# Basic Lingo Scripting

*Lingo* is Director's scripting language, and allows you to add interactivity and intelligence to your presentations. We've already seen some simple Lingo with the use of the "go" command. Lingo commands are contained in cast members called *scripts*, which may be of several types, and are organized into *handlers*, individual segments of code which respond to specific *events* or messages during playback. The four basic kinds of scripts are:

- Movie script - global handlers executed from any script in the movie
- Sprite script - handlers executed for the specific sprite to which they are attached
- Frame script - handlers executed for the specific frame in which they reside
- Cast script - handlers executed for the cast member to which they are attached

(A fifth type, parent script, is beyond the scope of this course.) Sprite and Frame scripts are also called "behaviors". Most of your scripts will be either Frame or Sprite scripts/behaviors. The use of Cast scripts is generally not recommended, since this makes the cast members less flexible to use. More sophisticated scripting will occur in Movie scripts, where you create handlers to perform global tasks used by many different handlers throughout your movie.

Below is a list of some of the basic and most common Lingo commands and statements. In some cases, part of the command is optional - the optional part is shown in [brackets].

---

## go

This command will send the playback head in the score to …

- a different frame location in a movie
- a different marker location in a movie
- a different movie
- a different frame location found in another movie
- a different marker location found in another movie

Examples of how you would use this command…

| | |
|---|---|
| `go [to] frame 7` | This will send the playback head to frame number 7. |
| `go [to] the frame` | Causes the playback head to loop continuously on a single frame. |
| `go [to] the frame - 2` | Causes the playback head to loop continuously between the frame this script is called from and a location 2 frames earlier. |
| `go [to] [frame] "A"` | Sends the playback head to the marker labeled "A". |
| `go [to] movie "blue"` | Causes Director to jump to the movie "blue". |
| `go [to] frame 7 of movie "blue"` | Jumps to frame 7 of the movie "blue". |
| `go [to] [frame] "A" of movie "blue"` | Jumps to marker "A" of the movie "blue". |
| `go previous` | Jumps to the previous marker in the Score. |
| `go next` | Jumps to the next marker in the Score. |
| `go loop` | Jumps to the nearest marker to the left of the playback head in the Score. If the playback head is already located at a marker, it will remain on that frame. |

---

## go *(continued)*

Examples of how you would see this command used in scripts...

```
on exitFrame
   go to frame 7
end

on exitFrame
   go the frame          --Note that the use of “to” is optional
end

on exitFrame
   go to the frame - 7
end

on mouseDown
   go to frame "A"
end

on mouseUp
   go to frame "A" of movie "blue"
end
```

## beep

This command will cause your computer to beep using the standard System beep.

Examples of how you would use this command …

```
beep                  Causes the computer to beep one time.

beep 3                These cause the computer to beep three times.
beep (3)
```

Examples of how you would see this command used…

```
on exitFrame
   beep
end

on exitFrame
   beep 3
end

on mouseUp
   beep (3)
end
```

## quit

This command causes your Director movie to quit. It will also quit out of the Director application, if you're in Authoring mode. If the Director source file has been turned into a Projector and a "quit" command is executed, the Projector will quit.

Examples of how you would use this command …

| | |
|---|---|
| `quit` | This causes the Director movie and application to quit, or the Director Projector to quit. |

Examples of how you would see this command used…

```
on exitFrame
   quit
end

on mouseUp
   quit
end
```

## halt

Similar to the "quit" command , this command will cause your Director movie to stop, but it will not quit the Director application. If the Director file has been turned into a Projector and a "halt" has been executed, the Projector will quit. This command is preferable over the "quit" command during development.

Examples of how you would use this command …

| | |
|---|---|
| `halt` | Causes the Director movie to stop playing ,or the Director projector to quit. |

Examples of how you would see this command used…

```
on exitFrame
   halt
end

on mouseUp
   halt
end
```

## --

This is referred to as a "comment". You use this when you want to make notes in the Lingo code you are writing - any text appearing after the two hyphens until the end of the line will be ignored by Director and not executed. This command can also be used to "comment-out" a line of code. If a line of code has been "commented-out," it will not execute.

