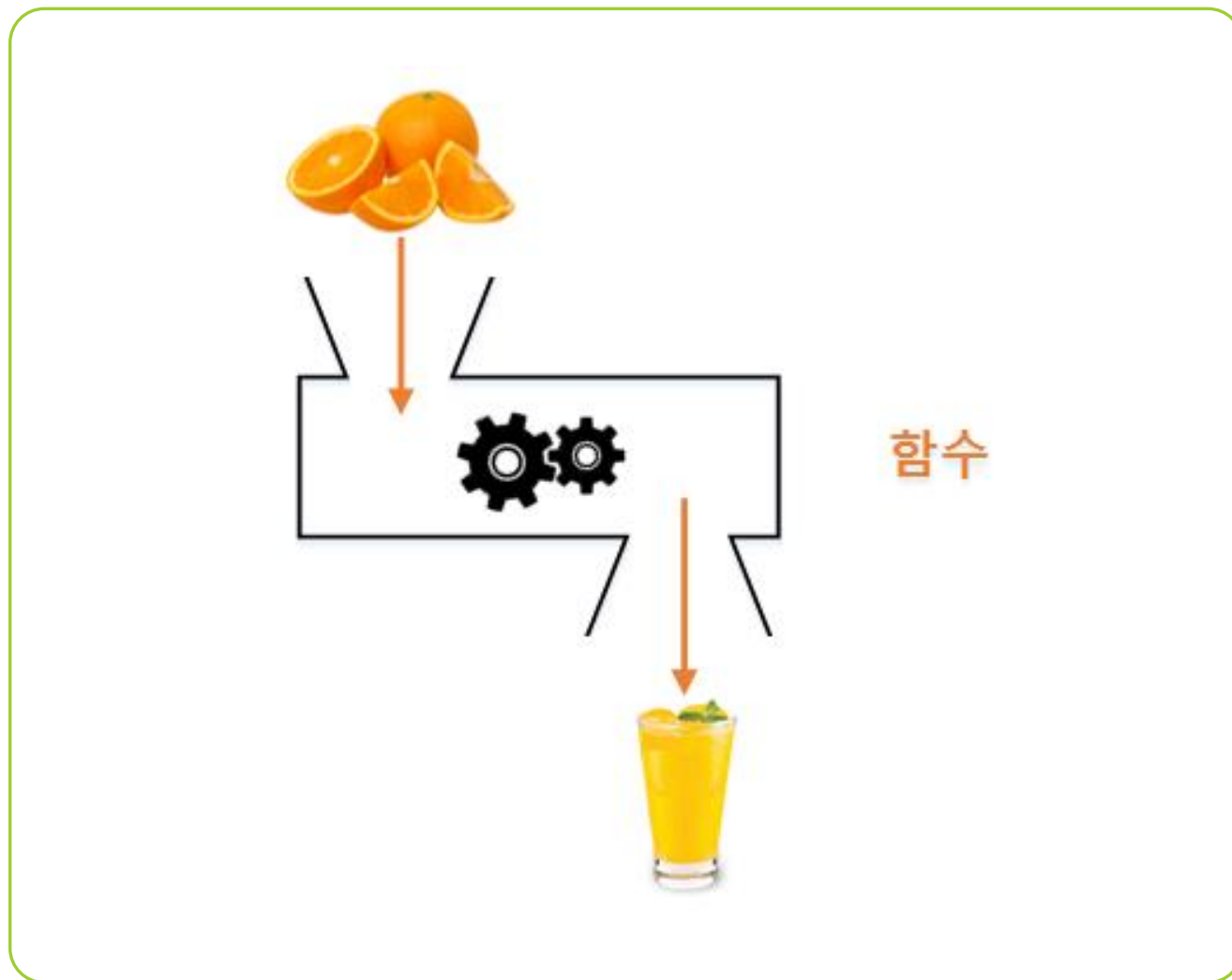


JavaScript

함수1 - 김근형 강사

함수

함수(function)



함수

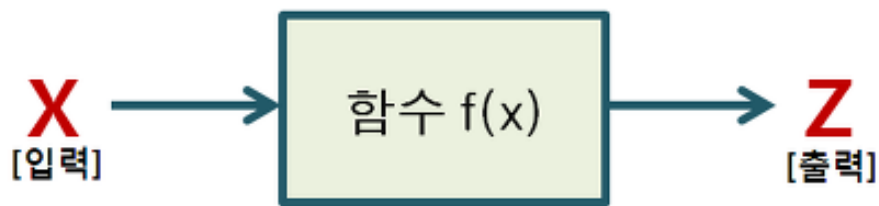
○ 함수란?


