Unit 2: ActionScript 3 Fundamentals

Unit Objectives

After completing this unit, you should be able to:

- Write basic AS3 statements
- Comment and trace your code
- Declare and use variables of specific data types
- Convert between data types
- Group statements into reusable functions

Unit Topics

- Getting started with coding
- ActionScript Basics
- Operators and Expressions
- Using Functions

Getting started with coding

When you open Flash CS3 for the first time, the application will prompt you with the Flash CS3 Start Page. This screen will allow you to:

- open a recently opened document
- open any document
- create a new document
- select an existing template

The Create New list gives you three types of Flash files. For the purposes of this course, it is important that we select Flash File (ActionScript 3.0).

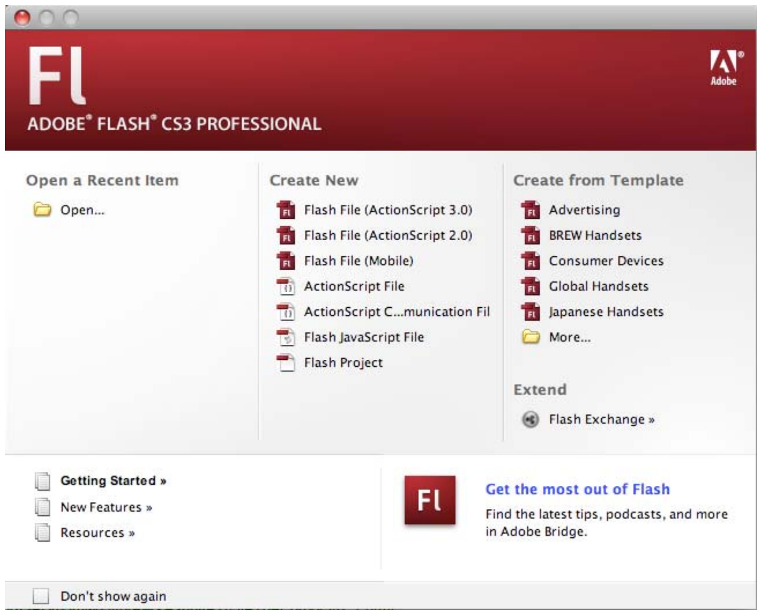


Figure: The Flash CS3 Start Page

The ActionScript Panel

Before we start writing ActionScript, it is important that we understand the programming interface in Flash CS3. The ActionScript Panel is the core interface for our programming, and includes many features to improve your development speed, help debug your application, and quickly format your code.

Some of the tools the Panel includes are:

- Code Checking
- Auto Formatting
- Single and Multi-line comments
- Quick dragging and dropping of classes
- Tree list of ActionScript

Using these tools we can greatly speed up our development time. We can also check our code for any errors before running it. For many beginning developers understanding these tools and using them as you program can help you become much more productive, and quickly grasp the ActionScript language.

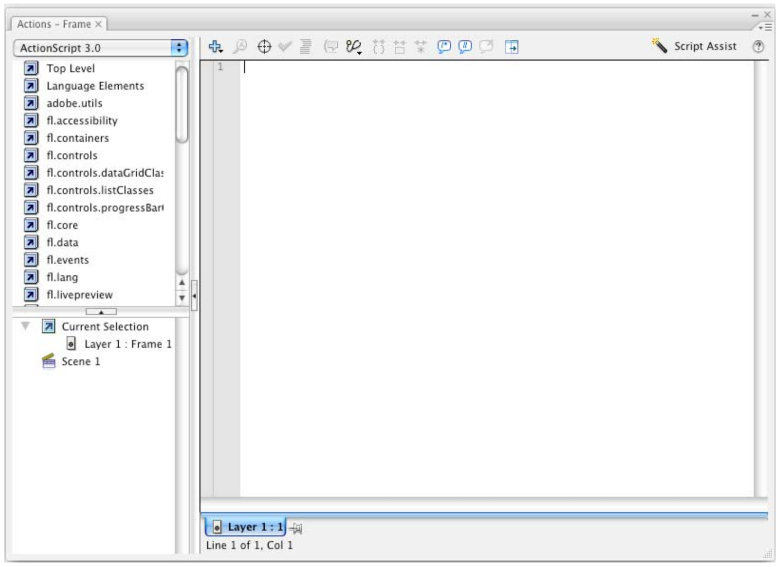
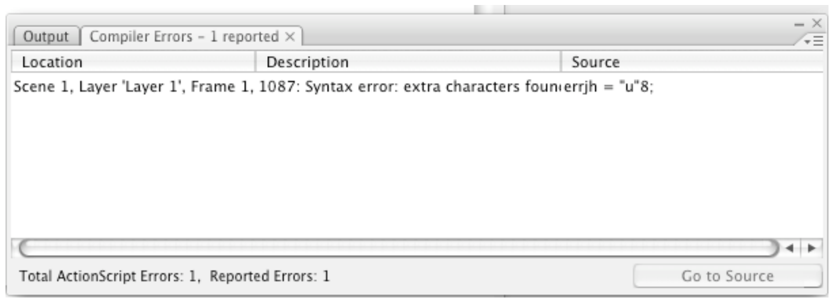


