

디지털 컨버전스 기반 UI/UX Front 전문 개발자 양성 과정(BITCamp)

강사 : 이상훈

수강생: 오진욱

- JavaScript -

Equality

Loose
equality

`==`

Type conversion
데이터 타입 상관 X

Ex) '5' == 5

True

Strict
equality

`====`

No Type conversion
데이터 타입 까지 상관

Ex) '5' === 5

False

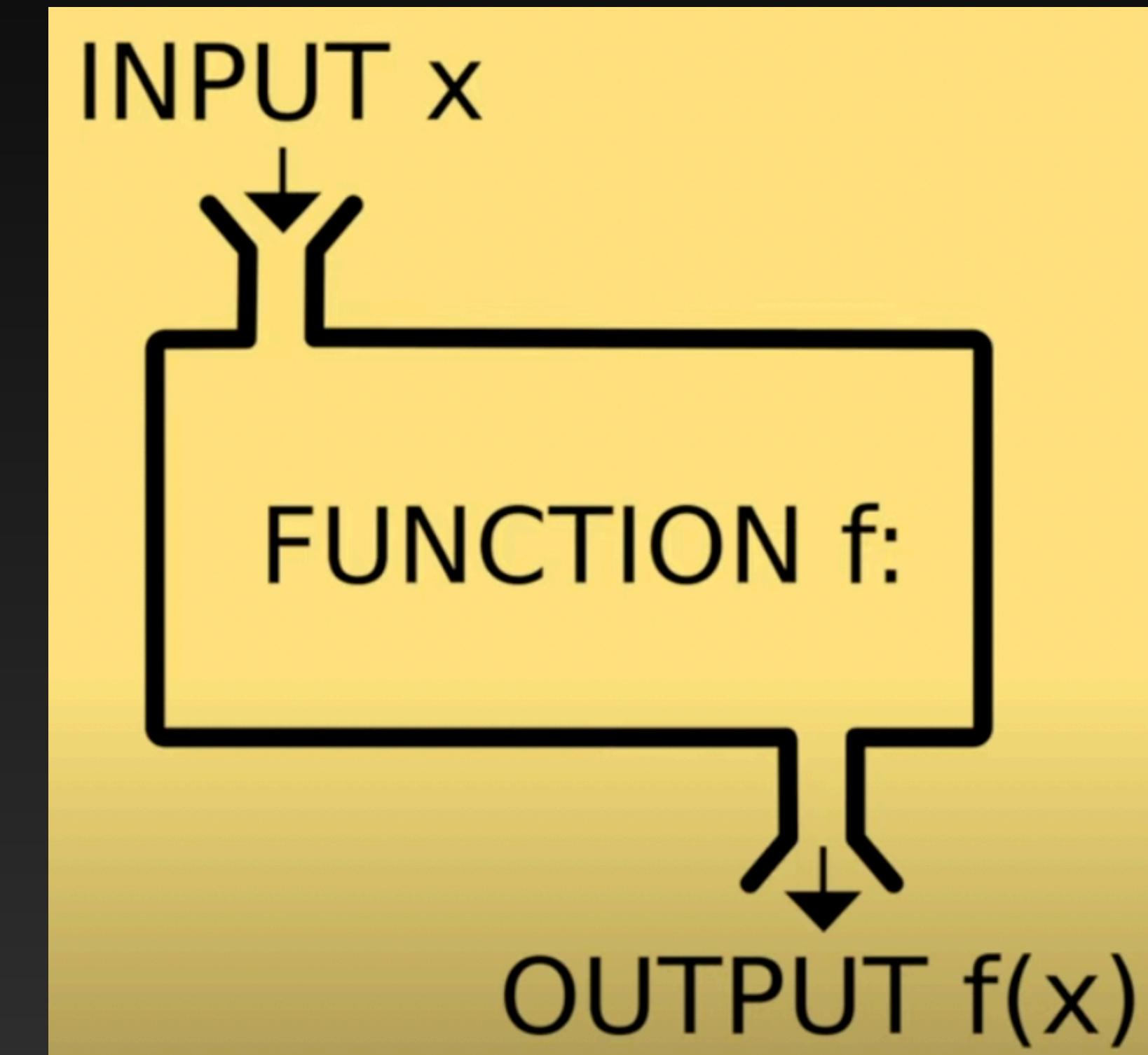
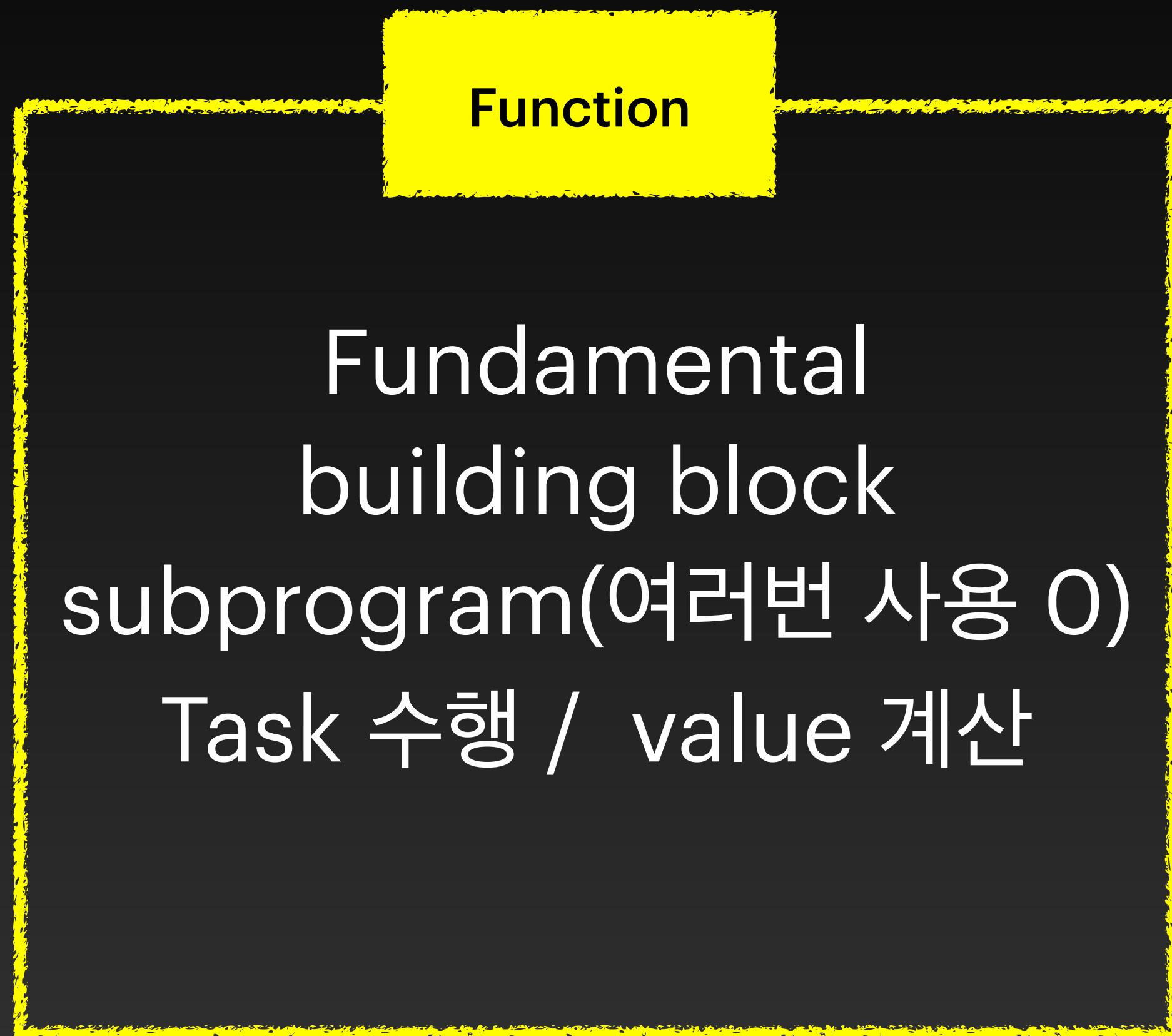
Equality

