VanillaJS 2주차 내용 정리

- 21. 기본적인 javascript 코드 실행해 보기
  - console.log("Hello, World~!!");
  - var a = 1;
  - var b = 2;
  - console.log(a+b);

- 22.자바스크립트 실행하는 방법
  - 1) browser
    - 마우스 오른쪽 클릭 -> 검사
  - 2) node
    - 해당 javascript 파일 위치로 이동
    - node app.js(파일명)
  - 3) html페이지 내에 넣어서 실행
  - 〈head〉〈/head〉태그 안에 넣어서 실행
  - 〈html〉
  - 〈head〉
  - 〈script type="text/javascript" src="js/app.js"〉〈/script〉
  - 〈/head〉
  - 〈body〉
  - 〈/body〉
  - 〈/html〉

- 23. 변수 선언
  - var sum;
  - var a, b;
  - x = 5; // 바로 대입
  - var a=1, b=2, c=3;
  -
  - console.log(x); // 변수가 선언되어 있지 않아 error 발생
  - console.log(x); // 변수가 선언되어 있지 않고, 출력하는 아래에 할당되어 있지만
  - 위로 끌어올려서 존재는 인식하고 있고, 값을 출력하지 못함.
  - var x=5;
  - console.log(x); // 위에 선언되어 있어서 값을 출력
  - undefined : 값이 없을때 출력되는 것
  -
  - /*
  - 자바를 배웠던 분들에게 좀 더 이해하기 쉽게 설명을 하자면
  - 아래와 같이 클래스를 선언했을때 void형태의 메소드를 실행시켰을때 결과가 undefined가 나온다고 보면 좀 더 쉬울 것 같다. 리턴값이 있을때는 리턴값이 출력되지만, 리턴값이 없을때는 아무것도 안나오는 것이 정상이지만 javascript는 리턴값이 없으면 undefined값이 출력이 된다.
  - class A{
  - public void print(){
  - System.out.println("안녕하세요~!");
  - }

- }
- */

- 24.let 키워드
  - let 키워드는 변수를 선언할 때 사용되는데, ES6에서 추가가 되었다.
  - {}안에서만 사용이 가능하고,
  - var로 똑같은 변수가 선언이 되어있으면, let으로 선언된 것이 우선이 된다.
  - var a;
  - for (let a=0;a<10;a++){
  - console.log(a); // let a가 사용된다.
  - }

- 실습코드 01
  - 일단 기본적인 전역변수와 로컬 변수를 만들어서 출력을 해보자.
    - var name = "global var";
    - function test1(){
    - var localvar = "local var";
    - for(var i=0;i<100;i++){
    - }
    - console.log(i);
    - }
    - test1();

  - let 키워드를 추가해서 출력을 해보자.
    - var name = "global var";
    - function test2(){
    - var localvar = "local var";
    - for(let i=0;i<100;i++){
    - }
    - console.log(i);
    - }
    - test2();

  - let 키워드를 실험할 수 있는 예제를 하나 더 실행해보자.
    - var name = "global var";
    - function test3(){
    - var localvar = "local var";
    - for(let i=0;i<100;i++){
    - console.log(i);
    - }
    -
    - if(true){
    - let check_if = "if값 체크";
    - }
    - console.log(check_if);
    - }
    - test3();

- 25. 객체(Object)
  - 자바스크립트의 타입 : primitive type, reference type(object type)
  - 기본타입을 제외한 모든 타입은 object(reference) type
  - 사용방법
  - 1) var card = { "suit":"하트","rank":"A" };
  - 2) var card = new Card("하트","A");

- 26. 프로퍼티, 1) var card = { "suit":"하트","rank":"A" };
  - "suit":"하트" -> 프로퍼티
  - suit, rank -> 프로퍼티 이름, 프로퍼티 키
  - 프로퍼티 값을 읽거나 사용할 때 : '.' or '[]'
  -
  - ex)
  - var card = { "suit":"하트","rank":"A" };
  - card.suit // -> 하트
  - console.log(card.suit);
  -
  - card["rank"] // -> A
  - console.log(card["rank"]);
  -
  - console.log(card.color); // -> undefined 객체에 없는 프로퍼티 호출시

