박재성 강사님

[01-16]

- ! + tap : 자동 구문
    - 자바스크립트의 변수는 레퍼런스를 가져오는 것

```
//변수, 자바와 달리 타입을 미지정
    var name = "홍길동";
    let name2 = "이순신"; //2015년이후
    //상수
    const PI = 3.14; //2015년이후, ECMAScript 이후
    name = "홍길돈"; //변경가능
    PI=3.2; //변경불가
```

- 장단점으로 자바와 달리 타입을 미지정하므로 메모리 관리에 불리함이 있음

```
//배열, 자바에 비해 크기 미정 가능
let arr = []; //빈배열
arr[0]=10;
arr[1]=11;
arr[2]="www";
console.log(arr[0]);

//변수, 타입 미지정
let age;
console.log(age); //출력시 undefined, type가 지정되지 않아서
```

- for문의 순서

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

    - let i=0 : 가장 처음 한번만 실행
    - ; i<arr.length; : 두번째로 실행
    - console.log(arr[i]); : 세번째로 실행
    - i++ : 네번째로 실행
        - 이후 2,3,4 가 반복되는 구조
        - ( ) 가 거짓이 될 때, 해당 구문은 false 므로 탈출

- 객체(object) : 자바스크립트는 처음에 클래스가 없으므로, 바로 { } 를 사용

```
let person = {name:"이순신", age:10 }; //JSON에서는 속성명에 ""를 붙임
console.log(person.name); //"name" : "이순신"
console.log(person.age);
```

    - 속성 : 값 의 형태로 자료를 나열
    - 출력의 경우 person.name 의 형태로 진행
    - XML은 무겁고 파싱에 시간이 걸림

- 객체안에 배열넣기 : 객체의 속성의 값으로 배열을 넣을 수 있어 자유로운 편{ } [ ]

```
let student = {name:"홍길동", subject:["컴공","인문학"]}
console.log(student.subject[0]);  //객체에 배열을 넣는 것
console.log(student.subject[1]);
```

- 배열에 객체를 넣기 : 배열의 속성의 값으로 객체를 넣을 수 있어 자유로운 편

```
let list=[
   {name:"이순신", age:10 },
   {name:"홍길동", age:20 }
]; //가독성을 위해 내리는편
console.log(list[0].name);
console.log(list[1].age); //이렇게 접근이 됨
```

- Document Object Model(DOM) : 엘리먼트를 찾는 방법
    - html 이 먼저 읽히고 script가 뒤에 읽히는것에 주의

```
<div id="box1">box1</div>
<script> // id 로 elements 찾기
   document.getElementById("box1").innerText="박스1";
</script>
```

- Head 에 넣은 경우, onload 가 한번더 불러주는 것

```
<script>
   //로딩이 된후 실행됨
   window.onload=function(){
      document.getElementById("box1").innerText="박스1";
   };
</script>
```

- 단수(하나로) 표현되는 애들은 지정해서 찾을 수 있다.
    - id, querySelector
- 복수로 표현되는 애들은 배열로 찾아야 한다.
    - tag, class, querySelectorAll

- id와 달리 class는 중복되어 찾을 수 있다

```
<div id="box1">box1</div>
<div class="box">box</div>
<div class="box">box</div>
```

- class는 중복이 가능하므로 배열 형태로 호출해야 가능

```
//class로 elements 찾기, 얘는 배열이기에 이렇게 써야한다.
documents.getElementsByClassName("box")[0].innerText="박스"
documents.getElementsByClassName("box")[1].innerText="박스"
```

- querySelectorAll 는 전부 다 찾아줌 . 을 찍어야함
    - (div>div>li) 처럼 다 찾을 수 있음
    - innerText 는 그대로 출력하고
    - innerHTML 은 자바스크립트를 해석해서 표현해줌 // *js*

```
document.querySelectorAll(".box2")[0].innerHTML="<i>js</i>";
```

- querySelector 는 순차적으로 첫번째 것만 찾아줌

```
document.querySelector(".box2");
document.querySelector("#box3").innerText="박스3";
```

```
<div class="box2">box2</div>
<div id="box3">box3</div>
```

