Javascript 반복문

키워드 : 배열 (https://www.w3schools.com/js/js_arrays.asp),
for 구문 (https://www.w3schools.com/js/js_loop_for.asp),
while 구문 (https://www.w3schools.com/js/js_loop_while.asp),
break & continue 구문 (https://www.w3schools.com/js/js_break.asp)

컴퓨터는 우리가 잘 알고 있듯이 굉장히 빠른 연산 속도를 자랑합니다. 컴퓨터를 새로 구매할 때 CPU의 성능 및 메모리의 성능 등을 확인하고 구매하는 이유도 이러한 컴퓨터의 연산 속도를 극대화해서 사용하기 위함이라고 할 수 있습니다.
이러한 컴퓨터의 빠른 연산 속도를 활용한 코드 구문이 이번 장에서 정리할 반복문입니다.

반복문은 특정 조건이 만족할 경우에 실행하고 싶은 반복 코드를 하나의 블록({})으로 묶는 코드의 흐름 제어 구문입니다.

for 구문

우선, 흐름 제어를 위한 코드 블록 중 하나인 for 구문에 대해서 살펴보도록 하겠습니다.

`for` 구문은 아래와 같은 기본 문법 구조를 갖습니다.

```
//기본 형태
for(구문1; 구문2; 구문3;){
    //반복 실행 코드
}

//코드 예시
for(var i = 0; i < 5; ++i){
    console.log(i); //0, 1, 2, 3, 4
}
```

위 기본 형태를 통해 `for` 구문은 다음과 같이 세가지로 구성 되는 것을 알 수 있습니다.

명칭	의미
구문1	반복 실행 코드가 실행되기 전, 한 번 동작하는 구문(초기화)
구문2	반복 실행 코드가 실행되기 위한 조건 구문
구문3	반복 실행 코드가 실행되는 동안 계속해서 실행되는 구문

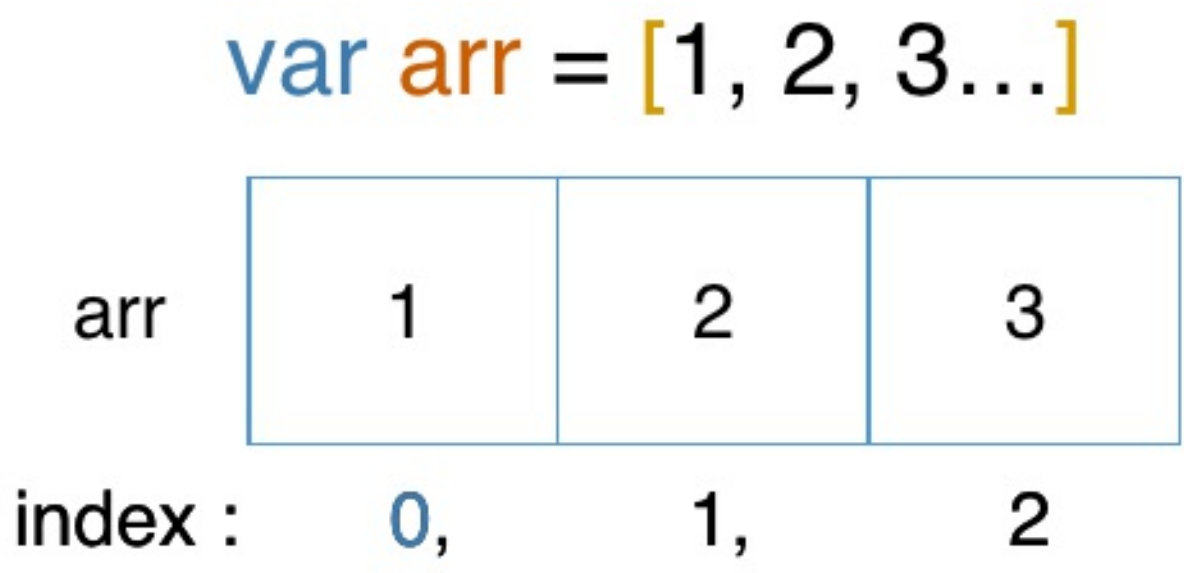
마찬가지로 위 코드 예시를 통해 반복문을 하나하나 살펴보도록 하겠습니다.

