LEARNING SPOONS

LEARNING SPOONS ONLINE

자바스크립트 기초

자바스크립트 기초다지기

LEARNING SPOONS

Unit

1. 객체와 함수 이해하기
2. 스코프 이해하기
3. ES6의 향상된 문법 살펴보기
4. [실습] To-do 앱 클론코딩 1차

Unit 2-1

객체와 함수 이해하기

함수 정의

1. 이름있는 함수 선언문 a named function declaration
2. 익명함수 표현식 an anonymous function expression
3. 이름있는 함수 표현식 an anonymous function expression
4. 즉각실행 함수 표현식 an immediately-invoked function expression
5. 함수 생성자 a function constructor

```
function sum(x, y) { return x + y; }
var sum2 = function(x, y) { return x + y; };
var sum3 = function sum3(x, y) { return x + y; };
var sum4result = (function(x, y) { return  x + y; })(1, 2);
var sum5 = new Function('x', 'y', 'return x + y;');
```

객체 정의

1. 객체 리터럴 이용
2. Object를 사용
3. new 키워드를 이용

```
// 가장 간단한 방법은 객체 리터럴을 이용
var person = {
 name: 'jay'
};

// new 키워드를 이용하여 생성
var obj = new Object(); // 빈 객체 생성, 리터럴 표현인 {}와 같다.
var arr = new Array(); // 빈 배열 객체 생성, 리터럴 표현인 []와 같다.
var date = new Date(); // 빈 현재 날짜 객체 생성

// Object.create를 이용하여 전달 받은 객체를 상속하는 객체 생성
var obj2 = Object.create({x:1, y:2});
```

객체 속성과 메소드

```
var person = {
 name: 'jay',
 hello: function() {
   console.log('hello')
 }
};

var name = person.name; // person 객체의 name 속성에 접근한다.
person.age = 20; // 속성이 없으면 새로운 속성을 추가하고 값을 할당한다.
person.name = 'jin'; // 있으면 기존값을 덮어쓴다.

var age = person["age"] // person 객체는 일종의 연관배열로 배열의 요소에 접근하듯 접근할 수 있다.

// 속성 접근 방법 #1 object.property
// 속성 접근 방법 #2 object["property"]

var numOfChildren = person.children.length; // 에러가 발생

delete person.age; // 속성 제거

person.hasOwnProperty("age"); // 속성 존재 확인
```

getter와 setter

```
var person = {
 firstName: 'jay',
 lastName: 'ko',
 get fullName() {
     return this.firstName + ' ' + this.lastName;
 },
 set fullName (name) {
     var words = name.toString().split(' ');
     this.firstName = words[0] || '';
     this.lastName = words[1] || '';
 }
}

console.log(person.fullName);
person.fullName = 'jay ko';
console.log(person.firstName); // jay
console.log(person.lastName) // ko
```

값 타입과 참조 타입

1. 자바스크립트에서 제공하는 모든 원시형 타입인 숫자(number), 문자열(string), 불린(boolean), 그리고 정의되지 않은(undefined), 객체가 없음(null)은 값 타입이다.
2. 값 타입 데이터를 제외한 객체, 배열, 함수는 모두 참조 타입의 데이터이다.

```
var people = [ { name: 'jay', age: 20 }, {name: 'jin', age: 10 } ];
var makeCapital = function( name ) {
   name = name.charAt(0).toUpperCase() + name.slice(1);
   return name;
}
var increaseAge = function( person) { person.age += 1; }
var addPerson = function( people, name, age ) {
    people.push( { name: name, age:age } );
}

makeCapital(people[0].name);
increaseAge(people[1]);

addPerson( people, 'jim', 32 );
console.table( people );
```

LEARNING SPOONS

arguments 객체

1. arguments 객체를 통하여 함수에 전달되는 인자에 접근
2. arguments 객체의 length 속성을 통하여 인자의 갯수를 확인
3. 개별 인자에는 배열의 인자 접근과 같은방식으로 접근
4. arguments 객체는 배열이 아님

```
function sumof(){
 var total =0;
 for (var i=0; i < arguments.length; i++){
   total += arguments[i];
 }
 return total;
}

sumeof(2,3,4)
```

LEARNING SPOONS

함수의 특성

1. 호출 할 수 있다 → 함수 바디의 문장들이 실행된다.
2. 자바스크립트에서 함수는 일급객체first-class object 이다.
   a. 리터럴로 생성되고 변수나 데이터 구조에 할달될 수 있다.
   b. 함수의 인자로 전달 할 수 있다.
   c. 반환값으로 사용할 수 있다.
   d. 런타임에 생성될 수 있다.

