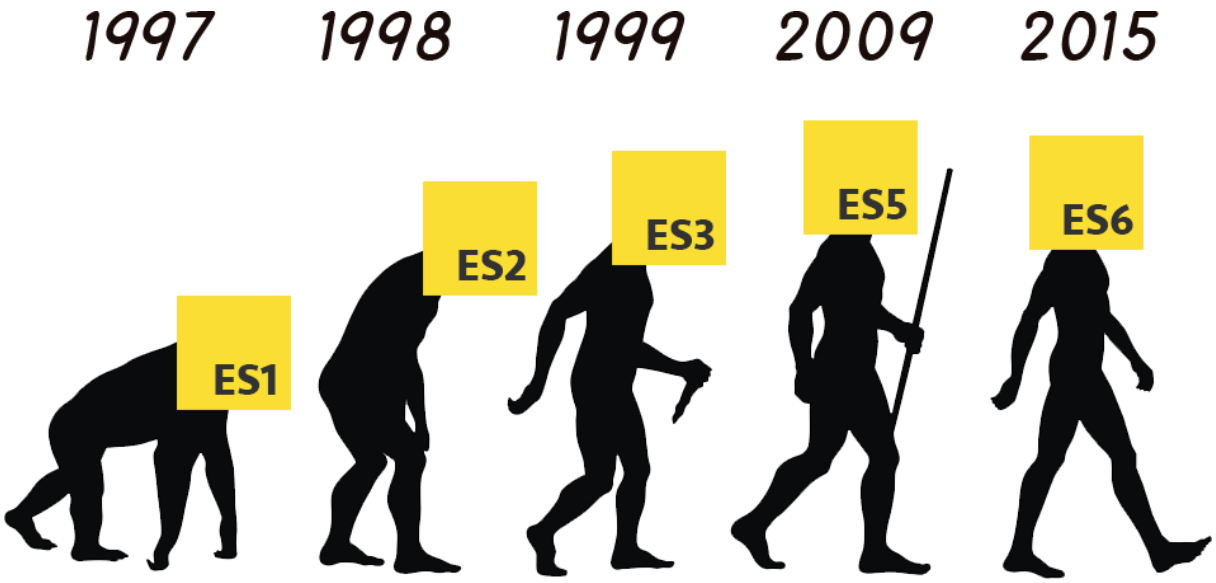
ECMAScript – 표준화된 자바스크립트



0. ES6란?

ES6(ECMAScript 6)는 ECMAScript 표준의 가장 최신 버전이다.

현대적인 코드를 사용하면, 코드가 간결해지고 생산성이 향상된다. 이것이 ES6를 사용해야 하는 이유다.

1. 변수 선언 방식의 차이 (var, let, const)

- var은 재할당과 재선언 모두 가능하다.
- let은 가변변수로 재할당은 가능하지만 재선언은 불가능하다.
- const는 불변변수로 재할당과 재선언 모두 불가능하다.

* 재선언: 똑같은 이름의 변수를 다시 만드는 것
* 재할당: 값이 지정된 변수에 값을 바꾸려는 것
* 스코프(scope) : 식별자(ex. 변수명, 함수명, 클래스명 등)의 유효범위

```
//변수 선언
var x = 2;
// 재정의
x = 4;
// 재선언
var x = 4;
```

var의 문제점

① 변수 선언이 유연하기 때문에 예기치 못한 값을 반환할 수 있다.

② 코드가 길어진다면 어디에서 어떻게 사용될지 파악하기 힘들다.

③ 함수 레벨 스코프로 인해 함수 외부에서 선언한 변수는 모두 전역 변수로 된다.
④ 변수 선언문 이전에 변수를 참조하면 언제나 undefined를 반환한다. (호이스팅 발생)
→ 따라서, var 보다는 let과 const 키워드를 사용하는 것을 권장한다.

2. 템플릿 리터럴 (Template literals)

템플릿 리터럴은 ES6부터 새로 도입된 문자열 표기법으로, 문자열 생성시 따옴표 대신, 백틱(`)을 사용한다. 따옴표와 달리 백틱 안에서는 줄바꿈이 반영된다. 또한, 문자열 사이에 변수나 연산을 넣을 때는 ${} 사이에 표현식을 삽입한다.

```
var js = "자바스크립트";

// 기존 코드
console.log("이건 " + js + "입니다.");

// 템플릿 리터럴 방식
console.log(`이건 ${js}입니다.`);

// 출력 결과 -> 이건 자바스크립트입니다.
```

3. 화살표 함수 (Arrow function)

화살표 함수는 함수 표현식을 보다 단순하고 간결한 작성하는 문법이다.

```
// 기본 함수 형식
let sum = function(a, b) {
  return a + b;
};

// 화살표 함수 형식
let sum = (a, b) => a + b;
```

인수가 하나밖에 없다면 인수를 감싸는 괄호를 생략할 수 있다.
인수가 하나도 없을 땐 괄호를 비워 놓는다. 이 때 괄호는 생략할 수 없다.
본문이 한 줄 밖에 없다면 중괄호를 생략할 수 있다.
중괄호는 본문 여러 줄로 구성되어 있음을 알려주며, 중괄호를 사용했다면 return으로 결괏값을 반환해주어야 한다.

