예습 : 4/26
복습 : 4/28, 5/2

1. 블록문
2. 조건문
2.1. if...else 문
2.2. switch 문
3. 반복문
3.1. for 문
3.2. while 문
3.3. do...while 문
4. break 문
5. continue 문

8. 제어문

제어문(Control flow statement)은 주어진 조건에 따라 코드 블록을 실행(조건문)하거나 반복 실행(반복문)할 때 사용한다. 일반적으로 코드는 위에서 아래 방향으로 순차적으로 실행된다. 제어문을 사용하면 코드의 실행 흐름을 인위적으로 제어할 수 있다.

하지만 코드의 실행 순서가 변경된다는 것은 단순하게 위에서 아래로 순차적으로 진행하는 직관적인 코드의 흐름을 혼란스럽게 만든다. 따라서 제어문은 코드의 흐름을 이해하기 어렵게 만들어 가독성을 해치는 단점이 있다. 가독성이 좋지 않은 코드는 오류를 발생시키는 원인이 된다. 나중에 살펴볼 forEach, map, filter, reduce와 같은 고차 함수를 사용한 함수형 프로그래밍 기법에서는 제어문의 사용을 억제하여 복잡성을 해결하려고 노력한다.

2보 전진을 위해서는 먼저 첫발을 내디뎌야 한다. 제어문을 바르게 이해하는 것은 코딩 스킬에 많은 영향을 준다. 특히 for 문은 매우 중요하므로 확실히 이해하도록 하자.

#1. 블록문

블록문(Block statement/Compound statement)는 0개 이상의 문을 중괄호로 묶은 것으로 코드 블록 또는 블록이라고 부르기도 한다. 자바스크립트는 블록문을 하나의 실행 단위로 취급한다. 블록문은 단독으로 사용할 수도 있으나 일반적으로 제어문이나 함수를 정의할 때 사용하는 것이 일반적이다.

아래는 블록문이 사용되는 다양한 예제이다. 문의 끝에는 세미 콜론(;)을 붙이는 것이 일반적이지만 블록문의 끝에는 세미콜론을 붙이지 않는다는 것에 주의하기 바란다.

JAVASCRIPT

```
// 블록문
{
```

```
  var foo = 10;
  console.log(foo);
}

// 제어문
var x = 0;
while (x < 10) {
  x++;
}
console.log(x); // 10

// 함수 선언문
function sum(a, b) {
  return a + b;
}
console.log(sum(1, 2)); // 3
```

2. 조건문

조건문(conditional statement)은 주어진 조건식(conditional expression)의 평가 결과에 따라 코드 블럭(블록문)의 실행을 결정한다. 조건식은 불리언 값으로 평가될 수 있는 표현식이다.

자바스크립트는 2가지의 조건문 if...else 문과 switch 문을 제공한다.
조건식은 표현식 안에 포함되는 것.
0도 조건식. 왜? true, false로 평가될 수 있으므로

2.1. if...else 문

if...else 문은 주어진 조건식(불리언 값으로 평가될 수 있는 표현식)의 평가 결과, 즉 논리적 참 또는 거짓에 따라 실행할 코드 블록을 결정한다. 조건식의 평가 결과가 참(true)일 경우, if 문 다음의 코드 블록이 실행되고 거짓(false)일 경우, else 문 다음의 코드 블록이 실행된다.

JAVASCRIPT

```
if (조건식) {
  // 조건식이 참이면 이 코드 블록이 실행된다.
} else {
```

```
  // 조건식이 거짓이면 이 코드 블록이 실행된다.
}
```

if 문의 조건식은 불리언 값으로 평가되어야 한다. 만약 if 문의 조건식이 불리언 값이 아닌 값으로 평가되면 자바스크립트 엔진에 의해 암묵적으로 데이터 타입이 불리언 값으로 강제 변환되어 실행할 코드 블록을 결정한다. 이에 대해서는 "9.2. 암묵적 타입 변환"에서 살펴볼 것이다.