Object Equality
By reference

Object는 메모리에 올라갈때
Reference 형태로 저장
똑같은 object를 저장해도
메모리에서는 다르다고 인식

```
let equality1 = { name: 'jinwook' };
let equality2 = { name: 'jinwook' };
let equality3 = equality1
console.log(equality1 == equality2) // false
console.log(equality1 === equality2) // false
console.log(equality1 === equality3) // true
```

Function



Function

Function

Function name(입력값){

 수행 내용}

*한가지 일만 수행

*동사형으로 naming하기

function은 object

Object

변수로 할당 가능
파라미터로 전달
리턴이 가능

Function

Parameter

Primitive
->value로 전달
Object
->reference로 전달

Default Parameter

Parameter에 값을
지정 할 수 있는 것을
말함

```
function showMessage(message, from = 'unknown') {  
  console.log(`#${message} by ${from}`);
```

Rest Parameter

... 형태 흘뿌리기
배열 형태로 출력

Function Expression

Function
Declaration

함수는 define 되기 전에
호출이 가능하다(Hoisted)

```
sum(2, 3);  
// 6. Return a value  
function sum(a, b) {  
    return a + b;  
}
```

Function
Expression

할당된 다음부터 호출이 가능
Anonymous function
Named function

```
const print = function () {  
    // anonymous function  
    console.log('print');  
};  
print();  
const printAgain = print;  
printAgain();  
const sumAgain = sum;  
console.log(sumAgain(1, 3));
```

Function Expression

Call back

전달된 함수들 중 원하는 함수
를 불러오는 것을 말함

```
function bitcamp(classNum, printYes, printNo){  
    if(classNum === 502){  
        printYes();  
    }else{  
        printNo();  
    }  
  
    const printYes = function () {  
        console.log('yes!');  
    };  
  
    const printNo = function print() {  
        console.log('no!');  
    };  
  
    bitcamp(502,printYes,printNo) // yes  
    bitcamp(501,printYes,printNo) // no
```

Function Expression

Arrow
Function

Function 표현에 비해 구문이
짧음
익명함수
생성자 사용 X

```
(param1, param2, ..., paramN) => { statements }
(param1, param2, ..., paramN) => expression
// 다음과 동일함: => { return expression; }
```

```
// 매개변수가 하나뿐인 경우 괄호는 선택사항:
(singleParam) => { statements }
singleParam => { statements }
```

```
// 매개변수가 없는 함수는 괄호가 필요:
() => { statements }
```

```
// 객체 리터럴 표현을 반환하기 위해서는 함수 본문(body)을 괄호 속에 넣음:
params => ({foo: bar})
```

```
// 나머지 매개변수 및 기본 매개변수를 지원함
(param1, param2, ...rest) => { statements }
(param1 = defaultValue1, param2, ..., paramN = defaultValueN) => { statements }
```

```
// 매개변수 목록 내 구조분해할당도 지원됨
var f = ([a, b] = [1, 2], {x: c} = {x: a + b}) => a + b + c;
f(); // 6
```

Class & Object

Class

Fields, Methods 의 뮤음

Template

Declare once

No data in

Encapsulation (캡슐화) 가능

상속과 다형성 가능

Syntactical sugar (JS class)

Object

Instance of a class

Created many times

Data in

소프트웨어에서 구현해야 할 대상

Class & Object

Class Checking

instanceof
클래스안의 인스턴스
인지 알 수 있음
상속 받은 클래스 이면
상속한 클래스 또한
true로 나온다

```
console.log(rectangle instanceof Rectangle);
console.log(triangle instanceof Rectangle);
console.log(triangle instanceof Triangle);
console.log(triangle instanceof Shape);
console.log(triangle instanceof Object);
```

Array.methods

indexOf()