  - 객체안에 어떤 프로퍼티도 선언하지 않으면 빈객체 생성
    - var obj = {};
    - console.log(obj);

  - 프로퍼티 추가
    - card.value = 14;
    - console.log(card); // Object{suit:"하트", rank:"A", value:14}

  - 프로퍼티 삭제
    - delete card.rank;
    - console.log(card); // Object{suit:"하트", value:14}

  - 특정 프로퍼티가 있는지 확인
    - var card = { "suit":"하트","rank":"A" };
    - console.log("suit" in card); // true
    - console.log("color" in card); // false
    - console.log("toString" in card); // true 상속받은 프로퍼티까지 체크함, 모든 객체는 Object를 상속받음
  - 프로퍼티 표현하는 예제
    - var p = {x:1.0, y:2.5} // 좌표의 평면에 점을 표시하는 데이터

  - 원을 표현하는 객체
    - var circle = {
    - center : {x:1.0, y:2.0},

        - radius : 2.5
        - }
        - circle.center.x // 1.0

    - 회원 정보 표현
        - var person ={
        - name:"수지",
        - age:20,
        - sex:"여",
        - married:false
        - }

    - 프로퍼티에 저장된 값이 함수일때 이 프로퍼티를 메소드라고 한다.
    - 객체는 참조타입
        - var card = { "suit":"하트","rank":"A" };
        - var a = card;
        -
        - console.log(a.suit); // 하트
        - a.suit="스페이드";
        - console.log(a.suit); // 스페이드
        - console.log(card.suit); // 스페이드

- 27. 변수 선언
    - 변수 선언 방법
        - 1) var sum;
        - 2) var sum, a;
    - undefined : 변수안에 값이 정의되지 않았음을 뜻함
    - x = 5; -> x변수에 5라는 숫자값을 대입한다.
    - var a=1, b=2, c=3; -> ','를 사용해서 여러개 값 할당가능

- 28. 변수 선언을 생략하면 오류가 발생한다.
    - console.log(x); // ReferenceError : x is not defined(오류 메시지)

- 29. 변수 선언을 생략한 상태에서 값을 대입하면 오류가 발생하지 않는다.
    - x=2;
    - console.log(x); // 2 출력
    - 변수를 선언하지 않은 상태에서 값을 대입하면 자바스크립트 엔진이
    - 그 변수를 자동으로 전역 변수로 선언한다.

- 30. 변수 끌어올림과 변수 중복 선언
    - console.log(x); // undefined
    - var x;
    - 위의 예제에서 첫번째 줄은 변수가 선언되지 않았기 때문에 오류가 발생할것 같지만,
    - 실제로는 오류가 발생하지 않고 undefined가 출력 된다.
    - 이는 프로그램 중간에서 변수를 선언하더라도 변수가 프로그램 첫 머리에 선언된 것처럼
    - 다른 문장 앞에 생성되기 때문입니다. -> 변수 선언의 끌어올림(hoisting)
    -

    - 선언과 동시에 대입하는 코드는 끌어올리지 않습니다.
    - console.log(x); // undefined
    - var x = 5;
    - console.log(x); // 5

- 31. 변수의 명명 규칙
    - 사용할 수 있는 문자는 알파벳(a~z,A~Z), 숫자(0~9), 밑줄(_), 달러 기호($)다.
    - 첫 글자로는 숫자를 사용할 수 없다. 즉, 첫 글자는 알파벳(a~z,A~Z),밑줄(_),달러 기호($) 중 하나
    - 예약어를 식별자로 사용할 수 없다.
    - 카멜 표기법(camel)
        - 시작은 소문자-> 의미가 변경될때 첫글자 대문자 -> 소문자
    - 밑줄 표기법(snake 표기법)
        - 모든 단어를 소문자로 표기하고 단어와 단어를 밑줄(_)로 구분
        - 변수 이름을 지을때 일반적으로 사용하는 표기법
    - 카멜 표기법이나 밑줄 표기법을 사용하여 변수의 의미를 파악할 수 있도록 이름을 붙인다.
    - 기본적으로 영어 단어를 사용한다.
    - 루프 카운터 변수 이름으로는 i,j,k등을 사용한다.
    - 상수는 대문자로 표현한다.
    - 논리값을 표현하는 변수에는 이름 앞에 is를 붙인다.
    - 생성자 이름을 붙일 때는 파스칼 표기법을 사용한다.

