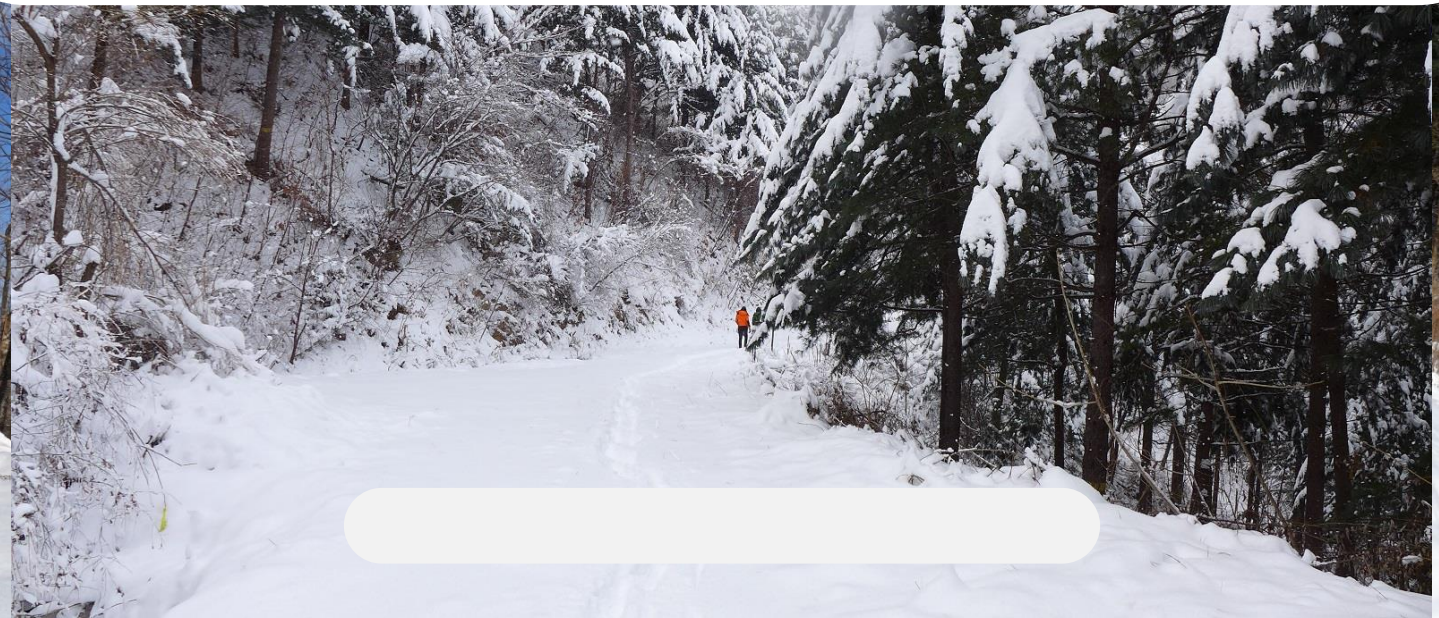



JavaScript

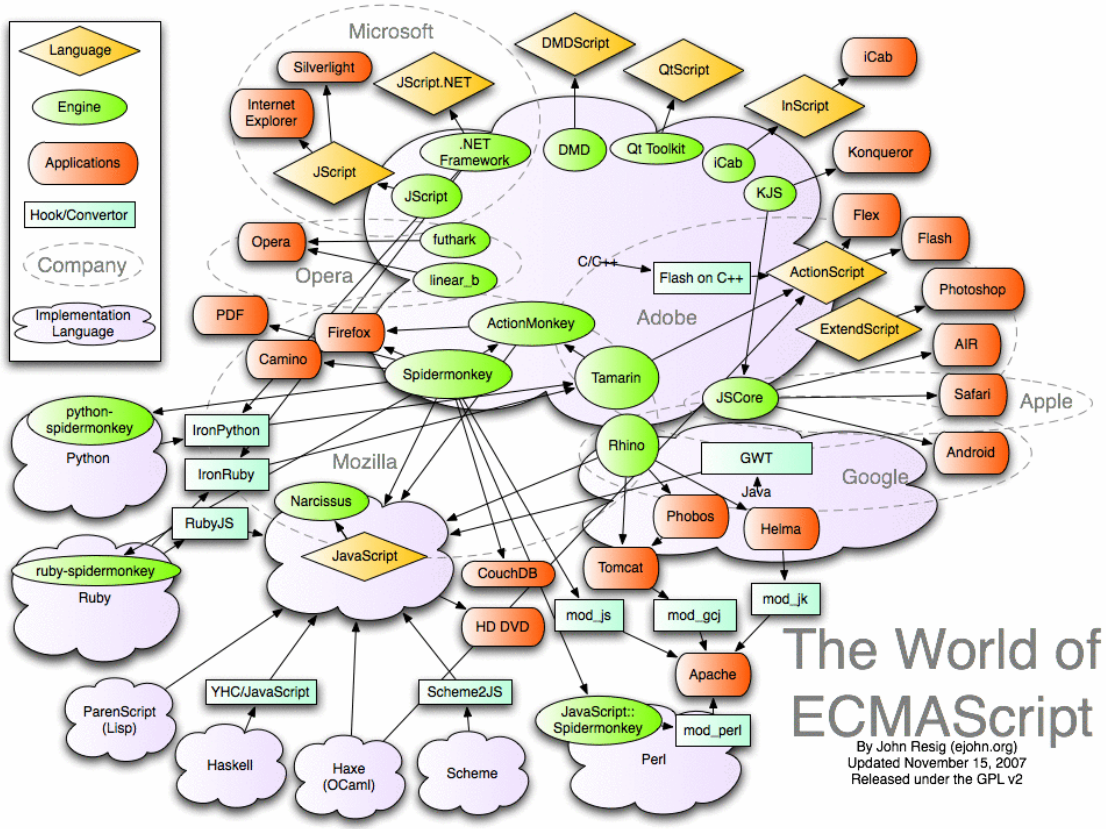




목차 (Table of Contents)

1. JavaScript 소개

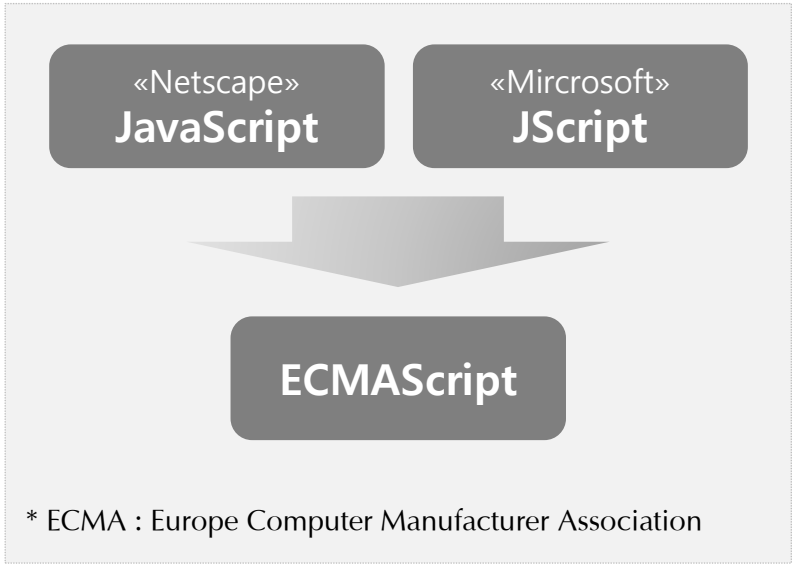
1.1 JavaScript 개요

- ✓ 프로토타입 기반의 객체지향 스크립트 프로그래밍 언어입니다.
- ✓ 웹 브라우저가 Javascript를 HTML과 함께 다운로드하여 실행합니다.
- ✓ 웹브라우저가 HTML문서를 읽어 들이는 시점에서, Javascript 엔진은 java script 를 실행합니다.
- ✓ 대부분의 렌더링 엔진은 ECMAScript 표준을 지원합니다.



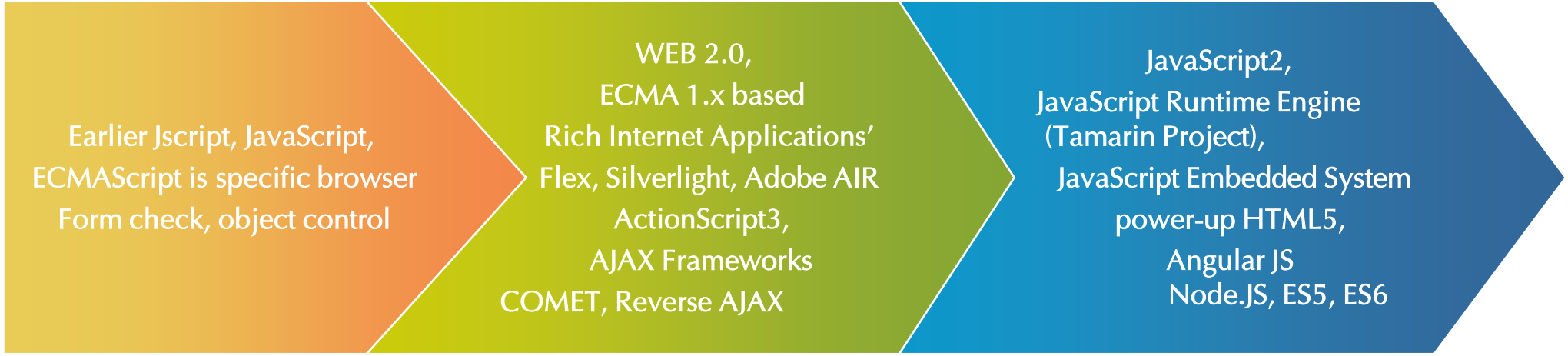
1.2 ECMAScript

✓ 1996년 3월 넷스케이프는 'Netscape Navigator 2.0'을 출시하면서 JavaScript를 지원합니다.
✓ 마이크로소프트사는 웹에 호환되는 J스크립트를 개발해서 1996년 8월 인터넷 익스플로러 3.0에 포함하며 출시합니다.
✓ 넷스케이프는 JavaScript 표준화를 위해 기술 규격을 ECMA에 제출하였습니다.
✓ 1997년 6월 ECMA는 ECMA-262초안을 일반회의에서 채택합니다.



1.3 JavaScript의 과거와 미래

✓ 1998년 ECMAScript(Edition 2)는 ISO 표준으로 개정됩니다.
✓ 1999년 정규 표현식, 문자열 처리, 새로운 제어문, 예외처리, 오류 정의, 포매팅 을 표준으로 추가하였습니다.(Edition 3)
✓ Edition 4는 정치적인 문제로 폐기되었습니다.
✓ 현재 Edition 6까지 표준화 하였고, Edition 7은 진행중입니다.




2. JavaScript 프로그램 예제

2.1 JavaScript 선언 (1/2)

- ✓ JavaScript는 HTML에서 사용하기 위해 <script> 태그를 이용합니다.
- ✓ <script> 태그의 'src'와 'type' 속성을 사용하여 JavaScript를 선언합니다.
- ✓ src속성은 독립된 자바스크립트 파일의 위치를 지정할 때 사용하며 생략이 가능합니다.
- ✓ type속성은 스크립트에서 사용하는 미디어 타입을 지정합니다.

