1부

액션스크립트 기초

1부에서는 변수, 데이터, 선언문, 함수, 이벤트 핸들러, 배열, 객체, 무비 클립과 같은 액션스크립트의 핵심적인 구문 및 문법에 대해 배운다. 1부만 읽어도 액션스크립트 프로그램을 만드는 데 필요한 모든 것을 알 수 있다.

- 1장. 프로그래밍을 모르는 독자를 위한 기본 소개
- 2장. 변수
- 3장. 데이터와 데이터형
- 4장. 원시 데이터형
- 5장. 연산자
- 6장. 선언문
- 7장. 조건문
- 8장. 순환문
- 9장. 함수
- 10장. 이벤트 및 이벤트 핸들러
- 11장. 배열
- 12장. 객체와 클래스
- 13장. 무비 클립
- 14장. 렉시컬 구조
- 15장. 고급 주제

1

프로그래밍을 모르는 독자를 위한 기본 소개

이 책은 플래시에 관한 책이다.

단지 플래시에서 프로그래밍 작업을 하는 것이 아니라 플래시에 어떤 작업을 하도록 지시하고 어떤 결과가 나오는지 지켜보는 것이다. 이것은 은유나 수사학적인 표현이 아니다. 바로 프로그래밍에 대한 철학적인 접근이다.

프로그래밍 언어는 컴퓨터에 정보를 보내고 컴퓨터로부터 그 정보를 다시 받아들이는 데 사용한다. 사람들이 쓰는 언어와 마찬가지로 프로그래밍 언어도 어휘와 문법으로 이루어진다. 프로그래밍 언어를 이용하면 컴퓨터에 어떤 일을 하도록 지시할 수도 있고 정보를 요청할 수도 있다. 프로그래밍 언어에서는 사용자의 명령을 기다리고 주어진 작업을 처리하며 그에 따르는 응답을 한다. 따라서 프로그래밍을 배우기 위해 이 책을 읽는다면 플래시와 의사소통하는 법을 배우기 위해 이 책을 읽는다고 할 수도 있다. 그렇다고 해서 플래시에서 영어나 불어, 독일어, 중국어 같은 언어를 사용하는 것은 아니다. 플래시의 모국어는 액션스크립트이며, 이제 우리가 배우고자 하는 언어가 바로 액션스크립트이다.

컴퓨터 언어를 배운다는 것은 종종 프로그래밍을 배우는 것과 같은 의미로 쓰인다. 하지만 프로그래밍은 단순히 언어의 구조를 배우는 것 이상의 작업이다. 플래시에서 우리말을 사용한다면 어떨까? 만약 플래시와 의사소통하기 위해 액션스크립트를 따로 배우지 않아도 된다면 말이다.

만약 "플래시, 화면에 공이 튀어 다니게 해봐" 라고 말한다면 어떻게 될까?

플래시에서는 우리의 요청을 제대로 처리하지 못할 것이다. '공' 이라는 단어를 이해하지 못하기 때문이다. 이러한 문제는 사실 말뜻을 이해하는 문제에 지나지 않는다. 플래시는 사람들이 플래시가 알고 있는 객체(무비 클립, 버튼, 프레임과 같은 것)를 이용하여 설명하기를 원한다. 그렇다면 이제 플래시가 알아들을 수 있는 말로 고쳐서 플래시에 명령을 내려보자. "플래시, ball_one이라는 무비 클립을 화면에서 돌아다니게 만들어 봐."

하지만 이렇게 한다고 해서 플래시에서 우리의 요청을 처리해줄 리가 없다. 공의 크기가 어느 정도 되어야 할까? 또 공의 위치는 어디쯤이 좋을까? 처음에 어느 방향으로 움직여야 할까? 속도는? 화면의 어느 부분에서 공이 반사되어야 할까? 그리고 얼마나 오랫동안 움직여야 하나? 2차원으로 움직일까, 3차원으로 움직일까? 흠... 이러한 질문은 우리가 예상하고 있던 것은 아니다. 물론 플래시에서 진짜 이러한 질문을 하지는 않는다. 우리의 요청을 이해하지 못하는 경우에는 그 요청을 처리하지 않거나 오류 메시지를 내보낼 뿐이다. 일단 플래시에서 조금 더 직설적으로 지시해 달라는 요청을 했다고 가정하고 우리의 요청을 몇 가지 단계로 재구성해 보자.

1. 공은 ball이라는 이름을 가진 원 모양의 무비 클립이다.
2. 직사각형은 square라는 이름을 가진 사각형 모양의 무비 클립이다.
3. 지름이 50픽셀인 녹색 공을 새로 만든다.
4. 새로운 공을 ball_one이라고 부른다.
5. 너비가 300픽셀인 새로운 검은 직사각형을 만들어 스테이지 중앙에 놓는다.
6. ball_one은 직사각형 맨 윗부분에 놓는다.
7. ball_one을 초당 75픽셀의 속도로 임의의 방향으로 움직인다.
8. ball_one이 직사각형의 변에 부딪히면 반사시킨다.
9. 정지 명령을 내릴 때까지 계속 움직인다.

모두 우리말로 설명하긴 했지만 플래시에 우리의 뜻을 이해시키려면 화면 위에서 튀어 다니는 공을 논리적으로 설명해야 한다. 물론 프로그래밍이 단순히 프로그래밍 언어의 구조에 불과한 것은 아니다. 우리말과 마찬가지로 단어를 많이 안다고 해서 말을 잘 할 수 있는 것은 아니다.

지금까지 가상적으로 우리말을 알아듣는 플래시 예제를 생각해 보았는데, 여기서 반드시 알아야 할 프로그래밍의 중요한 네 가지 요소를 짚고 넘어가자.

- 어떤 언어를 사용하든지 프로그래밍의 진수는 논리적인 단계를 체계적으로 서술하는 데 있다.
- 컴퓨터 언어로 무엇인가를 말하기 전에 우선 우리말로 어떤 명령을 내릴지 생각해보는 것이 좋다.
- 한 언어에서 쓰이는 문장을 다른 언어로 옮기더라도 결국 기본적인 구문은 그대로 남게 된다.
- 컴퓨터는 가정하는 데 그다지 능숙하지 않으며, 어휘도 상당히 제한되어 있다.

대부분의 프로그래밍 과정은 코드를 작성하는 것과는 전혀 상관이 없다. 단 한 줄의 액션스크립트 코드를 짜더라도 만들려고 하는 것이 정확히 무엇인지 생각해보고 시스템의 기능을 순서도나 청사진으로 미리 만들어야 한다. 개념적인 수준에서 프로그램이 완벽하다고 생각되면 그때 액션스크립트로 그 생각을 옮기면 된다.

사랑이나 정치, 비즈니스와 마찬가지로 프로그래밍에서도 효율적인 의사소통이 성공의 열쇠가 된다. 플래시에서 독자가 만든 액션스크립트를 이해하려면 따옴표, 등호, 세미콜론과 같은 사소한 부분까지 문법을 정확하게 맞추어야 한다. 또한 프로그래머의 의도를 정확하게 전달하려면 플래시에서 알 수 있는 표현만을 이용하여 플래시 세계에 있는 것만을 이야기해야 한다. 컴퓨터 프로그래밍을 어린 아이에게 말하는 것과 같다고 생각해보자. 어떤 것도 당연하게 여기지 말고 자세한 내용까지 직접 말해야 하고 하나의 작업을 완료하기 위해 필요한 모든 단계를 가르쳐줘야 한다. 하지만 플래시는 어린 아이와는 달리 시키는 일을 정확하게 처리하고 시키지 않은 일은 전혀 하지 않는다는 것을 기억해 두자.

몇 가지 기본 문장

어떤 언어를 배우든지 첫날에는 아주 간단한 문장을 배운다("안녕하세요" 같은 것). 각 단어의 의미를 정확하게 알지 못한 채로 문장을 통째로 외운다고 해도 그 문장의 의미와 그러한 뜻을 전달할 때 같은 문장을 반복하면 된다는 것은 알 수 있다. 일단 문법 규칙을 배우고 어휘를 늘리고 외우고 있는 문장을 응용하여 다양한 단어를 사용하다 보면, 처음에 배운 문장을 더 깊이 이해할 수 있다. 이 장에서는 이제 외국어 학교에서 첫날 하는 것과 마찬가지로 코드의 일부분을 살펴보고 몇 가지 기본 문법을 배우게 될 것이다. 그리고 나서 이 책의 나머지 부분에서는 기본 문법을 바탕으로 더 복잡한 것을 배우게 된다. 이 책을 모두 읽고 나서 1장을 다시 읽어 보면 얼마나 많은 것을 배웠는지 새삼 느낄 수 있을 것이다.

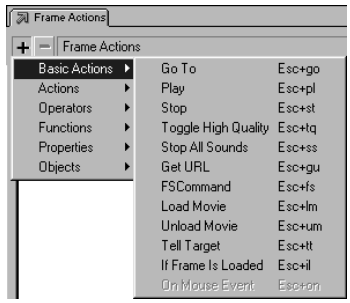
코드 작성

첫째 예제로 플래시 무비에 네 줄 짜리 간단한 코드를 추가하는 법을 배워보자. 거의 모든 액션스크립트 프로그래밍은 액션 패널(Actions panel)에서 이루어진다. 플래시 무비를 재생할 때 플래시에서는 액션 패널에 있는 모든 지시 사항을 수행한다. 다음과 같이 액션 패널을 열어보자.

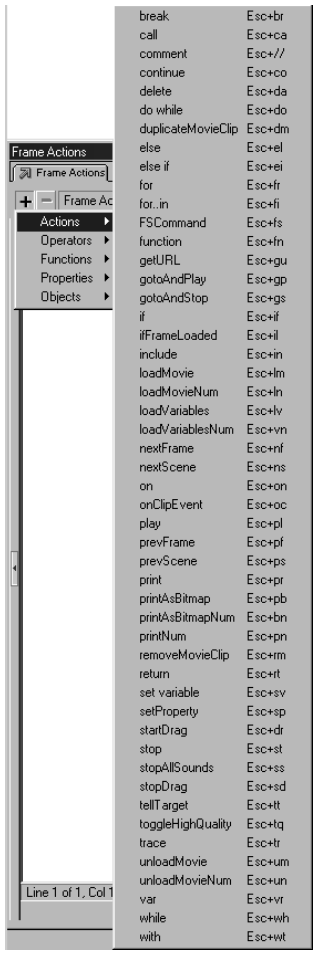
1. 플래시를 시작하고 새 문서를 만든다.
2. 메인 타임라인에서 레이어 1의 프레임 1을 선택한다.
3. Window → Actions를 선택한다.

액션 패널은 스크립트 틀(Script pane, 오른쪽 부분)과 도구상자 틀(Toolbox pane, 왼쪽 부분)로 나뉜다. 스크립트 틀에는 모든 코드가 들어간다. 도구상자 틀을 이용하면 액션(Action), 연산자(Operator), 함수(Function), 속성(Property), 그리고 액션스크립트 객체(Object)를 쉽게 이용할 수 있다. 이전 버전에서 이미 〔그림 1-1〕에 나온 것과 같은 Basic Actions 메뉴를 많이 보았을 것이다.

하지만 도구상자 틀에는 더 많은 기능이 들어 있다. 〔그림 1-2〕에 플래시 2, 3이나 4에서부터 사용하던 것이 포함된 모든 액션이 나와 있다. 도구상자 틀을 살펴보면 사운드(Sound), 배열(Array), XML 같은 것도 들어 있다. 이 책을 끝마치고 나면 이러한 내용을 모두 익힐 수 있을 것이다.



