

O p e n Y e a r R o u n d 3 조

조건문 반복문 객체

발표자

박현우
이현동
조현진

CONTENTS

01

조건문

- If / else if/ else
- ? 연산자
- Switch 문

02

반복문

- While 반복문
- Do-while 반복문
- For 반복문
- break, continue, 레이블

03

객체

- 객체
- 객체 생성
- 객체 접근

04

배열

- 배열 생성
- 배열 접근
- 배열 추가
- 배열 삭제
- 배열 복사

05

과제

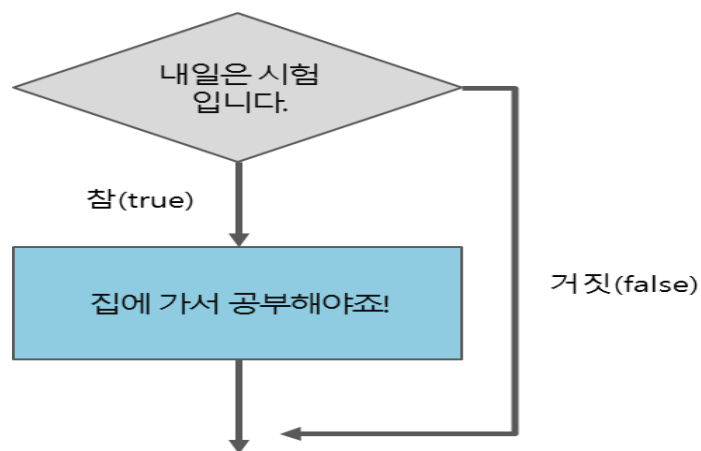
- 코드 리뷰

“

조건문

”

- 조건에 따라 다른 행동을 취해야 할 때 사용합니다.
- if문과 물음표 연산자(?)를 이용합니다.
- switch 문으로도 조건문을 나타낼 수 있습니다.



조건문

if문

- if(...)문은 괄호 안에 들어가는 조건을 평가하는데, 그 결과가 true인 경우 코드 블록이 실행됩니다.
- 구문이 한 줄인 경우 중괄호를 생략할 수 있습니다.
- 그러나, 가독성을 위하여 구문이 한 줄이라도 중괄호로 코드를 블록으로 감싸는 것을 추천합니다.

조건문

boolean 형으로의 변환

- if(...)문은 괄호 안에 들어가는 조건을 평가하는데, 그 결과를 boolean 값으로 변환합니다.
- 숫자 0, 빈 문자열 "", null, undefined, NaN은 false로 변환되어 코드 블록이 실행되지 않습니다.
- 그 외의 경우(0이 아닌 숫자, 비어 있지 않은 문자열)은 true로 변환되어 코드 블록이 실행됩니다.

조건문

```
1 let year = prompt('ECMAScript-2015 명세는 몇 년도에 출판되었을까요?', '');  
2  
3 // 2015를 입력받은 경우 아래 조건문이 참이 되어 실행됩니다.  
4 ✓ if (year == 2015) {  
5   |   alert( '정답입니다!' );  
6   | }
```

- 2015를 입력받은 경우 괄호 안의 `year == 2015`의 값이 `true`로 변환되고, `alert` 문이 출력됩니다.
- `alert` 구문이 한 줄이지만 중괄호로 가독성을 위하여 중괄호를 이용하여 감싼 것을 볼 수 있습니다.

조건문

```
1  if(''){
2      |    alert('절대 실행되지 않습니다');
3      |
4      }
5
6  if(0){
7      |    alert('절대 실행되지 않습니다');
8      |
9      }
10
11 if(null){
12     |    alert('절대 실행되지 않습니다');
13     |
14     }
15
16 if(undefined){
17     |    alert('절대 실행되지 않습니다');
18     |
19     }
20
21 if(1){
22     |    alert('무조건 실행됩니다. ');
23     |
24     }
```