- 32. 예약어
    - 이미 역할이 부여되어 정의되어 있어서, 변수명으로 사용하면 안되는 키워드
        - break, case, catch, class, const, continue
        - debugger, default, delete, do, else, export
        - extends, false, finally, for, function, if
        - import, in, instanceof, new, null, return
        - super, switch, this, throw, true, try
        - typeof, var, void, while, with, yield
        - (ECMAScript 6 이후에 추가될 예정인 예약어)
        - await, enum, implements, package, protected, interface
        - private, public

- 33. 미리 정의된 전역 변수와 전역함수
    - arguments, Array, Boolean, Date
    - decodeURI, decodeURIComponent, encodeURI, encodeURIComponent
    - Error, eval, EvalError, function
    - Infinity, isFinite, isNaN, JSON
    - Math, NaN, Number, Object
    - parseFloat, parseInt, RangeError, RefferenceError
    - RegExp, String, SyntaxError, TypeError
    - undefined, URIError

- 34. Window 객체의 이름과 DOM에서 사용하고 있는 객체 이름도 사용하지 말것!!
    - window.close();

- 35. 데이터 타입
  - 숫자나 문자열처럼 변수에 저장하는 데이터 종류
  - primitive type(기본타입), reference type(참조 타입)
  - 정적 타입 언어(static typed language) : 변수를 지정할때 타입을 미리 지정하는 언어
    - ex) Java
    - int a = 10;
    - String str = "레드벨벳";
  - 동적 타입 언어(dynamic typed language) : 실행할때 변수에 저장된 데이터 타입을 동적으로 바꿀 수 있는 언어
    - ex) Javascript
    - var pi = 3.14;
    - console.log(pi); // 3.14
    - pi = "원주율";
    - console.log(pi); // 원주율

- 36. 데이터 타입의 분류
  - 기본타입(primitive type) : 숫자, 문자열, 논리값, 특수값(undefined,null), 심벌
  - 참조타입(reference/Object type) : Object, primitive type이 아닌 것

- 37. pass by value VS pass by reference
  - 기본타입으로 지정된 값을 다른 변수에 할당하면, 값 자체가 저장된다.(pass by value)
    - var a = 10;
    - b = a; // a = 10, b=10
    - a=20; // a에 값을 20으로 할당해도 b값은 변하지 않는다.
    - console.log(b); // 10
  - 참조타입으로 지정된 값을 다른 변수에 할당하면, 주소값을 복사한다.(pass by reference)
    - var list = [1,2,3,4,5];
    - var cpList = list;
    - console.log("list = ",list," cpList = ",cpList);
    - cpList[2]=5;
    - console.log("list = ",list," cpList = ",cpList);

- 38. 숫자타입
  - 숫자가 64비트 부동소숫점
  - 정수 리터럴
  - 부동 소숫점 리터럴
  - 특수한 값
  - (분류) (표기법) (설명)
  - 전역변수 Infinity 플러스 무한대
  - 전역변수 NAN Not a Number(부정값)
  - Number의 프로퍼티 Number.POSITIVE_INFINITY 플러스 무한대
  - Number의 프로퍼티 Number.NEGATIVE_INFINITY 마이너스 무한대
  - Number의 프로퍼티 Number.MAX_VALUE 표현할 수 있는 최대값
  - Number의 프로퍼티 Number.MIN_VALUE 표현할 수 있는 최솟값
  - Number의 프로퍼티 Number.NaN 부정 값(Not a Number)
  - Number의 프로퍼티 Number.EPSILON 2.22044604925031e-16 [ES6]
  - Number의 프로퍼티 Number.MIN_SAFE_INTEGER -9007199254740990 [ES6]

- Number의 프로퍼티 Number.MAX_SAFE_INTEGER 9007199254740990 [ES6]

