

혼자 공부하는 자바스크립트



한국교통대학교 컴퓨터소프트웨어과
최일준 교수
cij0319@ut.ac.kr, cij0319@naver.com

이 책의 학습 목표

▪ CHAPTER 01: 자바스크립트 개요와 개발환경 설정

- 자바스크립트 개발환경 설치와 자바스크립트 프로그래밍 기본 용어 학습

▪ CHAPTER 02: 자료와 변수

- 프로그램 개발의 첫걸음. 자료형과 변수 학습

▪ CHAPTER 03: 조건문

- 프로그램의 흐름을 변화시키는 요소. 조건문의 종류를 알아보고 사용 방법을 이해

▪ CHAPTER 04: 반복문

- 배열의 개념과 문법을 익혀 while 반복문과 for 반복문 학습

▪ CHAPTER 05: 함수

- 다양한 형태의 함수를 만들기과 매개변수를 다루는 방법 이해

▪ CHAPTER 06: 객체

- 객체의 속성과 메소드, 생성, 관리하는 기본 문법 학습

▪ CHAPTER 07: 문서 객체 모델

- DOMContentLoaded 이벤트를 사용한 문서 객체 조작과 다양한 이벤트의 사용 방법 이해

▪ CHAPTER 08: 예외 처리

- 구문 오류와 예외를 구분하고, 예외 처리의 필요성과 예외를 강제로 발생시키는 방법을 이해

▪ CHAPTER 09: 클래스

- 객체 지향을 이해하고 클래스의 개념과 문법 학습

▪ CHAPTER 10: 리액트 라이브러리

- 리액트 라이브러리 사용 방법과 간단한 애플리케이션을 만드는 방법 학습

Contents

- CHAPTER 05: 함수

SECTION 5-1 함수의 기본 형태

SECTION 5-2 함수 고급



CHAPTER 05 함수

다양한 형태의 함수를 만들기과 매개변수를 다루는 방법 이해

함수 : 코드의 집합

SECTION 5-1 함수의 기본 형태(1)

- 함수를 사용하는 것 - 함수 호출, 즉, "함수를 호출한다."
- 함수를 호출할 때는 괄호 내부에 여러 가지 자료를 넣는데, 이러한 자료를 - 매개변수
- 함수를 호출해서 최종적으로 나오는 결과 - 리턴값

함수를 이용하면 좋은 점

1. 반복되는 코드를 한 번만 정의해 놓고 필요할 때마다 호출하므로 반복 작업을 피할 수 있음.
2. 긴 프로그램을 기능별로 나눠 여러 함수로 나누어 작성하면 모듈화로 전체 코드의 가독성이 좋아짐
3. 기능별(함수별)로 수정이 가능하므로 유지보수가 쉬움

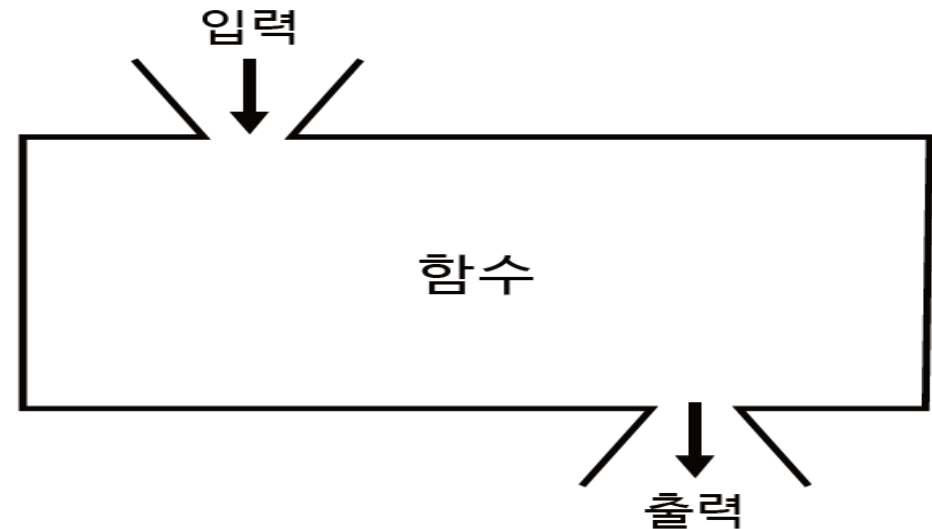


그림 5-1 함수 상자

SECTION 5-1 함수의 기본 형태(1)

- 익명 함수(anonymous function) : 이름이 붙어 있지 않은 함수

- 함수는 코드의 집합을 나타내는 자료형. 함수를 실행하면 코드를 한번에 묶어서 실행할 수 있으며, 필요할 때마다 호출하여 반복적으로 사용할 수 있음.

- f() {} f는 함수, {} 내부에 코드를 넣음

- 익명 함수 선언하기 (소스 코드 5-1-1.html)

```
01 <script>
02 // 변수를 생성합니다.
03 const 함수 = function () {
04   console.log('함수 내부의 코드입니다 ... 1')
05   console.log('함수 내부의 코드입니다 ... 2')
06   console.log('함수 내부의 코드입니다 ... 3')
07   console.log('')
08 }
09
10 // 함수를 호출합니다.
11 함수()
12 함수()
13
14 // 출력합니다.
15 console.log(typeof 함수)
16 console.log(함수)
17 </script>
```

함수의 자료형은 function 이며, 현재 코드에서 함수를 출력하면 f() {}라고 출력. 이 때 f는 함수를 나타냄. 함수를 출력했을 때 별다른 이름이 붙어있지 않은 것이 있는데, 이처럼 이름이 붙어있지 않는 함수 - 익명 함수(anonymous function)

우리가 만든 함수도 기존의 alert(), prompt() 함수처럼 호출할 수 있음

함수의 자료형을 확인

함수 자체도 단순한 자료이므로 출력 가능

SECTION 5-1 함수의 기본 형태(2)

- 익명 함수 (교재 198page)
 - 익명 함수 선언하기 (소스 코드 5-1-1.html) – 실행 결과

```
function f () {  
  console.log('함수 내부의 코드입니다 ... 1')  
  console.log('함수 내부의 코드입니다 ... 2')  
  console.log('함수 내부의 코드입니다 ... 3')  
  console.log('')  
}
```

실행 결과

함수 내부의 코드입니다 ... 1
함수 내부의 코드입니다 ... 2
함수 내부의 코드입니다 ... 3

11행에서 함수를 호출한 결과
여러 코드의 집합이 한 번에 실행

함수 내부의 코드입니다 ... 1
함수 내부의 코드입니다 ... 2
함수 내부의 코드입니다 ... 3

12행에서 두 번째 호출했으므로 코드 집합이 한 번 더 실행

함수의 자료형

함수를 출력한 결과

SECTION 5-1 함수의 기본 형태(3)

◦ 선언적 함수 (교재 199page)

- 선언적 함수는 이름을 붙여 생성

```
function 함수() {  
}
```

- 선언적 함수는 다음 코드와 같은 기능을 수행. (SECTION 5-2 좀 더 알아보기 참조)

```
let 함수 = function () { };
```

선언적 함수

이름이 있는 함수를 많이 사용함. 이렇게 생성한 함수를 선언적 함수라 함

```
1 // 익명 함수  
2 const f = function (매개변수, 매개변수) {  
3   | return 리턴값  
4 }  
5  
6 // 선언적 함수  
7 function f (매개변수, 매개변수) {  
8   | return 리턴값  
9 }
```