Figure: The ActionScript 3 Editor

Code Checking

The check mark in the ActionScript panel's tool bar allows you to look for syntax errors in your code before you run it. If an error is found, Flash will display an error panel (screenshot below) with details on the error and its line number. Double clicking on the error message in the panel, Flash will highlight the line where the error exists for closer inspection. It is a good idea to check for errors before running your code. It is also helpful to incrementally check for errors while writing large blocks of code.

As a general rule, the first error in this list may cause the errors that follow it. For that reason, you should always start fixing errors from the beginning of the Compiler Error log and work your way down, rechecking after each change.



Location	Description	Source
Scene 1, Layer 'Layer 1', Frame 1,	1087: Syntax error: extra characters foun	errjh = "u"8;

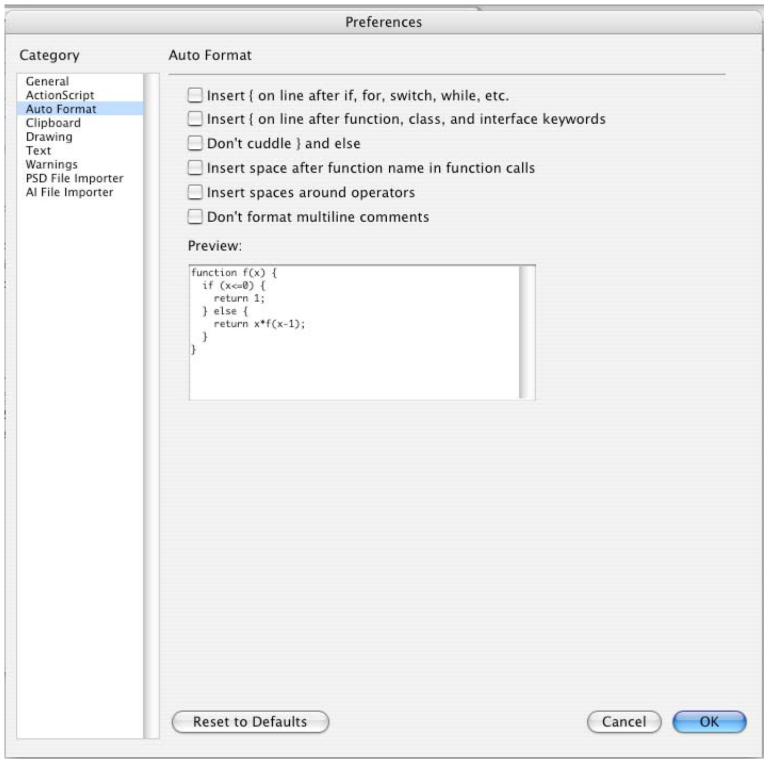
Total ActionScript Errors: 1, Reported Errors: 1

Figure: Error panel

Auto Formatting

Formatting your code can make it much easier to read or skim through. If you are copying code from another application, or simply want to format your code to give it a consistent look, the formatting tool on the ActionScript Panel's toolbar can be a great help.

