매개변수 기본값, Rest 파라미터, Spread 문법, Rest/Spread 프로퍼티

매개변수 기본값 (Default Parameter value)

함수를 호출할 때 매개변수의 개수만큼 인수를 전달하는 것이 일반적이지만 그렇지 않은 경우에도 에러가 발생하지는 않는다. 함수는 매개변수의 개수와 인수의 개수를 체크하지 않는다. 인수가 부족한 경우, 매개변수의 값은 undefined이다.

```
function sum(x, y) {
  return x + y;
}

console.log(sum(1)); // NaN
```

따라서 매개변수에 적절한 인수가 전달되었는지 함수 내부에서 확인할 필요가 있다.

```
function sum(x, y) {
  // 매개변수의 값이 falsy value인 경우, 기본값을 할당한다.
  x = x || 0;
  y = y || 0;

  return x + y;
}

console.log(sum(1));    // 1
console.log(sum(1, 2)); // 3
```

ES6에서는 매개변수 기본값을 사용하여 함수 내에서 수행하던 인수 체크 및 초기화를 간소화할 수 있다. 매개변수 기본값은 매개변수에 인수를 전달하지 않았을 경우에만 유효하다.

```
function sum(x = 0, y = 0) {
  return x + y;
}

console.log(sum(1));    // 1
console.log(sum(1, 2)); // 3
```

매개변수 기본값은 함수 정의 시 선언한 매개변수 개수를 나타내는 함수 객체의 length 프로퍼티와 arguments 객체에 영향을 주지 않는다.

```
function foo(x, y = 0) {
  console.log(arguments);
}

console.log(foo.length); // 1

sum(1);    // Arguments { '0': 1 }
sum(1, 2); // Arguments { '0': 1, '1': 2 }
```

Rest 파라미터

기본 문법

Rest 파라미터(Rest Parameter, 나머지 매개변수)는 매개변수 이름 앞에 세개의 점 `...`을 붙여서 정의한 매개변수를 의미한다. Rest 파라미터는 함수에 전달된 인수들의 목록을 배열로 전달받는다.

```
function foo(...rest) {
  console.log(Array.isArray(rest)); // true
  console.log(rest); // [ 1, 2, 3, 4, 5 ]
}
```

```
foo(1, 2, 3, 4, 5);
```

함수에 전달된 인수들은 순차적으로 파라미터와 Rest 파라미터에 할당된다.

```
function foo(param, ...rest) {
  console.log(param); // 1
  console.log(rest);  // [ 2, 3, 4, 5 ]
}

foo(1, 2, 3, 4, 5);

function bar(param1, param2, ...rest) {
  console.log(param1); // 1
  console.log(param2); // 2
  console.log(rest);   // [ 3, 4, 5 ]
}

bar(1, 2, 3, 4, 5);
```

Rest 파라미터는 이름 그대로 먼저 선언된 파라미터에 할당된 인수를 제외한 나머지 인수들이 모두 배열에 담겨 할당된다. 따라서 Rest 파라미터는 반드시 마지막 파라미터이어야 한다.

```
function foo( ...rest, param1, param2) { }

foo(1, 2, 3, 4, 5);
// SyntaxError: Rest parameter must be last formal paramete
r
```

Rest 파라미터는 함수 정의 시 선언한 매개변수 개수를 나타내는 함수 객체의 length 프로퍼티에 영향을 주지 않는다.

```
function foo(...rest) {}
console.log(foo.length); // 0

function bar(x, ...rest) {}
console.log(bar.length); // 1

function baz(x, y, ...rest) {}
console.log(baz.length); // 2
```

arguments와 rest 파라미터

ES5에서는 인자의 개수를 사전에 알 수 없는 가변 인자 함수의 경우, arguments 객체를 통해 인수를 확인한다. arguments 객체는 함수 호출 시 전달된 인수(argument)들의 정보를 담고 있는 순회가능한(iterable) 유사 배열 객체(array-like object)이며 함수 내부에서 지역 변수처럼 사용할 수 있다.

arguments 프로퍼티는 현재 일부 브라우저에서 지원하고 있지만 ES3부터 표준에서 deprecated 되었다. Function.arguments와 같은 사용 방법은 권장되지 않으며 함수 내부에서 지역변수처럼 사용할 수 있는 arguments 객체를 참조하도록 한다.

```
// ES5
var foo = function () {
  console.log(arguments);
};

foo(1, 2); // { '0': 1, '1': 2 }
```

