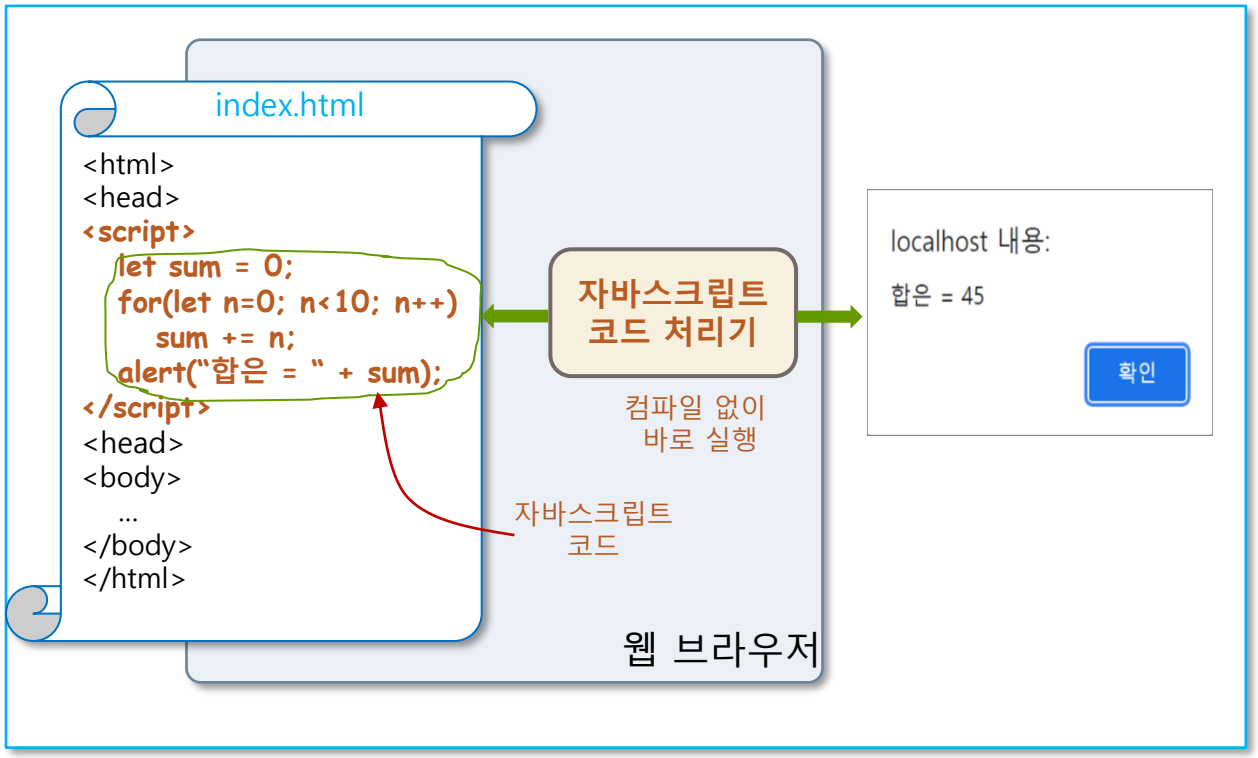
1

6장 자바스크립트 언어

자바스크립트 언어

- Javascript
  - 1995년 넷스케이프 개발, Netscape Navigator 2.0 브라우저에 최초 탑재

- 특징



자바스크립트 코드의 위치

- 자바스크립트 코드 작성이 가능한 위치
  1. HTML 태그의 이벤트 리스너 속성에 작성
  2. <script></script> 태그에 작성
  3. 자바스크립트 파일에 작성
  4. URL 부분에 작성

자바스크립트 코드의 위치

1. HTML 태그의 이벤트 리스너에 자바스크립트 코드 작성

onclick 이벤트 리스너 속성

자바스크립트 코드 (이미지를 banana.png로 교체)

```
<img src="apple.png" alt="img" onclick="this.src='banana.png'">
```