〔그림 1-1〕 플래시 5의 Basic Actions



〔그림 1-2〕 Actions 메뉴를 확장한 모습

도구상자 틀의 메뉴를 이용하여 액션스크립트 코드를 만들 수도 있다. 하지만 구문이나 원리, 액션스크립트의 전반적인 구조를 배우고자 한다면 코드를 직접 입력하는 것이 좋다.

액션이라는 것은 단순히 어떤 동작만을 나타내는 것은 아니다. 액션에는 변수, 조건문, 순환문, 주석, 함수 호출과 같은 기본적인 프로그래밍 언어와 관련된 다양한 도구도 포함된다. 이러한 것을 모두 하나의 메뉴에 합쳐 놓았지만 액션이라는 일반적인 용어 때문에 프로그래밍 구조의 의미가 약간 불명확하게 느껴진다.

이 책에서는 액션을 프로그래머 입장에서 여러 분야로 나눌 것이다. 상황에 따라 액션을 적절한 프로그래밍 용어로 설명한다. 예를 들면 'while 액션을 추가한다' 라는 표현 대신 'while 순환문을 만든다' 라고 하거나, 'if 액션을 추가한다' 대신 '새로운 조건문을 만든다' 라고 하거나, 또는 'play 액션을 추가한다' 대신 'play() 함수(또는 메소드)를 호출한다' 라는 표현을 사용할 것이다. 이처럼 각 요소를 정확하게 분류하는 것은 액션스크립트를 배우는 데 매우 중요한 부분이다.

이제 준비가 되었다면 프로그래밍을 시작해보자.

플래시에 인사하기

액션 틀에 코드를 입력하기 전에 다음과 같이 액션스크립트의 자동 입력 기능을 해제해야 한다.

1. Edit → Preferences 선택
2. General 탭에서 Actions Panel 선택 → Mode → Expert Mode
3. 액션 패널의 맨 오른쪽에 있는 화살표를 누르면 나타나는 팝업 메뉴에서 Expert Mode를 선택해도 되지만 이렇게 하면 현재 작업 중인 프레임의 모드만 변경된다('16장. 액션스크립트 저작 환경' 참조).

전문가 모드(Expert Mode)로 바꿔놓고 나면 이미 전문가가 된 기분이 들 것이다. 전문가 모드로 들어가면 액션 패널 아래 부분에 있는 인자 틀(Parameter pane)이 없어진다. 하지만 메뉴를 이용하여 프로그래밍을 하지 않는다면 걱정하지 않아도 된다.

이제 레이어 1의 프레임 1을 선택하자. 액션스크립트(즉 코드)는 반드시 프레임이나 무비 클립, 또는 버튼에 연결되어 있어야 한다. 프레임 1을 선택하면 새로 만드는 코드가 그 프레임에 연결된다. 전문가 모드에서는 액션 패널 오른쪽에 있는 스크립트 틀에 직접 입력하면 된다. 앞으로 계속 이러한 방식으로 프로그램을 입력할 것이다.

이제 첫째 프로그램을 만들어보자. 플래시에 간단한 인사말을 전달하는 프로그램이다. 스크립트 틀에 다음과 같은 코드를 입력하자.

```
var message = "Hi there, Flash!";
```

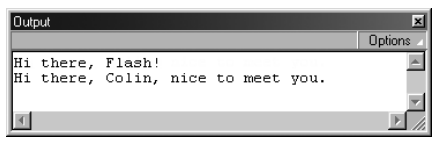
위와 같은 한 줄의 코드만으로도 완벽한 하나의 명령, 즉 선언문이 성립된다. 그 밑에 이 문단 아래에 나와있는 코드를 추가해 보자. your name here 부분에는 독자의 이름을 적당히 집어넣으면 된다(코드 중에서 이탤릭체로 표시된 부분은 독자가 내용을 입력하는 부분이다).

```
var firstName = "your name here";
trace (message);
```

하지만 이렇게 한다고 해도 아직 특별한 결과가 나타나지는 않는다. 이 코드를 .swf 파일로 저장하고 무비를 재생하기 전에는 이 코드가 실행되지 않기 때문이다. 그 전에 플래시에서 우리의 인사에 적당한 응답을 할 수 있도록 해보자. 코드의 마지막 줄에 다음과 같이 한 줄을 추가하면 된다.

```
trace ("Hi there, " + firstName + ", nice to meet you.");
```

이제 우리의 첫 프로그램을 실행해 볼 때가 되었다. Control → Test Movie를 선택한 다음, 어떤 결과가 나오는지 지켜보자. [그림 1-3]과 같이 텍스트가 화면에 나타날 것이다.



[그림 1-3] 첫 프로그램을 실행한 화면

신기하지 않은가? 이제 위 프로그램이 어떻게 동작하는지 알아보자.

변수

프로그래밍이라는 것은 컴퓨터와 의사소통하는 것에 불과하다고 말했던 것을 아마 기억하고 있을 것이다. 물론 그렇긴 하지만 컴퓨터는 보통 사람과는 약간 성격이 다르다. 첫 줄에 있는 코드에서 실제로 플래시에게 인사를 하는 것은 아니다.

```
var message = "Hi there, Flash!";
```

위 선언문을 설명하자면 다음과 같다.

> 플래시, "Hi there, Flash!"라는 정보를 기억해 줘. 나중에 써야 될지도 모르니까 message라는 이름을 붙이기로 하지. 나중에 message라는 것을 찾으면 "Hi there, Flash!"라는 문장을 돌려주면 돼.

그냥 '안녕' 하고 인사하는 것처럼 쉽지는 않지만, 이러한 선언문이 바로 프로그래밍의 기반이 된다. 플래시에서는 나중에 필요할 때 부르기 위한 이름만 붙여준다면 그 내용을 기억할 수 있다. 예를 들어 둘째 줄에서는 플래시에서 독자의 이름을 기억하도록 하고 그 내용을 firstName이라고 부르도록 만들었다. 플래시에서는 이렇게 미리 독자의 이름을 기억해 두고 있다가 무비를 테스트할 때 출력 윈도우에 내보낸다.

독자에게 필요한 변수를 저장할 수 있다는 것은 프로그래밍에서 매우 중요하다. 플래시에서는 텍스트(이름 같은 것), 숫자(3.14159 같은 것) 및 앞으로 배울 여러 가지 복잡한 데이터형도 저장할 수 있다.

변수 선언

플래시에서 대상을 기억하는 법에 대한 몇 가지 형식적인 용어를 배워보자. 플래시에서 정보를 기억할 수 있다는 것은 이미 배웠다. 각각의 정보를 데이터라고 부른다. 데이터(예를 들면 "Hi there, Flash!"와 같은 것)와 그 데이터를 가리키는 레이블(예를 들면 message 같은 것)을 합쳐서 '변수(variable)'라고 부른다. 변수의 레이블은 '이름(name)'이라고 부르고 변수의 데이터를 '값(value)'이라고 부른다. 또한 '변수에 값을 저장한다', 또는 '변수에 어떤 값이 들어 있다'라는 표현을 사용한다.

"Hi there, Flash!"는 그 값이 문자열 또는 텍스트라는 것을 나타내기 위해 큰따옴표로 묶여 있다는 점에 유의하자.

우리가 처음 작성한 코드에서 message라는 변수 값을 지정하였다. 변수 값을 지정하는 것을 '변수 값 대입', 또는 '대입'이라고 부른다. 하지만 변수에 값을 대입하기 전에 우선 변수를 만들어야 한다. 변수를 만들 때는 앞에서 본 것처럼 var라는 특별한 키워드를 이용하여 변수를 선언하면 된다.

한 예로, 앞에서 만든 코드를 방금 설명한 용어들을 이용하여 설명해 보자. message라는 이름의 새 변수를 선언하고 그 변수에 "Hi there, Flash!"라는 초기 값을 대입하면 다음과 같이 쓸 수 있다.

```
var message = "Hi there, Flash!";
```

무대 뒤의 마법사(인터프리터)

처음에 만들었던 코드를 다시 떠올려보자.

```
var message = "Hi there, Flash!";
var firstName = "your name here";
```

위에 있는 두 개의 선언문에서 각각 하나의 변수를 만들고 그 변수에 값을 대입하였다. 하지만 셋째와 넷째 줄은 약간 다르다.

```
trace (message);
trace ("Hi there, " + firstName + ", nice to meet you.");
```

이 선언문에서는 trace() 명령어를 사용한다. 이 명령어는 이미 실행해 봐서 알겠지만 플래시에서 Output 창에 텍스트를 출력하도록 하는 역할을 한다. 셋째 줄에서는 message라는 변수 값을 화면에 표시한다. 마지막 줄에서는 firstName이라는 변수를 그 변수에 입력된 값으로 바꾸고 "Hi there"라는 문장 바로 뒤에 그 문장을 집어넣는다. 그리고 나서 trace() 명령어에서는 주어진 데이터를 Output 창에 출력하는 작업을 처리한다(Output 창은 프로그램 실행 도중에 어떤 일이 일어나는지 알아보는 데 사용하면 편리하다).

그런데 trace() 명령에서는 어떻게 Output 창에 텍스트를 출력시킬까? 어떤 명령을 내리는 것은 곧 액션스크립트 인터프리터에 명령을 내리는 것이며, 액션스크립트 인터프리터는 프로그램을 실행시키고 코드를 관리하고 명령이 들어오는지 감시하며, 액션스크립트 명령을 수행하고 선언문을 실행하고 데이터를 저장하며, 사용자에게 정보를 전달하고 값을 계산하고 플래시 플레이어에서 무비를 로딩할 때 기본적인 프로그래밍 환경을 구축하는 작업도 처리한다.

인터프리터는 액션스크립트를 컴퓨터에서 이해할 수 있는 언어로 번역하고 코드에 주어진 대로 작업을 수행한다. 무비를 재생하는 도중에 인터프리터는 항상 활성화된 상태로 사용자의 명령을 받아들인다. 인터프리터에서 사용자의 명령을 이해할 수 있다면 그 명령은 컴퓨터의 CPU로 전달되어 실행된다. 명령을 통해 어떤 결과가 만들어지면 인터프리터는 그 결과를 사용자에게 전달한다. 만약 인터프리터에서 명령을 이해할 수 없는 경우에는 오류 메시지를 내보낸다. 따라서 인터프리터는 사용자의 명령을 기다리는 역할도 하고 플래시에서 나오는 정보를 사용자에게 다시 전달하는 대표자 역할도 한다.

인터프리터의 작동 방법을 자세히 알아보기 위해 간단한 trace() 명령을 처리하는 방법을 살펴보자.

다음과 같은 명령을 인터프리터에서 어떻게 다룰지 생각해보자.

```
trace ("Nice night to learn ActionScript.");
```

인터프리터에서는 미리 정해진 명령어 이름 목록에서 trace라는 키워드를 바로 감지한다. 또한 trace()가 Output 창에 텍스트를 표시하는 데 쓰인다는 것을 이미 알고 있기 때문에, 어떤 텍스트를 화면에 표시할지 알아내려고 한다. 그러므로 trace라는 단어 뒤에 오는 괄호 사이에 들어 있는 "Nice night to learn ActionScript"라는 문장을 찾아내고 "아, 나한테 필요한 문장이 바로 이거구나. Output 창으로 이 문장을 보내야겠다"라고 생각한다.

