6

선언문

지금까지는 액션스크립트에서 데이터를 저장하고 조작하는 방법에 대해 배웠다. 이 장에서는 그러한 데이터를 가지고 직접 작업을 수행하는 방법에 대해 알아보자. 플래시 명령어는 인터프리터에 어떤 작업을 수행하도록 지시를 내리는 선언문 또는 코드 구문의 형태로 사용한다.

플래시에서 (소리가 멈추거나 무비를 재생하거나 함수를 실행하거나 어떤 코드를 반복해서 실행시키는 것과 같은) 어떤 일을 처리하려면 선언문을 이용해야 한다. 사실 액션스크립트 프로그램은 플래시에서 어떤 일을 하도록 인터프리터에 명령을 내리는 선언문 목록이라고 정의할 수 있다. 아래 액션스크립트 코드도 하나의 완성된 액션스크립트 프로그램이며, 네 개의 선언문만으로 인터프리터에 어떤 웹 페이지를 브라우저로 읽어들이라는 명령을 내릴 수 있다.

```
var protocol = "http";                                  // 1번 선언문
var domain = "www.moock.org";                           // 2번 선언문
var path = "webdesign/flash/";                          // 3번 선언문
getURL(protocol + "://" + domain + "/" + path);         // 4번 선언문
```

액션스크립트를 이용하여 무비 스크립트를 만드는 일은 단지 프레임, 무비 클립, 버튼과 같은 것에 선언문을 추가하는 일에 지나지 않는다. 이 장에서는 선언문

을 체계적으로 구성하는 방법에 대해 배우고 일반적인 선언문의 범주에 대해 알아보기로 하자. 이 장에서 액션스크립트에서 쓰이는 모든 선언문을 언급하기는 하겠지만, 몇 가지 중요한 내용은 이 장이 끝나고 난 뒤에 다른 장에서 다시 자세히 설명하겠다.

선언문의 유형

개념적으로 본다면 선언문에는 다섯 가지 핵심적인 유형이 있다.

프로그램의 실행 흐름을 제어하는 선언문

순환문

조건문

ifFrameLoaded

변수를 선언하는 선언문

var

set

함수를 선언하고 호출하고 함수에서 값을 리턴하는 선언문

function

함수 호출

call

return

객체를 다루기 위한 선언문

with

for ... in

데이터 값을 나타내는 선언문

임의의 표현식(특히 부가적인 효과가 있는 표현식)

위와 같이 범주를 나누어 보면 선언문을 이용하여 인터프리터에 어떤 작업을 명령할 수 있는지 이해하기 쉽다. 언뜻 보면 이 목록에 모든 선언문이 들어 있지 않은 것 같아 보이지만, 다시 한 번 생각해 보면 조건문이나 순환문에는 다양한 종류가 있으며 함수 호출을 통해 수많은 작업을 처리할 수 있다는 것을 깨달을 수 있다. 우선 선언문이 어떻게 형성되는지 알아보자.

선언문의 문법

선언문은 일반적으로 키워드와 표현식으로 구성된다. 이미 앞에서 배운 var 선언문은 변수를 선언하는 데 쓰이며 추가적으로 변수를 초기화할 수도 있다. var 선언문의 일반적인 문법은 다음과 같다.

```
var numFrames;
```

키워드(이 경우에는 var)는 선언문의 시작을 나타낸다. 그 뒤에는 선언문에 따라 필요한 구문이 나온다. 위 예제에서는 단순히 변수의 이름으로 numFrames가 쓰였다. 마지막으로 세미콜론으로 선언문이 끝났음을 표시한다.

액션스크립트에서는 모든 선언문을 끝마칠 때 세미콜론을 사용한다(세미콜론을 사용하는 형식은 좋은 형식이긴 하지만 공식적으로 반드시 필요한 것은 아니다. '14장. 렉시컬 구조'에서 세미콜론을 사용하는 것에 대해 조금 더 생각해 보자).

선언문 중에는 여러 가지 형식을 사용할 수 있는 것도 있다. 예를 들어 var 선언문에서는 초기 값을 설정해도 되고 하지 않아도 된다(아래 경우에는 10을 x에 대입한다).

```
var x;       // 선언만 하는 경우
var x = 10;  // 선언하면서 값을 대입하는 경우
```

이 장 전반에 걸쳐 나머지 선언문의 문법에 대해 살펴볼 것이다.