객체

```
var person = {}; // 변수에 할당한다.
person.job = {}; // 속성을 생성하고 새로운 객체를 할당
hide({}); // 인자로 전달한다.

function returnNewObject() { // 실행결과로 리턴한다.
 return {};
```

함수

```
var noop = function() {}; // 변수에 할당한다.
person.eat = function(food) {}; // 객체의 속성에 할당
function ask(func) {
 func();
}
ask(function() {});// 인자로 전달한다.

function returnNewFunc() { // 실행결과로 리턴한다.
 return function() {};
}
```

Unit 2-2

스코프 이해하기

LEARNING SPOONS

간단한 퀴즈

```
function foo() {
 return a;
 a = 10;
 function a() {
   console.log('안녕하세요.');
 }
}

console.log(foo()); // ?
```

스코프 개요

1. 컴퓨터 과학에서 scope 란 name-binding 관 연관이 있다.
2. name-binding 이란 말 그대로 이름을 연결 하는 것라고 생각하면 된다.
3. 변수/코드(식별자) 가 어느 개체(실제 메모리 주소) 에 연결하는 된다는 것을 말한다.
4. scope 란 name-binding 이 유효한 범위를 나타낸다. 이러한 범위를 scope block 이라고 한다.
5. 변수는 유효 범위 내에서 어떤 개체를 참조 할지 알 수 있다.

스코프 종류

1. Lexical Scope
    a. 변수, 함수 등이 선언되는 시점의 scope를 기준으로 삼는다.
        i. C, C#, JAVA, Javascript
2. Dynamic Scope
    a. 변수, 함수 등이 호출 되는 시점의 scope 를 기준으로 삼는다.
        i. Perl, Lips, Clousure, Perl ..

LEARNING SPOONS

Lexical scope vs Dynamic Scope

```javascript
// javascript
var a = 'global';
function foo() {
 console.log(a);
}

function bar() {
 var a = 'local';
 foo();
}

foo(); // ?
bar(); // ?
```

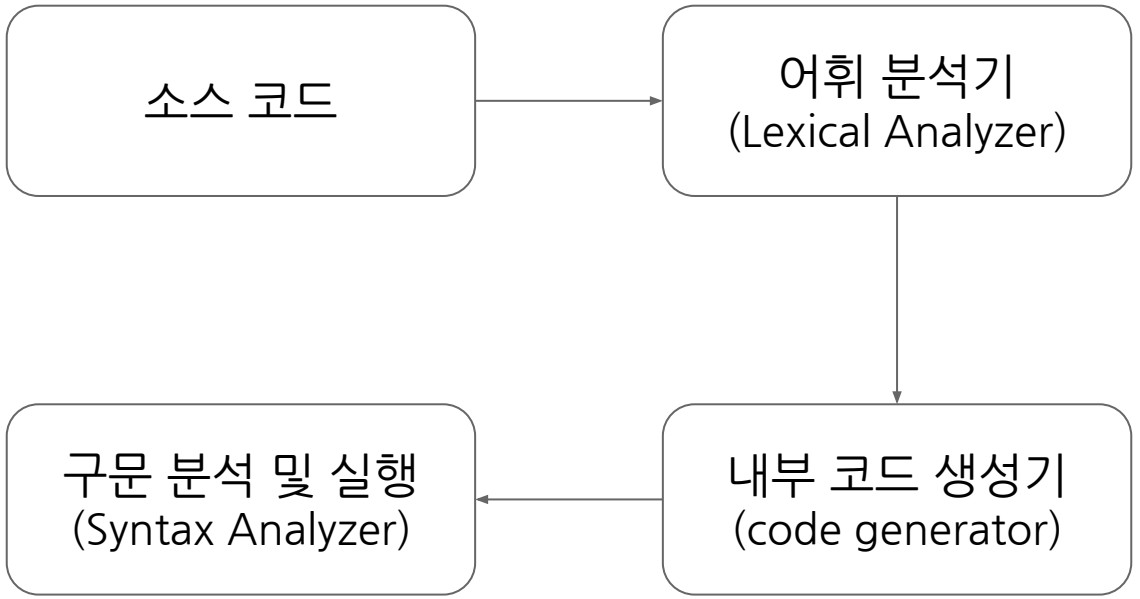
```perl
# perl
$a = "global";
sub foo {
 print "$a\n";
}

sub bar {
 local $a = 'local';
 foo();
}

foo(); # ?
bar(); # ?
```