SECTION 5-1 함수의 기본 형태(4)

- 선언적 함수

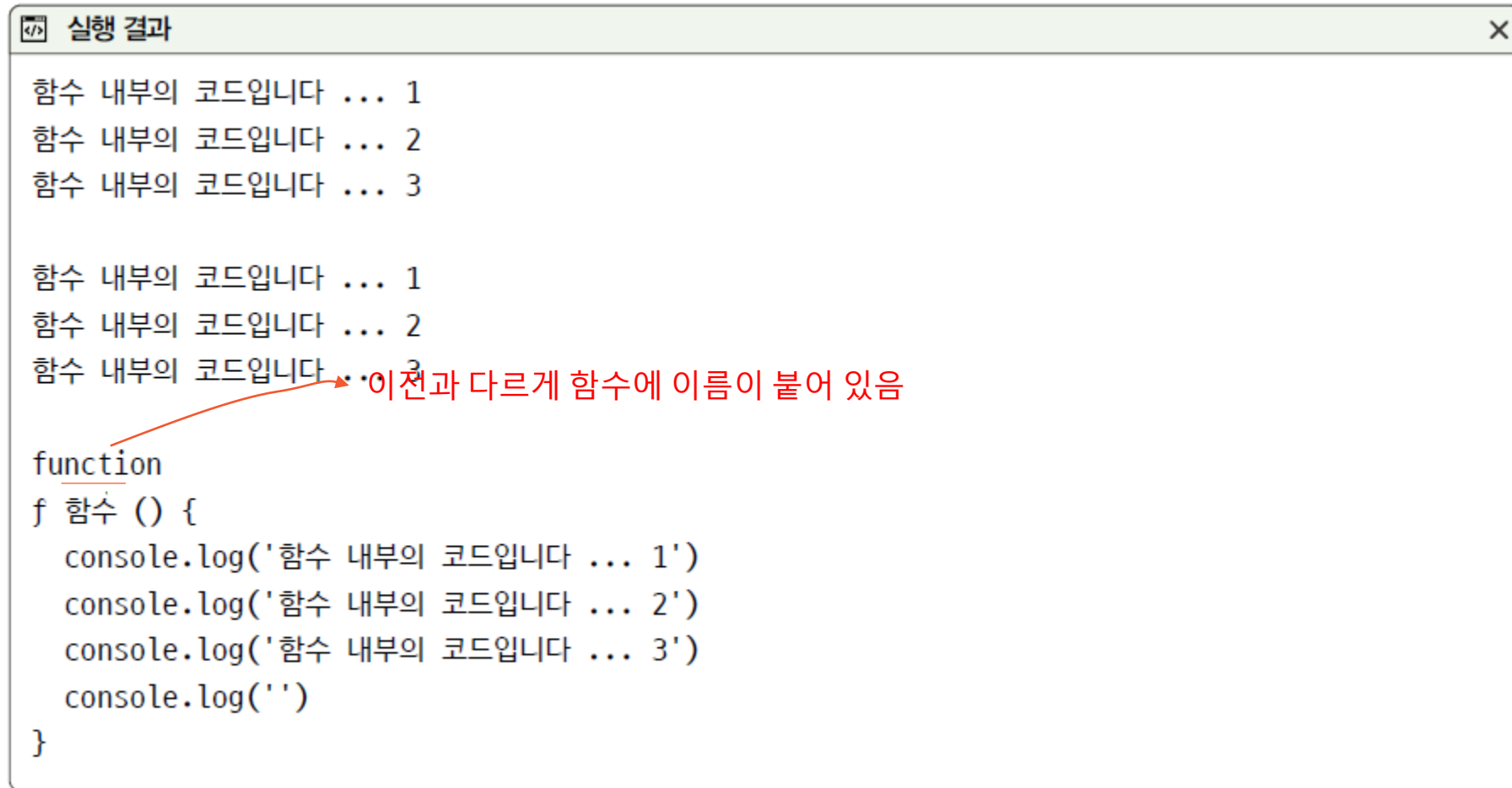
- 선언적 함수 선언하기 (소스 코드 5-1-2.html)

```
01 <script>
02  // 함수를 생성합니다.
03  function 함수 () {
04    console.log('함수 내부의 코드입니다 ... 1')
05    console.log('함수 내부의 코드입니다 ... 2')
06    console.log('함수 내부의 코드입니다 ... 3')
07    console.log('')
08  }
09
10  // 함수를 호출합니다.
11  함수()
12  함수()
13
14  // 출력합니다.
15  console.log(typeof 함수)
16  console.log(함수)
17 </script>
```

SECTION 5-1 함수의 기본 형태(5)

선언적 함수

- 선언적 함수 선언하기 (소스 코드 5-1-2.html) 실행 결과
- 익명함수와의 차이점은 함수를 출력했을 때 함수에 이름이 붙어있다는 것뿐임. 별다른 차이가 없음.



The screenshot shows a web browser window with a title bar that says "실행 결과" (Execution Results). The console displays the output of a function named "함수" (function). The output consists of three lines of text: "함수 내부의 코드입니다 ... 1", "함수 내부의 코드입니다 ... 2", and "함수 내부의 코드입니다 ... 3". Below this output, the source code for the function is shown. The function is defined as follows:

```
function
f 함수 () {
  console.log('함수 내부의 코드입니다 ... 1')
  console.log('함수 내부의 코드입니다 ... 2')
  console.log('함수 내부의 코드입니다 ... 3')
  console.log('')
}
```

A red arrow points from the text "이전과 다르게 함수에 이름이 붙어 있음" (Different from before, the function has a name attached) to the word "함수" in the function definition.

프로시저 형태의 함수

```
1 const f = function () {  
2   console.log('안녕하세요')  
3   console.log('안녕하세요')  
4   console.log('안녕하세요')  
5 }  
6  
7 alert()  
8 prompt()  
9 f()  
10  
11 함수()  
12 → 함수를 호출!  
13 = 함수의 본문을 실행
```

```
> const f = function () {  
  console.log('안녕하세요')  
  console.log('안녕하세요')  
  console.log('안녕하세요')  
}  
  
f()  
안녕하세요  
안녕하세요  
안녕하세요  
⏪ undefined
```

점프: 호출 위치에서 함수 본문으로 이동하는 것
리턴: 함수 본문에서 호출 위치로 나오는 것

```
1 const f = function () {  
2   // 점프!  
3   console.log('안녕하세요')  
4   console.log('안녕하세요')  
5   console.log('안녕하세요')  
6   // 함수 끝 → 원래 위치로 돌아감 = 리턴  
7 }  
8  
9 f()
```

```
1 // 프로시저 형태의 함수  
2 const f = function () {  
3   // 점프  
4   console.log('안녕하세요 + ${x}')  
5   console.log('안녕하세요 + ${x}')  
6   console.log('안녕하세요 + ${x}')  
7 }  
8  
9 const x = 10  
10 f() // 호출
```

```
1 const f = function () {  
2   // 점프!  
3   console.log('안녕하세요 + ${x}')  
4   console.log('안녕하세요 + ${x}')  
5   console.log('안녕하세요 + ${x}')  
6   // 함수 끝 → 원래 위치로 돌아감 = 리턴  
7 }  
8  
9 const x = 10  
10 f()
```

```
> const f = function () {  
  // 점프!  
  console.log('안녕하세요 + ${x}')  
  console.log('안녕하세요 + ${x}')  
  console.log('안녕하세요 + ${x}')  
  // 함수 끝 → 원래 위치로 돌아감 = 리턴  
}  
  
const x = 10  
f()  
안녕하세요 + 10  
안녕하세요 + 10  
안녕하세요 + 10  
⏪ undefined
```