- 39. 문자열
  - 16비트 유니코드 문자(UTF-16)
  - 전세계 모든 문자 표현가능
  - " 나 ""를 사용해서 표현할 수 있다.
  - 문자열 안에 문자열을 표시해야할 경우에는 ""안에 "를 사용
  - ex) <input ... onclick="alert('입력되었습니다!')"/>
  - 이스케이프 문자목록(escape)
  - ₩0 널 문자
  - ₩b 백스페이스 문자
  - ₩t 수평 탭 문자
  - ₩n 개행문자
  - ₩v 수직 탭 문자
  - ₩f 다음 페이지 문자
  - ₩r 캐리지 리턴 문자(CR)
  - ₩' 작은 따옴표
  - ₩" 큰 따옴표 문자
  - ₩₩ 역슬래시 문자
  - ₩xXX 두자릿수 16진수 XX로 지정된 Latin-1 문자
  - ₩uXXXX 네자릿수 16진수 XX로 지정된 유니코드 문자
  - ₩u{XXXXXX} 16진수 코드 포인트 XXXXXX로 지정된 유니코드 문자

- 40. 논리값
  - 조건식이 참인지 거짓인지 표현하기 위해 사용하는 값
  - true or false

- 41. 특수한 값
  - 값이없음 : null , undefined
  - null : 아무것도 없음
  - undefined(자바스크립트 엔진이 변수를 undefined로 초기화)
    - . 정의되지 않은 상태
    - . 값을 아직 할당하지 않은 변수의 값
    - . 없는 객체의 프로퍼티를 읽으려고 시도했을 때의 값
    - . 없는 배열의 요소를 읽으려고 시도했을 때의 값
    - . 아무것도 반환하지 않는 함수가 반환하는 값
    - . 함수를 호출했을 때 전달받지 못한 인수의 값

- 42. 템플릿 리터럴
  - ECMAScript 6에서 추가됨
  - `(역따옴표)로 묶은 문자열을 말한다.
  - 줄바꿈 표시 방법
  - 1) 템플릿 리터럴
    - var t = `Man errs as long as
    - he strives.`;
    - var t = `Man errs as long as₩nhe strives.`;
  - 2) 문자열 리터럴

        - var t = "Man errs as long as₩nhe strives.";
    - 이스케이프 문자열을 출력하려면 String.raw를 붙이면 됨.
        - var t = String.raw`Man errs as long as₩nhe strives.`;
    - 문자열 리터럴로 표현
        - var t = "Man errs as long as₩₩nhe strives.";

- 43. 보간 표현식
    - 템플릿 리터럴 안에는 플레이스 홀더를 넣을 수 있다.
    - 플레이스 홀더는 ${...}로 표기한다.
    - 플레이스 홀더 : 실제 값을 나중에 넣기 위해 확보한 공간(장소)
    - ${}를 활용하여 문자열 안에 변수나 표현식의 결괏값을 삽입할 수 있다.
    - ex)
    - var a=2, b=3;
    - console.log(`${a}+${b}=${a+b}`); // -> 2+3 = 5
    - var now = new Date();
    - console.log(`오늘은 ${now.getMonth()+1} 월 ${now.getDate()} 일 입니다. `);

- 44. 함수(function)
    - 인수 : 함수의 입력값 (argument)
    - 반환값 : 함수의 출력값(return)
    - 함수 정의 : function square(x) { return x*x; }
    - return 문 다음에 엔터키를 치지 말것!! 오류가능성
    - 함수이름 : 명명표기법 사용
    - 함수호출 : square(3);
    - 함수 호출시 전달하는 값 : 인수(argument)
    - 함수정의문의 인수 : 인자(parameter)

- 45. 함수의 실행흐름
    - 호출한 코드에 있는 인수가 함수 정의문의 인자에 대입된다.
    - 함수 정의문의 중괄호 안에 작성된 프로그램이 순차적으로 실행된다.
    - return문이 실행되면 호출한 코드로 돌아간다.
    - return문의 값은 함수의 반환값이 된다.
    - return문이 실행되지 않은 상태로 마지막 문장이 실행되면,
      호출한 코드로 돌아간 후에 undefined가 함수의 반환값이 된다.

