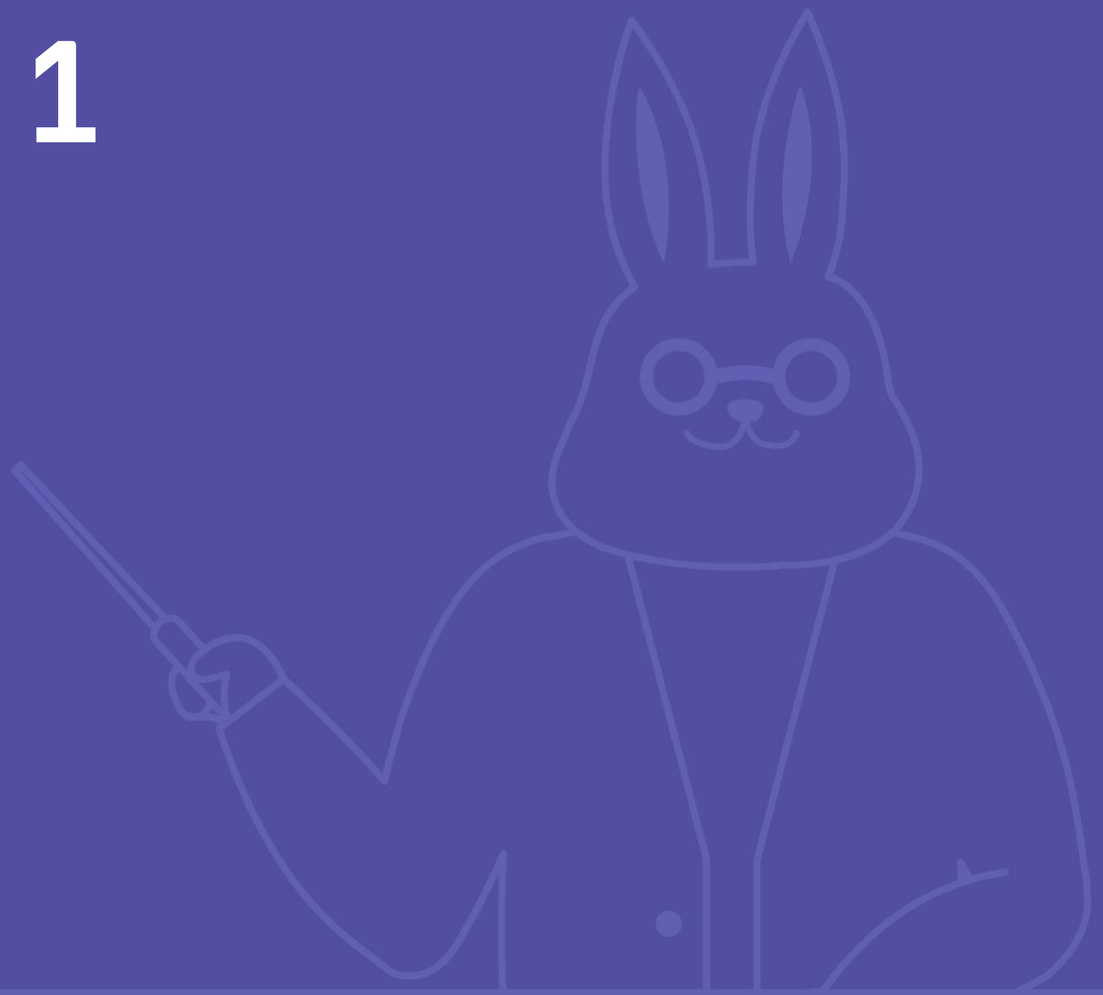
code

```
function varFor() {
  for (var i = 0; i < 3; ++i) {
    setTimeout(() => console.log("i: ",
i), 0);
  }
}

function letFor() {
  for (let i = 0; i < 3; ++i) {
    setTimeout(() => console.log("i: ",
i), 0);
  }
}

varFor(); // 3 3 3
letFor(); // 0 1 2
```

- varFor 에서 i는 varFor 함수 범위에 존재하는 변수이다.
- 따라서 setTimeout이 호출될 때, i는 for 블럭이 끝난 시점에 소멸하지 않는다.
- letFor에서 i는 for 블럭 안에 존재하는 변수이다.
- 각 for block이 실행되고 i는 소멸한다. 다만, 이 경우 각 화살표 함수의 closure에 저장된다.

