🚀ES6 (ECMAScript6)

ES6는 아래의 새로운 기능들을 포함하고 있다.

- const and let
- Arrow functions(화살표 함수)
- Template Literals(템플릿 리터럴)
- Default parameters(기본 매개 변수)
- Array and object destructing(배열 및 객체 비구조화)
- Import and export(가져오기 및 내보내기)
- Promises(프로미스)
- Rest parameter and Spread operator(나머지 매개 변수 및 확산 연산자)
- Classes(클래스)

📌 01_템플릿 문자열

```
<title>
 ES6 : 템플릿 문자열 = 문자열안에 변수와 연산식을 혼합하여 사용
</title>
```

```
 <script>
  // ES5 (옛날 방식) : + 연산자로 연결
  var str1 = "안녕하세요";
  var str2 = "반갑습니다";
  var greeting = str1 + " " + str2;
  document.write(greeting, "<br>");

  // 객체 데이터
  var product = { name: '도서', price: '4200원' };
  var message = '제품 ' + product.name + '의 가격은 ' + product.price + '입
니다.';
  document.write(message + "<br>");

  // 숫자값
  var val1 = 1, val2 = 2;
  var op1 = '곱셈값은 ' + (val1 * val2) + ' 입니다.';
  document.write(op1, '<br>');

  // 논리값
  var boolVal = false;
  var op2 = '논리값은 ' + (boolVal ? '참' : '거짓') + ' 입니다.';
  document.write(op2, '<br>');

  //document.write('<br><br>==========================<br><br>');

  // ES6 (새 방식) : 백틱(`) + ${변수/식}
  const s1 = "ES6 안녕하세요";
  const s2 = "ES6 반갑습니다.";
  const greeting2 = `${s1} ${s2}`;
  document.write(greeting2, "<br>");

  // 객체 데이터
  const product2 = { name: '도서', price: 4200 };
  const message2 = `제품 ${product2.name}의 가격은 ${product2.price.toL
ocaleString()}원 입니다.`;
  document.write(message2 + '<br>');

  // 숫자값
```

```
    const v1 = 1, v2 = 2;
    const op3 = `곱셈값은 ${v1 * v2} 입니다.`;
    document.write(op3, "<br>");

    // 논리값
    const boolVal2 = false;
    const op4 = `논리값은 ${boolVal2 ? '참' : '거짓'} 입니다.`;
    document.write(op4, '<br>');
</script>
```

1. const (불변 변수)

- **값 변경 불가**
- 숫자, 문자열, 논리형 → 새로운 값 재할당 ❌
- 객체/배열 → **참조 주소는 고정**, 내부 속성/요소는 수정 가능 (하지만 권장 ❌ → 무결성 깨짐)

---

2. let (가변 변수)

- 값 재할당 가능
- 반복문, 조건문 등에서 많이 사용
- **블록 스코프**(중괄호 `{}` 안에서만 유효)

---

3. var (옛날 방식)

- ES6 이전 변수 선언 방법
- **함수 스코프**만 적용 (블록 `{}` 무시)
- 재선언 가능 → 버그 발생하기 쉬움 (권장 ❌)

📌 `var` vs `let` 차이점

1. 스코프 (Scope: 유효 범위)

- `var` → **함수 스코프 (function scope)**
  - 함수 안에서 선언하면 함수 전체에서 유효
  - 블록(`{}`) 안에서 선언해도 블록 밖에서 접근 가능

- `let` → **블록 스코프 (block scope)**
  - `{}` 블록 안에서만 유효
  - 블록 밖에서는 접근 불가

```
if (true) {
  var a = 10;
  let b = 20;
}
console.log(a); // ✅ 10  - 스코프 상관없이 찾음 - 디버깅 발생시킴
console.log(b); // ❌ 못찾음 - 스코프 밖에서 찾지못함  - 디버깅 최소화
```

---

2. 호이스팅 (Hoisting)