The ActionScript preferences will give you options to customize formatting. This way you can auto-format your code to your specific liking. Every developer has specific preferences for how they prefer to see their code. As you become more accustomed to writing ActionScript you will determine your own style and preferences.



- ☐ Insert { on line after if, for, switch, while, etc.
- ☐ Insert { on line after function, class, and interface keywords
- ☐ Don't cuddle } and else
- ☐ Insert space after function name in function calls
- ☐ Insert spaces around operators
- ☐ Don't format multiline comments

Preview:

```
function f(x) {
  if (x<=0) {
    return 1;
  } else {
    return x*f(x-1);
  }
}
```

Figure: Auto-Format preferences

Commenting your code

As you write more complex code you may find the need to temporarily or permanently remove code from an operation. Commenting allows you to disable code while keeping it within your application for future reference.

ActionScript supports two types of comments, single line and multiline comments. A single line comment uses two forward slashes and prevents any code or text to its right from executing.

```
//This is a comment
```

A multiline comment allows you to comment out a block of code that is one line or more. A multiline comment begins with a backslash and an asterisk, and ends with an asterisk and backslash. Flash will ignore any characters within the comment.

```
/*
	This entire block
	of text is now a
	comment
*/
```

The toolbar on the ActionScript Panel provides icons for applying single and multiline comments to existing code within your application. Comments can also provide a useful method for documenting your code for later reference as you write it.

A new tool in ActionScript 3 will also allow you to generate documentation from your code using comments. If you plan to share your code publicly, or within your organization, these tools can make good comments even more important.

Tracing Your Code

To view the current values of variables or the output of an operation, you can use the `trace` statement. The `trace` statement is a great tool for debugging, and allows you to output values during development to the Output Panel.

The following code demonstrates a `trace` statement that outputs the text **`“Hello World”`** to the Output Panel.

```
trace( "Hello World" );
```

We will routinely use the `trace` statement to examine the values of variables or the output from operations while running. We will also use it to test code that does not have a visual interface.

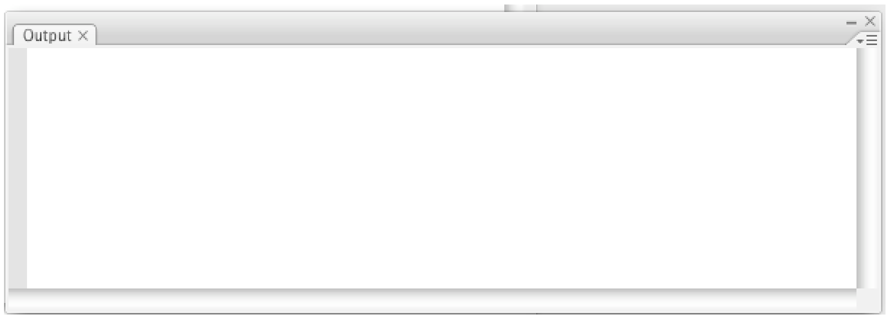


Figure: The `trace` output window

Running your Application

To run your application select **File** > **Control** > **Test Movie**. You can also use the short cut **Ctrl - Enter** (**Command - Enter** on a Mac).

ActionScript Basics

ActionScript is the programming language of the Adobe Flash Player runtime. Originally developed as a way for Flash developers to program interactivity, ActionScript enables efficient programming of Adobe Flash applications for everything from simple animations to complex, data-rich, interactive application interfaces.

ActionScript 3.0, introduced in Flash Player 9, is based on ECMAScript—the same standard that is the basis for JavaScript. ActionScript 2.0 continues to be supported in Flash Player 9.

As the core of the ActionScript language we need to explore keywords that are reserved by the language, case sensitivity, and how to declare basic statements as well as fundamental elements, variables, and functions. We will build on these principles as our applications become more complex.

Declaring Variables

A variable is a bucket that we will use to store information or values. Later on we can access its content in an operation or update the content within it. When declaring a variable, it must have a name (we call them identifiers or instance names) and also state the type of data that can be stored within it.

