디지털컨버전스 기반 UI UX Front 전문 개발자 양성과정

Lecturer silenc3502 Sanghoon Lee (이상훈)
gcccompil3r@gmail.com

Student may_hz HyeonJeong Choi (최현정)
hyeonjeong9943@gmail.com

Class

DATE . 2021_02_01

```
class Person {
    name
    age
    speak()
}
```

속성 (field)

행동 (method)

Class

- template
- declare once
- no data in

Object

- instance of a class
- created many times
- data in

Class

DATE . 2021_02_01

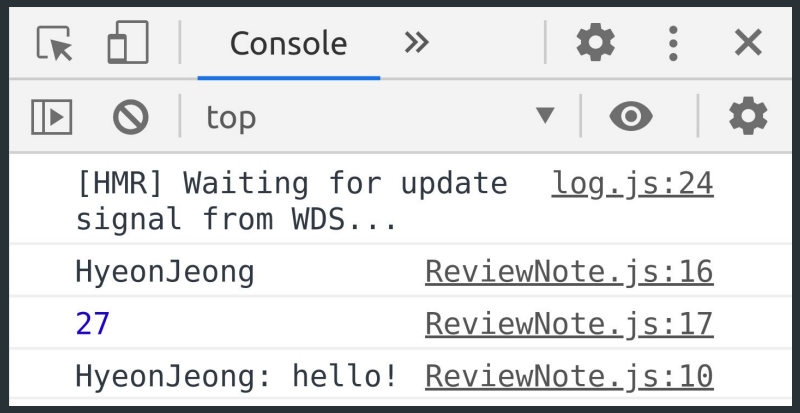
```
class Person {
  constructor(name, age) {
    this.name = name
    this.age = age
  }
  speak() {
    console.log(`${this.name}: hello!`)
  }
}
```

constructor

fields

methods

```
const may = new Person('HyeonJeong', 27)
console.log(may.name)
console.log(may.age)
May.speak()
```



Static

DATE . 2021_02_01

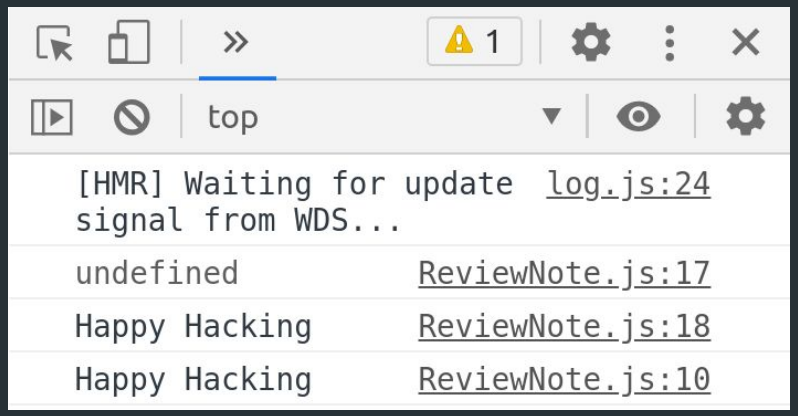
```
class Article {
  static publisher = 'Happy Hacking'
  constructor(number) {
    this.number = number
  }
  static printPublisher() {
    console.log(Article.publisher)
  }
}
const article1 = new Article(1)
const article2 = new Article(2)

console.log( article1 . publisher )
console.log( Article .publisher )
Article .printPublisher()
```

