JS-중급 16

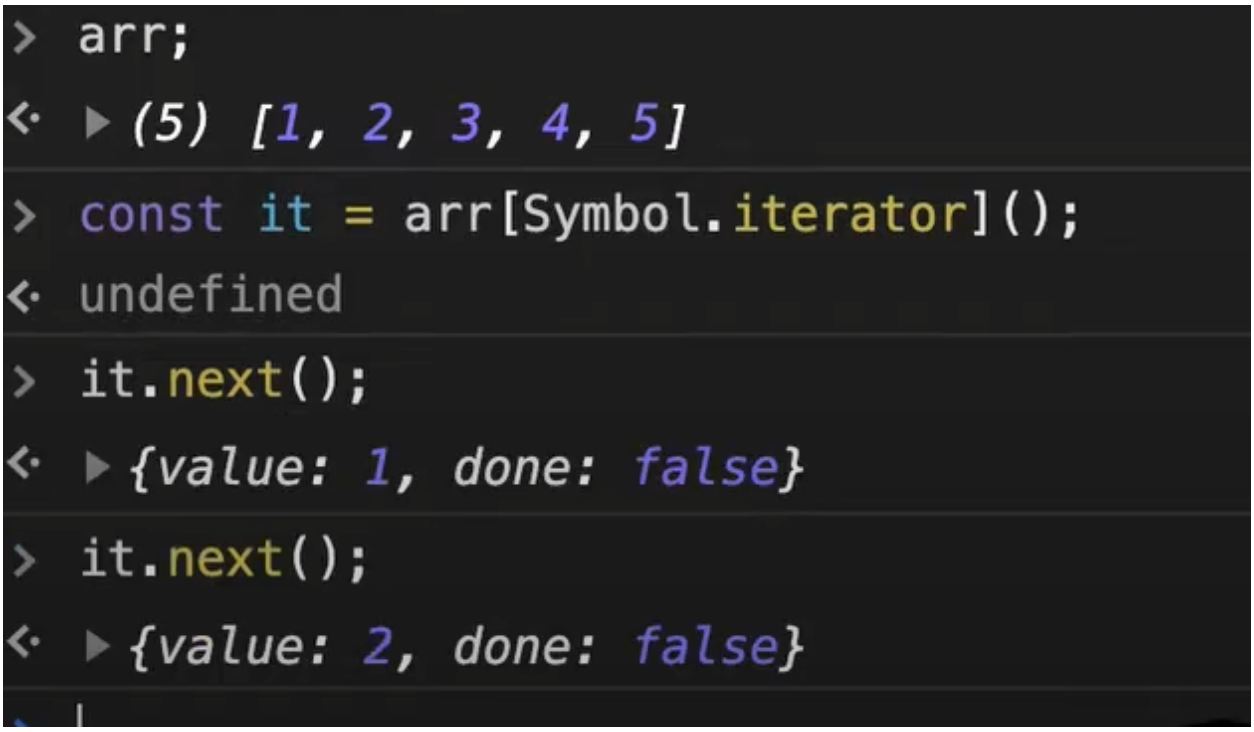
태그	
날짜	@2023년 6월 7일

Generator

```
Generator : 함수의 실행을 중간에 멈췄다가 재개할 수 있는 기능
next(), return(), throw()

function* fn() {
  try{
    console.log(1);
    yield 1;
    console.log(2);
    yield 2;

    console.log(3);
    console.log(4);
    yield 3;
    return "finish";
  } catch(e){
    console.log(e);
  }
}

const a = fn();

a.next();
1
// [object Object]
{
  "value": 1,
  "done": false
}
// [object Object]
{
  "value": 1,
  "done": false
}
a.next();
2
// [object Object]
```

```
{
  "value": 2,
  "done": false
}
a.next();
3
4
// [object Object]
{
  "value": 3,
  "done": false
}
// [object Object]
{
  "value": 3,
  "done": false
}
a.next();
// [object Object]
{
  "value": "finish",
  "done": true
}

a.return('END'); 를 호출하면
그 즉시 done 값이 true가 된다.
```

```
Generator :
iterable - 반복이 가능하다.
- Symbol.iterator 메서드가 있다.
- Symbol.iterator는 iterator를 반환해야 한다.

iterator
- next 메서드를 가진다.
- next 메서드는 value 와 done 속성을 가진 객체를 반환한다.
- 작업이 끝나면 done은 true가 된다.
```




```
function* fn() {
  yield 4;
  yield 5;
  yield 6;
}

const a = fn();

a[Symbol.iterator]() === a;
true

자기 자신이 나온다. 즉 generator는 iterator 객체인것이다.

for(let num of a) { console.log(num);}
4
5
6
undefined

for of 가 실행되면 Symbol.iterator를 호출하고 없으면
에러가 발생한다.
done이 true가 될때 까지 반환한다.
```

```
Generator : next()에 인수 전달

function* fn() {
  const num1 = yield " 첫번째 숫자를 입력";
  console.log(num1);

  const num2 = yield " 두번째 숫자를 입력";
  console.log(num2);

  return num1 + num2;
}

const a = fn();

a.next();
// [object Object]
{
  "value": " 첫번째 숫자를 입력",
  "done": false
}
a.next(2);
2
// [object Object]
{
  "value": " 두번째 숫자를 입력",
  "done": false
}
a.next(4);
4
// [object Object]
{
  "value": 6,
  "done": true
}

next()는 외부로 부터 값을 입력 받을 수 있다.
```

```
Generator : 값을 미리 만들어 두지 않음
메모리 측면에서 효율적이다.
while문으로 무한 반복을 만들어도 브라우저가 뻗지 않는다.
실행 시에만 값을 만들기 때문

function* fn() {
```

```
  let index = 0;
  while (true) {
    yield index++;
  }
}
const a = fn();

a.next();
// [object Object]
{
  "value": 0,
  "done": false
}
a.next();
// [object Object]
{
  "value": 1,
  "done": false
}
```

```
Generator : yield* 이용

function* gen1() {
    yield "w";
    yield "o";
    yield "r";
    yield "l";
    yield "d";
}

function* gen2(){
  yield "hello";
  yield* gen1();
  yield "!";
}

gen2 에서 gen1 호출

console.log(...gen2());
"hello" "w" "o" "r" "l" "d" "!"
undefined

...gen2 처럼 구조분해할당을 사용했는데
for of와 마찬가지로 done이 true가 될때까지 값을 펼쳐주는
역할을 한다.
```

Generator

제너레이터는 다른 작업을 하다가 다시 돌아와서
next( ) 해주면 진행이 멈췄던 부분 부터 이어서 실행

ex) Redux Saga