조건문

else if문

- 유사하지만 약간씩 차이가 있는 조건 여러 개를 처리해야 하는 경우 사용합니다.

```
1  if (condition1){  
2  |    //condition1이 true이면 실행되는 부분  
3  |  
4  |}  
5  
6  else if(condition2){  
7  |    //condition1이 false고 condition2가 true이면 실행되는 부분  
8  |  
9  |}  
10  
11 else if(condition2){  
12 |    //condition1, condition2가 모두 false고 condition3가 true이면 실행되는 부분  
13 |  
14 |}  
15  
16 else{  
17 |    //위의 모든 condition이 false이면 실행되는 부분  
18 |  
19 |}
```


조건문

else문

- If문이 거짓이거나 else if 문이 있는 경우 모든 else if문이 거짓인 경우 실행됩니다.

```
1 let year = prompt('ECMAScript-2015 명세는 몇 년도에 출판되었을까요?', '');
2
3 if (year == 2015) {
4     alert( '정답입니다!' );
5 }
6
7 else {
8     alert( '오답입니다!' ); // 2015 이외의 값을 입력한 경우 }
9 }
```

조건문

조건부 연산자 '?'

- 조건에 따라 다른 값을 변수에 할당해줘야 하는 경우 쓰입니다.
- 물음표 연산자라고 불리며, 조건문을 사용한 경우보다 짧고 간결하게 표현할 수 있습니다.
- '?' 기호를 사용하며, 삼항 연산자라고 부르기도 합니다.

```
1 let result = condition ? value1 : value2;
```

평가 대상인 `condition` 이 `truthy`라면 `value1` 이, 그렇지 않으면 `value2` 가 반환됩니다.

조건문

다중 '?'

- 물음표 연산자를 여러 개 연결하면 복수의 조건을 처리할 수 있습니다.

```
1 let age = prompt('나이를 입력해주세요.', 18);  
2  
3 let message = (age < 3) ? '아기야 안녕?' :  
4   (age < 18) ? '안녕!' :  
5   (age < 100) ? '환영합니다!' :  
6   '나이가 아주 많으시거나, 나이가 아닌 값을 입력 하셨군요!';  
7  
8 alert( message );
```

물음표 연산자를 이런 방식으로 쓰는 걸 처음 본 분이라면 이 코드가 어떻게 동작하는지 파악하기 힘들 수 있습니다. 그러나 주의를 집중하고 보면, 단순히 여러 조건을 나열한 코드임에 불과하다는 것을 알 수 있습니다.

1. 첫 번째 물음표에선 조건문 `age < 3` 을 검사합니다.
2. 그 결과가 참이면 `'아기야 안녕?'` 를 반환합니다. 그렇지 않다면 첫 번째 콜론 `":"` 에 이어지는 조건문 `age < 18` 을 검사합니다.
3. 그 결과가 참이면 `'안녕!'` 를 반환합니다. 그렇지 않다면 다음 콜론 `":"` 에 이어지는 조건문 `age < 100` 을 검사합니다.
4. 그 결과가 참이면 `'환영합니다!'` 를 반환합니다. 그렇지 않다면 마지막 콜론 `":"` 이후의 표현식인 `'나이가 아주 많으시거나, 나이가 아닌 값을 입력 하셨군요!'` 를 반환합니다.

조건문

부적절한 '?'

- 물음표 연산자를 if 대용으로 쓰는 경우가 종종 있습니다.
- 가독성이 떨어지는 경우도 있으므로 주의해야 합니다.

```
1 let company = prompt('자바스크립트는 어떤 회사가 만들었을까요?', '');  
2  
3 (company == 'Netscape') ?  
4   alert('정답입니다!') : alert('오답입니다!');
```

조건 `company == 'Netscape'`의 검사 결과에 따라 `?` 뒤에 이어지는 첫 번째 혹은 두 번째 표현식이 실행되어 얼럿 창이 뜹니다.

위 예시에선 평가 결과를 변수에 할당하지 않고, 결과에 따라 실행되는 표현식이 달라지도록 하였습니다.

그런데 이런 식으로 물음표 연산자를 사용하는 것은 좋지 않습니다.

