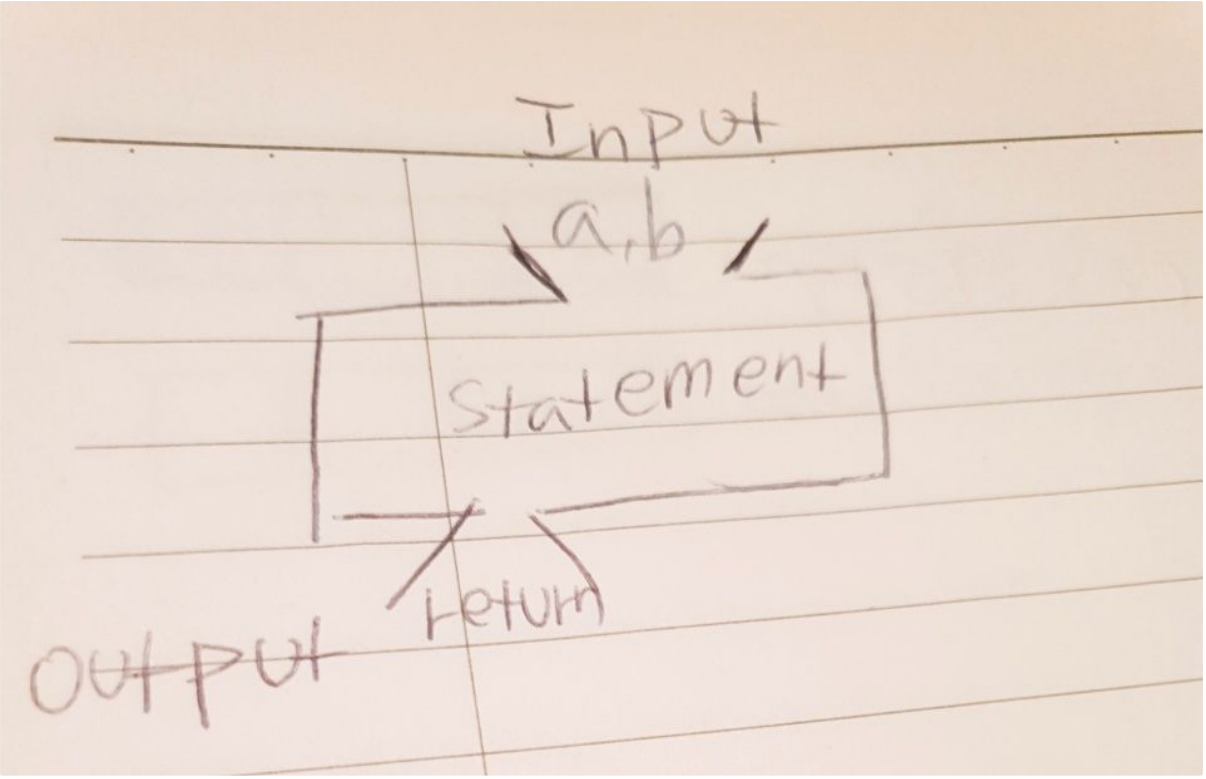
[js]WhatIsTheFunction

[js] function이 대체 뭐야?

1. function이란 무엇인가
2. 역할
3. 함수를 정의하는 방법
    - 함수 선언식
    - 함수 표현식
    - Function() 생성자 함수

function이란 무엇인가?



1. 어떤 특정 작업을 수행하기 위해 필요한 일련의 구문(statement)들을 그룹화하기 위한 개념. 만일 스크립트의 다른 부분에서도 동일한 작업을 반복적으로 수행해야 한다면(동일한 구문을 계속 반복 작성 하는 것이 아니라) 미리 작성된 함수를 재사용할 수도 있음.(코드의 재사용)
2. 함수의 일반적인 기능은 특정 작업을 수행하는 구문들의 집합을 정의하고 필요시에 호출하여 **필요한 값 또는 수행 결과를 얻는 것**임. 이러한 일반적인 기능(**코드의 재사용성**) 이외에 객체 생성, 객체의 행위 지정(**메소드**), **정보의 구성 및 은닉**, **클로저**, **모듈화** 등의 기능을 수행할 수 있음.

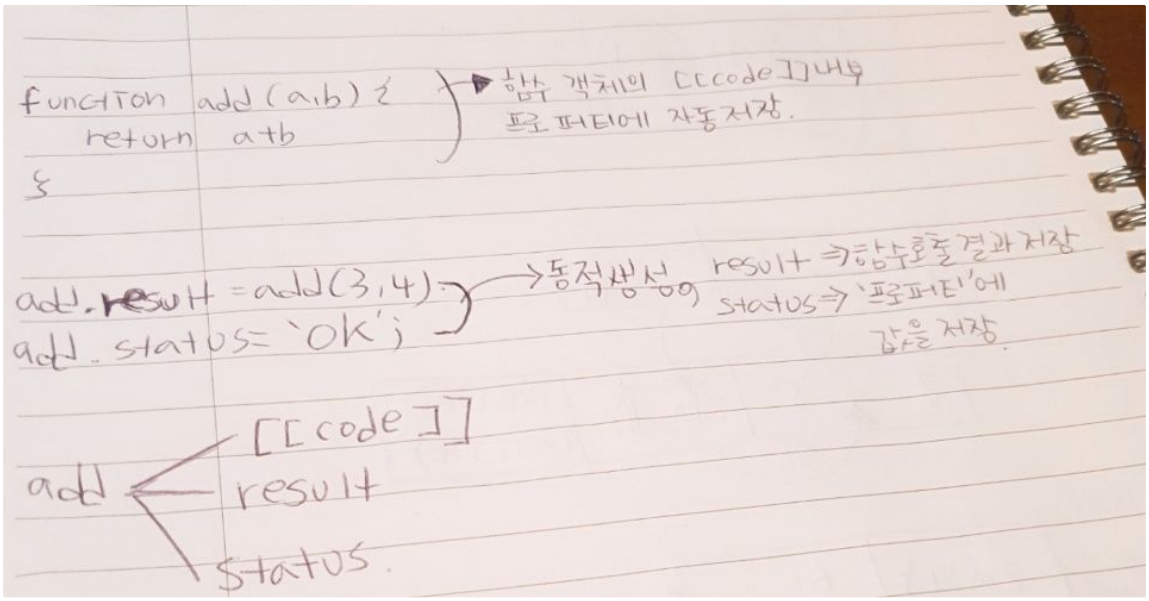
함수는 구문의 집합으로 모듈화의 근간이 됨.
~~얼반적으로 프로그래밍 기술은 요구 사항의~~ 집합을 자료구조와 함수의 집합으로 변환하는 것.