(object.object) ⇨ undefined

(class.static) ⇨ Happy Hacking

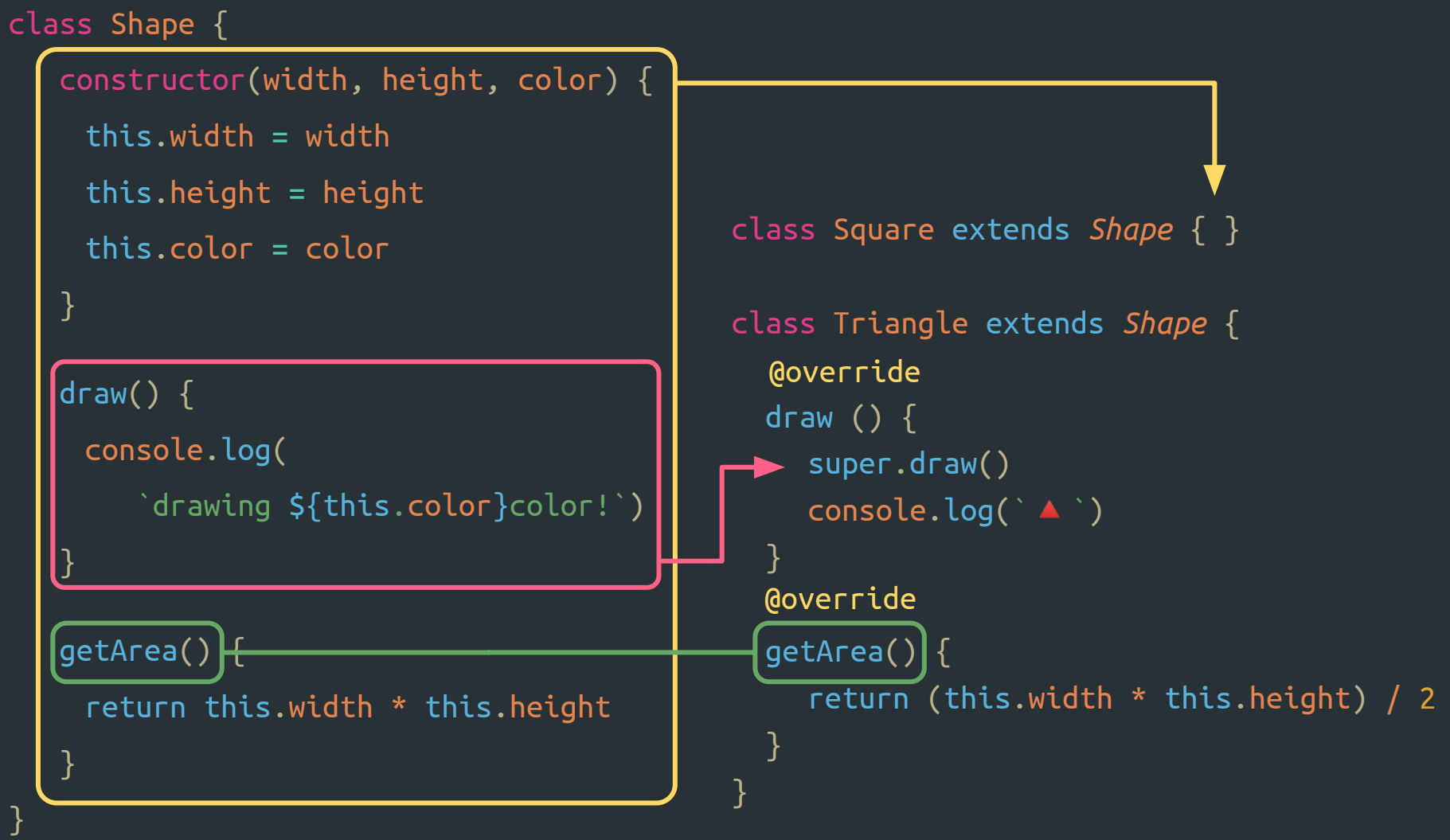
new Class() ⇨ different object

Object에 들어오는 Data에 상관없이 공통적으로 Class에서 사용할 수 있는거라면 Static & Static Method를 이용하여 작성하는 것이 메모리의 사용을 줄여줄 수 있다.