For example, I can create a variable named **`firstName`** that stores a string of characters and use it to store your name for later use. In this case, my code will look something like this:

```
var firstName:String;
```

This statement gives the variable a name we can identify it with, our instance name, and declares the type of data we can store within it. The **`var`** keyword alerts ActionScript that the code following it will declare an instance of a variable.

In many cases, we will want to populate our variables with initial information as soon as we create them. The process of placing initial data into our variable is known as instantiation. You can instantiate the above variable **`firstName`** with the string value "Jason" using the following statement:

```
var firstName:String = "Jason";
```

While we could instantiate the variable later on in code, as a best practice we want to instantiate our variables with a value as soon as we can. This will prevent us from accidentally using a variable in an operation that has no value.

Note: All statements in ActionScript end with a semi-colon. This alerts ActionScript that end of the statement has been reached.

If you attempt to trace a variable that has no value or has not been instantiated it will have a default value applied depending on the variable type. Many ActionScript mistakes are a direct result of not instantiating a variable.

Default values are assigned as follows:

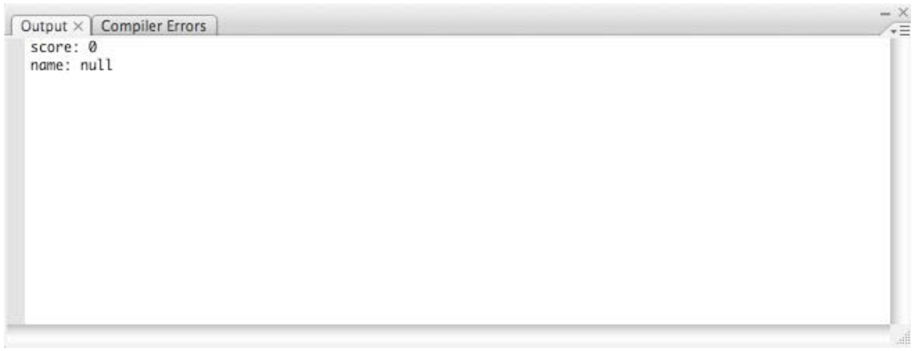
- Variables of type int, or uint are given a default value of 0.
- Number variables are given a default value of "NaN" (for "not a number").
- Strings are given a default value of null.

To test the value before using it, trace the value of the variable to the Output Window.

For example, the following code generates output depicted in the figure below:

```
var playerScore:int;
var userName:String;

trace( "score: " + playerScore );
trace( "name: " + userName );
```



```
score: 0
name: null
```

Figure: Output from a simple variable declaration and trace

Declaring the Data Type

In the previous example the variable **`firstName`** is of type **`String`**. The data type defines the type of values we can refer to with a variable. ActionScript's basic or primitive types include the following:

Data Type	Description	Default Value
int	A positive or negative integer	0
uint	A positive (unsigned) integer	0
Number	A decimal (also known as double or floating point) or integer value	NaN (Not a Number)
String	A series of characters enclosed within quotation marks (")	null
Boolean	A true or false value	false

The following example illustrates variables of these types instantiated with an initial value:

```
var playerScore:int     = 2000;
var timeElapsed:uint    = 150000;
var price:Number        = 99.45;
var firstName:String    = "Jason";
var isAdmin:Boolean     = true;
```

Notice how each value used to instantiate these variables has a specific value.

Case Sensitivity

ActionScript is a case sensitive language. Because of this, it is very important that we use a consistent case when defining a variable's instance name. The following code block defines three unique variables:

```
var helloWorld:String;
var HELLOWORLD:String;
var HelloWorld:String;
```

For developers, maintaining proper case is extremely important and improper use can lead to debugging problems. As a best practice, most ActionScript developers use the camel case naming convention for their identifiers. Adobe also uses the camel case naming convention within their ActionScript code.

Camel case requires an identifier to start with a lower case character, and every word after the first word begins with a capital letter. For example:

```
var camelCase:String;
var myNameIsJason:String;
```

Note: As you will learn later in this course, the only exception to this rule is the identifier for a type or class. When writing your own class or declaring an instance of a class the type declaration will begin with an upper case character and every new word will also begin with a capital character.