> `var i = 0`: 반복문이 실행되기 전에 변수를 초기화합니다.

> `i < 5`: i 변수가 5보다 작을 경우까지 반복 코드를 실행하도록 조건 구문을 추가합니다.

> `++i`: "console.log"로 i 변수를 콘솔로 남기는 동안 매번 1씩 증가시켜 반복 구문이 5번 반복되도록 합니다.

각각의 구문은 세미 콜론(;)을 작성해서 해당 코드의 끝을 알립니다.



일반적으로 for 구문은 배열과 같이 여러 값을 하나로 묶는 자료형의 값을 순차적으로 인덱싱(인덱스를 통해 값을 하나하나 접근하는 방법) 하기 위해 사용되곤 합니다. 0번 째부터 시작하는 배열의 값을 하나씩 순차적으로 접근하기 위한 방법으로 반복문을 사용하는 것보다 좋은 방법은 없습니다.

인덱스를 이용한 접근이 가능하다는 점은 우리가 반복문에서 인덱스의 접근을 자유자재로 변경할 수 있다는 점을 시사합니다.

아래 코드는 인덱스 접근을 위해 사용할 수 있는 `var i` 변수의 값을 어떻게 사용하느냐에 따라 하나의 배열을 서로 다른 방식으로 접근할 수 있음을 보여줍니다.

```
//1~10 까지 정수 값을 하나로 묶는 배열 선언
var testArr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var len = testArr.length; //배열의 길이 10
for(var i = 0; i < len; i += 2){
    console.log(testArr[i]); // 1, 3, 5, 7, 9 (홀수만 출력)
}

for(var i = len - 1; i >= 0; --i){
    console.log(testArr[i]); // 10, 9, 8...1 (거꾸로 출력)
}
```

while 구문

다음은 흐름 제어를 위한 코드 블록 중 하나인 while 구문에 대해서 살펴보도록 하겠습니다.

`while` 구문 구문은 아래와 같은 기본 문법 구조를 갖습니다.

```
//기본 형태
while (조건){
    //반복 실행 코드
}

//코드 예시
var i = 1;
while (i < 11) {
    console.log(i); //1, 2, 3...10
    i++;
}
```

while 구문은 for 구문을 이해하셨다면 어렵지 않게 이해할 수 있을 거라 생각됩니다. 위 기본 형태와 코드 예시를 통해 while 구문은 특정 조건에 해당하는 경우 코드가 반복 실행되게 됩니다.

while 구문은 for 구문과 다르게 do~while 구문을 통해 조건 검사를 하기 전 **무조건 한 번 실행**하는 코드를 작성할 수 있습니다.

```
//기본형태
do {
    //반복 실행 코드
}while(조건)

//코드 예시
var i = 1;
do {
    console.log(i); //1, 2, 3...10
    i++;
}while(i < 11);
```

위 코드는 **최소한 한 번은 실행**되는 코드라고 할 수 있습니다. 조건을 검사하는 i < 11 코드 구문이 실행 코드보다 아래 작성된 것을 통해 어렵지 않게 이해할 수 있습니다. 단, 적절한 종료 조건을 작성해주지 않으면 안 됩니다. 이와 관련된 내용은 아래 무한반복 절에서 더 살펴보도록 하겠습니다.

for 구문 과 break 구문은 모두 반복을 위해 사용되는 코드 구문입니다. for 구문에서 "구문1"과 "구문3"이 없는 형태가 while 구문이라고 할 수 있습니다. 사실, for 구문에서도 "구문1"과 "구문3"을 생략할 수 있습니다. 하지만, 우리 강의에선 코드의 일관성을 위해서 다루지 않습니다. 또한, while 구문 같은 경우는 조건에서 처리할 변수를 while 구문 이전에 선언한다는 점도 기억해주세요!