- 예제 6-1

<script></script> 태그에 자바스크립트 작성

- 특징
  - <head></head>나 <body></body> 내 어디든 가능
  - 웹 페이지 내에 여러 번 삽입 가능

- 예제 6-2

자바스크립트 코드를 별도 파일에 작성

- 자바스크립트 코드 파일 저장
  - 확장자 .js 파일에 저장
  - <script> 태그 없이 자바스크립트 코드만 저장

- 여러 웹 페이지에서 불러 사용
  - 웹 페이지마다 자바스크립트 코드 작성 중복 불필요
  - <script> 태그의 src 속성으로 파일을 불러 사용

```
<script src="파일이름.js">
    // HTML5부터 이곳에 자바스크립트 코드 추가 작성하면 안 됨
</script>
```

- 예제 6-3

URL 부분에 자바스크립트 코드 작성


```
<a href="javascript:자바스크립트코드">링크</a>
```

- 예제 6-4

자바스크립트로 HTML 콘텐츠 출력

- 자바스크립트로 HTML 콘텐츠를 웹 페이지에 직접 삽입
  - document.write()

    예) document.write("<h3>Welcome!</h3>");

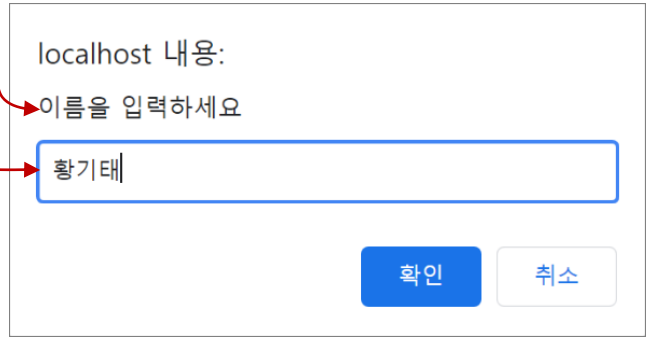
  - document.writeln()
    - writeln()은 텍스트에 '₩n ' 을 덧붙여 출력
    - '₩n'을 덧붙이는 것은 고작해야 빈칸 하나 출력

  - 예제6-5

자바스크립트 다이얼로그 : 프롬프트 다이얼로그

- prompt("메시지", "디폴트 입력값") 함수
  - 사용자로부터 문자열을 입력 받아 리턴

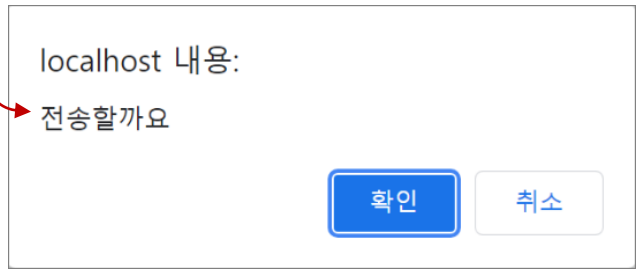
```
let ret = prompt("이름을 입력하세요", "황기태");
if(ret == null) {
   // 취소 버튼이나 다이얼로그를 닫은 경우
}
else if(ret == "") {
   // 문자열 입력 없이 확인 버튼 누른 경우
}
else {
   // ret에는 사용자가 입력한 문자열
}
```



자바스크립트 다이얼로그 : 확인 다이얼로그

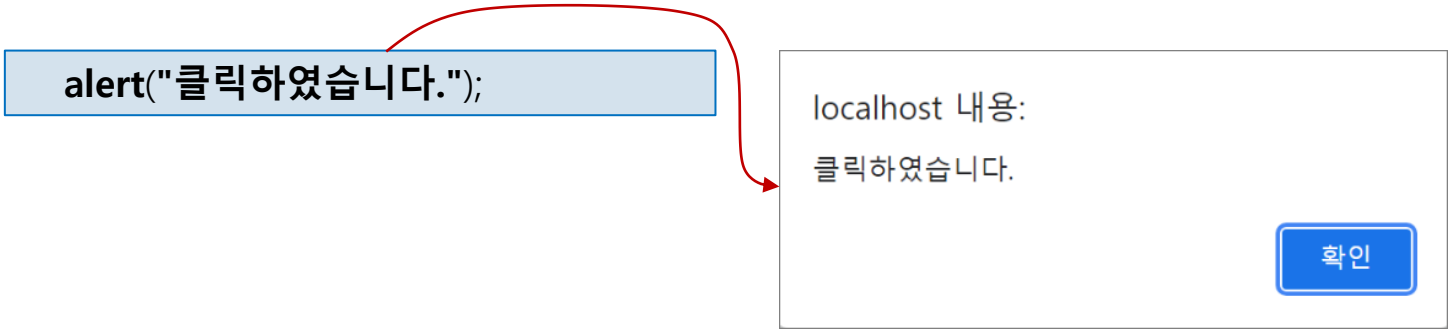
- confirm("메시지") 함수
  - '확인' 버튼을 누르면 true, '취소' 버튼이나 강제로 다이얼로그를 닫으면 false 리턴

```
let ret = confirm("전송할까요");
if(ret == true) {
   // 사용자가 "확인" 버튼을 누른 경우
}
else {
   // 취소 버튼이나 다이얼로그를 닫은 경우
}
```



