반복문

Javascript 반복문

- 특정 코드를 반복하고 싶을 때 사용합니다!
- 100번을 같은 동작을 반복하는데, 100줄을 쓰면 너무나 비효율적이니까요!

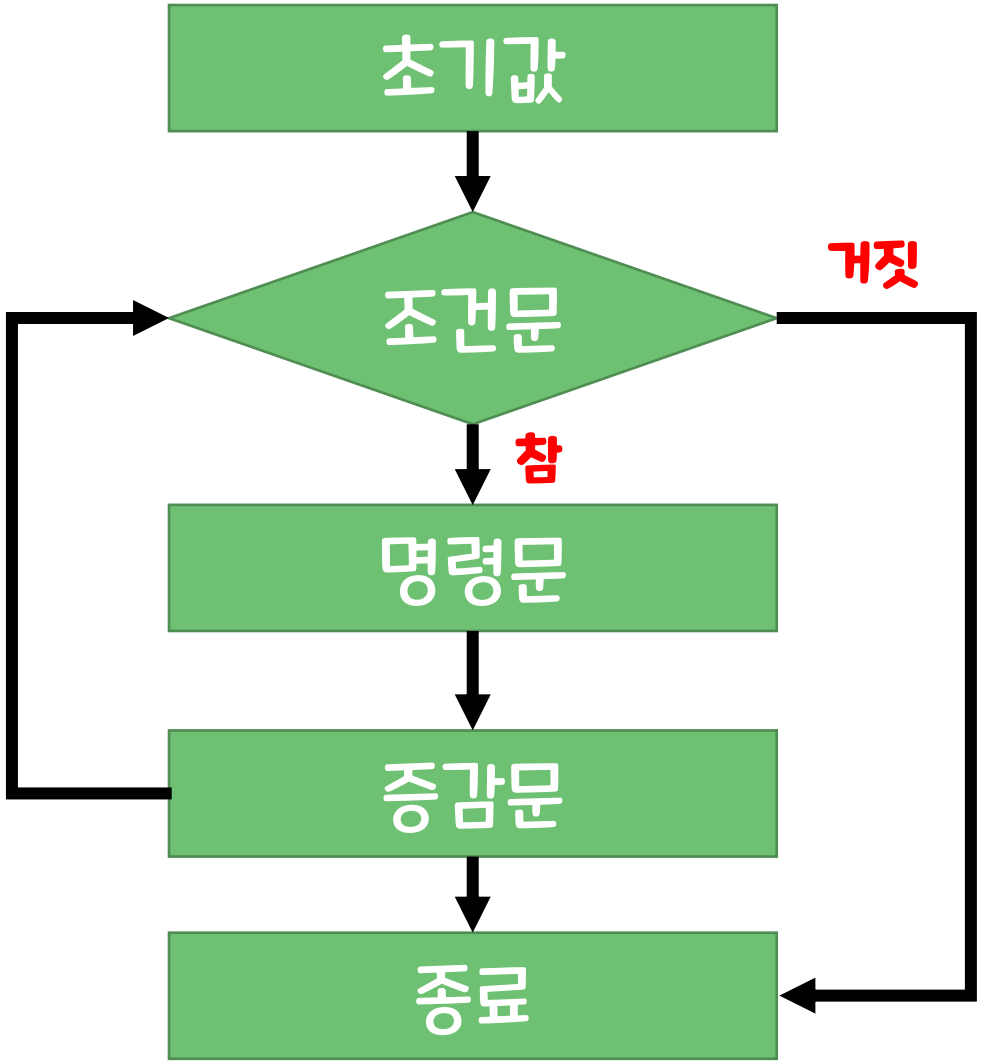
for

while

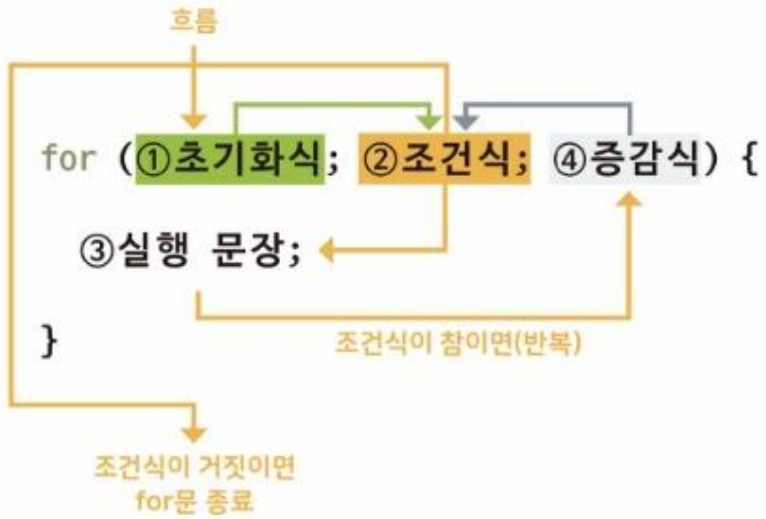
for / of

do / while


흐름
for (①초기화식; ②조건식; ④증감식) {
③실행 문장;
}
조건식이 참이면(반복)
조건식이 거짓이면
for문 종료


초기값
조건문
거짓
참
명령문
증감문
종료

for 문




```
// for 문
for(let index = 0; index < 10; index++) {
  console.log("인사를 ", index+1, "번째 드립니다!
😆");
}
```

```
인사를 1 번째 드립니다! 😆
인사를 2 번째 드립니다! 😆
인사를 3 번째 드립니다! 😆
인사를 4 번째 드립니다! 😆
인사를 5 번째 드립니다! 😆
인사를 6 번째 드립니다! 😆
인사를 7 번째 드립니다! 😆
인사를 8 번째 드립니다! 😆
인사를 9 번째 드립니다! 😆
인사를 10 번째 드립니다! 😆
```

중첩 for 문

- 중첩 if 문이 가능하듯 for문도 중첩이 가능하다!

```
for (let i = 0; i < 5; i++) {
  for (let k = 0; k < 3; k++) {
    // 반복할 문장
  }
}
```

while 문

```
while(조건){
   // 조건이 참일 때 실행할 코드
}
```

- for 문과는 달리 값을 제어하는 구문이 기본적으로 포함이 되어 있지 않기 때문에 무한 루프 가능
  - 조건이 항상 참이라면? while문을 빠져나가지 못하고 끝없이 반복하겠지요?
- 따라서 주의하여 사용 필요합니다!

```
// while 문

// 1번 타입, 조건문을 사용
let index = 0;

while (index < 10) {
  console.log("인사를 ", index + 1, "번째 드립니다! 😆");
  index++;
}

// 2번 타입, 조건문을 사용하지 않고 if 문 + break 사용
let index2 = 0;

while (true) {
  console.log("절을 ", index2 + 1, "번째 드립니다! 😆");
  index2++;
  if (index2 == 10) {
    break;
  }
}
```

```
인사를 1 번째 드립니다! 😆
인사를 2 번째 드립니다! 😆
인사를 3 번째 드립니다! 😆
인사를 4 번째 드립니다! 😆
인사를 5 번째 드립니다! 😆
인사를 6 번째 드립니다! 😆
인사를 7 번째 드립니다! 😆
인사를 8 번째 드립니다! 😆
인사를 9 번째 드립니다! 😆