수학적 함수

"순수 함수와 비순수 함수", "왜 프로시저가 함수로 발전했는가?", "스코프의 개념"을 한꺼번에 설명하려고 프로시저와 함수를 구분했는데, 난이도가 급상승 해버린 것 같아서 잘라버렸습니다(프로시저가 뭔지 기억 안 해주셔도 됩니다)....

```
1 // 수학적 함수
2 // f(x) = x + 5
3 // f(1) = 1 + 5 = 6
4 // f(2) = 2 + 5 = 7
5 const f = function (아무거나) {
6   return 아무거나 + 5
7 }
8
9 console.log(f(1))
10 console.log(f(2))
```

```
> const f = function (아무거나) {
  return 아무거나 + 5
}

console.log(f(1))
console.log(f(2))

6
7
< undefined
```

```
1 // 수학적 함수
2 // f(x) = x + 5
3 // f(1) = 1 + 5 = 6
4 // f(2) = 2 + 5 = 7
5 const f = function (아무거나 = 1) {
6   // 점프!
7   return 아무거나 + 5 // 리턴 값
8 }
9
10 console.log(f(1))
11 console.log(f(2))
```

```
1 // 수학적 함수
2 const f = function (x/* 매개변수 */) {
3   // 점프!
4   return x + 5 // 리턴값
5 }
6
7 console.log(f(1))
8 console.log(f(2))
```

함수의 괄호 안에 오는 것
= 매개변수

f(1)의 리턴값은 6이다.
f(2)의 리턴값은 7이다.

코드의 재사용

```
1 // 코드의 재사용
2 const sum = function (limit) {
3   let output = 0
4   for (let i = 1; i <= limit; i++) {
5     output += i
6   }
7   return output
8 }
9
10 console.log(`합은 ${sum(10)}`)
11 console.log(`합은 ${sum(20)}`)
12 console.log(`합은 ${sum(30)}`)
```

```
> // 코드의 재사용
const sum = function (limit) {
  let output = 0
  for (let i = 1; i <= limit; i++) {
    output += i
  }
  return output
}
```

```
console.log(`합은 ${sum(10)}`)
console.log(`합은 ${sum(20)}`)
console.log(`합은 ${sum(30)}`)
```

합은 55

합은 210

합은 465

< undefined

SECTION 5-1 함수의 기본 형태(6)

- 매개변수(함수를 호출할 때 괄호 안에 적는 것)와 리턴값(함수의 최종 결과)

- prompt() 함수의 매개변수와 리턴값

```
function prompt(message?:string,_default?:string): string
```

```
Prompt()
```

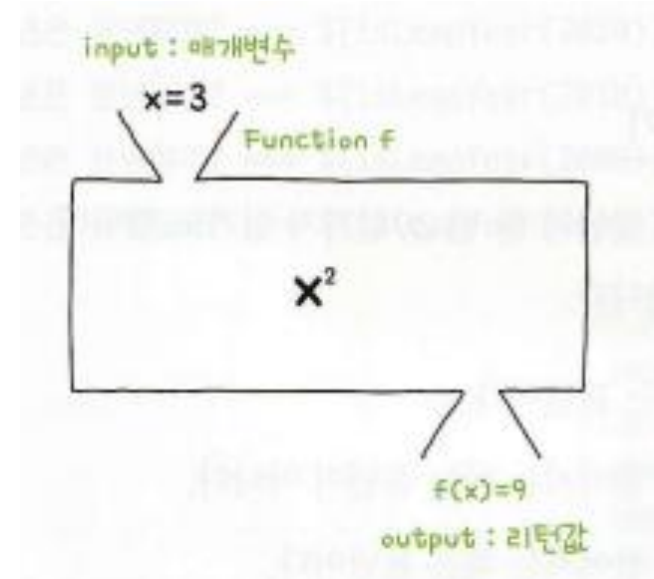
- 사용자 정의 함수의 매개변수와 리턴값

```
function 함수():void
```

```
함수()
```

- 매개변수와 리턴값을 갖는 함수

```
function 함수(매개변수, 매개변수, 매개변수) {  
  문장  
  문장  
  return 리턴값  
}
```

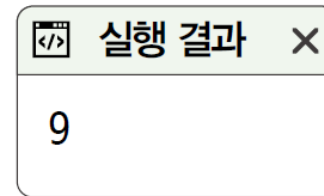


함수에 넣는 input이 매개변수이고, 결과로 나오는 output이 리턴값입니다. 리턴값은 함수 내부에 **return** 키워드를 입력하고 뒤에 값을 넣어서 생성합니다.

SECTION 5-1 함수의 기본 형태(7)

- 매개변수와 리턴값
 - 기본 형태의 함수 만들기 (소스 코드 5-1-3.html)
 - 매개변수로 x 를 넣으면 x^2 을 리턴하는 함수

```
01 <script>
02  // 함수를 선언합니다.
03  function f(x) {
04    return x * x
05  }
06
07  // 함수를 호출합니다.
08  console.log(f(3))
09 </script>
```



SECTION 5-1 함수의 기본 형태(8)

◦ 기본적인 함수 예제 (교재 202page)

▪ 윤년을 확인하는 함수 만들기

- 조건 1) 4로 나누어 떨어지는 해는 윤년
- 조건 2) 100으로 나누어 떨어지는 해는 윤년이 아님
- 조건 3) 400으로 나누어 떨어지는 해는 윤년

▪ 윤년인지 확인하는 함수 (소스 코드 5-1-4.html)

```
01 <script>
02 function isLeapYear(year) {
03   return (year % 4 === 0) && (year % 100 !== 0) || (year % 400 === 0)
04 }
05
06 console.log(`2020년은 윤년일까? === ${isLeapYear(2020)}`)
07 console.log(`2010년은 윤년일까? === ${isLeapYear(2010)}`)
08 console.log(`2000년은 윤년일까? === ${isLeapYear(2000)}`)
09 console.log(`1900년은 윤년일까? === ${isLeapYear(1900)}`)
10 </script>
```

윤년의 특징을 isLeapYear()라는 이름의 함수로 구현해봅시다. 숫자인 년도를 매개변수로 입력했을 때 윤년이면 true, 윤년이 아니면 false를 리턴해주면 됩니다. 간단한 함수이므로 직접 구현해보고 코드를 살펴보기 바랍니다.

윤년을 확인하는 함수 만들기

보통 2월은 28일까지 있지만 몇년에 한 번 29일까지 있기도 합니다. 이런 해를 윤년^{leap year}이라고 부르고 다음과 같은 특징이 있습니다.



