VanillaJS 3주차 내용 정리

- 63. window.onload 이벤트
  - 문서의 모든 컨텐츠(html,css,js) 동일한 문서에는 onload가 하나만 존재해야 한다.
  - 중복될 경우에는 마지막에 선언된 onload가 실행이된다.

- 64. innerHTML 속성
  - 〈div〉〈/div〉나 〈p〉〈/p〉태그 안에 보여질 값을 가져오거나 출력을 할때 사용
  - 〈div id="start"〉innerHTML〈/div〉
  - document.getElementById("start").innerHTML ="0.00";
  - alert(document.getElementById("start").innerHTML);

- 65. 폼 컨트롤의 입력 값 읽기

요소	type 속성의 값	프로퍼티	설명
input	number,text	value	입력된 값을 문자열로 변환한 값
	checkbox,radio	checked	항목의 선택 여부를 뜻하는 논리 값
select		selectedIndex	선택된 option 요소를 가리키는 0부터 시작하는 번호
textarea		value	입력된 문자열

- 66. document.write()
  - javascript로 브라우저에 결과를 출력할때 사용
  - document.write("〈p〉안녕~!!!〈/p〉");

- 67. 함수를 정의하는 방법
  - 1) 함수 선언문으로 정의하는 방법
    - function square(x) { return x*x; };
  - 2) 함수 리터럴로 정의하는 방법
    - var square = function(x){ return x*x; };
  - 3) Function 생성자로 정의하는 방법
    - var square = new Function("x","return x*x");
  - 4) 화살표 함수 표현식으로 정의하는 방법[ECMAScript 6에서 추가]
    - arrow function
    - var square = x => x*x;

- 68. 함수 선언문을 프로그램의 첫머리로 끌어올린다.(hoisting)
  - 자바스크립트 엔진은 함수 선언문을 프로그램의 첫머리 또는 함수의 첫머리로 끌어올린다.
  - 따라서 함수 선언문으로 정의한 함수는 호출문이 그보다 앞에 위치해도 호출할 수 있다.
  - console.log(square(2)); // 4
  - function square(x){ return x*x;}
  -
  - Function 생성자, 화살표 함수 표현식
  - 변수에 그 함수의 참조를 할당해야 비로소 사용할 수 있는 상태가 된다.
  - 따라서 호출하는 코드보다 함수가 앞에 와야 한다.
  -
  - var sqaure = function(x){ return x*x; }
  - console.log(square(2)); // 4

-
- var sqaure = new Function("x","return x*x");
- console.log(square(2)); // 4
-
- var sqaure = x => x*x;
- console.log(square(2)); // 4

- 69. 중첩함수(Nested Function)
  - 지역함수, 내부함수(Inner Function)
  - 외부 함수의 최상위 레벨(제일 바깥쪽 함수)에만 중첩함수를 작성할 수 있다.
  - 함수안의 if문과 while문 등의 문장 블록 안에는 중첩함수를 작성할 수 없다.
  -
  - 중첩함수(내부함수) 안에 있는 변수는 중첩함수(내부함수) 안의 지역변수에 저장된다.
  - 외부 함수의 변수 유효 범위는 내부 함수(중첩함수)에서도 사용이 가능하다.
  -
  - function norm(x){
  - var sum2 = sumSquare();
  - return Math.sqaure(sum2);
  - function sumSquare(){
  - sum = 0;
  - for(var i=0; i<x.length; i++){
  - sum += x[i]*x[i];
  - }
  - return sum;
  - }
  - }
  - var a = [2,1,3,5,7];
  - var n = norm(a);
  - console.log(n);

- 70. 함수 호출하는 방법
  - 함수 호출
    - var s = square(5);
  - 메소드 호출
    - 객체의 프로퍼티에 저장된 값이 함수 타입일 때는 그 프로퍼티를 메소드라고 한다.
    - 메소드를 호출할 때는 그룹 연산자인 ()를 붙여서 호출합니다.
    - obj.m = function() {...};
    - obj.m();
  - 생성자 호출
    - 함수의 참조를 저장한 변수 앞에 new 키워드를 추가하면 함수가 생성자로 동작한다.
    - var obj = new Object();
    - --------
    - 00. 생성자
    - - 생성자로 객체 생성하기
    - function Card(suit, rank){
    - this.suit = suit;
    - this.rank = rank;