조건식을 추가하여 조건에 따라 실행될 코드 블록을 늘리고 싶으면 else if 문을 사용한다.

JAVASCRIPT

```
if (조건식1) {
  // 조건식1이 참이면 이 코드 블록이 실행된다.
} else if (조건식2) {
  // 조건식2이 참이면 이 코드 블록이 실행된다.
} else {
  // 조건식1과 조건식2가 모두 거짓이면 이 코드 블록이 실행된다.
}
```

else if 문과 else 문은 옵션이다. 즉, 사용할 수도 있고 사용하지 않을 수도 있다. if 문과 else 문은 2번 이상 사용할 수 없지만 else if 문은 여러 번 사용할 수 있다.

JAVASCRIPT

```
var num = 2;
var kind;

// if 문
if (num > 0) {
  kind = '양수'; // 음수를 구별할 수 없다
}
console.log(kind); // 양수

// if…else 문
if (num > 0) {
  kind = '양수';
} else {
  kind = '음수'; // 0은 음수가 아니다.
}
```

```
console.log(kind); // 양수

// if…else if 문   ->삼항 조건 연산자로 바꿔보기
if (num > 0) {
  kind = '양수';
} else if (num < 0) {
  kind = '음수';
} else {
  kind = '영';
}
console.log(kind); // 양수
```

```
var x = 1;
var kind = x ? (x > 0 ? '양수' : '음수') : '0';
console.log(kind);
```

만약 코드 블록 내의 문이 하나뿐이라면 중괄호를 생략할 수 있다.

JAVASCRIPT

```
var num = 2;
var kind;

if (num > 0)      kind = '양수';
else if (num < 0) kind = '음수';
else              kind = '영';

console.log(kind); // 양수
```

대부분의 if...else 문은 삼항 조건 연산자("7.4. 삼항 조건 연산자" 참고)로 바꿔 쓸 수 있다. 아래 예제를 살펴보자.

JAVASCRIPT

<예제 풀기>

```
// x가 짝수이면 문자열 '짝수'를 반환하고 홀수이면 문자열 '홀수'를 반환한다.
var x = 2;
var result;

if (x % 2) { // 2 % 2는 0이다. 이때 0은 false로 암묵적 강제 변환된다.
  result = '홀수';
} else {
  result = '짝수';
}
```

짝수는 2로 나누었을 때 나머지가 0이 나와야 함. 즉 false 값
짝수는 false이므로 else 문 뒤에 '짝수'를 입력해야 함

```
console.log(result); // 짝수
```

위 예제는 아래와 같이 삼항 조건 연산자로 바꿔 쓸 수 있다.

JAVASCRIPT

```
// x가 짝수이면 문자열 '짝수'를 반환하고 홀수이면 문자열 '홀수'를 반환한다.
var x = 2;

// 0은 false로 취급된다.
var result = x % 2 ? '홀수' : '짝수';

console.log(result); // 짝수
```

위 예제는 두가지 경우의 수('홀수' 또는 '짝수')를 갖는 경우이다. 만약 세가지 경우의 수(양수, 음수, 영)를 갖는 경우는 아래와 같이 바꿔 쓸 수 있다.

JAVASCRIPT

```
var num = 2;

// 0은 false로 취급된다.
var kind = num ? (num > 0 ? '양수' : '음수') : '영';

console.log(kind); // 양수
```

`num > 0 ? '양수' : '음수'` 는 표현식이다. 즉, 삼항 조건 연산자는 값으로 평가되는 표현식을 만든다. 하지만 if...else 문은 표현식이 아닌 문이다. 따라서 삼항 조건 연산자 표현식은 값처럼 사용할 수 있기 때문에 변수에 할당할 수 있다. 하지만 if...else 문은 값처럼 사용할 수 없기 때문에 변수에 할당할 수 없다는 차이가 있다.

