

# 자바스크립트

함수 기본

# 목차

- 함수 기본
- 디버깅

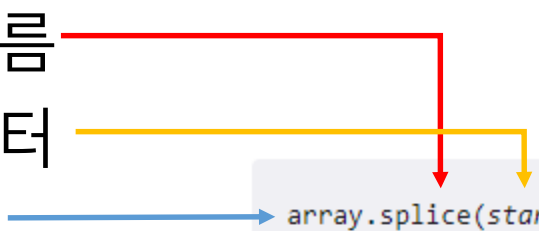
# 함수 기본

- 함수 원형 (Function Prototype)

- 함수이름

- 파라미터

- 리턴값



```
array.splice(start[, deleteCount[, item1[, item2[, ...]]]])
```

- 함수의 종류

- 선언함수 : 함수의 이름이 있음.

- 익명함수 : 함수의 이름이 없음. (확장)

# 함수 기본

- 익명함수
  - 형식 : function() {}
  - 사용예

```
<script>
  const func = function()
  {
    console.log('call function');
  }

  func();
</script>
```

참고 : 함수에 대한 MDN 문서

<https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide>

- 함수

함수 선언하기

함수 호출하기

함수 범위

클로저(Closures)

아규먼트 & 파라미터

Arrow 함수

# 함수 기본

- 선언적 함수
  - 형식 : `function function_name() { }`
  - 사용예

```
<script>  
  function func()  
  {  
    console.log('call function');  
  }  
  
  func();  
</script>
```

익명함수와 선언적 함수는 그 메커니즘이 완전히 같다.

# 함수 기본

- 함수의 파라미터와 리턴
  - 파라미터는 let, const등으로 표기하지 않음
  - 리턴형은 생략

```
<script>  
  function func(a, b)  
  {  
    return a + b;  
  }  
  
  const val = func(4, 5);  
  console.log(val);  
</script>
```

# 함수 기본

- 가변 파라미터
  - 가변적인 파라미터 개수를 처리하기 위한 기법
  - 형식 : `function function_name(...rest_parameter) { }`

```
<script>
  function func(...items)
  {
    console.log(items);
  }

  func(1, 2, 3);
  func(1, 2, 3, 4, 5);
</script>
```

```
▼ (3) [1, 2, 3] ⓘ
  0: 1
  1: 2
  2: 3
  length: 3
  ▶ [[Prototype]]: Array(0)

▼ (5) [1, 2, 3, 4, 5] ⓘ
  0: 1
  1: 2
  2: 3
  3: 4
  4: 5
  length: 5
  ▶ [[Prototype]]: Array(0)
```

# 함수 기본

- 가변 파라미터와 일반 파라미터의 조합
  - 가변파라미터를 뒤에 배치
  - 형식 : function function\_name(param#1, param#2, param#n, ...rest\_parameter) { }

```
<script>
  function func(a, b, ...items)
  {
    console.log(a);
    console.log(b);
    console.log(items);
  }

  func(1, 2);
  func(1, 2, 3);
  func(1, 2, 3, 4, 5);
</script>
```

```
1
2
▶ []
1
2
▼ [3] ⓘ
  0: 3
  length: 1
  ▶ [[Prototype]]: Array(0)
1
2
▼ (3) [3, 4, 5] ⓘ
  0: 3
  1: 4
  2: 5
  length: 3
  ▶ [[Prototype]]: Array(0)
>
```



# 함수 기본

- 가변 파라미터의 활용기법
  - 첫번째 파라미터가 어떤 타입인지를 검사하여 로직을 구성
  - 버블소트를 아래와 같이 호출하고자 할 때 가변 파라미터를 활용

```
let result;  
result = bubbleSort(43, 5, 1, 2, 6);  
console.log(result);
```

```
result = bubbleSort([43, 5, 1, 2, 6]);  
console.log(result);
```

---

# 함수 기본

- 가변 파라미터의 활용기법
  - 버블소트의 예

```
function bubbleSort(first, ...values)
{
    let source;
    let index;
    let temp;

    if ('number' == typeof(first))
    {
        source = values;
        source.unshift(first)
    }
    else if (Array.isArray(first))
    {
        source = first;
    }
}
```

```
    for (let i = 0; i < source.length - 1; i++)
    {
        index = i + 1;

        for (let k = 0; k < source.length - i - 1; k++)
        {
            if (source[i] > source[index])
            {
                temp = source[i];
                source[i] = source[index];
                source[index] = temp;
            }

            index++;
        }
    }

    return source;
```