[src속성을 생략한 경우]

```
<html>
<body>
    <script type="text/javascript">
       // src속성을 사용하지 않을 경우 내용을 포함해야 합니다.
       // JavaScript 내용
    </script>
</body>
</html>
```

[src속성을 사용한 경우]

```
<html>
<body>
    <script src="common.js" type="text/javascript"/>
    // src속성에 embeded할 독립된 자바스크립트의 경로를 선언합니다.
</body>
</html>
```

2.1 JavaScript 선언 (2/2)

✓ <script>태그는 HTML내의 어느 위치에서나 선언 가능합니다.
✓ HTML문서 내의 <head>와 <body>태그 내부에 선언하는 방식을 권장합니다.
✓ 헤더부분에 위치한 자바스크립트는 브라우저의 각종 입/출력 발생 이전에 초기화되므로 브라우저가 먼저 점검합니다.

```
<html>
  <head>
    <title>script tag의 위치</title>
    <!-- script tag의 위치 #1 -->
    <script type="text/javascript">
      document.write("반갑습니다. 여러분!");
    </script>
  </head>
  <body>
    즐거운 하루입니다.
    <!-- script tag의 위치 #2 -->
    <script type="text/javascript">
      document.write("자바스크립트 공부하기 참~ 좋은 날씨입니다.");
    </script>
  </body>
</html>
```

<head>태그 사이에 삽입된 <script>태그

<body>태그 사이에 삽입된 <script>태그

2.2 구구단 출력

✓ 변수는 숫자와 문자 모두 var 키워드를 사용합니다.
✓ for문을 사용해서 9번 반복문을 작성합니다.
✓ 연산 결과를 선언한 변수에 저장합니다.
✓ console.log를 사용해서 브라우저 콘솔 창에 결과를 출력합니다.

```
<html>
  <body>
    <script type="text/javascript">
      //변수를 선언합니다.
      var mul = 0;

      //반복문 중 for문을 사용해서 9번 반복합니다.
      for(var i = 1 ; i < 10 ; i++){

        //반복문 마다 곱셉 연산을 수행하고 저장 합니다.
        mul = 2 * i;

        //결과를 콘솔 창에 출력합니다.
        console.log( 2 + " * " + i + " = " + mul);
      }
    </script>
  </body>
</html>
```

RESULT

```
연산결과 :

  2 * 1 = 2
  2 * 2 = 4
  2 * 3 = 6
  2 * 4 = 8
  2 * 5 = 10
  2 * 6 = 12
  2 * 7 = 14
  2 * 8 = 16
  2 * 9 = 18
```

2.3 시계 구현

- ✓ JavaScript에서 제공하는 Date 객체를 사용해서 웹 브라우저에 현재 시간을 표기할 수 있습니다.
- ✓ Date는 JavaScript에서 제공하는 내장 객체입니다.
- ✓ toLocalTimeString으로 현재 시간을 얻을 수 있습니다.
- ✓ setInterval로 1초마다 화면을 갱신합니다.

```
<html>
<head>
<script type="text/javascript">
  //화면이 로딩되면 수행되는 구문을 작성합니다.
      window.onload = function() {
        setInterval(showClock, 1000);
      };
      //화면에 시간을 출력합니다.
      function showClock(){
        //clock 요소를 검색합니다.
        var clockE = document.getElementById("clock");

        //현재 시간을 계산합니다.
        var date = new Date();
        clockE.innerHTML = date.toLocaleTimeString();
      }
    </script>
</head>
  <body>
    <span id="clock"></span>
  </body>
</html>
```

RESULT

수행결과 :

오후 9:15:58


3. JavaScript 기초

3.1 주석

✓ JavaScript는 2가지 주석처리 방법을 제공합니다.
✓ 한 줄 주석은 //로 표기합니다.
✓ 여러 줄 주석은 /* */로 표기하며 /* 부터 */까지가 주석입니다.
✓ 가능하면 /* */보다 //사용을 권장합니다.

```
<html>
  <head>
    <title>주석문</title>
    <script type="text/javascript">
      // 한줄 주석처리

      /*
         여러 줄의 주석처리
         여러 줄의 주석처리
         C, Java와 동일함.
      */
    </script>
  </head>
  <body>
  </body>
</html>
```

3.2 변수

✓ JavaScript는 ECMAScript 표준을 따라 낙타 표기법을 사용합니다.
✓ 낙타표기법은 기본적으로 소문자로 작성하되 2개 이상의 단어일 때 단어의 시작은 대문자로 표기하는 방법입니다.
✓ JavaScript에서는 변수 선언 시 타입을 명시하지 않습니다.
✓ 변수는 유일한 이름으로 식별하고 함수와 혼동되지 않는 이름을 사용합니다. (변수는 명사, 함수는 동사)

```
<html>
  <body>
    <script type="text/javascript">
      //변수
      var myName;
      var myVariable;

      //좋은예
      var listCount = 0;
      var userName = "홍길동";
      var selected = false;

      //나쁜예  - 변수인지 함수인지 알기 어렵습니다.
      var getListCount = 10;
      var isSelected = false;
    </script>
  </body>
</html>
```

3.3 자료 형 (1/2)

✓ 프로그램은 정적인 데이터 값을 동적으로 변환해 가면서 원하는 정보를 얻습니다.
✓ 프로그램에서 다루는 데이터 값의 종류들을 자료 형(타입)이라고 표현합니다.
✓ JavaScript에서 다루는 자료 형은 원시타입과 객체 타입으로 분류합니다.
✓ 원시타입으로는 숫자, 문자열, boolean, null, undefined이 있습니다. 이를 제외한 모든 값은 객체 타입입니다.

자료 형	typeof 출력 값	설명
수치형	number	정수 또는 실수형
문자열	string	문자, 작은따옴표나 큰따옴표로 표기
부울형	boolean	참(true)과 거짓(false)
Undefined	undefined	변수가 선언 되었지만 초기화 되지 않음
NULL	object	객체가 아님을 뜻함

3.3 자료 형 (2/2)

✓ JavaScript는 자료 형에 대해 매우 유연합니다.
✓ 어떤 자료 형이던 전달 할 수 있고, 그 값을 필요에 따라 변환 할 수 있습니다.
✓ 서로 다른 자료 형의 연산이 가능합니다.
✓ 모든 자료 형을 var로 선언하기 때문에 변수 선언은 수월하지만 엄격하지 않은 규칙 때문에 혼란을 일으킬수도 있습니다.

```
<html>
<body>
    <script type="text/javascript">
        var answer = 42;
        console.log("answer " + answer); // answer 42
        answer = "Thanks for all the fish...";
        console.log(answer); // Thanks for all the fish...

        console.log("The answer is " + 42); // The answer is 42
        console.log(42 + " is the answer"); // 42 is the answer

        console.log("37" - 7); // 30
        console.log("37" + 7); // 377

        console.log(parseInt("123.45") + 1); // 124
        console.log(parseFloat("123.45") + 1); // 124.45

        console.log("1.1" + "1.1"); // 1.11.1
        console.log((+"1.1") + (+"1.1")); // 2.2
    </script>
</body>
</html>
```

3.4 상수

✓ ECMAScript6 이전까지 상수표현이 지원되지 않았습니다.
✓ 변수에 저장된 값이 변경 불가능한 상수와 일반 변수를 구분하고자 낙타표기법과 C언어 규칙을 동시에 사용했습니다.
✓ 상수의 표기법은 모든 문자를 대문자를 사용하고 단어 사이는 "_"로 표기합니다.
✓ ECMAScript6 에서는 const 키워드를 추가해서 상수를 지원합니다.

```
<html>
  <body>
    <script type="text/javascript">
      // 변수
      var listCount = 0;
      var userName = "홍길동";
      var selected = false;

      //상수 - ECMAScript6 이전 보편적 사용 방법
      var LIST_COUNT = 10;
      var SELECATED = false;

      if(listCount < LIST_COUNT){
        //doSomethig()
      }
    </script>
  </body>
</html>
```

3.5 연산자 (1/4)

- ✓ JavaScript에서 기본적으로 제공하는 약속된 문자의 표현 식을 연산자라고 합니다.
- ✓ 연산자에는 산술 연산자 , 비교 연산자, 논리 연산자, 값 설정 연산자, 기타 연산자 등을 제공합니다.
- ✓ 표현 식에서 2개 이상의 연산자를 동시에 사용했을 경우 우선순위별로 표현 식을 해석합니다.
- ✓ 괄호를 사용하여 우선순위를 조절할 수 있습니다.

연산자	설명
++	선행 ++는 현재 값을 반환하고 값을 증가합니다. 후행 ++는 값을 증가시키고 결과를 반환합니다.
--	선행 --는 현재 값을 반환하고 값을 감소합니다. 후행 --는 값을 감소 한 후 결과를 반환합니다.
-	부호를 전환해서 결과를 반환합니다.
+	숫자로 값을 변환합니다.
~	비트단위 연산에서 사용하며 NOT 연산을 수행합니다.
!	논리연산에서 사용하며 boolean 결과를 반환합니다.
delete	프로퍼티를 제거합니다.
typeof	피연산자 타입을 반환합니다.
void	undefined 값을 반환합니다.
*,/,%	곱하기, 나누기, 나머지의 연산결과를 반환합니다.

연산자	설명
+,-	값이 숫자일 때는 더하기, 빼기의 연산결과를 반환합니다.
+	값이 문자열이면 문자열을 서로 결합합니다.
<<	비트연산자로 값을 왼쪽으로 이동합니다.
>>	비트연산자로 값을 오른쪽으로 이동합니다.
>>>	부호비트 확장 없이 값을 오른쪽으로 이동합니다.
<,<=,>,>=	숫자를 비교합니다.
instanceof	객체 타입을 확인합니다.
in	프로퍼티가 존재하는지 확인합니다.
==	동등관계인지 확인합니다.
!=	동등하지 않은지 확인합니다.

3.5 연산자 (2/4)

✓ 연산자는 연산의 대상이 되는 값에 따라서 동작이 결정됩니다.
✓ “+”연산자는 대상의 값이 모두 숫자인 경우 산술 연산을 수행합니다.
✓ “+”연산자는 대상 중에 문자열이 포함된 경우 모든 연산 대상을 문자열로 변환하고 문자열을 붙입니다.
✓ 연산자는 종류에 따라 1항 연산자, 2항 연산자, 3항 연산자로 구분됩니다.

연산자	설명
===	값이 일치하는지 확인합니다.
!==	값이 일치하지 않는지 확인합니다.
&	비트 단위 연산자로 AND 연산을 수행합니다.
^	비트 단위 연산자로 XOR 연산을 수행합니다.
|	비트 단위 연산자로 OR 연산을 수행합니다.
&&	AND 연산을 수행합니다.
||	|| 연산을 수행합니다.
?:	조건에 해당하는 구문을 수행합니다.
=	변수 또는 프로퍼티에 값을 할당합니다.
,	1번째 구문은 버리고 다음 구문 값을 반환합니다.