자바스크립트 다이얼로그 : 경고 다이얼로그

- alert("메시지") 함수
  - 메시지'와 '확인' 버튼을 가진 다이얼로그 출력, 메시지 전달



자바스크립트 식별자

- 식별자(identifier)
  - 식별자 만드는 규칙
    - 첫 번째 문자 : 알파벳(A-Z, a-z), 언더스코어(_), $ 문자만 사용 가능
    - 두 번째 이상 문자 : 알파벳, 언더스코어(_), 0-9, $ 사용 가능
    - 대소문자는 구분되어 다루어짐
    - 자바스크립트 예약어 사용 불가

자바스크립트 문장

- 문장

```
i = i + 1              // (0) 한 줄에 한 문장만 있는 경우 세미콜론 생략 가능
j = j + 1;             // (0)
k = k + 1; m = m + 1;    // (0) 한 줄에 여러 문장
```

  - 주석문

```
// 한 라인 주석. 라인의 끝까지 주석 처리
/*
   여러 라인 주석
*/
```

데이터 타입

- 자바스크립트 언어에서 다루는 데이터 종류
  - 숫자 타입 : 정수, 실수(예: 42, 3.14)
  - 논리 타입 : 참, 거짓(예: true, false)
  - 문자열 타입(예: ‘좋은 세상’, "a", "365", "2+4")
  - 객체 레퍼런스 타입 : 객체를 가리킴. C 언어의 포인터와 유사
  - null : 값이 없음을 표시하는 특수 키워드. Null, NULL과는 다름

변수

- 변수 선언
  - 변수 이름을 정하고, 저장 공간 할당
  - 3가지 방법(현재 3가지 방법 모두 사용)
    - var 키워드 이용
    - let 키워드 이용
    - const 키워드 이용

```
var score;                  // 변수 score 선언
var year, month, day;       // year, month, day의 3 개의 변수 선언
var address = "서울시"; // address 변수를 선언하고 "서울시"로 초기화
```

```
let score;                  // 변수 score 선언
let year, month, day;       // year, month, day의 3 개의 변수 선언
let address = "서울시";  // address 변수를 선언하고 "서울시"로 초기화
```

```
age = 21;        // var나 let 없이 변수 age가 선언. 동시에 21로 초기화
```

변수의 사용 범위(scope)와 생명(life)

	선언	사용 범위	변수의 생명
전역 변수	함수 밖에서 선언 혹은 var/let 키워드 없이 아무 곳에서나 선언	프로그램 전역	프로그램이 실행을 시작할 때 생성 프로그램 종료 때 소멸
지역 변수	함수 내에 let으로 선언	선언된 함수 내	함수가 실행될 때 생성 함수가 종료할 때 소멸
블록 변수	let으로 if, while, for 등 블록 내에 선언	선언된 블록 내	블록의 실행 시작 시 생성 블록이 끝나면 소멸

this로 전역변수 접근

- 지역 변수와 전역 변수의 이름을 같을 때
  - 전역 변수에 접근하고자 할 때 : **this.전역변수**

```
var x;        // 전역변수

function f() {
  var x;      // 지역변수

  x = 1;      // 지역변수 x에 1 저장
  this.x = 100; // 전역변수 x에 100 저장
}
```

  - 주의
    - let으로 선언된 전역 변수는 this로 접근할 수 없다.