Extends

DATE . 2021_02_01



Extends

DATE . 2021_02_01

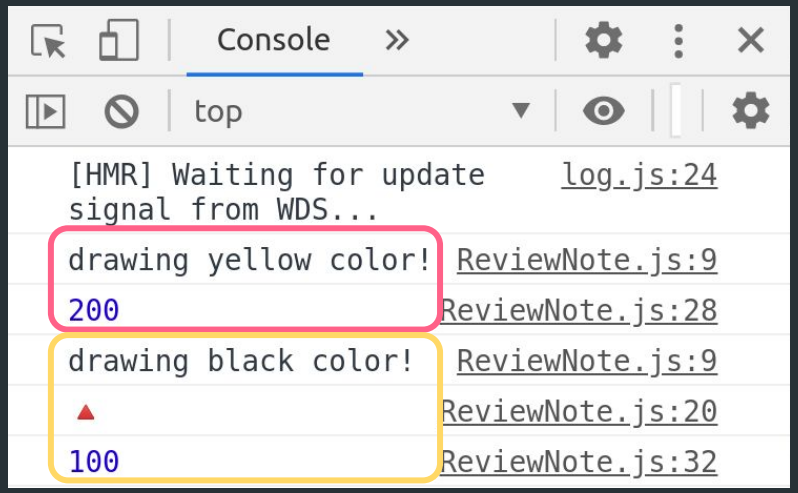
```
const square = new Square(10,20,'yellow')
square.draw()
console.log(square.getArea())

const triangle = new Triangle(10,20,'black')
triangle.draw()
console.log(triangle.getArea())
```

Square는 Shape을 extends 하면서 @override 없음

```
draw() {
  console.log(
    `drawing ${this.color}color!`)
}

getArea() {
  return this.width * this.height
}
```

Triangle은 draw(), getArea()를 @override

```
@override
draw () {
  super.draw()
  console.log(` ▲ `)
}
@override
getArea() {
  return (this.width * this.height) / 2
}
```



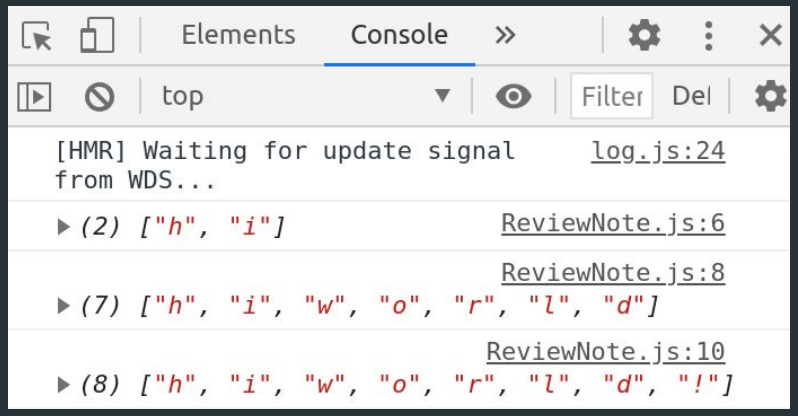
Concat

DATE . 2021_02_01

arr.concat(value)

배열의 끝에 value 요소들을 추가한 새로운 사본을 반환

```
var hi = ['h', 'i']
var world = ['w', 'o', 'r', 'l', 'd']
var mark = ['!']

console.log(hi)
hi = hi.concat(world)
console.log(hi)
hi = hi.concat(mark)
console.log(hi)
```



Join

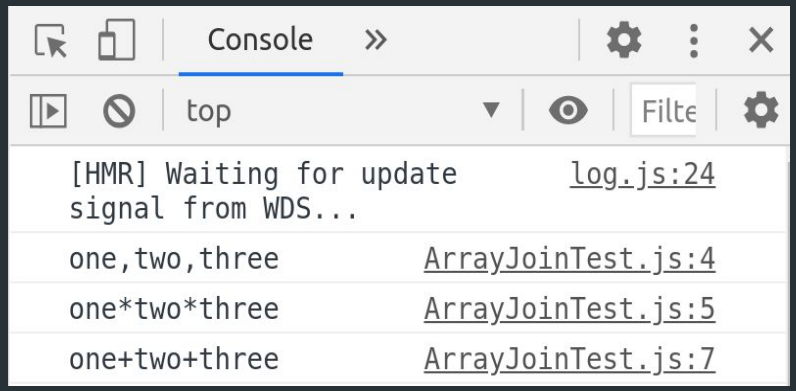
DATE . 2021_02_01

arr.join(seperator)