- 함수(function)란 하나의 특별한 목적의 작업을 수행하도록 설계된 독립적인 블록을 의미한다.
- 이러한 함수는 필요할 때마다 호출하여 해당 작업을 반복해서 수행할 수 있다.
- 자바스크립트에서는 함수도 하나의 타입(datatype)이다.
- 따라서 함수를 변수에 대입하거나, 함수에 프로퍼티를 지정하는 것도 가능하다.
- 또한, 자바스크립트 함수는 다른 함수 내에 중첩되어 정의될 수도 있다.

함수

○ 함수의 기본 형태

- 함수는 function이라는 명령어 뒤에 함수의 명을 입력한다.
- 함수의 명 뒤에는 매개변수를 담을 수 있는 ()가 있으며 매개변수는 원하는 만큼 선언이 가능하다.
- 브레이스({})에는 구현하고자 하는 로직이 들어간다.
- return을 통해 호출한 곳에 데이터를 전달할 수 있다.



 [자바스크립트 함수 표현 방법]

```
function kwang(x)  -- ① 함수 이름
{
  b = x + 12;      -- ② 연산식 (스크립트)
  return b;        -- ③ 출력 값
};

kwang(7);          -- ④ 함수 실행
```

함수

- 함수 사용 방법
 - 함수의 이름과 함께 옆에 인자를 붙여서 사용한다.

함수명(인자1,인자2,인자3...);

함수 선언 종류

- 함수 선언문
 - 가장 자주 쓰이는 함수 선언문
 - 함수 선언 뒤에 함수에 대한 이름이 등장하며 아래 로직이 선언된다.

```
function 함수이름(인자){  
    함수 로직...  
}
```

```
// add() 함수 선언문  
function add(x, y) {  
    return x + y;  
}  
  
console.log(add(3, 4)); // 7
```

함수 선언 종류

○ 기명 함수 표현식

- 기명 함수 표현식은 기존 이름을 호칭했던 함수를 변수에 넣어 선언하는 방식이다.
- 기존 함수 선언식과 차이점은 앞에 함수를 호출할 변수가 붙는다.
- 주로 재귀함수를 호출할 경우 사용하는 경우가 많다.

```
var 변수명 = function 함수이름(인자){  
    함수 로직...  
}
```

```
// add() 기명 함수 표현식  
var add = function add1(x, y) {  
    return x + y;  
};  
  
var plus = add;  
  
console.log(add(3,4)); // 7  
console.log(plus(5,6)); // 11
```

함수 선언 종류

- 익명 함수 표현식
 - 기존의 기명 함수 표현식에서 function의 이름이 빠진 형태의 함수 표현식
 - 함수를 쓸 수도 있지만 함수의 내용을 다른 변수에 복사해서 마치 값처럼 쓰는 것이 가능하다.

```
var 변수명 = function (인자){  
    함수 로직...  
}
```

```
// add() 익명 함수 표현식  
var add = function (x, y) {  
    return x + y;  
};  
  
var plus = add;  
  
console.log(add(3,4)); // 7  
console.log(plus(5,6)); // 11
```


함수 선언 종류

- Function 생성자 함수 이용
 - Function이란 생성자 함수를 사용해서 쓰는 방식
 - 생성자 함수 안에는 들어갈 변수의 명과 로직이 들어간다.
 - 많이 사용되지는 않는 방법
 - 로직이 복잡하지 않은 함수를 생성할 경우 사용하는 방법

```
var 변수명 = new Function (인자1,인자2,...,로직)
```

```
// Function 생성자를 활용한 방법  
var add = new Function('x', 'y', 'return x + y');  
console.log(add(3, 4)); // 7
```