3.5 연산자 (3/4) – 활용 (1/2)

✓ a++와 a-- 는 각각 a=a+1과 a=a-1 연산의 축약형태로써 증감연산자 라고 합니다.
✓ 증감연산자가 앞에 오면 연산을 먼저 실행하고, 뒤에 오면 해당 라인을 진행 후 연산을 실행합니다.
✓ !연산은 NOT의 의미로써 boolean형의 값을 반대로 반환합니다.
✓ typeof 경우 해당 변수의 타입을 반환합니다.

```
var number1 = 10;
var number2 = 20;
var boolean1 = true;

console.log("number1++ : " + number1++);
console.log("number1 : " + number1);
console.log("--number : " + --number1);
console.log("!boolean1 : " + !boolean1);

console.log("typeof boolean1 : " + typeof boolean1);
console.log("typeof number1: " + typeof number1);


console.log("number1 + number2 : " + (number1 + number2));
console.log("number1 - number2 : " + (number1 - number2));
console.log("number1 * number2 : " + (number1 * number2));
console.log("number1 / number2 : " + (number1 / number2));
```

```
number1++ : 10
number1 : 11
--number : 10
!boolean1 : false
typeof boolean1 : boolean
typeof number1: number
number1 + number2 : 30
number1 - number2 : -10
number1 * number2 : 200
number1 / number2 : 0.5
```

3.5 연산자 (4/4) – 활용 (2/2)

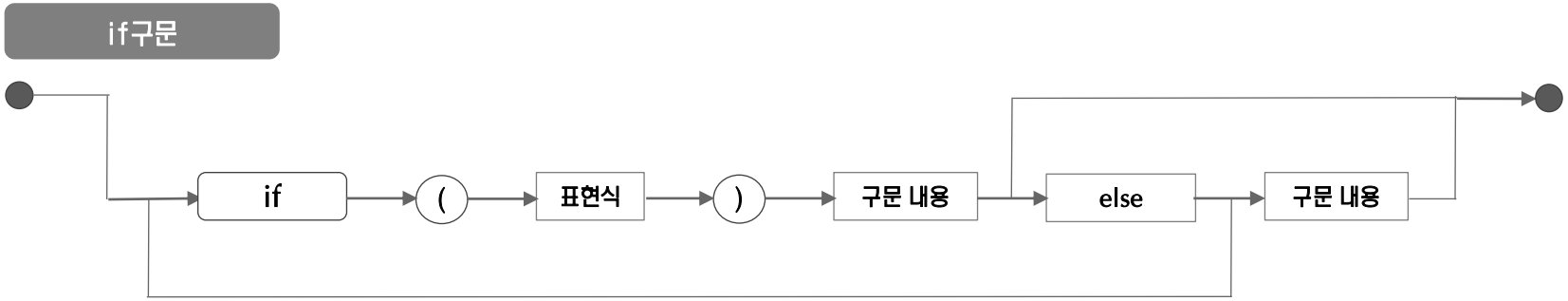
✓ 논리값을 비교하여 참(true)과 거짓(false)을 판단할 수 있습니다.
✓ 비교연산자 ==, ===의 차이점은 자료 형 까지 비교하는지 아닌지의 여부입니다.
✓ 비교연산자 &&은 둘 중 하나라도 거짓(false)이면 false, ||는 둘 중 하나라도 참(true)이면 true를 반환합니다.
✓ 3항 연산자의 ? 앞 비교 값이 참(true)이면 :앞의 값을 반환하고, 거짓(false)이면 :뒤의 값을 반환합니다.

```
var number1 = 10;
var string1 = "10";

console.log("number1 == string1 : " + (number1 == string1));
console.log("number1 === string1 : " + (number1 === string1));

string1 = "20";

console.log("number1 != string1 : " + (number1 != string1));
console.log("number1 !== string1 : " + (number1 !== string1));

console.log("true && true : " + (true && true));
console.log("true && false : " + (true && false));
console.log("true || true : " + (true || true));
console.log("true || false : " + (true || false));

console.log("number1 > number2 ? 'true' : 'false' " + (number1 > number2 ? 'true' : 'false'));
```

```
number1 == string1 : true
number1 === string1 : false
number1 != string1 : true
number1 !== string1 : true
true && true : true
true && false : false
true || true : true
true || false : true
number1 > number2 ? 'true' : 'false' false
```

3.6 조건문 (1/3) – if

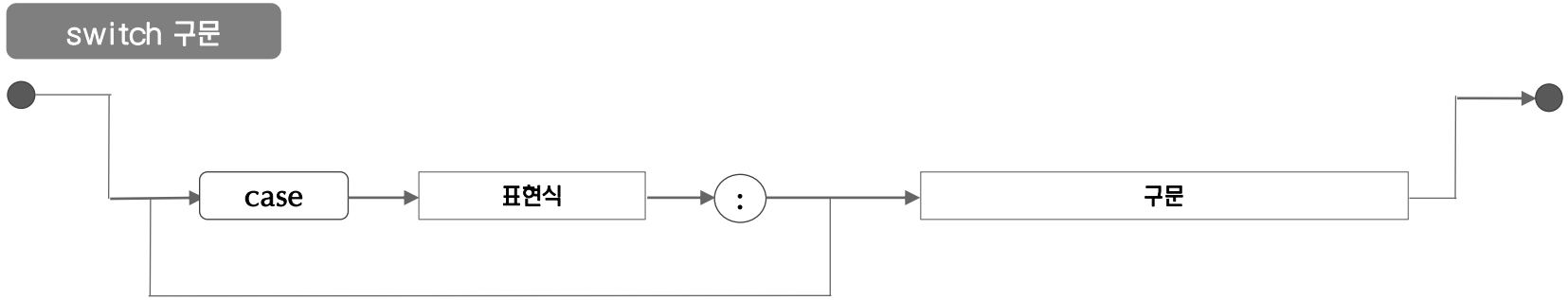
✓ 표현 식의 값에 따라서 특정 구문들을 실행 하거나 실행 하지 않도록 제어합니다.
✓ 조건 구문을 분기 구문이라고도 표현하는데 2개 이상의 경로 중에서 특정 경로를 선택할 수 있습니다.
✓ if 구문은 단순한 결정을 내리거나 정교한 구문들을 표현하는데 사용하는 구문입니다.
✓ if와 else if의 표현 식 결과 값이 참(true)일 경우 구문을 실행 합니다. 모두 거짓(false)일 경우 else문을 실행합니다.




```
if (표현식) {
  //구문내용
}
else if (표현식) {
  //구문내용
}
else {
  //구문내용
}
```

3.6 조건문 (2/3) – switch

✓ switch 구문은 동일한 표현식이 반복될 때 효과적인 구문입니다.
✓ 값이 case와 동일 할 경우 해당 구문내용을 실행합니다.
✓ break문은 switch 구문을 종료하며, break문이 없을 경우 다음 case를 실행합니다.
✓ 동일한 case가 없을 경우 default 구문을 실행합니다.




```
switch (값) {
  case 1:
  //구문내용
  break;
  case 2:
  //구문내용
  break;
  default:
  //구문내용
  break;
}
```

3.6 조건문 (3/3) – 활용

✓ if 구문의 예제는 변수 number1과 number2를 비교하여 해당하는 조건의 구문을 실행하는 예제입니다.
✓ if 조건이 거짓이면 순차적으로 else if 조건을 확인하고, 모두 거짓일 경우에 else 문을 실행합니다.
✓ switch 구문의 예제는 변수 number1의 값을 비교하여 해당하는 조건의 구문을 실행하는 예제입니다.
✓ case 문에 break가 누락되지 않도록 주의해야 합니다.

```
var number1 = 10;
var number2 = 20;

if (number2 > number1) {
  console.log(“number2 > number1”);
}
else if (number2 < number1) {
  console.log(“number2 < number1”);
}
else {
  console.log(“else…”);
}
```

```
number2 > number1
```

```
var number1 = 20;

switch (number1) {
  case 10:
    console.log(“number1 is 10”);
  break;
  case 20:
    console.log(“number1 is 20”);
  default:
    console.log(“default…”);
}
```

```
number1 is 20
default…
```

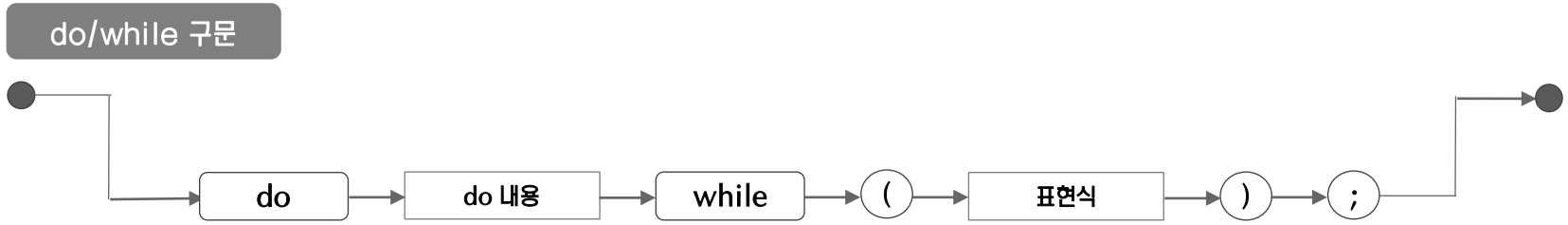
3.7 반복문 (1/4) – while

- ✓ 표현 식의 값이 참일 때 선언된 구문을 수행합니다.
- ✓ 표현 식의 값이 거짓일 때 while문을 종료합니다.
- ✓ while 구문은 무한 반복에 빠지는 상황에 쉽게 노출됩니다.
- ✓ 표현 식의 값이 변수를 참조 하고 변수가 거짓일 될 수 있는 상황을 만들어서 무한 반복 상황을 피하도록 합니다.


```
while (표현식) {
  //구문내용
}
```

3.7 반복문 (2/4) – do/while

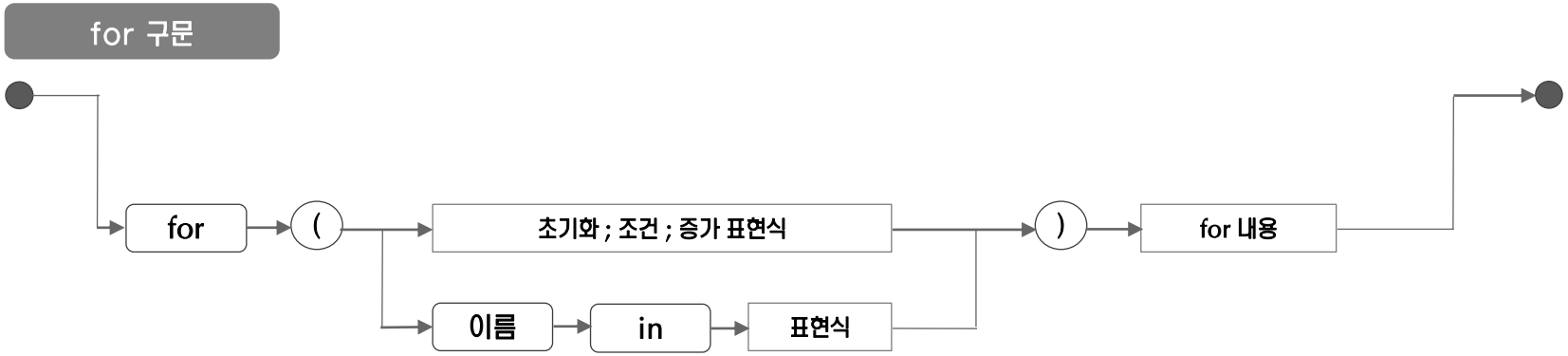
- ✓ 표현 식의 값을 확인하는 시점이 구문의 마지막이므로 최소 한번 이상 반복 구문이 수행됩니다.
- ✓ while과 문법적인 차이만 있을 뿐 수행 방법은 비슷합니다.
- ✓ 최초 1회 내용(body)를 수행하고 표현 식의 값이 참일 때 다음 내용을 수행합니다.
- ✓ 표현 식의 값이 거짓일 때 do문을 종료합니다.




```
do {
  //구문내용
} while (표현식)
```