2.2. switch 문

switch 문은 주어진 표현식을 평가하여 그 값과 일치하는 표현식을 갖는 case 문으로 실행 순서를 이동시킨다. case 문은 상황(case)을 의미하는 표현식을 지정하고 콜론으로 마친다. 그리고 그 뒤에 실행할 문들을 위치시킨다.

switch 문의 표현식과 일치하는 표현식을 갖는 case 문이 없다면 실행 순서는 default 문으로 이동한다. default 옵션으로 사용할 수도 있고 사용하지 않을 수도 있다.

JAVASCRIPT

```
switch (표현식) {
  case 표현식1:
    switch 문의 표현식과 표현식1이 일치하면 실행될 문;
    break;
  case 표현식2:
    switch 문의 표현식과 표현식2가 일치하면 실행될 문;
    break;
  default:
    switch 문의 표현식과 일치하는 표현식을 갖는 case 문이 없을 때 실행될 문;
}
```

if...else 문의 조건식은 반드시 불리언 값으로 평가되지만 switch 문의 표현식은 불리언 값보다는 문자열, 숫자 값인 경우가 많다. if...else 문은 논리적 참, 거짓으로 실행할 코드 블록을 결정한다. switch 문은 논리적 참, 거짓보다는 다양한 상황(case)에 따라 실행할 코드 블록을 결정할 때 사용한다.

아래 예제를 살펴보자. switch 문의 표현식, 즉 변수 month의 평가 결과인 숫자 값 11과 일치하는 case 문으로 실행 순서가 이동한다.

JAVASCRIPT

```
// 월을 영어로 변환한다. (11 → 'November')
var month = 11;
var monthName;

switch (month) {
  case 1:
    monthName = 'January';
  case 2:
    monthName = 'February';
  case 3:
    monthName = 'March';
  case 4:
```

```
    monthName = 'April';
  case 5:
    monthName = 'May';
  case 6:
    monthName = 'June';
  case 7:
    monthName = 'July';
  case 8:
    monthName = 'August';
  case 9:
    monthName = 'September';
  case 10:
    monthName = 'October';
  case 11:
    monthName = 'November';
  case 12:
    monthName = 'December';
  default:
    monthName = 'Invalid month';
}

console.log(monthName); // Invalid month
```

그런데 위 예제를 실행해 보면 'November'가 출력되지 않고 'Invalid month'가 출력된다. 이는 switch 문의 표현식의 평가 결과와 일치하는 case 문으로 실행 순서가 이동하여 문을 실행한 것은 맞지만, 문을 실행한 후 switch 문을 탈출하지 않고 switch 문이 끝날 때까지 이후의 모든 case 문과 default 문을 실행했기 때문이다. 이를 **폴스루**(fall through)라 한다. 다시 말해 변수 monthName에 'November'가 할당된 후 switch 문을 탈출하지 않고 연이어 'December'가 재할당되고 마지막으로 'Invalid month'가 재할당되었다.

결과가 이러한 이유는 case 문에 해당하는 문의 마지막에 break 문을 사용하지 않았기 때문이다. break 키워드로 구성된 break 문은 코드 블록에서 탈출하는 역할을 한다. break 문이 없다면 case 문의 표현식과 일치하지 않더라도 실행 순서는 다음 case 문으로 연이어 이동한다. 올바른 switch 문은 아래와 같다.

JAVASCRIPT

```
// 월을 영어로 변환한다. (11 → 'November')
var month = 11;
var monthName;
```

```
switch (month) {
  case 1:
    monthName = 'January';
    break;
  case 2:
    monthName = 'February';
    break;
  case 3:
    monthName = 'March';
    break;
  case 4:
    monthName = 'April';
    break;
  case 5:
    monthName = 'May';
    break;
  case 6:
    monthName = 'June';
    break;
  case 7:
    monthName = 'July';
    break;
  case 8:
    monthName = 'August';
    break;
  case 9:
    monthName = 'September';
    break;
  case 10:
    monthName = 'October';
    break;
  case 11:
    monthName = 'November';
    break;
  case 12:
    monthName = 'December';
    break;
  default:
    monthName = 'Invalid month';
}
```

```
console.log(monthName); // November
```

default 문에는 break 문을 생략하는 것이 일반적이다. default 문은 switch 문의 가장 마지막에 위치하므로 default 문의 실행이 종료하면 switch 문을 빠져나간다. 따라서 별도로 break 문이 필요 없다.

break 문을 생략한 폴스루(fall through)가 유용한 경우도 있다. 아래 예제와 같이 폴스루를 활용해 여러 개의 case 문을 하나의 조건으로 사용할 수도 있다. 아래는 윤년인지 판별해서 2월의 일수를 계산하는 예제다.