개발자 입장에선 `if` 문을 사용할 때 보다 코드 길이가 짧아진다는 점 때문에 물음표 `?`를 `if` 대용으로 쓰는 게 매력적일 순 있습니다. 하지만 이렇게 코드를 작성하면 가독성이 떨어집니다.

아래는 `if`를 사용해 변형한 코드입니다. 어느 코드가 더 읽기 쉬운지 직접 비교해 보시기 바랍니다.

조건문

switch문

- 복수의 if 조건문은 switch 문으로 바꿀 수 있습니다.
- switch문을 사용한 비교법은 특정 변수를 다양한 상황에서 비교할 수 있게 해주며, 코드 자체가 비교 상황을 잘 설명하는 장점도 있습니다.

조건문

- 기본 문법이며, 괄호 안에 다양한 자료형이 올 수 있습니다.

```
1 switch(x) {  
2     case 'value1': // if (x === 'value1')  
3         ...  
4         [break]  
5  
6     case 'value2': // if (x === 'value2')  
7         ...  
8         [break]  
9  
10    default:  
11        ...  
12        [break]  
13 }
```

조건문

- a의 값과 일치하는 해당하는 case 구문이 출력됩니다.

```
1 let a = 2 + 2;
2
3 switch (a) {
4   case 3:
5     alert( '비교하려는 값보다 작습니다.' );
6     break;
7   case 4:
8     alert( '비교하려는 값과 일치합니다.' );
9     break;
10  case 5:
11    alert( '비교하려는 값보다 큼니다.' );
12    break;
13  default:
14    alert( "어떤 값인지 파악이 되지 않습니다." );
15 }
```

조건문

- **break문이 없으면 해당하는 case부터 모든 구문이 실행됩니다.**

```
1 let a = 2 + 2;  
2  
3 switch (a) {  
4   case 3:  
5     alert( '비교하려는 값보다 작습니다.' );  
6   case 4:  
7     alert( '비교하려는 값과 일치합니다.' );  
8   case 5:  
9     alert( '비교하려는 값보다 큼니다.' );  
10  default:  
11    alert( "어떤 값인지 파악이 되지 않습니다." );  
12 }
```


“

반복문

”

- 여러 동작을 반복해야 하는 경우에 사용합니다.
- 상품 목록에서 상품을 차례대로 출력하거나 숫자를 1부터 10까지 하나씩 증가시키면서 동일한 코드를 반복 실행하는 경우 사용합니다.
- 반복문(loop)을 사용하면 동일한 코드를 여러 번 반복할 수 있습니다.

반복문

반복문을 사용하게 되면?

공격0
공격0
공격0
공격0
공격0
공격0
공격0
공격0
공격0

<반복문을 사용하지 않은 경우>

AutoLab



```
loop, 10
{
    공격0
}
```

<반복문을 사용한 경우>

AutoLab

반복문

while 반복문

- 괄호 안의 조건이 참인 경우 본문이 실행됩니다.
- 무한 루프에 빠질 가능성이 있으므로 종료 조건을 잘 살펴야 합니다.

```
1 while (condition) {  
2     // 코드  
3     // '반복문 본문(body)'이라 불림  
4 }
```

반복문

do-while 반복문

- 조건을 판단하는 while문이 맨 뒤에 위치합니다.
- 조건이 거짓인 경우라도 최소 한번의 실행은 보장합니다.

반복문

for 반복문

- while 반복문보다 복잡하지만 가장 많이 쓰이는 반복문입니다.

```
1 for (begin; condition; step) {  
2     // ... 반복문 본문 ...  
3 }
```

반복문

For 반복문

구성 요소

Begin	$i = 0$	반복문에 진입할 때 단 한 번 실행됩니다.
Condition	$i < 3$	반복마다 해당 조건이 확인됩니다. False면 반복문을 멈춥니다.
Body	Alert(i)	Condition이 truthy일 동안 계속해서 실행됩니다.
step	$i++$	각 반복의 body가 실행된 이후에 실행됩니다.