3. 함수도 객체임. 다른 객체와 구분될 수 있는 특징은 호출할 수 있다는 것임.

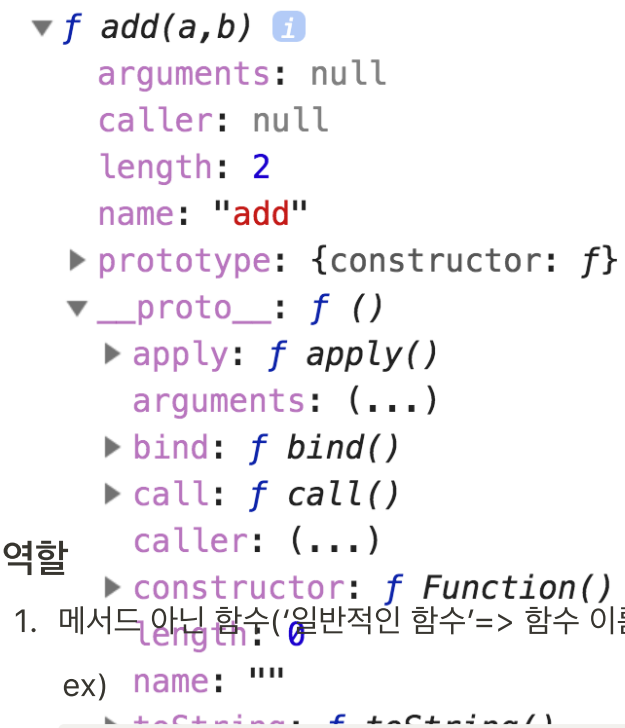
   ex)

```
function add(a,b) { return a+b; } add.result = add(3,4);
add.status='OK';
```

Plain Text



4. 함수도 객체(일급 객체 First-class Object)이므로 다른 값들처럼 사용할 수 있음. 즉 변수나 객체, 배열 등에 저장될 수 있고 다른 함수에 전달되는 인수로도 사용될 수 있으며 함수의 반환값이 될 수도 있음.
5. 함수는 Function의 인스턴스임.
6. Function.prototype에서 메서드를 가져옴.



역할

1. 메서드 아닌 함수('일반적인 함수'=> 함수 이름은 소문자로 시작함.)

   ex)

   ```
   function sum(a,b) { return a+b; }
   ```

   Plain Text

2. 생성자 => 이름은 대문자로 시작

   ex)

   ```
   new Date
   ```

   Plain Text

3. 메서드 => 이름은 소문자로 시작, 객체의 프로퍼티에 저장

   ex)

   ```
   obj.method()
   ```

   Plain Text

함수 정의하는 방법

1. 함수 선언식 (Function declaration)
2. 함수 표현식 (Function expression)
3. Function() 생성자 함수

함수 선언식(Function declaration)

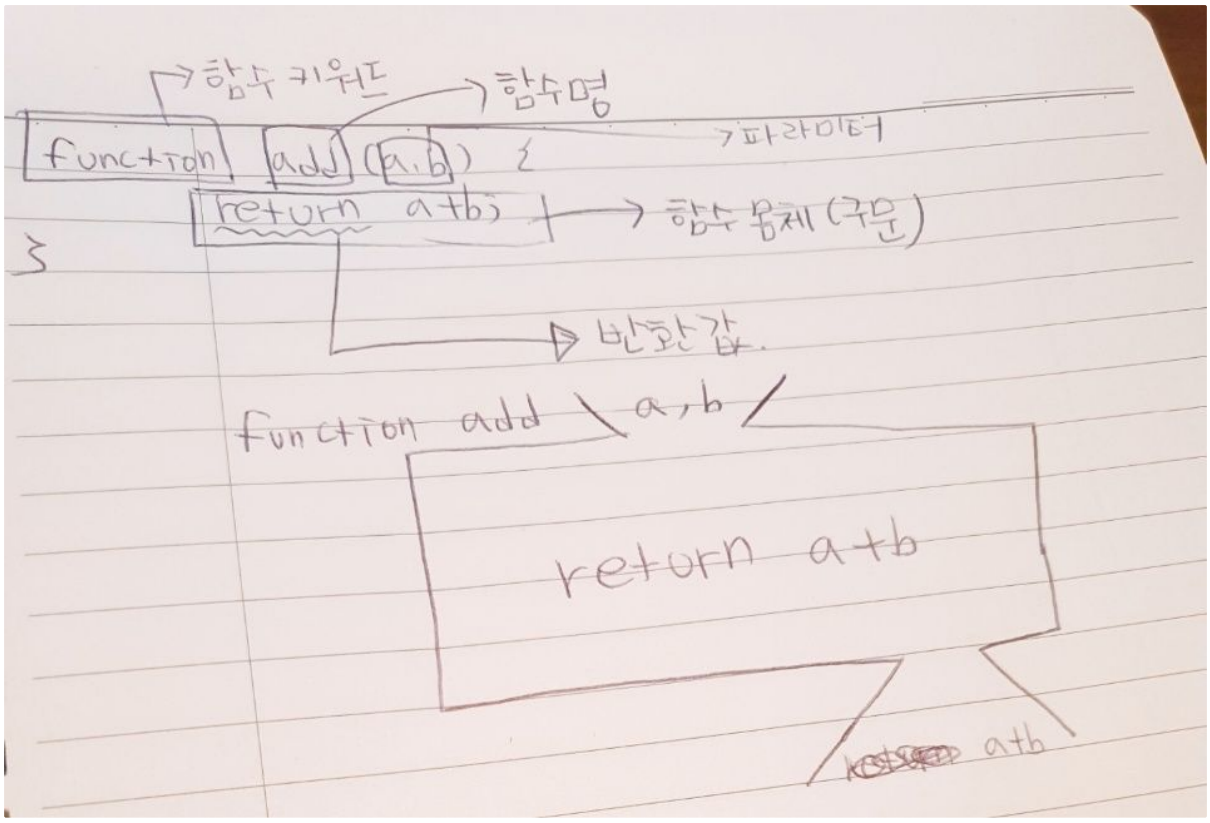
```
* 함수명 > 함수 선언식의 경우, 함수명은 생략할 수 없음. 함수명은 함수 몸체에서 자신을 재귀적(recursive) 호출하거나 자바스크립트 디버거가 해당 함수를 구분할 수 있는 식별자의 역할을 함. * 매개변수(파라미터) 목록 > 0개 이상의 목록으로 괄호로 감싸고 콤마로 분리함. 다른 언어와의 차이점은 매개변수의 자료형을 기술하지 않는 다는 것임. 이 때문에 함수 몸체 내에서 매개변수의 자료형 체크가 필요할 수 있음. * 함수 몸체(구문) > 실제 함수가 호출 되었을 때 실행되는 구문들의 집합임. 중괄호({})로 구문들을 감싸고 return문으로 결과값을 반환할 수 있음. => 이를 반환값이라 함.
```