JAVASCRIPT

```
var year = 2000; // 2000년은 윤년으로 2월이 29일이다.
var month = 2;
var days = 0;

switch (month) {
  case 1: case 3: case 5: case 7: case 8: case 10: case 12:
    days = 31;
    break;
  case 4: case 6: case 9: case 11:
    days = 30;
    break;
  case 2:
    // 윤년 계산 알고리즘
    // 1. 년도가 4로 나누어 떨어지는 해는 윤년(2000, 2004, 2008, 2012, 2016, 2020…)
    // 2. 그 중에서 년도가 100으로 나누어 떨어지는 해는 평년(2000, 2100, 2200...)
    // 3. 그 중에서 년도가 400으로 나누어 떨어지는 해는 윤년(2000, 2400, 2800...)
    days = ((year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0)) ? 29
: 28;
    break;
  default:
    console.log('Invalid month');
}

console.log(days); // 29
```

switch 문은 case, default, break 등 다양한 키워드를 사용해야 하고 상황에 따라 실행될 코드 블록이 중괄호로 묶어 있지 않으며 폴스루가 발생하는 등 문법도 복잡하다. 만약 if...else 문으로 해결할 수 있다

면 switch 문보다 if...else 문을 사용하는 편이 좋다. 하지만 if...else 문보다 switch 문을 사용했을 때 가독성이 더 좋다면 switch 문을 사용하는 편이 좋다.

3. 반복문

반복문(Loop statement)은 주어진 조건식의 평가 결과가 참인 경우 코드 블럭을 실행한다. 그 후 조건식을 다시 검사하여 여전히 참인 경우 코드 블록을 다시 실행한다. 이는 조건식이 거짓일 때까지 반복된다.

자바스크립트는 3가지의 반복문 for 문, while 문, do...while 문을 제공한다. 그 외에도 for..in 문, ES6에서 새롭게 도입된 for...of 문이 있다. for..in 문과 for...of 문에 대해서는 나중에 살펴보기로 하자.

3.1. for 문

for 문은 조건식이 거짓으로 판별될 때까지 코드 블록을 반복 실행한다. 가장 일반적으로 사용되는 반복문의 형태는 아래와 같다. 변수 선언문의 변수 이름은 반복을 의미하는 iteration의 i를 사용하는 것이 일반적이다.

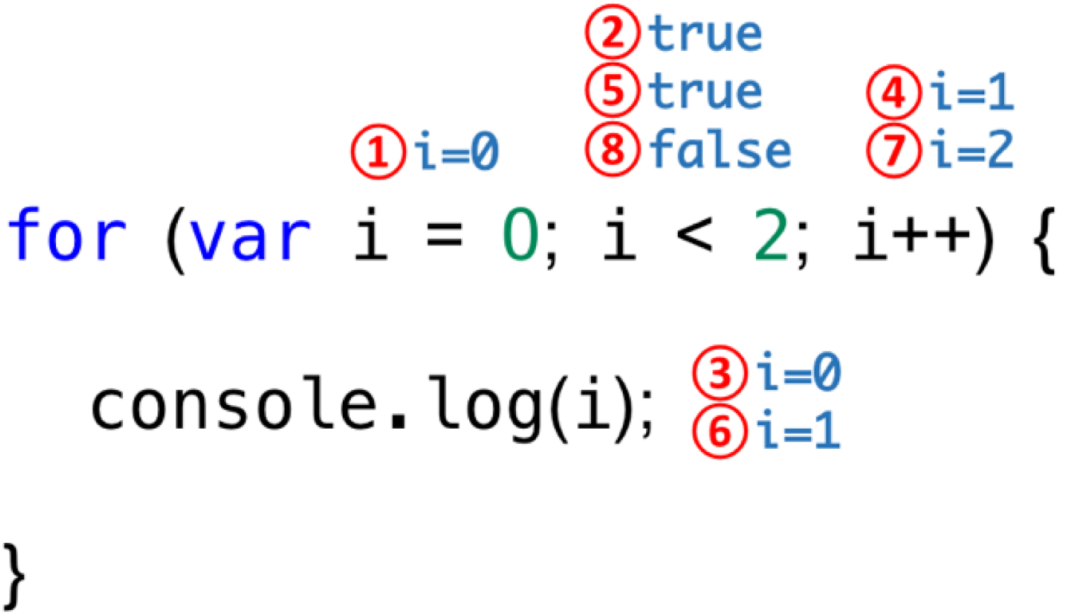
JAVASCRIPT

```
for (변수 선언문 또는 할당문; 조건식; 증감식) {
  조건식이 참인 경우 반복 실행될 문;
}
```