배열의 모든 요소를 인수로 전달받은 구분문자를 사용해 하나의 문자열로 만들어 반환

만약 구분문자를 지정해 주지 않으면 기본값은 쉼표 " , "

```
var arr = ["one", "two", "three"]

console.log(arr.toString())
console.log(arr.join("*"))

console.log(arr.join("+"))
```



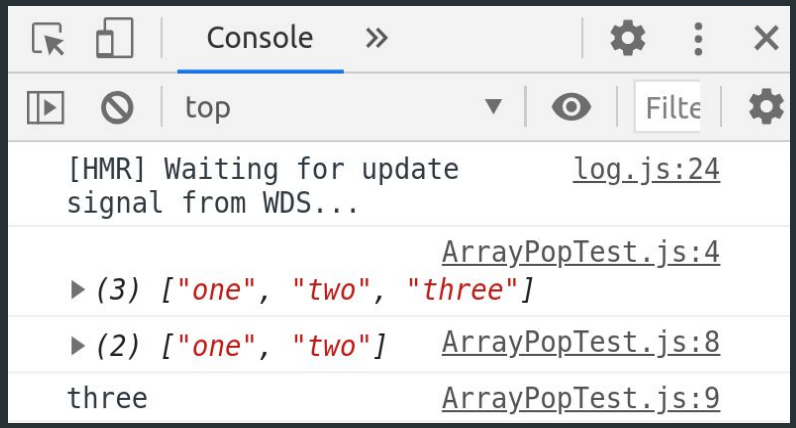
Pop

DATE . 2021_02_01

arr.pop()

배열의 마지막 요소를 제거한 후, 제거한 요소를 반환

```
var arr = ["one", "two", "three"]

console.log(arr)

let resultPop = arr.pop();

console.log(arr)
console.log(resultPop)
```



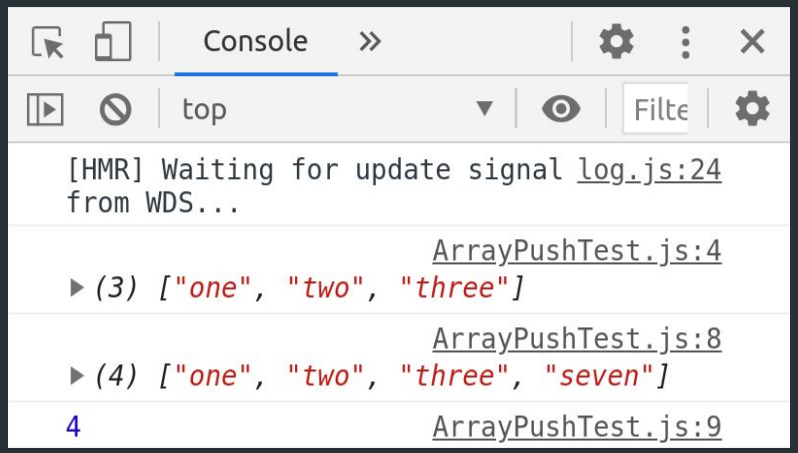
Push

DATE . 2021_02_01

arr.push()

배열의 마지막에 새로운 요소를 추가한 후, 변경된 배열의 길이를 반환

```
var arr = ["one", "two", "three"]

console.log(arr)

let resultPush = arr.push("seven")

console.log(arr)
console.log(resultPush)
```



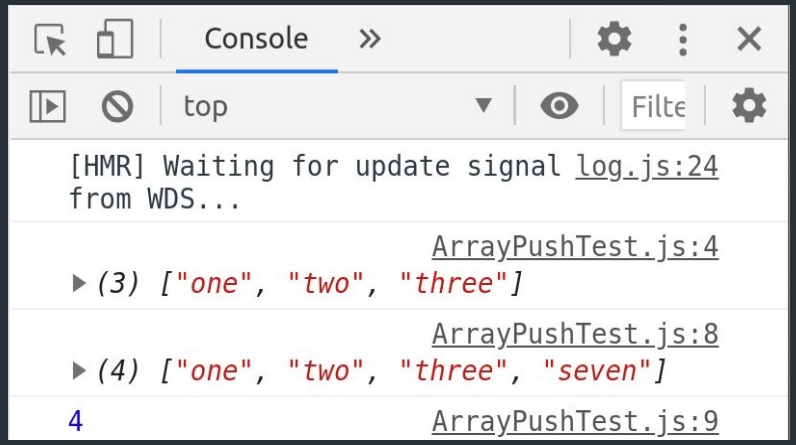
Shift

DATE . 2021_02_01

arr.shift()