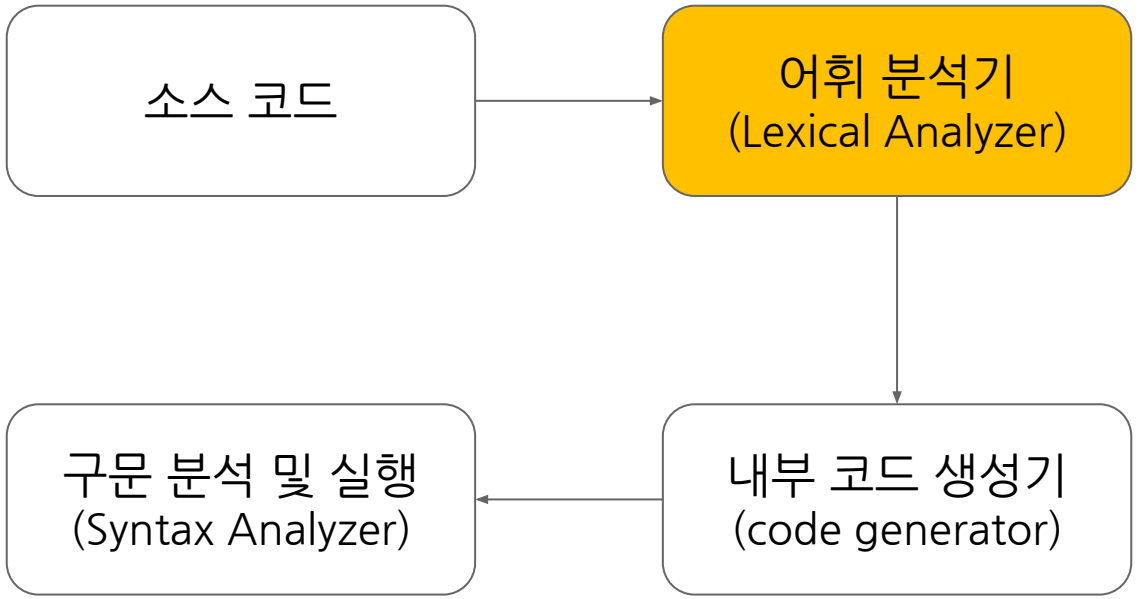
LEARNING SPOONS

Lexical scope



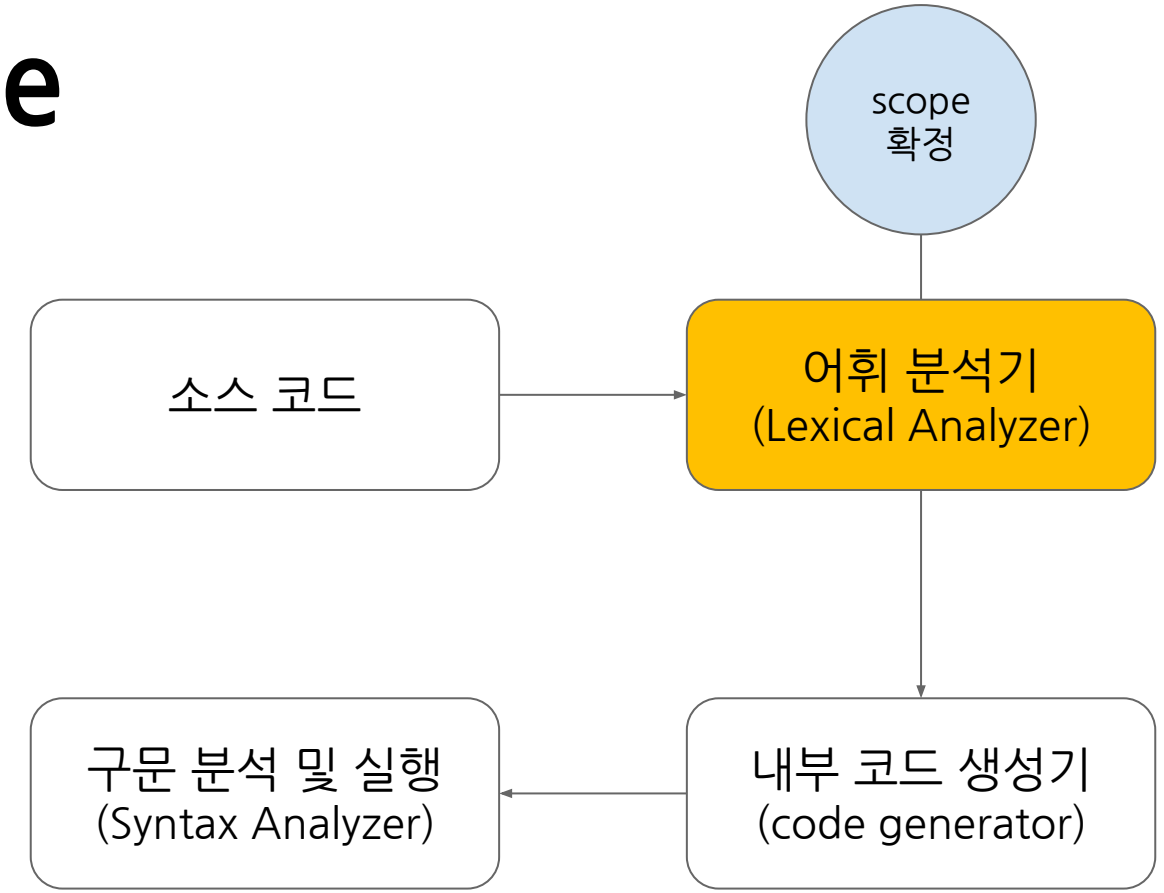
LEARNING SPOONS

Lexical scope