for 문은 매우 중요하다. 아직 for 문에 익숙하지 않다면 많은 연습을 통해 확실히 이해하기를 권장한다. 아래 예제를 통해 for 문이 어떻게 동작하는지 살펴보자.

JAVASCRIPT

```
for (var i = 0; i < 2; i++) {
  console.log(i);
}
```



for문의 실행 순서

위 예제의 for 문은 변수 i가 0으로 초기화된 상태에서 시작하여 i가 2보다 작을 때까지 코드 블록을 2번 반복 실행한다. for 문의 실행 순서를 따라가며 어떻게 동작하는지 살펴보자.

1. for 문을 실행하면 가장 먼저 변수 선언문 var i = 0이 실행된다. 변수 선언문은 단 한번만 실행된다.
2. 변수 선언문의 실행이 종료되면 조건식으로 실행 순서가 이동한다. 현재 변수 i는 0이므로 조건식의 평가 결과는 true다.
3. 조건식의 평가 결과가 true이므로 실행 순서가 코드 블록으로 이동하여 실행된다. 증감문으로 실행 순서가 이동하는 것이 아니라 코드 블록으로 실행 순서가 이동하는 것에 주의하자.
4. 코드 블록의 실행이 종료하면 증감식으로 실행 순서가 이동한다. 증감식 i++가 실행되어 i는 1이 된다.
5. 증감식 실행이 종료되면 다시 조건식으로 실행 순서가 이동한다. 변수 선언문으로 실행 순서가 이동하는 것이 아니라 조건식으로 실행 순서가 이동하는 것에 주의하자. 변수 선언문은 단 한번만 실행된다. 현재 변수 i는 1이므로 조건식의 평가 결과는 true다.
6. 조건식의 평가 결과가 true이므로 실행 순서가 코드 블록으로 이동하여 실행된다.
7. 코드 블록의 실행이 종료하면 증감식으로 실행 순서가 이동한다. 증감식 i++가 실행되어 i는 2가 된다.
8. 증감식 실행이 종료되면 다시 조건식으로 실행 순서가 이동한다. 현재 변수 i는 2이므로 조건식의 평가 결과는 false다. 조건식의 평가 결과가 false이므로 for 문의 실행이 종료된다.

아래 예제는 위 예제를 역으로 반복하는 for 문이다. 변수 i가 1으로 초기화된 상태에서 시작하여 i가 0보다 같거나 커질 때까지 코드 블록을 2번 반복 실행한다.

JAVASCRIPT

```
for (var i = 1; i >= 0; i--) {
  console.log(i);
```

```
}
```

for 문의 변수 선언문, 조건식, 증감식은 모두 옵션이므로 반드시 사용할 필요는 없다. 어떤 식도 선언하지 않으면 무한 루프가 된다.

JAVASCRIPT

```
// 무한루프
for (;;) { ... }
```

<예제 풀기>

for 문 내에 for 문을 중첩해 사용할 수 있다. 아래는 두 개의 주사위를 던졌을 때, 두 눈의 합이 6이 되는 모든 경우의 수를 출력하는 예제다.