        - }
        - =>
        - var card = {};
        - card.suit = "하트";
        - card.rank = "A";
        -
        - 생성자로 객체를 생성할 때는 new 연산자를 사용한다.
        - var card = new Card("하트","A");
        - console.log(card); // -> Card
        -
        - 생성자는 객체를 생성하고 초기화하는 역할을 한다.
        - 생성자를 사용하면 이름은 같지만 프로퍼티 값이 다른 객체(인스턴스) 여러 개를
        - 간단히 생성할 수 있다.
        -
        - var card1 = new Card("하트", "A");
        - var card2 = new Card("클럽", "K");
        - var card3 = new Card("스페이드", "2");
        -
        - 메소드를 가진 객체를 생성하는 생성자
        - function Circle(center, radiuds){
        -     this.center = center;
        -     this.radiuds = radiuds;
        -     this.area = function(){
        -     return Math.PI * this.radiuds * this.radiuds;
        -     }
        - }
        -
        - var p = {x:0, y:0};
        - var c = new Circle(p, 2.0);
        - console.log("넓이 = "+ c.area()); // 넓이 = 12.56637...
        - --------
        - call,apply를 사용한 간접 호출
        - 함수의 call과 apply 메소드를 사용하면 함수를 간접적으로 호출할 수 있습니다.
        - ex) toString.call(obj);// 문자열 형태로 출력을 할 수 있다.

- 71. 즉시 함수 실행
    - 익명함수를 실행할때는 익명함수의 참조를 변수에 할당한 후에 그룹 연산자인()를 붙여서 실행한다.
    - 일반 함수
        - ex1)
        - var f = function(){...};
        - f();
    - 즉시 실행 함수
        - ex2)
        - (function(){...})();
        - (function(){...}());
    - 함수 정의식을 함수 값으로 만들 수 있다.

        - ex3)
        - +function(){...}()
        - +function(){
        - return 1+1;
        - }()
    - 즉시 실행 함수에도 인자값을 넘길 수 있다.
        - ex4) (function(a,b){...})(1,2);
    - 즉시 실행 함수에도 이름을 붙일 수 있지만, 함수 내부에서만 유효하다.
        - ex5)
        - (function fact(n){
        - if(n<=1) return 1;
        - return n*fact(n-1);
        - })(5); // 120
    - 즉시 실행 함수 실행결과를 변수에 할당할 수 있으며, 표현식 안에서 사용할 수 있다.
    - 즉시 실행 함수는 전역 유효 범위를 오염시키지 않는 이름 공간(name space)을 생성할 때 사용한다.
    - (오염 : 같은 이름이나 컨텍스로 등록되어있는 것)
    - var x = (function(){...})();
    - 1) 함수 선언문으로 정의하는 방법 (hoisting)
        - 전역, 로컬 컨텍스트에 등록이 되는 것(함수로 등록)
        - function square(x) { return x*x; };
    - 2) 함수 리터럴로 정의하는 방법
        - 함수가 아니라 var의 객체로서 등록이된다.
        - var square = function(x){ return x*x; };
    - 3) Function 생성자로 정의하는 방법
        - var square = new Function("x","return x*x");
    - 4) 화살표 함수 표현식으로 정의하는 방법[ECMAScript 6에서 추가]
        - var square = x => x*x;

- 72. 함수의 인수(인자, parameter, argument)
    - 1) 인수의 생략
    - 입력할 인수의 개수보다 적게 입력하면 입력되지 않은 인자는 undefined가 된다.
        - function f(x,y){
        - console.log("x = "+x+", y = "+y);
        - }
        - f(2); // x=2, y= undefined
        -
    - 인자값이 부족하게 들어올 경우에 기본값을 지정해서 처리하기
        - function multiple(a,b){
        - b = b || 1;
        - return a*b;
        - }
        - console.log(multiple(2,3)); // 6
        - console.log(multiple(2)); // 2

    - 2) 가변길이 인수 목록(Arguments 객체)
    - 모든 함수에서 사용할 수 있는 지역변수 arguments (구현은 Arguments 객체에 되어있음)