Plain Text

함수 리터럴

```
function add(a,b) { return a+b; }
```

Plain Text

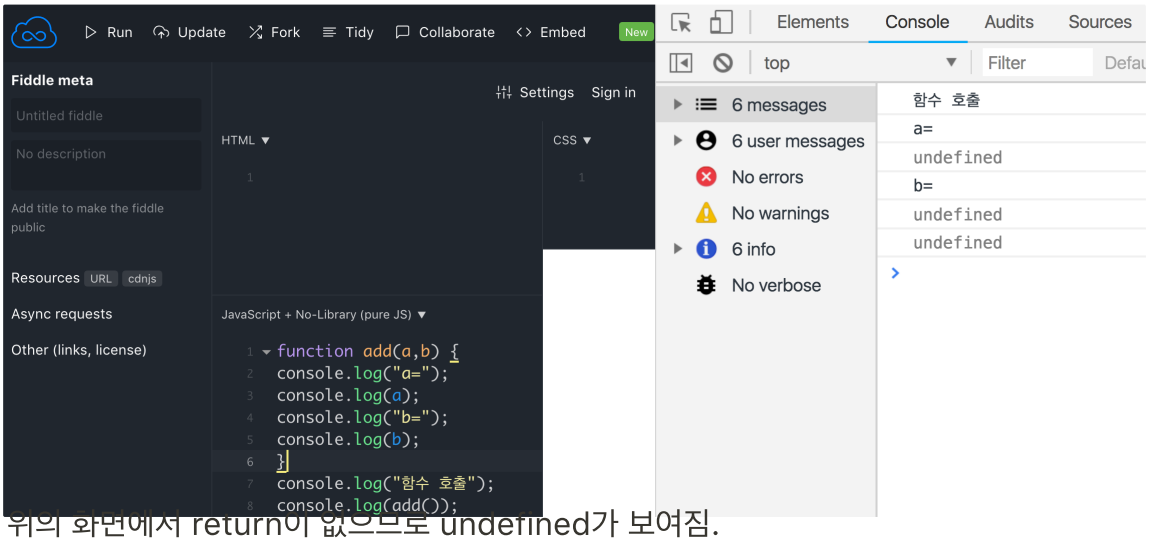


-함수명 -> 함수명은 선택사항임. 있는 경우에는 기명함수라고 하고 없는 경우에는 익명함수(무기명함수)라고 함.

-반환값 -> 반환값이 없으면 묵시적으로 undefined임. return문을 만나는 즉시 실행을 멈추고 빠져 나옴.

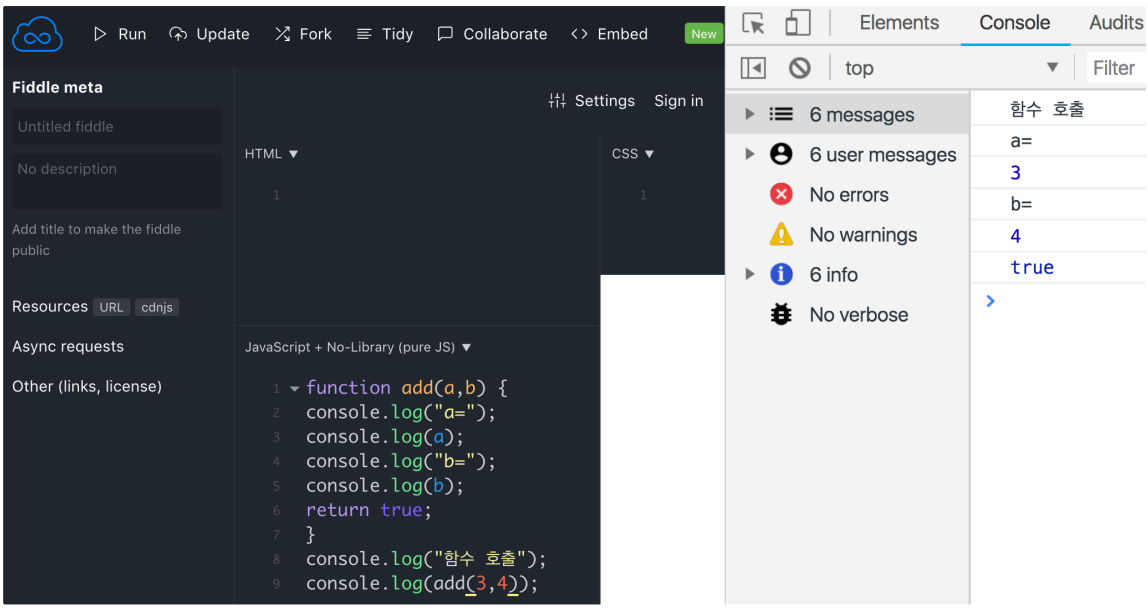
*참고

1. 파라미터 : 변수와 동일하게 메모리 공간을 확보함. 변수는 전달된 매개변수에 대입. undefined로 초기화함.