  - 예제6-6

let의 특징

- 특징
  - let으로 동일한 변수 재 선언 불가

```
let x = 1;
let x = 2; // 오류.
           //변수 x에 대한 재 선언 불가
```

  - let은 변수 사용 범위를 블록 내로 제한

```
if(a == b) {
   let x = 10; // x는 if 블록에서만 사용
}
x++; // 오류. x 사용할 수 없음
```

```
for(let n=0; n<10; n++) {
   let x = 10; // n과 x는 for 블록에서만 사용
}
x++; // 오류.  x 사용할 수 없음
n++; // 오류. n 사용할 수 없음
```

상수

- 상수 : 변하지 않는 값을 가지는 이름, **const**로 선언

```
const MAX = 10; // 10의 값을 가지는 상수 MAX 선언
```

자바스크립트의 리터럴

- 리터럴(literal)
  - 데이터 값 그 자체
- 리터럴 종류(예제6-7)

종류		특징	예
정수	8진수	0으로 시작	let n = 015; // 8진수 15. 10진수로 13
	10진수		let n = 15; // 10진수 15
	16진수	0x로 시작	let n = 0x15; // 16진수 15. 10진수로 21
실수	소수형		let height = 0.1234;
	지수형		let height = 1234E-4; // $1234 \times 10^{-4} = 0.1234$
논리	참	true	let condition = true;
	거짓	false	let condition = false;
문자열		""로 묶음	let hello = "안녕하세요";
		''로 묶음	let name = 'kitae';
기타	null	값이 없음을 뜻함	let ret = null;
	NaN	수가 아님을 뜻함	let n = parseInt("abc"); // 이때 parseInt()는 NaN을 리턴

자바스크립트의 식과 연산

- 자바스크립트의 연산과 연산자 종류

연산 종류	연산자	연산 종류	연산자
산술	+ - * / %	대입	= *= /= += -= &= ^= |= <<= >>= >>>=
증감	++ --	비교	> < >= <= == !=
비트	& | ^ ~	논리	&& || !
시프트	>> << >>>	조건	? :

- 예제 6-8 ~ 예제 6-14

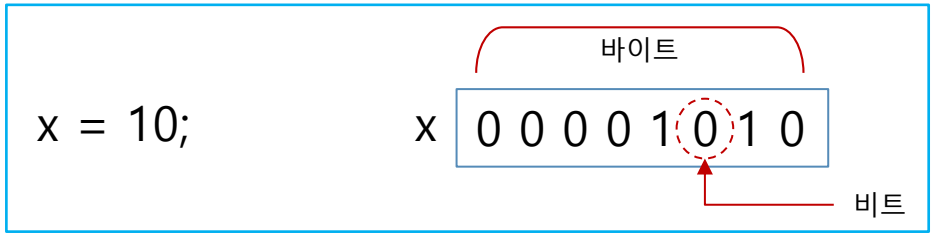
조건 연산자

- 조건 연산
  - condition ? expT : expF
    - condition이 true이면 전체 결과는 expT의 계산 값
    - false이면 expF의 계산 값

```
let x=5, y=3;
let big = (x>y) ? x : y; // (x>y)가 true이므로 x 값 5가 big에 대입된다.
```

