CISC 110 Week 7

Data Structures, Functions, For Loops
Today

- Data structures
- More on functions
- For loops

- Project info
- Test review
- USATS (volunteer)
Projects

- First part of project – Storyboard, due Nov 10 (2 weeks!) in class
  - Brainstorming
  - 3 Descriptions of possible projects
  - 1 detailed description of chosen project
- Phase 1/phase 2 code due Nov 16, 23 in lab
- Final Project due Dec 11, TBD
  - User manual

- More details will be posted online tomorrow
Data Structures

- Last week we saw how arrays are data structures
- *Data structures* are ways to organize data elements
- Common ways to structure data:
  - Lists
  - Tables
  - Records
  - Trees
Lists

- In ActionScript we have seen these as 1D arrays
  - They can store several items such as strings, integers, numbers, variables and even objects
    - You can even store another array!
      - A 2D array is a table
In ActionScript these are 2D arrays:

Physical Structure:

You can imagine the above table as an array holding 8 other arrays (columns) of length 4 (rows)

Tables can be used to represent game boards (like chess) or to store extra information (periodic table)
Records

- In ActionScript we see these as objects

Examples:

- Student record
  - Name
  - Address
  - Phone number

- MovieClip
  - Length
  - Width
  - Position
  - Alpha
  - Etc…
Trees

- In ActionScript we see these as objects and references to objects

- Examples:
  - Family tree
  - Hierarchy within a company
  - Nesting in MovieClips
For Loops

- We have already seen while loops:
  - Allow instructions to be repeated \textit{while} a certain condition is true

- For loops
  - Allow instructions to be repeated for a specified number of times (rather than relying on a true or false condition)
For Loops

// Repeat the set of instructions until counter reaches limit
for ( start counter; check counter; change counter )
{
    ... execute these lines of code ...
}

For Loops

**Example For Loop:**

```javascript
for (var i = 0; i < 5; i++) {
    trace(i);
}
```

**Initialization of Counter variable**

**Repetition continues if test is true**

**before each iteration**

**Change to counter after each iteration**

**Equivalent While Loop:**

```javascript
var i = 0;
while (i < 5) {
    trace(i);
    i ++;
}
```
For Loops

Example:  
for (var k = 1; k <= 5; k ++)  
{  
    trace( “!” );  
}  // end for

General Syntax:

for (<initialize counter> < test counter > <change counter>)  
{  
    < instructions >  // Repeat instructions until test is false  
}
Nested For Loops

for (var j = 1; j <= 2; j++)
{
    for (var k = 1; k <= 4; k++)
    {
        trace(j + " " + k);
    }
    // end inner for loop
}
// end outer for loop

Output:
1 1
1 2
1 3
1 4
2 1
2 2
2 3
2 4
Arrays and For Loops

- Can use for loops to iterate through arrays

```javascript
var list : Array = [ 33, 2, 91, 12, 8 ];

// Display array values
for( var i: int=0; i < list.length; i++ )
{
    trace( list[ i ] );
}
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>33</td>
<td>2</td>
<td>91</td>
<td>12</td>
<td>8</td>
</tr>
</tbody>
</table>
```
Local vs. Global Variables

Variable Scope means where variables exist, and can therefore be accessed, in a program.

Local Variables: Variables defined inside a function only exist within that function. They are created when the function starts executing and are destroyed when it finishes.

Global Variables: Variables defined outside of a function exist everywhere after they’re defined, except inside a function with a duplicate variable name.
Variable Scope

var myString: String = "Hello"; // Global Variable

public class Assign2 extends MovieClip {
    public function Assign2 ( ) { // Constructor Function
        printSomething( ); // Function Call
        printSomethingElse( ); // Function Call
    }

    function printSomething( ) { // Function Definition
        var localString: String = "World"; // Local Variable
        trace( myString + " " + localString);
    }

    function printSomethingElse( ) { // Function Definition
        var localString: String = "Everybody"; // Local Variable
        trace( myString + " " + localString );
    }
}
Defining New Functions

- **Purpose**: to create a block of code that we can reuse and refer to later in our program without having re-write code
Defining New Functions

Purpose: To define and name a new operation

Example:

```javascript
function printMessage ( )
{
    trace( "Next turn: Rupa" );
}
```

General Syntax:

```javascript
function <new function name> ( )
{
    <function instructions>
}
```
Defining New Functions - Example

// Function Calls (inside some other function)

printMessage();

printMessage();

// Function Definition

function printMessage()
{
    trace("Next turn: Rupa");
}

Output Window:  
Next turn: Rupa
Next turn: Rupa
Functions with Parameters

- We have already seen some ActionScript functions with parameters:
  - Math.round(3.14159)
  - Splice(3, 0, 44)
  - Etc...