03

자바스크립트 내장 객체 1



자바스크립트의 내장 객체들

- 자바스크립트는 여러 용도에 활용하는 객체를 내장하고 있다.
- 숫자 다루기, 문자 다루기, 날짜 다루기, JSON 객체 다루기 등에 유용한 객체를 제공한다.
- 핵심 내장 객체들의 기능을 이해하면, 실제 프로젝트에서 유용하게 활용할 수 있다.

globalThis

- globalThis는 전역 객체를 지칭하는 변수이다.
- 전역 객체는 환경에 따라 다르다.
- 브라우저 환경은 window, node 환경은 global 객체를 지칭한다.
- globalThis는 환경별 차이를 통일하여 하나의 변수로 서로 다른 전역 객체를 가리키게 한다.

window

code

```
const targetURL =
  "https://www.naver.com";
const windowSize =
`height=${window.innerHeight},width=
${window.innerWidth}`;
window.open(
  targetURL,
  "Target",
  windowSize
);
```

- DOM document를 포함하는 창을 나타내는 객체.
- 전역 스코프에 선언된 변수는 모두 window의 property가 된다.
- 현재 창의 정보를 얻거나, 창을 조작한다.

globalThis

code

```
const targetURL =
  "https://www.naver.com";
const windowSize =
`height=${globalThis.innerHeight},wi
dth=${globalThis.innerWidth}`;
globalThis.open(
  targetURL,
  "Target",
  windowSize
);
```

- globalThis는 브라우저 환경에서 window 객체와 같다.

document

code

```
function printDocumentInfo() {
  console.log("문서 URL:",
document.URL);
  console.log("문서 타이틀:",
document.title);
  console.log("모든 노드:",
document.querySelectorAll("*"));
}
```

- 브라우저에 로드된 웹페이지.
- 문서의 title, URL 등의 정보를 얻는다.
- element 생성, 검색 등의 기능 제공.

document

code

```
function createTodoList(todos) {
  return todos
    .map((todo) => {
      const li =
document.createElement("li")
li.appendChild(document.createTextNode(tod
o))
      return li
    })
    .reduce((ul, li) => {
      ul.appendChild(li)
      return ul
    }, document.createElement("ul"))
}
```

- createElement, createTextNode는 동적으로 원소를 생성한다.
- 이를 이용해 자바스크립트만으로 원소를 구성할 수 있다.

04

자바스크립트 내장 객체 2

Number, NaN

- 자바스크립트의 number 원시타입을 감싸는 객체.
- 유의미한 상수값, 숫자를 변환하는 메서드 등을 제공한다.
- NaN - Not a Number를 나타내는 객체.
- isNaN() - 전역 함수로, 입력값을 숫자로 변환했을 때 NaN이 되는지를 검사.

Number, NaN

code

```
function changeToUsd(krw) {
  const rate = 1046;
  return (krw /
rate).toFixed(2);
}

const krw = 1000000;
console.log(changeToUsd(krw));
```

- Number.toFixed 메서드는 숫자의 소숫점 자릿수를 제어한다.
- 반환된 값은 반올림된 문자열이다.
- changeToUsd에서 변환된 krw를 소숫점 둘째자리까지만 처리하도록 한다.

Number, NaN

code

```
function formatNumber(n) {
  if (isNaN(n)) return '0';
  return Number(n).toFixed(2);
}

formatNumber('12.345') // 12.35
```

- isNaN과 함께 활용하여, 유저의 입력을 포맷팅할 수 있다.
- formatNumber는 isNaN 함수로 빈 문자열, 잘못된 입력 등의 경우를 처리한다.

Math

- 기본적인 수학 연산 메서드, 상수를 다루는 객체.
- BigInt 타입과 호환되지 않고, Number 타입만을 인자로 다룬다.