선언문 블록

선언문에 따라 다른 선언문, 즉 '하위 선언문(substatement)'을 포함하고 있는 것도 있다. 예를 들어 if 선언문은 다음과 같은 형식으로 쓰인다.

```
if (expression) substatement;
```

substatement는 if문의 expression이 true인 경우에만 실행되는데 다음처럼 변수 선언 선언문과 같이 하나의 선언문일 수도 있다.

```
if (x == 5) var numFrames = 2;
```

또는 '선언문 블록(statement block)'으로 묶어 놓은 일련의 선언문일 수도 있다.

```
if (x == 5) {
  var numFrames;
  numFrames = 10;
  play();
}
```

위에서 볼 수 있듯이 선언문 블록은 여러 줄에 걸친 여러 개의 선언문을 중괄호({})로 묶어 놓은 것이다.

```
{ statement1; statement2; statement3... }
```

선언문 블록을 if 선언문의 하위 선언문으로 사용하여 원래는 하나의 선언문만 들어갈 수 있는 부분에 여러 개의 선언문을 사용할 수 있다. 쓰다 보면 이러한 기능이 상당히 편리하다는 것을 느끼게 될 것이다.

액션스크립트에서 하나의 선언문이 들어가는 곳이라면 어디든지 선언문 블록을 사용할 수 있다. 선언문 블록을 반드시 사용해야 하는 경우도 있다. 예를 들어 function 선언문에서는 그 함수가 한 줄로 끝나는 함수라고 하더라도 선언문 블록을 사용해야 한다.

```
function aheadTwoFrames() {
  gotoAndStop(_currentframe + 2);
}
```

가독성을 높이기 위해 선언문 블록은 일반적으로 다음과 같은 형식으로 사용한다.

```
parent_syntax {
   substatement1;
   substatement2;
   substatement3;
}
```

여기서 parent_syntax는 선언문 블록을 정의해야 하는 선언문을 나타낸다.

위 코드에서는 중괄호가 시작되는 부분을 parent_syntax가 있는 첫째 줄 끝에 놓았다. 선언문 블록의 하위 선언문은 각각 한 줄에 하나씩, 두 칸 들여쓰기된 상태로 입력되어 있다. 마지막으로 오른쪽 중괄호는 한 줄에 분리되어 입력해 놓았으며 원래 선언문과 같은 줄에 맞추어 들여쓰기를 한다. 선언문 블록에 있는 하위 선언문은 세미콜론으로 끝나야 하지만 중괄호가 있는 줄에서는 세미콜론을 사용하지 않는다.

들여쓰기 형식은 문법에 반드시 필요한 것은 아니고 관용적으로 쓰일 뿐이다. 이 책에서는 플래시 액션스크립트 데이터에서 사용하는 스타일을 사용한다.

액션스크립트 선언문

지금까지 일반적인 선언문 형식을 배웠다. 〔표 6-1〕을 쭉 훑어보고 액션스크립트 선언문의 기능을 익히도록 하자.

〔표 6-1〕 액션스크립트 선언문

선언문	문법	기능
break	break;	루프를 취소한다.
call	call (frame);	원격 프레임 있는 스크립트를 실행한다.
continue	continue;	현재 순환문을 다시 시작한다.
do-while	do { *statements* } while (expression);	while 순환문을 약간 변형한 순환문
비어있는 선언문	;	선언문이 들어갈 자리. 일반 모드에서 evaluate와 함께 쓰인다.

for	for (init; test; increment) {
	statements
	}
		어떤 코드를 반복하여 실행한다.

선언문	문법	기능

for-in	for (property in object) {
	statements
	}
		객체의 모든 속성에 대하여 루프를 돌린다.

function	function name(parameters) {
	statements
	}
		함수를 선언한다.

if-else if-else	if (expression) {
	statements
	} else if (expression) {
	statements
	} else {
	statements
	}
		하나 이상의 조건을 기준으로 코드를 실행한다.

ifFrameLoaded	ifFrameLoaded (frame) {
	statements
	}
		특정 프레임을 로드하면 코드를 실행한다. 플래시 5에서는 쓰이지 않는다.

return	return; return *expression*;	함수에서 빠져나가거나 함수의 어떤 값을 리턴한다.
set	set (*variable, value*);	동적으로 이름이 변하는 변수에 값을 대입한다.
var	var *variableName*; var *variableName* = *expression*;	변수를 선언하고 추가적으로 초기화할 수도 있다.

while	while (*expression*) {
	statements
	}
		코드를 반복하여 실행한다.

with	with (*objectName*) {
	statements
	}
		주어진 객체 내에서 어떤 코드를 실행한다.