배열에서 지정된 요소를 찾을 수 있는 첫번째 인덱스를 반환하고
존재하지 않으면 -1를 반환

.length

배열의 길이를 반환
배열의 최대 인덱스보다 항상 크다

Array.methods

.shift

배열에서 첫번째 요소를 제거하고
그 요소를 반환 한다

```
var myFish = ['angel', 'clown', 'mandarin', 'surgeon'];

console.log('myFish before: ' + myFish);
// "제거전 myFish 배열: angel,clown,mandarin,surgeon"
```

.pop

배열에서 마지막 요소를 제거하고
그 요소를 반환 한다

```
const plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];

console.log(plants.pop());
// expected output: "tomato"
```

Array.methods

.toString()

지정된 배열 및 그 요소를 나타내는 문자열을 반환

```
const array1 = [1, 2, 'a', '1a'];
console.log(array1.toString());
// expected output: "1,2,a,1a"
```

.join()

배열의 모든 요소를 연결해 하나의 문자열로 만듬

```
const elements = ['Fire', 'Air', 'Water'];

console.log(elements.join());
// expected output: "Fire,Air,Water"

console.log(elements.join(''));
// expected output: "FireAirWater"

console.log(elements.join('-'));
// expected output: "Fire-Air-Water"
```

Array.methods

.slice()

배열의 시작부터 끝까지에 대한 복사본을 새로운 배열 객체로 반환

```
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));
// expected output: Array ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));
// expected output: Array ["camel", "duck"]

console.log(animals.slice(1, 5));
// expected output: Array ["bison", "camel", "duck", "elephant"]
```

.splice()

배열의 기존 요소를 삭제 또는 교체하거나 새로운 요소를 추가하여 배열의 내용을 변경

```
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb');
// inserts at index 1
console.log(months);
// expected output: Array ["Jan", "Feb", "March", "April", "June"]

months.splice(4, 1, 'May');
// replaces 1 element at index 4
console.log(months);
// expected output: Array ["Jan", "Feb", "March", "April", "May"]
```

Array.methods

.push()

배열의 끝에 하나 이상의 요소를
추가하고, 배열의 새로운 길이를
반환

```
var sports = ['축구', '야구'];
var total = sports.push('미식축구', '수영');

console.log(sports); // ['축구', '야구', '미식축구', '수영']
console.log(total); // 4
```

.value()

배열의 각 인덱스에 대한 값을 가
지는 새로운 ArrayIterator 객체
를 반환

```
const array1 = ['a', 'b', 'c'];
const iterator = array1.values();

for (const value of iterator) {
  console.log(value);
}

// expected output: "a"
// expected output: "b"
// expected output: "c"
```

Array.methods

.sort()

배열의 요소를 적절한 위치에 정
렬한 후 그 배열을 반환

-1은 내림차순

1은 오름차순

.forEach()

주어진 함수를 배열 요소
각각에 실행

```
const ArrayRepeatTest = () => {
  let arr = [1,3,8,10,5,7,11,19,77,33]
  let txt = ""
  function testFunc(value) {
    txt = txt + value + "<br>"
  }

  console.log("Before testFunc: " + txt)
  // 배열에 있는 요소 하나하나를 testFunc의 입력 인자로 설정함
  // 모든 요소를 순회하고 더이상 요소가 없으면 종지함
  arr.forEach(testFunc)
  console.log("After testFunc: " + txt)
```

Array.methods

.every()

배열 안의 **모든 요소**가 주어진 판별 함수를 통과 하는지 테스트
(**&&연산**)

```
const isBelowThreshold = (currentValue) => currentValue < 40;

const array1 = [1, 30, 39, 29, 10, 13];

console.log(array1.every(isBelowThreshold));
// expected output: true
```

.some()

배열 안의 **어떤 요소**라도 주어진 판별 함수를 통과 하는지 테스트
(**||연산**)

```
const array = [1, 2, 3, 4, 5];

// checks whether an element is even
const even = (element) => element % 2 === 0;

console.log(array.some(even));
// expected output: true
```

Array.methods

.filter()

주어진 함수의 테스트를
통과 하는 모든 요소를
모아 새로운 배열로 반환

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);
// expected output: Array ["exuberant", "destruction", "present"]
```

.map()

배열 내의 모든 요소 각각에
대하여 주어진 함수를 호출한
결과를 모아 새로운 배열에 반환