Examples of how you would use this command …

`--note to myself` – To insert a comment in your code about something you want to remember.

`--beep` – Here, the comment characters are used to disable the command to "beep" in this instance.

`--quit` – Here, the comment characters are used to disable the command to "quit" in this instance.

Examples of how you would see comments used…

```
on exitFrame
   --loop on a frame
   go to the frame
end

on mouseUp
   --if this button is pressed then quit
   quit
end

--This routine would go to frame 7 and then end.
--The "quit" command will not execute because it is commented-out.
on mouseUp
   go to frame 7
   --quit
end
```

## on mouseUp

This statement begins a handler that is executed when the user has …

1.) clicked on a cast member on the Stage and
2.) the mouse button has been released while the cursor is over that cast member.

Such a handler is typically attached to a sprite in the Score (referred to as a "Sprite Script" or "Behavior"), or to a cast member in the Cast (referred to as a "Cast Script").

Select a sprite in the Score, and click on the script bar. Or, select a cast member from the Cast window by clicking on it with the keys Control and Option (ALT on Windows machines) held down. A Script window will open. When you first open the Script window to attach a script to a sprite or a cast member, the default text below is automatically placed in the window before you enter your code. The flashing "I" beam cursor will appear in between these two lines of code indicating the area where you can begin entering your commands.

```
on mouseUp

end
```

Example of how you would see this statement used…

```
on mouseUp
   go to frame 7
   beep
end
```

## on mouseDown

This statement begins a handler that is executed when the user has clicked down on a cast member on the Stage with the mouse.

Such a handler is typically attached to a sprite in the Score (referred to as a “Sprite Script” or “Behavior”), or to a cast member in the Cast (referred to as a “Cast Script”).

Select a sprite in the Score and click on the script bar. Or, select a cast member from the Cast window by clicking on it with the keys Control and Option (ALT on Windows machines) held down. A Script window will open: "on mouseUp" is inserted automatically by default, so you need to change the "mouseUp" to "mouseDown". A flashing "I" beam cursor will appear in between these two lines of code indicating the area where you can begin entering your commands.

```
on mouseDown

end
```

Example of how you would see this statement used…

```
on mouseDown
   go to frame 7
   beep
end
```

## on mouseEnter

This statement begins a handler that is executed when the user's cursor enters the area of a sprite. If the sprite has any ink effect other than "Matte", the bounding box of the sprite is the triggering area; if the "Matte" ink is assigned to the sprite, then the actual visible area of the sprite is the triggering area.

Such a handler is typically attached to a sprite in the Score (referred to as a "Sprite Script" or "Behavior"), or to a cast member in the Cast (referred to as a "Cast Script").

Example of how you would see this statement used…

```
on mouseEnter
   --Assign the "rollover" graphic for this sprite.
   --Assumes it immediately follows the "normal" graphic
   --in the Cast.

   set the memberNum of sprite (the currentSpriteNum) = ¬
           the memberNum of sprite (the currentSpriteNum) + 1
end
```

(Note the use of the continuation character "¬", which continues a logical line onto two or more physical lines. You enter it by typing Option-Return. Lines continued with this character are not auto-indented - you must add spaces yourself for formatting.)

## on mouseLeave

This statement begins a *handler* that is executed when the user's cursor leaves the area of a sprite. If the sprite has any ink effect other than "Matte", the bounding box of the sprite is the triggering area; if the "Matte" ink is assigned to the sprite, then the actual visible area of the sprite is the triggering area.

Such a handler is typically attached to a sprite in the Score (referred to as a "Sprite Script" or "Behavior"), or to a cast member in the Cast (referred to as a "Cast Script").

