자바스크립트 핵심 개념

scope(유효범위) 와 호이스팅

- 변수와 매개변수의 접근성과 생존기간

global scope, local scope

- 협업시 충돌날 가능성을 대비해 글로벌 스코프 보다는 로컬 스코프를 권장한다.

```
    var globla_scope = "global"; //전역스코프
        var local_function = function() {
            var local_scope = 'local'; //지역스코프
            console.log(global_scope); //전역스코프 참조 가능. global
            console.log(local_scope); //한수 내이기 때문에 지역 스코프 참조 가능
local
        }
    console.log(local_scope); // local_scope은 지역 시코프이기 때문에 에러 발생
```

- 위와 같이 다른 함수끼리 참조할 수 가 없다.
- 하지만 함수 안의 함수는 참조 할 수 있을까?

스코프체인

- inner->outer->global 꼬리에 꼬리를 무는 스코프 체인
- 반대로는 안된다.

```
    var a = 1
    function outer(){
        var b =2;
        console.log(a)// 1
        function inner(){
            var c =3;
            console.log(b);
            console.log(a);
        }
        inner();// 2 , 1;
    }
    outer();
    // 1, 2, 1
    console.log(c); // c is not defind
```

Lexical scope(정적 범위)

- 한번 !선언! 된 이상 선언 된 기준으로 스코프를 찾아본다.

- 어디서 호출하는지가 아니라 어떤 스코프에서 선언하였는지에 따라 결정된다.
- 동적 스코프의 반댓말

```
var a = 'global';

function text(){
    console.log(a);
}

function read(){
    var a = 'read';
    text();
}
read(); // global
```

- 해당 문제를 해결 하기 위해서는 지역변수 read 함수에 var a 를 선언하지말고 전역 변수의 값을 바꾸어 주면 된다.

```
var a = 'global';

function text(){
    console.log(a);
}

function read(){
    a = 'read';
    foo();
}
read();
```

호이스팅

- 끌어올린다는 사전적 어원이다.
- 변수와 함수 모두 호이스팅이 된다.
- 함수 안에 변수를 선언할 때 어떤 위치에 있던 함수의 시작 위치로 끌어올린다.

```
// 변수의 호이스팅

function foo(){
    // var a; 가 있는 것과 같다.
    console.log(a); //udefined
    var a = 100;
    console.log(a); //100
}
```

```
// 함수의 호이스팅

function human(name){
    console.log("My name is" + name);
}
human("kim");

// My name is kim

human("kim");

function human(name){
    console.log("My name is" + name);
}

// My name is kim

// 함수 표현식에서는 오류 발생
human("kim"); // human is not a function

var human = function(name){
    console.log("My name is" + name);
}
// 이는 아래와 같기 때문이다.
// human 선언을 위로 호이스팅 해버린다.

var human;

human("kim");

human = function(name){
    console.log("My name is" + name);
}
```

Closure(클로저)

- 현재 상태를 기억하고 변견된 최신 상태를 유지해준다.
- 객체지향을 흉내낼 수 있다.
- private 키워드를 흉내낼 수 있다.
- 메모리 누수의 원인이 될 수 있으므로 남용해서는 안된다.

```
var base = "name is :";
function sayHello(name){
    var text = base + name;
    return function(){
        console.log(text);
    }
}
var test1 = sayHello('김일');
```

```
    var test2 = sayHello('김이');
    var test2 = sayHello('김삼');
    // 함수가 종료 되지 않고 살아 있다.

    test1();
    //  name is : 김일
    test2();
    // name is : 김이
    test3();
    // name is : 김삼
```

- 반복문의 클로저

```
    function count(){
        var i
        for (i = 0; i < 5; i++ ) {
            setTimeout(function() {
                console.log(i);
            }, 100);
        }
    }
    count();
    // 결과
    // 5
    // 5
    // 5
    // 5
    // 5

// 순차적으로 출력하기 하기 위한 조치

   function count(){
        var i
            for (i = 0; i < 10; i++ ) {
                (function(j){
                setTimeout(function() {
                    console.log(j);
                }, 100);
            })(i);
        }
    }
    count();
    //결과
    // 1
    // 2
    // 3
    // 4
    // 5
```

- 실제 클로저 활용 바로가기
- 이해한 개념을 통해 실제 코드 작성시 응용해본다.