인사를 10 번째 드립니다! 😆
절을 1 번째 드립니다! 😆
절을 2 번째 드립니다! 😆
절을 3 번째 드립니다! 😆
절을 4 번째 드립니다! 😆
절을 5 번째 드립니다! 😆
절을 6 번째 드립니다! 😆
절을 7 번째 드립니다! 😆
절을 8 번째 드립니다! 😆
절을 9 번째 드립니다! 😆
절을 10 번째 드립니다! 😆
```

```
// 구구단 while 버전
let i = 2, j = 1;

while(i < 10) {
    while(j<10) {
        console.log(i, "x", j, "=", i*j);
        j++;
    }
    i++;
    j = 1;
}
```

do-while 문

```
do{
   // 조건이 참일 때 실행할 코드
} while(조건)
```

- while문과는 다르게 조건에 관계 없이 처음 한 번은 무조건 실행!

break & continue

break

- 반복문을 멈추고 빠져나감

```
// break

for(let i = 0; i < 100; i++) {
    if(i==10) {
        console.log("멈춰!");
        break;
    }
    console.log(i);
}
```

```
0
1
2
3
4
5
6
7
8
9
멈춰!
```

continue

- 반복문을 이번 반복 회차를 스킵하고 다음 반복 회차로 진행

```
// continue
let sum = 0;

for(let i = 0; i < 100; i++) {
    if(i%2 == 0) {
        continue;
    }
    sum += i;
}

console.log(sum);
```

```
2500
```

배열에서의 For

배열, 기본 for 문 사용하기

```
let numbers = [1, 2, 3, 4, 5, 6];
let fruits = ["사과", "바나나", "수박", "포도", "파인애플"];

for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
}

for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
```

배열, for of 반복문

```
let numbers = [1, 2, 3, 4, 5, 6];
let fruits = ["사과", "바나나", "수박", "포도", "파인애플"];

let numbersLength = numbers.length;
let fruitsLength = fruits.length;

for (let number of numbers) {
  console.log(number);
}

for (let fruit of fruits) {
  console.log(fruit);
}
```