배열의 첫 번째 요소를 제거한 후, 제거한 요소를 반환

```
let arr = ["one","two","three"]

console.log(arr)

let resultShift = arr.shift()

console.log(arr)
console.log(resultShift)
```



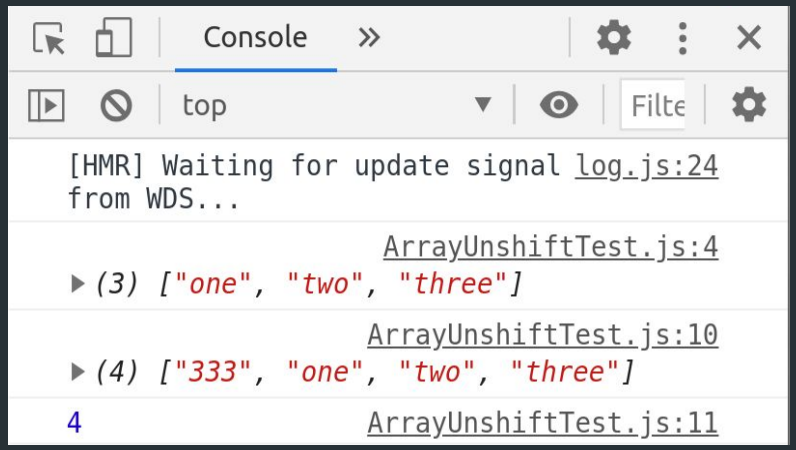
Unshift

DATE . 2021_02_01

arr.unshift()

배열의 처음에 요소 추가. 배열의 크기 리턴

```
let arr = ["one","two","three"]

console.log(arr)

let resultUnshift = arr.unshift("333")

console.log(arr)
console.log(resultUnshift)
```



Slice

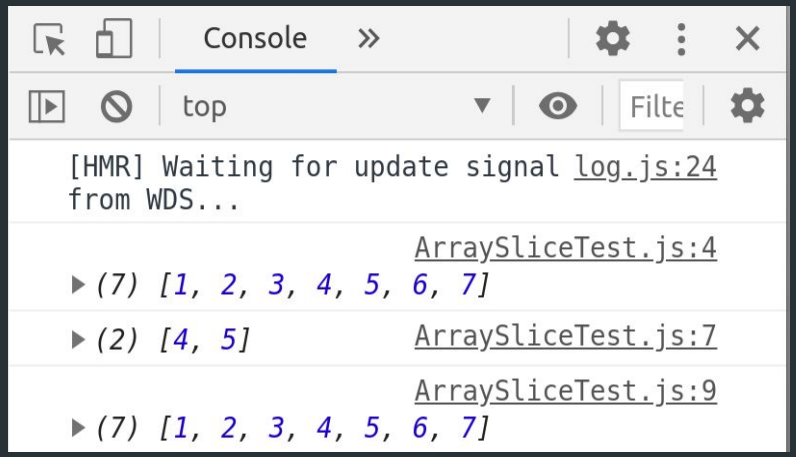
DATE . 2021_02_01

arr.slice(start,end)

인수를 통해 지정한 만큼의 요소를 잘라낸 후 해당 배열을 반환

but.원본 배열은 그대로

```
let arr = [1, 2, 3, 4, 5, 6, 7]

console.log(arr)

let slicedArr = arr.slice(3,5)

console.log(slicedArr)
console.log(arr)
```