명령어는 세미콜론(;)으로 끝난다는 점을 기억해 두자. 세미콜론은 문장 맨 뒤에 있는 마침표에 해당한다. 몇 가지 예외를 제외하면 모든 액션스크립트 선언문은 세미콜론으로 끝난다. 선언문을 제대로 이해하고 필요한 정보를 갖추고 나면 인터프리터에서는 명령을 번역하고 CPU에 전달하여 Output 화면에 텍스트가 나타나도록 처리한다.

설명하다 보니 인터프리터와 CPU에서 일어나는 일을 너무 심하게 단순화하긴 했지만, 다음과 같은 요점은 확인할 수 있다.

- 인터프리터에서는 언제나 사용자의 명령을 기다린다.
- 인터프리터에서는 사용자의 코드를 한 글자씩 읽어들이고 그 명령을 이해하려고 한다. 사람이 책을 읽을 때 책에 있는 문장을 이해하려고 하는 과정과 똑같다고 보면 된다.
- 인터프리터에서는 사용자가 만든 액션스크립트를 읽어들일 때 엄격한 규칙을 적용한다. 예를 들어 trace() 선언문에서 괄호가 빠져 있다면 인터프리터에서는 그 문장을 이해할 수 없기 때문에 명령을 제대로 수행할 수 없다.

인터프리터에 대해 전혀 몰랐던 독자도 있겠지만 머지 않아 애인과 함께 있는 것만큼 인터프리터가 편하게 느껴질 것이다. 싸울 일도 많고 소리지를 일도 많겠지만("왜 내 얘기를 제대로 듣지 않는 거지?")... 서로를 완벽하게 이해할 수 있다면 행복한 일도 많을 것이다. 조금 이상하게 들릴지 모르지만 필자의 아버지는 외국어를 배우려면 그 언어를 사용하는 외국인을 사귀는 게 제일 좋다고 말씀하곤 하셨다. 그러한 점에서 필자는 여러분이 액션스크립트 인터프리터와 좋은 관계를 유지하길 바란다. 이제부터는 액션스크립트 명령이 수행되는 과정을 설명할 때 플래시 대신 인터프리터라는 용어를 자주 사용할 것이다.

추가 정보(인자)

이미 앞에서 trace() 명령어를 사용할 때 인터프리터에 화면에 표시할 텍스트를 전달하는 경우를 보았다. 이러한 방법은 일반적으로 많이 쓰이는 방법이다. 앞으로 인터프리터에 명령을 내릴 때 그 명령을 실행하기 위해 필요한 보조 데이터를 전달하는 일이 많을 것이다. 명령어와 함께 보내는 데이터는 '인자(argument, parameter)'라고 부른다. 명령어에 인자를 제공하려면 그 인자를 괄호 안에 집어넣으면 된다.

```
command(argument);
```

하나의 명령어에 여러 인자를 전달할 때는 쉼표(,)를 이용하여 분리한다.

```
command(argument1, argument2, argument3);
```

명령어에 인자를 제공하는 것을 보통 인자를 전달한다고 한다. 예를 들어 gotoAndPlay(5)라는 코드에서 gotoAndPlay는 명령어 이름이고 5는 그 명령으로 전달되는 인자이다(이 경우에는 프레임 번호를 가리킨다). stop()과 같은 몇몇 명령어에는 괄호가 필요하지만 인자를 받아들이지는 않는다. '9장. 함수'에서 그 이유를 알아보기로 하자.

액션스크립트용 풀(연산자)

다음과 같은 trace() 선언문이 있는 처음에 다루었던 코드의 넷째 줄을 다시 살펴보자.

```
trace ("Hi there, " + firstName + ", nice to meet you.");
```

여기서 덧셈 기호(+)를 사용한다는 점에 유의하자. 이 기호는 텍스트를 합치는 데 쓰이는 연산자로 액션스크립트에서 많이 쓰이는 연산자 중 하나이다. 프로그래밍 언어에서 쓰이는 연산자는 사람들이 사용하는 언어에서 접속사(그리고, 또는, 하지만 등)와 비슷하다. trace() 예제에서는 덧셈 기호를 이용하여 "Hi there"와 firstName에 들어 있는 텍스트를 합친다.

모든 연산자는 코드 구문을 합치고 그 과정에서 구문을 조작하는 역할을 한다. 그 구문이 텍스트이거나 숫자, 또는 다른 데이터형인 경우에도 연산자는 일종의 변환 작업을 수행한다. 덧셈 연산자와 마찬가지로 많은 연산자에서 두 가지 대상을 합치는 작업을 처리한다. 하지만 값을 비교하거나 대입하고 논리적인 결정을 내리고 데이터형을 알아내거나 새로운 객체를 만드는 것과 같은 다양한 작업을 처리하는 연산자도 많이 있다.

두 개의 숫자 피연산자와 함께 사용하면 덧셈 기호(+)와 뺄셈 기호(-)를 이용하여 간단한 산수 계산을 처리할 수 있다. 다음과 같은 명령을 내리면 "3"이 Output 창에 출력된다.

```
trace(5 - 2);
```

< 연산자는 두 개의 수 중에서 어느 쪽이 더 큰지, 또는 두 분자 중에서 어느 쪽이 알파벳순으로 더 앞에 있는지 알아내는 데 쓰인다.

```
if (3 < 300) {
  // 어떤 작업을 처리한다.
}

if ("a" < "z") {
  // 다른 작업을 처리한다.
}
```

조합, 비교, 대입과 같이 연산자를 통해 이루어지는 조작을 '연산'이라고 부른다. 수식 연산은 덧셈(+), 뺄셈(-), 곱셈(*), 나눗셈(/)과 같이 산수 시간에 배우는 규칙을 그대로 따르기 때문에 제일 이해하기 쉽다. 하지만 전문적인 프로그래밍 작업에만 쓰이는 연산자는 조금 생소할 수도 있다. 예를 들어 typeof 연산자를 생각해 보자. 이 연산자는 변수에 어떤 종류의 데이터가 들어있는지 알려주는 역할을 한다. 만약 x라는 변수를 만들어서 4를 대입한 후에 x의 데이터형을 알고 싶다면 다음과 같이 하면 된다.

```
var x = 4;
trace (typeof x);
```

플래시에서 이 코드를 실행하면 Output 창에 "number"라는 단어가 출력된다. typeof 연산자를 사용할 때에도 피연산자는 필요하지만 그 변수를 괄호 안에 집어넣지는 않고 그냥 typeof x 라고 하는 것을 기억해 두자. 이렇게 되면 x가 typeof의 인자인지 아닌지 궁금할 것이다. 사실 x는 인자(코드 구문을 계산하기 위해 필요한 보조 데이터)와 똑같은 역할을 하긴 하지만 연산자의 개념에서 본다면 x는 공식적으로 피연산자라고 부르는 것이 옳다. 피연산자는 연산자가 연산을 수행할 대상이 되는 아이템을 가리키는 말이다. 예를 들어 4 + 9라는 수식에서 4와 9라는 숫자는 + 연산자의 피연산자가 된다.

액션스크립트에서 쓰이는 연산자에 관한 것은 '5장. 연산자'에서 자세히 다룰 것이다. 일단 연산자가 몇 가지 변환 과정에서 코드 구문을 이어주는 역할을 한다는 것을 기억해 두자.

모두 합치기

지금까지 배운 것을 다시 훑어보자. 다시 첫 줄을 살펴보자.

```
var message = "Hi there, Flash!";
```

여기에서 var라는 키워드는 새로운 변수를 선언한다는 것을 인터프리터에 알려준다. message라는 단어는 변수의 이름이다. 등호 연산자에서는 텍스트 문자열("Hi there, Flash!")을 message라는 변수에 대입한다. 따라서 "Hi there, Flash!"라는 텍스트가 message 값이 된다. 마지막으로 세미콜론(;)은 첫째 선언문이 끝났다는 것을 인터프리터에 알려주는 역할을 한다.

둘째 줄은 첫째 줄과 거의 비슷하다.

```
var firstName = "your name here";
```

여기서 your name here 자리에 입력한 텍스트를 firstName이라는 변수에 대입한다. 세미콜론으로 둘째 선언문이 끝났다는 것을 표시한다.

그리고 나서 셋째, 넷째 줄에서는 message와 firstName 변수를 사용한다.

```
trace (message);
trace ("Hi there, " + firstName + ", nice to meet you.");
```

trace라는 키워드는 인터프리터에 Output 창으로 텍스트를 표시하고 싶다는 것을 알려주는 역할을 한다. 이 때 출력하고자 하는 텍스트를 인자로 전달한다. 왼쪽 괄호는 인자의 시작 부분을 가리킨다. 넷째 줄에서는 인자에 두 개의 연산이 포함되어 있으며 두 연산 모두 덧셈 기호를 연산자로 사용한다. 첫째 연산에서는 첫째 피연산자인 "Hi there,"와 둘째 피연산자인 firstName을 합친다. 둘째 연산에서는 첫째 연산에서 나온 결과에 ", nice to meet you."를 덧붙인다. 오른쪽 괄호로 인자의 끝 부분을 나타내며 세미콜론을 이용하여 선언문의 끝을 나타낸다.

액션스크립트 심화 개념

지금까지 데이터, 변수, 연산자, 선언문, 함수, 인자와 같은 액션스크립트를 이루는 몇 가지 기본 요소를 알아보았다. 이러한 주제를 더 깊이 다루기 전에 액션스크립트의 나머지 핵심 기능에 대해 간단히 알아보자.

플래시 프로그램

대부분의 사용자는 프로그램을 어도비 포토샵이나 매크로미디어 드림위버 같은 애플리케이션이라고 생각한다. 하지만 플래시에서 애플리케이션을 만들지는 않는다. 프로그래머들은 프로그램을 코드의 모음(일련의 선언문)이라고 정의하지만 그러한 정의는 우리가 하는 일의 일부분에 지나지 않는다.

플래시 무비는 단순한 코드의 모음이 아니다. 플래시 코드는 프레임이나 버튼과 같은 플래시 무비의 기본 요소와 연결되어 있다. 우리는 그러한 요소에 코드를 추가하여 서로 상호 작용할 수 있도록 해주는 작업을 한다.

결국 '프로그램'을 고전적인 의미로 본다면 플래시에는 프로그램이라는 것이 없다. 프로그램을 모두 액션스크립트를 이용하여 만드는 것이 아니라 스크립트를 사용할 뿐이다. 마치 HTML 문서에 자바스크립트를 이용하여 프로그램과 같은 성질을 부여하는 것처럼 무비에 액션스크립트를 이용하여 프로그램과 같은 성질을 부여한다고 보면 된다. 우리가 실제로 만드는 것은 프로그램이 아니라 (코드, 타임라인, 비주얼, 사운드 및 기타 요소가 포함된) 무비이다.

스크립트에는 C++나 자바에서 화면에 그래픽을 표시하거나 사운드 큐를 저장하기 위해 사용하는 운영체제 수준의 코드를 제외하면 전통적인 프로그램에서 사용하는 거의 모든 요소가 포함된다. 스크립트를 이용하면 그래픽 및 사운드 프로그래밍을 관리하지 않아도 되므로 무비의 속성을 설계하는 것에만 집중하면 된다.