- arguments가 가지고 있는 속성
- arguments.length : 인수의 개수
- arguments.callee : 현재 실행되고 있는 함수의 참조
  - function f(x,y){
  - arguments[1] = 3;
  - console.log("x = "+x+", y ="+y);
  - }
  - f(1,2); // -> x=1, y=3
  -
- 함수에 인수를 여러개 입력해서 호출했을때, 인수를 하나만 받게되면(separator)
- 맨 첫번째 인수 하나만 들어간다. 하지만 arguments는 여러개 입력받은 인자값을 가지고
- 있기때문에 아래와 같이 처리할 수 있다.
- 또한 arguments는 배열처럼 보이지만 배열은 아니고 가짜 배열이다.
  - var params = "";
  - function myConcat(separator){
  - var s = "";
  - for(var i=1; i<arguments.length; i++){
  - s += arguments[i]; // apple/orange/peach
  - if(i< arguments.length-1) s += separator; // 구분자 /
  - }
  - console.log([].slice.call(arguments)); // 배열 객체로 변환할때 사용
  - return s;
  - }
  - // arguments
  - console.log(myConcat("/","apple","orange","peach")); // apple/orange/peach
- 실제로 함수에 arguments.length를 출력해서 확인해 봐도 된다.
  - myConcat(2,3,4,5);
  - function myConcat(a,b,c,d){
  - console.log(arguments.length);
  - }

- 73.Object 선언하고 사용하기(개선된 방법 es6)
  - 우선은 일반적으로 객체를 선언하고 사용할때는 아래와 같이 작성한다.
    - const name = "yong";
    - const age = 23;
    -
    - const obj = {
    - name: name,
    - age: age
    - }
    - console.log(obj);

  - 개선된 방법으로 객체를 선언하고 사용해 보자.
    - function getObj(){
    - const name = "김건모";
    - const getName = function(){
    - return name;

```
    }
    const setName = function(newName){
        name = newName;
    }
    const printName = function(){
        console.log(name);
    }
   return {
        getName : getName,
        setName : setName
    }
}
var obj = getObj();
console.log(obj);
```

- 여기서 name값을 출력하려면 어떻게 해야할까?
- console.log()를 조금 수정을해보자.
  - `console.log(obj.getName());`

- 이 함수를 만든 목적은 기본적으로 객체를 개선된 방법으로 리턴해 보기 위해서 하는 것인데, const로 변수나 함수를 선언하게되면 외부에서 접근을 할 수는 있으나 숨김처럼 동작하기 위한 목적으로 사용하기 때문에 값을 리턴받아서 처리만 하면된다.
- 또한, 값을 넣고 변화된 값을 보고 싶으면 'const name;' 을 'let name;' 으로 변경한 후에 값을 셋팅을 하게되면 넣은 값이 셋팅이 된다.
- 그래서 아래와 같이 코드를 변경해서 실행을 하면 입력된 값을 출력하게 된다.

```
function getObj(){
    let name = "김건모";
    const getName = function(){
        return name;
    }
    let setName = function(newName){
        name = newName;
    }
    const printName = function(){
        console.log(name);
        return name;
    }
    const test = function(){
        return printName();
    }
    return {getName,setName}
}
var obj = getObj();
obj.setName("설현");
console.log(obj.getName()); // 설현
```

- 그런데 여기서 알아둬야 할 것은 이렇게 작성을 해서 return을 하는 순간 나머지는 메모리에서 사라지며, 그렇기 때문에 내부에 있는 함수들은 접근을 할 수가 없다.

- 그리고 이렇게 작성한 객체 내부에서의 함수들을 서로 호출을 할수는 있으나, 실질적으로 외부에서는 사용할 수 없기 때문에 객체를 리턴하는 형태로 사용하면 된다.

- 위에서와 같이 property값이 동일하면 값을 두번 작성하지 않고
- 한번만 작성해도 값을 가져올 수 있다.