- tag로 elements 찾기

```
document.getElementsByTagName("h1")[0].innerText="dom";
```

2) 이벤트처리

- 버튼 : 버튼의 기본 타입은 summit

```
<button type="button">버튼1</button>
<input type="button" value="버튼2"></button>
```

- 이벤트핸들러 : 이벤트가 발생시 어떻게 처리할건지(핸들) //자바의 이벤트리스너
    - 자바와 달리 리턴값의 타입을 안적음 -> 그래서 MS가 타입을 선언하는게 Typescript

```
<script>
    function click1(){
        console.log("버튼")}
</script>
```

```
<button type="button" onclick="click1()">버튼1</button>
```

- 버튼을 찾아서 이벤트 구현 1, 2 //예전코드 그러나 기억은 해야함

```
<button type="button" onclick="click1()">버튼1</button>
<input id="button2" type="button" value="버튼2"></button>
<input id="button3" type="button" value="버튼3"></button>
```

```
window.onload=function(){
    const button2=document.getElementById("button2");
    button2.onclick=function(){
        console.log("버튼2");
    };
};
```

- 버튼을 찾아서 이벤트 구현 3 //이게 표준 코드

```
    const button3 = document.getElementById("button3");
    button3.addEventListener("click", function(){
        console.log("버튼3");
    })
```

3) validation

- 애로 쓰면 새페이지 넘어감

```
<body>
  <form action ="/login" method = "post">
    id <input type = "text" name="id"><br>
    pw <input type = "password" name ="pw"><br>
    <input type="submit" value="로그인">
  </form>
</body>
```

- 애로 쓰면 새페이지 넘어감

```
    <button>로그인</button>
```

- type 이 button 이면 안넘어감

```
    <input type="button" value="로그인">
```

- 이걸해줘야 미입력, 원하지 않을때 전송(다음페이지)가 안되도록

```
<button onclick="return false;">로그인</button>
```

```
  <script>
    function check(){
      if(document.loginForm.id.value==""){
        alert("id를 입력하세요");
        return; //함수종료, 이렇게해야 pw 얼러트까지 안뜸
      }
      if(document.loginForm.password.value==""){
        alert("pw를 입력하세요");
        return;
      }
      //버튼은 다음페이지가 안넘어가므로
      document.loginForm.submit(); //form submit 으로 넘어가게 해줌
```

```
    };
  </script>
</head>
<body>
  <form name = "loginForm" action ="/login" method = "post">
    id <input type = "text" name="id"><br>
    pw <input type = "password" name ="password"><br>
    <input type="button" value="로그인" onclick="check()">
    <button onclick="return false;">로그인</button>
  </form>
</body>
</html>
```

함수의 리턴값은 호출한 위치로 돌아감.
아래 코드는 자체가 전송을하니 submit 이 없음
"return check2(); 에서 불렀으니까
return false 가 되는 것

```
    const check2=function(){
      if(document.loginForm.id.value==""){
        alert("id를 입력하세요");
        return false; //함수종료, false 리턴
      }
      if(document.loginForm.password.value==""){
        alert("pw를 입력하세요");
        return false; //함수종료, false 리턴
      }
    }
  </script>
</head>
<body>
  <form name = "loginForm" action ="/login" method = "post">
    id <input type = "text" name="id"><br>
    pw <input type = "password" name ="password"><br>
    <input type="button" value="로그인" onclick="check()">
    <!-- submit 버튼 클릭시 return false는 전송을 방지-->
    <button onclick="return check2();">로그인</button>
  </form>
</body>
```

- 함수의 호출방법 2가지

```
    //함수명이 add인것
    function add(a,b){
      return a+b;
    }
```

```
//이 함수는 이름이 없지만 add2가 이름 역할을 하는것
const add2 = function(a,b){
    return a+b;
}
//람다식 표현, return은 안써도 자동으로 return 되는 것
const add3=(a,b) => a+b;

console.log(add(10,20));
console.log(add2(10,20));
console.log(add3(10,20));
```

- https://www.w3schools.com/js/js_es6.asp
- https://developer.mozilla.org/ko/docs/Web/JavaScript