```
const array1 = [1, 4, 9, 16];

// pass a function to map
const map1 = array1.map(x => x * 2);

console.log(map1);
// expected output: Array [2, 8, 18, 32]
```

Array.methods

.reduce()

배열의 각 요소에 대해
주어진 리듀스 함수를 실행하고,
하나의 결과 값을 반환

```
const array1 = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) => accumulator + currentValue;

// 1 + 2 + 3 + 4
console.log(array1.reduce(reducer));
// expected output: 10

// 5 + 1 + 2 + 3 + 4
console.log(array1.reduce(reducer, 5));
// expected output: 15
```

.find()

주어진 판별 함수를 만족하는 첫
번째 요소 값을 반환

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

console.log(found);
// expected output: 12
```

Array.methods

.lastIndexOf()

배열에서 주어진 값을 발견 할 수 있는 마지막 인덱스를 반환,
없으면 -1을 반환

```
const animals = ['Dodo', 'Tiger', 'Penguin', 'Dodo'];

console.log(animals.lastIndexOf('Dodo'));
// expected output: 3

console.log(animals.lastIndexOf('Tiger'));
// expected output: 1
```

.find()

주어진 판별 함수를 만족하는 첫 번째 요소 값을 반환

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

console.log(found);
// expected output: 12
```

Math.methods

.max()

입력값으로 받은 숫자 중 가장
큰수 반환

```
Math.max(10, 20); // 20  
Math.max(-10, -20); // -10  
Math.max(-10, 20); // 20
```

.min()

입력값으로 받은 숫자 중 가장
작은수 반환

Math.methods

.floor()

주어진 숫자와 같거나 작은 정수
중 가장 큰 수를 반환

.fround()

표현할 수 있는 실수들 중 가장 가
까운 수를 반환

Math.methods

.ceil()

주어진 숫자보다 크거나 같은 숫자 중 가장 작은 숫자를 Integer로 반환

```
Math.ceil(.95);      // 1  
Math.ceil(4);       // 4  
Math.ceil(7.004);   // 8  
Math.ceil(-0.95);  // -0  
Math.ceil(-4);     // -4  
Math.ceil(-7.004); // -7
```

.round()

입력 값을 반올림한 수와 가장 가까운 정수 값을 반환

Set.methods

Set

자료형에 상관 없이 유일한
값을 저장 하는 객체

`new Set([값s])`

값이 없거나 null 넣으면
비어있는 상태로 출력
비어있는 상태로 출력

삽입 순서대로 순회가 가능함

```
let setData = new Set(["Banana", "Watermelon"])
```

```
console.log("NewSetInitTest: " + setData) // {}
```

```
console.log("NewSetInitTest: " + setData.size) // 2
```

Set.methods

.add()

Set 객체 맨 뒤에 주어진
Value 값을 추가함

```
let setData = new Set()  
  
setData.add("cherry")  
setData.add("strawberry")  
setData.add("apple")  
  
console.log(setData) //{"cherry", "strawberry", "apple"}
```

.has()

Set 객체에 주어진 요소가
존재하는지 판별

```
const set1 = new Set([1, 2, 3, 4, 5]);  
  
console.log(set1.has(1));  
// expected output: true  
  
console.log(set1.has(5));  
// expected output: true  
  
console.log(set1.has(6));  
// expected output: false
```

Set.methods

.delete()

지정한 요소를
Set 객체에서 제거

```
const set1 = new Set();
set1.add({ x: 10, y: 20 }).add({ x: 20, y: 30 });

// Delete any point with `x > 10`.
set1.forEach((point) => {
  if (point.x > 10) {
    set1.delete(point);
  }
});

console.log(set1.size);
// expected output: 1
```

.clear()

Set 객체의
모든 내용을 제거

```
const set1 = new Set();
set1.add(1);
set1.add('foo');

console.log(set1.size);
// expected output: 2

set1.clear();

console.log(set1.size);
// expected output: 0
```

Set.methods

.entries()

Set 객체를 순회하는 iterator 리턴
iterator는 Set 객체 안 각각의 an array of [value,value]
를 가지고 있음