4. 모듈 내보내고 가져오기 (export/import)

모듈을 내보내는 방법으로는 named export와 default export가 있다.

```
// named export 기본 형식
export { 모듈명1, 모듈명2 };
import { 모듈명1, 모듈명2 } from 'js 파일 경로';

// default export 기본 형식
export default 모듈명;
import 모듈명 from 'js 파일 경로';
```

- named export는 한 파일에서 여러 개를 export할 때 사용 가능하다. export한 이름과 동일한 이름으로 import해야 하며, 중괄호에 묶어서 import 해야 한다.
다른 이름으로 import 하려면 as를 사용하고, 한 파일에 있는 클래스나 변수들을 한 번에 import 하려면 * as를 사용한다.

```
// named export는 중괄호 포함 import
import { named1, named2 } from './example.js';

// named export에서 as를 사용하여 다른 이름으로 import
import { named1 as myExport, named2 } from './example.js';

// 한 파일에 있는 모든 클래스나 변수를 * as를 사용하여 한 번에 import
import * as Hello from './example.js';
```

- default export는 하나의 파일에서 단 하나의 변수 또는 클래스 등등만 export 할 수 있다. 또한 import 할 때 아무 이름으로나 자유롭게 import 가능하며, 중괄호에 묶지 않아도 된다.

```
// default export 는 중괄호 없이 import
import default1 from './example.js';
```

5. 클래스 (class)

Class는 객체를 생성하기 위한 템플릿으로, 틀과 같은 역할을 한다.

- 클래스를 선언하려면 class 키워드와 함께 클래스의 이름을 작성한다.
- 클래스는 함수로 호출될 수 없다.
- 클래스 선언은 let과 const처럼 블록 스코프에 선언되며, 호이스팅(hoisting)이 일어나지

않는다. 클래스는 반드시 정의한 뒤에 사용한다.

- 클래스의 메소드 안에서 super 키워드를 사용할 수 있다.
- static 키워드를 메소드 이름 앞에 붙여주면 해당 메소드는 정적 메소드가 됩니다.
- Getter 혹은 Setter를 정의하고 싶을 때는 메소드 이름 앞에 get 또는 set을 붙여준다.
- extends 키워드를 사용하여 클래스에서 다른 클래스로 상속하면서 클래스의 기능을 확장할 수 있다.
- 클래스에서 일반적인 방식으로 프로퍼티를 선언하고 할당하면 Public Property(공개 프로퍼티)이다. → 외부에서 프로퍼티에 접근하여 값을 사용하거나 수정이 가능
- 클래스에서 프로퍼티 앞에 # 키워드를 작성하여 선언하면 Private Property (비공개 프로퍼티)가 된다. → 오직 클래스 안에서만 사용, 변경이 가능. 외부 접근 불가

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  nextYearAge() {   // 메서드 생성
    return Number(this.age) + 1;
  }
}

// 클래스 상속
class introducePerson extends Person {
  constructor(name, age, city, futureHope) {
    // super 키워드를 이용해서 자식 class에서 부모 메서드를 호출
    super(name, age, city);
    this.futureHope = futureHope
  }
  introduce () {
    return `저는 ${this.city}에 사는 ${this.name} 입니다.
        내년엔 ${super.nextYearAge()}살이며,
        장래희망은 ${this.futureHope} 입니다.`
  }
}
```

```
let kim = new introducePerson('kim','23','seoul', '개발자');
console.log(kim.introduce())
```

6. 구조분해할당 (destructing)

객체와 배열의 값을 쉽게 변수로 저장할 수 잇다.

객체에서 값을 꺼낼 때는 중괄호를 사용해서 key와 같은 이름으로 꺼내올 수 있고, key와 다른 이름으로 꺼낼 때는 변수이름: 키값으로 꺼내올 수 있다.

```
const introduce = {name: 'unico', age: 23};
// key와 같은 이름으로 변수 선언
const { name, age } = introduce;
// 다른 이름으로 변수 선언 -> 변수이름: 키값
const { name : myName, age : myAge } = introduce;

console.log(myName) // unico
console.log(myAge) // 23
//배열에서 값을 꺼낼 때는 대괄호를 사용해서 앞에서부터 순차적으로 꺼내올 수 있다.

const fruits = ['apple', 'mango', 'grape'];
// 앞에서부터 순차적으로 변수 선언 가능
const [zero, one, two] = fruits;

console.log(zero) // apple
```

7. Rest Operator / Spread Operator

- Rest Operator(나머지 매개변수)

Rest Operator(나머지 매개변수)는 나머지 후속 매개변수들을 묶어 하나의 배열에 저장해서 사용하는 것이다.
묶어줄 매개변수 앞에 ...을 붙여서 작성하면 된다.
즉, Rest Operator는 배열과 함수의 인자 중 나머지를 가리키며, 객체의 나머지 필드를 가리킨다.