순환문과 조건문

순환문과 조건문은 지금까지 여러 번 사용해 보았다. 이 두 선언문 유형은 프로그램의 흐름을 제어할 때 가장 많이 사용된다. 순환문을 이용하면 선언문을 반복해서 실행할 수 있으며, 조건문을 이용하면 특정한 상황에서만 선언문을 실행한다. 자세한 내용은 '7장. 조건문' 과 '8장. 순환문' 에서 알아보기로 하자.

표현식 선언문

모든 표현식은 선언문에 속하지만 표현식에서 아무런 작업도 처리하지 않고 그 결과로 아무런 작업도 하지 않는다면 그 표현식은 무의미하다.

```
"hi there";  // 별 역할이 없다.
345 + 5;     // 이것도 마찬가지
```

하지만 어떤 표현식은 변수나 속성에 값을 대입하거나 플래시 환경을 직접 바꿈으로써 시스템 상태를 변경하는 부수적인 효과를 가져온다. 아래 예제에서 쓰인 표현식은 x의 값을 변화시킨다.

```
x = 345 + 5;  // 이러한 표현식은 유용하다.
```

함수 호출은 표현식 선언문 중 가장 강력한 유형에 속한다. 함수에서 유용한 값을 리턴하지 않는다고 하더라도 매우 유용한 부수 효과가 있을 수 있다. 예를 들어 gotoAndStop( ) 함수는 undefined 값을 리턴하지만 플레이헤드를 다른 프레임으로 옮겨주는 매우 중요한 부수 효과를 가져온다.

```
gotoAndStop(5);  // _currentframe 값이 5로 바뀐다.
```

함수 호출에 관한 자세한 내용은 '9장. 함수'의 '함수 실행' 부분에서 배워보자.

var 선언문

var 선언문에서는 새로운 변수를 선언한다.

```
var x;
```

var 선언문에서 바로 새로운 변수의 초기 값을 지정할 수도 있다.

```
var x = 10;
```

함수 밖에서 var 선언문을 사용하면 선언문으로 포함하고 있는 타임라인 전체에서 사용할 수 있는 변수가 만들어진다. 함수 내부에서 var 선언문을 사용하면 그 함수에서만 사용할 수 있는 지역변수가 만들어진다(즉 함수를 사용하고 나면 없어진다). 이 부분에 대한 내용은 '2장. 변수'를 참조하기 바란다.

set 선언문(변수 설정)

대부분 변수에 어떤 값을 대입할 때는 다음과 같이 대입 표현식을 선언문으로 사용한다.

```
x = 30;
```

하지만 이러한 표현식을 사용하려면 변수 이름을 먼저 알아야 한다. 변수 이름을 동적으로 만들어서 그 변수에 어떤 값을 대입할 때는 다음과 같은 형태로 set 선언문을 사용하면 된다.

```
set (variable, expression);
```

여기서 variable은 변수 이름으로 쓰일 문자열 표현식이고 expression은 그 변수에 대입할 값이다.

```
var x;
set ("x", 10);      // x의 값이 10이 된다.

var firstName;
set ("first" + "Name", "jane");  // firstName의 값이 "jane"이 된다.
```

아래 예제는 조금 까다로운데, set 선언문에서 변수 y 값을 선언하는 것이 아니라 y 값("x")을 읽어들여서 그 문자열을 변수 이름으로 사용하고 그 변수에 값을 대입한다.

```
// 주의 깊게 살펴보자.
var y = "x";
var x;
set(y, 15);  // x의 값이 15가 되고, y는 그대로 "x"이다.
```

플래시 4에서는 set 선언문을 Set Variable이라는 액션으로 간주한다. 플래시 4에서는 프로그래밍을 통해 순서대로 이름을 붙인 변수에 동적으로 값을 대입할 때 주로 이 기능을 사용한다. 이러한 방식으로 플래시 4 액션스크립트에서는 지원되지 않는 변수(배열)를 흉내낼 수 있다. 플래시 5에서는 배열이 추가되었기 때문에 set을 거의 사용하지 않는다. set 선언문을 이용하여 배열과 같은 효과를 내는 것에 대해서는 2장을 참조하기 바란다.