- `var` → 선언이 끌어올려지지만, 값은 `undefined`로 초기화됨
- `let` → 선언만 끌어올려지고, **초기화 되기 전까지 "TDZ(Temporal Dead Zone)"** 때문에 접근 불가

```
console.log(x); // ✅ undefined (호이스팅됨)
var x = 5;

console.log(y); // ❌ ReferenceError (TDZ)
let y = 5;
```

---

3. 재선언 가능 여부

- `var` → 같은 스코프 안에서 **재선언 가능** (버그 발생 위험)
- `let` → 같은 스코프 안에서 **재선언 불가**

```
var a = 1;
var a = 2; // ✅ 가능

let b = 1;
let b = 2; // ❌ SyntaxError
```

---

4. 전역 객체(window)와의 관계

- `var` 로 전역 선언 시 → `window` 객체의 속성이 됨
- `let` 은 그렇지 않음

```
var x = 100;
let y = 200;

console.log(window.x); // ✅ 100
console.log(window.y); // ❌ undefined
```

✅ 정리

구분	var	let
스코프	함수 스코프	블록 스코프
호이스팅	됨(초기값 undefined)	됨(초기값 없음, TDZ 발생)
재선언 가능 여부	가능	불가능
전역 객체 연결	연결됨(window.x)	연결 안 됨

🧊 TDZ란?
변수가 선언되었지만 아직 초기화 되지 않는 상태를 말 한다.
쉽게 말 해 '선언만 되고 아직 초기화 되지 않는 변수가 머무는 공간'이라고 생각하면 된다.

JS에서 'let'이나 'const'로 선언한 변수들이 TDZ을 거쳐 간다.
이 공간에 있는 변수를 참조하려고 하면 'ReferenceError'를 마주할 것이다.
그럼 TDZ가 왜 필요한데?
TDZ의 주요 목적은 프로그래밍 오류를 줄이는데 있다.
대표적으로 초기화 되지 않는 변수를 사용하는 것을 방지할 수 있다.

👉 그래서 **ES6 이후부터는 `let` (혹은 `const`)을 쓰는 게 권장된다.**

✅ ES6 템플릿 리터럴 (Template Literals)

1. 백틱(`) 사용

- 문자열을 `''` , `""` 대신 **백틱(`)** 으로 감쌈

```
const str = `Hello ES6`;
```

---

2. 보간법 (Interpolation)

- `${변수}` , `${식}` 을 문자열 안에 바로 넣을 수 있음

```
const name = "철수";
const age = 10;
console.log(`이름: ${name}, 나이: ${age + 1}`);
```

---

3. 여러 줄 문자열

- 줄바꿈(`\n`) 안 써도, 줄을 바꾸면 그대로 반영됨

```
const poem = `
장미는 붉고
하늘은 푸르다
ES6 짱!
`;
```

---

4. 표현식/함수 실행 가능

- 단순 변수뿐만 아니라 **계산식, 함수 호출**도 가능

```
const a = 5, b = 3;
console.log(`합: ${a + b}, 랜덤: ${Math.random()}`);
```