비트 연산

- 비트 개념



- 비트 연산 종류
  - 비트들끼리의 비트 논리 연산
  - 비트 시프트 연산

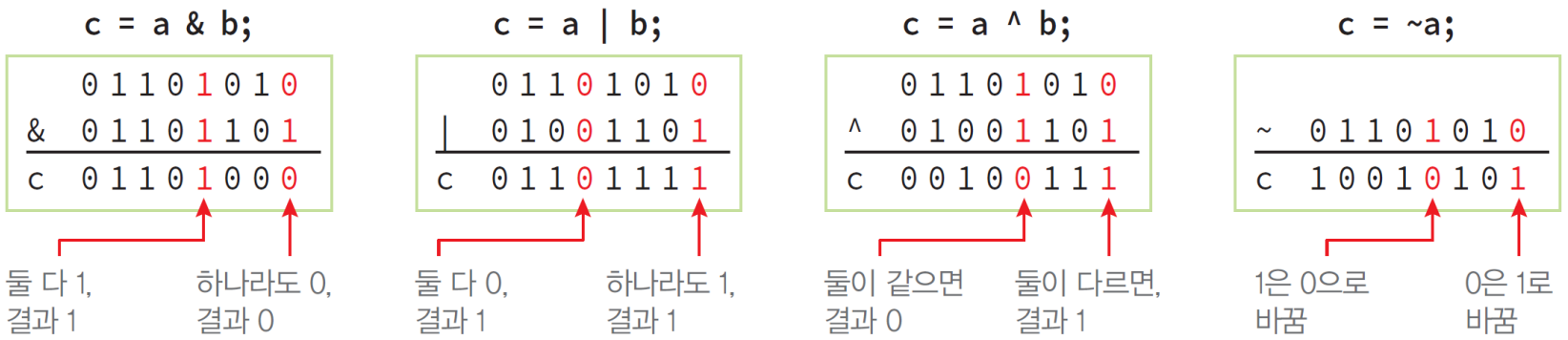
비트 논리 연산

비트 논리 연산

연산자	별칭	연산 설명
a & b	비트 AND 연산	두 비트 모두 1이면 1. 그렇지 않으면 0
a | b	비트 OR 연산	두 비트 모두 0이면 0. 그렇지 않으면 1
a ^ b	비트 XOR 연산	두 비트가 다르면 1, 같으면 0
~ a	비트NOT 연산	1을 0으로, 0을 1로 변환

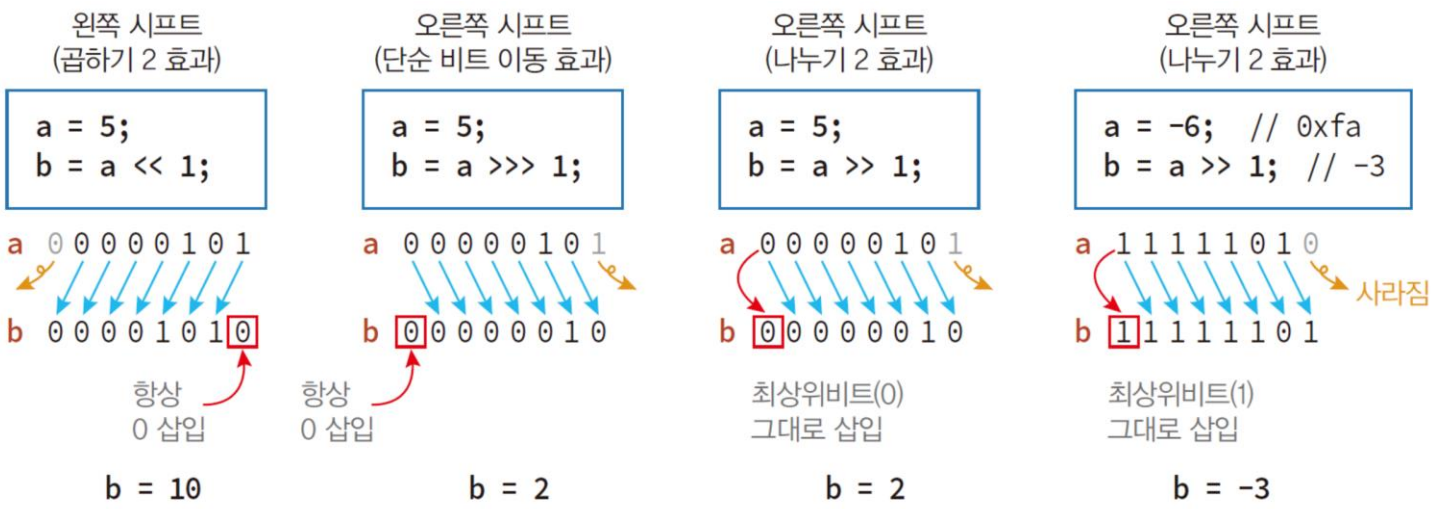
a = 106; 01101010

b = 77; 01001101



비트 시프트 연산

- 시프트 : 저장 공간에서 비트들의 오른쪽/왼쪽 이동



연산자	별칭	설명
a << b	산술적 왼쪽 시프트	a의 비트들을 왼쪽으로 b번 이동. 최하위 비트의 빈자리는 0으로 채움. 한 비트 시프트마다 곱하기 2의 효과 발생. a 값은 변화 없음
a >> b	산술적 오른쪽 시프트	a의 비트들을 오른쪽으로 b번 이동. 최상위 비트의 빈자리는 시프트 전 최상위비트로 채움. 한 비트 시프트마다 나누기 2의 효과 발생. a 값은 변화 없음
a >>> b	논리적 오른쪽 시프트	a의 비트들을 오른쪽으로 b번 이동. 최상위 비트의 빈자리는 0으로 채움. a 값은 변화 없음