```
var mySet = new Set();
mySet.add('foobar');
mySet.add(1);
mySet.add('baz');

var setIter = mySet.entries();

console.log(setIter.next().value); // ["foobar", "foobar"]
console.log(setIter.next().value); // [1, 1]
console.log(setIter.next().value); // ["baz", "baz"]
```

.keys()

Set 객체를 순회하는 iterator리턴
iterator는 Set 객체 안 요소를 포함

```
let setData = new Set()
setData.add("cherry")
setData.add("strawberry")
setData.add("apple")

console.log(setData)

var setIter = setData.keys()

for(var key of setIter){
  console.log(key) // cherry
  // strawberry
  // apple
```

Set.methods

.forEach()

주어진 함수를
Set 객체에 순서대로 삽입

```
let setData = new Set()  
  
setData.add("Cherry")  
setData.add("strawberry")  
setData.add("apple")  
  
console.log(setData)  
  
setData.forEach(function(val1, val2) {  
  console.log(val1 + " : " + val2)  
  // [Log] Cherry : Cherry  
  // [Log] strawberry : strawberry  
  // [Log] apple : apple
```

.value()

Set 객체 요소에 삽입된 순서대로
각 요소의 값을 순환 할 수 있는
Iterator 리턴

```
const set1 = new Set();  
set1.add(42);  
set1.add('forty two');  
  
const iterator1 = set1.values();  
  
console.log(iterator1.next().value);  
// expected output: 42  
  
console.log(iterator1.next().value);  
// expected output: "forty two"
```

Map.methods

Map()

Key - 값 쌍으로 저장하고
각 쌍의 삽입 순서도 기억하는 콜렉션
모든 값이 키와 값이 될 수 있음

```
let mapData = new Map([["apple", "red"], ["grape", "purple"]])  
  
console.log(mapData) // {"apple" => "red", "grape" => "purple"}  
console.log("Size: " + mapData.size) // Size: 2
```

.methods()

Set.methods과 유사
출력값이 다르게 나온다는 것 주의

Map.methods

.get()

Map 객체에서 지정한 요소를 회수

```
const map1 = new Map();
map1.set('bar', 'foo');

console.log(map1.get('bar'));
// expected output: "foo"

console.log(map1.get('baz'));
// expected output: undefined
```

.set()

Map 객체에서 주어진 키를 가진 요소를 추가하고 키의 요소가 이미 있다면 대체함

Map.methods

.delete()

Map 객체에서 특정 요소를 제거

```
const map1 = new Map();
map1.set('bar', 'foo');

console.log(map1.delete('bar'));
// expected result: true
// (true indicates successful removal)

console.log(map1.has('bar'));
// expected result: false
```

.clear()

Map 객체의 모든 요소를 제거

Map.methods

.value()

Map 객체 요소에 삽입된 순서대로
각 value를 순환 할 수 있는
Iterator 리턴

```
const map1 = new Map();
map1.set('0', 'foo');
map1.set(1, 'bar');

const iterator1 = map1.values();
console.log(iterator1.next().value);
// expected output: "foo"
console.log(iterator1.next().value);
// expected output: "bar"
```

.forEach()

Map 객체 내의 key/value
쌍의 개수 만큼 주어진 함수를
순서대로 실행

```
function logMapElements(value, key, map) {
  console.log(`m[${key}] = ${value}`);
}

new Map([['foo', 3], ['bar', {}], ['baz', undefined]])
  .forEach(logMapElements);

// expected output: "m[foo] = 3"
// expected output: "m[bar] = [object Object]"
// expected output: "m[baz] = undefined"
```

Map.methods

.entries()

Map 객체의 각 요소에 해당하는 [key , value] 쌍을 Map에 등록한 순서대로 포함하는 새로운 intorator 객체 반환