function 선언문

변수를 선언(생성)할 때 var 선언문을 사용하는 것과 마찬가지로 함수를 선언할 때는 function 선언문을 이용한다. function 선언문은 다음과 같은 형식으로 쓰인다.

```
function funcName (param1, param2, param3,...paramn) {
  statements
}
```

이 선언문은 function이라는 키워드로 시작한다. funcName은 선언하고자 하는 함수의 이름이다. param1에서 paramn까지는 함수를 실행할 때 필요한 인자를 정의한다. statements에는 함수가 호출되었을 때 실행할 한 개 이상의 선언문이 들어간다.

function 선언문에서는 나중에 사용할 수 있도록 함수를 만들긴 하지만 그 함수를 실행하진 않는다. 함수를 실행하려면 함수 호출 선언문에서 그 함수의 이름을 사용하면 된다.

함수 호출 선언문

함수 호출 선언문에서는 내장 함수 또는 사용자 정의 함수를 실행한다. 함수를 호출할 때는 함수의 이름을 사용하며 함수에서 작업을 처리하는 데 필요한 인자를 전달해야 한다. 함수 이름을 사용하여 그 함수를 실행시키는 것을 보통 함수를 '호출(call, evoke)' 또는 '실행(run)' 한다고 부른다. 함수 호출은 일반적으로 다음과 같은 형식으로 사용한다.

```
funcName (arg1, arg2, ... argn);
```

funcName은 실행하려는 함수의 이름, arg1에서 argn까지는 함수를 실행하는 데 필요한 인자(입력 값)의 목록이다.

함수 호출 선언문은 엄청나게 강력하면서도 아주 기본적인 도구로, 플래시 무비를 제어하는 가장 기본적인 방법이다. 무비를 제어할 때는 보통 함수를 호출한다. 몇 가지 예를 들어보면 다음과 같다.

```
play();                          // 현재 무비를 재생한다.
gotoAndStop(5);                  // 플레이헤드를 5번 프레임으로 보낸다.
startDrag("crosshair", true);    // "crosshair" 인스턴스가 마우스 포인터를
                                 // 따라 움직이게 만든다.
```

또한 객체에서 함수 호출을 이용하여 메소드를 호출할 수도 있다.

```
circle.area();
today.getDate();
```

무비 클립 인스턴스도 객체이므로 다음과 같은 메소드 호출을 스크립트에서 자주 사용하게 된다.

```
ball.play()
intro.gotoAndStop("end");
```

함수를 사용하는 법은 9장에서, 함수가 객체의 메소드가 되는 방법은 12장에서 자세하게 배울 것이다.

call() 선언문

기초적인 수준에서 본다면 함수는 프로그램이 실행되는 동안 언제나 재사용할 수 있는 일련의 선언문이다. 플래시 4에서는 함수를 제대로 지원하지 않기 때문에, 원격 호출 활성화를 통해 함수와 비슷한 기능을 제공한다. 플래시 4에서는 레이블이 붙어 있는 프레임에 선언문 목록을 붙여 함수와 비슷한 기능을 구현할 수 있다. 이러한 식으로 만들어진 유사함수는 call() 선언문을 통해 실행시킨다.

```
call (frame);
```

call() 선언문에서는 frame이 가리키는 프레임에 있는 스크립트를 실행한다. 이때 frame은 프레임 레이블 또는 프레임 번호이다. 주어진 프레임을 로드할 수 없으면 call() 선언문은 다른 특별한 반응이 없이 그냥 끝난다.

물론 플래시 5의 실제 함수에 비하면 플래시 4의 유사함수는 그리 좋지 않기 때문에, 플래시 5 이후 버전을 위한 프로그램을 만들 때는 call() 선언문을 사용할 이유가 전혀 없다. 하지만 플래시 4 무비를 만들 때는 예전에 쓰던 방식대로 서브루틴을 만들고 플래시 4의 call() 선언문을 사용해야 한다.

return 선언문

함수를 호출하고 실행할 때 필요한 한 개 이상의 값(인자)을 전달할 수 있다. 함수에서도 마찬가지로 리턴 값(함수를 실행한 결과로 나오는 값으로 함수를 호출한 부분으로 다시 보내진다)을 전달한다. 함수 내부에서는 return 선언문을 이용하여 함수 실행을 마치거나 어떤 값을 리턴할 수 있다. return 선언문은 다음 두 가지 중 한 가지 형식으로 쓰인다.

```
return;
return expression;
```