실행 결과	
2020년은 윤년일까? ===	true
2010년은 윤년일까? ===	false
2000년은 윤년일까? ===	true
1900년은 윤년일까? ===	false

SECTION 5-1 함수의 기본 형태(8)

- 1 - 4로 나누어 떨어지는 해는 윤년이다.
- 2 - 하지만 100으로 나누어 떨어지는 해는 윤년이 아니다.
- 3 - 하지만 400으로 나누어 떨어지는 해는 윤년이다.

```
5 const isLeapYear = function (연도) {  
6     const 윤년이면 =  
7         (연도 % 4 === 0) // 윤년  
8         && (연도 % 100 !== 0) // 윤년  
9         || (연도 % 400 === 0) // 윤년  
10  
11     if (윤년이면) {  
12         return true  
13     } else {  
14         return false  
15     }  
16 }
```

```
1 const isLeapYear = function (연도) {  
2     return (연도 % 4 === 0)  
3         && (연도 % 100 !== 0)  
4         || (연도 % 400 === 0)  
5 }  
6  
7 console.log(`2020년은 윤년일까? === ${isLeapYear(2020)}`) // 윤년 → true  
8 console.log(`2010년은 윤년일까? === ${isLeapYear(2010)}`) // 윤년X → false  
9 console.log(`2000년은 윤년일까? === ${isLeapYear(2000)}`) // 윤년 → true  
10 console.log(`1900년은 윤년일까? === ${isLeapYear(1900)}`) // 윤년X → false
```

```
> const isLeapYear = function (연도) {  
    return (연도 % 4 === 0)  
        && (연도 % 100 !== 0)  
        || (연도 % 400 === 0)  
}  
  
console.log(`2020년은 윤년일까? === ${isLeapYear(2020)}`) // 윤년 → true  
console.log(`2010년은 윤년일까? === ${isLeapYear(2010)}`) // 윤년X → false  
console.log(`2000년은 윤년일까? === ${isLeapYear(2000)}`) // 윤년 → true  
console.log(`1900년은 윤년일까? === ${isLeapYear(1900)}`) // 윤년X → false
```

```
2020년은 윤년일까? === true  
2010년은 윤년일까? === false  
2000년은 윤년일까? === true  
1900년은 윤년일까? === false
```

SECTION 5-1 함수의 기본 형태(9)

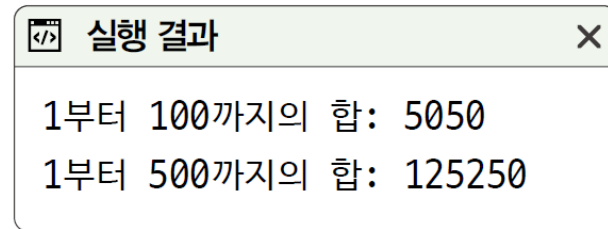
◦ 기본적인 함수 예제 (교재 203-204page)

▪ A부터 B까지 더하는 함수 만들기

- A부터 B까지라는 범위를 지정했을 때 범위 안에 있는 숫자를 모두 더하는 함수를 만들어 줌
- 예를들어) 1부터 5까지 더하라고 하면 매개변수로 1과 5를 입력하고, 리턴값으로 1부터 5까지 더한 값인 15가 나오면 됨
- A부터 b까지 곱하는 함수를 만들 때 주의할 점은 초깃값을 0으로 하면 어떤 수를 곱해도 0이 되므로, 어떤 수를 초깃값으로 할지 고민해야 함.

▪ A부터 b까지 더하는 함수 (소스 코드 5-1-5.html)

```
01 <script>
02  function sumAll(a, b) {
03    let output = 0
04    for (let i = a; i <= b; i++) {
05      output += i
06    }
07    return output
08  }
09
10  console.log(`1부터 100까지의 합: ${sumAll(1, 100)}`)
11  console.log(`1부터 500까지의 합: ${sumAll(1, 500)}`)
12 </script>
```




SECTION 5-1 함수의 기본 형태(10)

- 기본적인 함수 예제 (교재 204-205page)

- 최솟값을 구하는 함수 (소스 코드 5-1-6.html)

```
01 <script>
02 function min(array) {
03   let output = array[0]
04   for (const item of array) {
05     // 현재 output보다 더 작은 item이 있다면
06     if (output > item) {
07       // output의 값을 item으로 변경
08       output = item
09     }
10   }
11   return output
12 }
13
14 const testArray = [52, 273, 32, 103, 275, 24, 57]
15 console.log(`${testArray} 중에서`)
16 console.log(`최솟값 = ${min(testArray)}`)
17 </script>
```

```
1  const min = function (배열) {
2
3  }
4
5  console.log(min([52, 273, 32, 103, 275, 24, 57])) // 24
6
```

 실행 결과 ×

52,273,32,103,275,24,57 중에서
최솟값 = 24

최솟값을 구하는 함수 (소스 코드 5-1-6.html)

```
1 const min = function (배열) {  
2   let output = 배열[0]  
3   console.log(`처음 실행 때의 output = ${output}`)  
4  
5   for (const value of 배열) {  
6     console.log(`현재 비교 대상 ${output}과 ${value} 중에 작은 것은?`)  
7     if (output > value) {  
8       output = value  
9     }  
10    console.log(`= ${output}`)  
11  }  
12  return output  
13 }  
14  
15 console.log(min([52, 273, 32, 103, 275, 24, 57])) // 24
```

```
console.log(min([52, 273, 32, 103, 275, 24, 57])) // 24
```

처음 실행 때의 output = 52

현재 비교 대상 52과 52 중에 작은 것은?

= 52

현재 비교 대상 52과 273 중에 작은 것은?

= 52

현재 비교 대상 52과 32 중에 작은 것은?

= 32

현재 비교 대상 32과 103 중에 작은 것은?

= 32

현재 비교 대상 32과 275 중에 작은 것은?

= 32

현재 비교 대상 32과 24 중에 작은 것은?

= 24

현재 비교 대상 24과 57 중에 작은 것은?

= 24

24

최댓값을 구하는 함수 = **max**

나머지 매개변수(rest parameter) : 함수 호출

```
1 API
2 → Application Programming Interface(약속)
3 → __애플리케이션 프로그램을 만들 때의 약속__
4
5 ```javascript
6 alert('메시지')
7 console.log('메시지')
8
9 ```
```

```
1 const 함수 = function (...매개변수) {
2   console.log(매개변수)
3 }
4
5 함수()
6 함수(1)
7 함수(1, 2)
8 함수(1, 2, 3)
9 함수(1, 2, 3, 4)
```

```
> const 함수 = function (...매개변수) {
  console.log(매개변수)
}

함수()
함수(1)
함수(1, 2)
함수(1, 2, 3)
함수(1, 2, 3, 4)

▶ []
▶ [1]
▶ (2) [1, 2]
▶ (3) [1, 2, 3]
▶ (4) [1, 2, 3, 4]
< undefined
```

Web API, WinAPI, JavaScript API, Node API 등등
종류가 거대합니다.

나머지 매개변수(rest parameter) : 함수 호출

```
1 const 함수 = function (a, b, ...매개변수) {
2   console.log(a, b, 매개변수)
3 }
4
5 함수()
6 함수(1)
7 함수(1, 2)
8 함수(1, 2, 3)
9 함수(1, 2, 3, 4)
```

```
> const 함수 = function (a, b, ...매개변수) {
  console.log(a, b, 매개변수)
}

함수()
함수(1)
함수(1, 2)
함수(1, 2, 3)
함수(1, 2, 3, 4)
```

```
undefined undefined ▶ []
1 undefined ▶ []
1 2 ▶ []
1 2 ▶ [3]
1 2 ▶ (2) [3, 4]
< undefined
```

```
1 const 함수 = function (a, b, ...매개변수) {
2   console.log(a, b, 매개변수)
3 }
4
5 const 잘못된예 = function (...매개변수, a, b) {
6
7 }
8
9 함수()
10 함수(1)
11 함수(1, 2)
12 함수(1, 2, 3)
13 함수(1, 2, 3, 4)
```

```
> const 잘못된예 = function (...매개변수, a, b) { }
```