```
const map1 = new Map();
map1.set('0', 'foo');
map1.set(1, 'bar');

const iterator1 = map1.entries();
console.log(iterator1.next().value);
// expected output: ["0", "foo"]

console.log(iterator1.next().value);
// expected output: [1, "bar"]
```

.key()

Map 객체의 각 요소에 해당하는 key값을 map에 등록한 순서대로 포함하는 새로운 intorator 객체 반환

```
const map1 = new Map();
map1.set('0', 'foo');
map1.set(1, 'bar');

const iterator1 = map1.keys();
console.log(iterator1.next().value);
// expected output: "0"

console.log(iterator1.next().value);
// expected output: 1
```

Symbol

Symbol

Symbol은 객체가 아니라
변경 불가능한 원시 타입의 값

이름의 충돌 위험이 없는 유일한
객체의 프로퍼티 키를 만들기 위해 사용

new 연산자 사용 X
() 안에는 문자열 인자 전달 가능
그러나 아무 영향 X
설명 또는 디버깅용도로 사용

```
// 심볼 mySymbol은 이름의 충돌 위험이 없는 유일한 프로퍼티 키
let mySymbol = Symbol();

console.log(mySymbol);           // Symbol()
console.log(typeof mySymbol);   // symbol
```

Symbol

Symbol.iterator

어떤 객체가 symbol.iterator
프로퍼티 키로 사용한 메소드를
가지고 있으면 순회하는 정보인
iterator를 사용한다고 간주하고 동작

// 이터블

// Symbol.iterator를 프로퍼티 key로 사용한 메소드를 구현하여야 한다.
// 배열에는 Array.prototype[Symbol.iterator] 메소드가 구현되어 있다.

```
const iterable = ['a', 'b', 'c'];
```

// 이터레이터

// 이터블의 Symbol.iterator를 프로퍼티 key로 사용한 메소드는 이터레이터를 반환한다.

```
const iterator = iterable[Symbol.iterator]();
```

// 이터레이터는 순회 가능한 자료 구조인 이터러블의 요소를 탐색하기 위한 포인터로서 value, done 프로퍼티를 갖는 객체를 반환하는 next() 함수를 메소드로 갖는 객체이다. 이터레이터의 next() 메소드를 통해 이터러블 객체를 순회할 수 있다.

```
console.log(iterator.next()); // { value: 'a', done: false }
console.log(iterator.next()); // { value: 'b', done: false }
console.log(iterator.next()); // { value: 'c', done: false }
console.log(iterator.next()); // { value: undefined, done: true }
```

Symbol

Symbol.for()

인자로 전달받은 문자열을 키로 사용하여
symbol 값들이 저장 되어 있는 전역
symbol 레지스트리에서 해당 키와 일치하는
저장된 symbol 값을 검색 후 검색되면
검색된 심볼값 리턴
실패하면 새로운 심볼값 생성 하여
레지스트리에 저장 후 리턴

```
// 전역 Symbol 레지스트리에 foo라는 키로 저장된 Symbol이 없으면 새로운 Symbol 생성
const s1 = Symbol.for('foo');

// 전역 Symbol 레지스트리에 foo라는 키로 저장된 Symbol이 있으면 해당 Symbol을 반환
const s2 = Symbol.for('foo');

console.log(s1 === s2); // true
```

```
const shareSymbol = Symbol.for('myKey');
const key1 = Symbol.keyFor(shareSymbol);
console.log(key1); // myKey

const unsharedSymbol = Symbol('myKey');
const key2 = Symbol.keyFor(unsharedSymbol);
console.log(key2); // undefined
```

Iterator

Iterator

Array, Set, Map 객체를
next()로 순환 할 수 있는 객체

```
const IteratorTest= () => {
  let iterData = ["one", "two", "three"]

  for(let entry of iterData){
    console.log(entry)
  }

  let varData = ["four", "five", "six"]

  let itr = varData[Symbol.iterator]()

  console.log(itr.next())
  console.log(itr.next())
  console.log(itr.next())
  console.log(itr.next())
  // iteration 이 완료 되면 true 리턴함

  let varItr = varData.values()

  console.log(varItr.next())
  console.log(varItr.next())
  console.log(varItr.next())
  console.log(varItr.next())
```