“

객체

”

- 자료구조의 한 형태로, 속성과 행위로 구분된다.
- JS는 객체기반 스크립트 언어로, 원시 타입을 제외한 것들은 모두 객체이다.
- JS의 객체는 키와 값으로 구성된 Property의 집합이다.

객체

1. 객체 생성하기

```
-  
var person = {  
  name: 'Lee',  
  gender: 'male',  
  sayHello: function () {  
    console.log('Hi! My name is ' + this.name);  
  }  
};  
  
console.log(typeof person); // object  
console.log(person); // {name: "Lee", gender: "male", sayHello: f}  
  
person.sayHello(); // Hi! My name is Lee
```


객체

1. 객체 생성하기

- 단축 프로퍼티

```
> hi = 30
30
> hyun = {hi}
{ hi: 30 }
> hyun
{ hi: 30 }
```

```
function makeUser(name, age) {
  return {
    name, // name: name 과 같음
    age,  // age: age 와 같음
    // ...
  };
}
```

객체

1. 객체 생성하기

```
- // 프로퍼티 추가
person.name = 'Lee';
person.gender = 'male';
person.sayHello = function () {
  console.log('Hi! My name is ' + this.name);
};

console.log(typeof person); // object
console.log(person); // {name: "Lee", gender: "male", sayHello: f}

person.sayHello(); // Hi! My name is Lee|
```

객체

1. 객체 생성하기

- 생성자 함수

```
// 생성자 함수
function Person(name, gender) {
  this.name = name;
  this.gender = gender;
  this.sayHello = function(){
    console.log('Hi! My name is ' + this.name);
  };
}

// 인스턴스의 생성
var person1 = new Person('Lee', 'male');
var person2 = new Person('Kim', 'female');

console.log('person1: ', typeof person1);
console.log('person2: ', typeof person2);
console.log('person1: ', person1);
console.log('person2: ', person2);

person1.sayHello();
person2.sayHello();
```

객체

1. 객체 생성하기 특징

- 생성자 함수 이름은 일반적으로 **대문자**로 시작한다.
- **this**는 생성자 함수가 생성할 **인스턴스(instance)**를 가리킨다.
- **this**에 연결(바인딩)되어 있는 프로퍼티와 메소드는 **public**
- 생성자 함수 내에서 선언된 일반 변수는 **private**

객체

2. 객체 프로퍼티 접근

- 프로퍼티 키

```
var person = {  
  'first-name': 'Hyun-dong',  
  // first-name: 'Hyun-dong', SyntaxError: Unexpected token -  
  'last-name': 'Lee',  
  gender: 'male',  
  1: 10,  
  function: 1 // OK. 하지만 예약어는 사용하지 말아야 한다.  
};  
  
console.log(person);
```

객체

2. 객체 프로퍼티 접근

- 값 읽기

```
var person = {  
  'first-name': 'Ung-mo',  
  'last-name': 'Lee',  
  gender: 'male',  
  1: 10  
};  
  
console.log(person);  
  
console.log(person.first-name); // NaN: undefined-undefined  
console.log(person[first-name]); // ReferenceError: first is not defined  
console.log(person['first-name']); // 'Ung-mo'  
  
console.log(person.gender); // 'male'  
console.log(person[gender]); // ReferenceError: gender is not defined  
console.log(person['gender']); // 'male'  
  
console.log(person['1']); // 10  
console.log(person[1]); // 10 : person[1] -> person['1']  
console.log(person.1); // SyntaxError
```

객체

2. 객체 프로퍼티 접근

- 갱신

```
var person = {  
  'first-name': 'Hyun-dong',  
  'last-name': 'Lee',  
  gender: 'male',  
};  
  
person['first-name'] = 'Kim';  
console.log(person['first-name'] ); // 'Kim'
```

객체

2. 객체 프로퍼티 접근

- 동적 생성

```
var person = {  
  'first-name': 'Hyun-dong',  
  'last-name': 'Lee',  
  gender: 'male',  
};  
  
person.isBirthDay= true;  
console.log(person.isBirthDay); // true
```


객체

2. 객체 프로퍼티 저그

- for-in 문

```
let user = {  
  name: "John",  
  age: 30,  
  isAdmin: true
```

```
for (key in user) {  
  // 각 :  
}
```

```
for (let key in user) {  
  // 키
```

```
  alert( key ); // name, age, isAdmin
```

```
  // 키에 해당하는 값
```

```
  alert( user[key] ); // John, 30, true
```

```
}
```

!행합니다.