가변 인자 함수는 파라미터를 통해 인수를 전달받는 것이 불가능하므로 arguments 객체를 활용하여 인수를 전달받는다. 하지만 arguments 객체는 유사 배열 객체이므로 배열 메소드를 사용하려면 Function.prototype.call을 사용해야 하는 번거로움이 있다.

```
// ES5
function sum() {
  /*
  가변 인자 함수는 arguments 객체를 통해 인수를 전달받는다.
```

```
  유사 배열 객체인 arguments 객체를 배열로 변환한다.
  */
  var array = Array.prototype.slice.call(arguments);
  return array.reduce(function (pre, cur) {
    return pre + cur;
  });
}

console.log(sum(1, 2, 3, 4, 5)); // 15
```

ES6에서는 rest 파라미터를 사용하여 가변 인자의 목록을 **배열**로 전달받을 수 있다. 이를 통해 유사 배열인 arguments 객체를 배열로 변환하는 번거로움을 피할 수 있다.

```
// ES6
function sum(...args) {
  console.log(arguments); // Arguments(5) [1, 2, 3, 4, 5, c
allee: (...), Symbol(Symbol.iterator): f]
  console.log(Array.isArray(args)); // true
  return args.reduce((pre, cur) => pre + cur);
}
console.log(sum(1, 2, 3, 4, 5)); // 1
```

하지만 ES6의 화살표 함수에는 함수 객체의 arguments 프로퍼티가 없다. 따라서 화살표 함수로 가변 인자 함수를 구현해야 할 때는 반드시 rest 파라미터를 사용해야 한다.

```
var normalFunc = function () {};
console.log(normalFunc.hasOwnProperty('arguments')); // tru
e

const arrowFunc = () => {};
console.log(arrowFunc.hasOwnProperty('arguments')); // fals
e
```

Spread 문법

Spread 문법(Spread Syntax, `...`)는 대상을 개별 요소로 분리한다. Spread 문법의 대상은 이터러블이어야 한다.

```
// ...[1, 2, 3]는 [1, 2, 3]을 개별 요소로 분리한다(→ 1, 2, 3)
console.log(...[1, 2, 3]) // 1, 2, 3

// 문자열은 이터러블이다.
console.log(...'Hello');  // H e l l o

// Map과 Set은 이터러블이다.
console.log(...new Map([['a', '1'], ['b', '2']]));  // [
'a', '1' ] [ 'b', '2' ]
console.log(...new Set([1, 2, 3]));  // 1 2 3

// 이터러블이 아닌 일반 객체는 Spread 문법의 대상이 될 수 없다.
console.log(...{ a: 1, b: 2 });
// TypeError: Found non-callable @@iterator
```

함수의 인수로 사용하는 경우

배열을 분해하여 배열의 각 요소를 파라미터에 전달하고 싶은 경우,
Function.prototype.apply를 사용하는 것이 일반적이다.

```
// ES5
function foo(x, y, z) {
  console.log(x); // 1
  console.log(y); // 2
  console.log(z); // 3
}

// 배열을 분해하여 배열의 각 요소를 파라미터에 전달하려고 한다.
const arr = [1, 2, 3];

// apply 함수의 2번째 인수(배열)는 분해되어 함수 foo의 파라이터에 전
달된다.
foo.apply(null, arr);
// foo.call(null, 1, 2, 3);
```

ES6의 Spread 문법(...)을 사용한 배열을 인수로 함수에 전달하면 배열의 요소를 분해하여 순차적으로 파라미터에 할당한다.

```
// ES6
function foo(x, y, z) {
  console.log(x); // 1
  console.log(y); // 2
  console.log(z); // 3
}

// 배열을 foo 함수의 인자로 전달하려고 한다.
const arr = [1, 2, 3];

/* ...[1, 2, 3]는 [1, 2, 3]을 개별 요소로 분리한다(→ 1, 2, 3)
   spread 문법에 의해 분리된 배열의 요소는 개별적인 인자로서 각각의 매개변수에 전달된다. */
foo(...arr);
```