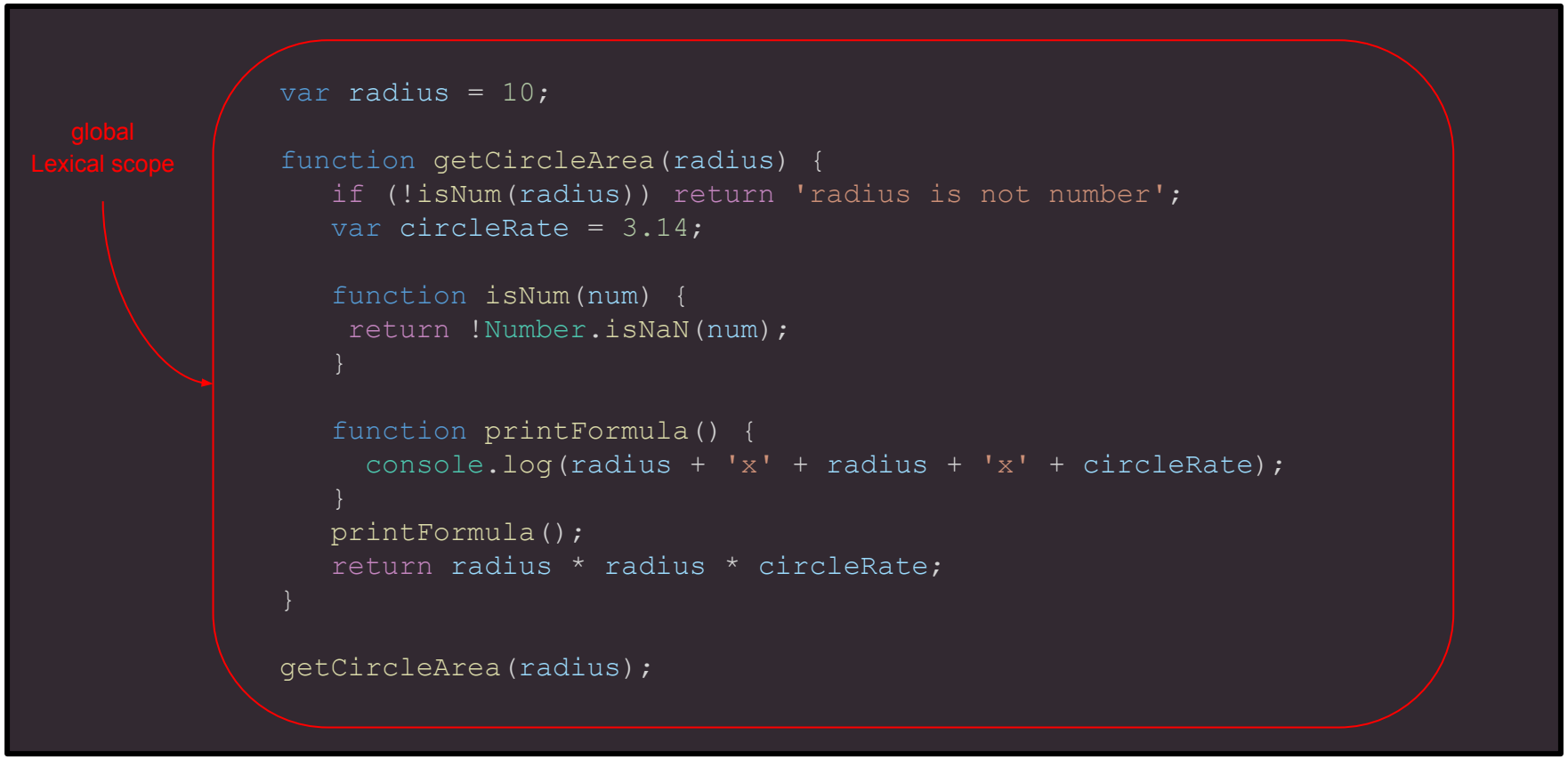
LEARNING SPOONS

Lexical scope



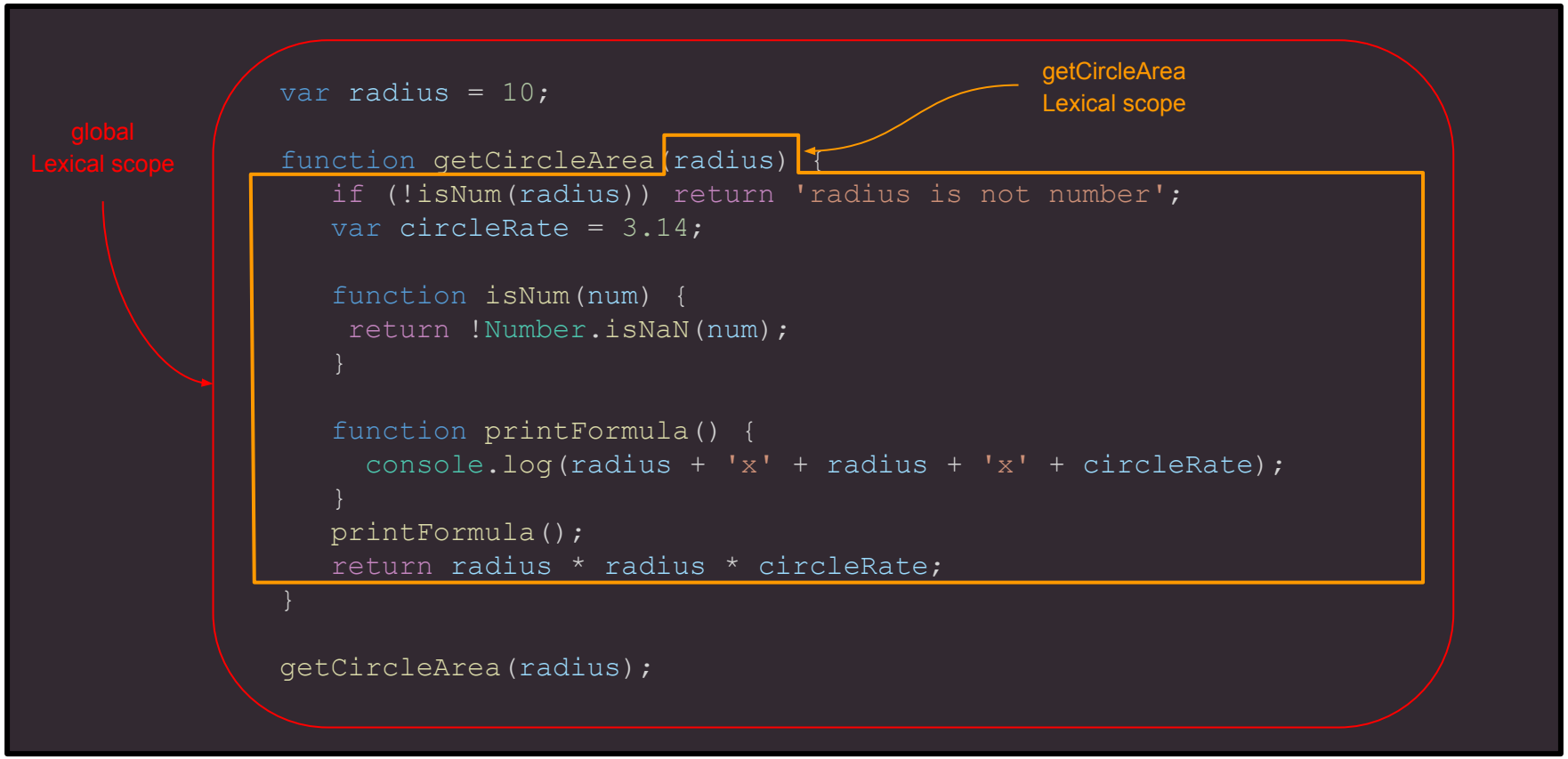
LEARNING SPOONS

Lexical scope