표현식

앞에서 배운 것처럼 스크립트의 선언문에는 스크립트 명령어가 들어 있다. 하지만 대부분의 명령어는 데이터가 없으면 아무런 쓸모가 없다. 예를 들어 변수를 설정할 때 변수 값에 해당하는 어떤 데이터를 대입해야 하는 것처럼 말이다. trace() 명

령을 이용할 때는 Output 창에 출력할 데이터를 인자로 전달한다. 데이터는 우리가 액션스크립트 코드에서 조작할 내용물이다. 스크립트 전반에 걸쳐 수많은 데이터를 읽어들이고 내보내고 저장하고 여기저기로 이동하게 될 것이다.

프로그램이 실행될 때 하나의 데이터를 만들어내는 구문을 '표현식(expression)'이라고 부른다. 7이라는 숫자, 또는 "제 웹 사이트에 오신 것을 환영합니다" 같은 문자열은 모두 간단한 표현식에 해당한다. 이 두 표현식은 프로그램이 실행될 때 있는 그대로 쓰이는 간단한 데이터를 나타낸다. 따라서 이러한 표현식은 '리터럴 표현식(literal expression)' 또는 간단하게 '리터럴(literal)'이라고 부른다.

리터럴은 여러 가지 표현식 중 하나일 뿐이다. 변수도 표현식이 될 수 있다(변수는 데이터를 나타내기 때문에 표현식이라고 볼 수 있다). 표현식을 연산자로 합치면 더 특이한 모양이 나올 수도 있다. 예를 들어 4+5라는 표현식은 두 개의 피연산자(4와 5)가 있는 표현식이지만 덧셈 연산자가 있기 때문에 전체적으로 9라는 하나의 값을 갖는 표현식이 된다. 코드 전체를 하나의 값으로 변환할 수만 있다면 복잡한 표현식에 다른 짧은 표현식을 포함시킬 수도 있다.

다음과 같은 변수 message를 생각해보자.

```
var message = "Hi there, Flash!";
```

원한다면 다음과 같이 변수 표현식인 message를 "How are you?"라는 리터럴 표현식과 합칠 수도 있다.

```
message + "How are you?"
```

이렇게 하면 프로그램이 실행될 때 이 표현식 값은 "Hi there, Flash! How are you?"가 된다. 수식 계산을 하다 보면 긴 표현식에 짧은 표현식이 포함되어 있는 것을 자주 볼 수 있다.

```
(2 + 3) * (4 / 2.5) - 1
```

프로그래밍 개념을 설명할 때 표현식이라는 용어를 자주 사용하기 때문에 프로그래밍을 배우는 과정에서 표현식을 일찌감치 알아두는 것이 좋다. 예를 들어 필자가 '변수에 값을 대입하려면 변수 이름, 등호, 임의의 표현식을 차례로 입력하면 된다'라는 식으로 설명하는 경우가 적지 않게 있을 것이다.

두 가지 필수 선언문 유형 : 조건문과 순환문

'조건문(conditional)'과 '순환문(loop)'은 각각 논리적인 작업과 반복 작업을 처리하는 데 쓰이기 때문에 거의 모든 프로그램에서 사용된다.

조건문을 이용한 선택

플래시 프로그래밍의 가장 뛰어난 점은 무비를 더 똑똑하게 만들 수 있다는 것이다. 여기에서 똑똑하다는 말은 다음과 같이 설명할 수 있다. 웬디라는 여학생이 옷이 젖는 것을 좋아하지 않는다고 가정하자. 웬디가 매일 아침 집에서 나오기 전에 그녀는 창문을 열어서 날씨를 확인해 본다. 만약 비가 내리면 그녀는 우산을 가지고 나간다. 그러면 웬디는 똑똑하다고 말할 수 있다. 그녀는 기본적인 논리(일련의 선택사항을 살펴보고 주어진 상황을 기반으로 무엇을 할지 결정하는 것)를 이용할 줄 안다. 상호작용할 수 있는 플래시 무비를 만들 때도 똑같은 기본 논리를 이용한다.

플래시 무비에서 논리의 예는 다음과 같다.

- 무비에 세 개의 섹션이 있다고 가정하자. 사용자가 각 섹션에 들어가면 논리를 이용하여 그 섹션의 도입부를 보여줄지 결정한다. 사용자가 전에 그 섹션에 들어가 본 적이 있다면 도입부를 건너뛰고 그렇지 않다면 도입부를 보여준다.

- 무비에서 제한 구역이 있다고 가정하자. 그 제한 구역에 들어가려면 사용자가 암호를 입력해야 한다. 사용자가 정확한 암호를 입력하면 그 내용을 보여준다. 그렇지 않으면 보여주지 않는다.

- 화면 위에서 공이 움직일 때 벽에서 반사되도록 한다고 가정하자. 공이 정해진 점을 지나면 공의 방향을 반대로 돌린다. 그렇지 않으면 공이 원래 움직이던 방향으로 계속 움직이도록 한다.

무비에 논리를 적용하려면 조건문이라는 특별한 형태의 선언문을 사용해야 한다. 조건문을 이용하여 어떤 부분의 코드를 실행시키거나 실행시키지 않을 조건을 지정할 수 있다. 조건문의 예는 다음과 같다.

```
if (userName == "James Bond") {
  trace ("Welcome to my web site, 007.");
}
```

조건문의 일반적인 구조는 다음과 같다.

```
if (조건) {
  실행해야 할 코드
}
```

자세한 문법은 '7장. 조건문'에서 배울 수 있다. 일단 조건문을 이용하여 플래시에서 논리적인 결정을 내릴 수 있다는 점만 기억하고 넘어가자.

순환문을 이용한 반복 작업

무비에서 논리적인 결정을 내리는 것 외에 사람 대신 지루한 반복 작업을 처리하는 것도 필요하다. 그러니까 컴퓨터가 온 세계를 집어삼키고 사람을 노예로 만들어서 조그만 에너지 주머니에 넣고 부려먹기 전까지는... 잠깐... 방금 말한 것은 잊어주기 바란다. 흠... 특정 숫자부터 시작해서 다섯 개의 연속된 숫자를 output 창에 출력한다고 가정하자. 시작하는 숫자가 10이라면 수열을 다음과 같이 출력할 수 있다.

```
trace (10);
trace (11);
trace (12);
trace (13);
trace (14);
```

하지만 그 수열을 513부터 시작하려면 모든 숫자를 다음과 같이 고쳐야 한다.

```
trace (513);
trace (514);
trace (515);
trace (516);
trace (517);
```

trace() 선언문과 변수를 함께 사용하면 일일이 다시 입력하지 않아도 된다.

```
var x = 1;
trace (x);
x = x + 1;
trace (x);
x = x + 1;
trace (x);
x = x + 1;
```

```
trace (x);
x = x + 1;
trace (x);
```

첫 줄에서 변수 x의 값을 1로 설정하였다. 그리고 둘째 줄에서 그 값을 Output 창으로 보낸다. 셋째 줄에서는 x의 현재 값을 읽어들여 그 값에 1을 더하고 그 결과를 변수 x에 다시 대입한다. 따라서 x 값은 2가 된다. 그리고 x 값을 다시 Output 창으로 보낸다. 이 과정을 세 번 더 반복한다. 이 작업을 마치고 나면 Output 창에 다섯 개의 연속적인 숫자가 출력된다. 이렇게 하면 수열을 시작하는 숫자를 바꿀 때 x의 초기 값만 변경하면 된다는 장점이 있다. 나머지 코드는 x를 기반으로 돌아가기 때문에 프로그램이 실행될 때 모든 숫자가 자동으로 바뀐다.

이 방법은 처음 방법보다는 더 발전된 방법이며, 다섯 개의 숫자만 출력할 때는 그럭저럭 쓸만하다. 하지만 500개 정도의 숫자를 출력할 때는 그다지 쓸모가 없다. 여러 번 반복되는 작업을 수행하려면 순환문(코드 블록이 임의의 횟수만큼 반복되도록 하는 선언문)을 이용하면 된다. 순환문에는 몇 가지 종류가 있으며 각 순환문마다 문법이 조금씩 다르다. 가장 흔하게 쓰이는 것으로 while 선언문이 있다. 선언문을 반복해서 사용하는 대신 while 루프를 이용하면 다음과 같이 간단하게 작업을 처리할 수 있다.

```
var x = 1;
while (x <= 5) {
   trace (x);
   x = x + 1;
}
```

while이라는 키워드는 순환문이 시작된다는 것을 나타낸다. (x <= 5)라는 표현식은 순환문을 반복해야 할 횟수를 결정한다(x가 5 이하면 계속 코드를 실행한다). trace (x); 와 x = x + 1;은 순환문이 반복되는 동안 매번 실행된다. 앞의 두 코드를 단순하게 비교한다면 겨우 다섯 줄밖에 줄어들지 않았지만 더 많은 숫자를 출력할 경우에는 수백 줄을 줄일 수도 있다. 또한 순환문은 매우 유연하다. 순환문에서 500까지 숫자를 출력하려고 한다면 간단히 (x <= 5)를 (x <= 500)으로 고치기만 하면 된다.

```
var x = 1;
while (x <= 500) {
   trace (x);
   x = x + 1;
}
```

조건문과 마찬가지로 순환문은 프로그래밍에서 가장 많이 쓰이면서 중요한 선언문이다.

모듈화된 코드(함수)

지금까지는 제일 긴 스크립트가 네 줄짜리였다. 하지만 머지않아 네 줄이 아니라 400줄 또는 4,000줄짜리 코드를 만들게 될 것이다. 조만간 코드를 손쉽게 관리할 수 있으며 작업량을 줄이고 여러 가지 상황에 적용하기 좋은 코드를 만들 수 있는 방법을 찾아야 할 것이다. 그 때 쯤이면 '함수(function)'가 반드시 필요하다. 함수는 패키지화된 일련의 선언문이다. 보통 함수는 재사용할 수 있는 코드 블록 역할을 한다.

다섯 개의 사각형 넓이를 계산하는 코드를 만들어보자. 아래에서 볼 수 있듯이 사각형 한 개의 넓이를 계산하는 코드의 다섯 배 분량의 코드가 필요하다.

```
var height1 = 10;
var width1 = 15;
var area1 = height1 * width1;
var height2 = 11;
var width2 = 16;
var area2 = height2 * width2;
var height3 = 12;
var width3 = 17;
var area3 = height3 * width3;
var height4 = 13;
var width4 = 18;
var area4 = height4 * width4;
var height5 = 20;
var width5 = 5;
var area5 = height5 * width5;
```

넓이 계산을 계속 반복해야 하므로 하나의 함수에 모든 기능을 집어넣고 그 함수를 여러 번 실행하는 것이 좋다.

```
function area(height, width){
  return height * width;
}
area1 = area(10, 15);
area2 = area(11, 16);
area3 = area(12, 17);
area4 = area(13, 18);
area5 = area(20, 5);
```

우선 var를 이용하여 변수를 선언하듯이 function 선언문을 이용하여 넓이를 계산하는 함수를 만들었다. 그리고 변수 이름을 붙이는 것과 마찬가지로 함수에 area라는 이름을 부여한다. 괄호 사이에는 함수를 사용할 때마다 함수에서 받아들여야 하는 인자의 목록(height, width)을 집어넣는다. 그리고 나서 중괄호({ }) 안에는 함수에서 실행해야 할 선언문을 포함시킨다.

```
return height * width;
```

함수를 만든 후에는 무비 안에 있는 모든 위치에서 그 이름만을 이용하여 함수 안에 있는 코드를 실행시킬 수 있다. 앞에서 사용한 예제에서는 area( ) 함수를 다섯 번 호출하였으며, 각각 높이와 너비 값(height와 width)을 인자로 전달하였다(area(10, 15), area(11, 16) 등). 각 계산 결과는 사용자에게 리턴되며 그 값은 area1부터 area5까지 변수에 저장된다. 함수를 사용하지 않는 버전의 코드에 비하면 훨씬 적은 작업량으로 깔끔하고 멋지게 작업을 처리할 수 있다.

