2주 3회차

1. 암묵적 타입 변환이 무조건적으로 좋지 않은 문화이자 기능인가?

- 무조건적으로 좋지 않은 것은 아니다.
- 암묵적 타입 변환의 장점
  - 코드가 더 간결해 일부 상황에서 코드 작성이 더 쉽다.
  - 코드 길이가 짧아진다.
- 암묵적 타입 변환의 단점
  - 의도치 않은 동작을 유발할 수 있다.
  - 많이 사용하면 코드의 동작을 예측하기 어려울 수 있고, 디버깅이 어려워질 수 있다.
- 코드의 명확성을 높여야 할 경우에는 명시적 타입 변환을, 코드를 간결하게 작성해야 할 경우에는 암묵적 타입 변환을 사용하는 것이 좋다.

2. 아래 설명에 따라 단축 평가를 이용하여 아래의 if문처럼 작동하는 true 값 여부를 판별하는 코드를 빈칸에 알맞게 작성해보시오. 결과도 내시길 바랍니다.

```
var isThereMessage = true;
var message ='';
if(isThereMessage) message = '멘토는 죽어있다.';
message = _______ _______ _______ ;
console.log(message);
```

- 코드

```
var isThereMessage = true;
var message ='';
message = isThereMessage  && '멘토는 죽어있다.';
console.log(message);
```

- 출력결과

```
멘토는 죽어있다.
```

- `&&` 연산자를 사용하여 `isThereMessage`가 `true`일 때는 `멘토는 죽어있다.` 가 `message` 변수에 할당되도록 하고, `isThereMessage`가 `false`일 때는 `false` 가 `message` 변수에 할당되도록 한다.

3. 아래의 코드를 실행해보시고 왜 결과 값이 그렇게 나오는지 생각해보세요

```
var person = {
firstName : 'turtle',
last-name : 'park'
};
console.log(person);
//================================================
var word1 = {
var: '',
function: ''
};
console.log(word1);
//================================================
//프로퍼티 키 동적 생성
var objES5 = {}
var keyES5 = 'ES5'
objES5[keyES5] = 'world';
console.log(objES5);
//================================================
//계산된 프로퍼티 이름
var keyES6 = 'HELL';
var objES6 = {[keyES6]: 'o'};
```

```
console.log(objES6);
//===========================================
var emptyObj = {
'' : ''
};
console.log(emptyObj);
//===========================================
var numObj = {
1 : 0,
2 : 1,
3 : 2
};
console.log(numObj);
//===========================================
var duplicateObj = {
name : 'park',
name : 'kim'
};
console.log(duplicateObj);
//===========================================
```

- 출력결과

```
SyntaxError
{var: '', function: ''}
{ES5: 'world'}
{HELL: 'o'}
{'': ''}
{1: 0, 2: 1, 3: 2}
{name: 'kim'}
```

- 첫 번째 코드 : 객체 `person`을 정의하는 코드다. 그러나 프로퍼티 키가 `last-name`으로 하이픈이 사용되었다. 이것은 유효한 이름이 아니라 SyntaxError가 발생할 것이다.
- 두 번째 코드 : 객체 `word1`을 정의하는 코드다. `var`와 `function`이라는 예약어가 프로퍼티 키로 사용되었는데 해당 코드는 에러가 발생하지 않는다. 하지만 예상치 못한 에러가 발생할 수 있어 예약어를 프로퍼티 키로 사용하는 것은 좋지 않다.
- 세 번째 코드 : `objES5`라는 빈 객체를 생성하고 `keyES5`라는 변수에 문자열 `ES5`를 할당한다. 그리고 객체에 프로퍼티를 추가하는데 `keyES5` 변수의 값인 `ES5`가 프로퍼티 키로 사용된다. 그럼 `ES5`라는 프로퍼티 키가 객체에 추가되고, 프로퍼티 값으로 `world` 가 할당된다.
- 네 번째 코드 : `keyES6` 변수에 문자열 `HELL`을 할당한다. 그리고 객체 프로퍼티 키를 대괄호로 묶고, 대괄호 내부에 `keyES6` 변수를 넣어 계산된 프로퍼티 키를 생성

한다. 이렇게 하면 `keyES6` 변수의 값인 `HELL`이 객체의 프로퍼티 키로 사용된다. 그럼 `HELL`이라는 프로퍼티 키가 객체에 추가되고, 프로퍼티 값으로 `o` 가 할당된다.

- 다섯 번째 코드 : 객체의 프로퍼티 키는 일반적으로 문자열로 표현되는데 빈 문자열도 유효한 프로퍼티 키다.
- 여섯 번째 코드 : 객체의 프로퍼티 키는 문자열로 취급되므로 `1`, `2`, `3`은 문자열로 처리된다.
- 일곱 번째 코드 : 중복된 프로퍼티 키를 가지는 객체다. 중복된 프로퍼티 키를 가진 경우, 나중에 선언된 속성이 이전의 속성을 덮어쓴다.