3.7 반복문 (3/4) – for

✓ 프로그래밍 언어에서 가장 많이 사용하는 구문입니다.
✓ for 구문은 카운터 변수를 사용하는 구문과 in 키워드를 사용하는 구문으로 분류됩니다.
✓ 카운터 변수를 사용하는 for 구문은 카운터 변수가 표현 식에 명시된 조건만큼 증가되었을 때 반복이 종료됩니다.
✓ in 키워드를 사용하는 for 구문은 배열 또는 객체들이 가진 프로퍼티를 순회하면서 반복 구문이 수행됩니다.




```
for (초기화; 조건; 증감 표현식;){
  //구문내용
}
```

```
for (변수 in 배열){
  //구문내용
}
```

3.7 반복문 (4/4) – 활용

✓ 1부터 10의 합을 반복 문을 통하여 계산 하는 예제입니다.
✓ 다른 반복 문과는 다르게 do-while문은 꼭 한번은 실행되어야 할 때 사용합니다.
✓ break문을 만나면 표현식이 참이더라도 반복문을 종료합니다.
✓ continue문을 만나면 현재 반복문 블록의 하위 로직을 수행하지 않고 다음 반복을 계속 합니다.

```
var number = 1;
var sum = 0;

while (number<=10) {
  sum += number; //sum = sum + number;
  number++;
}
console.log(sum);
```

55

```
var sum = 0;

for (var number = 1; number<=10;
number++){
  sum += number; //sum = sum + number;
}
console.log(sum);
```

55

```
var number = 1;
var sum = 0;

do {
  sum += number; //sum = sum + number;
  number++;
} while (number<=10)
console.log(sum);
```

55

```
var number = 1;
var sum = 0;
while (number<=10) {
  sum += number; //sum = sum + number;
  if (number == 5) {
    break;
  }
  number++;
}
console.log(sum);
```

15

3.8 함수 (1/3) – 선언, 호출

✓ JavaScript에서 함수는 일급(first-class) 객체입니다.
✓ 함수는 변수, 객체, 배열 등에 저장될 수 있고 다른 함수에 전달하는 전달인자 또는 반환 값으로 사용할 수 있습니다.
✓ 함수는 프로그램 실행 중에 동적으로 생성될 수 있습니다.
✓ 함수 정의 방법은 함수 선언문, 함수 표현식, Function 생성자 세가지 방식이 있습니다.

```
// 함수 선언문
function 함수이름(매개변수1, 매개변수2, … , 매개변수n){
  //함수 내용
}
```

```
// 함수 표현식
var 함수 명 = function(매개변수1, 매개변수2, … , 매개변수n) {
  //함수 내용
}
```

```
// Function 생성자
var 함수 명 = new Function("매개변수1", "매개변수2" , … , "매개변수n" , "함수내용");
```

```
// 함수호출
함수 명(매개변수1, 매개변수2, … , 매개변수n);
```

3.8 함수 (2/3) – 매개변수

✓ 함수의 정의부분에 나열된 변수를 매개변수라 합니다.
✓ 함수를 호출할 때 전달되는 실제 값을 전달인자라 합니다.
✓ JavaScript에서 함수 정의 시 매개변수에 대한 형식은 명시하지 않습니다.
✓ 함수 호출 시 정의된 매개변수와 전달인자의 개수가 일치하지 않더라도 호출이 가능합니다.

```
function getName(p1, p2) {
  if ( p2 === undefined ) p2 = [];
  for (var property in p1){
       p2.push(property);
  }
  return p2;
}

var a = getName(p1);
getName(p3, p2);
```

3.8 함수 (3/3) – 활용

✓ 다음은 함수의 간단한 예제입니다.
✓ 1부터 매개변수 number 까지의 합을 구하는 예제 입니다.
✓ 각각 함수 선언문 , 함수 표현식, Function 생성자 방식의 선언과 호출입니다.
✓ 방식은 달라도 함수 호출방식은 똑같습니다.

```
// 함수 선언문
function func1(n){
  var sum = 0;

  for (var number = 1; number<=n; number++){
    sum += number;
  }
  console.log(sum);
}
func1(10);
```

55

```
// 함수 표현식
var func2 = function(n) {
  var sum = 0;

  for (var number = 1; number<=n; number++){
    sum += number;
  }
  console.log(sum);
}
func2(10);
```

55

```
// Function 생성자
var func3 = new Function("n", "var sum = 0; for(var number = 1; number<=n; number++){sum += number; } console.log(sum);" );
func3(10);
```

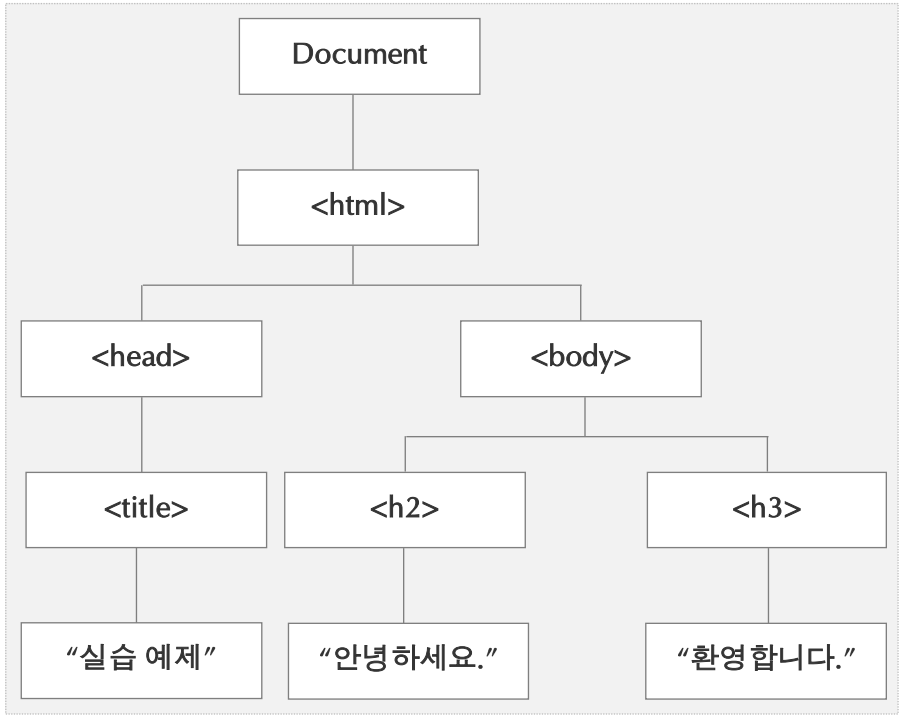
55


4. 웹브라우저와 JavaScript

4.1 DOM 개요 (1/2)

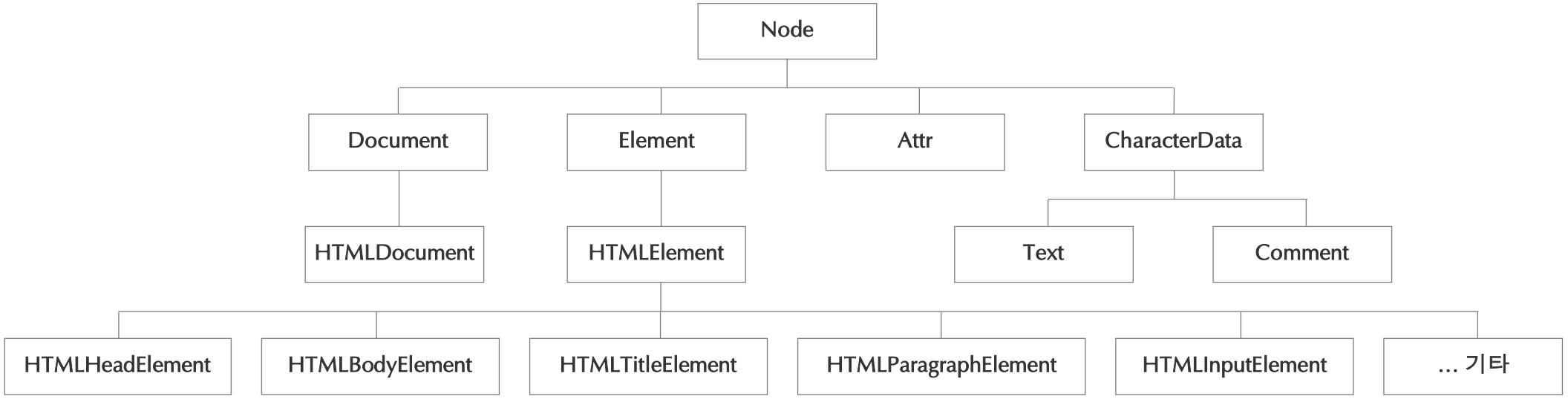
- ✓ DOM(Document Object Model)은 HTML과 XML 문서의 내용을 조작하는 API입니다.
- ✓ DOM은 HTML이 객체의 계층 구조로 표현됩니다.
- ✓ HTML계층 구조의 가장 상위에는 Document 노드가 위치해 있습니다.
- ✓ 하위로는 HTML태그나 요소들을 나타내는 노드와 문자열을 포함하는 노드로 구성되어 있습니다.

```
<html>
  <head>
    <title>실습 예제</title>
  </head>
  <body>
    <h2>안녕하세요.</h2>
    <h3>환영합니다.</h3>
  </body>
</html>
```



4.1 DOM 개요 (1/2) – 문서계층구조

✓ Document는 HTML 또는 XML 문서를 나타냅니다.
✓ HTMLDocument는 HTML의 문서와 요소만을 나타냅니다.
✓ HTMLElement의 하위 타입은 HTML 단일 요소나 요소 집합의 속성에 해당하는 JavaScript 프로퍼티를 정의합니다.
✓ Comment노드는 HTML이나 XML의 주석을 나타냅니다.