문자열 연산자

- 문자열 연결
  - +, +=

```
"abc" + "de"          // "abcde"
"abc" + 23            // "abc23"
23 + "abc"            // "23abc"
23 + "35"             // "2335"
23 + 35               // 58, 정수 더하기
```

  - 순서에 유의

```
23 + 35 + "abc";     // 23 + 35 -> 58로 먼저 계산, 58 + "abc" -> "58abc"
"abc" + 23 + 35;     // "abc" + 23 -> "abc23"로 먼저 계산, "abc23" + 35 -> "abc2335"
```

- 문자열 비교
  - 비교 연산자(!=, ==, > , <, <=, >=)는 문자열 비교에 사용
  - 사전 순으로 비교 결과 리턴

```
let name = "kitae";
let res = (name == "kitae"); // 비교 결과 true, res = true
let res = (name > "park"); // name이 "park"보다 사전순으로 앞에 나오므로 res = false
```

if, if-else

if, if-else 문

```
if(조건식) {
    … 실행문 … // 조건식이 참인 경우
}
```

```
if(a > b) {
    document.write("a가 크다");
}
```

```
if(조건식) {
    … 실행문1 … // 조건식이 참인 경우
}
else {
    … 실행문2 … // 조건식이 거짓인 경우
}
```

```
if(a > b) {
    document.write("a가 크다");
}
else {
    document.write("a가 크지 않다");
}
```

```
if(조건식1) {
    실행문1 // 조건식1이 참인 경우
}
else if(조건식2) {
    실행문2 // 조건식2가 참인 경우
}
    ……

else {
    실행문n; // 앞의 모든 조건이 거짓인 경우
}
```

```
if(a > b) {
    document.write("a가 크다");
}
else if(a < b) {
    document.write("b가 크다");
}
else
    document.write("a와 b는 같다");
```

switch 문

- switch 문
  - 값에 따라 서로 다른 코드를 실행할 때, switch 문 적합

```
switch(식) {
   case 값1: // 식의 결과가 값1과 같을 때
      실행 문장 1;
      break;
   case 값2: // 식의 결과가 값2와 같을 때
      실행 문장 2;
      break;
      ...
   case 값m:
      실행 문장 m; // 식의 결과가 값과 같을 때
      break;
   default: // 어느 값과도 같지 않을 때
      실행 문장 n;
}
```

```
let fruits="사과";
switch(fruits) {
   case "바나나":
      price = 200; break;
   case "사과":
      price = 300; break;
   case "체리":
      price = 400; break;
   default:
      document.write("팔지 않습니다.");
      price = 0;
}

// switch 문의 실행 결과 price=300
```

case 문의 '값'

- case 문의 '값'은 const로 선언된 상수나 리터럴만 가능
  - 잘 작성된 case 문

```
const MAX = 100;
…
…
case 1 :
case 2.7 :
case "Seoul" :
case MAX :
case true :
case 2+3 : // 2+3은 먼저 5로 계산되어 case 5:와 동일
```

- case 문의 '값'에 변수나 식은 사용 불가
  - 잘못 작성된 case 문

```
case a :              // 오류. 변수 a 사용 불가
case a > 3 :          // 오류. 식(a>3) 사용 불가
```

