Basic Programming Concepts w/Ian
Take from the now discontinued Director Course

Please note that all code examples herein are written in Lingo.

The fundamental concepts remain the same, but the syntax of ActionScript, C#, etc, are different.

Operations and Precedence

When working with multiple operations (be them logical, arithmetic, etc), a certain order of events must be defined. This order of operations is known as ***precedence***.

For example:

```
	trace 10 + 5
	-- 15
	trace 5 * 2
	-- 10
But:
	trace 10 + 5 * 2
```

What order does the system carry out the math? Does it add 10 and 5 (15) then multiply by 2 (total 30)? Or is it calculated, 5 multiplied by 2 (10) plus 10 (result 20)?

If you try the above example in the Message Window you should receive an answer of 20. Why? Multiplication and division operations take precedence, or are calculated first, followed by addition and subtraction.

What if you want to specify a specific precedence? The use of parenthesis takes care of this. Anything inside of parenthesis will be evaluated first.

For example:

```
	trace (10 + 5) * 2
	-- 30
```

In this case (10 + 5) is evaluated, which then became (15) * 2, which equals 30.

The use of multiple brackets allows for ever increasing levels of complexity in precedence:

```
	a = (((5+5)-(3*2)) + 2) * 2
```

The order would break down as:

- `(5+5) = 10`
- `(3*2) = 6`
- `(10-6) = 4`
- `(4 + 2) = 6`
- `6 * 2 = 12`

Note:

Because of this precedence variables can be assigned in a manner that may appear strange. The follow is perfectly valid:

a = a + 5

This just means 'take whatever *a* happens to be and add 5 to it, then make *a* equal to the new amount'

Example:

```
a = 5
a = a + 5
trace a
```

The result is: -- 10

This can be useful for many different situations. One such situation might be creating controls to move a character left and right on the screen. What the button actually does it to add or subtract a certain amount from the current position.

An irritating limitation of Lingo is the lack of an increment function. Many languages allow you to increase or decrease a variables value by one using a syntax similar to:

```
a++
```

whereas with Lingo it would be written:

```
a = a + 1
```

Precedence with Logic

This also works with comparisons logic:

```
if ((loginName = "yourName") AND (loginPass = "yourPasswd")) then
      _move.go("Welcome") -- go here if the statement is TRUE
else
      _move.go("Denied") -- go here if the statement is FALSE
end
```

Assuming the variable loginName = "yourName"
and loginPass = "yourPasswd", the IF statement would break down as follows:

- if ((loginName = "yourName") AND (loginPass = "yourPasswd")) then
- if ((TRUE) AND (TRUE)) then
- if (TRUE)
- → `_move.go("Welcome")`

If the loginName was correct, but the loginPass was not, it would break down like this:

- if ((loginName = "yourName") AND (loginPass = "yourPasswd")) then
- if ((TRUE) AND (**FALSE**)) then
- if (FALSE) then
- → `_move.go("Denied")`

Remember: in a (Boolean) logical AND operation both variableA and variableB must be TRUE for the test to return TRUE. See the ***truth table*** in last weeks' handout.

Note: in some instances, such as the IF statement above, parenthesis are not required. Ie, I could write:

```
      If loginName = "yourName" AND loginPass="loginPass" then
```

However by using parenthesis you are specifically defining the order in which the operation will be carried out. This can prevent procedural errors from creeping in should you assume an operation will be carried out in one way, while Director actually carries it out in another.

Repeat With & Repeat While Loops

Sometimes you may want the program to keep doing something until either a certain number of iterations have completed or a particular condition occurs. These are examples of loops.

A quick and easy example:

```
on countTo10
     repeat with i = 1 to 10
          trace i
     end repeat
end
```

This loop will *repeat with* **i** counting up from 1 through to 10.
Why ***i*** as the variable name? Historically **i** has been used in loops as a short form for 'increment' or 'incremental'. Though you can, and often should, use a more descriptive variable name.

A *repeat while* loop works similarly, except it keeps repeating until a certain condition is **true**. For example:

```
on repeatTo10
     i = 1
     repeat while i < 10
          put i
          i = i + 1
     end repeat
end
```

This time the loop continues until **i** is no longer less than 10.

Warning:
1) While the application is executing a loop it is unable to perform ***any*** other operations.
2) Repeat While loops can easily lead to an ***infinite loop*** condition if not programmed correctly. In an infinite loop the exit condition never occurs, meaning the computer is stuck inside the loop indefinitely. Depending what operations you are instructing the computer to do inside the loop, this can make the application appear to freeze or, in severe cases, could potentially crash the computer. While authoring you can usually escape an infinite loop by pressing the **ESC** key or **CTRL** and **.** (control & period) - this tells the program to stop running.

Variations on these Repeat structures exist. Please reference your textbooks on *Loops*.