- *CamelCase*
- *MyNameIsJason*

*The data types **`int`** and **`uint`** are the only two types in the ActionScript framework that do not follow this convention.*

Walkthrough 2-1: Declaring and Using Variables

In this walkthrough, you will declare variables to store the user's name and score. You will also use the trace statement to monitor the values of these variables after instantiation.

Steps

1. Start or return to Flash.
2. From the Flash Start Page create a new **Flash File (ActionScript 3.0)**.
3. Save the file as **FLAS3/walk/wt2-1/ wt2-1.fla**.
4. Select the layer's name and rename it Actions.
5. Select the **Actions** layer and open the Actions Panel (**F9**).
6. Declare the following variables:

   ```
   var userName:String = "Jason";
   var playerScore:int = 2500;
   ```

7. View the output of these variables using the Flash output window:

   ```
   trace(userName);
   trace(playerScore);
   ```

8. Run the Flash File by pressing **Ctrl-Enter** (Command-Enter on a Mac) or selecting **Control > Test Movie**.
9. Return to your code and delete the values that you assigned to the two variables.
10. Run the Flash file.
11. Discuss the values returned and what they mean with your instructor.
12. Save the document and keep it open for the next walkthrough.

- End of Walkthrough -

Operators and Expressions

Every language has operators and expressions that allow you to perform mathematical calculations, concatenate strings, and invoke functions.

Math Operators

Our mathematical operators are predefined characters we can use for common math operations. These operators are listed in the table below.

addition or concatenation	+	x = 2 + 4; //x becomes 6
subtraction	-	x = 2 - 4; //x becomes 2
multiplication	*	x = 2 * 4; //x becomes 8
division	/	x = 4 / 2; //x becomes 2
modulus (remainder after division)	%	x = 4 % 2; //x becomes 0

It is important to note that math in ActionScript follows mathematic rules for order of operation. This means that all operations will multiply, divide, add, then subtract. For this reason the following statement will multiply, then add.

```
x = 5 + 5 * 10;     //x becomes 55
```

To explicitly control the order of operation, use parentheses. In the following example we force addition to occur before multiplication.

```
x = ( 5 + 5 ) * 10;  //x becomes 100
```

Compound Operators

To speed up common operations ActionScript also provides several compound operators. Our compound operators provide a short-hand notation for writing common operations. The following table lists these operators:

x = x + 2;	x += 2;
x = x - 2;	x -= 2;
x = x * 2;	x *= 2;
x = x / 2;	x /= 2;
x = x + 1;	x++;
x = x - 1;	x--;

Compound operators can be very useful. They provide an easy method for doing common operations on variables with long instance names. Keep in mind that while these are shortcuts, and are technically optional, they can nevertheless greatly reduce the amount of code needed for an operation.

String Concatenation

While the "+" symbol is commonly used as a mathematic operator to add values, it can also be used to combine two strings together. The process of joining strings is referred to as concatenation.

The following example creates a new string by joining a **`firstName`** variable with the statement "**`Welcome to my application`**".

```
var firstName:String = "Jason";
var welcomeMessage:String =
                "Welcome to my application, " + firstName;
```

Once this code is run the **`welcomeMessage`** variable will store the joined statement. This means that outputting this variable will provide us with the statement "**`Welcome to my application, Jason`**".

*Note: The above code adds a space to insure the newly created string reads "**`Welcome to my application, Jason`**" instead of "**`Welcome to my application,Jason`**".*

Data Type Conversion

In the following example, ActionScript may not know if you want to add or to combine the values, because of the type mismatch between the "1" value and the number 2 value:

```
var x:String = "1" + 2; //x becomes "12"
```

Here ActionScript assumes we want to concatenate the above values since a mathematical operation between a string and a number is impossible. If we attempted to add the string and a number, and get back a numeric value, this would generate an error:

```
var y:int = "1" + 2;  //type mismatch error occurs
```

To alert ActionScript that we want to treat the string value "1" as a number and not as a string we need to convert it to the proper data type. This process of converting is commonly referred to as *casting*.

The following list shows how to convert our primitive types (int, uint, Number, String, and Boolean) to other primitive types.

```
var x:int = int( "5" );   //returns the value 5 as an int
var y:uint = uint( "5" ); //returns the value 5 as an uint
var a:String = String(12); //returns the value 12 as
String
```

Note: An illegal cast will generate an error. For example attempting to cast the string "Fred" to an int will return NaN or not a number as the result.