만약 expression을 포함시키면 그 값이 함수 값으로 리턴된다. return 선언문에 리턴 값을 적어주지 않으면 그냥 함수를 끝마치고 undefined 값을 리턴한다. return 선언문을 사용하면 반드시 함수를 끝내도록 되어 있다. 하지만 함수에서 return 선언문을 반드시 사용해야 하는 것은 아니다. return 선언문이 없는 함수는 함수 본체에 있는 마지막 선언문이 실행되고 나면 자동으로 함수 동작이 끝나며 이 때 undefined라는 값을 리턴한다. 함수를 만들고 호출하고 종료하는 것과 관련된 자세한 사항은 9장을 참조하기 바란다.

with 선언문

with 선언문은 객체 이름을 반복해서 입력하지 않고도 같은 객체의 속성을 계속해서 참조할 수 있도록 해주는 선언문이다. with 선언문은 다음과 같은 형식으로 쓰인다.

```
with (object) {
  statements
}
```

with 선언문 블록 안에서 어떤 속성을 참조하면 object에 그 속성이 들어 있는지 검사한다. 만약 object에 그 속성이 있으면 그 객체의 속성으로부터 필요한 속성을 참조하게 된다. 만약 그 속성이 object에 들어 있지 않으면 현재의 타임라인 또는 함수로부터 원하는 속성을 찾는다.

다음 예는 with 선언문 안과 밖에서 선언문을 실행할 때의 차이점을 보여준다.

```
PI = 10;                  // 타임라인 변수인 PI를 설정한다.
with (Math) {             // Math 안에서 선언문을 실행한다.
  trace("pi is: "+ PI);   // 3.1459...가 출력된다.
                          // 여기서 PI는 Math의 속성이다.
}
trace("PI is: "+ PI);     // 10이 출력된다(Math 객체를 사용하지 않는다).
```

객체 속성을 편리하게 사용하는 것 외에도 with를 이용하여 객체에 있는 메소드도 호출할 수 있다.

```
x = 10;
y = 11;
with (Math) {
  larger = max(x, y);
}
trace(larger);  // 11이 출력된다.
```

with 선언문의 대상인 객체에서는 새로운 속성을 정의할 수 없다. 위 예제에서 larger라는 변수는 Math 객체에 정의되어 있지 않기 때문에, 속성 레퍼런스는 with 선언문을 포함하고 있는 타임라인이나 함수에 영향을 미치게 된다. 아래 코드에서는 myClip에서 변수를 선언하려는 시도를 하지만, myClip 객체 안에 새로운 변수가 정의되지 않는다.

```
with (myClip) {
  var x = 10;  // x는 myClip이 아니라 현재 타임라인의 변수가 된다.
}
```

하지만 with를 이용하여 무비 클립 인스턴스에 영향을 미칠 수는 있다. 이 방법을 이용하면 복잡하게 여러 단계로 나누어진 인스턴스 구조를 간편하게 다룰 수 있다. 예를 들면 다음과 같은 코드를

```
_root.form.userProfile.userID = "U346BX";
_root.form.userProfile.gotoAndPlay("questionnaire");
```

아래와 같은 형태로 고칠 수 있다.

```
with (_root.form.userProfile) {
  userID = "U346BX";              // userProfile 인스턴스에 원래
                                  // 들어있는 변수를 다시 설정한다.
```

```
  gotoAndPlay("questionnaire");   // userProfile 인스턴스에 대해
                                  // 함수를 실행한다.
}
```

하지만 이러한 작업을 할 때 with를 이용하는 방법 외의 다른 방법이 없는 것은 아니다. 인스턴스를 다른 변수에 대입하고 나서 그 변수를 이용하여 원하는 작업을 해도 된다.

```
var userForm = _root.form.userProfile;
userForm.useriD = "U346BX";
userForm.gotoAndPlay("questionnaire");
```

대다수의 개발자들은 변수를 이용한 방법이 더 가독성도 좋고 작업하기도 편하다고 생각한다. 하지만 위 두 가지 방법은 모두 유효하다. 무비 클립을 객체로 다루는 것에 대해서는 '13장. 무비 클립'에서 자세히 배우도록 하자.