위의 화면에서 return이 없으므로 undefined가 보여짐.

a와 b는 undefined가 출력되는 것을 볼 수 있음.



위의 화면에서는 return이 true이므로 true가 출력이 되고, 함수 호출 시 해당 파라미터에 대한 값을 던져서 각 각 3,4가 출력 되는 것을 확인할 수 있음.

2. 매개변수(=파라미터)와 Argument

- 매개변수(=파라미터)

> =formal Parameter = formal argument => 지정된 매개변수

> 매개변수 숫자를 따지지 않고, 데이터 타입도 체크하지 않음. => 매개변수가 내부적으로 배열로 간주하기 때문임.

> 이 배열은 항상 함수에 전달되지만 함수는 배열에 어떤 값이 들어가 있는지 체크하지 않음.

> 빈 배열이 들어와도 상관없고(파라미터가 없어도 상관없고), 필요한 매

> 개변수보다 더 많이 들어와도 괜찮음.

ex)

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

Plain Text

- Argument

> =actual Parameter = actual Argument = 호출 시 매개 변수

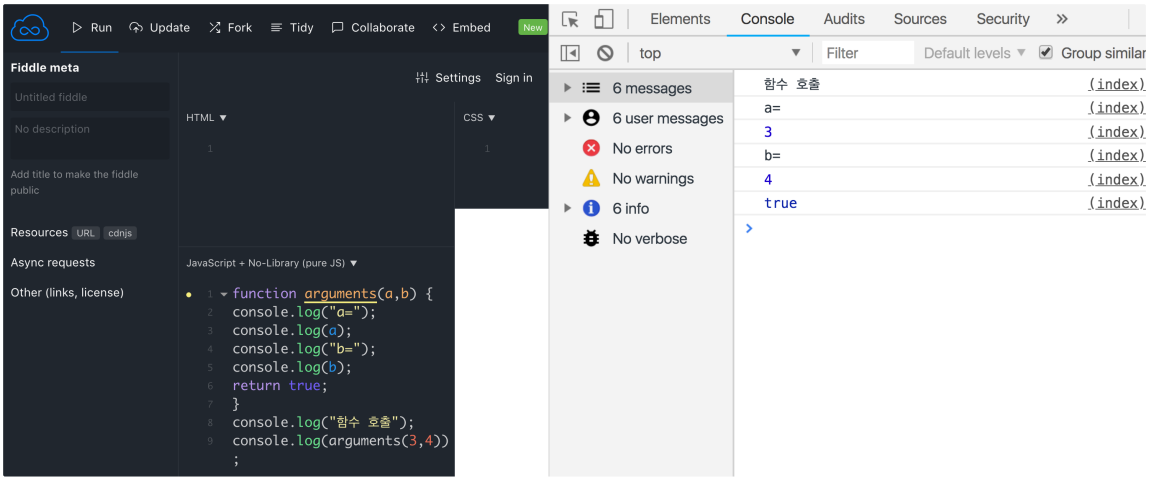
> 함수 호출

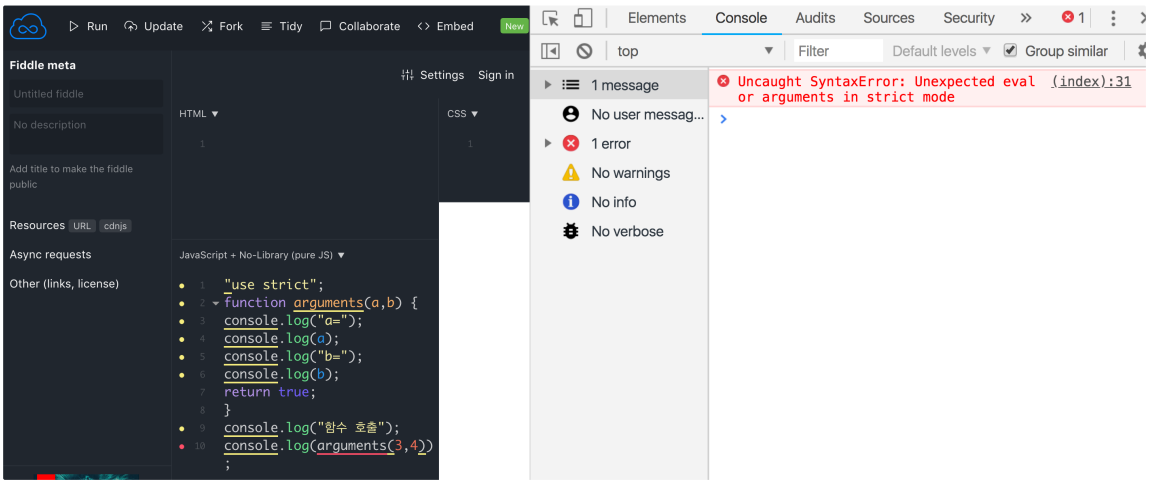
> 인수로 초기화

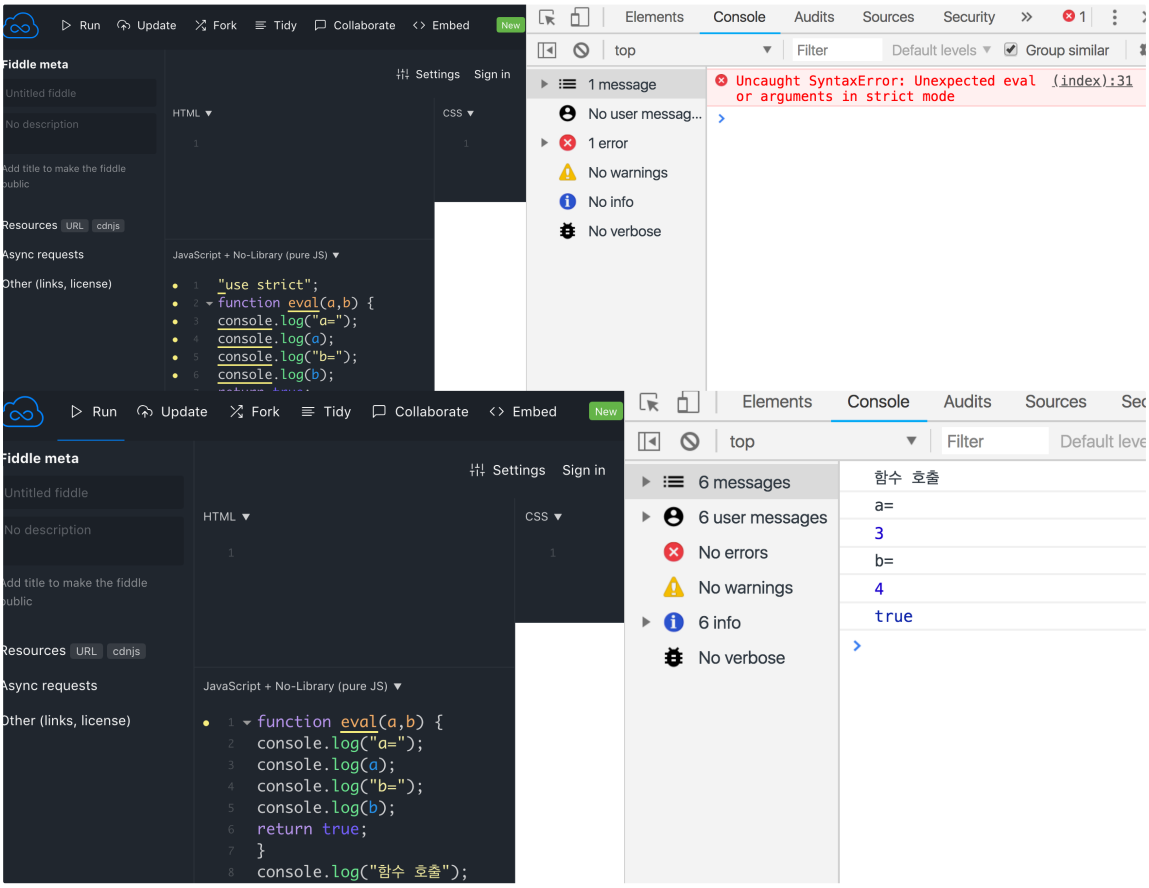
ex) foo(3,4);

3. 스트릭트 모드에서 함수의 제한

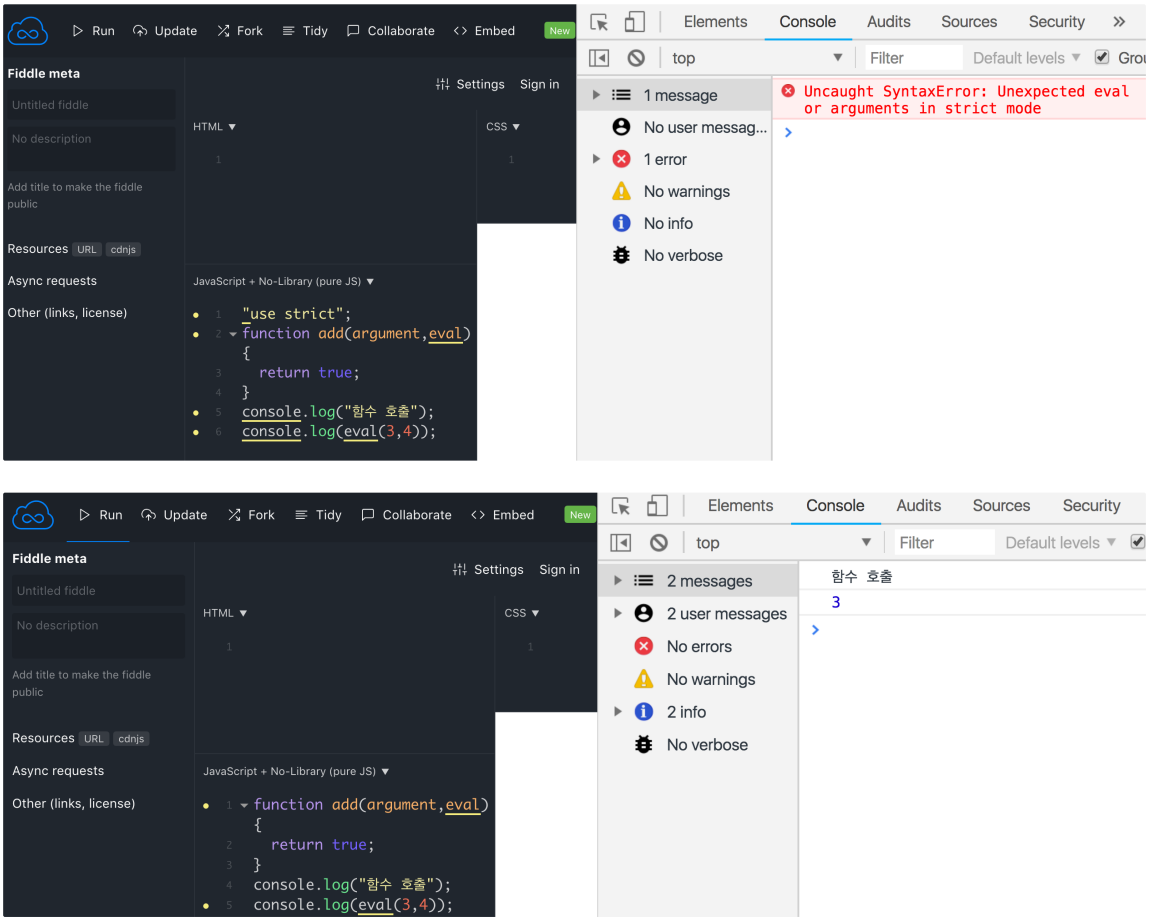
> 함수 이름에 eval이나 argument는 사용할 수 없음.