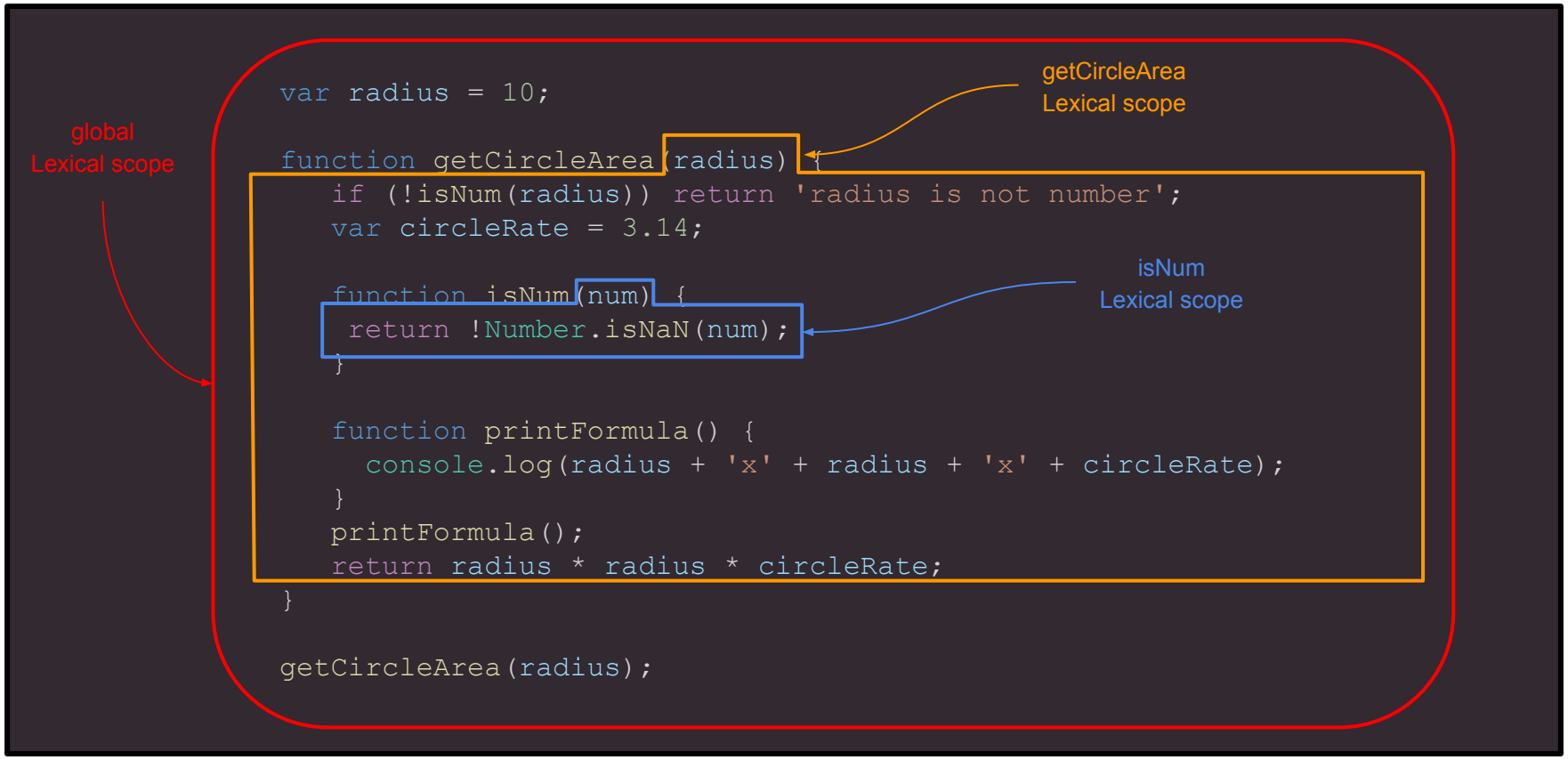
LEARNING SPOONS

Lexical scope



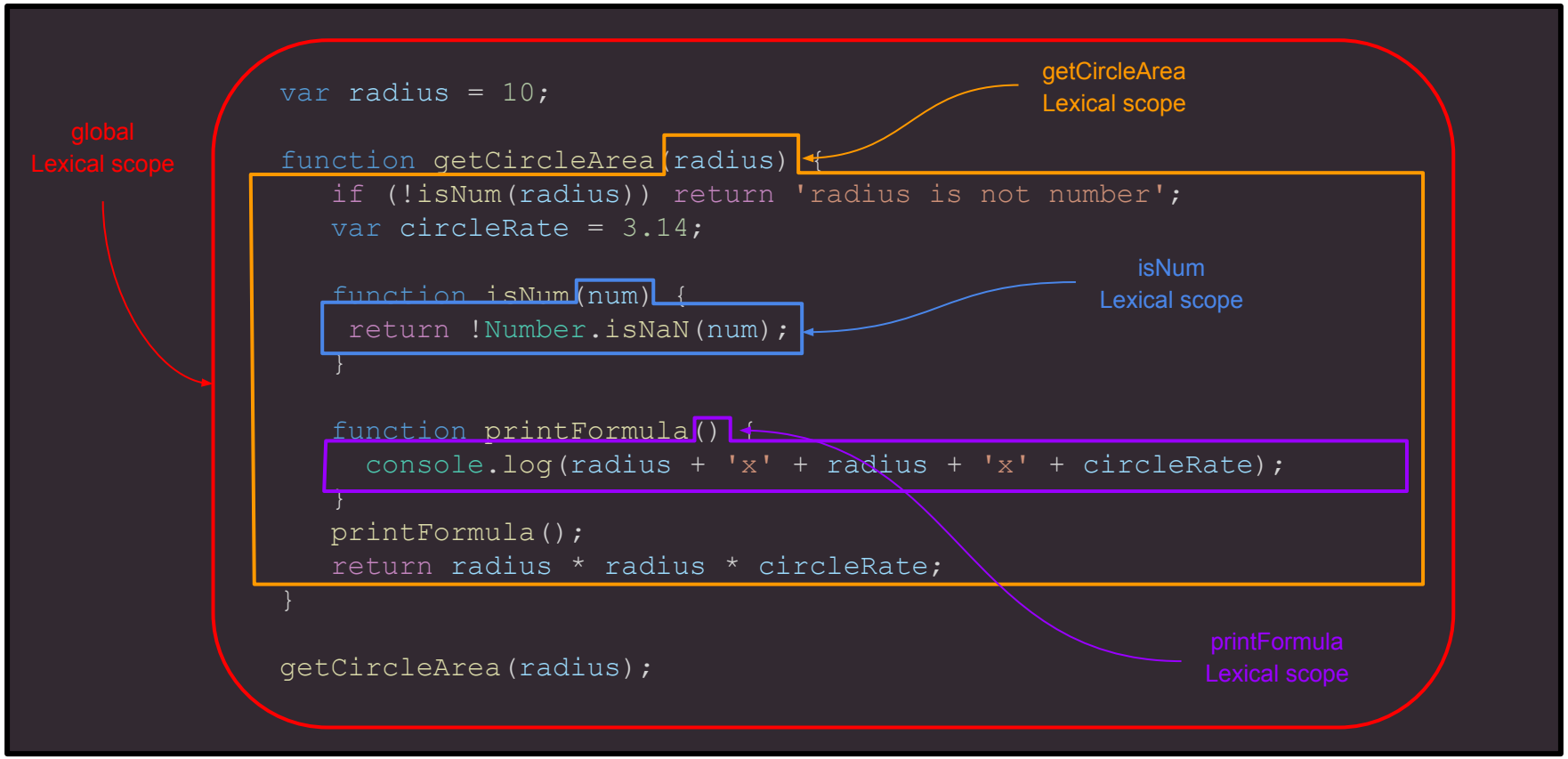
LEARNING SPOONS

Lexical scope



LEARNING SPOONS

Lexical scope



호이스팅

1. 자바스크립트에서는 함수를 선언하기 전에 호출이 가능한데 이러한 현상을 호이스팅이라고 한다.

```
hello(); // 안녕하세요
function hello() {
  console.log('안녕하세요');
}
```

```
hello(); // Uncaught TypeError: hello is not a function
var hello = function ()
  console.log('안녕하세요');
}
```