```
// args에 1,2,3,4,5가 한꺼번에 배열로 담겨 인자로 넘겨진다.
function func1(...args) {
        console.log(`args: [${args}]`)
  // args: [1,2,3,4,5]
```

```
}
func1(1,2,3,4,5);

// arg1에는 1, arg2에는 2, arg3에는 나머지 3,4,5가 배열로 담겨 인자로 넘겨진다.
function func2(arg1, arg2, ...arg3) {
        console.log(`arg1: ${arg1}, arg2: ${arg2}, arg3: [${arg3}]`)
  // arg1: 1, arg2: 2, arg3: [3,4,5]
}
func2(1,2,3,4,5);
```

func(인자1, 인자2, ...인자들)로 넘겨주게 되면 인자1, 인자2처럼 지정된 인자는 앞에서부터 각각의 값을 넣어주고 그 뒤의 나머지 후속 인자들을 배열에 인자들로 묶어서 보내주는 것이다.

Rest Operator는 함수 정의에는 하나의 ...만 존재할 수 있으며, 반드시 마지막 매개변수여야 한다.

```
func(...wrong, arg2, arg3)
// 틀린 예. ...wrong이 마지막으로 와야 한다.
```

- Spread Operator(전개 구문)

Spread Operator(전개 구문)는 묶인 배열 혹은 객체를 개별적인 요소로 분리한다. 즉, Rest와 반대 개념이라고 생각하면 되고, 마찬가지로 전개할 매개변수 앞에 ...을 붙여서 작성하면 된다.
따라서, 배열과 함수에선 또 다른 배열과 함수의 인자로의 전개를, 객체에선 또 다른 객체로의 전개를 한다.

```
let arr = [1, 2, 3, 4, 5];
console.log(...arr);
// 1 2 3 4 5

var str = 'javascript';
console.log(...str);
// "j" "a" "v" "a" "s" "c" "r" "i" "p" "t"
```

Spread Operator도 Rest Operator와 마찬가지로 ...의 작성 순서에 주의해야 한다. 등장 순서에 따라, 덮어씌워 질 수 있기 때문이다.

```
var obj = { name: '짱구', species: 'human'};
obj = { ...obj, age: 5};
```

```
console.log(obj)
// {name: "짱구", species: "human", age: 5}

obj = { ...obj, name: '짱아', age: 1};

console.log(obj);
// {name: "짱아", species: "human", age: 11}
```

위 예제와 같이 ...obj가 먼저 나오고 name과 age가 나중에 등장함으로써 덮어씌워져 값이 변경된 것을 확인할 수 있다.

요약하자면, Rest Operator(나머지 매개변수)는 배열로 묶는 역할을,
Spread Operator(전개 구문)는 개별적인 요소로 분리하는 역할을 한다.
둘 다 "..." 을 붙여서 사용하며, 작성 순서에 주의해야 한다.

8. forEach() / map() / reduce()

forEach()와 map()은 반복문을 돌며 배열 안의 요소들을 1대1로 짝지어 주는 역할을 한다.

- forEach() : 배열 요소마다 한 번씩 주어진 함수(콜백)를 실행

```
배열.forEach((요소, 인덱스, 배열) => { return 요소 });
```

- map() : 배열 내의 모든 요소 각각에 대하여 주어진 함수(콜백)를 호출한 결과를 모아 새로운 배열을 반환

```
배열.map((요소, 인덱스, 배열) => { return 요소 });
```

하지만 forEach()와 map()은 역할은 같지만, 리턴값의 차이가 있다.
forEach()는 기존의 배열을 변경하는 반면, map()은 결과값으로 새로운 배열을 반환한다.

```
var arr = [1,2,3,4,5];
// forEach()
var newArr = arr.forEach(function(e, i) {
  return e;
});
// return -> undefined
```

```
// map()
var newArr = arr.map(function(v, i, arr) {
  return v + 1;
});
// return -> 2, 3, 4, 5, 6
```

reduce() : 배열의 각 요소를 순회하며 callback함수의 실행 값을 누적하여 하나의 결과값을 반환

```
배열.reduce((누적값, 현잿값, 인덱스, 요소) => { return 결과 }, 초깃값);

result = sum.reduce((prev, curr, i) => {
  console.log(prev, curr, i);
  return prev + curr;
}, 0);
// 0 1 0
// 1 2 1
// 3 3 2
result; // 6
```

* 초깃값을 적어주지 않으면 자동으로 초깃값이 0번째 인덱스의 값이 된다.
* reduce()는 초깃값을 배열로 만들고, 배열에 값들을 push하면 map과 같아진다.