- 74. 재귀함수
  - 재귀 함수 : 재귀 호출을 수행하는 함수
  - 재귀 호출 : 함수가 자기 자신을 호출하는 행위(recursive call)
  - 재귀 함수의 기본
    - function fact(n){
    - if(n<=1) return 1;
    - return n*fact(n-1);
    - }
    - fact(5); // -> 120
  - 함수 리터럴로 표현
    - var fact2 = function f(n){
    - if(n<=1) return 1;
    - return n*f(n-1);
    - }
    - fact2(5);
    -
    - arguments.callee로 익명함수 호출
    - var fact3 = function(n){
    - if(n<=1) return 1;
    - return n*arguments.callee(n-1);
    - }
    - fact3(5);

- 75. 재귀함수 호출할 때 유의할 사항
  - 1) 재귀 호출은 반드시 멈춰야 한다. (무한루프 가능성)
  - 2) 재귀 호출로 문제를 간단하게 해결할 수 있을 때만 사용한다.(메모리를 많이 사용 가능)

- 76. module의 이해
  - import/export
  - import는 사용하고자 하는 모듈을 가져올때 쓴다.
  - export는 현재 만들어진 모듈을 내보낼때 사용한다.
  - 기본적으로 javascript는 자신이 필요한 js파일들을 따로 만들어 놓고 관리하기도 하고, 다른 프레임웍을 사용하기도 하기 때문에 다른 js파일에 들어있는 함수나 객체를 가져오기도 해야하고 다른데서 사용하기도 해야 한다. 그럴때 사용하는 것이 import와 export 이다.
    - **[import 예제]**
    - import log from './app03_module.js'; (js파일에서 한개의 함수 가져올때)
    - import log2, {getCurrentHour,MyLogger} from "./myLogger.js" (js파일에서 여러개의 함수나 객체를 가져올 때)
    -
    - **[export 예제]**
    - export class default CodingSchool{}

        - export default function() {}
        - export const getTime(){}
        -
        - **[주의할 부분]**
        - export default는 하나의 js파일에 여러개를 사용하면 오류가 발생함.
        - export default const는 같이 사용하면 안됨. 오류발생
    - 파일 구성
        - index3.html
        - js/app03_module.js
        - js/myLogger.js

- 77.selector
    - document.querySelector()
    - document.querySelector()를 사용하면 ()안에 작성된 태그나 속성에 맞는 객체 하나를 가져오게 된다.
    - 대략 아래와 같은 것들을 가져올 수 있다고 보면 되겠다.
        - document.querySelector("#div");
        - document.querySelector(".myClass");
        - document.querySelector("div.user-panel.main input[name=login]");
        - const root = document.querySelector("#root");
    - document.querySelectorAll()
    - document.querySelectorAll()을 사용하면 ()안에 작성된 태그나 속성에 맞는 객체를 모두 가져올 때 사용된다.
        - var matches = document.querySelectorAll("p");
        - var matches = document.querySelectorAll("div.note, div.alert");

- 78.Destructuring(es6)
    - 비구조화 할당(Destructuring)은 배열,객체, 반복 가능 객체에서 값을 꺼내어 그 값을 별도의 변수에 대입하는 문장이다.
    - 이 방식은 변수에 필요한 값을 할당하는데 유리하다.
        - let data = ["penguin2","suji2","irine2"];

    - 보통 배열에 관련된 데이터 처리를 아래와 같이 한다.
        - let penguin = data[0];
        - let irine = data[2];

    - Destructuring 은 아래와 같이 한다.
        - let[penguin,,irine] = data;
        - console.log(penguin, irine);
        -
        - let data = ["penguin2","suji2","irine2"];
        - let Obj ={
        - name : "penguin2",
        - address : "korea",
        - age : 10
        - }

- let{name,age} = Obj;
- console.log(name,age);

- let{name:myName,age:myAge} = Obj;
- console.log(myName,myAge);