Walkthrough 2-2: Converting Data Types and Operators

In this walkthrough you will use operators to concatenate a string and update the player's score. You will also convert the score to a string using the `String` conversion function.

Steps

1. Return to wt2-1.fla in Flash.
2. Save the file as **FLAS3/walk/wt2-2/wt2-2.fla**.

Resolving a Data Mismatch

3. Open the Actions Panel (**F9**) with the **Actions** layer selected.
4. Declare a variable named `output` as type `String`:

   ```
   var output: String;
   ```

5. Set the `output` string equal to the value `playerScore`:

   ```
   output = playerScore;
   ```

6. Add a `trace` statement to display the value of `output.`

   ```
   trace(output);
   ```

7. Run the Flash file. Note that you receive a type mismatch error.
8. Return to Flash and correct the data mismatch error by casting to a `String:`

   ```
   output = String( playerScore );
   ```

9. Run the Flash file.

Concatenate Strings

10. Make the `output` string equal to a new string containing the variables `userName` and `playerScore` using concatenation:

    ```
    output = userName + "has a score of " + playerScore;
    ```

    Note: Don't forget that concatenation will implicitly convert a data type to a String. This allows us to still treat `playerScore` *as a String without converting it before hand. Also, remember to add spaces to insure enough white space exists in the rendered output.*

11. Run the Flash file.

- End of Walkthrough -

Using Functions

A function is a reusable block of code that can be executed by calling its name. The function can optionally return data once executed and can accept one or more arguments (also called parameters).

The function keyword

To write a function in ActionScript, we use the **`function`** keyword which alerts ActionScript that a function declaration is forthcoming. A function takes the following syntax:

```
function functionName( [arg1:Type, arg2:Type,…]):Type
{
   var value:Type = “value”;

   return value;
}
```

*Note: As a best practice a function’s name is written as a verb-noun pattern. For example **`doThis()`**, **`stopThat()`**, or **`goThere()`**. This alerts the user that this function will perform some form of task.*

The following function adds two numbers and returns the sum:

```
function addIntegers( int1:int, int2:int ):int
{
   return int1 + int2;
}
```

This function:

- Has a name or identifier of **`addIntegers`**()
- Accepts two arguments, **`int1`** and **`int2`**
- Declaration also states that once the function is called it will return a value of type **`int`**.

The code within the curly-braces is the operation that will be performed when the function is called. In this case, it adds the two arguments, **`int1`** and **`int2`**, and returns this value to the caller.

Calling a function

A function declaration simply defines what the function will do. To run the function, we call it passing any required arguments. The following code calls the **`addIntegers()`** function and traces the result to the Flash Output Panel.

```
var x:int = addNumbers( 5, 10 );
trace( x );  // displays 15
```

Returning Data from a Function

Some functions simply perform an action, but do not provide information back to the caller once the task is completed. A function can optionally return data to the caller.

For example, a function named `updateText()` may update a value but not provide any information back once the operation is complete.

Setting the return type to void

When a function does not return a value you, must define the function's return type as `void`. The `void` keyword alerts ActionScript that the return type should be ignored or voided. In the following example, the function increments the variable `score` but does not return a type.

```
var score:int = 0;

function updateScore( interval:int ):void
{
    score += interval;
}

updateScore( 1000 );
```

Using the return keyword

You can return any data type from a function call using the return keyword. In the `addNumbers()` function, it will return a type `int` to the caller. It also used a `return` statement to return the sum to the caller.

Note: Whenever a return statement is executed in a function, control returns immediately to the calling invocation.

Variable Scoping

Variables can be defined within a function, on the timeline, or within a class. When a variable is defined within a function, it is available only within that function and exists only during the execution of the function. Variables declared within the function have local or function-scope. Once the function completes that variable is removed from memory.

However, functions can access variables from the timeline or within the same class. Variables defined on the timeline exist in timeline-scope. Both variables in the timeline or in a class can be accessed and changed by the function and will continue to exist even after the function's execution.