📌02_가변변수불변변수

```
<title>
  ES6 가변변수 / 불변변수
  : 기본 자바스크립트 문법 변수 선언 = var 키워드 사용
```

```
        ES6는 값을 수정할 수 있는 가변변수 = let 키워드 사용
        ES6는 값을 수정할 수 없는 불변변수 = const 키워드 사용
    </title>
    <script>
        /* 1. const 불변변수 = 변수값 못 바꿈
            (참조) const 선언 변수값을 JS의 내장함수로 값을 변경하는 방법도 있음
            => 무결성 유지를 위해 배열이나 객체의 내장함수로 수정하는 건 안하는게 좋
음 */
        // a. 숫자형 데이터
         const num = 1;
        num = 3;
        console.log(num);

        // b. 문자열 데이터
         const str = '김쌤';
         str = '엠씨쌤';
         console.log(str);

        // c. 논리형 데이터
         const bool = true;
         bool = false;
         console.log(bool);

        // d. 객체 데이터
         const obj = {};
         obj = {name:'me'};
         console.log(obj);

        /* 2. let 가변변수 */
        // 배열 array에 1,2,3값 할당
        const array = [1,2,3];

        // a. for문 = 모든 배열요소 출력
         for(let i=0; i<array.length; i++){
            document.write(array[i],"<br>");
          }
        // b. for-in 루프 = iterator방식
        for(const item in array){
```

```
        // console.log('for-in구문' + item); // 인덱스
        console.log('for-in구문' + array[item]); // 요소
    }
    // c. forEach(화살표 함수사용)
    array.forEach((item,index)⇒{
        // console.log('배열 index값 : ' + index);
        // console.log('배열 item값 : ' + item);
        // 템플릿 문자열로 코딩
        console.log(`배열 index값 = ${index} 배열 item 값 = ${item}`);
    })

        /* (참고) var과 let 스코프(범위)
    #1. 스코프 = 적용범위
      ✔ var: 함수 스코프 블록({}를 무시함)
      ✔ let: 블록 스코프 = {}안에서만 동작 */
    if(true) {
        var v = 1;
        let l = 2;
    }
    console.log('var 스코프 => ' + v);
    console.log('let 스코프 => ' + l);
</script>
```

✅ 반복문 예시 (array = [1,2,3])

- **for문**: 전통적인 방식 (인덱스 기반)
- **for-in**: 배열의 인덱스를 순회
- **forEach**: 배열 요소 + 인덱스 함께 다룸 (화살표 함수와 자주 사용)

```
// for-in
for(const item in array){
  console.log(array[item]); // 요소
}

// forEach
array.forEach((item,index)⇒{
```

```
  console.log(`index=${index}, item=${item}`);
})
```

📌 정리

- **const** : 불변 변수 (재할당 ❌, 객체/배열 내부는 변경 가능)
- **let** : 가변 변수 (재할당 가능, 블록 스코프)
- **var** : 옛날 방식 (재선언 가능, 함수 스코프 → 버그 유발 위험)

👉 실무에서는 **기본은 const**, 변경 필요할 때만 **let**, `var`는 거의 쓰지 않음.

📌 03_화살표함수

```
  <title>
      ES6 화살표 함수(arrow function)
      : ES6에 추가된 표현식을 사용하는 함수
       화살표 => 로 함수선언
       기본적으로 익명함수로 선언, 기존 익명함수 변수에 할당하여 사용하는 방법과 유사
       function 키워드 생략
       인자블록(), 본문블록{} 사이에 =>를 표기
       기본 표기법: (인자)=>{실행문;}
  </title>
  <script>
      /* 1. ES5 */
      /*  1-a. 선언적 함수 = add , (기능) 값2개 더하는 기능 => (경고창)결괏값 출력 = 캡쳐, 넘버링 */
      function add(first,second){
          return first + second;
      }
      alert(add(1,2));

      /*  1-b.  익명함수 = add , (기능) 값2개 더하는 기능 => (경고창)결괏값 출력 = 캡쳐, 넘버링 */
      var add = function(first,second){
          return first + second;
      }
```

```
    alert(add(1,2));

    /* 2. ES6 */
    /* 2-a. 화살표 함수 => 기본 사용법(매개변수가 있는 경우) */
    var add = (first,second)⇒{
      return first + second;
    }
    alert(add(3,3));
    /*  2-b. 화살표 함수 => 기본 사용법(매개변수가 없는 경우) */
    var add = ()⇒{
      let first = 1;
      let second = 2;
      return first + second;
    }
    alert(add());

    /* 2-c. 화살표 함수 => 본문 블록을 생략할 수 있는 경우 = 결괏값을 바로 반환
하는 경우 */
    var add = (first,second)⇒ first + second;
    alert(add(10,10));

    /* 2-d. 화살표 함수 => 반환값이 객체면 ()로 감싸서 간결하게 표현 가능 */
    var add = (first,second) ⇒ ({add:first + second, multi: first * second});
    alert(add(10,10).multi);
  </script>
```