JAVASCRIPT

```
for (var i = 1; i ⩽ 6; i++) {
  for (var j = 1; j ⩽ 6; j++) {
    if (i + j === 6) console.log(`[${i}, ${j}]`);
  }
}
```

출력 결과는 아래와 같다.

CODE

```
[1, 5]
[2, 4]
[3, 3]
[4, 2]
[5, 1]
```

3.2. while 문

while 문은 주어진 조건식의 평가 결과가 참이면 코드 블록을 계속해서 반복 실행한다. 조건문의 평가 결과가 거짓이 되면 실행을 종료한다. 만약 조건식의 평가 결과가 불리언 값이 아니면 불리언 값으로 강제

변환되어 논리적 참, 거짓을 구별한다

JAVASCRIPT **<예제 풀기> count가 3보다 작을 때까지 코드 블록 반복 실행**

```
var count = 0; 1. 초기화

// count가 3보다 작을 때까지 코드 블록을 계속 반복 실행한다.
while (count < 3) { 2. 조건식
  console.log(count);
  count++; 3. 증감식
} // 0 1 2
```

조건식의 평가 결과가 언제나 참이면 무한루프가 된다.

JAVASCRIPT

```
// 무한루프
while (true) { ... }

// 무한루프
for (;;) { ... }
```

무한루프를 탈출하기 위해서는 코드 블럭 내에 if 문으로 탈출 조건을 만들고 break 문으로 코드 블럭을 탈출한다.

JAVASCRIPT

```
var count = 0;

// 무한루프
while (true) {
  console.log(count);
  count++;
  // count가 3이면 코드 블록을 탈출한다.
  if (count === 3) break;
} // 0 1 2
```

3.3. do...while 문

do...while 문은 코드 블록을 먼저 실행하고 조건식을 평가한다. 따라서 코드 블록은 무조건 한번 이상 실행된다.

JAVASCRIPT

```
var count = 0;

// count가 3보다 작을 때까지 코드 블록을 계속 반복 실행한다.
do {
  console.log(count);
  count++;
} while (count < 3); // 0 1 2
```

4. break 문

switch 문과 while 문에서 살펴보았듯이 break 문은 코드 블록을 탈출한다. 좀 더 정확히 표현하자면 코드 블록을 탈출하는 것이 아니라 레이블 문, 반복문(for, for...in, for...of, while, do...while) 또는 switch 문의 코드 블록을 탈출한다. 레이블 문, 반복문, switch 문의 코드 블록 이외에 break 문을 사용하면 SyntaxError(문법 에러)가 발생한다.

JAVASCRIPT

```
if (true) {
  break; // Uncaught SyntaxError: Illegal break statement
}
```

레이블 문(Label statement)이란 식별자가 붙은 문을 말한다.

JAVASCRIPT

```
// foo라는 레이블 식별자가 붙은 레이블 문
foo: console.log('foo');
```

레이블 문은 프로그램의 실행 순서를 제어하기 위해 사용한다. 사실 switch 문의 case 문과 default 문도 레이블 문이다. 레이블 문을 탈출하려면 break 문에 레이블 식별자를 지정한다.

JAVASCRIPT

```
// foo라는 식별자가 붙은 레이블 블록문
foo: {
  console.log(1);
  break foo; // foo 레이블 블록문을 탈출한다.
  console.log(2);
}

console.log('Done!');
```

중첩된 for 문의 내부 for 문에서 break 문을 실행하면 내부 for 문을 탈출하여 외부 for 문으로 진입한다. 이때 내부 for 문이 아닌 외부 for 문을 탈출하려면 레이블 문을 사용한다.