Math

code

```
function getMaxDiff(nums) {
  return Math.max(...nums) -
Math.min(...nums)
}

getMaxDiff([-1, -4, -7, 11])) // 18
```

- Math.max, Math.min은 개별 숫자를 인자로 받아 각각 최대, 최솟값을 리턴한다.
- getMaxDiff는 배열의 최댓값, 최솟값의 차이를 계산한다.

Math

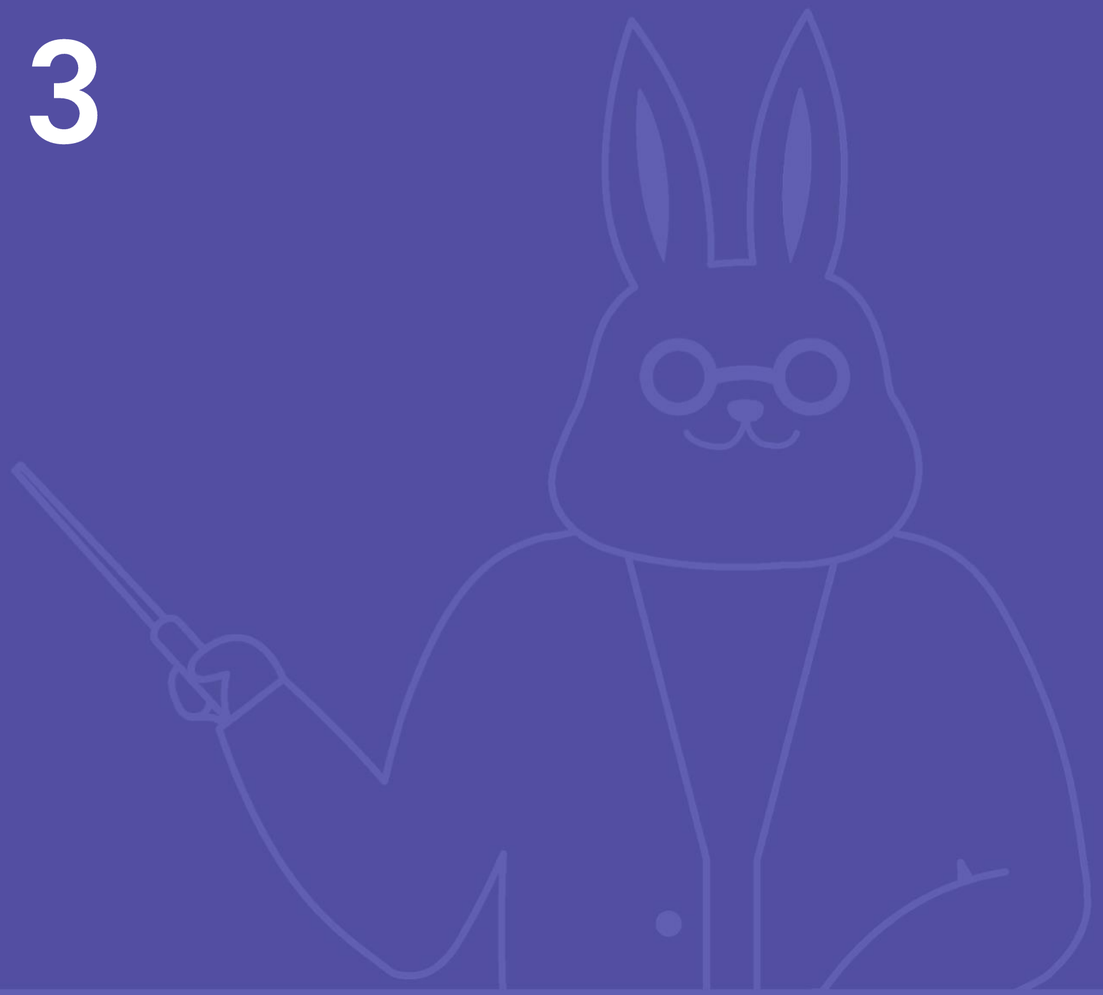
code

```
function getRandomNumberInRange(min,
max) {
  return Math.floor(Math.random() *
(max - min + 1)) + min;
}

Array.from({ length: 10 }).map(() =>
getRandomNumberInRange(50, 100))
```

- Math.random()은 0에서 1 사이의 float number을 구한다.
- Math.floor는 소숫점 이하 숫자를 버린다.
- getRandomNumberInRange 함수는 max, min 범위의 랜덤 숫자를 구한다.