중첩 for, while 구문

위 코드의 예시는 모두 반복 구문이 하나로만 구성된 단일 반복 구문입니다. 하지만, 실제 코드를 작성할 때 하나의 for 구문 안에(혹은 while 구문 안에) 또 다른 반복 구문을 추가해서 중첩된 구조로 작성할 수 있습니다. 예를 들면 다음과 같습니다.

```
//1번 for 구문
for(var i = 0; i < 5; ++i){
    //2번 for 구문
    for(var j = 0; j < 5; ++j){
        console.log(j); // 0, 1, 2, 3, 4 -> 5번 출력
    }
}
```

위 코드는 0, 1, 2, 3, 4를 5번 출력하는 코드입니다. 우선, 바깥쪽 1번 for 구문이 실행될 때마다 안 쪽 2번 for 구문이 반복이 완료될 때까지 실행되는 구조입니다. 즉, 변수 i 값이 1 증가할 때마다 안 쪽 2번 for 구문의 j 값은 0~4 까지 모두 반복하는 작업을 실행하며, 이 작업을 바깥쪽 1번 for 구문의 반복 중지 조건을 만족할 때까지 반복하게 됩니다.

계속해서 반복문과 관련된 내용에 대해서 더 살펴보도록 하겠습니다.

break와 continue 구문

지난 장에서 우리는 조건문에 대해서 살펴보았습니다. 조건문을 이야기할 때 마지막 부분에서 break 키워드에 대해서 잠깐 언급했습니다. 이번 장에서는 반복문에서 사용할 수 있는 키워드인 break 구문과 continue 구문에 대해서 알아보도록 하겠습니다.

사실 우리는 이미 switch~case구문을 이야기할 때 특정 case를 찾았다면, 더 이상 코드를 실행할 필요 없이 switch~case 구문의 블록을 벗어나게 하기 위해 break 구문 사용할 수 있다고 알고 있습니다. 마찬가지로, 반복문에서도 break 구문을 이용해서 특정 반복 코드 구문을 벗어나게 할 수 있습니다.

아래 코드 예시는 break 구문을 이용한 for 구문과 while 구문을 벗어나게 하는 코드입니다.

```
//1번
for(var i = 0; i < 5; ++i){
    console.log(i); //0, 1, 2
    if(i == 2){
        break;
    }
}
//0, 1, 2 까지 출력 후 아래 코드를 실행
console.log("for문 반복 중지!"); //for문 반복 중지!

//2번
var i = 10;
while(i > 0){
    console.log(i); //10, 9, 8
    i--;
    if(i == 7){
        break;
    }
}
//10, 9, 8 까지 출력 후 아래 코드를 실행
console.log("while문 반복 중지!"); //while문 반복 중지!
```

1번 코드와 2번 코드 모두 i < 5, i > 0이라는 기존의 조건이 존재하지만, if 구문에 작성된 조건에 의해 break 구문을 만나는 순간 앞선 조건은 무시되고, 해당 반복문을 빠져나오게 됩니다. 빠져나온 뒤에 코드 구문이 존재할 경우 마찬가지로, 한 줄 한 줄씩 코드가 실행되게 됩니다.

다음은 continue 구문입니다. continue 구문은 break 구문과 마찬가지로 반복문 내에서 코드를 제어하기 위해서 사용되는 코드 구문입니다. 반복문 내에서 continue 구문을 만날 경우에 반복 실행 코드를 실행하지 않고 바로 다음 조건을 검사하게 됩니다. 또한, break 구문과 마찬가지로 조건문과 함께 사용됩니다.