✖ Uncaught SyntaxError: Rest parameter must be last formal parameter

```
>
```

전개 연산자: 함수 호출

```
1 const 함수 = function (a, b, c) {  
2   console.log(a, b, c)  
3 }  
4  
5 const a = [1, 2, 3]  
6 함수(a[0], a[1], a[2])  
7 함수(...a)
```

```
> const 함수 = function (a, b, c) {  
  console.log(a, b, c)  
}
```

```
const a = [1, 2, 3]  
함수(a[0], a[1], a[2])  
함수(...a)
```

1 2 3

1 2 3

< undefined

```
1 const min = function (배열) {  
2   let output = 배열[0]  
3   for (const value of 배열) {  
4     if (output > value) {  
5       output = value  
6     }  
7   }  
8   return output  
9 }  
10 console.log(min([52, 273, 32, 103, 275, 24, 57]))  
11  
12 const min = function (...배열) {  
13   let output = 배열[0]  
14   for (const value of 배열) {  
15     if (output > value) {  
16       output = value  
17     }  
18   }  
19   return output  
20 }  
21 console.log(min(52, 273, 32, 103, 275, 24, 57))
```

```
1 const min = function (배열) {  
2   let output = 배열[0]  
3   for (const value of 배열) {  
4     if (output > value) {  
5       output = value  
6     }  
7   }  
8   return output  
9 }  
10 console.log(min([52, 273, 32, 103, 275, 24, 57]))  
11  
12 const min = function (...배열) {  
13   let output = 배열[0]  
14   for (const value of 배열) {  
15     if (output > value) {  
16       output = value  
17     }  
18   }  
19   return output  
20 }  
21 const a = [52, 273, 32, 103, 275, 24, 57]  
22 console.log(min(...a))
```

209page min함수 전개

정리

```
1  // 나머지 매개변수
2  const 함수 = function (...매개변수) {}
3  // → 배열!
4
5  // 전개 연산자
6  함수(...배열)
```


SECTION 5-1 함수의 기본 형태(11)

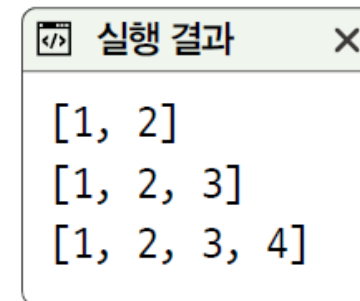
◦ 나머지 매개변수

- **가변 매개변수 함수**: 호출할 때 매개변수의 개수가 고정적이지 않은 함수
- 자바스크립트에서 이러한 함수를 구현할 때는 **나머지 매개변수(rest parameter)**라는 특이한 형태의 문법을 사용

`function 함수 이름(...나머지 매개변수) {}`

- 함수의 매개변수 앞에 **마침표 3개(...)**를 입력하면 매개변수들이 **배열**로 들어옴.
- 나머지 매개변수를 사용한 배열 만들기 (소스 코드 5-1-7.html)

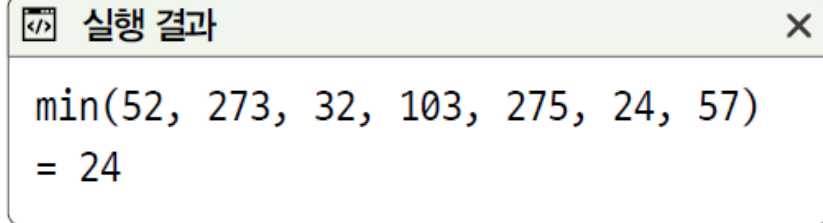
```
01 <script>
02 function sample(...items) {
03   console.log(items)
04 }
05
06 sample(1, 2)
07 sample(1, 2, 3)
08 sample(1, 2, 3, 4)
09 </script>
```



SECTION 5-1 함수의 기본 형태(12)

- 나머지 매개변수
 - 나머지 매개변수를 사용한 min() 함수 (소스 코드 5-1-8.html)

```
01 <script>
02 // 나머지 매개변수를 사용한 함수 만들기
03 function min(...items) {
04   // 매개변수 items는 배열처럼 사용합니다.
05   let output = items[0]
06   for (const item of items) {
07     if (output > item) {
08       output = item
09     }
10   }
11   return output
12 }
13
14 // 함수 호출하기
15 console.log('min(52, 273, 32, 103, 275, 24, 57)')
16 console.log(`= ${min(52, 273, 32, 103, 275, 24, 57)}`)
17 </script>
```



실행 결과

```
min(52, 273, 32, 103, 275, 24, 57)
= 24
```

SECTION 5-1 함수의 기본 형태(13)

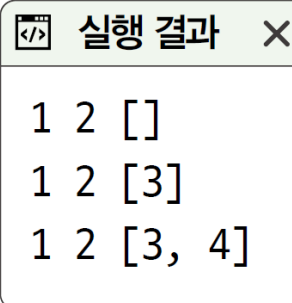
◦ 나머지 매개변수

- 나머지 매개변수와 일반 매개변수 조합하기

```
function 함수 이름(매개변수, 매개변수, ...나머지 매개변수) {}
```

- 나머지 매개변수와 일반 매개변수를 갖는 함수, (a, b, ...c)를 매개변수로 갖는 함수, (소스 코드 5-1-9.html)
 - 함수를 호출할 때 매개변수 a, b가 먼저 들어가고, 남은 것들은 모두 c에 배열 형태로 들어감.

```
01 <script>
02 function sample(a, b, ...c) {
03   console.log(a, b, c)
04 }
05
06 sample(1, 2)
07 sample(1, 2, 3)
08 sample(1, 2, 3, 4)
09 </script>
```



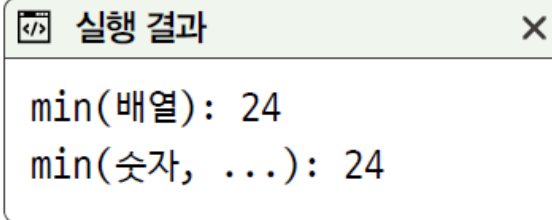
실행 결과		
1	2	[]
1	2	[3]
1	2	[3, 4]

- min(배열) 형태로 매개변수에 배열을 넣으면 배열 내부에서 최솟값을 찾아주는 함수
- min(숫자, 숫자, ...) 형태로 매개변수를 넣으면 숫자들 중에서 최솟값을 찾아주는 함수

SECTION 5-1 함수의 기본 형태(14)

- 나머지 매개변수, 매개변수로 들어온 자료형이 배열인지 숫자인지 확인 – **typeof 연산자**
 - Array.isArray() 메소드
 - 매개변수로 들어온 자료형이 배열인지 숫자인지 확인. **typeof(배열) == 'object'** 와 같은 형태로 확인 가능
 - 매개변수의 자료형에 따라 다르게 작동하는 min() 함수 (소스 코드 5-1-10.html)

```
01 <script>
02 function min(first, ...rests) {
03   // 변수 선언하기
04   let output
05   let items
06
07   // 매개변수의 자료형에 따라 조건 분기하기
08   if (Array.isArray(first)) {
09     output = first[0]
10     items = first
11   } else if (typeof(first) === 'number') {
12     output = first
13     items = rests
14   }
15
16   // 이전 절에서 살펴보았던 최솟값 구하는 공식
17   for (const item of items) {
18     if (output > item) {
19       output = item
20     }
21   }
22   return output
23 }
24
25 console.log(`min(배열): ${min([52, 273, 32, 103, 275, 24, 57])}`)
26 console.log(`min(숫자, ...): ${min(52, 273, 32, 103, 275, 24, 57)}`)
27 </script>
```