Unit 3

ES6의 향상된 문법 살펴보기

LEARNING SPOONS

var vs let

1. var로 생성된 변수는 함수단위의 스코프를 가진다.
2. let은 블록단위의 스코프를 가진다.

```
(function () {
    var foo = 'bar';
    if (foo) console.log(foo) //bar
}());
console.log(foo); // ReferenceError: foo is not
defined
```

```
let getValue, setValue;
{
 let data = {};
 setValue = function (key, val) {
   data[key] = val;
 }
 getValue = function (key) {
   return data[key];
 }
}
setValue('user1', 'jay');
getValue('user1');
console.log(data);
```

const

1. let과 마찬가지로 블록단위의 스코프를 가진다.
2. 재할당을 할 수 없지만 불변객체는 아니다.

```
const URL = 'http://hello';
URL = "http://opps" //TypeError: Assignment to constant variable.

const CONST_OBJ = {};
CONST_OBJ.foo = 'hello';
CONST_OBJ.foo = 'hi';

CONST_OBJ = {foo: 'opps'} //TypeError: Assignment to constant variable.
```

템플릿 문자열

1. 문자열을 위한 새로운 리터릴이고 back-tick(`)을 사용하여 정의 (기존 문자열은 '따옴표' 혹은 "쌍따옴표"로 정의) 로 정의한다.
2. Multiline 문자열처리 및 데이터 삽입 처리가 가능하다.

```
let cart = [
 { name: '옷', price: 2000 },
 { name: '가방', price: 1000 }
];
let interpolated = `카트에 ${cart.length}개의 아이템이 있습니다`;
let interpolated2 = `카트에는 ${
 cart.map(function(item){ return item.name})
}이 있습니다.`;

console.log(interpolated); //카트에 1개의 아이템이 있습니다
console.log(interpolated2); //카트의 옷, 가방이 있습니다.

let multiline = `Hello,
World`;


console.log(multiline);
```

Symbol

1. ES6의 새로운 기본형이고 리터럴 표현식이 없다.
2. 심볼로 만들어진 키는 for...in 루프, Object.keys, Object.getOwnPropertyNames에서 숨겨진다. (JSON.stringify 조차도!)

```
const symbol = Symbol();
const symbol2 = new Symbol() // <- TypeError
const hello = Symbol('hello') // 디버깅 목적

console.log(Number(3) === Number(3)) // <- true
console.log(Symbol() === Symbol()) // <- false
console.log(Symbol('symbol') === Symbol('symbol')) // <- false

console.log(typeof Symbol()) // <- 'symbol'

const nationility = Symbol('nationility')
const user = {
 name: 'john',
 [nationility]: 'korean'
}
console.log(user[nationility]) // korean
```

Default Parameters

1. 함수의 파라미터가 호출 시 주어지지 않을 때의 기본값을 줄 수 있다.
2. 기본 파라미터에 표현식을 줄 수 있고 이는 함수 바디에서 실행된다.
3. 기본 파라미터는 함수가 호출될 때 평가된다.

```
function buildChart(width = 200, height = width / 2) {
 // 차트를 만든다.
}

const chartA = buildChart() // 200, 100
const chartB = buildChart(100) // 100, 50
```

Rest Parameters

1. 3개의 .으로 … 표현할 수 있다.
2. 무조건 매개변수 목록중 마지막에 와야한다.
3. 기본 파라미터는 함수가 호출될 때 평가된다.

```
function avg() {
 const args = Array.prototype.slice.call(arguments);
 //...
}

function avg() {
 const args = Array.from(arguments);
 //...
}

function avg(...args) {
 //...
}
```

LEARNING SPOONS

화살표 함수Arrow Function

1. 더 간결하게 함수를 만들수 있는 표현식이다.

```
const double = x => x + x;

const add = (a, b) => a + b;

const rand1to10 = () => Math.floor(1 + Math.random() * 10);

const log = (...args) => console.log(args);

const doThings = (a, b) => {
 doA();
 doB();
}
```

파라미터가 1개이면 괄호가 필요없다

1개 이상이나 없을 경우 괄호가 필요

rest 파라미터일 경우 괄호 필요

표현식 결과가 리턴값

표현식이 1개 이상일 경우 괄호 필요하고 명시적으로 return을 써줘야함

실습

To-do 앱 클론코딩 1차