앞에서 살펴본 Rest 파라미터는 Spread 문법을 사용하여 파라미터를 정의한 것을 의미한다. 형태가 동일하여 혼동할 수 있으므로 주의가 필요하다.

```
/* Spread 문법을 사용한 매개변수 정의 (= Rest 파라미터)
   ...rest는 분리된 요소들을 함수 내부에 배열로 전달한다. */
function foo(param, ...rest) {
  console.log(param); // 1
  console.log(rest);  // [ 2, 3 ]
}
foo(1, 2, 3);

/* Spread 문법을 사용한 인수
  배열 인수는 분리되어 순차적으로 매개변수에 할당 */
function bar(x, y, z) {
  console.log(x); // 1
  console.log(y); // 2
  console.log(z); // 3
}
```

```
// ...[1, 2, 3]는 [1, 2, 3]을 개별 요소로 분리한다(-> 1, 2, 3)
// spread 문법에 의해 분리된 배열의 요소는 개별적인 인자로서 각각의 매
개변수에 전달된다.
bar(...[1, 2, 3]);
```

Rest 파라미터는 반드시 마지막 파라미터이어야 하지만 Spread 문법을 사용한 인수는 자유롭게 사용할 수 있다.

```
// ES6
function foo(v, w, x, y, z) {
  console.log(v); // 1
  console.log(w); // 2
  console.log(x); // 3
  console.log(y); // 4
  console.log(z); // 5
}

// ...[2, 3]는 [2, 3]을 개별 요소로 분리한다(→ 2, 3)
// spread 문법에 의해 분리된 배열의 요소는 개별적인 인자로서 각각의 매
개변수에 전달된다.
foo(1, ...[2, 3], 4, ...[5]);
```

배열에서 사용하는 경우

Spread 문법을 배열에서 사용하면 보다 간결하고 가독성 좋게 표현할 수 있다. ES5에서 사용하던 방식과 비교하여 살펴보도록 하자.

concat

ES5에서 기존 배열의 요소를 새로운 배열 요소의 일부로 만들고 싶은 경우, 배열 리터럴 만으로 해결할 수 없고 concat 메소드를 사용해야 한다.

```
// ES5
var arr = [1, 2, 3];
```

```
console.log(arr.concat([4, 5, 6])); // [ 1, 2, 3, 4, 5, 6 ]
```

Spread 문법을 사용하면 배열 리터럴 만으로 기존 배열의 요소를 새로운 배열 요소의 일부로 만들 수 있다.

```
// ES6
const arr = [1, 2, 3];
// ...arr은 [1, 2, 3]을 개별 요소로 분리한다
console.log([...arr, 4, 5, 6]); // [ 1, 2, 3, 4, 5, 6 ]
```

push

ES5에서 기존 배열에 다른 배열의 개별 요소를 각각 push하려면 아래와 같은 방법을 사용한다.

```
// ES5
var arr1 = [1, 2, 3];
var arr2 = [4, 5, 6];

// apply 메소드의 2번째 인자는 배열. 이것은 개별 인자로 push 메소드에 전달된다.
Array.prototype.push.apply(arr1, arr2);

console.log(arr1); // [ 1, 2, 3, 4, 5, 6 ]
```

Spread 문법을 사용하면 아래와 같이 보다 간편하게 표현할 수 있다.

```
// ES6
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];

// ...arr2는 [4, 5, 6]을 개별 요소로 분리한다
arr1.push(...arr2); // == arr1.push(4, 5, 6);
```

```
console.log(arr1); // [ 1, 2, 3, 4, 5, 6 ]
```

splice

ES5에서 기존 배열에 다른 배열의 개별 요소를 삽입하려면 아래와 같은 방법을 사용한다.

```
// ES5
var arr1 = [1, 2, 3, 6];
var arr2 = [4, 5];

/*
apply 메소드의 2번째 인자는 배열. 이것은 개별 인자로 splice 메소드에 전달된다.
[3, 0].concat(arr2) → [3, 0, 4, 5]
arr1.splice(3, 0, 4, 5) → arr1[3]부터 0개의 요소를 제거하고 그자리(arr1[3])에 새로운 요소(4, 5)를 추가한다.
*/
Array.prototype.splice.apply(arr1, [3, 0].concat(arr2));

console.log(arr1); // [ 1, 2, 3, 4, 5, 6 ]
```

Spread 문법을 사용하면 아래와 같이 보다 간편하게 표현할 수 있다.

```
// ES6
const arr1 = [1, 2, 3, 6];
const arr2 = [4, 5];

// ...arr2는 [4, 5]을 개별 요소로 분리한다
arr1.splice(3, 0, ...arr2); // == arr1.splice(3, 0, 4, 5);

console.log(arr1); // [ 1, 2, 3, 4, 5, 6 ]
```

copy

ES5에서 기존 배열을 복사하기 위해서는 slice 메소드를 사용한다.

```
// ES5
var arr  = [1, 2, 3];
var copy = arr.slice();

console.log(copy); // [ 1, 2, 3 ]

// copy를 변경한다.
copy.push(4);

console.log(copy); // [ 1, 2, 3, 4 ]
// arr은 변경되지 않는다.
console.log(arr);  // [ 1, 2, 3 ]
```