함수 선언 종류

- 함수 이름 작성 시 주의할 점
 - 함수 이름은 첫 글자가 숫자가 올 수 없다.
 - 함수 이름은 대소문자를 구분한다.
 - 자바스크립트에서 정의된 예약어는 이름으로 사용할 수 없다.
 - 특수문자는 주로 '_' 외엔 사용하지 않는다.

```

console.log(a());
console.log(b());
console.log(c());

function a() {
  return 'a';
}
var b = function bb() {
  return 'bb';
}
var c = function() {
  return 'c';
}

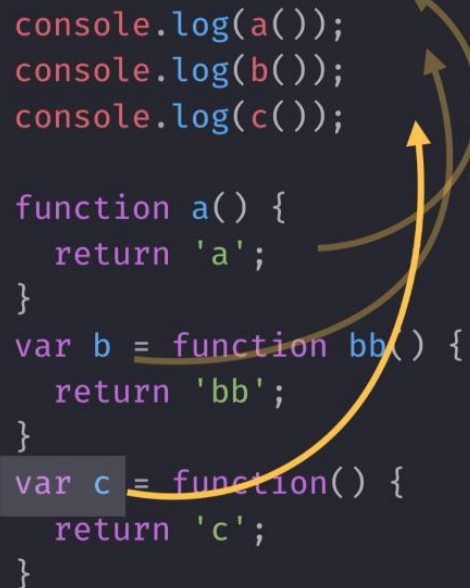
```

```

console.log(a());
console.log(b());
console.log(c());

function a() {
  return 'a';
}
var b = function bb() {
  return 'bb';
}
var c = function() {
  return 'c';
}

```



```

function a() {
  return 'a';
}
var b;
var c;
console.log(a());
console.log(b());
console.log(c());

b = function bb() {
  return 'bb';
}
c = function() {
  return 'c';
}

```

호이스팅

○ 호이스팅

- 변수 선언과 함수 선언을 끌어올리는 기법을 의미한다
- 변수를 통해 선언하는 식이나 값은 호이스팅 대상이 될 수 없다.
- 함수 선언문 방식으로는 호이스팅이 적용되지만 함수 표현식 형태로는 호이스팅이 적용되지 않는다.

함수 & 객체

- 함수 객체

- 함수는 기본 기능인 코드 실행 뿐만이 아니라 함수 자체가 일반 객체처럼 프로퍼티들을 가질 수 있다.

```
// 함수 선언 방식으로 add() 함수 정의
function add(x, y) {
    return x+y;
}

// add() 함수 객체에 result, status 프로퍼티 추가
add.result = add(3, 2);
add.status = 'OK';

console.log(add.result); // 5
console.log(add.status); // 'OK'
```

함수 & 객체

○ 객체 내 함수

- 객체 내 함수를 선언할 수 있는데 이러한 함수를 “메서드”라고 부른다.

```
var person = {  
  name: ['Bob', 'Smith'],  
  age: 32,  
  gender: 'male',  
  interests: ['music', 'skiing'],  
  bio: function() {  
    console.log(this.name[0] + ' ' + this.name[1] + ' is ' +  
      this.age + ' years old. He likes ' + this.interests[0] +  
      ' and ' + this.interests[1] + '.');  
  },  
  greeting: function() {  
    console.log('Hi! I\'m ' + this.name[0] + '.');  
  }  
};  
  
console.log(person.name);  
console.log(person.name[0]);  
console.log(person.age);  
console.log(person.interests[1]);  
person.bio();  
person.greeting();
```

▶ (2) ["Bob", "Smith"]

Bob

32

skiing

Bob Smith is 32 years old. He likes music and skiing.

Hi! I'm Bob.

일급 함수

- 일급 함수
 - 자바스크립트에서 함수는 값으로 취급된다.
 - 함수도 일반 객체처럼 취급할 수 있다.
 - 그래서 자바스크립트의 함수는 아래와 같은 동작이 가능하다.
 - 리터럴에 의해 생성
 - 변수나 배열의 요소, 객체의 프로퍼티 등에 할당 가능
 - 함수의 인자로 전달 가능
 - 함수의 리턴값으로 리턴 가능
 - 동적으로 프로퍼티를 생성 및 할당 가능
 - 이러한 함수를 일급(first class) 함수 라고 한다.