객체

2. 객체 프로퍼티 접근

- for-in 문

```
// 배열 요소들만을 순회하지 않는다.  
var array = ['one', 'two'];  
array.name = 'my array';  
  
for (var index in array) {  
    console.log(index + ': ' + array[index]);  
}  
  
/*  
0: one  
1: two  
name: my array  
*/
```

객체

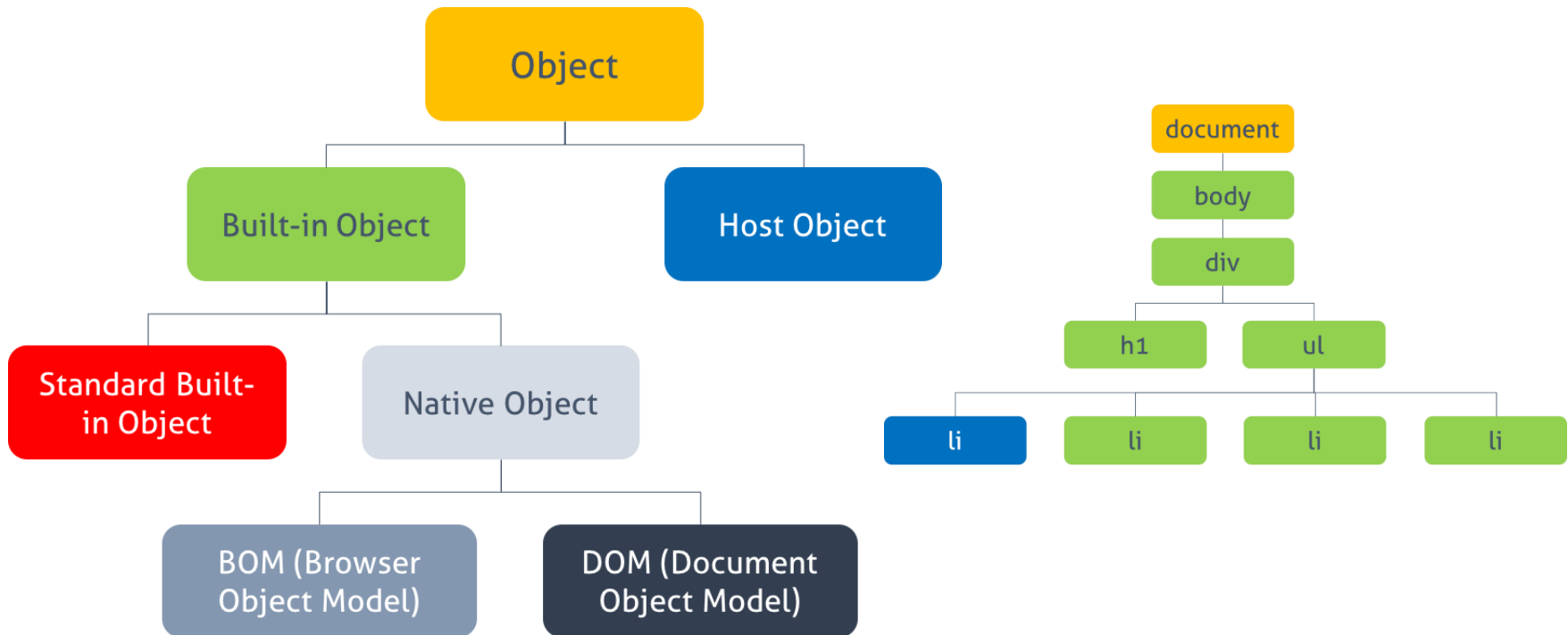
2. 객체 프로퍼티 접근

- 배열은 for-of 문

```
const array = [1, 2, 3];  
array.name = 'my array';  
  
for (const value of array) {  
    console.log(value);  
}  
  
/*  
1  
2  
3  
*/
```