Splice

DATE . 2021_02_01

수정할 배열 요소의 인덱스 — 배열에 추가될 요소

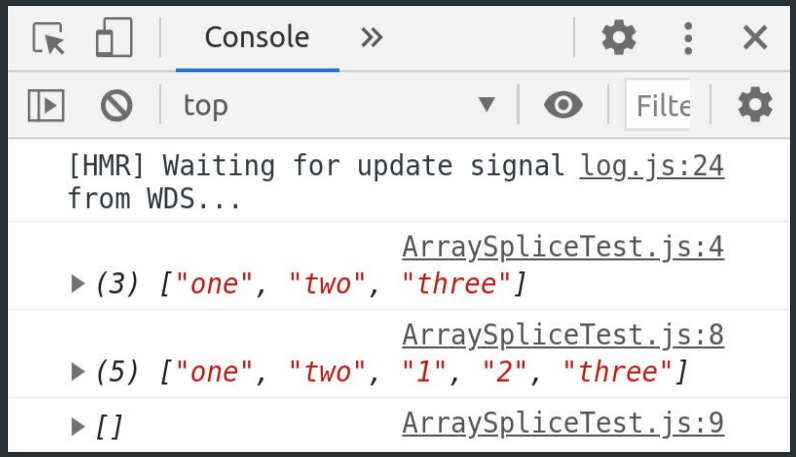
```
arr.splice(start,deleteCount,el)
```

삭제할 요소 개수. 제거하지 않을 경우 0

배열의 특정 위치에 배열 요소를 추가하거나 삭제하는데 사용

리턴값은 삭제한 배열 요소. 삭제한 요소가 없더라도 빈 배열을 반환

```
let arr = ["one","two","three"]

console.log(arr)

let result = arr.splice(2,0,"1","2")

console.log(arr)
console.log(result)
```



Sort

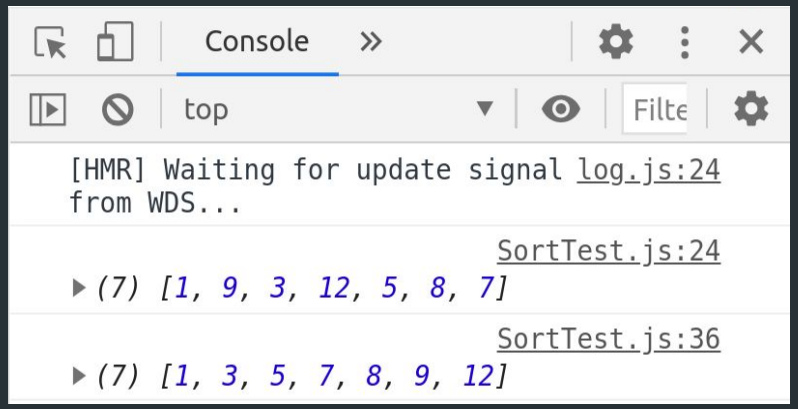
DATE . 2021_02_02

arr.sort(function(a,b) {})

기본적으로 배열의 요소를 문자열로 변환한 후 오름차순으로 정렬

결과값 < 0 : a 낮은 색인정렬
결과값 = 0 : a와 b의 순서 그대로
결과값 > 0 : b 낮은 색인 정렬

```
let arr = [1, 9, 3, 12, 5, 8, 7]

console.log(arr)
let sortedArr = arr.sort()
let sortedArr =
        arr.sort((a, b) => a - b)
console.log(sortedArr)
```



Arrow function

DATE . 2021_02_02

```
const simplePrint = function() {
    console.log('simplePrint!')
}

const add = function(a,b) {
    return a+b
}

const simplePrint = () => console.log('simplePrint!')
const add = (a,b) => a+b

const simpleMultiply = (a,b) => {
    // do something more...
    return a * b
}
```

화살표 함수(Arrow function)는 function 키워드 대신

화살표(=>)를 사용하여 보다 간략한 방법으로 함수를 선언하는 함수

but.모든 경우 화살표 함수를 사용할 수 있는 것은 아님

"{ }"를 사용하면 값을 반환할 때 return을 사용해야함

"{ }"를 사용하지 않으면 undefied를 반환함

"{ }"안에 내용을 여러줄을 썼을 때 return을 사용해야함