05

자바스크립트 내장 객체 3



Date

- 특정 시점의 날짜를 표시하기 위한 객체.
- 날짜와 관련된 작업을 하기 위한 여러 메서드를 포함한다.

Date

code

```
function isWeekend(today) {
  let day = today.getDay();
  return day === 0 || day ===
6;
}

console.log(isWeekend(new
Date("2021/9/12")));
```

- Date.getDay() 는 요일을 0(일요일)부터 6(토요일) 로 구한다.
- 이 외에 년도, 월, 일, 시, 분, 초, 밀리초 등을 구할 수 있다.

Date

code

```
function addDays(date, days) {
  date.setDate(date.getDate() +
days)
  return date.toDateString()
}

addDays(new Date("2021/9/22"),
100)) // Fri Dec 31 2021
```

- setDate() 등의 메서드로 시간을 설정한다.
- 설정 시 월 변경 등의 시간 변환은 Date 객체가 처리한다.
- toDateString() 메서드는 특정 포맷의 문자열을 반환한다.

Date

code

```
function timeDiff(date1, date2) {
  return date2.getTime() -
date1.getTime()
}
let dayTime = 60 * 24 * 60 * 1000
function fromNow(date) {
  let diff = timeDiff(date, new
Date())
  return `${Math.floor(diff /
dayTime)} days ago...`
}

fromNow(new Date("2021/9/1"))
```

- getTime() 메서드는 시간을 밀리초 단위로 반환한다.
- 이때 밀리초는 1970.1.1 시점부터 흐른 시간이다.
- fromNow는 주어진 시간이 현재로부터 며칠이나 흘렀는지를 계산한다.

String, JSON

- 자바스크립트의 문자열 원시 타입의 래퍼 객체.
- 문자열을 조작하기 위한 여러 메서드를 포함한다.
- JSON - JSON 객체와 관련된 메서드를 담은 객체.

String, JSON

code

```
function toUserList(users) {
  return users
    .filter((user) =>
!user.includes("Admin"))
    .map((user) =>
user.trim().toUpperCase())
    .map((user) => `<li>${user}</li>`)
    .join("")
}

console.log(toUserList(["Daniel", "Tom",
"Johnny", "Admin"]))

 // <li>DANIEL</li> <li>TOM</li>
<li>JOHNNY</li>
```

- trim(), toUpperCase() 등은 변환된 새로운 문자열을 리턴한다.
- includes() 메서드는 문자열 검색에 성공 시 true, 실패 시 false를 리턴한다.
- toUserList()는 이름의 배열을 받아 리스트 태그 목록의 문자열을 계산한다.

String, JSON

code

```
"Daniel,Kim,SW".split(',')
// [ 'Daniel', 'Kim', 'SW' ]
"Daniel,Kim,SW".replace(',', ' ')
// "Daniel Kim SW"
"Daniel,Kim,SW".includes("Kim")
// true
"   Daniel,Kim,SW   ".trim()
// "Daniel,Kim,SW"
"Daniel,Kim,SW".indexOf("Kim")
// 7
```

- split()은 주어진 문자열에 따라 타겟 문자열을 나눈다.
- replace()는 주어진 문자열을 검색하여 타겟 문자열로 변환한다.
- indexOf()는 특정 문자열을 검색하여 시작점의 인덱스를 반환한다. 없을 시 -1을 리턴한다.

String, JSON

code

```
JSON.stringify({ name:
"Daniel", age : 12})

// '{"name":"Daniel","age":12}'

JSON.parse('{"name":"Daniel","a
ge":12}')

// { name: 'Daniel', age: 12 }
```

- JSON.stringify()는 주어진 객체를 JSON 문자열로 만든다.
- JSON.parse()는 주어진 JSON 문자열을 자바스크립트에 맞는 결과 객체로 만든다.

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
김일식

강사
김일식

감수자
김일식

디자이너
강혜정

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io