- 46. 함수 선언문의 끌어올림(hosting)
    - 아래와 같이 작성을 해도 문제없이 실행이 됨.
    - console.log(square(5)); // 5
    - function square(x) { return x; }

- 47. 값으로서의 함수
    - function square(x) { return x; }
    - var sq = square;
    - console.log(sq(5)); // -> 5

- 48. pass by value VS pass by reference - 함수(function)
    - pass by value

        - function add1(x){
        - return x = x + 1;
        - }
        - var a = 3;
        - var b = add1(a);
        - console.log("a = "+a+", b = "+b); // -> a = 3, b = 4
    - pass by reference
        - function add2(p){
        - p.x = p.x + 1;
        - p.y = p.y + 1;
        - return p;
        - }
        - var a = {x:3, y:4};
        - var b = add2(a);
        - console.log(a,b); // Object {X=4, Y=5} Object{X=4, X=5}

- 49. 함수 안에서의 변수 선언 생략
    - 변수를 선언하지 않은 상태에서 값을 대입하면 전역변수가 된다.
        - function f(){
        - a = "local";
        - console.log(a); // local
        - return a;
        - }
        - f();
        - console.log(a); // local

- 50. 함수 리터럴로 함수 정의하기
    - 함수는 함수 리터럴로 정의할 수 있다.
    - 함수 리터럴 표현
        - var square = function (){
        - return 3;
        - }
        - console.log(square());
    - 이름이 없는 함수, 익명함수 또는 무명함수
        - var square = function(){ return X*X; }
        - function(){ return X*X; }
    - 익명 함수에도 이름을 붙일 수 있다.
    - var square = function sq(x){ return X*X; }

- 51. 객체의 메소드
    - 메소드 : 객체의 프로퍼티 중에서 함수 객체의 참조를 값으로 담고 있는 프로퍼티 함수를 값으로 가진 프로퍼티를 말한다.
    - 프로퍼티 : 함수를 값으로 갖고 있지 않은 프로퍼티를 뜻한다.
    - [내 생각 참고]
    - 객체 지향에서는 class라는 것으로 큰 틀을 묶고 그 안에 메소드를 선언을 하는데
    - 자바스크립트는 class라고 묶는 틀이 이후에 나왔기때문에 그 이전에 나온 버젼들에서
    - 객체지향처럼 사용하기위해서 메소드를 이와같이 사용하는 것 같다.

- var circle 이라는 것을 class로 이해하고
- adius:2.5 프로퍼티나 속성으로 보면
- 익명함수 형태로 사용되는 것의 역할이 메소드가 되는 것 같다.
- 따라서 객체지향 프로그램을 구현하기 위한 방법으로 아래와 같이 사용하는 것 같다.

```
var circle ={
  center: {x:1.0, y:2.0},   // 원의 중점을 표현하는 객체
  radius: 2.5,              // 원의 반지름
  area: function(){         // 원의 넓이를 구하는 메소드
      return Math.PI * this.radiuds * this.radius;
  }
}
```

- java code로 변경하면

```
class Circle{
    double x = 1.0;
    double y = 2.0;
    double radius = 2.5;
    double void area(){
        return Math.PI * this.radiuds * this.radiuds;
    }
}
```

- 52. 함수를 활용하면 얻을 수 있는 장점
  - 재사용할 수 있다.
  - 만든 프로그램을 이해하기 쉽다.
  - 프로그램 수정이 간단해진다.

- 53. 생성자(constructor)
  - 생성자로 객체 생성하기

```
function Card(suit, rank){
    this.suit = suit;
    this.rank = rank;
}

java code로 비교해보기 !!!!!
class Card{
  private String suit;
  private String rank;

  public Card(String suit, String rank){
      this.suit = suit;
      this.rank = rank;
  }
}

var card = {};
card.suit = "하트";
```

        - card.rank = "A";
    - 생성자로 객체를 생성할 때는 new 연산자를 사용한다.
        - Card card = new Card("하트","A"); // Java Code
        - 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 생성자의 예
        - 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...
    - 함수를 생성하는 API
        - function add(){}
        - var square = function (){}
        - var square = function sq(){}
        - var square = new Function("x","return x * x");
        - console.log(square(3));
        -
        - var square = new Function("x","return x * x","x");
        - console.log(square(3));
        -
        - var square = new Function("x","y","return x * y");
        - console.log(square(3,7));
    - var 변수이름 = new Function(첫 번째 인수,두 번째 인수, ... n번째 인수,"함수 바디");

- 54. const로 선언된 변수 체크해보기
    - 기본 local variable 출력하기
        - function checkConst(){
        - var localName = "my house";
        - localName = "your house";
        - console.log(localName);
        - }
        - checkConst();
    - const 적용하기
        - function checkConst2(){
        - const arr = [1,2,3,4];