Example of how you would see this statement used…

```
on mouseLeave
   --Restore the "normal" graphic for this sprite.
   --Assumes it immediately precedes the "rollover" graphic
   --in the Cast that is displayed using an "on mouseEnter"
   --handler while the cursor is over the sprite.

   set the memberNum of sprite (the currentSpriteNum) = ¬
           the memberNum of sprite (the currentSpriteNum) - 1
end
```

(Note the use of the continuation character "¬", which continues a logical line onto two or more physical lines. You enter it by typing Option-Return. Lines continued with this character are not auto-indented - you must add spaces yourself for formatting.)

## on enterFrame

This statement begins a handler that is executed whenever the playback head in the Score enters a frame.

Such a handler generally appears in a script found in the Script channel of the Score (known as a “Frame Script,” a type of Score script).

Select a cell in the Script channel of the Score and either double-click it, or click on the script button. A script window will open. When you first open a Script window in this manner, "on exitFrame" is inserted by default, so you need to change the "exitFrame" to "enterFrame". Then enter your Lingo code between these two lines.

```
on enterFrame

end
```

Examples of how you would see this statement used…

```
on enterFrame
   beep
end
```

## on exitFrame

This statement begins a handler that is executed whenever the playback head in the Score exits a frame.

Such a handler generally appears in a script found in the Script channel of the Score (known as a “Frame Script,” a type of Score script).

Select a cell in the Script channel of the Score and either double-click it, or click on the script button. A script window will open. When you first open a Script window in this manner, "on exitFrame" is inserted by default. A flashing "I" beam will appear in between these two lines indicating the area where you can enter your Lingo code.

```
on exitFrame

end
```

Examples of how you would see this statement used…

```
on exitFrame
   --now let's go to marker "A"
   go to "A"
end
```

## on keyUp

This statement begins a handler that is executed when any key on the keyboard is pressed, then released.

Such a handler is typically attached to a sprite in the Score containing a field cast member, or to a field cast member in the Cast. If there are no field sprites in the frame, and you wish to use "on keyUp" to merely trigger an action without actually entering text, you may place it in the frame script for the frame, or in a global movie script.

```
on keyUp

end
```

Example of how you would see this statement used…

```
on keyUp
   beep
end

on keyUp
   if the key = "X" then quit    --pressing the "X" key will quit
end

on keyUp
   --pressing the "1" key will go to a frame labeled "First"
   if the key = "1" then go frame "First"

   --pressing the "2" key will go to a frame labeled "Second"
   if the key = "2" then go frame "Second"
end
```

## on keyDown

This statement begins a handler that is executed when any key on the keyboard is pressed down.

Such a handler is typically attached to a sprite in the Score containing a field cast member, or to a field cast member in the Cast. If there are no field sprites in the frame, and you wish to use "on keyDown" to merely trigger an action without actually entering text, you may place it in the frame script for the frame, or in a global movie script.

Note that the effect of "on keyDown" is different from "on keyUp" - the former responds as soon as the key is pressed, while the latter responds only when the key is released.

```
on keyDown

end
```

Example of how you would see this statement used…

```
on keyDown
   beep
end

on keyDown
   if the key = "1" then
         alert "Right answer!"
   else
         alert "Sorry - try again."
      end if
end
```

## the colorDepth

This statement will ...

1.) tell you the current color depth setting of the monitor, or
2.) let you set the color depth of the monitor, if possible.

This statement can be used to test or set the colorDepth of a Macintosh. Windows PCs can also use this statement to test the colorDepth status, but not all PCs support setting the colorDepth.

To change the monitor setting to 8-bit (256 colors) or another mode, use this statement to set the colorDepth. To find out the current setting of your monitor, you can test or "put" this information.