실행 결과

```
min(배열): 24
min(숫자, ...): 24
```

SECTION 5-1 함수의 기본 형태(15)

◦ 나머지 매개변수

▪ Array.isArray() 메소드

- 매개변수로 들어온 자료형이 배열인지 숫자인지 확인
- 매개변수의 자료형에 따라 다르게 작동하는 min() 함수 (소스 코드 5-1-10.html)

```
01 <script>
02 function min(first, ...rests) {
03   // 변수 선언하기
04   let output
05   let items
06
07   // 매개변수의 자료형에 따라 조건 분기하기
08   if (Array.isArray(first)) {
09     output = first[0]
10     items = first
```

어떤 자료가 배열인지 확인할 때는 Array.isArray() 메소드를 사용 (일반적인 typeof 연산자로는 배열을 확인할 수 없음)

중간 과정 생략

```
24
25 console.log(`min(배열): ${min([52, 273, 32, 103, 275, 24, 57])}`)
26 console.log(`min(숫자, ...): ${min(52, 273, 32, 103, 275, 24, 57)}`)
27 </script>
```

실행 결과

```
min(배열): 24
min(숫자, ...): 24
```

SECTION 5-1 함수의 기본 형태(16)

◦ 나머지 매개변수

- 전개 연산자(spread operator): 배열을 전개해서 함수의 매개변수로 전달
- 전개 연산자는 배열 앞에 마침표 3개(...)를 붙이는 형태로 사용
 함수 이름(...배열)
- 전개 연산자의 활용 (소스 코드 5-1-11.html)
- 전개 연산자를 사용해서 매개변수를 전달하고, 전달받은 매개변수를 단순히 나머지 매개변수로 출력하는 예제

```
01 <script>
02 // 단순히 매개변수를 모두 출력하는 함수
03 function sample(...items) {
04   console.log(items)
05 }
06
07 // 전개 연산자 사용 여부 비교하기
08 const array = [1, 2, 3, 4]
09
10 console.log('# 전개 연산자를 사용하지 않은 경우')
11 sample(array)
12 console.log('# 전개 연산자를 사용한 경우')
13 sample(...array)
14 </script>
```

실행 결과

전개 연산자를 사용하지 않은 경우
[Array(4)] → 4개의 요소가 있는 배열이 들어옴

전개 연산자를 사용한 경우
[1, 2, 3, 4] → 숫자가 하나하나 들어옴

가독성 이란?

- 가독성이 중요한 이유(1) – 기업 입장의 비용 절감,
: 최대한 저렴한 개발자를 고용해도 코드를 읽고 이해하는데 문제가 없게 하는 것이 중요
- 가독성이 중요한 이유(2) – 프로그램이 너무 복잡해져서....
: 오류를 하나 찾는 데도 많은 시간이 걸리는 문제 야기.....
: 그래서 현대적인 개발에서는 “단위 테스트(unit test)”가 중요해졌음.
: **단위 테스트(unit test)** – 코드를 작성하고 빠르게 문제가 있는지 없는지 테스트해주는 과정을 자동화 한 것.
- 가독성이 중요한 이유(3) – 프로그램을 도와주는 시스템과 지원도구의 발전,
: 프로그램 실행기, 컴파일러, 개발환경 등 최적화 실행해 줌

SECTION 5-1 함수의 기본 형태(17)

- 기본 매개변수 (교재 212page) – 가독성을 위한 기능
 - 여러 번 반복 입력되는 매개변수에 기본값을 지정하여 사용
 - 기본 매개변수는 오른쪽 매개변수에 사용
- 함수 이름(매개변수, 매개변수=기본값, 매개변수=기본값)

- 매개변수로 시급과 시간을 입력받아 급여를 계산하는 함수 연습
 - 함수 이름: earnings
 - 매개변수: name(이름), wage(시급), hours(시간)
 - 함수의 역할: 이름, 시급, 시간을 출력하고, 시급과 시간을 곱한 최종 급여 출력
- 만약 wage와 hours를 입력하지 않고 실행하면 wage에 최저 임금이 들어가고, hours에 법정근로시간 1주일 40시간이 기본 매개변수로 입력
- 다음 페이지의 소스 코드 5-1-12.html 위의 작성 예임

SECTION 5-1 함수의 기본 형태(18)

◦ 기본 매개변수

- 기본 매개변수의 활용 (소스 코드 5-1-12.html)

```
01 <script>
02 function earnings (name, wage=8590, hours=40) {
03   console.log(`# ${name} 님의 급여 정보`)
04   console.log(`- 시급: ${wage}원`)
05   console.log(`- 근무 시간: ${hours}시간`)
06   console.log(`- 급여: ${wage * hours}원`)
07   console.log("")
08 }
09
10 // 최저 임금으로 최대한 일하는 경우
11 earnings('구름')
12
13 // 시급 1만원으로 최대한 일하는 경우
14 earnings('별', 10000)
15
16 // 시급 1만원으로 52시간 일한 경우
17 earnings('인성', 10000, 52)
18 </script>
```

실행 결과

```
# 구름 님의 급여 정보
- 시급: 8590원
- 근무 시간: 40시간
- 급여: 343600원

# 별 님의 급여 정보
- 시급: 10000원
- 근무 시간: 40시간
- 급여: 400000원

# 인성 님의 급여 정보
- 시급: 10000원
- 근무 시간: 52시간
- 급여: 520000원
```

SECTION 5-1 함수의 기본 형태(19)

◦ 기본 매개변수

- 기본 매개변수는 값이라면 무엇이든 넣을 수 있음
- **isLeapYear() 함수를 수정해서 매개변수를 입력하지 않은 경우 자동으로 올해가 윤년인지 확인하는 함수로 변경**
- 기본 매개변수를 추가한 윤년 함수 (소스 코드 5-1-13.html)

```
01 <script>
02 function isLeapYear(year=new Date().getFullYear()) {
03   console.log(`매개변수 year: ${year}`)
04   return (year % 4 === 0) && (year % 100 !== 0) || (year % 400 === 0)
05 }
06
07 console.log(`올해는 윤년일까? === ${isLeapYear()}`)
08 </script>
```

→ 기본값을 이렇게 넣을 수도 있음

실행 결과

```
매개변수 year: 2020
올해는 윤년일까? === true
```

기본 매개 변수

```
1 const isLeapYear = function (연도) {
2   return (연도 % 4 === 0)
3     && (연도 % 100 !== 0)
4     || (연도 % 400 === 0)
5 }
6
7 isLeapYear() // 올해가 윤년인지 알려준다!
```

```
> const isLeapYear = function (연도) {
  console.log(`연도: ${연도}`)
  console.log(`단순 식 계산 결과: ${연도 % 4}`)
  return (연도 % 4 === 0)
    && (연도 % 100 !== 0)
    || (연도 % 400 === 0)
}

console.log(isLeapYear()) // 올해가 윤년인지 알려준다
연도: undefined
단순 식 계산 결과: NaN
false
< undefined
```

```
1 const isLeapYear = function (연도 = new Date().getFullYear()) {
2   console.log(`구하고 있는 연도: ${연도}`)
3   return (연도 % 4 === 0)
4     && (연도 % 100 !== 0)
5     || (연도 % 400 === 0)
6 }
7
8 console.log(isLeapYear())
```

```
> const isLeapYear = function (연도 = new Date().getFullYear()) {
  console.log(`구하고 있는 연도: ${연도}`)
  return (연도 % 4 === 0)
    && (연도 % 100 !== 0)
    || (연도 % 400 === 0)
}

console.log(isLeapYear())
구하고 있는 연도: 2021
false
< undefined
```

