# 7_18_2022__js_ecma6

Step01_variable.html

~Step03_arrey.html

ecma6 에서 새로 생긴 기능들

- 변수 선언 방식

```
//예전에 변수를 선언하는 방식
        var num=10;

        //ecma6부터 변수를 선언하는 방식
        let num2=10;

        if(true){
            //myName 은 전역변수
            var myName="두부";
            // yourName은 if 문 블럭 안에서만 사용할 수 있는 지역 변수
            let yourName="유키"
        }

        //값을 변경하지 못하도록 상수(constant)화 시키기
        const pi=3.141592;
```

- 애로우 함수

```
//익명의 함수를 만들어서 f1이라는 변수에 대입하기
    let f1 = function(){};

//화살표 함수를 만들어서 f2 라는 변수에 대입하기.
    let f2 = ()=>{};     //arrow function


//두 수의 합을 콘솔에 추력하는 함수
    let showSum1 = function(num1, num2){
        let result1 = num1+num2;
        console.log("두수의 합"+result1);
    };
//두 수의 합을 콘솔에 출력하는 함수를 arrow function  으로 만들기
    let showSum2 =(num1,num2)=>{
```

```
        let result2 = num1+num2;
        console.log("두 수의 합"+result2)
    };

//두 수의 합을 리턴하는 함수
    let getSum1 = function(num1,num2){
        return num1+num2;
    };
//두 수의 합을 리턴하는 함수를 arrow function으로 만들기
    let getSum2 = (num1,num2)=>{
        return num1+num2;
    };

** 화살표 함수 안에서 this는 이벤트가 일어난 바로 그 요소를 가리키지 못한다.
```

- 람다함수(한줄짜리 함수)

```
let getSum3 = (num1,num2) => num1+num2;

중괄호도 return도 필요 없음
```

- 우리가 사용하는 변수들은 변경할 일이 거의 없다

```
 const

  값을 변경하지 못하도록 const를 사용할 수 있다.
  참조하는 변수들 등 이후 값을 수정하지 않는 것들

**
const a="kim";
const mem={name:"kim"}
mem.name ="park"


상수 a 안에 들어있는 "kim"을 다른 값으로 바꿀 수 없다.
a="park";  불가

상수 mem 안에 들어 있느 값을 다른 값으로 변경 불가

mem = {};          불가
mem = 10;          불가
mem = "park";      불가

mem.name ="park"   가능
  사물함 키 번호는 const이기 때문에 못바꾸지만
  안에 있는 내용물은 const가 아니어서 바꿀 수 있다.
  arrey 인 경우 착각하지 않도록 주의
```

- array

```
const names = ["두부","유키","소주"];

  let [a,b,c] = names;로 분해 할당이 가능
  let arr=[...names];

  let arr2=[..names,"xxx","yyy"] 로 추가 할당도 가능




******
// 콘솔에서 a,b,c 변수를 확인
        const names=["두부","유키","소주"];

            //콘솔에서 arr 확인
        let arr=[...names, "재즈", "여비"];



        const result1 = names.map((item)=>{
            console.log(item);
            return item+" 님!";
        });
        /*
            1. 배열은 map()함수를 원래 가지고 있다.
            2. map() 함수를 호출하면서 함수를 전달해야 한다.
            3. 전달한 함수는 배열의 size 만큼 즉시 반복 호출된다.
            4. 반복호출 하면서 그 함수에 배열에 저장된 item을 순서대로 전달 해준다.
            5. 함수안에서 리턴해주는 값을 순서대로 모아서 새로운 배열을 map() 함수로 리턴해준다.
        */
        const result2=names.map((item)=>item+ "님?")
```

map함수

값을 여러개 가지고 있는 배열에서 모든 값을 꺼내고자 할때 사용할 수 있다.

이런 배열이 있다고 생각해보자

```
let arr = [ 10, 20, 30, 40, 50 ];
```

arr 배열의 5개 값을 전부 꺼내 콘솔에 찍어보고 싶다면 for 문을 이용할 수도 있다.

```
const arr = [ 10, 20, 30, 40, 50 ];
```

```
//기본 for문 방식
console.log("기본 for문");
for(let index = 0 ; index < arr.length ; index++) {
    console.log(arr[index]);
}

//향상된 for문 방식
console.log("\n향상된 for문");
for(let item of arr) {
    console.log(item);
}
```

arr 배열의 값을 item에 하나씩 담아오는 개념이다.

map 함수는 값고 인덱스를 인자로 받아 자동으로 for 문을 돌려 값을 빼도록 해준다.

map 함수의 값 인자는 향상된 for 문의 item 인자와 같은 역할, map 함수의 인덱스 인자는 기본 for문 방식의 index와 같은 역할이라 보면된다.

아래는 위 예제 코드를 map()을 적용시켜 바꾼 예제 코드이다.

```
const arr = [ 10, 20, 30, 40, 50 ];

//일반 함수 형태
arr.map(function(item, index) {
    console.log(index+"번 값", item);
});

//화살표 함수 형태
arr.map((item, index) => {
    console.log(index+"번 값", item);
});
```

index 인자는 배열안의 인덱스(몇번째)를 의미하고 item에는 배열안의 값들이 하나씩 순서대로 담긴다.

능동적으로 여러 상황에 사용하기 위해 map()을 사용한다.