4. 브라우저 환경과 Nodejs 환경을 준비하고 아래의 코드를 돌려봅시다.

```
var wind = {
'last-name' : 'park',
1: 10
};
wind.'last-name';
wind.last-name;
wind[last-name];
wind['last-name'];
wind.1;
wind.'1';
wind[1];
wind['1']
```

- 출력결과(브라우저 환경)

```
SyntaxError
NaN
ReferenceError
'park'
SyntaxError
SyntaxError
```

```
10
10
```

- `wind.last-name;` 해당 코드를 실행하면 `last`와 `name`을 개별적인 식별자로 판단하고 뺄셈 연산을 하면 숫자가 아니므로 `NaN`이 뜨는 것 같다.

- 출력결과(Node.js 환경)

```
SyntaxError
ReferenceError
ReferenceError
'park'
SyntaxError
SyntaxError
10
10
```

- `wind.last-name;` 해당 코드를 실행하면 `-` 는 프로퍼티 키로 사용할 수 없으므로 유효하지 않은 표현식으로 간주되어 `ReferenceError`가 발생하는 것 같다.

5. 반복문이랑 조건문을 이용한 프로그램

- 백준 10997
- 실행은 되는데 시간 초과가 난다. (고민해봤지만 지금 내 수준으론 못할 것 같다.)

```
const fs = require("fs");
const filePath = process.platform === "linux" ? "/dev/stdin" : "./input2.txt";
let input = fs.readFileSync(filePath).toString()

// 가로 세로 길이
let row = 4 * input - 1, col = 4 * input - 3;

// *을 담을 2차원 배열
const map = Array.from(Array(row), () => new Array(col).fill(" "));

let i, j;

function func(a, N) {
    // N이 1일때
    if (N == 1) {
```

```
        map[a][a] = map[a + 1][a] = map[a + 2][a] = "*";
    } else {
        // 세로 가로 길이
        let row = 4 * N - 1, col = 4 * N - 3;

        // 가로
        for (i = a; i < a + col; i++)
            map[a][i] = map[a + row - 1][i] = "*";

        // 세로
        for (i = a + 2; i < a + row - 1; i++)
            map[i][a] = map[i][a + col - 1] = "*";

        // 가로 세로 빼고 나머지
        map[a + 1][a] = map[a + 2][a + col - 2] = "*";

        func(a + 2, N - 1);
    }
}

// 입력 값이 1일때
if (input === 1) {
    console.log("*");
// 입력값이 1이 아닐때
} else {
    func(0, input);
    // 2차원 배열 출력
    for (i = 0; i < row; i++)
    {
        // * 뒤에 공백이 없어야 형식이 잘못됐다고 안떠서 그 예외처리
        if (i != 1) {
            for (j = 0; j < col; j++)
                process.stdout.write(map[i][j]);
            process.stdout.write("\n");
        } else {
            console.log("*");
        }
    }
}
```

6. 단축평가에 대해서 한번 예시코드 제외하고 코딩을 한번 시도해보세요. 해당 단어나 기타 로직에 대해서 뜻이나 이런거 다 조사해오셔도 됩니다.

- 배열 길이 검사

```
// if문을 사용한 코드
if (arr.length > 0) {
  var firstElement = arr[0];
}

// 단축평가를 사용한 코드
var firstElement = arr[0] || '배열이 비어있습니다.';
```

- 사용자 이름

```
// if문을 사용한 코드
var username = '';
var defaultUsername = 'Guest';

if (!username) {
  username = defaultUsername;
}

// 단축평가를 사용한 코드
var username = '';
var defaultUsername = 'Guest';
var user = username || defaultUsername;
```

- 단축평가는 논리 연산자를 사용하여 조건을 왼쪽부터 오른쪽으로 검사할 때, 중간에 검사 결과가 나오면 끝까지 가지 않고 검사 결과를 반환하는 동작을 의미한다. 주로 논리 연산자인 `&&` (논리곱)와 `||` (논리합)을 사용할 때 나타난다.
- AND 연산자(&&)는 조건문 검사 중 조건이 `false`로 판별되면 다른 조건을 평가하지 않고 즉시 `false`를 반환한다. 그리고 모든 조건이 `true` 일때 `true`를 반환한다.
- OR 연산자 (||)는 조건문 검사 중 조건이 `true`로 판별되면 다른 조건을 평가하지 않고 즉시 `true`를 반환한다. 그리고 모든 조건이 `false`일 경우 `false`를 반환한다.