과거의 기본 매개 변수 코드 예

```
1 const isLeapYear = function (연도) {  
2   // if (typeof(연도) === 'undefined') {  
3   //   연도 = new Date().getFullYear()  
4   // }  
5   연도 = typeof(연도) === 'undefined' ? new Date().getFullYear() : 연도  
6  
7   console.log(`연도: ${연도}`)  
8  
9   return (연도 % 4 === 0  
10    && (연도 % 100 !== 0)  
11    || (연도 % 400 === 0)  
12 }  
13  
14 console.log(isLeapYear()) // 올해가 윤년인지 알려준다  
15
```

```
1 const isLeapYear = function (연도) {  
2   // if (typeof(연도) === 'undefined') {  
3   //   연도 = new Date().getFullYear()  
4   // }  
5   연도 = typeof(연도) === 'undefined' ? new Date().getFullYear() : 연도  
6   연도 = typeof(연도) !== 'undefined' ? 연도 : new Date().getFullYear()  
7  
8   console.log(`연도: ${연도}`)  
9  
10  return (연도 % 4 === 0  
11    && (연도 % 100 !== 0)  
12    || (연도 % 400 === 0)  
13 }  
14  
15 console.log(isLeapYear()) // 올해가 윤년인지 알려준다  
16
```

```
> const isLeapYear = function (연도) {  
  // if (typeof(연도) === 'undefined') {  
  //   연도 = new Date().getFullYear()  
  // }  
  
  // 연도 = typeof(연도) === 'undefined' ? new Date().getFullYear() : 연도  
  연도 = typeof(연도) !== 'undefined' ? 연도 : new Date().getFullYear()  
  
  console.log(`연도: ${연도}`)  
  
  return (연도 % 4 === 0  
    && (연도 % 100 !== 0)  
    || (연도 % 400 === 0)  
  )  
  
  console.log(isLeapYear()) // 올해가 윤년인지 알려준다  
연도: 2021  
false  
< undefined
```

과거의 기본 매개 변수 코드의 또 다른 예

```
1  const isLeapYear = function (연도) {  
2    // if (typeof(연도) === 'undefined') {  
3    //   연도 = new Date().getFullYear()  
4    // }  
5  
6    // 연도 = typeof(연도) === 'undefined' ? new Date().getFullYear() : 연도  
7    // 연도 = typeof(연도) !== 'undefined' ? 연도 : new Date().getFullYear()  
8  
9    연도 = 연도 || new Date().getFullYear()  
10   console.log(`연도: ${연도}`)  
11  
12   return (연도 % 4 === 0  
13     && (연도 % 100 !== 0)  
14     || (연도 % 400 === 0))  
15 }  
16  
17 console.log(isLeapYear()) // 올해가 윤년인지 알려준다  
18
```

[좀 더 알아보기①] 구 버전 자바스크립트에서 가변 매개변수 함수 구현하기

- 구 버전의 자바스크립트에서 가변 매개변수 함수를 구현할 때는 배열 내부에서 사용할 수 있는 특수한 변수인 arguments를 활용
- arguments를 사용한 가변 매개변수 함수 (소스 코드 5-1-14.html) - **for in, for of 반복문에서는 사용하지 않음**

```
01 <script>
02 function sample() {
03   console.log(arguments)
04   for (let i = 0; i < arguments.length; i++) {
05     console.log(`${i}번째 요소: ${arguments[i]}`)
06   }
07 }
08
09 sample(1, 2)
10 sample(1, 2, 3)
11 sample(1, 2, 3, 4)
12 </script>
```

- 교재 214page

실행 결과

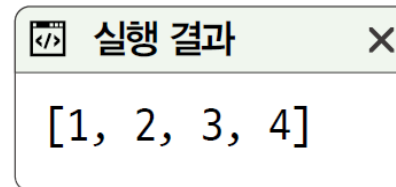
```
Arguments(2) [1, 2, callee: f, Symbol(Symbol.iterator): f]
0번째 요소: 1
1번째 요소: 2
Arguments(3) [1, 2, 3, callee: f, Symbol(Symbol.iterator): f]
0번째 요소: 1
1번째 요소: 2
2번째 요소: 3
Arguments(4) [1, 2, 3, 4, callee: f, Symbol(Symbol.iterator): f]
0번째 요소: 1
1번째 요소: 2
2번째 요소: 3
3번째 요소: 4
```

구 버전의 자바스크립트에서 가변 매개변수 함수를 구현할 때는 배열 내부에서 사용할 수 있는 특수한 변수인 arguments를 활용합니다. arguments는 매개변수와 관련된 여러 정보를 확인할 수 있고 배열과 비슷하게 사용할 수 있습니다.

[좀 더 알아보기②] 구 버전 자바스크립트에서 전개 연산자 구현하기

- 전개 연산자는 최신 버전의 자바스크립트에 추가된 기능
구 버전의 자바스크립트에서는 다음과 같이 `apply()` 함수를 사용한 굉장히 특이한 패턴의 코드를 사용
- 전개 연산자가 없던 구 버전에서 `apply()` 함수 사용하기 (소스 코드 5-1-15.html)

```
01 <script>
02 // 단순히 매개변수를 모두 출력하는 함수
03 function sample(...items) {
04   console.log(items)
05 }
06
07 // 전개 연산자 사용 여부 비교하기
08 const array = [1, 2, 3, 4]
09 console.log(sample.apply(null, array))
10 </script>
```



전개 연산자는 최신 버전의 자바스크립트에 추가된 기능입니다. 구 버전의 자바스크립트에서는 다음과 같이 `apply()` 함수를 사용한 굉장히 특이한 패턴의 코드를 사용했습니다. 이 코드는 용도를 모르면 아예 이해할 수가 없습니다.

[좀 더 알아보기③] 구 버전 자바스크립트에서 기본 매개변수 구현하기

- 함수의 매개변수에 바로 값을 입력하는 기본 매개변수는 최신 자바스크립트에서 추가된 기능
- 구 버전의 자바스크립트에서는 일반적으로 다음과 같은 코드를 사용해서 기본 매개변수를 구현

```
function earnings (wage, hours) {  
  wage = typeof(wage) !== undefined ? wage : 8590  
  hours = typeof(hours) !== undefined ? hours : 52  
  return wage * hours  
}
```

- 매개변수로 들어오는 값이 **false 또는 false로 변환되는 값(0, 빈 문자열 등)**이 아니라는 게 확실하다면 다음과 같이 짧은 조건문을 사용해서 기본 매개변수를 구현

```
function earnings (wage, hours) {  
  wage = wage || 8590  
  hours = hours || 52  
  return wage * hours  
}
```

- 인터넷에는 구 버전의 자바스크립트로 작성된 참고 자료가 많음. 구 버전에 대한 것을 기억해두면 다른 사람이 작성한 코드가 어떤 목적으로 작성했는지 쉽게 알 수 있음.

