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.

Variables, Data Types, & Arithmetic Operators

Variables act as a type of container in memory that you can store data in. This information could be anything – a users name, age, file names, movie start times, etc. All but the most basic of programs will require the use of variables.

It is important to note that a variable can only contain one piece of information at a time

In terms of variables, computers classify information (or data) into different data types. Depending on the data type, Director will handle the information differently. Some of the data types you will encounter include: ***Integers, Floating Points, and Strings.***
Note: different languages / environments will have more or less data types. All use these basic types in one form or other.

Integers are whole numbers - numbers without fractions. For example, 42 is an integer, while 42.2 is not. 13 is an integer, 13 ½ is not.

Floating Points are numbers with decimals. 3.14159 is an example of a floating point number. So is 42.2.
Note that a number with a 0 decimal place is still a floating point. Ie, 42.0 is a floating point, not an integer. The ***floating point accuracy*** *determines the number of decimal places displayed and can be programmatically set.*

To prevent confusion when dealing with integers and floating points, Director will always display decimal places if it understands a number to be a floating point.
For example:

- If the number is written as `42` Director considers it an integer.
- If the number is written as `42.0` (or `42.000`, etc) Director considers it a floating point.

Strings are some form of text, words, or characters that Director treats as words rather than as variable names or commands. Generally used with alpha characters, words, or series of words (you could have pages worth of text in one variable). In order for Director to differentiate a string from a command the string must be surrounded by quotation marks. Strings can contain numbers in addition to letters or special characters, but you cannot perform mathematical operations with strings.

For example,

- **`"Hello World"`** is a string. `"A"` is a string. `"This is yet another examples of a string"` also qualifies.

- `"Halt"` is a string, But if you wrote **`Halt`** (no quotes) Director would assume you meant the *command Halt*, and wanted the program to terminate.

When you give a variable a specific value you are ***assigning*** that value to the variable. To assign the string "Arthur" to the variable theName you would write:

```
theName = "Arthur"
```

The variable name is written first, followed by a '=' sign, then the value to be assigned. In this context the "=" sign means assign, as in 'place the following value into the variable'.

To assign the integer 42 to the variable deepThought you would write:

```
deepThought = 42
```

Notice in this instance there are no quotation marks surrounding the *integer*. Remember, only *strings* use the quotation marks. Integers and floating points do not.

If you assign a variable a numeric value, but place quotes around the number, it will be a string and won't be treated as a number. This means you can no longer do mathematical operations with the number.

For example,

```
a = 10
b = 20
c = a + b
```

The value of **c** would be **30**.

However,

```
a = "10"
b = 20
c = a + b
```

In this case we are attempting to add a string and an integer. What will c end up equaling? That will depend on the programming environment.

With Lingo the result will be **`30.0000`** – a floating point. This occurs as Lingo automatically attempts to convert the string into a numeric value.

But if we did the same thing in Flash using ActionScript would we get the same result?

This situation can occur in languages that are *loosely typed* languages that do not force you to define what data type a variable will be. In these languages one can assign any value to any variable. In a *strongly typed* language a variables data type must be declared. The variable can then only contain data of that type, any operations such as the one in the above example are forbidden.
(Note: in Flash MX2004 it is an option to use strongly typed variables)

Arithmetic Operators

Arithmetic operators refer to the math functions available in a programming language. Some of the functions are:

+	Addition
-	Subtraction
*	Multiplication
/	Division

For example:

```
a = 3
b = 2
c = a * b
```

C would then equal **6**.

Many other operators exist, including modulus (mod), square root (sqrt), and arctangent (atan), please see your text index under *Math* for a complete list.

Note: if you divide two *integers* together, a fraction (decimal) is the result, Director will simply drop the fraction – ***it will not round***.

Example:

```
a = 10
b = 6
c = a / b
```

What is the value of c?

If they were floating points instead of integers C would equal 1.7. You may expect that 1.67 would round up to 2. With integer operations this is not the case.

Comparison Operators and Conditional Statements

At some point in your application you will be faced with the need to do one thing or another based on certain criteria. In fact all problem solving can be broken down until you reach a Yes or No answer – or as the computer considers things True or False, 1 or 0. This is called ***branching***, the application will branch off and follow one line of logic or the other.

For example, you have an on-line ordering form in which the user can specify the quantity of items they wish to order. You application will probably need to check that they have some quantity specified, and that the quantity is greater than 0.

To do this you need to use a ***comparison operator***.

Some of the possible comparisons you can use are:

=	are the two numbers equal or are the two strings the same?
<	is the 1st number less than the 2nd, or does the first string come after the second string alphabetically?
>	is the 1st number greater than the 2nd, or does the first string come before the second string alphabetically?
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

NOTE: these apply to Lingo and Director. Different languages use slight variations. E.g., using == for equals or != for not equal. The concept is the same in any language.

For example (**not syntactically correct**):

```
is 10 > 5?
Yes (TRUE)

is "Baby" < "Jack"?
No (FALSE)

is 10 <> 10?
No (FALSE)
```

This works with variables as well:

```
a = 10
is a = 10?
TRUE

a = 5
b = 10
is a < b?
FALSE
```

How do we conduct such a test programmatically? With the use of an ***IF THEN ELSE*** statement which works something like this:

```
If a = b then
     trace("Yes, the numbers match")
else
     trace("No, the numbers do not match")
end if
```

In this case of:

```
a = 5
b = 5
```

then the message 'Yes, the numbers match' would appear in the Message Window.

