02

클래스와 인스턴스 활용

학습내용

01 클래스와 인스턴스

02 JavaScript에서의 객체 정의

03 배열 객체의 활용

학습목표

- 클래스와 인스턴스의 개념을 이해하고, JavaScript에서 클래스 기반 객체를 정의할 수 있다.
- JavaScript에서 객체를 다양한 방식(리터럴, 생성자 함수, 클래스)으로 정의하고 속성과 메소드를 조작할 수 있다.
- 배열 객체의 메소드를 활용하여 데이터를 효과적으로 탐색, 변형, 가공할 수 있다.

01

클래스와 인스턴스

1) 클래스와 인스턴스의 이해

클래스 = 붕어빵 틀

틀 = “먹을 수 없는”

객체 생성

객체 = 붕어빵

실체 = “먹을 수 있는”

=

=

클래스

틀 = “실행할 수 없는”

실체화(메모리 할당)

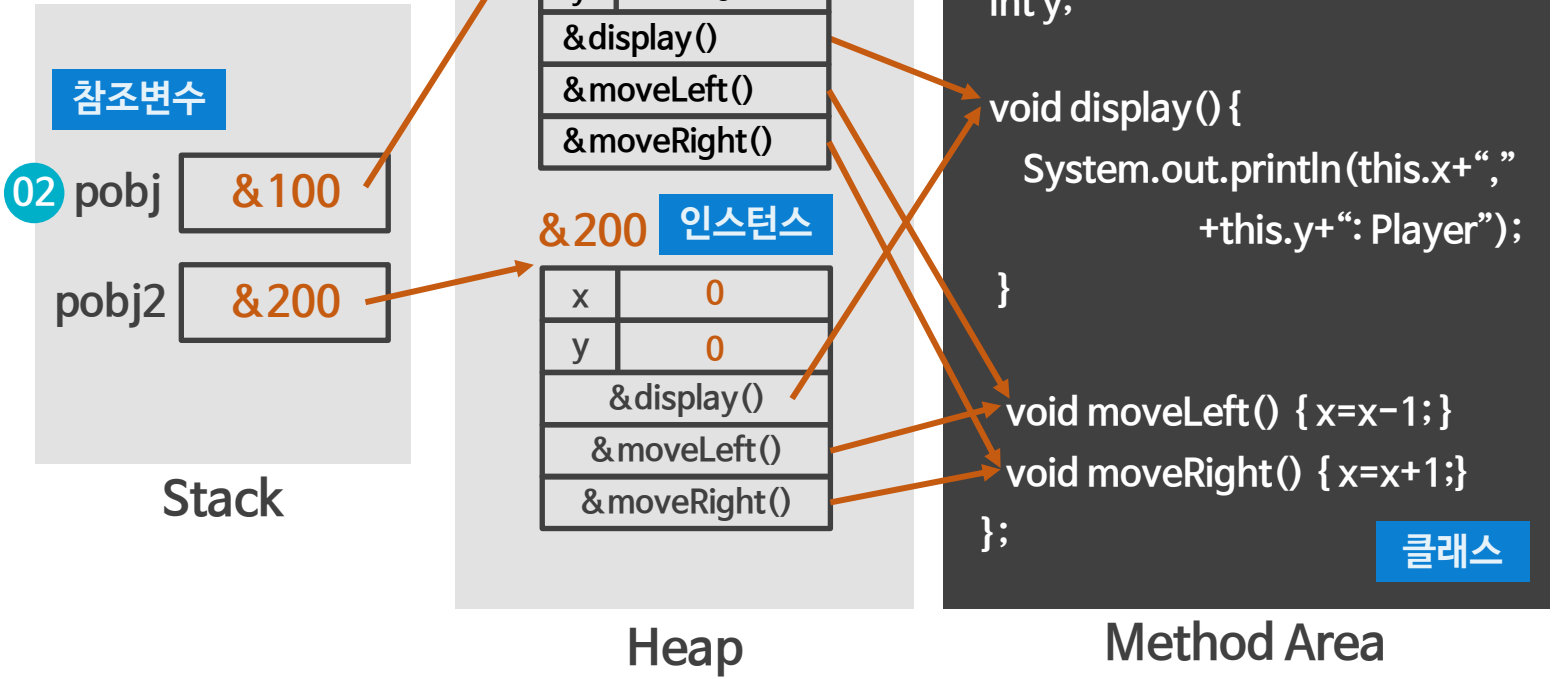
인스턴스(객체)