객체

3. 객체 분류



“

배열

”

- 복수의 자료를 저장할 수 있는 자료구조입니다.
- 데이터는 순차적으로 저장되며 중복이 가능합니다.
- 저장된 데이터에는 인덱스를 이용하여 접근이 가능합니다..

배열

1. 배열 생성하기

- 방법1: new명령어 사용

ex) `var myArray = new Array();`

`var myArray = new Array(10);`

`var myArray = new Array(1,2,3, " 홍길동" , "아무개 ");`

배열

1. 배열 생성하기

- 방법2: 리터럴 표기법 사용

ex) `var myArray = new Array[] ;`

`var myArray = new Array [1,2,3, " 홍길동" , "아무개 "];`

배열

1.배열 생성하기

- *참고* 방법1과 방법2의 차이점

(IE 에서는 new Array() 구문에서 마지막 콤마를 사용하면 에러 입니다. 리터럴 표기법을 사용할 경우 IE 9 이상에서는 정상적으로 동작하고, IE8 이하에서는 요소의 개수가 하나 더 포함됩니다.

-> 리터럴 표기법 사용을 권장합니다!

배열

2. 배열 요소에 접근하기

- 배열 이름[인덱스] 를 통해 요소에 접근할 수 있습니다.
- 인덱스는 0부터 시작합니다.
- 배열이름.length 를 통해 배열의 길이를 구할 수 있습니다.

```
fruits.push('망고')  
// ["딸기", "바나나", "망고"]  
  
let pos = fruits.indexOf("바나나")  
// 1
```

배열

3. 배열에 요소 추가하기

- Push 를 이용하여 배열 끝에 항목을 추가할 수 있습니다.

```
let newLength = fruits.push('오렌지')  
// ["사과", "바나나", "오렌지"]
```

- Unshift 를 이용하여 배열 앞에 항목을 추가할 수 있습니다.

```
let newLength = fruits.unshift('딸기') // 앞에 추가  
// ["딸기", "바나나"]
```

4. 배열에 요소 삭제하기

- Pop 을 이용하여 배열 끝에 항목을 삭제할 수 있습니다.

```
let last = fruits.pop() // 끝에있던 '오렌지'를 제거  
// ["사과", "바나나"]
```

- Shift 를 이용하여 배열 앞에 항목을 삭제할 수 있습니다.

```
let first = fruits.shift() // 제일 앞의 '사과'를 제거  
// ["바나나"]
```

배열

5. 배열 복사하기

- Slice를 사용해서 배열을 복사할 수 있습니다.

```
let shallowCopy = fruits.slice() // 사본을 만드는 방법  
// ["딸기", "망고"]
```

6. 배열에서 사용할 수 있는 메소드 목록

메소드	기능
<code>concat()</code>	두개 또는 그 이상의 배열을 연결한 배열을 반환합니다.
<code>copyWithin()</code>	배열 내에서 첫 번째 인자 위치의 요소를 두 번째 인자 위치에 복사합니다.
<code>every()</code>	인자로 주어진 함수로 배열내의 모든 요소를 테스트해서 참일 경우 <code>true</code> 를 반환합니다.
<code>fill()</code>	배열내의 모든 요소의 값을 주어진 값으로 채웁니다.
<code>filter()</code>	인자로 주어진 함수로 배열내의 모든 요소를 테스트해서 참인 요소만으로 만들어진 배열을 반환합니다.
<code>find()</code>	인자로 주어진 함수로 배열내의 요소를 테스트해서 참인 첫 번째 요소를 반환 합니다. 찾지 못하면 <code>undefined</code> 를 반환합니다.
<code>findIndex()</code>	인자로 주어진 함수로 배열내의 요소를 테스트해서 참인 첫 번째 요소의 인덱스를 반환합니다. 찾지 못하면 <code>-1</code> 을 반환합니다.
<code>forEach()</code>	배열내의 모든 요소에 대해 인자로 주어진 함수를 실행합니다.
<code>indexOf()</code>	인자로 주어진 값과 같은 값을 가지는 첫 번째 요소의 인덱스를 반환합니다. 찾지 못하면 <code>-1</code> 을 반환합니다.