✅ ES6 화살표 함수 (Arrow Function)

1. 문법

```
(매개변수) => { 실행문 }
```

- `function` 키워드 대신 `=>` 사용
- 기본적으로 **익명 함수**
- 변수에 할당해서 사용

---

2. 주요 형태

1. **일반 형태**

```
const add = (a, b) => {
  return a + b;
}
```

1. **매개변수 없음**

```
const hello = () => {
  return "안녕!";
}
```

1. **본문 블록 생략 (즉시 반환)**

```
const add = (a, b) => a + b;
```

1. **객체 반환 시 () 필요**

```
const makeUser = (name, age) => ({ name, age });
```

---

3. 장점

- 코드가 짧아짐 (간결성)
- 가독성 ↑

---

4. 주의점 (ES5 함수와 차이)

- **this 바인딩이 없음**
  - 화살표 함수 내부에서 `this`는 **자신을 둘러싼 상위 스코프**의 `this`를 사용
  - 그래서 이벤트 핸들러, 객체 메서드 정의에 직접 쓰면 의도와 다를 수 있음
- **arguments 객체 없음**
  - 필요하면 `...rest`(나머지 매개변수) 사용

📌 정리

- **표현식 기반 함수:** `()=>{}`
- **간단한 리턴:** `{}`와 `return` 생략 가능
- **객체 리턴:** `({})` 괄호 필요
- **this/arguments 없음** → 상위 스코프 참조

👉 실무에서는 **콜백 함수, 배열 메서드(map, forEach, filter 등)**에 가장 많이 사용

📌04_구조분해할당

```
<title>
    ES6 구조분해할당(destructuring)
    : 객체나 배열의 특정값을 손쉽게 추출할 수 있는 표현식
      = 객체나 배열의 속성을 해체하여, 그 값을 변수에 담을수 있게 하는 JS 표현식
      = 특히 JSON(JavaScriptObjectNotation)데이터 처리시에 유용함
  </title>
  <script>
    /* 1. 배열(Array) 구조분해할당 */
    // a. 기본형식
    const kind = ['shoes', 'pants', 'skirt']; // 배열데이터
    const [shoes,pants,skirt] = kind; // 구조분해할당
    console.log(shoes,pants,skirt); // 데이터 출력
    // b. 일부 값 사용
    const color = ['red','green','blue'];
    const [first, ,third] = color;
    console.log(first);
    console.log(third);
    // c. Rest 파라미터
    const [a,...rest] = [1,2,3,4,5];
    console.log(a);
    console.log(rest);

    /* 2. 객체(Object) 구조분해할당 */
    // a. 기본형식
    const user = { name: 'mcssam', age:33, academy:'Ezen'};
    const {name, age, academy} = user;
    console.log(name, age, academy);
    // b. 키명 변경
```

```
    const academyName = {first: "Ezen", last: "Computer"};
    const { first: rFirst, last: lLast} = academyName;
    console.log(rFirst, lLast);
    // c. 기본값 설정
    const setting = { backColor: 'silver'};
    const {backColor, colorFont = 'blue'} = setting;
    console.log(backColor, colorFont);

    /* 3. 함수 구조분해할당 */
    function intro({name, area}){
       console.log(`안녕하세요 ${name} 입니다. 사는곳은 ${area} 입니다.`);
    }
    const person = {name: "김명철", area:'성수동'};
    intro(person);

    /* 4. JSON 데이터에서 사용 */
    const jsonData = `{
       "title": "ES6",
       "computed": false,
       "user": {
          "name2": "gimssam",
          "email": "indopop@naver.com"
       }
    }`;
    console.log(jsonData);
    // JSON 문자열을 자바스크립트 객체로 변환
    const task = JSON.parse(jsonData);
    const {title, computed, user:{name2,email},} = task;
    console.log(title);
    console.log(computed);
    console.log(name2);
    console.log(email);
</script>
```