실체 = “실행 가능한”

2) 클래스 기반의 객체생성 원리(JAVA의 경우)

세종사이버대학교

```
Player pobj = new Player();
Player pobj2= new Player();
```



02

JavaScript에서의 객체 정의

1) JavaScript의 객체지향적 특징

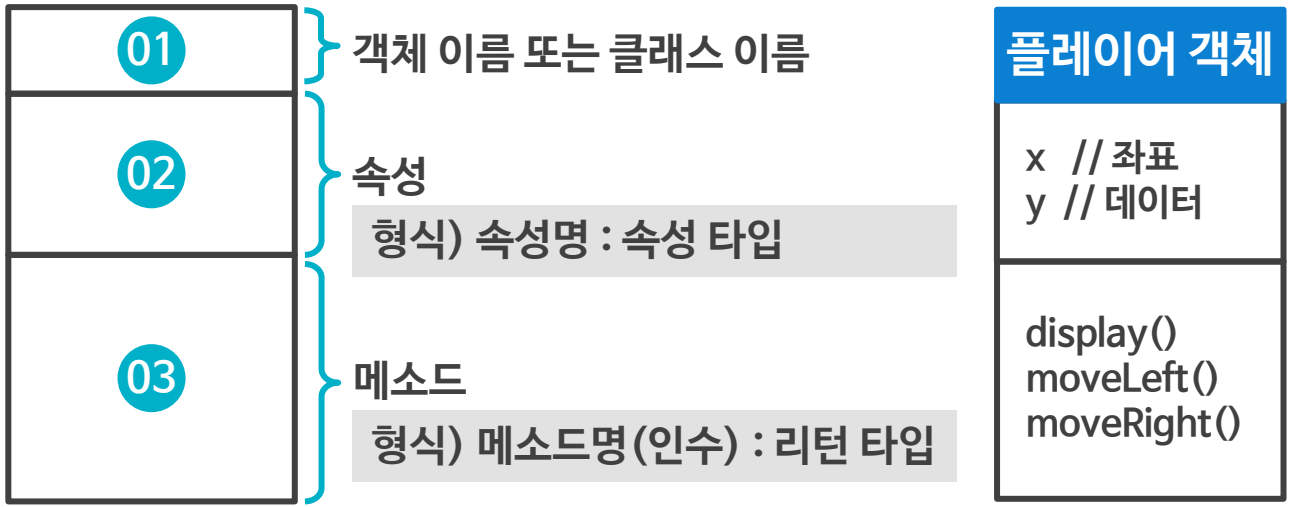
- JavaScript는 함수형 + 객체지향 혼합 지원
  - ➡ 멀티 패러다임 언어
- 전통적인 클래스 기반 X ➡ 프로토타입 기반 OOP
- ES6 이후부터는 Class 문법 제공으로 더 친숙하게 사용 가능

2) 객체 정의 방식

객체 리터럴 { }를 사용한 선언 방법

생성자 함수 패턴 활용(ES5) 선언 방법

클래스(ES6) 선언 방법



플레이어 객체

x // 좌표
y // 데이터

display()
moveLeft()
moveRight()

3) 객체 리터럴 { }를 사용한 방식


```
const player1 = {
  x: 0,
  y: 0,
  display: function() {
    console.log(this.x + "," + this.y + ": Player");
  },
  moveLeft: function() {
    this.x -= 1;
  },
  moveRight: function() {
    this.x += 1;
  }
};

// 사용
player1.moveRight();
player1.display(); // 예: 1,0: Player
```

- 단 하나의 객체만 만들 때 적합
- 동일한 구조의 여러 개의 객체를 만들기엔 비효율적 (복제 어려움)