6. 배열에서 사용할 수 있는 메소드 목록

isArray()	인자로 주어진 객체가 배열이면 true를 반환합니다. <code>Array.isArray(object);</code>
join()	배열의 모든 요소를 인자로 주어진 분리자로 연결한 문자열을 반환합니다. 분리자를 주지 않으면 콤마(,)가 분리자입니다.
lastIndexOf()	인자로 주어진 값을 배열의 뒤에서 부터 찾아서 첫 번째로 발견된 요소의 인덱스를 반환합니다. 찾지 못하면 -1을 반환합니다.
map()	인자로 주어진 함수에 모든 각각의 요소를 적용한 값으로 이루어진 배열을 반환합니다.
pop()	배열의 마지막 요소를 제거하고, 제거된 요소를 반환합니다. 배열의 길이가 1 줄어 듭니다.
push()	배열의 끝에 인자로 주어진 값을 추가 합니다. 새 배열의 길이를 반환합니다.
reduce()	인자로 주어진 함수를 모든 요소에 적용하여 만들어진 단일 값을 반환합니다. 배열 요소에 대해 왼쪽에서 오른쪽으로 적용해 나갑니다. <code>array.reduce(function(total, currentValue, currentIndex, arr), initialValue)</code>
reduceRight()	<code>reduce()</code> 와 같은 기능을 배열 요소에 대해 오른쪽에서 왼쪽으로 적용해 나갑니다.
reverse()	배열 요소의 순서가 뒤집힌 배열을 반환합니다.

배열

6. 배열에서 사용할 수 있는 메소드 목록

shift()	배열에서 첫번째 요소를 제거하고, 제거된 요소를 반환합니다.
slice()	배열의 일부분을 잘라서 배열로 반환합니다. <code>array.slice(start, end)</code>
some()	인자로 주어진 함수를 배열의 요소에 대해서 적용해서 하나라도 참이면 true 를 반환합니다.
sort()	배열의 요소를 정렬합니다. 인자로 정렬에 사용될 비교 함수를 줄수 있습니다. 인자가 없으면 알파벳순으로 오름차순으로 정렬합니다.
splice()	배열에 요소를 추가 또는 삭제 합니다.
toString()	배열을 문자열로 변환하여 그 결과를 반환합니다. 분리자로 콤마가 사용됩니다.
unshift()	배열의 첫 번째 자리에 새 요소를 추가합니다. 새로운 배열의 길이를 반환합니다.
valueOf()	배열 그 자체를 반환합니다. <code>var b = a.valueOf();</code> 라면 <code>a === b</code> 입니다.

과제3 코드 리뷰

```
// 이메일 체크 정규식
function isEmail(asValue) {

    let regExp = /^[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*@[0-9a-zA-Z]([-_\.]?[0-9a-zA-Z])*\.[a-zA-Z]{2,3}$/i;
    return regExp.test(asValue); // 형식에 맞는 경우 true 리턴
}

function isCelluar(asValue) {

    let regExp = /^01(?:0|1|[6-9])-(?:\d{3}|\d{4})-\d{4}$/;
    return regExp.test(asValue); // 형식에 맞는 경우 true 리턴
}

function isJobPassword(asValue) {

    let regExp = /^(?=.*\d)(?=.*[a-zA-Z])[0-9a-zA-Z]{8,10}$/; // 8 ~ 10자 영문, 숫자 조합
    return regExp.test(asValue); // 형식에 맞는 경우 true 리턴
}
```


**THANK
YOU**