✅ 구조분해할당 (Destructuring Assignment)

> 배열이나 객체 안의 값을 해체(Destructuring)해서 변수로 쉽게 꺼내 쓰는 문법

> JSON 처리 시 특히 유용

1. 배열 구조분해할당

```
const kind = ['shoes', 'pants', 'skirt'];
const [shoes, pants, skirt] = kind;
```

- 배열 요소를 순서대로 변수에 할당

응용

- 일부 값만 추출

```
const color = ['red','green','blue'];
const [first, , third] = color; // 중간 건 건너뜀
```

- 나머지(Rest) 모으기

```
const [a, ...rest] = [1,2,3,4,5];
// a=1, rest=[2,3,4,5]
```

2. 객체 구조분해할당

```
const user = { name: 'mcssam', age:33, academy:'Ezen'};
const { name, age, academy } = user;
```

- 객체의 key 이름과 같은 변수명으로 꺼냄

응용

- 키 이름 변경

```
const academyName = { first: "Ezen", last: "Computer" };
const { first: rFirst, last: lLast } = academyName;
// rFirst = "Ezen", lLast = "Computer"
```

- 기본값 설정

```
const setting = { backColor: 'silver' };
const { backColor, colorFont = 'blue' } = setting;
// colorFont 없으면 기본값 "blue"
```

---

3. 함수 매개변수에서 사용

```
function intro({ name, area }) {
  console.log(`안녕하세요 ${name} 입니다. ${area}에 살아요.`);
}
const person = { name: "김명철", area: "성수동" };
intro(person);
```

- 함수 파라미터에서 바로 분해해서 사용 가능

---

4. JSON 데이터에서 사용

```
const jsonData = `{
  "title": "ES6",
  "computed": false,
  "user": { "name2": "gimssam", "email": "indopop@naver.com" }
}`;
const task = JSON.parse(jsonData);
const { title, computed, user: { name2, email } } = task;
```

- JSON 문자열 → 객체 변환 후 필요한 값만 분해

📌 정리

- **배열**: 순서대로 값 추출, Rest(...) 가능
- **객체**: key 이름으로 추출, 이름 변경·기본값 가능
- **함수**: 매개변수 구조분해 바로 사용 가능
- **JSON**: 파싱 후 원하는 값만 꺼내 쓰기 좋음

👉 구조분해할당은 **코드를 짧고 가독성 있게** 만들어주는 ES6의 핵심 기능.

📌 05_스프레드연산자

```
<title>
    ES6 스프레드(...) 연산자 = 풀기 연산자
</title>
<script>
    // Array
    let arr1 = [1,2,3,4,5];
    console.log(arr1);
    let arr2 = [...arr1,6,7,8,9,10];
    console.log(arr2);

    // String
    let str1 = 'react app';
    console.log(str1);
    let str2 = [...str1];
    console.log(str2);
</script>
```

✅ 스프레드(...) 연산자

> 배열이나 객체, 문자열 등 이터러블(반복 가능한 값)을 하나씩 풀어서 전개하는 연산자

---

1. 배열에서 사용

```
let arr1 = [1,2,3,4,5];
let arr2 = [...arr1, 6,7,8,9,10];
```

- `arr1`의 요소가 전부 풀려서 새로운 배열 `arr2`에 들어감
- 결과: `[1,2,3,4,5,6,7,8,9,10]`

👉 **복사 / 병합**할 때 많이 사용

---

2. 문자열에서 사용

```
let str1 = 'react app';
let str2 = [...str1];
```

- 문자열을 문자 단위로 풀어서 배열로 변환
- 결과: `["r","e","a","c","t"," ","a","p","p"]`

📌 정리

- **스프레드(...)** = "풀기" 연산자
- 배열 → 새 배열에 풀기 (복사, 병합)
- 문자열 → 문자 하나하나 분리
- 객체에도 사용 가능 → `{...obj}` (속성 복사/병합)

👉 실무에서는 **배열 합치기, 객체 복사, 함수 인자 전달** 등에 가장 많이 활용됨.