4) 생성자 함수 패턴 활용(ES5) 선언 방식


```
function Player(x, y) {   // 생성자함수➡초기화
  this.x = x;
  this.y = y;
}

Player.prototype.display = function() {
  console.log(this.x + "," + this.y + ": Player");
};

Player.prototype.moveLeft = function() {
  this.x -= 1;
};

Player.prototype.moveRight = function() {
  this.x += 1;
};
```

1/2

- 여러 인스턴스 생성 가능 (new 키워드 사용)
- 메소드는 프로토타입에 정의하여 메모리 절약 ➡ 클래스의 상속 개념을 프로토타입 기반으로 흉내 낸 구조
- 실제로는 클래스가 아닌 객체의 원형(Prototype)을 참조하는 구조이지만, 사용자는 마치 클래스처럼 인스턴스를 만들고 공통 기능을 재사용 가능 ➡ 프로토타입 기반
- 동적인 속성 및 메소드 추가 가능

4) 생성자 함수 패턴 활용(ES5) 선언 방식


```
// 사용
var player1 = new Player(0, 0);
player1.moveLeft();
player1.display(); // -1,0: Player
```

인스턴스 생성

5) 클래스(ES6) 선언 방식


```
class Player {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  display() {
    console.log(this.x + “,” + this.y + “: Player”);
  }

  moveLeft() {
    this.x -= 1;
  }

  moveRight() {
    this.x += 1;
  }
} //class
```

1/2

- 문법이 Java와 유사하여 가독성이 높음
- 내부적으로는 여전히 프로토타입 기반임
- 생성자 함수보다 선언이 간결하고 직관적임

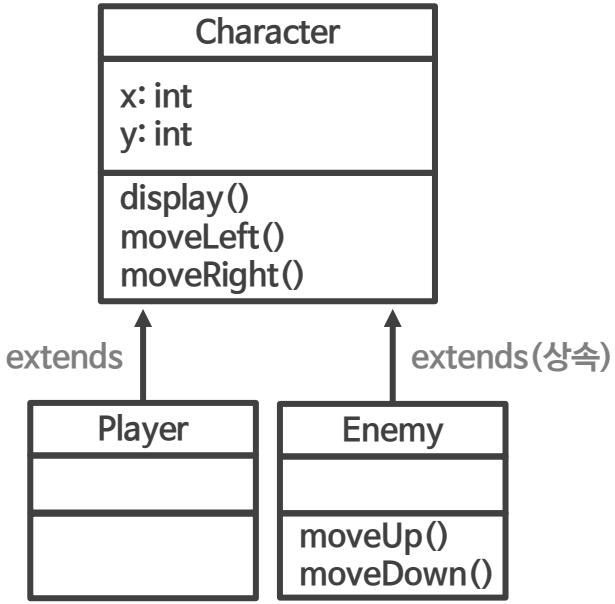
5) 클래스(ES6) 선언 방식


```
// 사용
const player1 = new Player(0, 0);
player1.moveRight();
player1.display(); // 1,0: Player
```

인스턴스 생성

6) 상속 기반 vs 프로토타입 기반

- Class 방식의 상속(JAVA의 경우)



JavaScript의 Class 선언 방식은 프로토타입 기반의 실행을 원칙으로 표현 방식만 Class(ES6)를 사용하는 것!!

6) 상속 기반 vs 프로토타입 기반

- 프로토타입 체인

- 자신의 부모 역할을 담당하는 객체와 연결되어 있음
- 객체지향의 상속 개념과 같이 부모 객체의 프로퍼티 또는 메소드를 공유받아 사용할 수 있게 함