Spread 문법을 사용하면 보다 간편하게 배열을 복사할 수 있다.

```
// ES6
const arr = [1, 2, 3];
// ...arr은 [1, 2, 3]을 개별 요소로 분리한다
const copy = [...arr];

console.log(copy); // [ 1, 2, 3 ]

// copy를 변경한다.
copy.push(4);

console.log(copy); // [ 1, 2, 3, 4 ]
// arr은 변경되지 않는다.
console.log(arr);  // [ 1, 2, 3 ]
```

이때 원본 배열의 각 요소를 얕은 복사(shallow copy)하여 새로운 복사본을 생성한다. 이는 Array#slice 메소드도 마찬가지다.

```
const todos = [
  { id: 1, content: 'HTML', completed: false },
```

```
  { id: 2, content: 'CSS', completed: true },
  { id: 3, content: 'Javascript', completed: false }
];

// shallow copy
// const _todos = todos.slice();
const _todos = [...todos];
console.log(_todos === todos); // false

// 배열의 요소는 같다. 즉, 얕은 복사되었다.
console.log(_todos[0] === todos[0]); // true
```

Spread 문법과 Object.assign는 원본을 shallow copy한다. Deep copy를 위해서는 lodash의 deepClone을 사용하는 것을 추천한다.

Spread 문법을 사용하면 유사 배열 객체(Array-like Object)를 배열로 손쉽게 변환할 수 있다.

```
const htmlCollection = document.getElementsByTagName('li');

// 유사 배열인 HTMLCollection을 배열로 변환한다.
const newArray = [...htmlCollection]; // Spread 문법

// ES6의 Array.from 메소드를 사용할 수도 있다.
// const newArray = Array.from(htmlCollection);
```

Rest/Spread 프로퍼티

ECMAScript 언어 표준에 제안(proposal)된 Rest/Spread 프로퍼티(Object Rest/Spread Properties)는 객체 리터럴을 분해하고 병합하는 편리한 기능을 제공한다.

```
// 객체 리터럴 Rest/Spread 프로퍼티
// Spread 프로퍼티
const n = { x: 1, y: 2, ...{ a: 3, b: 4 } };
console.log(n); // { x: 1, y: 2, a: 3, b: 4 }
```

```
// Rest 프로퍼티
const { x, y, ...z } = n;
console.log(x, y, z); // 1 2 { a: 3, b: 4 }
```

Spread 문법의 대상은 이터러블이어야 한다. Rest/Spread 프로퍼티는 일반 객체에 Spread 문법의 사용을 허용한다.

Rest/Spread 프로퍼티를 사용하면 객체를 손쉽게 병합 또는 변경할 수 있다. 이는 Object.assign을 대체할 수 있는 간편한 문법이다.

```
// 객체의 병합
const merged = { ...{ x: 1, y: 2 }, ...{ y: 10, z: 3 } };
console.log(merged); // { x: 1, y: 10, z: 3 }

// 특정 프로퍼티 변경
const changed = { ...{ x: 1, y: 2 }, y: 100 };
// changed = { ...{ x: 1, y: 2 }, ...{ y: 100 } }
console.log(changed); // { x: 1, y: 100 }

// 프로퍼티 추가
const added = { ...{ x: 1, y: 2 }, z: 0 };
// added = { ...{ x: 1, y: 2 }, ...{ z: 0 } }
console.log(added); // { x: 1, y: 2, z: 0 }
```

Object.assign 메소드를 사용해도 동일한 작업을 할 수 있다.

```
// 객체의 병합
const merged = Object.assign({}, { x: 1, y: 2 }, { y: 10,
z: 3 });
console.log(merged); // { x: 1, y: 10, z: 3 }

// 특정 프로퍼티 변경
const changed = Object.assign({}, { x: 1, y: 2 }, { y: 100
});
console.log(changed); // { x: 1, y: 100 }
```

```
// 프로퍼티 추가
const added = Object.assign({}, { x: 1, y: 2 }, { z: 0 });
console.log(added); // { x: 1, y: 2, z: 0 }
```