이 함수 예제를 제대로 이해하지 못하더라도 너무 초조해하지 말기 바란다. 함수에 대한 자세한 내용은 9장에서 배우게 될 것이다. 일단 함수가 복잡한 시스템을 만들 때 매우 큰 도움이 될 것이라는 점만 기억해 두자. 함수를 이용하면 코드와 패키지를 손쉽게 재사용할 수 있기 때문에 실질적으로 만들 수 있는 프로그램의 범위를 크게 넓힐 수 있다.

내장 함수

trace() 액션과 마찬가지로 함수에서도 인자를 받아들인다는 점에 유의하자. area(4, 5);와 같이 함수를 호출하는 것은 trace (x);와 같이 trace() 명령을 호출하는 것과 거의 똑같다. 이러한 유사성은 우연이 아니다. 앞에서 지적한 것처럼 trace() 액션을 포함한 적지 않은 액션이 사실 함수이다. 하지만 그 함수들은 (우리가 만든 area() 함수와 같은 사용자 정의 함수와 달리) 액션스크립트에 내장된 특별한 유형의 함수이다. 따라서 'gotoAndStop() 함수를 호출하라'라는 표현이 'gotoAndStop 액션을 실행하라'보다 더 합리적이고 정확하다. 내장 함수는 단지 편의상 액션스크립트에 포함되어 있는 재사용할 수 있는 코드 블록이다. 내장 함수를 이용하면 수학적인 계산에서 무비 클립 제어에 이르는 다양한 작업을 처리할 수 있다. 모든 내장 함수는 '3부. 레퍼런스'에 정리해 놓았다. 또한 액션스크립트의 기초를 배우다 보면 많은 내장 함수를 접하게 될 것이다.

무비 클립 인스턴스

지금까지 프로그래밍의 기본 사항에 대해 알아보았는데 그 과정에서 플래시의 기초를 잊어버리지 않았으면 한다. 플래시의 비주얼 프로그래밍의 핵심 중 하나로 무비 클립 인스턴스(instance)가 있다. 플래시 디자이너 또는 개발자라면 무비 클립이 그다지 낯설지 않을 것이다. 하지만 무비 클립을 프로그래밍 도구라고 여기면 안 된다.

모든 무비 클립은 플래시 무비의 라이브러리에 있는 심벌 정의를 포함하고 있다. 라이브러리에서 스테이지로 클립을 끌어서 놓는 방식으로 하나의 플래시 무비에 하나의 무비 클립 심벌에 대한 여러 개의 복사본 또는 인스턴스를 추가할 수 있다. 고급 플래시 프로그래밍에서는 무비 클립 인스턴스를 제어하는 문제가 꽤 큰 비중을 차지한다. 예를 들어 반사되는 공 프로그램은 무비 클립 인스턴스의 위치를 스테이지에서 계속해서 바꿔주는 프로그램이라고 볼 수 있다. 인스턴스의 위치, 크기, 현재 프레임, 회전과 같은 요소는 무비를 재생할 때 액션스크립트를 이용하여 변경할 수 있다.

무비 클립과 인스턴스에 대해 익숙하지 않다면 플래시 문서나 도움말 파일을 읽은 후에 이 책을 계속 보는 것이 좋다.

이벤트 기반 실행 모델

액션스크립트의 기초를 훑어볼 때 마지막으로 살펴보아야 할 주제는 무비에 있는 코드가 실행되는 시기를 결정하는 실행 모델이다. 무비에서 다양한 프레임, 버튼, 무비 클립에 코드를 추가하는 경우가 많이 있다. 하지만 그러한 코드가 언제 실행될까? 이 궁금증을 해결하기 위해 컴퓨터의 역사에 대해 잠시 알아보도록 하자.

컴퓨터가 나온 초기에는 프로그램의 명령어가 첫 줄부터 마지막 줄까지 입력된 순서대로 실행되었다. 프로그램은 어떤 작업을 수행하고 나서 멈추는 것을 의미하였다. 그러한 종류의 프로그램을 배치(batch) 프로그램이라고 부르며, 플래시같이 이벤트를 기반으로 하는 프로그래밍에서 요구되는 상호작용을 처리할 수는 없다.

이벤트 기반 프로그램은 배치 프로그램과 같이 순서대로 실행되지 않는다(이벤트 순환문에서). 연속적으로 실행되면서 어떤 일(이벤트)이 일어나는지 감시하고 그러한 이벤트에 따라 코드 세그먼트를 실행한다. 시각적인 상호작용 환경에서 사용하기 위해 만든 언어(액션스크립트나 자바스크립트)에서는 보통 사용자가 마우스를 클릭하거나 키보드의 키를 누르는 것과 같은 행동이 이벤트가 된다.

마우스 클릭과 같은 이벤트가 발생하면 인터프리터에서는 경고를 내보낸다. 그리고 나면 프로그램에서는 인터프리터에 적절한 코드 세그먼트를 실행하라고 요구함으로써 그러한 경고에 반응한다. 예를 들어 사용자가 무비에서 버튼을 클릭하면 무비의 다른 섹션을 보여주는 코드를 실행하거나(네비게이션) 데이터베이스에 변수를 제공(폼 보내기)하는 것과 같은 작업을 처리할 수 있다.

하지만 사용자가 이벤트 핸들러를 만들지 않으면 프로그램에서 이벤트에 따른 작업을 할 수 없다. 이벤트 핸들러는 일반적으로 다음과 같은 형태로 만든다.

```
만약 (이러한 이벤트가 발생하면) {
  이러한 코드를 실행한다
}
```

위와 같은 내용을 보통 다음과 같은 일반적인 형태로 쓸 수 있다.

```
on (이벤트) {
  선언문
}
```

예를 들어 플레이헤드를 200번 프레임으로 옮기는 버튼에 대한 이벤트 핸들러는 다음과 같이 만들 수 있다.

```
on (press) {
  gotoAndStop(200);
}
```

이벤트 기반 프로그램은 언제나 이벤트 순환문을 실행하면서 다음 이벤트에 반응할 태세를 갖추고 있기 때문에 살아있는 시스템과 비슷하다. 이벤트는 플래시 무비에서 매우 핵심적인 부분이다. 이벤트가 없다면 플레이헤드가 어떤 프레임에 들어갔을 때 그 프레임의 코드를 실행하는 것 외에는 스크립트로 할 수 있는 일이 전혀 없다. 이 경우에는 플레이헤드가 특정 프레임에 들어가는 일이 결국은 이벤트에 해당하는데, 이러한 작업의 경우에는 이벤트 핸들러를 따로 지정할 필요가 없다.

액션스크립트를 배우는 첫날부터 이벤트를 배우는 이유는 이벤트가 말 그대로 어떤 일이 일어나게 하는 역할을 하기 때문이다. 지금까지 스크립트를 만드는 방법, 스크립트가 실행되는 시기를 결정하는 방법(즉 이벤트)에 대해 배웠다. 이제 첫 실전 예제를 만들어 볼 준비가 된 셈이다.

퀴즈 프로그램 만들기

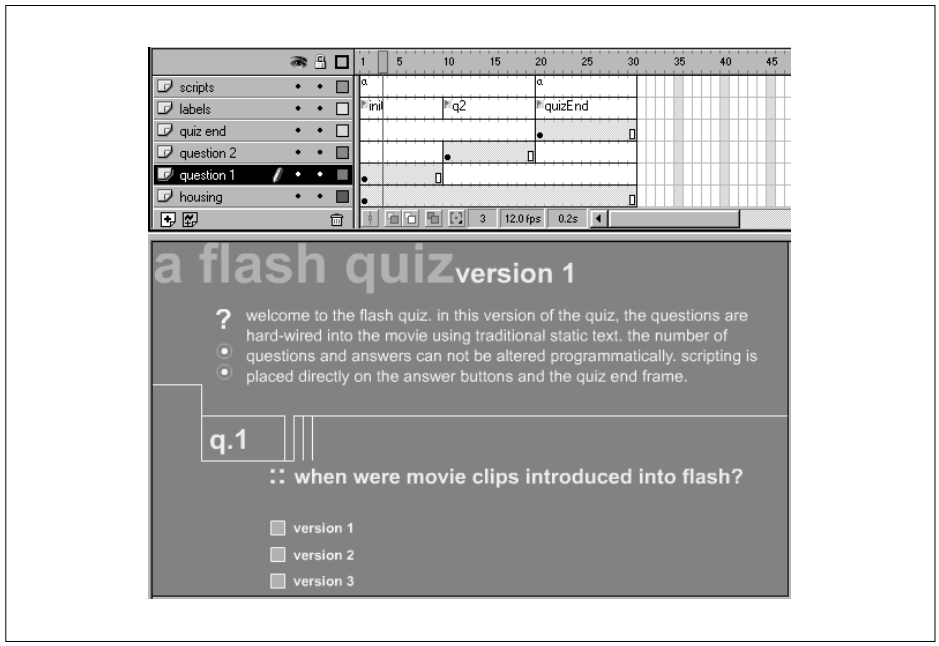
지금까지 액션스크립트의 기본 원리를 대강 알아보았으니 이제 이러한 원리를 실제 플래시 무비에 적용해 보자. 지금까지 배운 간단한 프로그래밍 기법을 이용하여 객관식 퀴즈를 만들어 보는 것으로 플래시 프로그래밍 공부를 시작해 보자. 2장 이후에서도 고급 프로그래밍 개념을 배우고 나서 이 퀴즈 프로그램을 더 향상시킬 수 있는 방법을 찾아볼 것이다. 결국에는 코드를 더 고급스럽게 만들어 확장 및 관리가 용이하며 퀴즈에 새로운 기능을 추가해서 임의의 개수 문제를 처리할 수 있도록 만들 것이다.

이 퀴즈의 .fla 파일은 온라인 코드 창고에서 구할 수 있다. 여기서는 플래시 프로그래밍에 대해 배우는 것이지 플래시를 만드는 것에 대해 배우지는 않는다는 점을 명심해 두자. 독자들이 이미 버튼을 만들고 사용하는 법, 레이어, 프레임, 키프레임 및 텍스트 도구에 익숙하다는 것을 가정하고 설명하겠다. 이 퀴즈에서는 액션스크립트 프로그래밍의 다음과 같은 요소에 대한 실전 응용법을 알아보겠다.

- 변수
- 함수를 이용한 플레이헤드 제어
- 버튼 이벤트 핸들러
- 간단한 조건문
- 정보를 화면에 보여주기 위한 텍스트 필드 변수

퀴즈 개요

우리가 만들 퀴즈의 일부가 〔그림 1-4〕에 나와 있다. 이 퀴즈에는 문제가 두 개밖에 없다. 각 문제에서는 세 개의 답 중에서 한 개의 답을 선택할 수 있다. 사용자가 답을 제출할 때는 원하는 답에 해당하는 버튼을 클릭하면 된다. 이렇게 선택한 값은 변수에 저장되어 나중에 사용자의 점수를 매기는 데 쓰인다. 모든 문제를 답하고 나면 정답의 개수를 계산하고 점수를 화면에 표시한다.