- We can define our own functions with parameters
  - Allows information to be passed to our functions
  - Allows options when using functions
    - Makes them more generic/usable
Defining Functions with Parameters

Purpose: To allow options when using function

Example:

```javascript
function printMessage ( player: String )
{
    trace( "Next turn: " + player );
}
```

General Syntax:

```javascript
function <new function name> ( <parameter list> )
{
    <function instructions>
}
```
Defining Functions with Parameters - Example

// Function Calls (inside some other function):
printMessage( "Rupa" );
printMessage( "Ahmad" );

// Function Definition
function printMessage ( player: String )
{
    trace( "Next turn: “ + player );
}

Output Window:  
Next turn: Rupa
Next turn: Ahmad
How Parameters Work

Function `printMessage` before it’s used (parameter has no value yet)

Use of `printMessage` (i.e., a call to the function) gives value to parameter
One last thing about functions

- Functions can “store” or *return* results
- Allows your function to calculate a value which can then be used and assigned to variables in your program
- For example:
  - The `Math.random()` function *returns* some random number
  - We can also write our own functions to return some value
Defining Functions with Results

Purpose: To allow operations that calculate a value

Example:

function updateScore ( ) : int

{ return firstPlayerScore + 10;
}

General Syntax:

function <new function name> ( ) : <return Type>

{ <function instructions>

}
Defining Functions with Results - Example

```javascript
var player1Score = 50; // Global Variable

public function Assign2 () {
    player1Score = updateScore();
    trace("First Player’s Score: " + player1Score);
}

// Function Definition

function updateScore () : int {
    return player1Score + 10;
}
```

Output Window: First Player’s Score: 60
Defining Functions with Parameters AND results

Purpose: To define and name a new operation that allows options and that calculates a value

Example:

```plaintext
function updateScore ( score: int, amount: int ) : int
{
    return score + amount;
}
```

General Syntax:

```plaintext
function <new function name> ( <parameter list> ) : <return Type>
{
    <function instructions>
}
```
Functions with Parameters and Results - Example

```javascript
var player1Score = 50;  var player2Score = 50;  // Global Variables
public function Assign2 ( )
{
  player1Score = updateScore( player1Score, 10 );
  player2Score = updateScore( player2Score, 25 );
  trace( "First Player’s Score: " + player1Score );
  trace( "Second player’s Score: " + player2Score );
}

function updateScore ( score: int, amount: int ) : int
{
  return score + amount;
}
```

Output Window:  
First Player’s Score: 60  
Second Player’s Score: 75
public class Assign2 extends MovieClip {

    var sum: Number = 50;  // Global Variable

    public function Assign2 ( )
    {
        trace( "Global sum: " + sum );
        trace( "Scope of sum in function: " + addNumbers(10,20 ) );
        trace( "Notice global sum has not changed: " + sum );
    }

    function addNumbers( num1: Number, num2: Number): Number
    {
        var sum: Number = num1 + num2;  // Local Variable
        return sum;
    }

    Output Window:
    Global sum: 50
    Scope of sum in function: 30
    Notice global sum has not changed: 50
Scope Example

```javascript
var player1Score = 50; // Global Variable

public function Assign2 ( )
{
    player1Score = updateScore( player1Score, 10 );
    trace( "First Player’s Score: “ + score );  // ERROR
}

function updateScore ( score: int, amount: int ) : int
{
    trace( "Initial Score:” + player1Score );
    return score + amount;
}
```

Compiler Error: Access of undefined property score
Test 2 Material Ends Here
Test 2 Info

- **When:** Tuesday Nov 3, 6:30-7:30pm
- **Where:** Here
- **What:** Everything up to and including today’s slides (test 1 + if/else statements, loops (for/while), logical operators, arrays)
- Programming in ActionScript only, nothing on animation
- Study Custom Notes, PowerPoint Slides, Exercises, Labs and Assignments
- Practice has been posted online, you will also be doing practice problems in your lab next week
Question 1

Write a while loop that rolls a pair of dice \( n \) times and counts how many of the pairs are two of a kind, i.e., the same value. For example, if when the dice are rolled, the two dice are a 5 and a 5, that’s two of a kind; if the two dice rolled are a 5 and a 3, that is not two of a kind.

Display the results in the output window via a trace statement. Assume that \( n \) is an integer variable that has already been given a positive value, which could be any positive integer.
Assume you are implementing a poker game that includes a tutorial section on how to analyze the value of poker hands and their likelihood of occurring. Part of the implementation of that tutorial will require writing a while loop to calculate the value $n!$ ($n$ factorial). The formula for $n!$ is:

$$0! = 1$$

$$n! = (n)(n-1)(n-2)(n-3) \ldots (3)(2)(1)$$

For example, $4! = (4)(3)(2)(1) = 24$, so if $n = 4$, your while loop would calculate 24.

Assume that an int variable, $n$, already has a value and is greater than or equal to 0.
Question 3

Create an array filled with 5 words.

Write a for loop that goes through your array and displays each word at odd numbered indices (starting from 1) in the output window. Do this first using if/else statements and then without using if/else statements.