배열과 while

2022년 10월 4일 화요일 오후 8:23

[while 반복문]

While (불_표현식) {
// 불 표현식이 참인 동안 실행할 문장
}

[반복문과 배열]

```
> for(let i = 0; i < 5; i++) {
      console.log("출력");
  }
5 출력
```

[배열 생성 방법]

Let 이름 = [자료, 자료, 자료, 자료, 자료]

```
> let array = [52, 22, '아침', 'wjatla' , '저ㅏ녁']
< undefined
> array
< ▶ (5) [52, 22, '아침', 'wjatla', '저ㅏ녁']
```

[배열의 요소와 인덱스]

배열 [인덱스]

```
> let array = [52, 22, '아침', 'wjatla' , '저ㅏ녁']
< undefined
> array
< ▶ (5) [52, 22, '아침', 'wjatla', '저ㅏ녁']
> let array = [52, 22, '아침', 'wjatla' , '저ㅏ녁']
< undefined
> array[0] = 0;
< 0
> console.log(array[0]);
  0
< undefined
> console.log(array[1]);
  22
< undefined
> console.log(array[2]);
  아침
< undefined
> console.log(array[3]);
  wjatla
< undefined
```

```
> console.log(array.length);
  5
< undefined
```

For 반복문

2022년 10월 5일 수요일 오후 7:20

[for 반복문]

For (let i = 0; I < 반복횟수 ; i++) {

}

[역 for 반복문]

배열 반복을 뒤에서 부터 실행 해야 할 때

For (let i = length -1; i>= 0; i--) {

}

```
> let array = [ 1, 2, 3, 4, 5, 6];
< undefined
> for (let i = array.length -1; i >=0; i--){
    console.log(array[i]);
  }
  6
  5
  4
  3
  2
  1
```

[for in 반복문과 for of 반복문]

1. for in
   For (let 인덱스 in 배열){

   }
2. for of
   For (let 요소 of 배열){

   }
3. for 반복문 사용과 같음
   For (let i = 0 ; I < 배열.길이; i++) {
   Let 인덱스 = I;
   Let 요소 = 배열[i];
   }

[중첩 반복문]

```
> let output = '';
< undefined
> for (let i = 0; i<10; i++){
    for (let j = 0; j<i+1; j++){
      output += '*';
    }
    output +='\n';
  }
< '*\n**\n***\n****\n*****\n******\n*******\n********\n*********\n**********\n'
> console.log(output);
  *
  **
  ***
  ****
  *****
  ******
  *******
  ********
  *********
  **********
```

키워드

2022년 10월 5일 수요일 오후 7:48

[break 키워드]
내부 break 키워드로 가까운 반복문 종료

[continue 키워드]
반복문 내부에서 현재 반복 멈추고 다음단계 진행

```
PS C:₩Users₩User> node
Welcome to Node.js v16.17.0.
Type ".help" for more information.
> for(let i = 1; i<10; i++){
... if (i %2 ==0){
... continue;
... }
... console.log(i)
... }
1
3
5
7
9
undefined
```

```
undefined
> for(let i = 1; i<10; i++){
... if(i % 2 !=0) {
... console
... .log
... (i)
... ;
... }
... }
1
3
5
7
9
undefined
```

[스코프]
:: 변수에 접근 할 수 있는 범위
식별자(변수)를 찾기 위한 규칙

스코프 == 블록
{
블록
}

```
var x = 'global';

function foo () {
  var x = 'function scope';
  console.log(x);
}

foo(); // ?
console.log(x); // ?
```

외부에서 x 참조 못함 foo 내부에서만 참조 가능

1) 블록 레벨 스코프
: 코드 블록 내에서만 참조 . 접근 가능 범위

```
int main(void) {
  // block-level scope
  if (1) {
    int x = 5;
    printf("x = %d\n", x);
  }

  printf("x = %d\n", x); // use of undeclared identifier 'x'

  return 0;
}
```

위 C 언어 코드를 보면 if 문 내에서 선언된 변수 x 는 if 문 코드 블록 내에서만 유효하다.
즉, if 문 코드 블록 밖에서는 참조가 불가능하다.

2) 함수 레벨 스코프
: 함수 코드 블록 내에서만 참조, 접근 가능한 범위

```
var x = 0;
{
  var x = 1;
  console.log(x); // 1
}
console.log(x);   // 1

let y = 0;
{
  let y = 1;
  console.log(y); // 1
}
console.log(y);   // 0
```

Let 을 사용하면 블록 레벨 스코프를 사용 할 수 있다.

3) 렉시컬 스코프
: Scope의 또다른 특징으로, 상위 스코프를 결정하는 방법을 들 수 있다.
상위 스코프를 결정하는 방법엔 두가지가 있다.

- 동적 스코프

    - Dynamic Scope
    - 함수를 어디서 호출 하였는지에 따라 상위 스코프를 결정
- 렉시컬 스코프
    - Lexical scope/Static scope
    - 함수를 어디서 선언 하였는지에 따라 상위 스코프를 결정
    - JavaScript 및 대부분의 프로그래밍 언어에서 사용하는 방법

자바스크립트는 렉시컬 스코프를 따르므로 함수를 선언한 시점에 상위 스코프가 결정된다.
함수를 어디에서 호출하였는지는 스코프 결정에 아무런 의미를 주지 않는다.