Different colorDepth settings:

| | |
|---|---|
| 1 | 2 colors |
| 2 | 4 colors |
| 4 | 16 colors |
| 8 | 256 colors |
| 16 | thousands of colors (32,768 colors) |
| 32 | millions of colors (16,777,216 colors) |

Examples of how you would use this statement…

| | |
|---|---|
| `put the colorDepth` | This script would put the current color depth. If you do not place this information into a variable, it will appear in the message window (Command - M opens the Message window). |
| `set the colorDepth to 8` | On Macintoshes and some PCs, this script changes the monitor setting to 256 colors (or 8 bit color). |
| `set the colorDepth to 16` | On Macintoshes and some PCs, this script changes the monitor setting to thousands of colors (or 16-bit color). |
| `set the colorDepth to 32` | On Macintoshes and some PCs, this script changes the monitor setting to millions of colors (or 32-bit color). |

Examples of how you would see this statement used…

```
on exitFrame
   set the colorDepth to 8
end

on mouseDown
   --look for the result of this script to appear
   --in the message window

   put the colorDepth
end

on mouseUp
   if the colorDepth <> 8 then
      set the colorDepth to 8
   end if
end
```

## cursor

This statement will set the cursor to a standard system cursor, or a cast member for a custom cursor.

If you are about to perform an operation that will take a few seconds or more to complete, like preloading a group of cast members, it's good practice to set the cursor to the wristwatch (hourglass on Windows). Or, to indicate that a button or on-screen area is "hot" (active), you might change the cursor to a pointing hand.

The built-in cursors:

| | |
|---|---|
| 0 | No custom cursor (returns cursor choice to the system) |
| -1 | Arrow cursor |
| 1 | Insertion (I-beam) cursor |
| 2 | Crosshair cursor |
| 3 | Thick cross (crossbar) cursor |
| 4 | Wristwatch (hourglass on Windows) cursor |
| 200 | Blank cursor (no cursor displayed) |

Examples of how you would use this statement with the built-in cursors…

| | |
|---|---|
| `cursor 4` | Sets the cursor to the wristwatch (hourglass on Windows). |
| `cursor -1` | Resets the cursor to the default (usually the arrow pointer). |

If you wish to set the cursor to a cast member of your own design, or from the Director cursor library (under the Xtras menu), you refer to it by its cast number or member name, enclosed in square brackets. Optionally, you may also specify a "mask" member, which affects the appearance of the cursor:

| | |
|---|---|
| `cursor [ 5 ]` | Sets the cursor to the cast member in Cast slot 5. (Note that the brackets here are required) |
| `cursor [ 5, 6 ]` | Sets the cursor to the cast member in Cast slot 5, with cast member 6 used as a mask. (Note that the brackets here are required) |

A cast member to be used as a custom cursor by this method must be a 1-bit graphic, and confined to a 16x16 pixel size. The registration point of the graphic will define the "hot" point of the cursor. A graphic larger than 16x16 pixels will display only the upper-left portion. Director 6.5 and later provides a "custom cursors" Xtra that allows colored and animated cursors. Its use is too complex for this document.

Examples of how you would see this statement used…

```
on startMovie
   cursor 4
   preloadMember "myfirstgraphic", "mylastgraphic"
   cursor -1
end

on mouseEnter
   --"pointing hand" must be a cast member in this movie
   cursor [ member "pointing hand" ]
 end
```

## preLoad
## preLoadMember
## preLoadMovie

These statements cause Director to load certain cast members into memory before proceeding, making them ready for playback without further delay. (The number of members actually read in will be limited by available memory.) It is useful for optimizing playback performance, especially for animation sequences, by putting the load time delay up front, or at controllable points in the playback. Note that there will still be a delay assocaited with any preloading; it's just that the delay occurs all at once rather than "on the fly" where it is harder to control and may adversely affect playback.

preLoad
preLoad *EndingFrameNumberOrMarker*
preLoad marker "*MarkerName*"
preLoad *StartingFrameNumberOrMarker*, *EndingFrameNumberOrMarker*

preloadMember *NameOrNumber*
preloadMember *FromNameOrNumber*, *ToNameOrNumber*

preloadMovie "*MovieName*"

