DigiPen
INSTITUTE OF TECHNOLOGY

CS 116 – Action Script Iterations

Elie Abi Chahine

Looping

Looping statements allow you to perform a specific block of code repeatedly using a series of values or variables.

I recommends that you always enclose the block of code in braces ({}). Although you can omit the braces if the block of code contains only one statement, this practice is not recommended for the same reason that it is not recommended for conditionals. If you later add a statement that you want to include in the block of code, but forget to add the necessary braces, the statement will not be executed as part of the loop.

While

The while loop form is:

```
while (expression)
{
    statements
}
following statement
```

•If ‘expression’ is true, the statements are executed until ‘expression’ becomes false.

•If ‘expression’ is false, the execution resumes at the following statement.

•The expression is evaluated before the statement is executed.
When the expression is false from the first time, the statement is never executed.

While

Example :

```
var i:int = 0;
while (i < 5)   /* Looping until I reaches the value 5 */
{
    trace(i);
    i++;
}
```

Note: Do not forget to increment the value of “i” or else you will end up with an infinite loop.

Do … While

The do while loop form is:

```
do
{
    statements
}
while(expression);
```

- The statements are executed at least once before the expression is checked
- If the expression is true, the statement is executed and the expression is evaluated,
- This will repeat until the statement becomes false.

Do … While

Eg:

```
var i:int = 5;
do                      /* The statements will be executed at least once */
{
    trace(i);
    i++;
} while (i < 5);

// output: 5
```

Note: **This is the only place where you can find the semicolon next to the while keyword**

for

The for loop general form is:

```
for(expression1; expression2; expression3)
{
    statements;
}
Following statements
```

- expression1: initializes the loop
- expression2: specifies the test made before each iteration.

- If expression2 is true, the statement is executed, then expression3 is executed
- Expression3 is evaluated after each iteration.

- The loop iterates until expression2 is false.
- If expression2 is false, the for loop will exit and resumes at “following statement”

for

Example:

```
var i:int;
for (i = 0; i < 5; i++)  /* will loop until I is equal to 5 */
{
    trace(i);    /* output should be from 0 to 4 */
 }
```

Note: **Any or all of the three expressions may be omitted, but the semicolon must remain.**

```
var i:int=0;
for (; i < 5;)  /* will loop until I is equal to 5 */
{
    trace(i);    /* output should be from 0 to 4 */
    i++;
 }
```

for

Note: The comma operator can be used to put multiple expressions within parts of the for statement.

```
var i:int;
var k:int;

for (i = 0, k = 10; i < 5; i++, k+=3)  /* will loop until i is equal to 5 */
{
    trace(i + k);
 }
```

Output: 10
14
18
22
26

Infinite loop

An infinite loop is obviously a loop that never ends.

Example :

```
while (1)
 {
     trace("I am an endless while loop");
}
```

```
for(  ;  ;  )
{
     trace("I am and endless for loop");
}
```

Infinite loop

How to get out from an endless loop? use the "**break"** keyword

```
var iCount:int = 0;
while(1)
{
    iCount++;
    if(iCount == 10)
    {
        trace (" I will break out from the loop now ");
        break;
    }
}
```

break

<u>Note:</u> Break will get out from the closest loop only

```
while(expression1)
{
    while(expression2)
    {
        .
        .
        .
        break;   /* breaking out from the while(expression2) loop */
    }
}
```

break

Example:

```
var day:int =0;
var hour:int = 0;

while( day<5 )
{
    while(1)
    {
        hour++;
        if( hour == 24 )
        {
            day++;
            hour = 0;
            break;      /* When this break is called, it will go out from the 2nd while loop's
                           scope but the 1st while  loop will continue until it's condition is
                           false */
        }
    }
}
```

continue

• The continue statement is similar to the break statement in that it causes the loop to deviate from its prescribed course. The difference is subtle, but very important.