배열, [].forEach => for 문법은 아니고, 메소드!

```
let numbers = [1, 2, 3, 4, 5, 6];
let fruits = ["사과", "바나나", "수박", "포도", "파인애플"];

numbers.forEach(function (number, index, array) {
  console.log(number, index, array);
});

numbers.forEach((number, index, array) => {
  console.log(number, index, array);
});

fruits.forEach(function (fruit, i, arr) {
  console.log(fruit, i, arr);
});

fruits.forEach((fruit, i, arr) => {
  console.log(fruit, i, arr);
});
```

배열의 합

```
let numbers = [1, 2, 3, 4, 5, 6];
var sum1 = 0;
var sum2 = 0;
var sum3 = 0;

for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
  sum1 = sum1 + numbers[i];
}

for (let num of numbers) {
  sum2 = sum2 + num;
}

numbers.forEach((num) => {
  sum3 = sum3 + num;
});

console.log(sum1, sum2, sum3);
```

배열에서의 기타 메소드

```
const arr = [1, 2, 3, 4, 5];
```

- arr.filter()

```
let arr_filter = arr.filter(function (a) {
    return a > 3;
});
```

```
arr_filter = [4, 5]
```

  - 배열 내부에서 조건에 부합하는 요소만 찾아서 "배열로" 반환

- arr.find()

```
let val_find = arr.find(function (a) {
    return a > 3;
});
```

```
val_find = 4
```

  - 배열 내부에서 조건에 부합하는 첫번째 요소를 찾아서 "값"으로 반환

- arr.map()

```
let arr_map = arr.map(function (b) {
    return b + 3;
});
```

```
arr_map = [4, 5, 6, 7, 8]
```

  - 익명함수에 쓰여진 연산결과를 새로운 배열로 반환

- 위의 메소드들은 매개변수로 익명함수가 들어간다는 공통점이 있음
  - 앞서 나온 forEach((val)=>{})메소드도 마찬가지!

배열 관련 내장 메소드

배열 관련 method

- arr.push(): 배열 끝에 추가
- arr.pop(): 배열 끝 요소 제거
- arr.shift(), arr.unshift(): 배열 앞에 제거/추가
- arr.include(요소): 배열에 해당 요소가 있는지 확인

배열 관련 method

- arr.length : 배열의 길이 반환
- arr.indexOf() : 문자열에서의 indexOf와 마찬가지로 매개변수에 해당하는 배열의 인덱스를 받아옴. 단, 매개변수로 문자열만 넣을 수 있는 것은 아님!
- arr.reverse() : 배열 순서 뒤집어서 반환
- arr.join(): join 안의 문자열 기준으로 문자열로 병합

문자열 관련 내장 메소드

let str = 'Happy day~! ';

- **length** : 문자열의 길이를 반환(공백포함)

  `str.length` 13

- **toUpperCase()&toLowerCase()** : 문자열 전체를 대문자, 혹은 소문자로 변경

  `str.toUpperCase()` HAPPY DAY~! `str.toLowerCase()`

- **indexOf('')** : 매개변수로 문자열을 받아서 몇번째 인덱스인지 숫자 반환

  `str.indexOf('p')` 2

- **slice(startIdx[,endIdx])** : start 부터 end-1 까지 슬라이싱, 빼옴(?) 매개변수로 음수값도 가능

  `str.slice(5, 9)` day

  `str.slice(2)` ppy day~!

- **replace(문자열1, 문자열2)** :문자열1을 문자열2로 변경

```
str.replace('p', 'a')
```
'Haapy day~! '

- **replaceAll(문자열1, 문자열2)** : 문자열1을 전부찾아서 문자열 2로 바꿔줌

```
str.replaceAll('p', 'a')
```
'Haaay day~! '

- **repeat(n)** : 문자열에 대해 n번 반복

```
str.repeat(3)
```
Happy day~! Happy day~! Happy day~!

- **trim()** : 문자열의 양끝 공백 없애기

```
str.trim()
```
//'Happy day~!'

- **split()** : 매개변수로 들어온 문자열을 기준으로 str을 쪼개서 배열로 저장

```
str.split('')
```
```
▶ (13) ['H', 'a', 'p', 'p', 'y', ' ', 'd', 'a', 'y', '~', '!', ' ', ' ']
```

```
str.split(' ')
```
```
▶ (4) ['Happy', 'day~!', '', '']
```

메소드 체이닝

메소드 체이닝 (method chaining)

- 여러 메소드를 연결해서 사용하는 개념!
- 단, 사용한 메소드가 반환(return) 값을 가지고 있는 경우에만 사용이 가능!

- 'hello'.split("") → ['H', 'e', 'l', 'l', 'o'] 라는 배열이 반환 됨

- 배열에는 reverse() 라는 메소드가 존재
- 'hello'.split("").reverse() 는 ['H', 'e', 'l', 'l', 'o'].reverse() 와 동일!
- ['H', 'e', 'l', 'l', 'o'].reverse() → ['o','l','l','e','H'] 와 동일

- 'hello'.split("").reverse().join("") → ['o','l','l','e','H'].join("") 과 동일