ifFrameLoaded 선언문

플래시 5의 ifFrameLoaded 선언문은 이전 버전의 If Frame Is Loaded 액션을 대체하는 선언문이다. if 선언문과 마찬가지로 ifFrameLoaded에는 특정한 상황에서만 실행되는 하위 선언문이 들어 있다. 이 선언문의 사용법은 다음과 같다.

```
ifFrameLoaded (expression) {
  statements
}
```

expression 자리에는 프레임 번호 또는 프레임 레이블을 나타내는 문자열이 들어간다. expression이 가리키는 프레임을 재생기에서 다운로드하고 나면 statements가 실행된다. 그렇지 않으면 선언문 블록을 건너뛴다.

ifFrameLoaded 선언문에서는 else 절을 사용할 수 없기 때문에, 프리로딩 스크립트로 쓰기에는 적절하지 않다. 따라서 플래시 5에서는 쓰이지 않으며 플래시 3 이전 버전에서 사용할 수 있는 무비를 만들 때만 사용해야 한다. 플래시 4 이후 버전에서는 if-else 선언문에서 _totalframes와 _framesloaded 속성을 이용하여 더 다양한 프리로더를 만들 수 있다. 예를 들면 다음과 같다.

```
if (_totalframes > 0 && _framesloaded == _totalframes) {
  gotoAndPlay("beginMovie");
} else {
  gotoAndPlay(_currentframe - 1);
}
```

비어있는 선언문

선언문의 종류를 모두 설명하자면 다음과 같이 아무런 내용도 없이 세미콜론만 덩그러니 있는 선언문도 문법적으로 아무 문제가 없다는 점을 언급하고 넘어가야 한다.

```
;
```

'비어있는 선언문(empty statement)' 은 선언문이 들어갈 자리를 미리 표시해 두는 경우 외에는 실질적으로 거의 사용할 필요가 없다. 코드에 빈 줄을 넣을 때도 비어있는 선언문을 사용하지 않아도 된다. 액션스크립트에서는 비어있는 줄을 그냥 건너뛰므로 적절히 아무 내용도 없는 빈 줄을 넣어서 코드를 읽기 쉽도록 만들 수 있다.

일반 모드에서는 코드 블록에 evaluate 액션을 추가하면 세미콜론만 있는 비어있는 선언문이 만들어진다. 그런 경우에는 Parameters 패널의 Expression 필드에 원하는 내용을 입력하여 어떤 선언문이든지 그 자리에 추가할 수 있다.

선언문과 액션

플래시 액션스크립트 편집 환경에서 '선언문(statement)' 이라는 단어는 나오지 않는다. 매크로미디어 플래시 5 액션스크립트 레퍼런스 가이드에서도 '액션(Action)' 이라는 단어와 '선언문' 이라는 단어를 혼용하고 있다.

'액션' 이라는 용어를 '선언문' 이라는 용어와 섞어서 사용하면 몇 가지 다른 액션스크립트 도구와의 차이점이 불분명해진다. 이러한 사실을 확인해보기 위해 액션스크립트 편집기를 열어서 [그림 1-2]에 나온 것처럼 Actions 폴더를 열어보자. 그

폴더에 있는 액션 목록에는 〔표 6-1〕에 나와 있는 선언문이 들어있다. 그 선언문들 사이에는 gotoAndPlay( ), getURL( ), startDrag( )와 같은 함수들도 적지 않게 들어 있다. Actions에 열거된 함수들을 선언문에서 사용할 수도 있지만 엄밀하게 보면 그 함수들이 선언문에 해당하는 것은 아니며, 단지 내장 함수일 뿐이다. 매크로미디어에서는 선언문, 몇 가지 내장 함수, 그리고 이벤트 핸들러를 모두 액션이라고 부른다. 이 책에서는 액션이라는 모호한 용어는 사용하지 않는다. 대신 각 액션을 형식적인 역할에 따라 선언문, 함수, 이벤트 핸들러라는 용어로 부르도록 하자.

앞으로 배울 내용

선언문에 대해 꽤 많은 내용을 배우긴 했지만 두 가지 중요한 선언문 유형(조건문과 순환문)은 아직 자세히 배우지 않았다. 앞에 나오는 두 장에서는 조건문과 순환문에 대해 자세히 살펴보자.