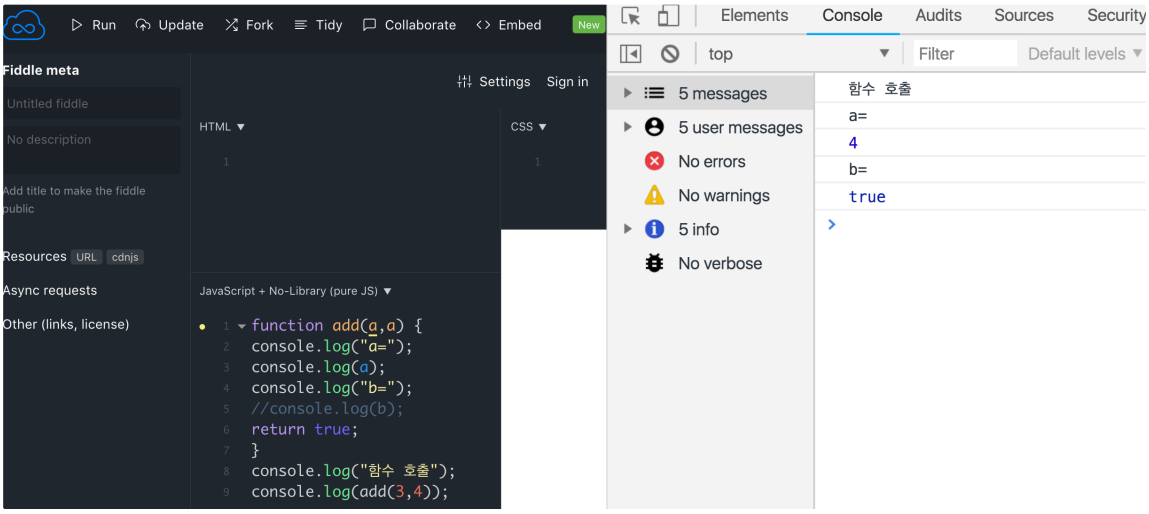


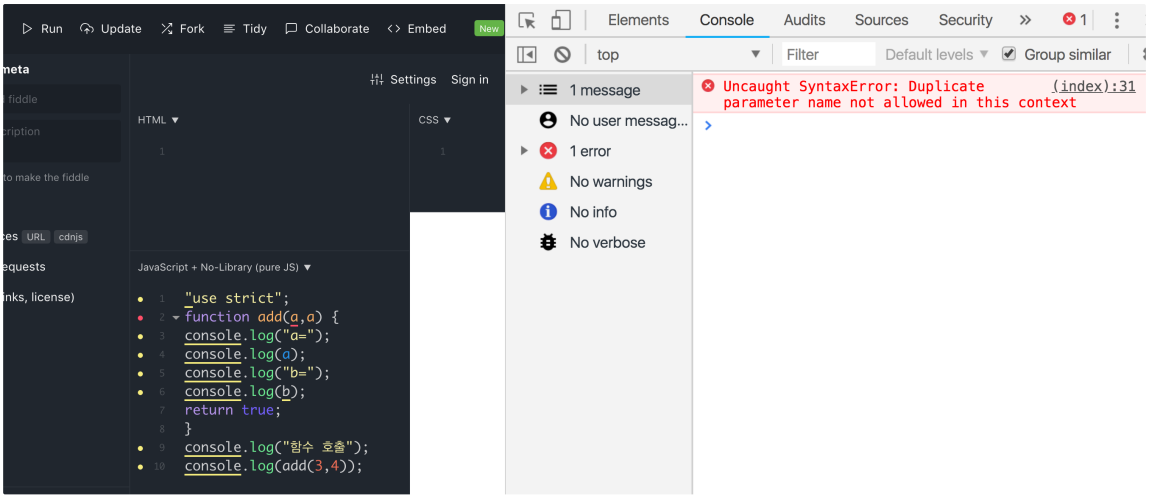
> 매개 변수 이름에도 eval이나 arguments를 쓸 수 없음.



■

> 서로 다른 매개변수에 결코 같은 이름을 쓸 수 없음.





함수 표현식

자바스크립트의 함수는 일급 객체이므로 아래와 같은 특징이 있음.

*참고

1. 일급객체 : 함수형 프로그래밍이 가능함, 함수가 일반 객체처럼 취급함.
2. 함수의 일급 객체 특징

> 무명의 리터럴로 표현이 가능함.

```
function(a,b) { return a+b; }
```

Plain Text

> 변수나 자료 구조(객체, 배열)에 저장할 수 있음.

```
// 배열에 저장하는 경우 var arr = []; var factorialVal =
```