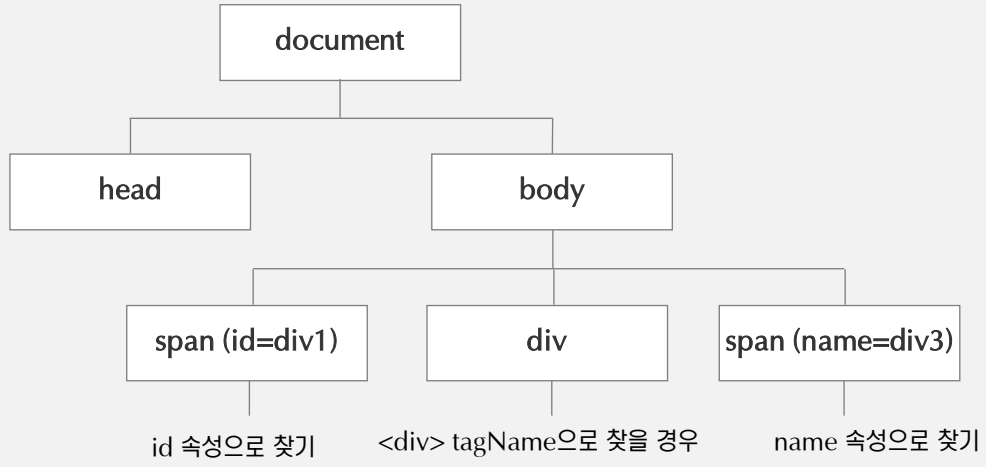


4.2 DOM 제어 (1/3)

- ✓ 웹 브라우저는 HTML, CSS, JavaScript가 서로 결합되어 동적인 화면을 구성합니다.
- ✓ JavaScript를 사용해서 DOM을 검색하거나 제어할 수 있습니다.
- ✓ JavaScript에서는 id, name, HTML태그 이름 등으로 문서 요소 검색이 가능합니다.
- ✓ Document 객체를 참조하기 위해서 document 전역 변수를 사용합니다.

```
<!DOCTYPE html>
<html>
<head></head>
<body>
    <span id="div1">div1 content</span>
    <div>div2 content</div>
    <span name="div3">div3 content</span>

    <script>
      // 1. id속성으로 찾는 방법
      document.getElementById("div1");
      // 2. tagName으로 찾는 방법
      document.getElementsByTagName("div");
      // 3. name속성으로 찾는 방법
      document.getElementsByName("div3");
    </script>

</body>
</html>
```

[HTML의 계층 구조]



4.2 DOM 제어 (2/3)

- ✓ JavaScript에서 window는 전역 객체입니다.
- ✓ window객체는 어디서나 접근할 수 있으며 document 프로퍼티를 포함합니다.
- ✓ 검색한 문서에 innerHTML을 통해서 또 다른 요소 추가가 가능합니다.

```
<!DOCTYPE html>
<html>

<body>
  <div id="div1"> div1 content</div>

  <script>
    var element1 = window.document.getElementById("div1");
    element1.innerHTML = "new content";
  </script>

</body>
</html>
```

RESULT

new content

4.2 DOM 제어 (3/3)

- ✓ JavaScript로 CSS의 클래스 변경이 가능합니다.
- ✓ 문서 요소에서 style.color로 문서 요소의 색상을 변경합니다.
- ✓ style.className으로 문서 요소 CSS 클래스를 변경합니다.
- ✓ style.backgroundColor로 문서요소 배경색 변경이 가능합니다.

```
<!DOCTYPE html>
<html>

<body>

 <div id="div1">div1 content</div></br>
 <div>div2 content</div>

 <script type="text/javascript">
    document.getElementById("div1").style.color = "red";
 </script>

</body>
</html>
```

RESULT

div1 content

div2 content

4.3 이벤트 개요 (1/6)

✓ 웹 페이지에서는 다양한 종류의 상호작용으로 인해 이벤트가 발생됩니다.

✓ 사용자가 마우스를 클릭하였을 경우, 키보드가 눌려졌을 경우 등 다양한 종류의 이벤트가 존재합니다.

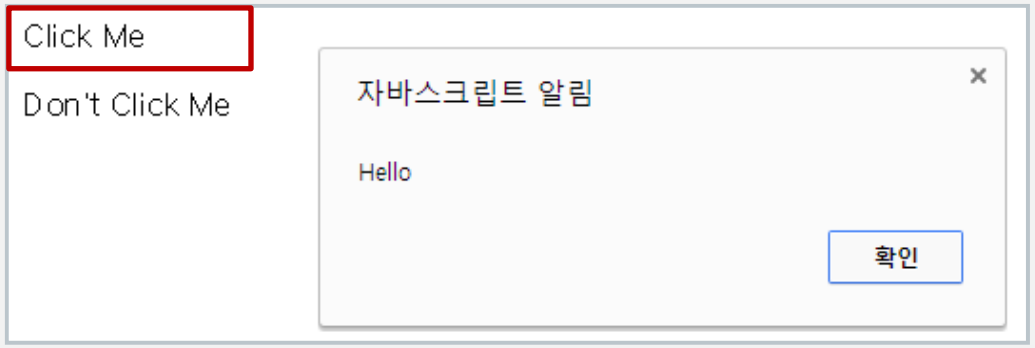
✓ JavaScript를 사용하여 DOM에서 발생되는 이벤트를 감지하고 이러한 이벤트에 대응하여 다양한 작업을 할 수 있습니다.

이벤트 발생 종류
웹 페이지가 로딩되었을 때
페이지를 스크롤 했을 때
브라우저 창의 크기를 조절했을 때
마우스를 클릭했을 때
키보드로 키를 입력했을 때
Form이 Submit되었을 때
Input의 내용이 변경되었을 때
마우스를 움직여서 Element 를 이동할 때

4.3 이벤트 개요 (2/6) – 예제

✓ 브라우저 내장 함수인 alert()을 호출하면 알림 창을 표시할 수 있습니다.
✓ click이벤트는 사용자가 마우스를 클릭했을 때 발생합니다.
✓ 특정 DOM 엘리먼트에 한하여 click이벤트를 제어할 수 있습니다.
✓ “Click Me”라는 문자열을 담고 있는 태그 영역을 클릭할 경우에만 “Hello” 알림 창이 표시됩니다.

[Click Me 영역을 클릭 시 이벤트 발생]



4.3 이벤트 개요 (3/6) – 마우스 이벤트

✓ 웹 초기에는 load, click 등 소수의 이벤트만 사용했습니다.
✓ 마우스 이벤트는 웹 애플리케이션에서 가장 많이 사용되는 이벤트입니다.
✓ 마우스 이벤트는 계층 구조를 따라 버블링 됩니다.
✓ 마우스 이벤트 핸들러에 전달되는 이벤트 객체에는 마우스 위치와 버튼 상태 등의 정보들을 담고 있습니다.

마우스 이벤트	설명
onclick	마우스로 Element를 클릭 했을 때 발생합니다.
ondbclick	마우스로 Element를 더블 클릭 했을 때 발생합니다.
onmouseup	마우스로 Element에서 마우스 버튼을 올렸을 때 발생합니다.
onmousedown	마우스로 Element에서 마우스 버튼을 눌렀을 때 발생합니다.
onmouseover	마우스를 움직여서 Element 위로 올릴 때 발생합니다.
onmouseout	마우스를 움직여서 Element 에서 벗어 날 때 발생합니다.
onmouseenter	마우스를 움직여서 Element 밖에서 안으로 들어 올 때 발생합니다.
onmouseleave	마우스를 움직여서 Element 안에서 밖으로 나갈 때 발생합니다.

4.3 이벤트 개요 (4/6) – 키보드 이벤트

✓ 키보드의 커서가 웹 브라우저에 나타나는 지점에서 키보드를 조작할 때 이벤트가 발생합니다.
✓ 키보드 조작은 운영체제에 영향을 받으므로 특정 키가 이벤트 핸들러에게 전달되지 않을 수 있습니다.
✓ 키보드 이벤트는 키보드 커서가 나타내는 위치에서 window객체까지 버블링 됩니다.
✓ 키보드 커서가 나타내는 요소가 없다면 Document에서 이벤트가 발생합니다.

이벤트	설명
onkeydown	키보드를 누르는 순간 발생합니다.
onkeypress	키보드가 눌려 졌을 때 발생합니다. (키를 눌렀다가 떼면 발생)
onkeyup	키보드 키가 눌려 졌다가 올려 질 때 발생합니다.

4.3 이벤트 개요 (5/6) – Frame 이벤트

✓ Frame 관련 이벤트는 특정 DOM 문서에 관련된 이벤트가 아니라 Frame 자체에 대한 이벤트입니다.
✓ Frame 이벤트 중에서 load 이벤트가 가장 많이 사용됩니다.
✓ load는 문서 및 자원들이 모두 웹 브라우저에 적용되면 이벤트가 수행됩니다.
✓ unload는 사용자가 브라우저를 떠날 때 이벤트가 발생하지만 사용자가 브라우저를 떠나는 것을 막을 수는 없습니다.

이벤트	설명
onabort	이미지 등의 내용을 로딩하는 도중 취소 등으로 중단 되었을 때 발생합니다.
onerror	이미지 등의 내용을 로딩 중 오류가 발생 했을 때 발생합니다.
onload	document, image, frame 등이 모두 로딩 되었을 때 발생합니다.
onresize	document, element의 크기가 변경 되었을 경우 발생됩니다.
onscroll	document, element가 스크롤 되었을 때 발생합니다.
onunload	페이지나 Frame등이 unloading 되었을 때 발생합니다.

4.3 이벤트 개요 (6/6) – 폼 이벤트

✓ 폼은 웹 초기부터 지원되어 여러 웹 브라우저에서 가장 안정적으로 동작하는 이벤트입니다.
✓ 가장 많이 사용되는 이벤트로 폼 전송될 때에는 submit 이벤트가 발생합니다.
✓ 폼이 초기화 될 대는 reset 이벤트가 발생합니다.
✓ submit과 reset은 이벤트 핸들러에서 취소할 수 있습니다.

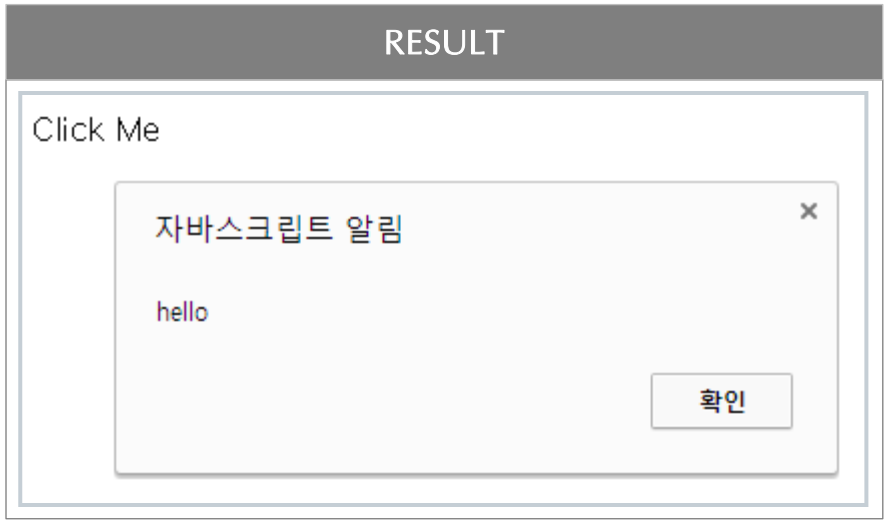
이벤트	설명
onblur	Input과 같은 Element등에서 입력 포커스가 다른 곳에서 이동할 때 발생합니다.
onchange	입력 내용이 변경 되었을 때 발생합니다.
onfocus	Input과 같은 Element에 입력 포커스가 들어 올 때 발생합니다.
onreset	입력 폼이 reset 될 때 발생합니다.
onselect	Input, Textarea에 입력 값 중 일부가 마우스 등으로 선택될 때 발생합니다.
onsubmit	form이 전송될 때 발생합니다.