Break statement

```
for (/* expressions */)
{
        /* first statement in loop */
        /* second statement in loop */
        /* etc... */
        break;

        /* last statement in loop */

}
[break jumps to here]
/* first statement after loop */
```

Continue statement

```
for (/* expressions */)
{
        /* first statement in loop */
        /* second statement in loop */
        /* etc... */
        continue;

        /* last statement in loop *
        [continue jumps to here]
}

/* first statement after loop */
```

continue

Using a for loop

```
var i:int = 1;
for (var i:int = 2; i <= 20; i++)
{

      if ( (i % 2) == 1 )
      {
        continue;
      }
      trace(i);
}
```

Using a while loop

```
while (i <= 20)
{
      i++;
      if ( (i % 2) == 1 )
      {
            continue;
      }
      trace(i);
}
```

Output:

2
4
6
8
10
12
14
16
18
20

for...in

The **for..in** loop form is:

```
for(var i in object)
{
    statements;
}
```

- The ***for..in*** loop iterates through the properties of an object (the elements of an array)
- For example, you can use a **for..in** loop to iterate through an array without worrying about the boundaries of the array (size)
- Still not sure what it's doing???? Need an example??? Go to the next slide…

for…in

Example:

```
var aMyArray:Array = [“Red” , “Green” , “Blue” ]
for(var i in aMyArray)
{
    trace(aMyArray[i]);
}
```

Output: Red
Green
Blue

- What happened is that **“i”** looped through the slots of the array having the values 0 then 1 then 2 *(positions to access in the array)*. It is exactly like:

```
var aMyArray:Array = [“Red” , “Green” , “Blue” ]
for(var i=0; i<3; i++)
{
    trace(aMyArray[i]);
}
```

for each … in

The **for each..in** loop general form is:

```
for each(var i in object)
{
    statements;
}
```

- The **"for each..in"** loop works like the **"for…in"** loop but the only difference is that **"i"** is equal to the actual content in the slot and not the position

- What did you just SAAAAAYYYYY !!!????!!!1 OK, lets do an example…

for each…in

Example:

```
var aMyArray:Array = [“Red” , “Green” , “Blue” ]
for(var i in aMyArray)
{
        trace(aMyArray[i]);
}

var aMyArray:Array = [“Red” , “Green” , “Blue” ]
for each(var i in aMyArray)
{
    trace(i);          /* Notice how we trace “i” instead of aMyArray[i] and we
                          get the same output */
}
```

Output: Red
Green
Blue

Iterations

Guides to writing good code:

- Also like in conditionals, the number one rule is going to be, **use braces.** Every time you write a for-loop, while-loop, or any kind of loop I want to see open and closed braces.

Example:

Bad Code:

```
while(i<10)
{
trace(i);
i=i+1;
}
```

```
while(i<10)
    trace(i);
    i=i+1;
```

Good Code:

```
while( i < 10 )
{
    trace(i);
    i = i+1;
}
```

Iterations

Guides to writing good code:

- Variables used in conventional ways can have very short names. The use of i and j for loop indices, is so frequent that there is little to be gained and perhaps some loss in longer names. Compare:

```
for (arrayIndex = 0; arrayIndex < numberOfElements; arrayIndex++)
{
        aElements[arrayIndex] = elementArray;
}
```

to

```
for (i = 0; i < iSize; i++)
{
      aElements[i] = i;
}
```

Iterations

Guides to writing good code:

- Indent to show structure.

An example of a badly formatted is:

```
for(n=0;n<100;aField[n++]=0);
i = 0; return ('\n');
```

Reformatting improves it a little bit:

```
for(n=0; n < 100; aField[n++] = 0)
    ;

i = 0;
return ('\n');
```

Iterations

Guides to writing good code:

Even better is to put the assignment in the body and separate the increment, so the loop takes a more conventional form and is thus easier to understand for both the reader and the programmer. A huge plus would be adding those braces to specify the for loop's scope:

```
for (n=0; n < 100; n++)
{
    aField[n] = 0;
}

i = 0;
return ('\n');
```

The End ☺