일급 함수

- 변수나 프로퍼티의 값으로 할당

```
// 변수에 함수 할당
var foo = 100;
var bar = function () { return 100; };
console.log(bar()); // 100

// 프로퍼티에 함수 할당
var obj = {};
obj.baz = function () {return 200; }
console.log(obj.baz()); // 200
```

일급 함수

- 함수를 인자로 전달

```
// 함수 표현식으로 foo() 함수 생성
var foo = function(func) {
    func(); // 인자로 받은 func() 함수 호출
};

// foo() 함수 실행
foo(function() {
    console.log('Function can be used as the argument.');
```


일급 함수

○ 리턴값으로 활용

```
// 함수를 리턴하는 foo() 함수 정의
var foo = function() {
    return function () {
        console.log('this function is the return value.')
    };
};

var bar = foo();
bar();
```

함수 객체

○ 함수 객체의 기본 프로퍼티

```
function add(x, y) {  
    return x + y;  
}  
  
console.dir(add);
```

```
▼ f add(x, y) ⓘ  
  arguments: null  
  caller: null  
  length: 2  
  name: "add"  
  ▶ prototype: {constructor: f}  
  ▼ __proto__: f ()  
    ▶ apply: f apply()  
      arguments: (...)  
    ▶ bind: f bind()  
    ▶ call: f call()  
      caller: (...)  
    ▶ constructor: f Function()  
      length: 0  
      name: ""  
    ▶ toString: f toString()  
    ▶ Symbol(Symbol.hasInstance): f [Symbol.hasInstance]()  
    ▶ get arguments: f ()  
    ▶ set arguments: f ()  
    ▶ get caller: f ()  
    ▶ set caller: f ()  
    ▶ __proto__: Object  
      [[FunctionLocation]]: <unknown>  
    ▶ [[Scopes]]: Scopes[0]  
      [[FunctionLocation]]: VM46:1  
    ▶ [[Scopes]]: Scopes[1]
```

함수 객체

- 함수 객체의 기본 프로퍼티
 - name : 함수의 이름, 익명함수의 경우 null로 떨어진다.
 - caller : 자신을 호출한 함수를 나타낸다.
 - arguments : 함수를 호출할 때 전달된 인자값을 나타낸다.
 - __proto__ : 함수 객체의 부모 역할을 하는 프로토타입 객체를 가리킨다(Function.prototype)

함수 객체

- 함수 객체의 기본 프로퍼티
 - `length` : 함수가 정상적으로 실행될 때 기대되는 인자의 개수를 표시함

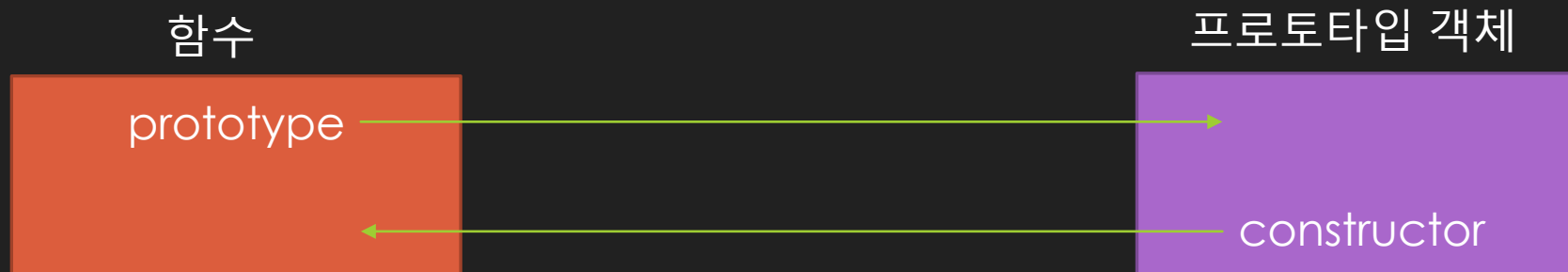
```
function func0() {  
}  
  
function func1(x) {  
    return x;  
}  
  
function func2(x, y) {  
    return x + y;  
}  
  
function func3(x, y, z) {  
    return x + y + z;  
}  
  
console.log('func0.length - ' + func0.length); // func0.length - 0  
console.log('func1.length - ' + func1.length); // func1.length - 1  
console.log('func2.length - ' + func2.length); // func2.length - 2  
console.log('func3.length - ' + func3.length); // func3.length - 3
```