아래 코드 예시는 continue 구문을 이용한 for 구문 값을 제어하는 코드입니다.

```
for(var i = 0; i < 5; ++i){
    // i의 값을 2로 나눈 값의 나머지가 0(홀수)이면 아래 코드를 실행하지 않고 조건 검사
    if(i % 2 == 0){
        continue;
    }
    console.log(i); //1, 3
}
//1, 3 (홀수만) 출력 후 아래 코드를 실행
console.log("for문 반복 중지!"); //for문 반복 중지!
```

어떤 값(정수)을 2로 나눴을 경우 나머지가 0인 경우 해당 값은 짝수라고 할 수 있습니다. 반복문에서 짝수를 제외한 값만 출력하고 싶은 경우 위와 같이 조건문과 `continue` 구문을 통해 코드의 출력을 제어할 수 있습니다.

무한반복(Infinite Loop)

우리는 지난 장에서 조건문을 살펴보았고, 이번 장에서는 반복문에 대해서 살펴보았습니다. 조건문과 반복문을 하나로 코드의 제어를 위해 사용되는 제어문이라고 표현할 수 있습니다. 어떤 실행 프로그램을 만들기 위해 프로그램 코드를 작성하는 것은 결국 사람입니다. 그렇기 때문에 여러 가지 실수를 범하게 됩니다. 특히 처음 제어문과 관련된 내용을 다룰 때 많은 실수로 인한 오류를 접하게 되는데요.

반복문과 조건문을 작성할 때 우리는 어느 정도 코드의 결괏값을 예측할 수 있어야 합니다. 만약, 그 예측이 잘 이루어지지 않았을 경우에 발생할 수 있는 오류 중 대표적인 것이 이번 절에서 살펴볼 무한반복 코드라고 할 수 있습니다.

사실, 의도적으로 무한반복 코드가 사용되는 부분도 있습니다. 쉬운 예로, 현관문에 달려있는 도어록과 같은 임베디드 시스템을 예로 들 수 있습니다. 일반적으로 이러한 임베디드 시스템은 전원이 공급되는 동안 똑같은 작업을 반복적으로 실행하게 됩니다. 제가 이번 절에서 말씀드릴 무한반복 코드는 이처럼 의도된 반복 동작이 아닌, 의도되지 않은 반복 작업을 일컫습니다.

우선, 아래 코드는 무한 반복 코드가 발생하는 코드 예시입니다.

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

위 코드는 반복을 나타내는 for 문의 `i >= 0` 조건 구문은 "0보다 크거나 같을 경우" 실행 코드를 반복하게 되는 조건입니다. 해당 코드만 놓고 보면, 반복문을 종료하고 다른 작업을 실행할 어떤 조건도 존재하지 않습니다. 위 코드를 VS Code를 통해 크롬에서 실행할 경우 탭 부분에 빙글빙글 도는 표시와 함께 코드가 무한 실행되는 것을 확인할 수 있습니다. (굳이 확인하지 마세요!)

다시 한번 말씀드리지만, 의도적으로 반복문을 무한 실행시키는 게 아닌 이상 적절한 반복 종료 조건이 존재하지 않을 경우 예상하지 못한 결과를 마주할 수 있다는 점 꼭 기억해주세요!

우리는 이번 장을 끝으로 조건문과 반복문 등의 자바스크립트의 논리적인 흐름을 제어하는 구문인 제어문에 대해서 살펴보았습니다. 어떤 면에선 굉장히 기초적인 부분이지만, 조건문과 반복문의 사용이 코드의 70% 이상을 차지한다고 해도 과언이 아닙니다. 그렇기 때문에, 제어문의 사용에 대한 이해는 백 번 천 번 강조해도 지나치지 않습니다.

우리가 코드 작성 시 알고리즘이라는 것을 자주 언급하게 됩니다. 이 알고리즘이라는 것도 결국 "제어문을 어떻게 사용할 것인가?"를 나타내는 척도라고 할 수 있습니다. 제어문 사용에 대해 꾸준한 연습을 진행해보세요!:)