4.4 이벤트 처리 (1/4) – 등록 (1/3)

✓ 이벤트를 감지하고 이에 대한 작업을 등록하는 방법은 여러 가지 방법이 존재합니다.
✓ 어떤 이벤트에 대해 작업을 등록하는 행위를 이벤트 핸들러를 등록한다고 합니다.
✓ 자바스크립트의 초기에는 HTML 요소의 내부에서 직접적으로 이벤트 핸들러를 등록하여 사용했습니다.
✓ 이러한 방식은 HTML 코드에 JavaScript 코드가 침범한다는 단점이 존재합니다.

```
<!DOCTYPE html>
<html>

<body>

    <div onclick='alert("hello")'>Click Me</div>

</body>
</html>
```

RESULT

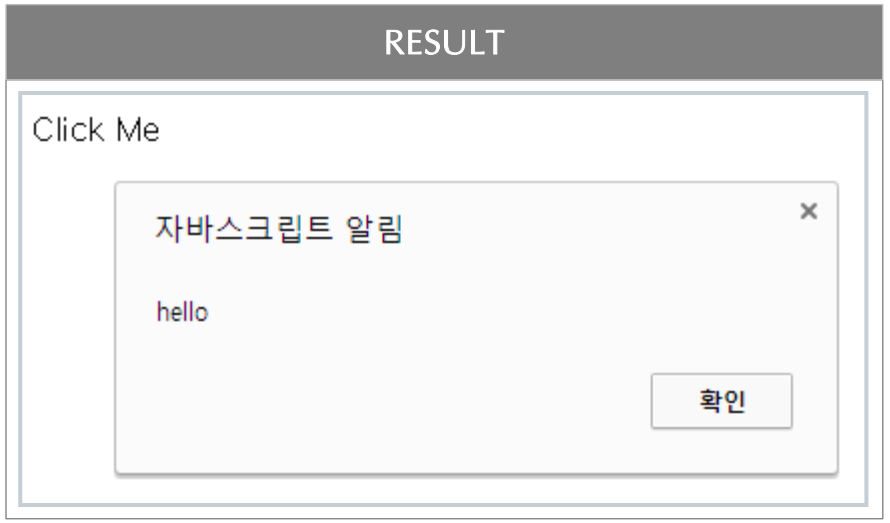


4.4 이벤트 처리 (2/4) – 등록 (2/3)

✓ HTML에 직접 이벤트 핸들러를 등록하는 방법 대신에 JavaScript에서 이벤트 핸들러를 등록하는 방법이 존재합니다.
✓ JavaScript에서 이벤트 핸들러를 등록함으로써 HTML코드를 깔끔하게 유지할 수 있습니다.
✓ 이벤트를 핸들링 할 특정 DOM을 선택하고 이벤트 핸들러를 등록합니다.
✓ "div1"엘리먼트에 클릭 이벤트가 발생하면 핸들러에 등록한 function이 실행됩니다.

```
<!DOCTYPE html>
<html>
<body>

    <div id="div1">Click Me</div>
    <script type="text/javascript">

     document.getElementById("div1").onclick = function()
     {
       alert("hello");
     };

    </script>

</body>
</html>
```

RESULT



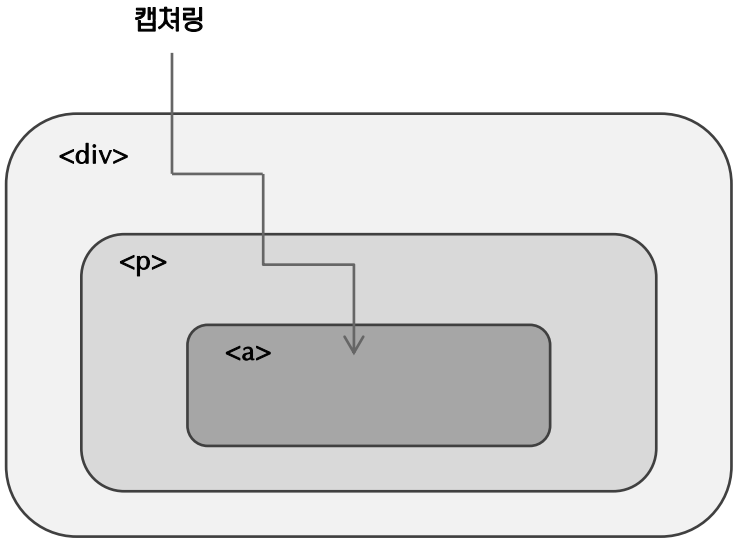
4.4 이벤트 처리 (3/4) – 등록 (3/3)

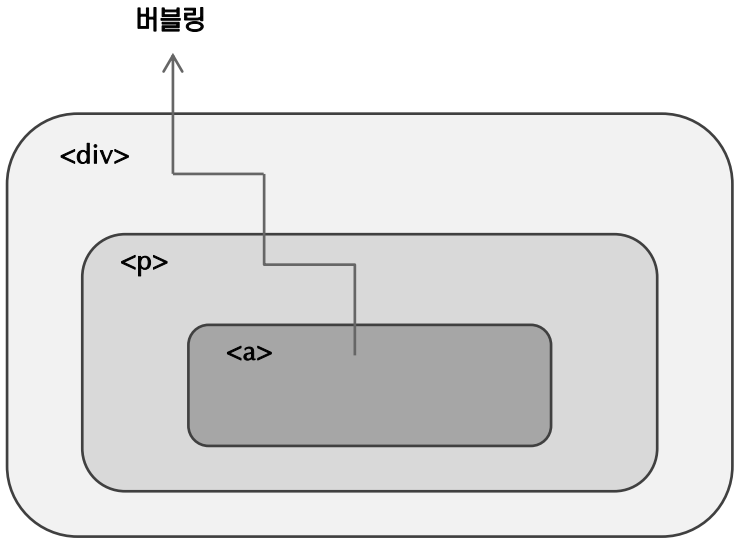
✓ 2000년에 발표된 DOM 레벨2 이벤트 명세의 addEventListener 를 이용하여 좀더 세밀한 이벤트 제어가 가능합니다.
✓ 첫 번째 매개변수로 이벤트명, 두 번째 매개변수로 이벤트 핸들링 함수, 세 번째 매개변수로 캡쳐링 여부를 전달합니다.
✓ 첫 번째 매개변수의 이벤트명에는 “on”을 제거한 이벤트명을 사용합니다.
✓ 이 방식은 Internet Exploler 9 이전 버전에서는 사용할 수 없습니다.

```
<!DOCTYPE html>
<html>
<body>

    <div id="div1">Click Me</div>
    <script type="text/javascript">

     // DOM 레벨2 이벤트 명세 방법
     // 이벤트명에 on을 생략
     // Internet Exploler 9 이전버전 사용불가
     document.getElementById("div1").addEventListener(“click”, fn_click, false);

     // Internet Exploler 9 이전버전에서 사용하기 위한 방법
     document.getElementById("div1").attachEvent(“onclick”, fn_click);

     // 이벤트 핸들링 함수
     function fn_click() {
       alert("hello");
     };
    </script>

</body>
</html>
```

4.4 이벤트 처리 (4/4) – 버블링과 캡쳐링

✓ 이벤트 발생 요소부터 요소를 포함하는 부모요소까지 올라가면서 이벤트를 검사하는 것을 이벤트 버블링이라 합니다.
✓ 이벤트 버블링에서 버블속성의 이벤트 핸들러가 등록되어 있으면 수행됩니다.
✓ 이벤트가 발생한 요소를 포함하는 부모 HTML로부터 이벤트 근원지인 자식요소까지 검사하는 것을 캡쳐링이라 합니다.
✓ 이벤트 캡쳐링에서 캡쳐속성의 이벤트 핸들러가 등록되어 있으면 수행됩니다.





4.5 이벤트 활용

✓ 하나의 DOM엘리먼트에 복수의 이벤트 핸들러를 등록할 수 있습니다.
✓ 마우스가 특정 DOM엘리먼트 영역 안으로 들어온 경우 mouseenter이벤트가 발생합니다.
✓ 반대로 마우스가 특정 DOM엘리먼트 영역 밖으로 나간 경우 mouseleave 이벤트가 발생합니다.
✓ 태그에 mouseenter, mouseleave 2가지 이벤트 핸들러를 등록합니다.

```
<!DOCTYPE html>
<html>
<body>
   <span id="span1"
         style="background:black;color:white">span1</span>

   <script>
   var span1 = document.getElementById("span1");

   // mouseenter 이벤트핸들러 등록
   span1.addEventListener("mouseenter", fn_mouseEnter, false);
   // mouseleave 이벤트핸들러 등록
   span1.addEventListener("mouseleave", fn_mouseLeave, false);
   // mouseenter 핸들러 함수
   function fn_mouseEnter() {
     span1.style.backgroundColor = "red";
   }
   // mouseleave 핸들러 함수
   function fn_mouseLeave() {
     span1.style.backgroundColor = "blue";
   }
   </script>
</body>
</html>
```

Default 상태

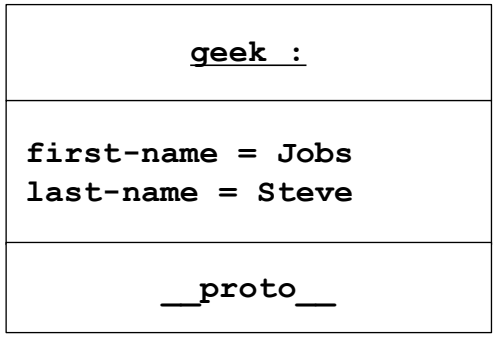
mouseenter 이벤트 발생 시

mouseleave 이벤트 발생 시


5. JavaScript 활용기술

5.1 객체 (1/6) – 개요

✓ 객체는 이름과 값으로 구성된 프로퍼티의 집합입니다.
✓ 문자열, 숫자, true/false, null, undefined을 제외한 모든 값은 객체입니다.
✓ 전역 객체를 제외한 JavaScript 객체는 프로퍼티를 동적으로 추가하거나 삭제가 가능합니다.
✓ JavaScript 객체는 프로토타입(prototype)이라는 특별한 프로퍼티를 포함합니다.