- Destructuring JSON 파싱
  - var news = [
  - {
  - "title":"sbs",
  - "imgurl":"http://static.naver.net/newsstand/2017/0313/article_img/9054/173200/001.jpg",
  - "newslist":[
  - "[가보니] 가상 경주도 즐기고, 내 손으로 자동차도 만들고",
  - "리캡차'가 사라진다.",
  - "갤럭시S8' 출시? '갤노트7' 처리 계획부터 밝혀야",
  - "블로코-삼성SDS, 블록체인 사업 '맞손'",
  - "[블록체인 톺아보기] 퍼블릭 블록체인의 한계와 프라이빗 블록체인"
  -]
  - },
  - {
  - "title":"mbc",
  - "imgurl":"http://static.naver.net/newsstand/2017/0313/article_img/9033/220451/001.jpg",
  - "newslist":[
  - "Lorem ipsum dolor sit amet, consectetur adipisicin",
  - "ipsum dolor sit amet, consectetur adipisicin",
  - "dolor sit amet, consectetur adipisicin",
  - "amet, consectetur adipisicin"
  -]
  - }
  -];

- mbc의 데이터만 가져와 보자
  - let [,mbc] = news;
  - console.log(mbc);
  - **// 이 부분은 수업시간에는 오류가 났었는데, {}를 사용하지 않고, []로 잘못 사용해서 오류가남.**
  - **// 아래처럼 해서 실습하면 title하고 imgurl 잘 가져옴**
  - let {title, imgurl} = mbc;
  - console.log(title, imgurl);
  - let [,{title,imgurl}] = news;
  - console.log(title);

- 이와 비슷하게 function으로 두번째 데이터를 가져올 수도 있다.
  - function getNewsList([,{newslist}]){
  - console.log(newslist);
  - }
  - getNewsList(news);

- 79. event 처리
  - 객체를 찾고
  - 이벤트를 생성하고
  - 객체에 붙인다.
  -
  - 이벤트 처리기를 등록하는 방법
  - 1) HTML 요소의 이벤트 처리기 속성에 설정하는 방법
    - <input type="button" onclick="changeColor();">

  - 2) DOM 요소 객체의 이벤트 처리기 프로퍼티에 설정하는 방법
    - var btn = document.getElementById("button");
    - btn.onclick = changeColor();

  - 3) addEventListener 메소드를 사용하는 방법
    - var btn = document.getElementById("button");
    - btn.addEventListener("click", changeColor, false);

    - <!DOCTYPE html>
    - <html>
    - <head>
    - <meta charset='utf-8'>
    - <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    - <title>Page Title</title>
    - <meta name='viewport' content='width=device-width, initial-scale=1'>
    - <script>
    - window.onload = function(){
    - var element = document.getElementById("box");
    - element.addEventListener("click", function(e){
    - e.currentTarget.style.backgroundColor = "red";
    - });
    - }
    - </script>
    - </head>
    - <body>
    - click me!!!
    - </body>
    - </html>

  - destructuring 이벤트 처리하기
  - 1. index3_04_destructuring.html

- 2. js/app03_ds_event.js
-
- [주의!!!]
- 자바스크립트는 HTML문서에 삽입되는 위치에 따라 스크립트 실행순서와 브라우저 렌더링에 영향을 미친다.
- 자바스크립트에서 DOM(Document Object Model) 구조가 필요한 자바스크립트의 경우 document.onload와 같은 이벤트에 적용하는 것이 좋으며, 그렇지 않을경우에는 DOM을 읽어오지 못할수도 있어 오류가 발생하게 된다.
- 따라서 addEventListener를 사용할 경우에는 객체(DOM)를 가져온 후에 이벤트가 실행되어야 하기때문에 <body></body>태그 사이에 작성을 하는데 맨 아랫부분에 작성을 한다.

- js/app03_ds_event.js
  - document.querySelector("div").addEventListener("click", function({type,target}){
  - console.log(type, target.tagName);
  - });

- index3_04_destructuring.html
  - <!DOCTYPE html>
  - <html>
  - <head>
  - <meta charset='utf-8'>
  - <meta http-equiv='X-UA-Compatible' content='IE=edge'>
  - <title>Destructuring 이벤트 처리</title>
  - <meta name='viewport' content='width=device-width, initial-scale=1'>
  - </head>
  - <body>
  -
  - Lorem ipsum dolor sit amet, consectetur adipisicin ipsum dolor sit amet, consectetur adipisicin dolor sit amet, consectetur adipisicin amet, consectetur adipisicin
  -
  - <script src="js/app03_ds_event.js"></script>
  - </body>
  - </html>