2023.06.04 Closure

클로저

1. **클로저(Closures)**

   클로저는 **함수와 함수가 선언된 어휘적(Lexical) 환경의 조합**입니다(A closure is **the combination of a function and the lexical environment** within which that function was declared). 즉, 함수가 선언될 당시의 환경을 기억했다가 호출될 때 이 환경에 따라 수행되는 함수라고 할 수 있습니다.

2. **Lexical Scoping**

   자바스크립트는 기본적으로 변수의 scope를 Lexical하게 지정합니다. 즉, 변수의 호출 시점이 아닌 선언 시점에서 유효 범위가 결정된다는 의미입니다.

3. Example

   a. 아래 코드에서, makeFunc()는 내부의 displayName 함수를 반환합니다. displayName은 함수 밖의 name 변수를 사용하고 있습니다. 반환된 displayName은 myFunc라는 변수(**클로저**)에 저장되고, myFunc을 호출하면 displayName은 함수 밖의 name 변수에 접근할 것입니다.

   b. 일반적인 다른 언어에서는 함수의 처리가 끝나면 함수 내부의 지역 변수(여기에서는 name)에 접근할 수 없기 때문에 아래 코드가 동작하지 않을 수 있습니다.

   c. 그러나 자바스크립트에서는 displayName의 외부에 있는 name에 접근하여 "DKBMC"를 alert합니다. displayName을 반환하면서 클로저를 형성하기 때문입니다. 다시 말해, **myFunc**은 클로저로, makeFunc()의 반환값인 displayName **함수뿐만 아니라,** displayName **함수가 선언된 어휘적 환경**인 name을 기억하고 있는 것입니다.

```
function makeFunc() {
  var name = "DKBMC";
  function displayName() {
    alert(name);
  }
  return displayName;
}
```

```
var myFunc = makeFunc();
myFunc();
```

예시

```
function showHelp(help) {
  document.getElementById('help').innerHTML = help;
}

function setupHelp() {
  var helpText = [
    {'id': 'email', 'help': 'Your e-mail address'},
    {'id': 'name', 'help': 'Your full name'},
    {'id': 'age', 'help': 'Your age (you must be over 16)'}
  ];

  for (var i = 0; i < helpText.length; i++) {
    var item = helpText[i];
    document.getElementById(item.id).onfocus = function() { showHelp(item.help); }
  }
}

setupHelp();
```

위 코드가 작동하지 않는 이유

`var item = helpText[i];`

loop에서 3개의 클로저가 만들어지는데, 각 클로저는 item 환경을 공유한다. 즉, item은 최종적으로 age를 가리킨다.

let이나 const로 변경하면 모든 클로저가 각각의 블록 범위에서 item을 가지게 되므로 이를 공유하지 않게 된다.

또는 forEach를 사용하는 방법도 있다.