        - arr = [5,6];
        - console.log(arr);
        - }
        - checkConst2();
    - var LOCAL_NAME =""; // es6 이전에는 이렇게 사용했다.
    - const를 사용하면 값을 다시 할당하면 안된다.
    - 변경이 되는 변수는 let을 사용하는 것이 좋다.
    - 될수 있으면 var는 사용을 자제하는 것이 좋다

- 55.const와 immutable array 만들기
    - 기본적인 배열을 선언하고 const 변수에 값을 재할당 해보자.
        - function fruit(){
        - const list = ["apple","orange","watermelon"];
        - list = "podo";
        - }
        - fruit();
    - 배열의 경우에 재할당은 안되지만, 추가는 가능하다.
        - function fruit2(){
        - const list = ["apple","orange","watermelon"];
        - list.push("podo");
        - console.log(list);
        - }
        - fruit2();
    - 예를들어서 앞으로가기나 뒤로가기를 눌러서 저장한 값을
    - 보여줘야 할 경우가 있다. 그럴때 어떻게 할까?
    - 값과 형태가 같아야 할경우에 ===를 사용한다.
    - 보통 같냐?라고 물어볼때는 ==아니라 ===(동치)를 사용한다.
        - const list = ["apple","orange","watermelon"];
        - list2 = [].concat(list,"banana");
        - console.log(list,list2);
        - console.log(list === list2);

- 56. 내장객체(기본객체)
    - 기본 생성자(내장 생성자)
    - ECMAScript 5의 기본 생성자
        - Object : 일반객체
        - String : 문자열 객체
        - Number : 숫자 객체
        - Boolean : 논리값 객체
        - Array : 배열
        - Date : 날짜와 시간을 다루는 객체
        - Function : 함수 객체
        - RegExp : 정규 표현식 객체
        - Error : 오류 객체
        - EvalError : eval() 함수와 관련된 오류를 표현하는 객체
        - InternalError : 자바스크립트 내부에서 발생한 오류를 표현하는 객체
        - RangeError : 값이 허용 범위를 넘었을 때 발생하는 오류를 표현하는 객체

        - ReferenceError : 없는 변수를 참조할 때 발생한 오류를 표현하는 객체
        - SyntaxError : 문법이 어긋날 때 발생한 오류를 표현하는 객체
        - TypeError : 값이 기대한 타입이 아닐 때 발생한 오류를 표현하는 객체
        - URIError : 잘못된 URI를 만났을 때 발생한 오류를 표현하는 객체

    - ECMAScript 6에서 추가된 생성자
        - // Symbol : 심벌을 생성
        - // Int8Array8 : 부호가 있는 8비트 정수 배열을 생성
        - // Uint8Array8 : 부호가 없는 8비트 정수 배열을 생성
        - // Int16Array16 : 부호가 있는 16비트 정수 배열을 생성
        - // Uint16Array16 : 부호가 없는 16비트 정수 배열을 생성
        - // Int32Array32 : 부호가 있는 32비트 정수 배열을 생성
        - // Uint32Array32 : 부호가 없는 32비트 정수 배열을 생성
        - // Float32Array32 : 32비트 부동 소숫점 배열을 생성
        - // Float64Array64 : 64비트 부동 소숫점 배열을 생성
        - // ArrayBuffer : 고정 길이 이진 데이터 버퍼를 생성
        - // DataView : ArrayBuffer 데이터를 읽고 쓸 수 있는 수단을 제공
        - // Promise : 처리 지연 및 비동기 처리를 관리하는 수단을 제공
        - // Generator : 제네레이터 함수를 다룰 수 있는 수단을 제공
        - // GeneratorFunction : 제네레이터 함수를 생성
        - // Proxy : 객체의 기본적인 동작을 재정의하는 기능을 제공
        - // Map : key/value 맵을 생성
        - // Set : 중복을 허용하지 않는 데이터 집합을 생성
        - // WeakMap : 약한 참조를 유지하는 key/value
        - // WeakSet : 약한 참조를 유지하는 고유한 데이터 집합을 생성