switch 문에서 break 문의 역할

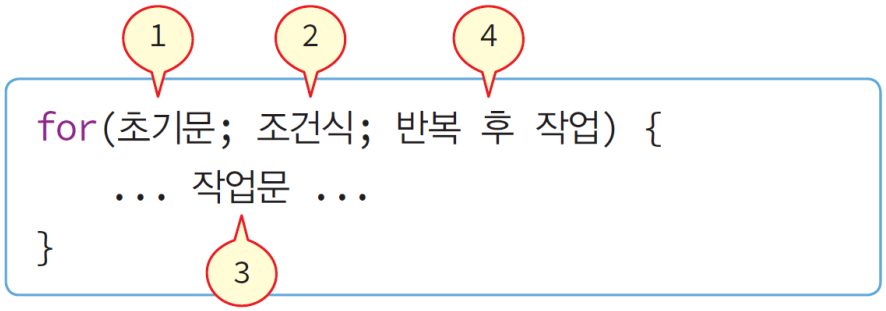
- break 문
  - switch 문 종료
    - break; 문을 만날 때까지 아래로 코드 계속 실행

```
let city="Seoul";
switch(city) {
   case "Seoul":
      document.write("서울");
      break;
   case "NewYork":
      document.write("뉴욕");
      break;
   case "Paris":
      document.write("파리");
      break;
}
```

```
let day="월";
switch(day) {
   case "월":
   case "화":
   case "수":
   case "목":
   case "금":  document.write("정상영업");
               break;
   case "토":
   case "일": document.write("휴일");
               break;
}
```

반복문

- for 문



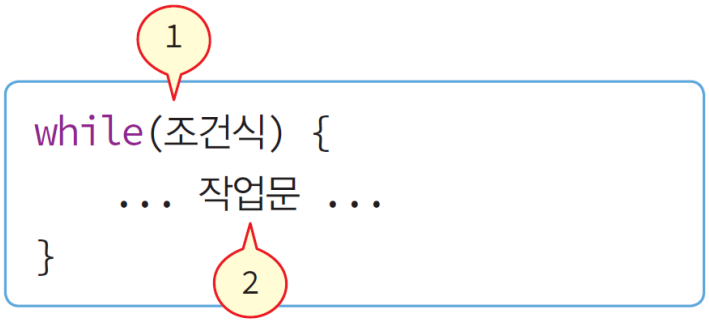

```
// 0에서 9까지 출력
for(let i=0; i<10; i++) {
   document.write(i);
}
```

```
0123456789
```

  - 예제 6-17

반복문

- while 문



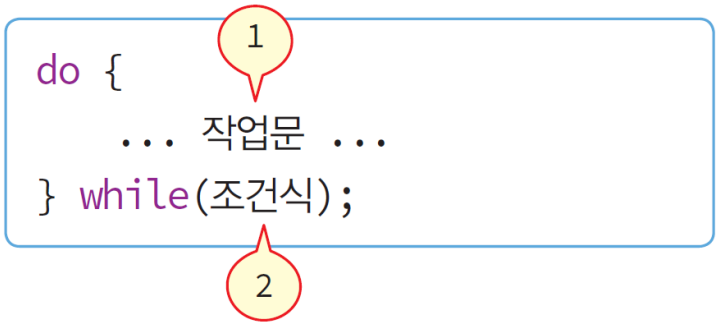

```
let i=0;
while(i<10) {  // i가 0에서 9까지 출력
   document.write(i);
   i++;
}
```

```
0123456789
```

  - 예제 6-18

반복문

- do-while 문




```
let i=0;
do {  // i가 0에서 9까지 출력
   document.write(i);
   i++;
} while(i<10);
```

```
0123456789
```

  - 예제 6-19

반복문 내의 break 문과 continue 문

- break 문 : 가장 안쪽 반복문 하나만 벗어나도록 제어

```
for( ... ) {
   .....
   break;
   .....
}
.....
.....
.....
```

```
for( ... ) {
   while( ... ) {
      .....
      break;
      .....
   }
   .....
}
```

  - 예제6-20

반복문 내의 break 문과 continue 문

- continue 문 : 반복 코드 실행 중단, 다음 반복으로 점프

```
for(초기문; 조건식; 반복 후 작업) {
    .....
    continue;
    .....
}
```

```
while(조건식) {
    .....
    continue;
    .....
}
```

```
do {
    .....
    continue;
    .....
} while(조건식);
```