[마무리①]

◦ 7가지 키워드로 정리하는 핵심 포인트

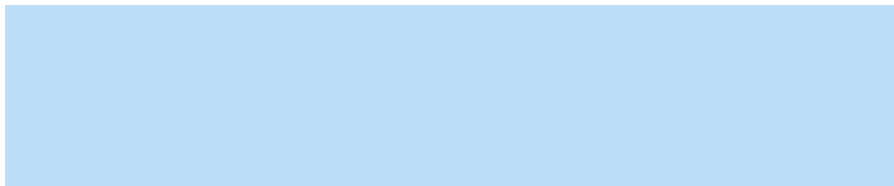
- **익명 함수**란 이름이 없는 함수로 `function () {}` 형태로 만들
- **선언적 함수**란 이름이 있는 함수로 `function 함수 이름 () {}` 형태로 만들
- 함수의 괄호 안에 넣는 변수를 **매개변수**라고 합니다. 매개변수를 통해 함수는 외부의 정보를 입력 받을 수 있음
- 함수의 최종적인 결과를 **리턴값**이라고 합니다. 함수 내부에 `return` 키워드를 입력하고 뒤에 값을 넣어서 생성
- **가변 매개변수 함수**란 매개변수의 개수가 고정되어 있지 않은 함수를 의미. 나머지 매개변수(...)를 활용해서 만들
- **전개 연산자**란 배열을 함수의 매개변수로서 전개하고 싶을 때 사용
- **기본 매개변수**란 매개변수에 기본값이 들어가게 하고 싶을 때 사용하는 매개변수

[마무리②]

◦ 확인 문제

1. A부터 B까지 범위를 지정했을 때 범위 안의 숫자를 모두 곱하는 함수를 만들어보기

```
<script>
```



```
console.log(multiplyAll(1, 2))
```

```
console.log(multiplyAll(1, 3))
```

```
</script>
```

```
1  const multiplyAll = function (start, end) {  
2    let output = 1  
3    for (let i = start; i <= end; i++) {  
4      output *= i  
5    }  
6    return output  
7  }  
8  
9  console.log(multiplyAll(1, 2))  
10 console.log(multiplyAll(1, 3))
```



실행 결과



2

6

```
> const multiplyAll = function (start, end) {  
  let output = 1  
  for (let i = start; i <= end; i++) {  
    output *= i  
  }  
  return output  
}
```

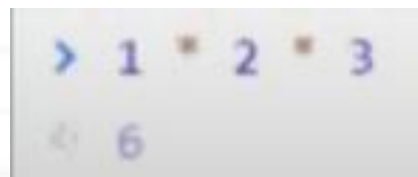
```
console.log(multiplyAll(1, 2))
```

```
console.log(multiplyAll(1, 3))
```

2

6

< undefined



[마무리②]

◦ 확인 문제

2. 다음 과정에 따라 최대값을 찾는 max() 함수를 만들어보기

- 매개변수로 max([1, 2, 3, 4])와 같은 배열을 받는 max() 함수 만들기

①

```
<script>
```

```
const max =          {  
  let output = array[0]
```

```
                                
```

```
  return output
```

```
}
```

```
console.log(max([1, 2, 3, 4]))
```

```
</script>
```

```
> const max = function (배열) {  
  let output = 배열[0]  
  for (const 값 of 배열) {  
    if (output < 값) {  
      output = 값  
    }  
  }  
  return output  
}  
  
console.log(max([1, 2, 3, 4])) // 4  
  
4  
← undefined
```

```
1  const max = function (배열) {  
2    let output = 배열[0]  
3    for (const 값 of 배열) {  
4      if (output < 값) {  
5        output = 값  
6      }  
7    }  
8    return output  
9  }  
10 |  
11 console.log(max([1, 2, 3, 4])) // 4
```

[마무리③]

◦ 확인 문제

2. 다음 과정에 따라 최대값을 찾는 max() 함수를 만들어보기

- 매개변수로 max(1, 2, 3, 4)와 같이 숫자를 배열을 받는 max() 함수 만들기

②

```
<script>
const max =                      {
  let output = array[0]
  
  return output
}
console.log(max(1, 2, 3, 4))
</script>
```

```
> const max = function (...배열) {
  let output = 배열[0]
  for (const 값 of 배열) {
    if (output < 값) {
      output = 값
    }
  }
  return output
}

console.log(max(1, 2, 3, 4)) // 4
4
< undefined
```

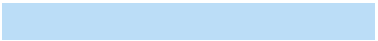
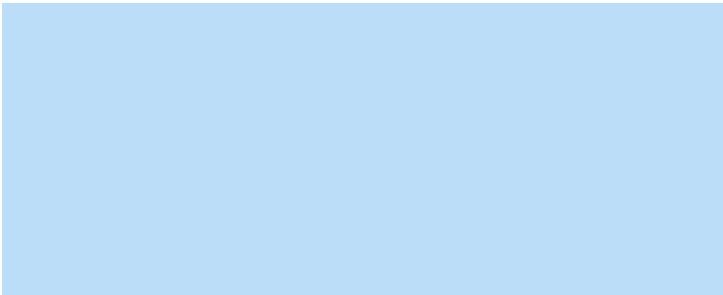
[마무리④]

◦ 확인 문제

2. 다음 과정에 따라 최대값을 찾는 max() 함수를 만들어보기

- max([1, 2, 3, 4]) 형태와 max(1, 2, 3, 4) 형태를 모두 입력할 수 있는 max() 함수 만들기

③

```
<script>
const max =  {
  let output
  let items
  
  return output
}
console.log(`max(배열): ${max([1,2,3,4])}`)
console.log(`max(숫자, ...): ${max(1,2,3,4)}`)
</script>
```

```
1  const max = function (첫번째요소, ...나머지) {
2    if (Array.isArray(첫번째요소)) {
3      let output = 첫번째요소[0]
4      for (const 값 of 첫번째요소) {
5        if (output < 값) {
6          output = 값
7        }
8      }
9      return output
10   } else {
11     let output = 첫번째요소
12     for (const 값 of 나머지) {
13       if (output < 값) {
14         output = 값
15       }
16     }
17     return output
18   }
19 }
20
21 console.log(max([1, 2, 3, 4])) // 4
22 console.log(max(1, 2, 3, 4))  // 4
```

앞 페이지 문제 설명

```
1 const max = function (첫번째요소, ...나머지) {  
2   if (Array.isArray(첫번째요소)) {  
3     let output = 첫번째요소[0]  
4     for (const 값 of 첫번째요소) {  
5       if (output < 값) {  
6         output = 값  
7       }  
8     }  
9     return output  
10  } else {  
11    let output = 첫번째요소  
12    for (const 값 of 나머지) {  
13      if (output < 값) {  
14        output = 값  
15      }  
16    }  
17    return output  
18  }  
19 }  
20  
21 console.log(max([1, 2, 3, 4])) // 4  
22 console.log(max(1, 2, 3, 4))  // 4
```

max(1, 2, 3, 4)로 호출하면
- 첫번째요소 ← 1
- 나머지 ← [2, 3, 4]
가 들어갑니다.

Array.isArray(배열)
→ 배열이 배열이면 true
→ 배열이 배열이 아니면 false

```
> const max = function (첫번째요소, ...나머지) {  
  if (Array.isArray(첫번째요소)) {  
    let output = 첫번째요소[0]  
    for (const 값 of 첫번째요소) {  
      if (output < 값) {  
        output = 값  
      }  
    }  
    return output  
  } else {  
    let output = 첫번째요소  
    for (const 값 of 나머지) {  
      if (output < 값) {  
        output = 값  
      }  
    }  
    return output  
  }  
}  
  
console.log(max([1, 2, 3, 4])) // 4  
console.log(max(1, 2, 3, 4))  // 4  
4  
4  
← undefined
```

오늘도 고생하셨습니다.