'프로토타입(Prototype) 객체' 또는
줄여서 '프로토타입(Prototype)'이라 함

프로토타입(Prototype) 객체는 생성자 함수에 의해 생성된
모든 객체가 함께 쓸 수 있는 속성과 기능을 담아두는 공간

6) 상속 기반 vs 프로토타입 기반

프로토타입 체인

프로토타입 체인 형성

- 프로퍼티/메서드 탐색 ➡ 없으면 [[Prototype]] 따라 상위 프로토타입 ➡ 최상위 Object.prototype까지 검색

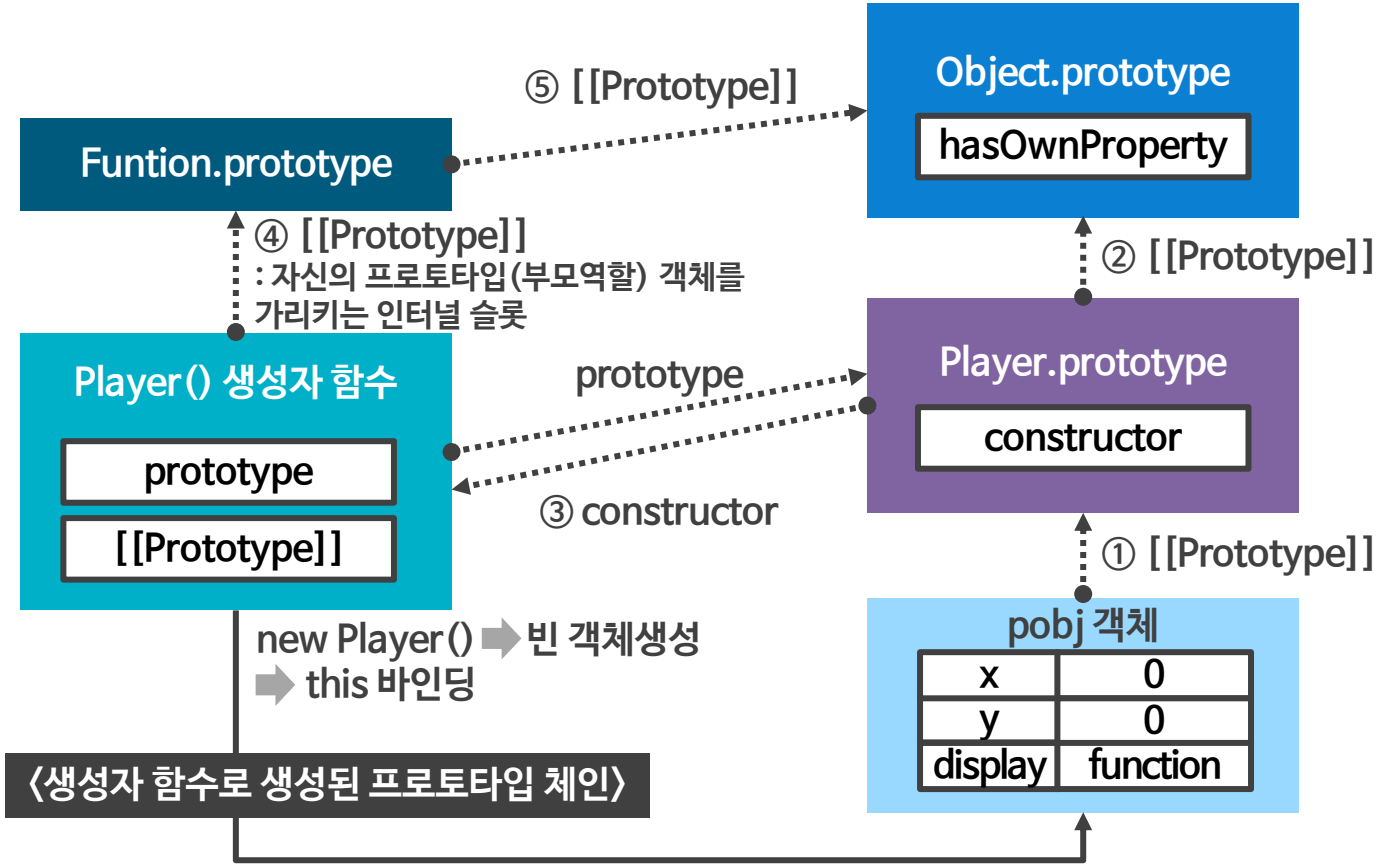
[[Prototype]] 인터널 슬롯(internal slot)