- 57. 자바스크립트 객체의 분류
    - 네이티브 객체(native object): ECMAScript 사양에 정의된 객체
        - Object, String, Number, Boolean, Array, Function, JSON, Math, Reflect 등
    - 호스트 객체(host object) : ECMAScript에는 정의되어 있지 않지만 자바스크립트 실행 환경에 정의된 객체
        - Window, Navigator, History, Location, Screen, Document,
        - DOM에 정의되어 있는 객체, Ajax를 위한 XMLHttpRequest객체, HTML5의 각종 API 등
    - 사용자 정의 객체(user define object) : 사용자가 정의한 자바스크립트 코드를 실행한 결과로 생성된 객체

- 58. 자바스크립트 API 사용(빌트인 오브젝트)
    - 0) https://developer.mozilla.org/ko/ (javascript api 찾아보는 url)
        - 참고도서 : 자바스크립트 완벽 가이드
        - http://www.yes24.com/Product/Goods/24769929?scode=032&OzSrank=10
    - 1) Date 생성자
        - var now = new Date();
        - console.log(now);
        - var then = new Date(2008,5,10);
        - console.log(then);
        - var elapsed = now - then;

        - console.log(elapsed);
        - ## 실행에 걸리는 시간 구하기
        - var start = new Date();
        - 작업이 진행될 코드를 작성하는 부분
        - var end = new Date();
        - var elapsed = end - start; // 프로그램 실행에 걸리는 시간
        -
        - Date API에서 제공하는 메소드
        - now.getFullYear();
        - now.getMonth();
        - now.getDate();
        - now.getDay();
        - now.getHours();
        - now.getMinutes();
        - now.getSeconds();
        - now.getMilliseconds();
        - now.toString();
        - now.toLocaleString(); // 지역화된 날짜와 시간 정보를 문자열 형태로 리턴
        - now.toLocaleDateString(); // 지역화된 날짜 정보를 문자열 형태로 리턴
        - now.toLocaleTimeString(); // 지역화된 시간 정보를 문자열형태로 리턴
        - now.getUTCHours(); // UTC 시각의 시간을 뜻하는 숫자 값
        - now.getUTCString(); // UTC 시간과 날짜정보

    - 2) Function 생성자
    - - 함수를 생성하는 API
    - function add(){}
    - var square = function (){}
    - var square = function sq(){}
    - var square = new Function("x","return x * x");
    - var 변수이름 = new Function(첫 번째 인수,두 번째 인수, ... n번째 인수,"함수 바디");
    - 3) 기타 API
        - 전역객체 : 프로그램 어디에서나 사용할 수 있는 객체
        - 전역객체의 property : undefined, NaN, Infinity
        - 생성자 : Object(), String(), Number() 등
        - 전역함수 : parseInt(), parseFloat(), isNaN()등
        - 내장 객체 : Math, JSON, Reflect
        - JSON : JSON을 처리하는 기능을 제공
        - Math : 수학적인 함수와 상수를 제공
        - Reflect : 프로그램의 흐름을 가로채는 기능을 제공
- 59. 배열(Array)
    - 여러개의 값을 나열해서 저장해 놓은 것
    - 배열은 참조타입이다.
        - 0 1 2 3 (index)
        - var evens = [2, 4, 6, 8];
        - var evens = ["2", "4", "6", "8"];
        - 인덱스(index) : []안에 들어있는 값들에 매겨져 있는 번호
        - 빈배열 생성 : var empty = [];