함수 객체

○ 함수 객체의 기본 프로퍼티

○ prototype 프로퍼티

- `[[Prototype]]` : 객체 입장에서 자신의 부모 역할을 하는 프로토타입 객체를 가리키는 프로퍼티
- 함수객체 `Prototype` : 이 함수가 생성자로 사용될 때 이 함수를 통해 생성된 객체의 부모 역할을 하는 프로토타입 객체
- 프로토타입 프로퍼티는 함수가 생성될 때 만들어지며, `constructor` 프로퍼티 하나만 있는 객체를 가리킨다.
- 그리고 `prototype` 프로퍼티가 가리키는 프로토타입 객체의 유일한 `constructor` 프로퍼티는 자신과 연결된 함수를 가리킨다.
- 즉 이 둘은 `prototype` 과 `constructor`라는 프로퍼티로 서로를 참조한다.



함수 객체

- 함수 객체의 기본 프로퍼티
 - prototype 프로퍼티

```
// myFunction 함수 정의
function myFunction() {
    return true;
}

console.dir(myFunction.prototype);
console.dir(myFunction.prototype.constructor);
```

```
▼ Object ⓘ
  ► constructor: f myFunction()
  ► __proto__: Object

▼ f myFunction() ⓘ
  arguments: null
  caller: null
  length: 0
  name: "myFunction"
  ► prototype: {constructor: f}
  ► __proto__: f ()
    [[FunctionLocation]]: VM58:2
  ► [[Scopes]]: Scopes[1]
```

함수 객체

○ 함수 객체와 기본 프로퍼티

○ arguments 객체

- js는 함수 형식에 맞춰 인자를 넘기지 않더라도 에러가 발생하지 않는다.
- 런타임 시에 호출된 인자의 개수를 확인하고 이에 따라 동작을 다르게 해줘야 할 경우 arguments 객체를 활용할 수 있다.
- arguments 객체는 함수를 호출할 때 넘긴 인자들이 배열 형태로 저장된 객체를 의미한다.
- 이 객체는 실제 배열이 아닌 유사 배열 객체이다.

```
function func(arg1, arg2) {  
    console.log(arg1, arg2);  
}  
  
func();    // undefined undefined  
func(1);   // 1 undefined  
func(1,2); // 1 2  
func(1,2,3); // 1 2
```

```
// add() 함수  
function add(a, b) {  
    // arguments 객체 출력  
    console.dir(arguments);  
    return a+b;  
}  
  
console.log(add(1)); // NaN  
console.log(add(1,2)); // 3  
console.log(add(1,2,3)); // 3
```

함수 객체

- arguments 객체 구성
 - 함수를 호출할 때 넘겨진 인자(배열 형태) : 함수를 호출할 때 첫 번째 인자는 0번 인덱스, 두 번째 인자는 1번 인덱스, ...
 - length 프로퍼티 : 호출할 때 넘겨진 인자의 개수를 의미
 - callee 프로퍼티 : 현재 실행 중인 함수의 참조 값

```
function sum(){  
    var result = 0;  
  
    for(var i=0; i < arguments.length; i++){  
        result += arguments[i];  
    }  
  
    return result;  
}  
  
console.log(sum(1,2,3)); // 6  
console.log(sum(1,2,3,4,5,6,7,8)); //36
```


함수 객체

- arguments.callee
 - 지정된 Function 객체의 본문에 해당하는 실행 중인 Function 객체를 반환한다.
 - callee 속성은 관련된 함수가 실행될 때만 사용 가능한 arguments 객체의 멤버이다.
 - callee 속성의 초기 값은 실행 중인 Function 객체이다. 이 속성은 익명 함수의 재귀를 허용한다.
 - 최신 JavaScript 엔진에서 arguments.callee를 사용하면 성능이 확 떨어진다.

```
function factorial(n){  
    if (n <= 0)  
        return 1;  
    else  
        return n * arguments.callee(n - 1)  
}  
console.log(factorial(4));
```