- 자신의 프로토타입(부모 역할) 객체를 가리키는 슬롯

프로토타입 객체는 constructor 프로퍼티를 가짐

constructor 프로퍼티	객체의 입장에서 자신을 생성한 객체

6) 상속 기반 vs 프로토타입 기반



7) 객체 정의 방법 비교

구분	객체 리터럴	ES5 생성자 + 프로토타입	ES6 클래스 선언문
객체생성방식	{} 사용 단일 객체만 생성	new 생성자 사용	class + constructor 사용

문법	const obj = { name: “Alice”, age: 25 };	function Person(name, age) {
		this.name = name;
		this.age = age;
		}
		const p1 = new Person(“Alice”, 25);
			class Person {
			constructor(name age) {
			this.name = name;
			this.age = age;
			}
			}
			const p2 = new Person(“Bob”, 30);

메모리 효율	낮음 (메소드가 매번 새로 생성)	높음 (공통 메소드는 공유)	높음 (내부적으로 프로토타입 활용)

8) 객체의 속성 접근과 동적 속성 접근


```
const user = {
  name: “Alice”,
  age: 25
};

// 1. 점 표기법(Dot notation)
console.log(user.name); // “Alice”

// 2. 대괄호 표기법(Bracket notation)
console.log(user[“age”]); // 25
```

user.name 또는 user[“name”]은 모두 “Alice”를 반환

8) 객체의 속성 접근과 동적 속성 접근

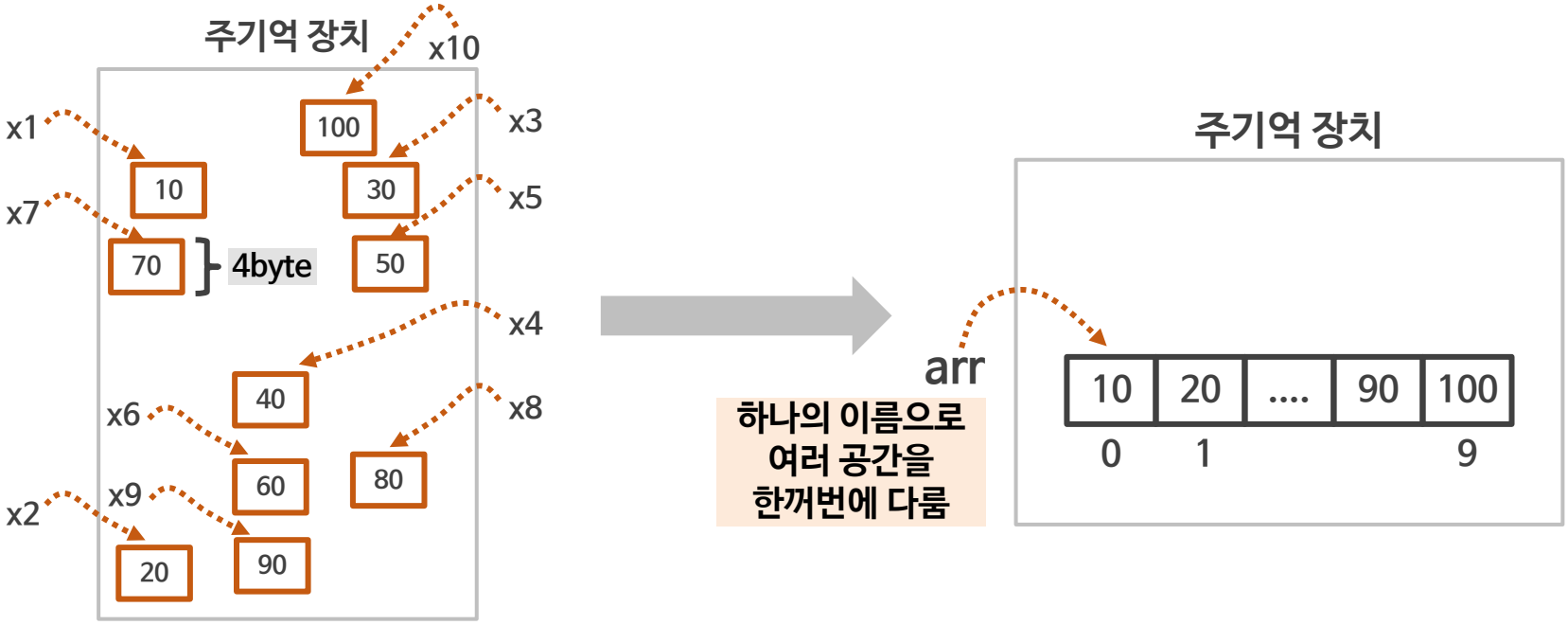

```
function printUserInfo(user, propName) {
  console.log( user[propName] );
  // 대괄호 표기법 : propName이 변수이기 때문
}

printUserInfo(user, "name" );  // "Alice"
printUserInfo(user, "age" ); // 25
```

