7_19_2022__js_ecma6

- filter()함수

item을 검사해서 제거할 것들은 제거해서 새로운 배열을 얻어내는 함수.

```
  const nums=[1,2,3,4,5,6,7,8,9,10];
/*
배열의 filter() 함수는 조건에 맞는 item으로 새롭게 구성된
배열을 리턴한다.
true가 리턴된 index의 item으로만 구성된 배열
*/
  let result=nums.filter(function(item){
//item을 2로 나눈 나머지가 0일때 true (짝수여부 확인)
  let isEven = item%2 == 0;
//리턴해 준다
  return isEven;
  })
//람다함수
  let result2=nums.filter((item)=>item%2==0);

//결과
  (5) [2, 4, 6, 8, 10]
```

find() 함수

가장 첫번째로 true를 리턴한 곳의 item을 return한다.

```
let result3=nums.find((item)=>item%3==0);
```

objects

```
<body>
    <input type="text" id="inputMsg" placeholder="문자열 입력...">

    <script>
        let mem={num:1, name:"두부",addr:"페어펙스"};
        /*
            let num=mem.num;
            let name=mem.name;
            위의 두 줄을 줄여서 아래와 같이 쓸 수 있다.
        */

        //객체를 분해해서 변수에 할당하기(객체 분해 할당)
        let {num,name}=mem;
```

```
        let useMem=function({num,name}){
            console.log(num+"|"+name);
        };

        useMem({num:2, name:"유키"});
        useMem(mem);

        //e라는 변수에 object를 담음 {keycode:x,...}
        document.querySelector("#inputMsg").addEventListener("keydown", function(e){
            console.log(e.keycode);
        });
        //어짜피 object 가있으니 {오브젝트}식으로 작성해도 된다.
        document.querySelector("#inputMsg").addEventListener("keydown", function({keycode}){
            console.log(keycode);
        });
    </script>
</body>
```

변수명을 값과 똑같이 입력하고 사용할 수 있다.

```
let num=1;
let name='두부';
let isMan=true;

let mem1={num:num, name:name, isMan:isMan, sing:function(){}};
        {명:값}
//위의 object와 아래의 object는 모양이 같다.
let mem2={num, name, isMan, sing(){}};
          {값}

//object 의 key 값으로 사용할 문자열을 변수에 대입하고
let a="num";
let b="name";
let c="isMans";
//object의 key값을 변수 안에 있는 값으로 결정할 수 있다
let mem3={[a]:2, [b]:"유키", [c]:false}
```

function

js에서 아무 값도 넣지 않으면 빈 것이 아니라 undefined가 들어있다.

```
function showInfo(num, name, addr){
          console.log(num+"|"+name+"|"+addr)
        }
         showInfo()
//결과
undefined|undefined|undefined
```

기본값(default)으로 지정 할 수 있다

```
function showInfo(num=0, name="누구게", addr="어디게"){
            console.log(num+"|"+name+"|"+addr)
        }

        showInfo()
//결과
0|누구게|어디게
```

배열 args

```
function useFunc(...args){
            console.log(args);

        }
        useFunc();//빈 배열
        useFunc(10);
        useFunc(10,20,30);
        useFunc(10,"Min",true);
//결과
[]
[10]
(3) [10, 20, 30]
(3) [10, 'Min', true]
```

template string

```
//backtick 기호를 이용해서 문자열을 만들 수 있다.
let info=`어쩌구... 저쩌구...`;
// backtick 은 여러줄의 문자열을 편하게 작성할 수 있다.
let html1=`
    <ul>
        <li>하나</li>
        <li>두울</li>
    </ul>
    `;
//bakctick은 변수안에 들어있는 수자나 문자열을 연결할 때도 편하다.
let name1="유키";
let name2="두부";
let html2=`
    <ul>
        <li>${name1}</li>
        <li>${name2}</li>
    </ul>
    `;//${js공간}
```

for 문

```
let names=["두부","유키","소주"];

//배열에 저장된 모든 item을 순서대로 참조해서 작업하는 기존의 방법
console.log("-- 기본 for 문 활용 --");
```

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

console.log("-- for in 활용 --");
for(let i in names){ //i 에 index가 자동으로 할당된다.
    let item=names[i];
    console.log(item);
}

console.log("-- 배열의 .forEach() 함수 사용 --");
names.forEach((item)=>{
    console.log(item);
});

console.log("-- ecma6 에서 추가된 for of 활용 --");
for(let item of names){
    console.log(item);
}
```

setTimeout

```
/*
  setTimeout() builtin 함수가 있다.

  setTimeout(콜백함수, 지연시간(ms))

  어떤 작업을 일정시간 지연 이후에 하고 싶을 때 사용한다.
*/

setTimeout(function(){
    console.log(" 5 초가 지났네?")
},5000);

//인자로 전달된 함수를 10초 이후에 호출해주는 함수
function wait(callback){
    setTimeout(function(){
        callback();
    },10000);
}
wait(function(){
    console.log("10초가 지났네?")
});
```

callback hell

```
//메뉴를 주문하는 함수라고 가정
function order(menu, callback){
    //메뉴를 준비하는데 걸리는 시간이 랜덤하다고 가정하자
    const delay=Math.random()*5000+5000; // 5000~10000 사이의 랜덤한 숫자
    setTimeout(function(){
        console.log(menu+" 가 준비되었습니다.");
        callback(menu);
    }, delay);
```

```
}
//js 특징 - 시간 순서를 정해주려면 함수 안으로 다른 함수가 들어가야 함.
order("커피",function(item){
    order("샌드위치",function(item){
        order("짜장면",function(item){
            order("녹두전",function(item){});
        });
    });
});// 순서대로 사용하고 싶을 때 callback이 쌓인다.
```

promise

```
//메뉴를 주문하는 함수라고 가정
function order(menu, callback) {
    //메뉴를 준비하는데 걸리는 시간이 랜덤하다고 가정하자
    const delay = Math.random() * 5000 + 5000; // 5000~10000 사이의 랜덤한 숫자
    setTimeout(function () {
        console.log("주문하신 " + menu + "가 준비되었습니다.");
        callback(menu);
    }, delay);
}
// promise 객체를 생성하면서 함수 전달
new Promise(function(resolve){
    //전달된 함수는 즉시 호출됨.
    //함수의 인자로 resolve function 이 전달되는데 해당 함수를 호출하면
    order("커피",resolve);
}).then(function(item){
    console.log(item+"맛나게 마셔요!")
    //샌드위치를 주문하는  promise 객체를 만들어서 리턴해준다.
    return new Promise(function(resolve){
        order("샌드위치",resolve);
    });
}).then(function(item){
    console.log(item+"을 맛나게 먹어요. 냠냠");
    return new Promise(function(resolve){
        order("아이스크림",resolve);
    });
}).then(function(item){
    console.log(item+"을 녹여먹어요 핥짝");
})
```