# 함수 기본

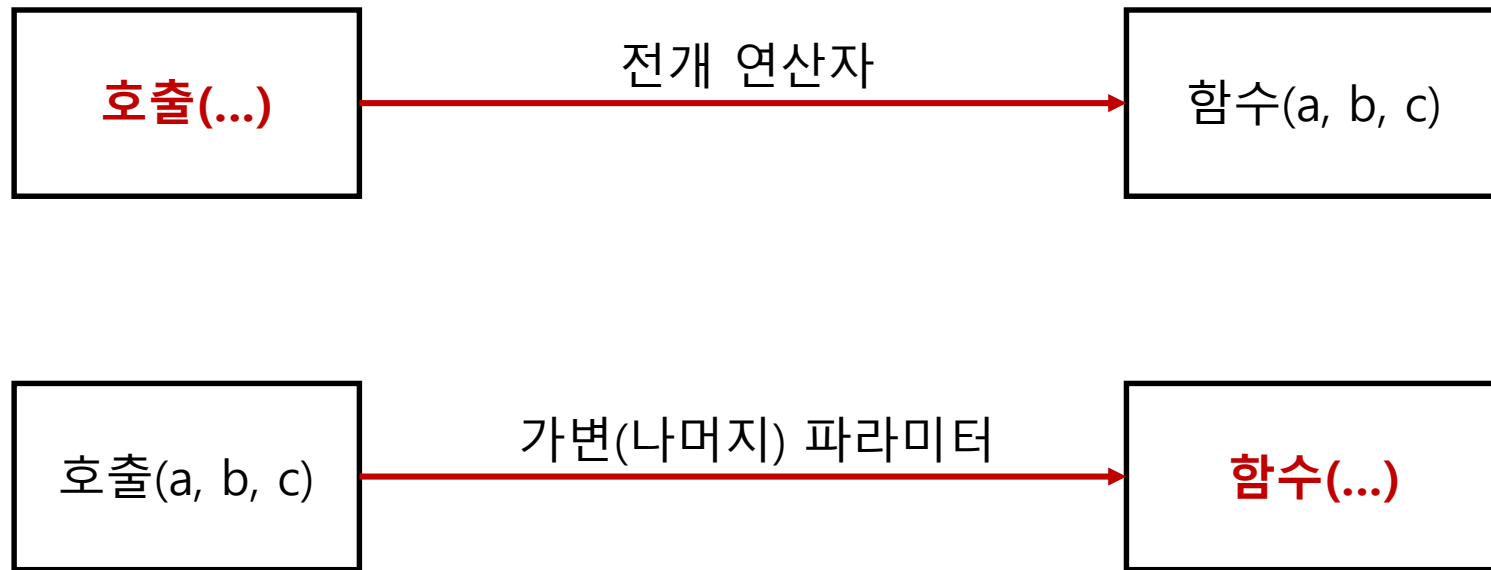
- 전개 연산자
  - 복수의 파라미터를 가진 함수에 배열을 전달하고자 할 때 사용하는 연산자.
  - 호출시 ...배열명 방식으로 호출.

```
const dummy = function(a, b, c)
{
  console.log(a);
  console.log(b);
  console.log(c);
}
```

```
const ar = [1, 2, 3];
dummy(ar[0], ar[1], ar[2]); // 일반적 호출
dummy(...ar)                // 전개연산자 이용
```

# 함수 기본

- 전개연산자와 가변 파라미터



# 함수기본

- 디폴트 파라미터

- 파라미터를 넣지 않은 경우, 미리 설정된 기본값을 사용하는 파라미터
- 과도한 디폴트 파라미터는 가독성(Readability)을 떨어뜨림

```
const def = function(x = 100, k)
{
  console.log('x=' + x);
  console.log('k=' + k);
}
```

```
def();
def(30);
def(30, 40);
```

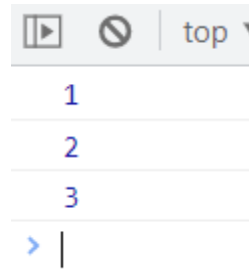
x=100
k=undefined
x=30
k=undefined
x=30
k=40

# 함수 기본

- 구형 ES5에서의 가변 파라미터
  - arguments 객체를 사용하여 파라미터 리스트를 구함.
  - arguments는 배열은 아니나 배열처럼 사용할 수 있음.

```
function dummy(a, b, c)
{
    console.log(arguments[0]);
    console.log(arguments[1]);
    console.log(arguments[2]);
}
```

```
dummy(1, 2, 3);
```