〔그림 1-4〕 플래시 퀴즈

레이어 구조 만들기

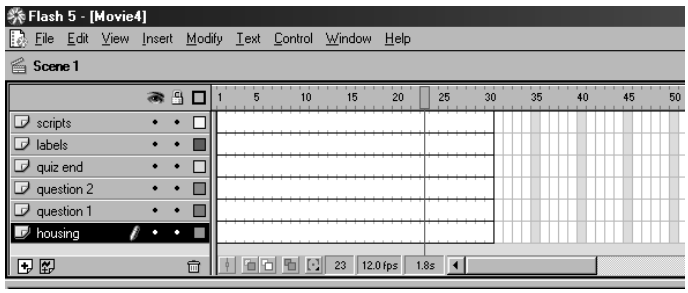
플래시 무비를 만들 때는 서로 다른 컨텐츠 요소를 별도의 레이어에 보관하여 전체 컨텐츠를 관리하기 편하도록 분리해서 정리해야 한다. 컨텐츠를 레이어로 나누는 것이 일반적으로 좋은 방법이기도 하지만 플래시 프로그래밍에서는 반드시 그렇게 해야 한다. 우리가 만드는 퀴즈를 포함하여 대부분의 스크립트를 이용해서 만든 무비에서는 모든 타임라인 스크립트를 scripts라는 독립된 레이어에 보관한다. 필자는 scripts 레이어를 레이어 스택의 맨 위에 오도록 해서 언제나 찾기 쉽도록 해놓는다.

그리고 모든 프레임 레이블을 labels라는 별도의 레이어에 관리한다. labels 레이어는 모든 타임라인에서 scripts 레이어 바로 밑에 둔다. 이 두 개의 표준 레이어(scripts와 labels) 외에도 퀴즈 무비에서는 다양한 컨텐츠를 분리하기 위해 일련의 컨텐츠 레이어를 이용한다.

퀴즈를 만들 때 가장 먼저 할 일은 새로운 레이어를 만들고 아래와 같은 이름을 붙인 다음, 여기 나온 순서대로 정렬하는 것이다.

scripts
labels
quiz end
question 2
question 1
housing

이제 각 레이어에 30개의 프레임을 추가한다. 그러면 〔그림 1-5〕 같은 모양의 타임라인이 만들어진다.



〔그림 1-5〕 퀴즈 타임라인 초기 설정

인터페이스와 문제 만들기

퀴즈를 실행시키는 스크립트를 만들기 전에 사용자가 퀴즈를 풀어나가는 데 사용할 문제와 인터페이스를 설정해야 한다.

다음 단계를 따라하면 된다.

1. housing 레이어를 선택하고 프레임 1에서 텍스트 도구를 이용하여 스테이지에 직접 타이틀을 입력한다.

2. question 1 레이어의 프레임 1에서 문제번호 '1'과 문제 1에 해당하는 텍스트를 입력한다. "When were movie clips introduced into Flash?" 이 때 문제 밑에 답과 버튼을 추가할 공간을 남겨둔다.

3. 텍스트 한 줄 높이보다 크지 않은 체크박스나 라디오 버튼 모양의 간단한 버튼을 만든다.

4. question 1 레이어에서 문제 밑에 세 개의 답을 입력한다. 'Version 1', 'Version 2', 'Version 3'을 각각 다른 줄에 입력한다.

5. 각 답 옆에 체크박스 버튼의 인스턴스를 추가한다.

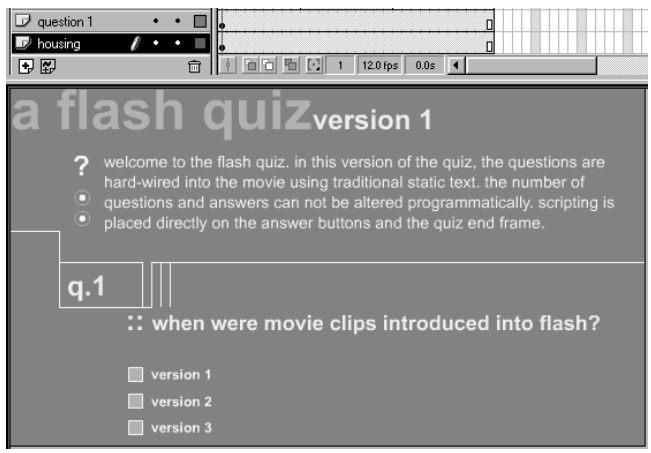
6. 문제 2를 만들 때는 문제 1을 템플릿으로 사용한다. question 1 레이어의 첫째 프레임을 선택하고 Edit → Copy Frames를 선택한다.

7. question 2 레이어의 10번 프레임을 선택하고 Edit → Paste Frames를 선택한다. 그러면 첫째 문제의 복사본이 question 2 레이어의 10번 프레임에 복사된다.

8. question 2 레이어의 10번 프레임에서 문제 번호를 1에서 2로 고치고 문제 텍스트를 "When was MP3 audio support added to Flash?"로, 답을 'Version 3', 'Version 4', 'Version 5'로 고친다.

9. 마지막으로 문제 1이 문제 2 밑에 나오는 것을 막기 위해 question 1 레이어의 10번 프레임에 비어 있는 키프레임을 추가한다.

〔그림 1-6〕에는 퀴즈에 첫째 문제를 추가했을 때의 플래시 무비가, 〔그림 1-7〕에는 퀴즈에 두 개의 문제를 입력했을 때의 타임라인이 나와 있다.



〔그림 1-6〕 퀴즈 제목과 문제 1

〔그림 1-7〕 두 개의 문제를 입력했을 때의 타임라인

퀴즈 초기화

거의 모든 스크립트에서 마찬가지지만 퀴즈 스크립트에서 가장 먼저 할 일은 무비 전반에 걸쳐 사용할 메인 타임라인 변수를 만드는 일이다. 지금 만드는 퀴즈에서는 무비의 첫째 프레임에서 이 작업을 하지만 다른 무비에서는 보통 무비의 일부 또는 전체를 미리 읽어들인 후에 이 작업을 처리한다. 어떤 방법을 사용하든지 다른 스크립트를 시작하기 전에 변수를 초기화해야 한다. 변수가 정의된 후에는 stop() 함수를 호출하여 첫째 프레임(퀴즈가 시작하는 지점)에서 무비가 멈출 수 있도록 한다.

더 복잡한 무비에서는 무비의 나머지 부분을 미리 준비해두기 위해 함수를 호출하거나 변수 값을 대입하여 초기 조건을 설정하는 경우도 있다. 이러한 단계를 보통 '초기화(initialization)'라고 부른다. 움직이는 과정을 시작하거나 시스템이 동작되는 초기 조건을 정의하는 함수에는 보통 init이라는 이름을 붙인다.

우리가 만드는 퀴즈의 init 코드는 〔예제 1-1〕에 나와 있으며 무비의 scripts 레이어의 1번 프레임에 들어간다.

〔예제 1-1〕 퀴즈의 init 코드

```
// 메인 타임라인 변수 초기화
var q1answer;                // 문제 1에 대한 사용자의 답변
var q2answer;                // 문제 2에 대한 사용자의 답변
var totalCorrect = 0;        // 정답 개수
var displayTotal;            // 사용자의 점수를 표시하기 위한 텍스트 필드

// 첫 번째 질문에서 무비를 정지시킴
stop();
```

init의 첫 줄은 코드 주석(code comment)이다. 코드 주석은 간단한 설명을 덧붙이는 데 쓰인다. 한 줄짜리 주석은 두 개의 슬래시를 연속해서 쓴 다음 공백을 추가하는 것으로 시작된다. 그리고 나서 그 뒤에 필요한 내용을 입력한다.

```
// 이 부분은 주석입니다.
```

다음과 같이 주석을 코드와 같은 줄에 넣어도 상관없다.

```
x = 5; // 이것도 주석입니다.
```

〔예제 1-1〕의 둘째 줄에서는 q1answer라는 변수를 만든다. 변수를 만들 때는 var라는 키워드 뒤에 변수 이름을 적는다는 것을 다시 떠올려 보자.

```
var favoriteColor;
```

둘째 줄부터 다섯째 줄까지는 앞으로 사용할 변수를 선언하는 부분이며 각 변수의 사용 목적을 주석으로 적어 놓았다.

- q1answer와 q2answer에는 사용자의 답변(사용자가 1, 2 또는 3 중에서 고른 값)이 저장된다. 이 값을 이용하여 사용자가 정답을 선택하는지 결정하게 된다.
- totalCorrect는 퀴즈의 끝 부분에서 정답의 개수를 세는 데 쓰인다.
- displayTotal은 totalCorrect의 값을 화면에 출력할 때 사용할 텍스트 필드의 이름이다.

〔예제 1-1〕의 넷째 줄을 자세히 살펴보자.

```
var totalCorrect = 0;  // 정답 개수
```

여기서는 두 가지 작업을 처리한다. 우선 totalCorrect라는 변수를 선언하고 대입 연산자인 등호(=)를 이용하여 그 변수에 0을 대입한다. 사용자가 한 문제도 제대로 맞추지 못했을 때를 대비하여 totalCorrect의 기본값을 0으로 설정한다. 다른 변수는 퀴즈가 진행되면서 직접 값을 대입하도록 되어 있으므로 초기값을 설정하지 않아도 된다.

변수를 정의한 다음에는 stop() 함수를 호출하여 퀴즈가 시작되는 1번 프레임에서 무비를 정지시킨다.

```
// 첫 번째 질문에서 무비를 정지시킴
stop();
```

stop() 함수는 플래시 4 이전 버전에서 사용한 stop 액션(플레이헤드를 현재 프레임에서 잠시 멈춘다)과 똑같은 역할을 한다.

stop() 함수를 호출하는 부분 위에 있는 주석을 살펴보자. 이 주석에서는 그 아래에 있는 코드를 작성한 이유를 설명한다. 주석은 필수적인 것은 아니지만 한동안 작업을 하지 않다가 다시 작업할 때, 또는 다른 사람에게 코드를 넘겨줄 때 코드를 이해하는 데 도움이 된다. 또한 주석이 있으면 코드를 훑어보기 편해지는데, 디버깅할 때 큰 도움이 된다.

이제 init 코드에서 어떤 작업을 처리하는지 알았으므로 퀴즈 무비에 이 코드를 추가해 보자.

1. scripts 레이어의 1번 프레임을 선택한다.
2. Window → Actions를 선택하면 Frame Actions 패널이 나타난다.
3. 전문가 모드(Expert mode)에서 사용 중인지 확인한다. Edit → Preferences에서 영구적으로 설정할 수 있다.
4. Frame Actions 패널의 오른쪽에 〔예제 1-1〕에 나온 대로 init 코드를 입력한다.