But if **a** and **b** had these values:

```
a = 5
b = 10
```

then the message 'No, the numbers do not match' would appear instead.

Thus it works like:

```
if valueA (condition) valueB is TRUE
     Do this
Otherwise it's FALSE, so:
     Do something else
End if
```

Using our on-line shopping example we want to know if the quantity is greater than 0, if it's not tell the user they must specify a quantity. If it is, proceed with the purchase:

```
if productQuantity > 0 then
      -- everything is okay, call my custom handler
      proceedToCheckout()
else
      -- an invalid quantity was specified, pop up a warning box
      alert("You must specify a quantity first!")
end if
```

Why use *`productQuantity`* **`>`** *`0`* instead of *`productQuantity`* **`<>`** *`0`* ?

Booleans and Logical Operators

As we've seen computers like 1's and 0's, TRUE and FALSE. Any time we conduct an **IF THEN ELSE** test we get a result that is either TRUE or FALSE.

A Boolean variable is a type of variable that can only be TRUE or FALSE, 1 or 0. They are very useful for keeping track of states. For example,

```
if myFavoriteShowIsOn = TRUE then
      ignorePhone()
else
      answerPhone()
end if
```

In this case depending if we'd how we set `myFavoriteShowIsOn` we would either answer or ignore the phone.

Boolean variables used in this manner are often called ***Flags***.

What if we have a situation where we need to test more than one condition simultaneously? Consider this scenario:

```
on goOutside
  if itsCold = TRUE then
      putOnJacket()
      walkOutDoor()
  else
      walkOutDoor()
  end if
end
```

This takes into account one condition, is it cold or not. But what if you wanted to check if it was raining as well as if it was cold?

Boolean logical operators allow us to consider this scenario. The logical operators are AND, OR, XOR*, and NOT. *(* Lingo does not specifically support XOR)*

Used in our scenario above:

```
On goOutSide
  If (itsCold = TRUE) OR (itsRaining = TRUE) then
      putOnJacket()
      walkOutDoor()
  Else
    walkOutDoor()
  End if
End
```

Now if it is cold outside, or if it is raining, we put on the jacket before going outside.

.
A ***truth table*** shows us the possible outcomes of a Boolean operation:

AND

Variable A	**Variable B**	**Result of Variable AND Variable B**
False	False	False
False	True	False
True	False	False
True	True	True

With AND both Variable A *and* Variable B must be TRUE for the result to be TRUE.

OR

Variable A	**Variable B**	**Result of Variable A OR Variable B**
False	False	False
False	True	True
True	False	True
True	True	True

With OR, either Variable A *or* Variable B must be TRUE for the result to be TRUE.

XOR (Exclusive OR)

Variable A	**Variable B**	**Result of Variable A XOR Variable B**
False	False	False
False	True	True
True	False	True
True	True	False

XOR will return TRUE if *only one* variable is TRUE.

NOT

Variable A	**NOT Variable A**
False	True
True	False

NOT returns the opposite value from the one provided. NOT TRUE is FALSE.
These operations can be compounded to create more complex test. Consider this scenario:

You have been invited to a party and would like to go – but only if your significant other can come and it's held on a Saturday night…. and as long as they don't play any country music.

Written programmatically you might have something like this:

```
If (partyDay = "Saturday") AND (canBringOther = TRUE) AND \
(countryMusic = FALSE) then
  _movie.go("Party") -- Would jump to a marker called "Party"
Else
  _movie.go("Home") -- Would jump to a marker called "Home"
end if
```

Note: the \ and the end of the first line means that the code has been split and continues onto a second line. Using this command lets you write one operation across multiple lines. This in not an issue in Director itself as you can scroll across the code horizontally. However if you wish to print the code out the print process will cause line wrapping to occur – splitting one line of code into two (without a visual indicator to warn you of this).

Variable Scope

Scope refers to where something can be seen or used. If something is in or within scope it is usable, if it is out of scope it is unusable.

Variables can have one of two scopes: *local* or *global*.
*** NOTE: Director allows for two scopes, other languages support more.**

Local variables: can only be declared and used within a single event handler. When the handler terminates the variable and its value are erased.

Global variables: once declared, can be used from any handler throughout the entire program or even across multiple Director programs. Global variables are *persistent*, and will not lose their value until specifically changed or the program ends.

By default Director assumes all variables are local unless they are specifically declared as being global.

This example demonstrates a local and global variable. This would be written as a Movie Script:

```
global gGlobalVar

on setVars
     localVar = "This is a local variable"
     gGlobalVar = "This is a global variable"
end


on showVars
     trace localVar
     trace gGlobalVar
end
```

In the Message window call the setVars handler by typing *setVars()*.This assigns values to our variables.

Then type *showVars()* to call the second handler. The global variable will still contain the string "`This is a global variable`" whereas the local variable is VOID meaning it has no value – there is no such thing as localVar outside the setVars handler.