# 함수 기본

- 구형 ES5에서의 전개 연산자
  - `apply()`를 이용하여 전개 연산자와 비슷한 기능 구현

```
function dummy(a, b, c)
{
  console.log(a);
  console.log(b);
  console.log(c);
}
```

```
var ar = [1, 2, 3];
dummy.apply(null, ar);
```

# 함수 기본

- 재귀 호출

- 동일한 로직이 반복되는 구조를 처리하기 위해 함수 자신을 다시 호출하는 기법
- 일반 함수의 호출 방식과 동일
- $\text{fac}(n)$ 을  $n!$ 를 구하는 함수라 했을때  $5!$ 은 아래와 같다.

$$\begin{aligned} 5! &= 5 \times 4 \times 3 \times 2 \times 1 \\ &= 5 \times (4 \times 3 \times 2 \times 1) \\ &= 5 * 4! \\ 4! &= 4 * 3! \end{aligned}$$

$$\begin{aligned} 5! &= 5 * 4! \\ 4! &= 4 * 3! \\ 3! &= 3 * 2! \\ 2! &= 2 * 1! \\ 1! &= 1 \end{aligned}$$

$$\begin{aligned} \text{fac}(5) &= 5 * \text{fac}(4) \\ \text{fac}(4) &= 4 * \text{fac}(3) \\ \text{fac}(3) &= 3 * \text{fac}(2) \\ \text{fac}(2) &= 2 * \text{fac}(1) \\ \text{fac}(1) &= 1 \end{aligned}$$



# 함수 기본

- 재귀 호출
  - factorial의 재귀 호출 코드
  - 재귀 호출의 장점
    - 간결한코드
  - 재귀 호출의 단점
    - 과도한 stack사용
    - 일반 loop문으로 구현 가능

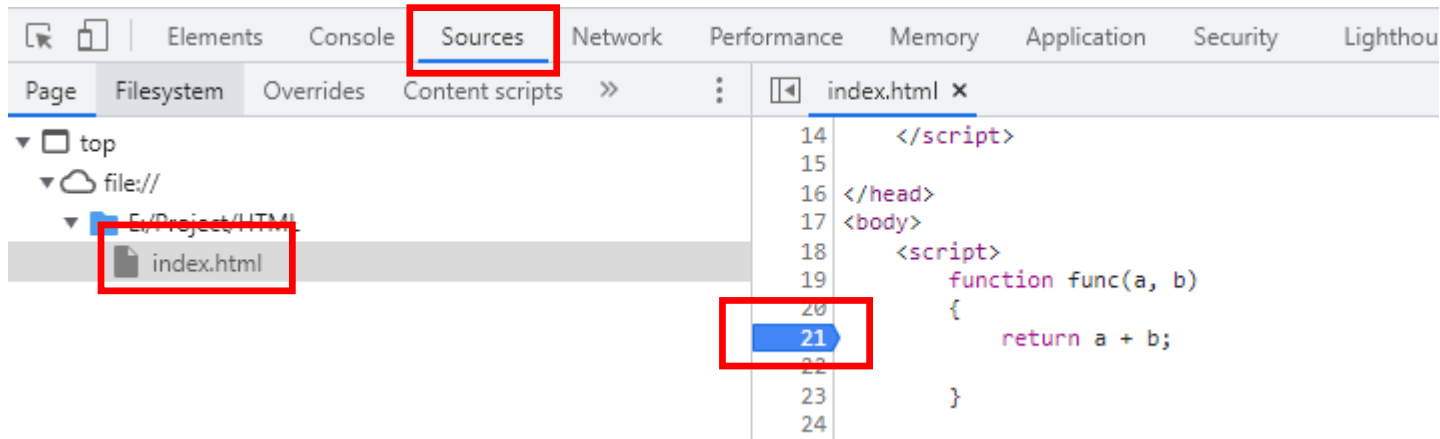
```
const fac = function(n)
{
    let total;

    if (n === 1)
    {
        return 1;
    }
    else
    {
        total = n * fac(n - 1)
    }

    return total;
}
```

# 디버깅

- Break-point
  - Sources 탭으로 이동후 html 선택
  - 해당 라인에서 클릭



# 디버깅

- Step into
  - Function 내부로 진입하여 수행
- Step over
  - Function 전체를 수행
- watch
  - 변수값을 조사
- Call stack
  - 현재 호출된 Function depth 표시