변수 명명법

지금까지 적지 않은 변수를 접해봤는데, 대소문자를 섞어 쓰는 것에 대해 궁금해하는 독자들도 있을 것이다. 프로그래밍 경험이 전혀 없는 독자라면 firstName이나 totalCorrect에서처럼 단어 중간에 대문자가 들어 있는 것이 조금 이상하게 느껴질지도 모른다. 변수 이름에서 두 번째 이후에 오는 단어를 대문자로 적으면 단어를 구분하기 편하다. 이런 방법을 쓰는 것은 스페이스나 대시를 변수 이름에 사용할 수 없기 때문이다. 하지만 변수 이름의 첫 글자는 대문자로 쓰면 안 된다. 대문자로 시작하는 이름은 변수 이름이 아니라 객체 클래스 이름으로 쓰이기 때문이다.

first_name이나 total_correct와 같은 식으로 대문자 대신 밑줄(_)을 이용하는 것을 선호한다면 일관되게 그러한 방식을 사용하는 것이 좋다. firstName 같은 형식과 second_name 같은 형식을 섞어 쓰는 것은 좋지 않다. 자신이 만든 코드를 다른 프로그래머가 쉽게 읽을 수 있도록 하려면 둘 중 한 가지 방법만 선택하여 사용하자. 언어에 따라 변수 이름의 대소문자를 구분하는 경우가 있기 때문에 firstName과 firstname은 서로 다른 변수로 쓰이는 경우도 있다. 하지만 액션스크립트에서는 대소문자를 구분하지 않는다. 그렇다고 해서 같은 변수를 대소문자 구분 없이 아무렇게나 표기하는 것은 매우 안 좋은 방법이다. 예를 들어 어떤 변수를 xPOS라고 쓰기로 했다면 다른 곳에서 xpos라고 쓰는 것은 피해야 한다.

변수나 함수 이름은 항상 그 변수나 함수의 의미를 쉽게 이해할 수 있도록 만드는 것이 좋다. 'foo'와 같이 아무 쓸모 없는 이름은 사용하지 않는 편이 낫고, 'x', 'i'와 같이 한 글자로 된 변수명은 루프의 인덱스(숫자를 세는 데 쓰는 변수)와 같이 간단한 경우에만 사용하도록 하자.

프레임 레이블 추가하기

퀴즈의 init 스크립트 및 문제 부분을 만들었다. 이제 퀴즈 재생을 제어할 수 있도록 프레임 레이블을 추가해야 한다.

사용자가 한 번에 한 문제씩 풀 수 있도록 하기 위해 문제 1과 문제 2의 내용을 1번 프레임과 10번 프레임으로 갈라놓았다. 플레이헤드를 그 키프레임으로 옮겨서 각 슬라이드에 하나씩 문제가 들어가는 슬라이드쇼 효과를 만들자. 문제 2가 10번

프레임에 있으므로 문제 2를 화면에 표시하려면 다음과 같이 gotoAndStop() 함수를 호출하면 된다.

```
gotoAndStop(10);
```

이렇게 하면 플레이헤드가 문제 2가 있는 10번 프레임으로 움직인다. 참 잘 만든 코드처럼 보인다. 하지만 잘 생각해 보면 그렇지 않다는 것을 알 수 있다. gotoAndStop() 함수를 호출할 때 10이라는 숫자를 써도 여기서는 제대로 동작하긴 하지만 그다지 확장성이 좋지는 않다. 예를 들어 10번 프레임 앞에 있는 타임라인에 다섯 개의 프레임을 새로 집어넣는다면 문제 2의 프레임 번호가 15번으로 갑자기 바뀌게 되므로 gotoAndStop(10) 이라는 명령어를 내리면 엉뚱한 프레임으로 이동하게 된다. 프레임이 타임라인에서 움직이더라도 코드가 제대로 작동하도록 하고 싶다면 프레임 번호 대신 프레임 레이블(frame label)을 사용한다. 프레임 레이블은 q2나 quizEnd와 같이 타임라인의 특정 위치를 표시하기 위한 이름이다. 위치의 레이블을 만들고 나면 숫자 대신 이름을 이용하여 그 프레임을 찾아낼 수 있다.

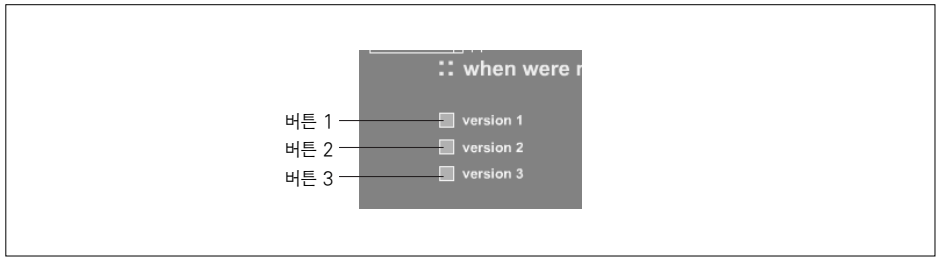
프레임 레이블이 없으면 프로그래밍이 거의 불가능해진다. 필자는 gotoAndStop()과 같은 재생 제어 함수에서 프레임 번호를 사용하는 일이 거의 없다. 퀴즈에 필요한 모든 프레임에 레이블을 붙여서 나중에 퀴즈 문제를 이동하는 데 사용하도록 하자.

1. labels 레이어에서 1번 프레임을 클릭한다.
2. Modify → Frame을 선택하면 Frame 패널이 화면에 나타난다.
3. Label 텍스트 필드에 init이라고 입력한다.
4. labels 레이어의 10번 프레임에 아무 내용이 없는 키프레임을 추가한다.
5. Frame 패널의 Label 텍스트 필드에 q2라고 입력한다.
6. labels 레이어의 20번 프레임에 아무 내용이 없는 키프레임을 추가한다.
7. Frame 패널의 Label 텍스트 필드에 quizEnd라고 입력한다.

답변 버튼 스크립트 만들기

지금까지 문제를 만들고 변수를 초기화하고 프레임 레이블도 붙여 보았다. 무비를 지금 테스트한다면, 문제 1과 세 개의 답변 버튼이 나타나긴 하지만 버튼을 눌렀을 때 아무런 응답도 없을 것이고 문제 2로 넘어갈 수도 없게 된다. 따라서 답변 버튼에 어떤 코드를 추가하여 한 문제에 응답하면 다음 문제가 나오는 식으로 만들어야 한다.

편의상 〔그림 1-8〕에 나온 것과 같이 답변 버튼을 각각 버튼 1, 버튼 2, 버튼 3으로 부르기로 하자.



〔그림 1-8〕 답변 버튼

세 개의 버튼에는 거의 비슷한 스크립트가 들어간다. 〔예제 1-2〕에서 〔예제 1-4〕까지의 내용은 각 버튼에 해당하는 코드이다.

〔예제 1-2〕 문제 1, 버튼 1의 코드

```
on (release) {
   q1answer = 1;
   gotoAndStop ("q2");
}
```

〔예제 1-3〕 문제 1, 버튼 2의 코드

```
on (release) {
   q1answer = 2;
   gotoAndStop ("q2");
}
```

〔예제 1-4〕 문제 1, 버튼 3의 코드

```
on (release) {
   q1answer = 3;
   gotoAndStop ("q2");
}
```

이 버튼 코드는 두 개의 선언문(둘째, 셋째 줄)으로 이루어지며, 이 코드는 마우스클릭이 감지되었을 때만 동작한다. 사람들이 쓰는 말로 표현하자면 각 버튼에 해당하는 코드는 '사용자가 버튼을 클릭하면 사용자가 1, 2, 3 중에 어떤 답을 골랐는지 기록한 후에 문제 2로 가라' 라고 쓸 수 있다. 자세한 동작 과정을 설명하면 다음과 같다.

첫째 줄은 이벤트 핸들러를 시작하는 부분이다.

```
on (release) {
```

이벤트 핸들러에서는 사용자가 버튼 1을 클릭할 때까지 끈기 있게 기다린다. 이벤트 핸들러는 무비가 실행되는 동안에 어떤 일(마우스 클릭 같은 일)이 일어날 때까지 기다린다는 것을 상기하자. 이벤트가 발생하면 적절한 핸들러에 있는 코드가 실행된다.

첫째 줄로 시작하는 이벤트 핸들러를 자세히 살펴보자. 'on' 이라는 키워드는 이벤트 핸들러의 시작을 알린다. 괄호 안에 들어 있는 'release' 라는 키워드는 이벤트 핸들러에서 기다리고 있는 이벤트 유형을 나타낸다. 이 경우에는 사용자가 마우스 버튼을 클릭했다가 버튼에서 손을 떼는 'release' 라는 이벤트를 기다린다. 왼쪽 중괄호({)는 release 이벤트가 발생했을 때 실행할 선언문 블록의 시작을 나타낸다. 코드 블록을 끝내는 넷째 줄에서는 오른쪽 중괄호(})를 이용하여 이벤트 핸들러가 끝남을 나타낸다.

둘째 줄은 release 이벤트가 발생했을 때 실행되는 첫 선언문이다. 둘째 줄에 있는 코드는 이제 독자들에게도 익숙할 것이다.

```
   q1answer = 1;
```

여기서는 q1answer라는 변수를 1로 설정한다(다른 답변 버튼에서는 그 변수를 각각 2나 3으로 설정한다). q1answer 변수에는 첫째 문제에 대한 사용자의 답변이

저장된다. 답변을 저장하고 나면 버튼 코드의 셋째 줄에 있는 코드를 통해 문제 2로 넘어간다.

```
gotoAndStop ("q2");
```

셋째 줄에서는 gotoAndStop( ) 함수를 호출하는데, 이 때 'q2' 레이블을 인자로 넘겨서 플레이헤드를 문제 2가 시작되는 q2라는 프레임으로 움직인다.

이제 버튼 코드가 작동하는 원리를 이해했으면 이 코드를 문제 1의 버튼에 추가해 보자.

1. Actions 패널이 열린 상태에서 스테이지에 있는 버튼 1을 선택한다. Frame Actions라는 타이틀이 Object Actions로 바뀐다. 지금부터 입력하는 코드는 모두 버튼 1(스테이지에서 선택한 객체)로 들어간다.

2. Actions 패널의 오른쪽에 〔예제 1-2〕에 있는 코드를 입력한다.

3. 1, 2 단계를 반복하여 버튼 2와 버튼 3에 코드를 추가한다. 〔예제 1-3〕과 〔예제 1-4〕에 나온 것처럼 버튼 2에서는 q1answer를 2로, 버튼 3에서는 3으로 설정한다.

문제 2 버튼의 코드도 결국은 문제 1 버튼의 코드와 똑같다(답변 변수의 이름과 gotoAndStop( )을 호출할 때 쓰이는 인자가 바뀔 뿐이다). 문제 2의 버튼 1에 해당하는 코드는 〔예제 1-5〕와 같다.

〔예제 1-5〕 문제 2, 버튼 1의 코드

```
on (release) {
   q2answer = 1;
   gotoAndStop ("quizEnd");
}
```

사용자가 문제 2에서 선택한 답을 기록해야 하기 때문에 q1answer 대신 q2answer를 이용한다. 또한 문제 2를 마친 후에 플레이헤드를 퀴즈의 끝(quizEnd라는 레이블이 붙은 프레임)으로 보내려면 gotoAndStop( ) 함수를 호출할 때 'quizEnd' 라는 인자를 사용해야 한다.

문제 2에 있는 버튼에 버튼 코드를 추가하자.

1. question 2 레이어의 10번 프레임을 클릭한다.
2. 버튼 1을 클릭한다.
3. Actions 패널에 〔예제 1-5〕에 나온 코드를 입력한다.
4. 2, 3 단계를 반복하여 버튼 2와 버튼 3에 코드를 추가한다. 버튼 2에서는 q1answer를 2로, 버튼 3에서는 3으로 설정한다.