- 점 표기법은 정적 접근(속성 이름이 고정일 때)
- 대괄호 표기법은 동적 접근(변수로 속성 이름을 넣을 때) 유용

03

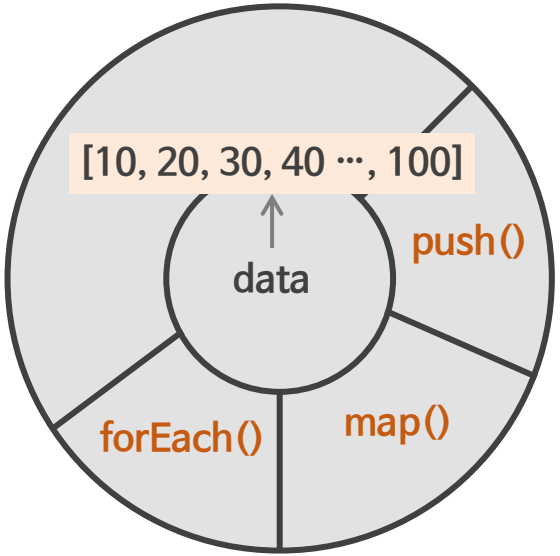
배열 객체의 활용

“여러 개의 데이터를 묶어
하나의 이름(주소)으로 관리하는 자료관리 기술”

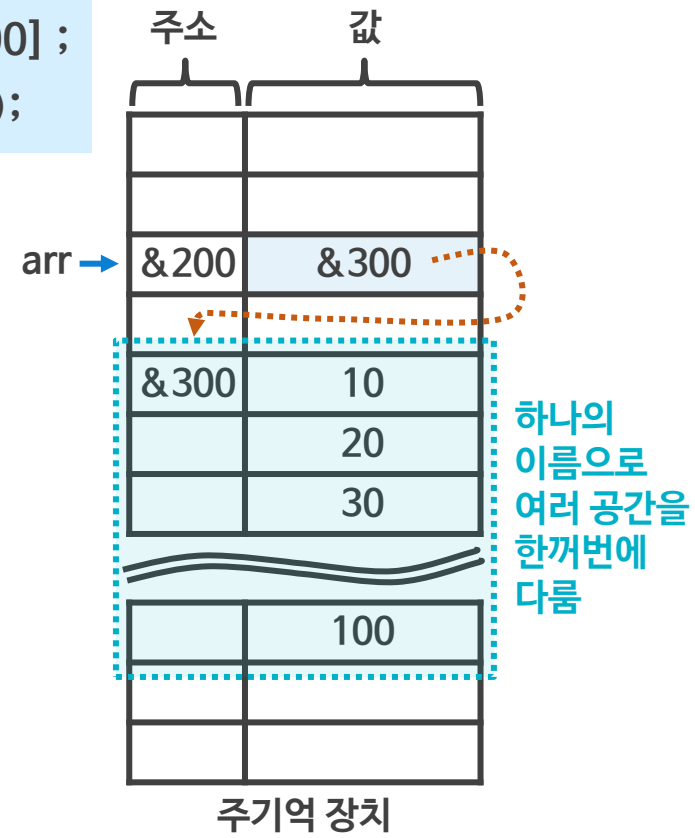


2) 배열 객체


```
var  arr = [ 10, 20, 30, 40 …, 100] ;
var  arr = new Array(10,20,30);
```