24

함수 객체

- arguments.caller
 - 현재 함수를 호출한 함수를 가져온다.
 - caller 속성을 문자열 컨텍스트에 사용하면 functionName.toString과 같은 결과가 나온다.
 - 이 속성은 더 이상 사용되지 않는다.

```
function CallLevel(){  
  if (CallLevel.caller == null)  
    return("CallLevel was called from the top level.");  
  else  
    return("CallLevel was called by another function.");  
}
```

```
console.log(CallLevel());
```

```
CallLevel was called from the top level.
```

Default Parameter

- 기본값 매개변수(Default Parameter)
 - 파라미터에 제대로 된 개수의 값이 들어오지 않을 경우 해당 파라미터를 미리 지정한 값으로 초기화 할 때 사용하는 기능이다.
 - 해당 파라미터의 값이 undefined 일 경우 초기화 값으로 대체되며 아래와 같이 사용 가능하다.

```
function fnName(param1 = defaultValue1, ..., paramN = defaultValueN) { ... }
```

```
function multiply(a, b = 1) {  
  return a * b  
}
```

```
multiply(5, 2)           // 10  
multiply(5)              // 5  
multiply(5, undefined)   // 5
```

```
function append(value, array = []) {  
  array.push(value);  
  return array;  
}
```

```
append(1); // [1]  
append(2); // [2], [1, 2]가 아니라
```

Rest Parameter

- Rest Parameter
 - Rest 파라미터는 Spread 연산자(...)를 사용하여 함수의 파라미터를 작성한 형태를 말한다.
 - 즉, Rest 파라미터를 사용하면 함수의 파라미터로 오는 값들을 "배열"로 전달받을 수 있다.

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

Rest Parameter

- Rest Parameter

- 함수의 마지막 파라미터의 앞에 ...를 붙여 (사용자가 제공한) 모든 나머지 인수를 "표준" 자바스크립트 배열로 대체한다.
- **마지막 파라미터만 "Rest 파라미터"가 될 수 있다.**

```
function myFun(a, b, ...manyMoreArgs) {  
  console.log("a", a);  
  console.log("b", b);  
  console.log("manyMoreArgs", manyMoreArgs);  
}
```

```
myFun("one", "two", "three", "four", "five", "six");  
// a, one  
// b, two  
// manyMoreArgs, [three, four, five, six]  
myFun("one", "two", "three");  
// a, one  
// b, two  
// manyMoreArgs, [three]  
myFun("one", "two");  
// a, one  
// b, two  
// manyMoreArgs, []
```

Rest Parameter

○ Rest Parameter

- arguments와 rest파라미터의 차이점은 arguments는 유사 배열 객체고 rest는 배열이다.
- arguments는 유사배열객체이기 때문에 Array 오브젝트의 메서드를 사용할 수 없다.
- 따라서 ES6에서는 arrow function에 arguments는 사용할 수 없을 뿐더러 Rest 파라미터를 사용하면 더 유연한 코드를 작성할 수 있는 것이기 때문에 Rest 파라미터 사용을 권장한다.

```
var foo = function () {  
    console.log(arguments);  
};  
foo(1, 2); // { '0': 1, '1': 2 }
```

```
Arguments(2) [1, 2, callee: f, Symbol(Symbol.iterator): f]  
  0: 1  
  1: 2  
> callee: f () {\n                console.log(arguments);\n  length: 2  
> __proto__: Object
```

Rest Parameter

- Rest Parameter

- arguments는 유사배열 객체다 보니 Rest 가 나오기 이전에는 위와 같은 방식으로 arguments 를 쓰기도 했다.

```
// Rest 파라미터 이전에, "arguments" 는 다음을 사용해 일반적인 배열로 변환될 수 있음
function f1(a, b) {
    var normalArray = Array.prototype.slice.call(arguments);
    // -- 또는 --
    var normalArray = [].slice.call(arguments);
    // -- 또는 --
    var normalArray = Array.from(arguments);

    var first = normalArray.shift(); // OK, 첫 번째 인수를 반환
    // var first = arguments.shift(); // ERROR (arguments 가 일반적인 배열이 아님)
    console.log(first); // 1
}

// 이제 rest 파라미터를 사용해 쉽게 일반적인 배열에 접근할 수 있음
function f2(...args) {
    var normalArray = args;
    var first = normalArray.shift(); // OK, 첫 번째 인수를 반환
    console.log(first); // 1
}

f1(1, 2);
f2(1, 2);
```