✅ 오늘 배운 ES6 문법 종합 정리 (한 줄 요약)

- **템플릿 리터럴** → 백틱 `` `${}` `` 로 변수·연산식 삽입, 여러 줄 문자열 가능
- **let** → 가변 변수, 재할당 가능, 블록 스코프
- **const** → 불변 변수, 재할당 불가, 블록 스코프 (객체/배열 내부는 수정 가능)
- **화살표 함수** → `()=>{}` 간단한 함수 표현식, `this/arguments` 없음, 한 줄 반환 시 `return` 생략 가능
- **구조분해할당** → 배열·객체 값 쉽게 추출, 이름 변경·기본값 설정·함수 매개변수에서도 활용 가능
- **스프레드 연산자(...)** → 배열/문자열/객체 요소를 풀어서 복사·병합·전달

종합 예제 코드

```
<!DOCTYPE html>
<html lang="ko">
<head>
```

```
  <meta charset="UTF-8">
  <title>ES6 종합 예제</title>
</head>
<body>
  <script>
    /* 1. let / const */
    let counter = 0;         // 가변변수 : 값 바꿀 수 있음
    const PI = 3.14;         // 불변변수 : 값 바꿀 수 없음

    counter = 5;             // 가능
    // PI = 3.1415;          // ❌ 오류 (const는 재할당 불가)
    console.log(`counter=${counter}, PI=${PI}`);

    /* 2. 템플릿 리터럴 */
    const name = "현수";
    const age = 33;
    console.log(`안녕하세요, 저는 ${name}이고 내년엔 ${age + 1}살이 됩니다.`);

    /* 3. 화살표 함수 */
    const add = (a, b) => a + b;        // 간단한 덧셈 함수
    const greet = () => "반가워요!";     // 매개변수 없는 함수
    console.log(add(2, 3));   // 5
    console.log(greet());     // 반가워요!

    /* 4. 구조분해할당 */
    const colors = ["red", "green", "blue"];
    const [first, , third] = colors;     // 배열 구조분해
    console.log(`첫 번째 색상=${first}, 세 번째 색상=${third}`);

    const user = { id: 1, nick: "코알라" };
    const { id, nick } = user;          // 객체 구조분해
    console.log(`id=${id}, nick=${nick}`);

    /* 5. 스프레드 연산자 */
    const arr1 = [1, 2, 3];
    const arr2 = [...arr1, 4, 5];       // 배열 풀어서 병합
    console.log(arr2);                  // [1,2,3,4,5]
```

```
    const str = "ES6";
    const chars = [...str];          // 문자열 풀어서 배열
    console.log(chars);               // ["E","S","6"]

    /* 6. 템플릿 리터럴 + 구조분해 + 함수 결합 */
    const intro = ({ name, area }) => `안녕하세요 ${name}, ${area}에 살아요!`;
    const person = { name: "김현수", area: "노원구" };
    console.log(intro(person));
  </script>
</body>
</html>
```

📌 각 문법 요약 & 설명

1. **let / const**
   - `let` → 값 바꿀 수 있음 (가변)
   - `const` → 값 못 바꿈 (불변), 단 객체/배열 내부는 수정 가능
2. **템플릿 리터럴**
   - 백틱(`) 사용
   - `${}` 안에 변수·계산식 넣을 수 있음
   - 여러 줄 문자열 그대로 표현 가능
3. **화살표 함수**
   - `()=>{}` 형식
   - `function` 키워드 대신 간결하게 작성
   - 한 줄 반환 시 `{}`와 `return` 생략 가능
4. **구조분해할당**
   - 배열/객체 안의 값을 쉽게 꺼내 변수에 할당
   - `[a,b] = arr` / `{x,y} = obj`
5. **스프레드 연산자(...)**
   - 배열, 문자열, 객체의 요소를 "풀어서" 복사·병합

- `[...arr, 4,5]` / `{...obj, age:20}`