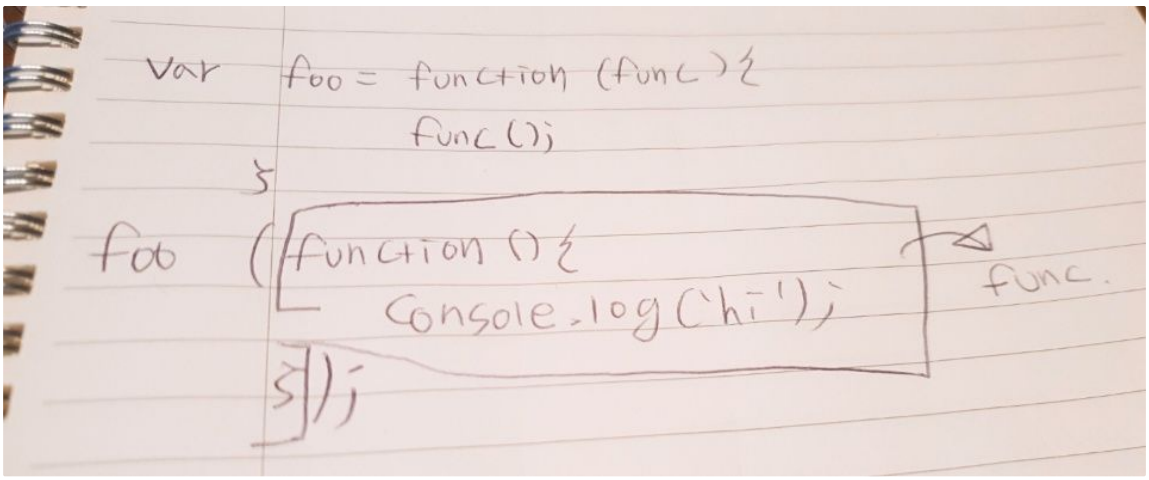
```
function factorial(n) { if(n<=1) { return 1; } return n *
factorial(n-1); } arr.result = factorialVal; arr.result(3);
console.dir(arr); console.log(arr); // 객체에 저장하는 경우 var
obj = {}; obj.baz = function() { return 200; }
console.log(obj); console.log(obj.baz); console.dir(obj);
```

Plain Text

> 함수의 파라미터로 전달할 수 있음.

```
var foo = function(func) { func(); } foo (function() {
console.log('hi');  } );
```

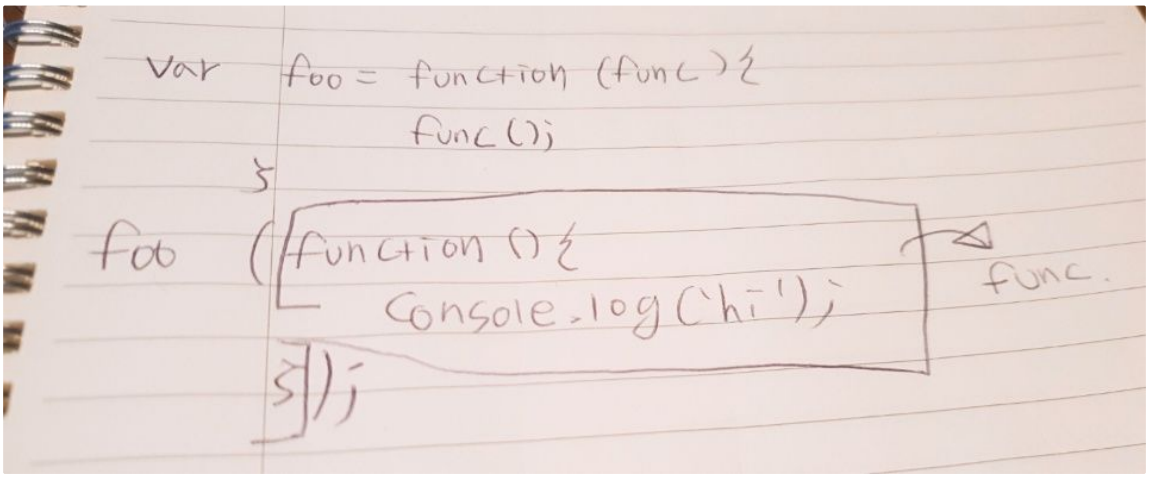
Plain Text



> 반환값(return value)로 사용할 수 있음.

```
var foo = function() { return function() {
console.log('hello?'); } }; var bar = foo; bar();
```

Plain Text



함수의 일급 객체의 특성을 이용하여 함수 리터럴 방식으로 함수를 정의하고 변수에 할당할

수 있는데 이러한 방식을 함수 표현식(Function expression)이라 함.
함수선언식으로 정의한 함수 add()을 함수 표현식으로 정의하면 아래와 같음.

```
var add = function(a,b) { return a+b; }
```

Plain Text

결과를 add에 할당하고, 그 변수(=> 위의 코드에서는 add)를 통해 호출함.

함수 표현식으로 사용된 함수 이름은 함수 내부에서 함수를 재귀적으로 호출(표현식으로 자기 자신을 참조할 수 있으므로) 하거나, 디버거 등 에서 함수를 구분할 때 사용. 함수 표현식 내부에서만 접근할 수 있음.

```
var factorialVal = function factorial(n) {      if(n<=1) {
return 1; }      return n * factorial(n-1) }
console.log(factorialVal(3)); console.log(factorial(3)); //
Uncaught ReferenceError: add is not defined
```

Plain Text

위의 코드에서 함수 표현식으로 사용된 함수 이름은 factorial임.

함수 표현식으로 정의한 함수는 함수명을 생략할 수 있음. 이러한 함수를 **익명함수**라 함.

함수 표현식에서는 함수명을 생략하는 것이 일반적임.

```
// 기명 함수 표현식 (named function expression) var foo = function
add(a,b) { return a+b; } // 익명함수(무기명함수) 표현식(anonymous
function expression) var bar = function(a,b) { return a+b; }
console.log(foo(1,2)) // 3 console.log(bar(1,2)) // 3
console.log(add(1,2)) // Uncaught ReferenceError: add is not
defined
```