5.1 객체 (2/6) – 속성 값 조회

✓ 객체는 마침표(.)을 사용하거나 대괄호([])를 사용해서 속성 값에 접근합니다.
✓ 객체에 없는 속성에 접근하면 undefined를 반환합니다.
✓ 객체 속성 값을 조회 할 때 || 연산자를 사용하는 방법도 많이 사용됩니다.
✓ (예 : var middle = stooge[“middle-name”] || “none”;)

```
<html>
<body>
  <script type="text/javascript">
     // 객체 리터럴
     var empty_object = {};
     var geek = {
       "first-name" : "Jobs",
       "last-name" : "Steve"
     };
  </script>
</body>
</html>
```

5.1 객체 (3/6) – 속성 값 변경

✓ 속성 값 변경은 마침표(.)을 사용하거나 대괄호([])를 사용해서 변경 합니다.

✓ 예 : flight[“airline”] = “sunder”;, flight.airline = “sunder”;

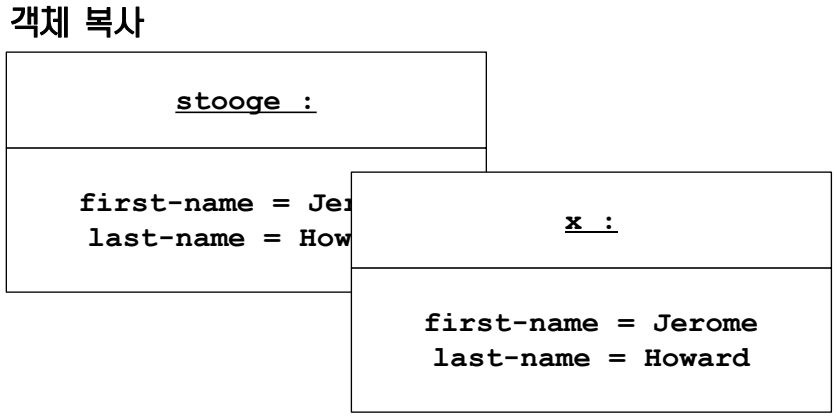
✓ 속성이 객체 내에 없는 경우는 해당 속성이 추가됩니다.

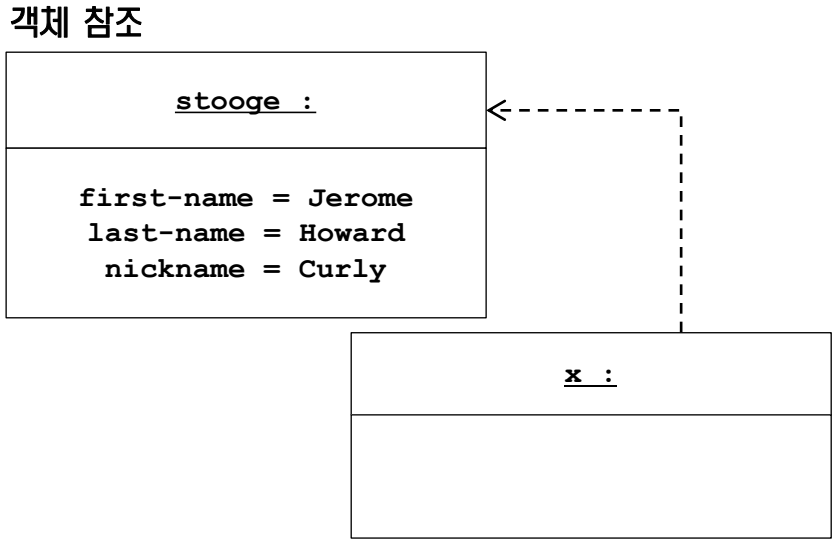
✓ 예 : flight[“nickName”] = “super - phonex”;

```
<html>
<body>
  <script type="text/javascript">
     // 객체 리터럴
     var flight = {
           airline : "phonex",
           number : 123,
           departure : {
             IATA : "INC",
             time : "",
             city : "Inchon"
           },
           arrival : {
             IATA : "LAX",
             time : "2004-09-23 10:42",
             city : "Los Angeles"
           }
         };
  </script>
</body>
</html>
```

5.1 객체 (4/6) – 참조

✓ 객체는 복사되지 않고 참조 됩니다.
✓ JavaScript에서 기본 데이터 타입이 아닌 모든 값은 참조 타입입니다.
✓ 참조 타입은 Object, Array, Date, Error를 포함합니다.
✓ 간단한 타입 확인 방법으로는 typeof 연산자가 있습니다. (null도 typeof 연산자는 object를 반환합니다.)

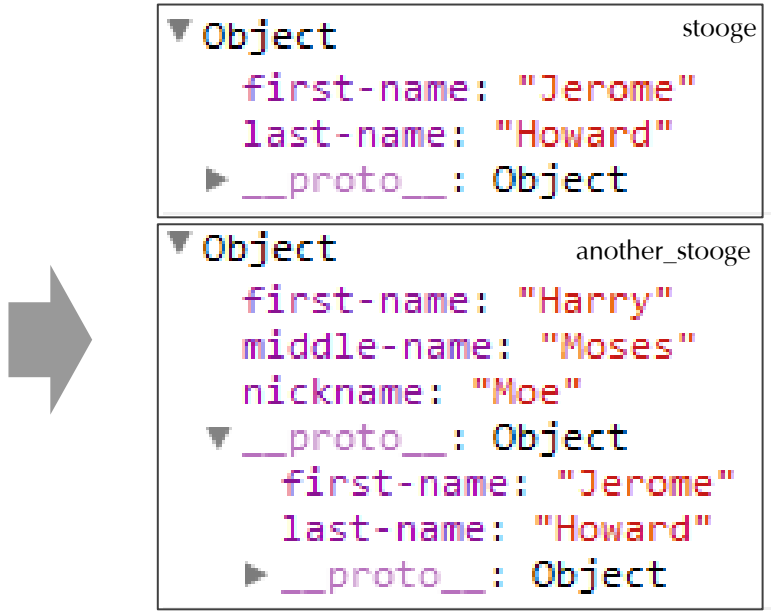




5.1 객체 (5/6) – 프로토타입

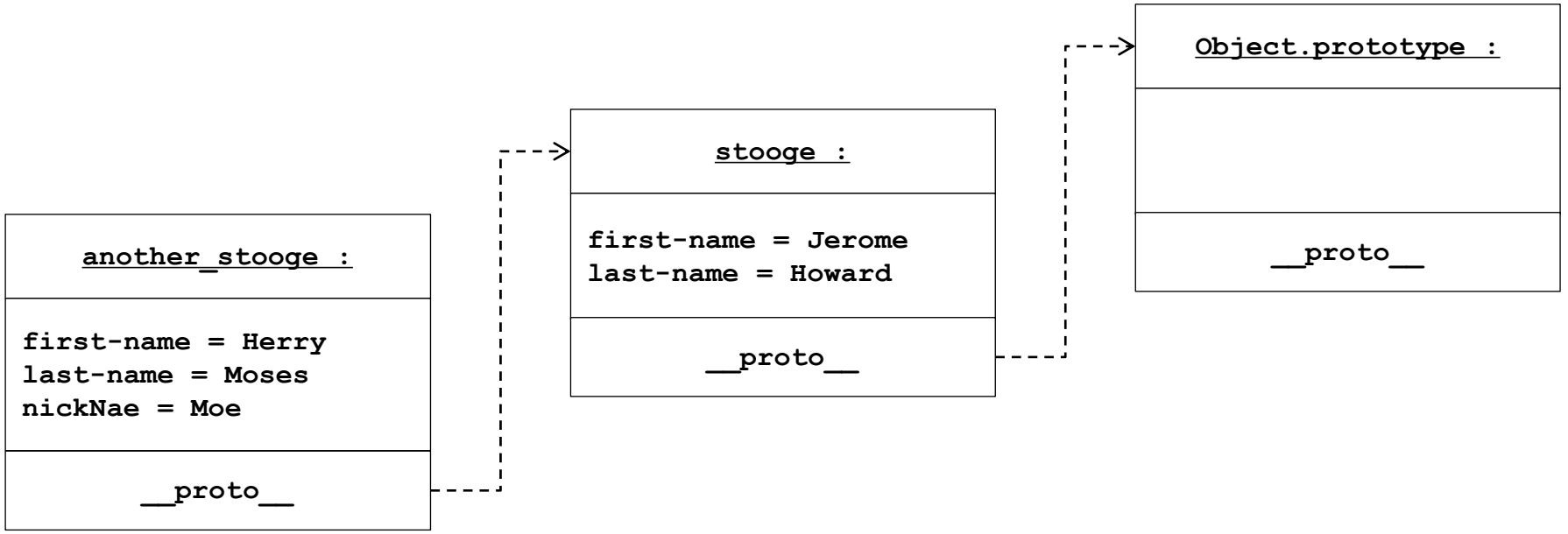
✓ 객체 리터럴로 생성되는 모든 객체는 Object.prototype에 연결됩니다.
✓ 객체 리터럴로 생성된 객체에서 프로토타입에 연결된 객체의 값 변경은 불가능합니다.
✓ 객체 리터럴로 생성된 객체에서 프로토타입에 연결된 객체의 속성 값을 변경하면 자신의 속성값이 생성되거나 변경됩니다.
✓ 프로토타입 연결은 객체의 프로퍼티를 읽을 때만 사용 가능합니다.

```
if (typeof Object.create !== 'function') {
   Object.create = function(o) {
      var F = function() {};
      F.prototype = o;
      return new F();
   };
}
var another_stooge = Object.create(stooge);

another_stooge['first-name'] = 'Harry';
another_stooge['middle-name'] = 'Moses';
another_stooge.nickname = 'Moe';
```



5.1 객체 (6/6) – 프로토타입 체인

✓ 객체 리터럴로 생성된 객체에서 속성 값을 읽을 때 속성이 없으면 연결된 프로토타입에서 속성을 찾으려고 합니다.
✓ 프로토타입으로 연결된 객체를 탐색하는 시도는 Object.prototype을 만날 때 가지 이어집니다.
✓ 프로토타입으로 찾으려는 속성 값을 끝까지 찾지 못하면 undefined를 반환합니다.
✓ 프로토타입 체인은 웹 브라우저마다 다를 수는 있으나 일반적으로 __proto__로 형성됩니다.