이제 여섯 개의 버튼에 모두 버튼 코드를 추가하고 나면 코드가 계속 반복된다는 것을 깨달았을 것이다. 각 버튼에 있는 코드는 서로 몇 글자씩밖에 다르지 않다. 이러한 방식은 그다지 효율적이지 않다. 이러한 코드 대신 일괄적으로 답변을 기록하고 다음 스크린으로 넘어가는 뭔가를 만드는 것이 현명할 것이다. 9장에서 함수를 이용하여 이런 작업을 일괄적으로 처리할 수 있는 방법을 살펴보자.

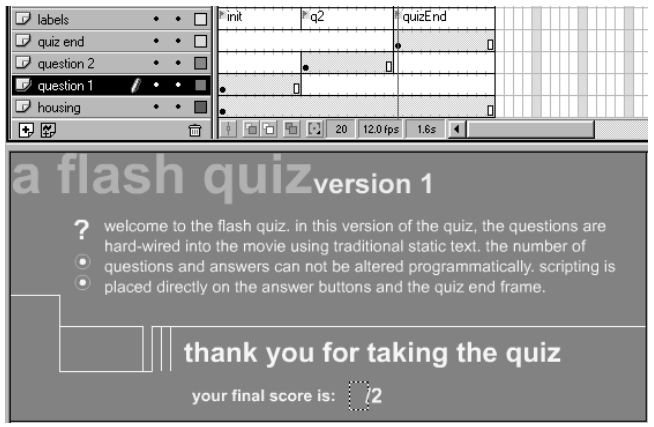
퀴즈 마지막 장면 만들기

퀴즈가 거의 완성되었다. 지금까지 사용자가 질문에 답하면 다음 문제로 넘어갈 수 있도록 해주는 스크립트가 들어 있는 문제를 두 개 만들었다. 이제 사용자의 성적을 알려주는 퀴즈의 마지막 장면을 만들어야 한다.

퀴즈 끝부분을 만들려면 기본적인 플래시를 만들고 스크립트도 짜야 한다. 우선 플래시를 만들어보자.

1. question 2 레이어의 20번 프레임에 새로운 키프레임을 추가한다. 이렇게 해야 퀴즈 마지막 장면에 문제 2가 겹쳐서 나타나는 것을 막을 수 있다.
2. quiz end 레이어의 20번 프레임에 새로운 키프레임을 추가한다.
3. 그 프레임에서 스테이지에 다음과 같은 텍스트를 입력한다. "Thank you for taking the quiz. Your final score is: /2." 점수를 출력하려면 'is:' 와 '/2' 사이에 빈 공간을 적당히 남겨두어야 한다.
4. scripts 레이어의 20번 프레임에 새로운 키프레임을 추가한다.

퀴즈의 마지막 장면은 위와 같은 단계를 거쳐서 만들 수 있다. 이렇게 하고 나면 〔그림 1-9〕에 나온 것과 같은 마지막 장면이 만들어진다.



〔그림 1-9〕 성적 공개

이제 마지막 장면의 스크립트를 만들 차례이다. 플레이헤드가 quizEnd 프레임으로 오면 사용자의 점수를 계산해야 한다. 따라서 플레이헤드가 20번 프레임에 도착할 때 점수를 계산하기 위한 스크립트가 필요하다. 타임라인의 키프레임에 있는 모든 스크립트는 플레이헤드가 그 프레임에 도달하는 순간에 실행되기 때문에 계산 스크립트는 scripts 레이어의 20번 프레임에 추가한 키프레임에 넣으면 된다.

계산 스크립트에서는 우선 사용자의 점수를 결정한 다음 그 점수를 화면에 표시한다.

```
// 사용자의 정답 개수를 센다.
if (q1answer == 3){
  totalCorrect = totalCorrect + 1;
}
if (q2answer == 2){
  totalCorrect++;
}
// 화면에 있는 텍스트 필드에 사용자의 점수를 표시한다.
displayTotal = totalCorrect;
```

첫째 줄과 여덟째 줄은 스크립트를 설명하는 코드 주석이다. 둘째 줄에서 두 조건문 중 첫 째 조건문이 시작된다. 여기서는 q1answer 변수를 이용하여 다음과 같은 코드를 만들었다.

```
if (q1answer == 3){
```

if라는 키워드는 어떤 조건이 만족될 때만 실행할 선언문의 리스트를 제공하겠다는 것을 의미한다. 그 조건은 if 키워드 바로 뒤에 오는 괄호 안에 들어간다(q1answer == 3). 그리고 왼쪽 중괄호는 조건에 따라 실행되는 선언문이 시작됨을 알린다. 따라서 둘째 줄은 'q1answer의 값이 3이면 중괄호 안에 있는 선언문을 실행시킨다' 라고 이해할 수 있다.

하지만 'q1answer == 3' 이라는 조건은 정확하게 어떤 식으로 작동할까? 이 구문을 풀어보자. q1answer가 문제 1에 대한 답변을 저장해 놓은 변수라는 것은 이미 알고 있을 것이다. 숫자 3은 문제 1의 정답이다. 무비 클립은 플래시 3에서 처음 도입되었기 때문이다. 변수와 숫자 3 사이에 있는 두 개의 등호(==)는 양쪽에 있는 표현식을 비교하는 동치(equality) 비교 연산자이다. 왼쪽에 있는 표현식(q1answer)과 오른쪽에 있는 표현식(3)이 같으면 이 조건이 만족되기 때문에, 중괄호 안에 있는 선언문이 실행된다. 그렇지 않으면 조건이 만족되지 않기 때문에 중괄호 안에 있는 선언문을 건너 뛰게 된다.

플래시는 퀴즈 문제의 정답을 알지 못한다. q1answer의 값이 3인지 아닌지를 조사함으로써 사용자가 문제 1의 정답을 맞췄는지 여부를 알아낼 수 있다. 만약 정답을 제대로 맞췄다면 다음과 같이 사용자의 총점을 1점 올려준다.

```
totalCorrect = totalCorrect + 1;
```

셋째 줄은 'totalCorrect의 원래 값에 1을 더한 새로운 값을 대입하라' 라는 것을 의미한다(즉, totalCorrect 값을 증가시킨다). 어떤 변수의 값을 1씩 증가시키는 작업은 꽤 자주 쓰이기 때문에 ++라는 연산자가 따로 지정되어 있다.

따라서 아래와 같이

```
totalCorrect = totalCorrect + 1;
```

라고 쓰는 대신 보통 다음과 같이 쓴다.

```
totalCorrect++;
```

이렇게 하면 같은 작업을 더 간결하게 처리할 수 있다.

넷째 줄에서는 첫째 조건이 만족되었을 때 실행되는 선언문 블록을 끝마친다.

```
}
```

다섯째에서 일곱째 줄까지는 다른 조건을 처리한다.

```
if (q2answer == 2){
  totalCorrect++;
}
```

여기서는 사용자가 문제 2의 정답(MP3는 플래시 4부터 지원되었다)을 제대로 맞췄는지 조사한다. 사용자가 두 번째 답변을 선택했다면 증가 연산자인 ++를 이용하여 totalCorrect 값을 하나 증가시킨다.

이 퀴즈에는 문제가 두 개밖에 없기 때문에 정답 개수를 세는 작업이 끝났다. 사용자가 정답을 선택한 개수만큼 totalCorrect 값을 증가시켰기 때문에, totalCorrect에는 사용자의 총점이 저장된 상태이다. 이제 할 일은 사용자에게 점수를 보여주는 것인데, 이 작업은 퀴즈 마지막 장면의 스크립트의 마지막 줄인 아홉째 줄에서 처리한다.

```
displayTotal = totalCorrect;
```

이제 변수에 대해서는 충분히 배웠으니 아홉째 줄에서 displayTotal이라는 변수에 totalCorrect의 값을 대입한다는 것은 이해할 수 있을 것이다. 하지만 화면에는 어떻게 점수를 표시할까? 아직까지는 화면에 점수가 표시되지 않는다. 점수를 화면에 표시하려면 화면에 직접 표시되는 텍스트 필드(text field) 변수라는 특별한 종류의 변수를 만들어야 한다. 이러한 변수를 직접 만들어보자.

1. Text 도구를 선택한다.
2. quiz end 레이어에서 20번 프레임을 선택한다.
3. 미리 만들어둔 텍스트의 '/2' 바로 앞에 포인터를 갖다 놓고 스테이지를 클릭한다.
4. 숫자 한 개가 들어갈 정도의 적당한 크기로 마우스를 드래그하여 텍스트 박스 크기를 조절한다.

5. Text → Options를 선택한다.

6. Text Options 패널에서 Static Text를 Dynamic Text로 변경한다.

7. Variable 텍스트 필드에 displayTotal을 입력한다.

이제 displayTotal 변수를 화면에 표시할 수 있다. 스크립트에서 displayTotal 값을 바꾸면 그에 해당하는 텍스트 필드 변수의 내용이 화면에 갱신된다.

퀴즈 테스트

이제 모든 작업이 끝났다. 퀴즈가 완성된 것이다. Control → Test Movie를 선택하여 퀴즈가 제대로 되는지 확인해보자. 다양한 답변을 입력해보고 점수가 제대로 나오는지 살펴보자. 새로운 버튼을 만들어서 그 버튼에 다음과 같은 코드를 집어넣으면 무비를 다시 재생시킬 수도 있다.

```
on (release) {
  gotoAndStop("init");
}
```

totalCorrect 값을 0으로 설정하는 부분이 init 프레임에 있기 때문에, 매번 플레이헤드가 init으로 움직일 때마다 그 값은 0으로 다시 설정된다.

퀴즈가 제대로 작동하지 않는다면 온라인 코드 창고에 있는 샘플 퀴즈와 비교해보기 바란다. 또한 '19장. 디버깅'에서 배우게 될 문제 해결 기법을 이용해도 좋다.

앞으로 배울 내용

자, 1장을 마치고 나니 기분이 어떤가? 지금까지 구문도 여러 개 배우고 문법과 단어도 조금 배웠으며, 플래시와 간단한 대화(퀴즈 프로그램)도 나누어 보았다. 이 정도면 첫날 치고는 많이 배운 셈이다.

독자들도 이미 알고 있겠지만 액션스크립트에 대해 배울 내용이 아직도 많이 남아 있다. 하지만 내용을 조금만 알아도 꽤 많은 작업을 할 수 있다. 지금까지 배운

것만으로도 이것저것 장난칠 만한 일들이 많을 것이다. 이 책의 나머지 부분에서는 지금까지 대략 훑어본 내용들을 더 깊이 배우고 실전 예제를 통해 직접 사용해봄으로써 액션스크립트에 대한 지식을 더욱 튼튼하게 할 것이다. 물론 지금까지 전혀 소개하지 않은 주제도 다룰 것이다.

의사소통, 협동을 염두에 두고 분명하게 자신의 뜻을 표현해야 한다는 점을 기억해두자. 그리고 뭔가 멋진 것을 만들고 그것을 다른 사람들과 함께 나누고 싶다면 http://www.moock.org/webdesign/flash/contact.html로 필자에게 보내주기 바란다.

이제 실용적인 내용은 어느 정도 익혔으므로 앞으로 몇 장에 걸쳐 자세한 기본 지식을 배워보도록 하자. 이 과정을 마치고 나면 액션스크립트를 더 깊이 이해하여 좀더 복잡한 무비도 만들 수 있게 될 것이다.