JAVASCRIPT

```
// outer라는 식별자가 붙은 레이블 for 문
outer: for (var i = 0; i < 3; i++) {
  for (var j = 0; j < 3; j++) {
    // i + j === 3이면 outer라는 식별자가 붙은 레이블 for 문을 탈출한다.
    if (i + j === 3) break outer;
    console.log('inner ' + j);
  }
}

console.log('Done!');
```

중첩된 for 문을 외부로 탈출할 때 레이블 문은 유용하지만 그 외의 경우 레이블 문은 일반적으로 권장하지 않는다. 레이블 문을 사용하면 프로그램의 흐름이 복잡해져서 가독성이 나빠지고 오류를 발생시킬 가능성이 높아지기 때문이다.

break 문은 레이블 문 뿐만이 아니라 반복문, switch 문에서도 사용할 수 있다. 이 경우에는 break 문에 레이블 식별자를 지정하지 않는다. break 문은 반복문을 더 이상 진행하지 않아도 될 때 불필요한 반복을 회피할 수 있어 유용하다.

아래는 문자열에서 특정 문자의 인덱스(위치)를 검색하는 예제이다.

<예제 풀기-'Hello World.'문자열에서 개별 문자가 'l'이면 반복문을 탈출 (아래 힌트 1도 보지 말기)

JAVASCRIPT

```
var string = 'Hello World.';
var search = 'l';
var index;

// 문자열은 유사배열이므로 for 문으로 순회할 수 있다.
for (var i = 0; i < string.length; i++) {
  // 문자열의 개별 문자가 'l'이면
  if (string[i] === search) {
    index = i;
    break; // 반복문을 탈출한다.
  }
}

console.log(index); // 2

// 참고로 String.prototype.indexOf 메소드를 사용해도 같은 동작을 한다.
console.log(string.indexOf(search)); // 2
```

length를 제공하는 이유? for 문을 돌기 위해 필요한것 .
배열에서도 length 제공
length가 있고 for 문으로돌릴 수 있는 애들 : 유사배열

5. continue 문

continue 문은 반복문의 코드 블록 실행을 현 지점에서 중단하고 반복문의 증감식으로 이동한다. break 문처럼 반복문을 탈출하지는 않는다.

아래는 문자열에서 특정 문자의 개수를 카운트하는 예제이다.

<예제 풀기-'Hello World.'에서 'l'의 총 개수 카운트>

JAVASCRIPT

```
var string = 'Hello World.';
var search = 'l';
var count = 0;

// 문자열은 유사배열이므로 for 문으로 순회할 수 있다.
for (var i = 0; i < string.length; i++) {
  // 'l'이 아니면 현 지점에서 실행을 중단하고 반복문의 증감식으로 이동한다.
  if (string[i] !== search) continue;
  count++; // continue 문이 실행되면 이 문은 실행되지 않는다.
}
```

```
console.log(count); // 3

// 참고로 String.prototype.match 메소드를 사용해도 같은 동작을 한다.
const regexp = new RegExp(search, 'g');
console.log(string.match(regexp).length); // 3
```

위 예제의 for 문은 아래와 동일하게 동작한다.

JAVASCRIPT

```
for (var i = 0; i < string.length; i++) {
  // 'l'이면 카운트를 증가시킨다.
  if (string[i] === search) count++;
}
```

위와 같이 if 문 내에서 실행해야 할 코드가 한 줄이라면 continue 문을 사용했을 때보다 간편하며 가독성도 좋다. 하지만 if 문 내에서 실행해야 할 코드가 길다면 들여쓰기가 한 단계 더 깊어지므로 continue 문을 사용하는 것이 가독성이 더 좋다.

JAVASCRIPT

```
// continue 문을 사용하지 않으면 if 문 내에 코드를 작성해야 한다.
for (var i = 0; i < string.length; i++) {
  // 'l'이면 카운트를 증가시킨다.
  if (string[i] === search) {
    count++;
    // code
    // code
    // code
  }
}

// continue 문을 사용하면 if 문 밖에 코드를 작성할 수 있다.
for (var i = 0; i < string.length; i++) {
  // 'l'이 아니면 카운트를 증가시키지 않는다.
  if (string[i] !== search) continue;

  count++;
```

```
    // code
    // code
    // code
}
```