Plain Text

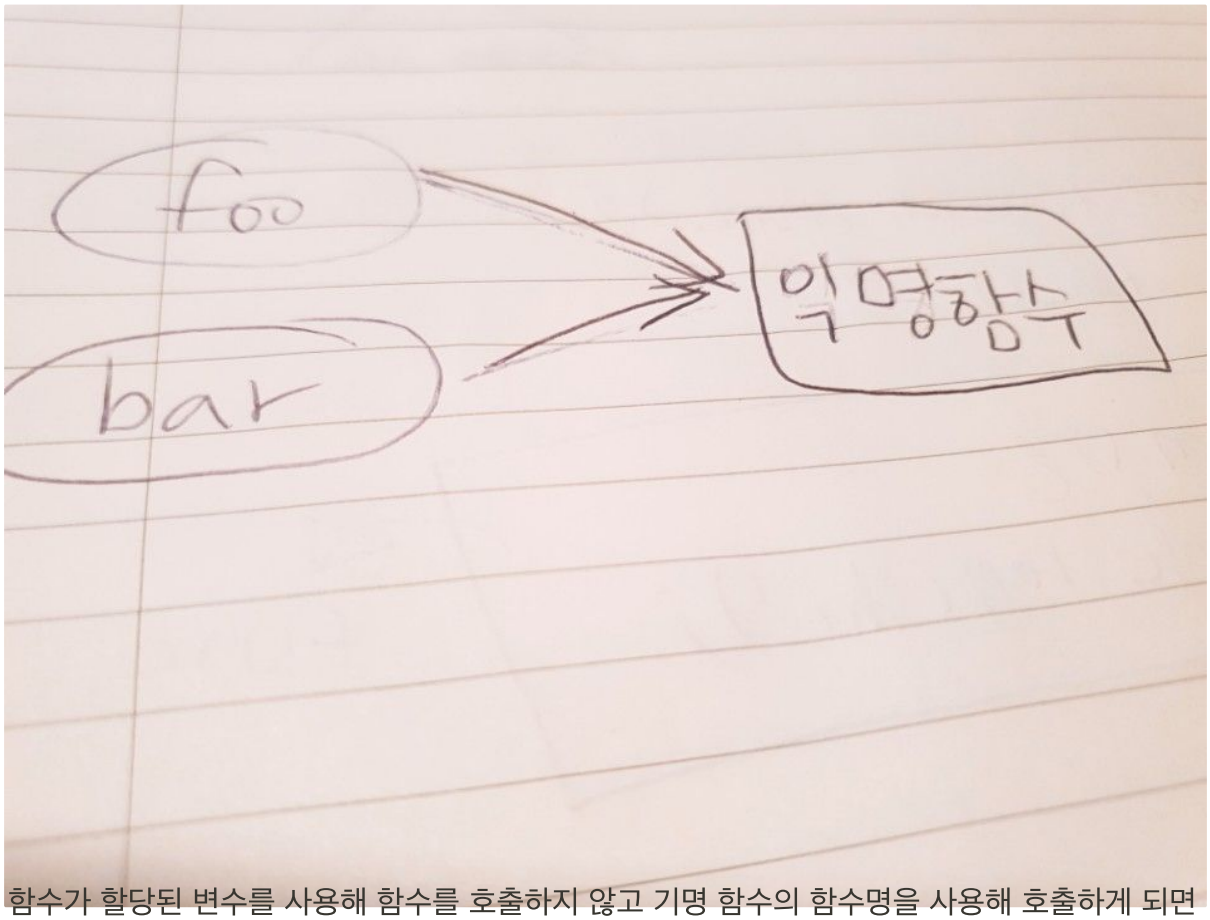
함수는 일급객체이기 때문에 변수에 할당할 수 있는 데 이 변수는 함수명이 아니라 할당된 함수를 가리키는 참조값을 저장하게 됨. 함수 호출 시 이 변수가 함수명 처럼 사용됨.

```
var foo = function(a,b) { return a+b; } var bar = foo;
console.log(foo(1,2)); // 3 console.log(bar(1,2)); // 3
```

Plain Text

foo -> 함수 리터럴로 생성한 함수를 참조하는 변수

bar -> 함수의 참조값을 가지므로 그 값을 그대로 할당할 수가 없음.



함수가 할당된 변수를 사용해 함수를 호출하지 않고 기명 함수의 함수명을 사용해 호출하게 되면 에러가 발생함.

이는 함수 표현식에서 사용한 함수명은 외부 코드에서 접근이 불가능하기 때문임(사실은 함수 선언식의 경우도 마찬가지임)

함수 표현식과 함수 선언식에서 사용한 함수명은 함수 몸체에서 재귀적 호출 하거나 자바스크립트디버거가 해당 함수를 구분할 수 있는 식별자 역할을 함.

함수선언식으로 정의한 add()의 경우, 함수명으로 호출할 수 있었는데 이는 자바스크립트 엔진에 의해 아래와 같은 함수 표현식으로 형태가 변경되기 때문임.

```
function add(a,b) { return a+b; }
```

Plain Text

> 함수선언식으로 정의한 add()

```
var add = function add(a,b) { return a+b; }
```

Plain Text

> 함수명과 함수 참조 값을 가진 변수명이 일치하므로 함수명으로 호출되는 듯 보이지만 사실은 변수명으로 호출된 것임.

> 결국 함수선언식도 함수 표현식과 동일하게 함수 리터럴 방식으로 정의되는 것임.

함수 선언식 : 끌어올려짐(호이스팅), 소스코드 상관없이 호출 가능. 이름이 있음. 함수 표현식 : 자바스크립트 엔진은 익명 함수 표현식 이름을 점점 더 잘 추론함. 그래서 이 방법을 더 많이 사용 해야함.

Function() 생성자 함수

함수표현식으로 함수를 정의할 때 함수 리터럴 방식을 사용함.

함수선언식도 내부적으로 자바스크립트 엔진이 기명함수 표현식으로 변환함으로 결국 함수 리터럴 방식을 사용함.

따라서 함수 선언식과 함수 표현식은 모두 함수 리터럴 방식으로 함수를 정의하는데, 이것은 결국 내장 함수 Function() 생성자 함수로 함수를 생성하는 것을 단순화 시킨 것임.

Function() 생성자 함수는 Function.prototype.constructor 프로퍼티로 접근할 수 있음.

Function() 생성자 함수로 함수를 생성하는 문법은 다음과 같음.

```
new Function(arg1, arg2, ... argN, functionBody)
```

Plain Text

```
var add = new Function('number','return number + nunber');
console.log(add(1)); // 2
```

Plain Text

Function() 생성자 함수로 함수를 생성하는 방식은 일반적으로 사용하지 않음. (왜냐하면 속도가 느리고, 문자열에 저장되므로 도구에서 접근할 수가 없음.)