5.2 유효범위 (1/2) – 개요

✓ 변수와 매개변수의 접근성과 생존기간을 제어합니다.
✓ 유효범위는 이름이 충돌하는 문제를 줄여주고 메모리를 효율적으로 운영할 수 있도록 합니다.
✓ JavaScript에서는 블록({ })유효범위를 지원하지 않습니다.
✓ 함수 내에서 var로 선언하면 지역 변수가 되고 함수 외부에서는 참조할 수 없습니다.

```
var park = function () {
   var d = 3, f = 5;

   var kim = function () {
     var f = 7, c = 11;
     // d:3, f:7, c:11
     d += f + c;
     // d:21, f:7, c:11
   };

   // d:3, f:5, c:undefined
   kim();
   // d:21, f:5
}
```

5.2 유효범위 (2/2) – 예제

✓ 지역유효범위의 변수는 함수들이 서로 공유할 수 없습니다.
✓ 함수 내에서 변수를 참조할 때 선언된 이름이 없다면 전역변수에 선언된 이름의 값을 반환합니다.
✓ 함수 내에서 var 키워드를 생략하여 변수를 선언하면 전역변수가 됩니다.
✓ 함수의 유효범위를 벗어나는 변수를 가진 함수를 클로저(Closure)라고 합니다.

```
function f1() {
   var a = 1;
   f2();
}

function f2() {
   return a;
}

f1();
// a is not defined
```

1. f1()에서 f2()를 호출합니다..
2. f1()과 f2()는 a변수(지역유효범위)를 공유할 수 없습니다.

```
var a = 5;
f1();
// 5

var a = 55;
f1();
// 55
```

1. f1()에서 f2()를 호출합니다.
2. f2()는 함수 유효범위 내에 a가 없으므로 전역변수 a의 값을 반환합니다.

```
var f2 = function () {
  return a * 2;
}

var a = 5;
f1();
// 10
```

1. f2() 함수가 선언됩니다.
2. f1()에서 f2()를 호출합니다.
3. f2()는 함수의 유효범위 내에 a가 없으므로 전역변수 a의 값인 5에 2를 곱한 결과10을 반환합니다.

5.3 함수 전달인자

✓ 함수 내에서(body) 사용 가능한 식별자로서 전달인자 객체를 참조하는 특별한 프로퍼티가 arguments 입니다.
✓ arguments는 함수에 전달된 전달인자배열로서 숫자를 이용해 접근하는 방법을 제공합니다.
✓ JavaScript는 오버로딩을 지원하지 않으므로 일부에서는 arguments를 이용해서 이를 구현하기도 합니다.
✓ arguments는 callee 프로퍼티를 제공하는데 현재 실행되고 있는 함수를 나타냅니다.

```
<html>
<body>
  <script type="text/javascript">
    function max() {
      var m = Number.NEGATIVE_INFINITY;
      for (var i = 0; i < arguments.length; i++) {
        if (arguments[i] > m)
          m = arguments[i];
      }
      return m;
    }

    alert(max(1, 2, 1000, 300, 99));
</script>
</body>
</html>
```

```
<html>
<body>
  <script type="text/javascript">
    function f(x) {
      console.log(x);
      arguments[0] = 100;
      console.log(x);
    }
    f(1);
    // 1
    // 100
</script>
</body>
</html>
```

```
<html>
<body>
  <script type="text/javascript">
    console.log(function(x) {
      if (x <= 1)
        return 1;
      return x + arguments.callee(x - 1)
      }(10));
  </script>
</body>
</html>
```

5.4 this

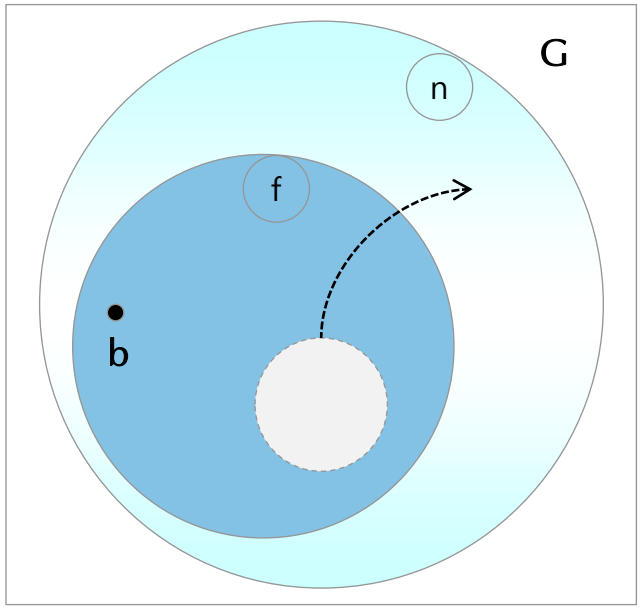
✓ 생성자 내부에서 사용되는 this는 최초 생성자를 사용해서 만들어지는 객체를 나타냅니다.
✓ 객체의 함수에서 사용되는 this는 객체를 나타냅니다.
✓ 전역함수에서 사용되는 this는 전역객체를 나타냅니다.
✓ 함수 빌려 쓰기(call, apply)와 같은 외부함수를 사용할 경우에는 전달된 객체를 나타냅니다. (첫 번 째 전달인자)

```
<html>
<body>
  <script type="text/javascript">

    var calculator = {
      operand1 : 1,
      operand2 : 2,
      compute : function() {
        this.result =
        this.operand1 + this.operand2;
      }
   }

   calculator.compute();
   console.log(calculator.result); // 3
```

```
    var calculator = {
      operand1 : 1,
      operand2 : 2,
      compute : function() {
        this.result =
          this.operand1 + this.operand2;
      }
    }

    var calculator2 = {
      operand1 : 4,
      operand2 : 7
    }

    calculator.compute.apply(calculator2, []);
    console.log(calculator2.result); // 11

</script>
</body>
</html>
```

5.5 클로저 (1/4)

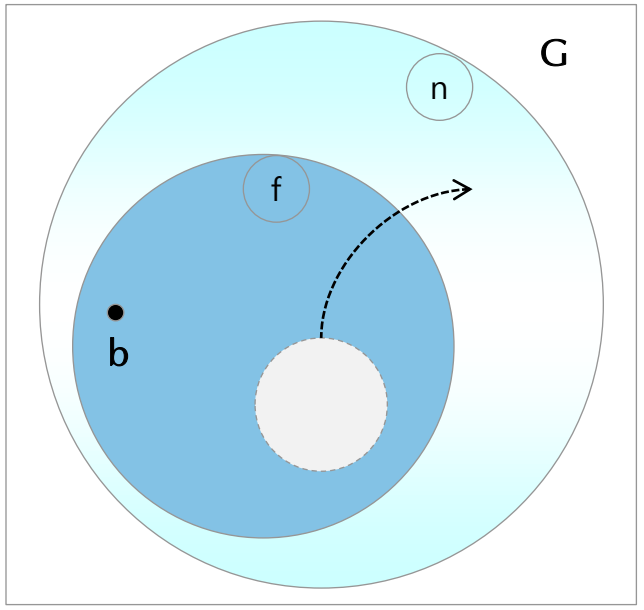
✓ 클로저는 자신의 유효범위에 제약을 받지 않는 변수를 참조하는 함수입니다.
✓ 아래 예제에서 f는 또 다른 함수를 반환합니다.
✓ 반환된 함수는 자신의 로컬영역, f의 영역, 글로벌영역에 접근이 가능합니다.
✓ 반환된 새로운 전역함수는 f의 로컬영역에 접근이 가능합니다.




```
function f() {
   var b = "b";
   return function() {
      return b;
   }
}

var n = f();
console.log(n());
// "b"
```

5.5 클로저 (2/4)

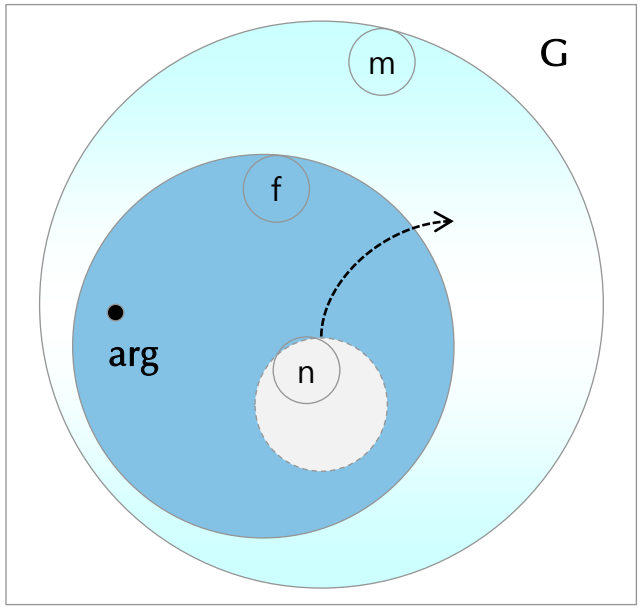
✓ 함수는 어휘적 유효범위를 가집니다.
✓ 아래 예제에서 n에 새로운 함수가 선언되었고 var문을 생략했기 때문에 n은 전역변수가 됩니다.
✓ n함수는 f함수 내부에서 선언되었으므로 f의 유효범위에 접근이 가능합니다.
✓ n이 글로벌영역에 위치하더라도 f의 유효범위에 접근이 가능합니다.




```
var n;
function f() {
   var b = "b";
   n = function() {
      return b;
   }
}

f();
console.log(n());
// "b"
```

5.5 클로저 (3/4)

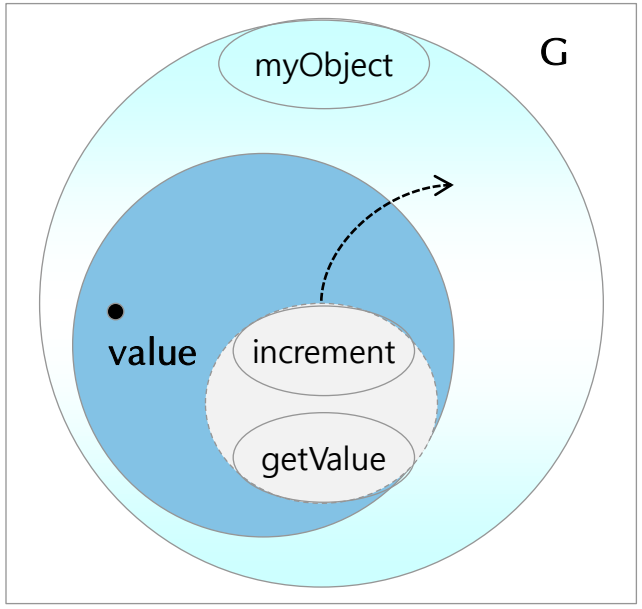
- ✓ 클로저는 하나의 함수가 부모함수로부터 반환된 후 부모함수의 유효범위와 연결 될 때 생성됩니다.
- ✓ 함수에 매개변수를 전달해서 지역변수처럼 사용할 수 있습니다.
- ✓ 아래 예제에서 n이 선언된 후에 arg 매개변수의 값이 증가합니다.
- ✓ m이 호출되었을 때 증가된 f의 arg에 접근하여 변환됩니다.




```
function f(arg) {
   var n = function() {
      return arg;
   };
   arg++;
   return n;
}

var m = f(123);
console.log(m());
// 124
```

5.5 클로저 (4/4)

- ✓ 유효범위 체인 규칙을 이해한다면 클로저를 이해하는데 도움이 됩니다.
- ✓ 아래 예제에서 value는 함수 내에서만 사용 가능한 지역변수로 선언됩니다.
- ✓ increment와 getValue를 통해서만 value에 접근이 가능합니다.
- ✓ increment와 gatValue를 제외한 나머지 부분에서는 value에 접근이 불가능 합니다.




```
var myObject = function() {
   var value = 0;
   return {
      increment : function(inc) {
         value += typeof inc === 'number' ? inc : 1;
      },
      getValue : function() {
         return value;
      }
   };
}(); // 즉시실행함수

myObject.getValue(); // 0
myObject.increment(5);
myObject.getValue(); // 5

value; // value is not defined
```