  - 예제 6-21

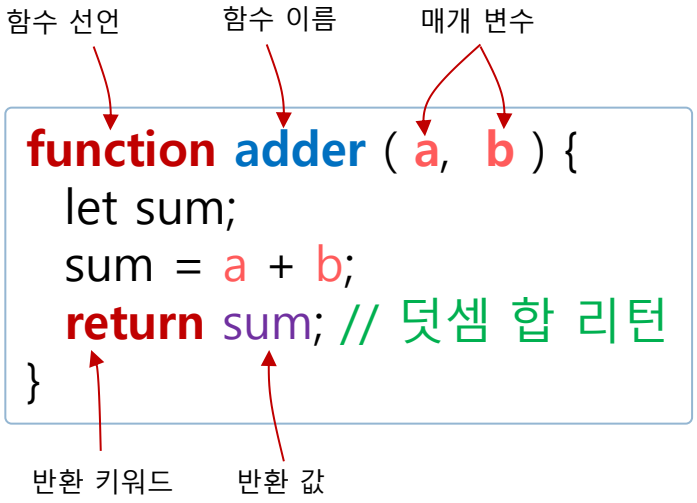
함수

- 함수란?
  - 목적을 가지고 작성된 코드 블록
  - 데이터 전달받아 처리한 후 결과를 돌려주는 코드 블록

함수의 구성과 호출

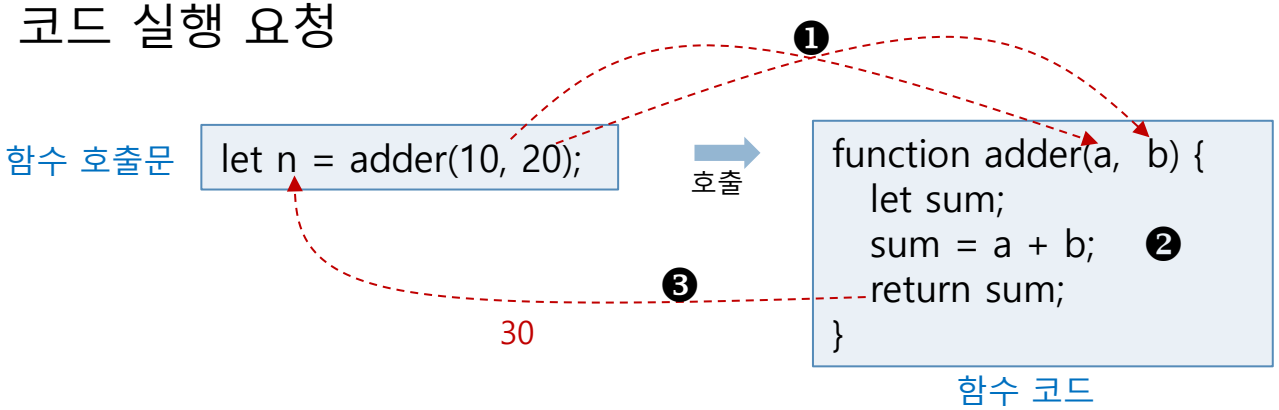
함수의 구성

```
function 함수이름(arg1, arg2,..., argn) {
    ...프로그램 코드...
    결과를 리턴하는 return 문
}
```



함수 호출

함수의 코드 실행 요청



자바스크립트에서 제공하는 전역 함수

전역 함수명	설명
eval(exp)	exp의 자바스크립트 식을 계산하고 결과 리턴
parseInt(str)	str 문자열을 10진 정수로 변환하여 리턴
parseInt(str, radix)	str 문자열을 radix 진수로 해석하고, 10진 정수로 바꾸어 리턴
parseFloat(str)	str 문자열을 실수로 바꾸어 리턴
isFinite(value)	value가 숫자이면 true 리턴
isNaN(value)	value가 숫자가 아니면 true 리턴

자바스크립트에서 제공하는 전역 함수

- 대표적인 자바스크립트 함수
  - eval() 함수

    예) let res = eval("2*3+4*6"); // res는 30
  - parseInt() 함수

    예) let I = parseInt("32"); // "32"를 10진수로 변환, 정수 32 리턴

    let n = parseInt("0x32"); // "0x32"를 16진수로 해석, 정수 50 리턴
  - isNaN() 함수

    예) isNaN(32) // false 리턴. 숫자이므로

    isNaN('32') // false 리턴. 숫자 값의 문자열이므로

    isNaN("32") // false 리턴. 숫자 값의 문자열이므로

    isNaN("hello") // true 리턴. 숫자가 아니므로

- 예제 6-23, 예제6-24