        - console.log(empty); // []
    - 배열요소 값중에 일부를 생략하면 그 요소는 생성되지 않는다.
        - var a = [2, ,4];
        - console.log(a); // [2, undefined, 4]
    - 배열 값에는 기본타입, 참조타입 어떤타입도 올 수 있다.
        - var various = [3.14, "pi", true, {x:1, y:2}, [2,4,6,8]];
    - 배열의 length 프로퍼티는 요소의 갯수를 리턴한다.
        - various.length 프로퍼티
        - various.length() 함수 X
    - 배열의 length 프로퍼티에 현재 배열 값들의 개수보다 작고 0보다 큰 정수값을 넣으면 배열의 길이가 줄어든다.
    - 그리고 입력한 길이 이후의 값들은 삭제된다.
        - var a = ["A","B","C","D"];
        - a.length = 2;
        - console.log(a); // ["A","B"];
    - 없는 배열 요소에 값을 대입하면 새로운 요소가 추가된다.
        - var a = ["A","B","C"];
        - a[3] = "D";
        - console.log(a);
    - push 메소드를 사용하면 요소를 배열 끝에 추가할 수 있다.
        - var b = ["A","B","C"];
        - b.push("D");
        - console.log(b);
    - delete 연산자를 사용해서 특정 배열 요소를 삭제할 수 있다.
        - var b = ["A","B","C","D"];
        - delete b[1];
        - console.log(b); // ["A",undefined,"C","D"]
        - console.log(b.length);
        - 실제로 코드를 돌려보면 undefined가 아니라 empty가 나옴
        - ["A", empty, "C", "D"]
    - delete로 배열의 데이터를 삭제해도 그 배열의 length 프로퍼티 값은 바뀌지 않는다.
- 60. 값비교 예제
    - javascript의 실제 값을 비교할때 어떻게 비교를 해야하는지를 확인해 보자.
    - 이때, '==' 과 '==='의 차이도 확인해 보자.
    - a==b 는 a와 b가 '값'이 같은지를 비교한다.
    - a===b는 a와 b는 '값' 하고 '형식(타입)'이 똑같은지를 비교한다.
    - console.log(null==undefined); // true
    - console.log(null===undefined); // false
    - console.log(1=="1"); // true
    - console.log("0xff"==255); // true
    - console.log(true==1); // true
    - console.log(true=="1"); // true
    - console.log((new String("a")) == "a"); // true
    - console.log((new Number(2)) == 2); // true
    - console.log([2]==2); // true
    - console.log(isNaN("NaN")); // true
    - console.log(isNaN("NAN")); // true

- console.log(isNaN(NaN)); // true
- console.log(NaN===NaN); // false
- console.log(null===undefined); // false
- console.log(1==="1"); // false
- console.log("0xff"===255); // false
- console.log(true===1); // false
- console.log(true==="1"); // false
- console.log((new String("a"))==="a"); // false
- console.log(new Number(2)===2); // false
- console.log([2]===2); // false

- 61. es6(ECMAScript 2015) String의 새로운 메소드들
  - let str = "hello world ~~ ^^";
  - let matchStr = "hello";
  - 시작이 matchStr와 같은 문자열이 있는가?
    - console.log(str.startsWith(matchStr)); // true
  - 끝이 matchStr 일치하는 문자열이 있는가?
    - console.log(str.endsWith(matchStr)); // false
  - "^" 문자열이 포함되어 있는가?
    - console.log(" include set ",str.includes("^")); //
- 62. 연산자
  - a+b 일때, +가 연산자이다.
    - 연산자 우선순위 : 먼저 계산이 되는 순서
    - 연산자 결합법칙 : 왼쪽과 결합해서 연산할지 아니면 오른쪽과 결합해서 연산할지를 결정하는 것
  - 연산자 우선순위와 결합법칙
    -

우선순위	연산자	결합법칙
1	()(그룹연산자)	없음
2	.,[]	왼쪽->오른쪽
	new(인수있음)	오른쪽->왼쪽
3	()(함수호출)	왼쪽->오른쪽
	new(인수없음)	오른쪽->왼쪽
4	++(후위),--(후위)	없음
5	!,~,+(단항),-(부호반전),typeof,void,delete	오른쪽->왼쪽
	++(전위),--(전위)	
6	*,/,%	왼쪽->오른쪽
7	+,-,+(문자열)	왼쪽->오른쪽

8	〈〈,〉〉,〉〉〉	왼쪽->오른쪽
9	〈,〈=,〉,〉=,in,instanceof	왼쪽->오른쪽
10	==,!=,===,!==	왼쪽->오른쪽
11	&	왼쪽->오른쪽
12	^	왼쪽->오른쪽
13	|	왼쪽->오른쪽
14	&&	왼쪽->오른쪽
15	||	왼쪽->오른쪽
16	?:	오른쪽->왼쪽
17	yeild, yeild*	오른쪽->왼쪽
18	=,+=,-=,*=,/=,%=,〈〈=,〉〉〉=,&=,^=,|=	오른쪽->왼쪽
19	...	없음
20	,	왼쪽->오른쪽

■