Example of how you would see these statements used…

| | | |
|---|---|---|
| preLoad | -- | preloads all cast members in the movie (limited by RAM). |
| preLoad 120 | -- | preloads all cast members used in frames 1 to 120. |
| preLoad marker "Main" | -- | preloads all cast members up to the frame marked "Main" |
| preLoad 25, 50 | -- | preloads all cast members used in frames 25 to 50. |
| preLoad "IntroStart", "IntroEnd" | -- | preloads all cast members used in from the frame marked "IntroStart" through the frame marked"IntroEnd" |
| preLoadMember | -- | preloads all cast members in the movie (limited by RAM). |
| preLoadMember 2 | -- | preloads the cast member in Cast slot 2. |
| preLoadMember "BigBackground" | -- | preloads the cast member named "BigBackground". |
| preLoadMember 10, 50 | -- | preloads cast members in Cast slots 10 through 50. |
| preLoadMember "Intro.1", "Intro.25" | -- | preloads cast members between member "Intro.1" through member "Intro.25". |
| preLoadMovie "Intro" | -- | preloads all cast members in the movie "Intro". |

## unLoad
## unLoadMember
## unLoadMovie

These statements are the inverse of their corresponding "preLoad" commands, instructing Director to purge certain cast members from memory before proceeding

## on cuePassed

This statement begins a *handler* that is executed when a sound or sprite passes an internal cue point during the course of playback. Cue points may be inserted into audio clips using SoundEdit 16.

Such a handler usually appears in a script found in the Script channel of the Score, but may also be placed in a global movie script.

The optional parameters to this statement are assigned values by Director when the cue point is passed. You may examine them or use them to perform actions based on their values.

```
on cuePassed

end
```

or -

```
on cuePassed channelID, cuePointNumber, cuePointName

end
```

Example of how you would see this statement used…

```
on cuePassed
   --returns to the previous marker whenever any cue point
   --is encountered

   go loop
end

on cuePassed channelID, cuePointNumber, cuePointName
   if cuePointName = "Rewind" then go frame "Start"
end

--The following handler”listens” for cue points only
--on sound channel 2.
on cuePassed 2, cuePointNumber, cuePointName
   if cuePointName = "Skip" then go the frame + 1
end
```

## on startMovie

This statement begins a *handler* that is executed when the movie is ready to begin playing (after any preloading, but before any frames are displayed on the Stage).

Such a handler must appear in a global movie script. Check that the Script window reads "Movie Script". If not, use the "Info" dialog box to change its type to "Movie Script". It is used to perform general initialization or other tasks to be done whenever playback of the movie begins, such as setting the colorDepth or sound volume.

```
on startMovie

end
```

Examples of how you would see this statement used…

```
on startMovie
   set the colorDepth = 16    --16-bit color (thousands) playback
   set the soundLevel = 4     --4 out of 7 - a reasonable level
end
```

## on stopMovie

This statement begins a *handler* that is executed when the movie playback is halted.

Such a handler must appear in a global movie script. Check that the Script window reads "Movie Script". If not, use the "Info" dialog box to change its type to "Movie Script". It is used to perform general cleanup or other tasks to be done whenever playback of the movie finishes.

```
on stopMovie

end
```

Examples of how you would see this statement used…

```
on stopMovie
   alert ("Th-th-th-that's all folks!")
end
```

## play
## play done

This statement pair offers similar capability to the "go" command, except that it provides an automatic return feature. With the "go" command, playback jumps to a new location, with no memory of the point from which it was called. By contrast, when the "play" command is executed, Director jumps to the specified segment of the current movie, or to an entirely separate movie, continuing playback from there, but remembering the location it was called from. Then, when the segment or other movie finishes, and the "play done" statement is encountered, playback resumes at the original location where the "play" command was issued.

To avoid a "memory leak" resulting from wasted RAM, always be sure to issue a "play done" for every "play" command.

```
play frameNameOrNumber
play frame NameOrNumber
play member MemberNameOrNumber
play movie MovieName
play frame NameOrNumber of movie MovieName
```

To resume normal playback sequence:

```
play done
```

Examples of how you would see these statements used…

```
play frame "IntroSequence"
play "AttractLoop"
play movie "Credits"
play frame "AboutUs" of movie "Info"
```

And when finished:

```
on exitFrame
  play done
end
```

Or triggered by a mouse click:

```
on mouseUp
  play done
end
```