배열 객체 = arr




&시작주소
&시작주소+1
…
&시작주소+8
&시작주소+9
arr
10
20
…
90
100
0
1
…
8
9
&시작주소
&시작주소로부터 떨어진 상대적 거리 = offset
시작 주소 번지로부터 0만큼 떨어진 위치=arr[0]

2) 배열 객체

- 생성과 구조
  - 배열 생성(권장)

```
const fruits = [“apple”, “banana”, “cherry”];   //배열 생성
```

  - 배열은 사실상 객체이지만 length와 index 기반으로 작동

```
//배열 참조
Array.isArray(fruits); // true
typeof fruits; // ‘object’
```

3) 자주 사용하는 배열 메소드

메소드	기능 설명	예시 코드
push()	배열 끝에 요소 추가	arr.push("grape")
map()	요소 변형 (새 배열 반환)	arr.map(x => x * 2)
filter()	조건에 맞는 요소만 추출	arr.filter(x => x > 5)
forEach()	배열의 각 요소를 하나씩 꺼내서, 지정한 콜백 함수를 실행	arr.forEach(x => console.log(x))
reduce()	누적 계산	arr.reduce((a, b) => a + b, 0)

4) 반복문과 배열의 연계

- for, for...of, forEach() 비교

구문 종류	문법 형태	순회 대상	break, continue 가능	콜백 함수 필요	특징
for	for (let i = 0; i < n ; i++)	배열, 문자열 등	가능	불필요	인덱스 직접 제어 가능
for...of	for (const item of arr)	배열, 문자열, Set 등	가능	불필요	값 중심 순회, 간결
forEach()	arr.forEach((item) => {})	배열만	불가능	필요	함수형 스타일, 종료 불가 (return, break 안됨)

4) 반복문과 배열의 연계

for, for...of, forEach() 함께 배열을 사용하면 기능이 강력해짐

```
const arr = [1, 2, 3, 4];
// 기본 for문 - 탐색 시 index를 활용
for (let i = 0; i < arr.length; i++) {
  console.log( arr[i] );
}
// for...of  - 값을 직접 순회, 간결문법
for (let item of arr) {
  console.log(item);
}
// forEach - 배열 전용 메소드, 콜백 사용
arr.forEach( item =>
          console.log(item));
```

- 사용자 목록 배열 출력
- 점수 배열을 순회하여 합계 및 평균 계산

5) 배열 vs. 객체의 차이점

배열(Array)

- 순서가 중요할 때 사용
- 리스트, 컬렉션

객체(Object)

- 의미 있는 속성(key)을 저장할 때 사용
- 사용자 정보, 설정 값 등

구분	배열(Array)	객체(Object)
접근 방식	숫자 인덱스 기반	문자형 키(key) 기반
용도	순서가 있는 데이터 집합에 적합	속성(이름:값) 형태의 정보에 적합
예시	const arr = [10, 20, 30];	const obj = { name: “Alice”, age: 25 };
접근명령	arr[0] ➡ 10	obj.name 또는 obj[“name”] ➡ “Alice”
반복	for, forEach, map, filter 등	for...in, Object.keys() 등 사용

6) 다차원 배열과 객체 배열

- 객체 배열

```
const  users = [ // 배열의 요소가 객체들로 나열됨 = 객체배열
   { name: “Alice”, age: 25 },
   { name: “Bob”, age: 30 }
];

// 모든 이름 출력
users.forEach(user => {  // 각 user 객체에 대해 콜백 실행
   console.log(user.name);
});
// 결과: Alice, Bob
```

6) 다차원 배열과 객체 배열

- 다차원 배열

```
// 배열의 요소로 또 다른 배열을 포함하는 배열
const matrix = [
  [1, 2, 3],
  [4, 5, 6]
];

console.log( matrix[0][1] ); // 2
```

세종사이버대학교

06주. 유지보수 비용을 줄이자! 객체지향 이야기

03

객체지향 프로그래밍 실습