

Events

We've described a computer program as a series of instructions that the computer carries out step-by-step. Some simple computer programs consist of nothing more than that; a few steps which the computer carries out, at which point the program ends.

However, ActionScript programs are designed to keep running, waiting for user input or other things to happen. Events are the mechanism that determines which instructions the computer carries out and when. You can listen for events in your code using event listeners. Event listeners use functions that you write to respond to specific events. When writing an event listener, you need three pieces of information:

1. The variable name of the object that is listening for the event. For example, a Movieclip named `triangle_mc`
2. The event that you are listening for. For example, a mouse click
3. The name of the function that you are going to write that contains the commands to execute when the event occurs. For example, `triangle_clicked`

You create an event listener by calling the `addEventListener` method as a child of the object listening, like this:

```
triangle_mc.addEventListener () ;
```

You need to pass in as parameters the event and function name, so the completed listener is:

```
triangle_mc.addEventListener (MouseEvent.CLICK , triangle_clicked);
```

In the next section we will see how to write the `triangle_clicked` function.

Functions

A function is simply a way to group multiple commands so that they can be run together with a single statement. Most times, when an event occurs, we will want to do many commands, not just one. Functions are like shortcuts - they provide a way to group a series of actions under a single name, and can be used to perform calculations. Functions are particularly important for handling events, but are also used as a general tool for grouping a series of instructions. The commands in a function only run when you call the function. You call a function by using its name followed by parentheses (). You use the parentheses to enclose any parameters you want to send to the function. Parameters are information or data that you can give to the function to use when it is running.

```
function moveRectangle()  
{  
rectangle_mc.x = rectangle_mc.x + 10;  
rectangle_mc.y = rectangle_mc.y + 10;  
}
```

in order to call this function we use the following statement:

```
moveRectangle ();
```

Every time we call the function, it will move the rectangle 10 pixels to the right, and 10 pixels down. If we re-write this function using parameters, it becomes a lot more powerful. Instead of simply adding 10 to both x and y properties, lets change it so we add whatever value we pass to the function to them instead.

```
function moveRectangle (moveX , moveY)  
{  
rectangle_mc.x = rectangle_mc.x + moveX;  
rectangle_mc.y = rectangle_mc.y + moveY;  
}
```

Now whenever we call the function, the first parameter (moveX) will be added to the x location, and the second parameter (moveY) will be added to the y location. In order to call this function, we use the same statement, but include the parameters in between the parentheses:

```
moveRectangle (10,-20);
```

This will move the rectangle 10 pixels to the right, and 20 pixels up. So now we know how to write a function, it's time to learn how to write an event listener function. An event listener function is like other functions, except it has a couple extra pieces of information you need to put in the declaration. Here is an example:

```
function triangle_clicked ( event : MouseEvent ) : void  
{  
rectangle_mc.x = rectangle_mc.x + 30;  
}
```

- **event : MouseEvent** is a parameter passed to the function by flash with information about the event such as the exact location of the mouse when it was clicked. This is required, even though we won't be using the information
- **: void** at the end indicates that the function does not return any value when it is finished. We won't use this either.
- **triangle_clicked** is a name we just made up, and can be whatever we want to call it, but it does need to be unique.