In the following example, the **`firstName`** variable exists in timeline scope, and will exist before and after the function's execution. The **`numChars`** variable exists in the function scope and is removed from memory once the function completes execution.

```
var firstName:String = "Jason";

function getFirstNameLength():int
{
   var numChars:int = firstName.length;

   return numChars;
}
```

Walkthrough 2-3: Using Functions and Methods

In this walkthrough you will write a function that uses both function and timeline scope.

Steps

1. Return to wt2-2.fla in Flash.
2. Save the file as **FLAS3/walk/wt2-3/wt2-3.fla**.

Define a function

3. Open the Actions Panel (**F9**) with the **Actions** layer selected.
4. Declare a function that takes an argument named `score` of type `int` and returns the new `playerScore` as an `int`. Your code should appear as follows:

```
function incrementScore( score:int ):int {

}
```

5. Increment the variable `playerScore` by the `score` amount and return the value to the user. Your code should appear as follows:

```
function incrementScore( score:int ):int {
    playerScore = playerScore + score;
    return playerScore;
}
```

Running a Function

6. Call the newly declared function with `100` as the argument. Trace the output from the call.

```
trace( incrementScore( 100 ) );
```

7. Run the Flash file.

Using Compound Operator

8. Alter the `incrementScore` function to use a compound operator. Notice that this accomplishes the same task but uses less code.

```
playerScore += score;
```

9. Run the Flash file.

Function Scope

10. Alter the function definition so it defines a local variable named **`playerScore`**.

    ```
    function incrementScore( score:int ):int
       {
       var playerScore:int = score;
       return playerScore;
       }
    ```

11. Trace the value of **`playerScore`** and the output of the function.

    ```
    trace( playerScore );
    trace( incrementScore( 100 ) );
    trace( playerScore );
    ```

12. Run the Flash file. Notice that **`playerScore`** does not change when the function is called due to the variable being defined in function scope.
13. Save the document and keep it open for the next walkthrough.

- End of Walkthrough -

Unit Summary

- Develop code in the ActionScript Panel.
- Set your preferences for automatic code formatting.
- Comment your code using either // or /* */.
- Use the **`trace()`** method to output values for use in debugging.
- ActionScript is case-sensitive.
- Group statements that comprise a logical unit of work into a function.
- Functions, like variables, must be assigned a data type.

Unit Review

1. Do you output the contents of a variable for debugging purposes?

2. List the five basic data types supported by ActionScript

3. ActionScript is/is not case-sensitive.

4. How do you cast one variable type into another?

5. I should not comment my code to ensure job security. (True/False)

6. Is ActionScript 3 strongly typed, weakly typed, or typeless?

Lab 2: Instantiating Variables

In this lab you will instantiate some variables that will be used later in the application.

Steps

1. From the Flash Start Page, create a new **Flash File (ActionScript 3.0)**.
2. Save the file as **FLAS3/lab/lab2.fla**.
3. On the timeline rename `Layer 1` as `Content`.
4. Add a new layer above the Content layer and name it `Actions.`
5. Open the Actions Panel with the Actions Layer selected.
6. Declare 3 variables:
   - `firstName` of type `String`
   - `lastName` of type `String`
   - `timesCalled` of type `int`
7. Instantiate firstName to your first name, lastName to your last name, and timesCalled to 0. Your code should resemble the following:

```
var firstName:String = "[your first name]";
var lastName:String = "[your last name]";
var timesCalled:int = 0;
```

8. Create a new function named `outputName()` that returns `void` as follows:

```
function outputName():void
{

}
```

9. Within the function, increment `timesCalled` by 1 as follows:

```
timesCalled += 1;
```

10. Within the function use a `trace` statement to display the `firstName` and `lastName`. Also display the `timesCalled` variable so we know how many times this statement has been displayed to the user.

```
trace( firstName + " " + lastName + " has been displayed "
+ timesCalled + " times " );
```

11. Call the function three (3) times

```
outputName();
outputName();
outputName();
```

12. Run the Flash file.

13. Notice the function accesses and increments the global variable. Look at the + operator and how it concatenates the first and last name.