Flow Control Continued

```
fn deleteAllSkinModifiers mrgVis_GeoOnly=
(
	all_obj = #()
	max modify mode
	while all_obj.count > 0  do
		deleteItem  all_obj 1
	if mrgVis_GeoOnly then
	(
		max select all
	)
	else
	(
		select $*
	)
	for o in selection do
		if o.modifiers[#skin] != undefined then
			appendIfUnique  all_obj o
	for m in all_obj do
	(
		max modify mode
		if m.modifiers[#skin] != undefined then
		(
			m_Skin = m.modifiers[#skin]
			deleteModifier m m_Skin
		)
	)
)
```

Loops: for, for-in (for each), while, recursive?

Control Flow Statement

Loop statement executes a block of code repeatedly until an exit condition is met.

Loops allow us to write scripts using less instructions to process multiple objects multiple times.

Loop statement allow us to easily run repeated operation on data with unknown size.

Control Flow Statement

Consider opening, exporting and closing every file in a directory. Without loop statement, we would have to do the following:

Open fileOne
Export fileOne
Close fileOne
Open fileTwo
Export fileTwo
Close fileTwo
…
Open lastFile
Export lastFile
Close lastFile

But with a loop statement we can write a more efficient code as follow:

for every file in my directory do open, export and close the file

Control Flow Statement

We can categorize loop statements by the iteration-control factor with varying syntax within each programming languages. The following are 3 loop categories:

- Count-controlled loops like a “for loop”
- Condition-controlled loops like a “while loop”
- Collection-controlled loops like a “for each loop”

Control Flow Statement

Looking closer at loops statement will discover that all loops are constructed using the following parts:

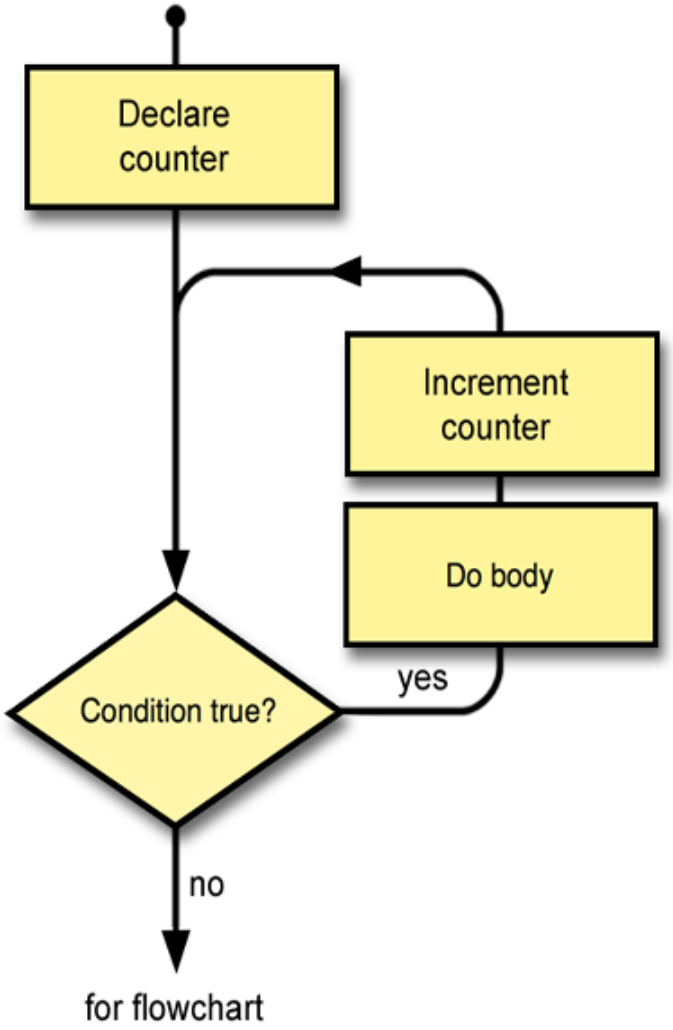
- Initial conditions that must be satisfied before starting execution of the loop.
- A pre or post test expression to evaluate continuing the loop.
- An iteration expression that brings the loop closer to termination.
- A body of action statements to be looped over.

Comparison & logical operators

Here are some comparison operators which may look different between scripting languages:

Operator/language	MEL	Maxscript	Python	Lua
Equal	= =	= =	= =	= =
Not equal	!=	!=	!=	~=
Greater than	>	>	>	>
Greater or equal	>=	>=	>=	>=
Less than	<	<	<	<
Less or equal	<=	<=	<=	<=
And	&&	and	and	and
Or	||	or	or	or
Not	!	not	not	not

For loop Statement

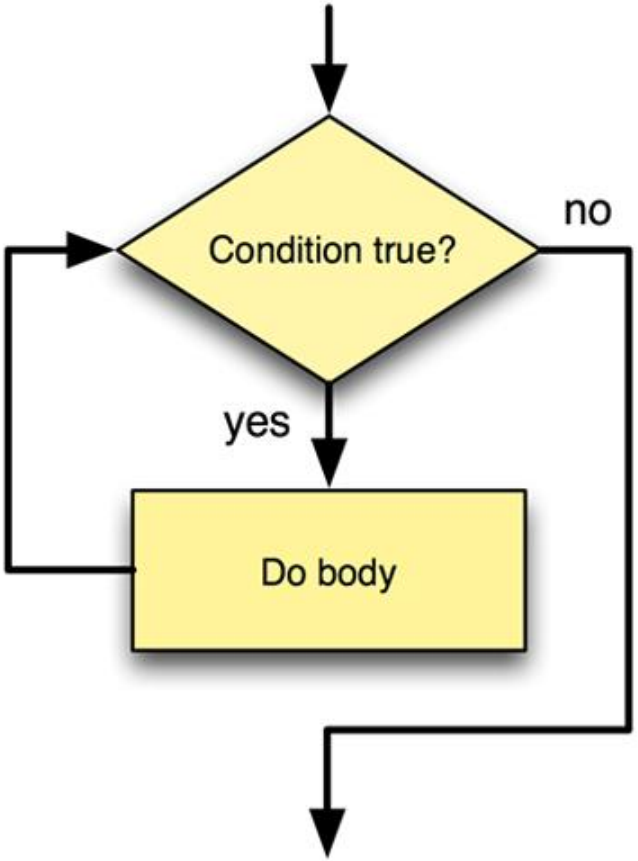


The “for statement” has two forms: one numeric and one generic.

The numeric for loop has a declared counter, a condition, a body and an arithmetic operator to increment or decrement the declared counter. It repeats a block of code until the condition tests false.

The generic “for statement” works over iterator function which is called to produce a new value, stopping when this new value is null.

While loop Statement



`while` flowchart

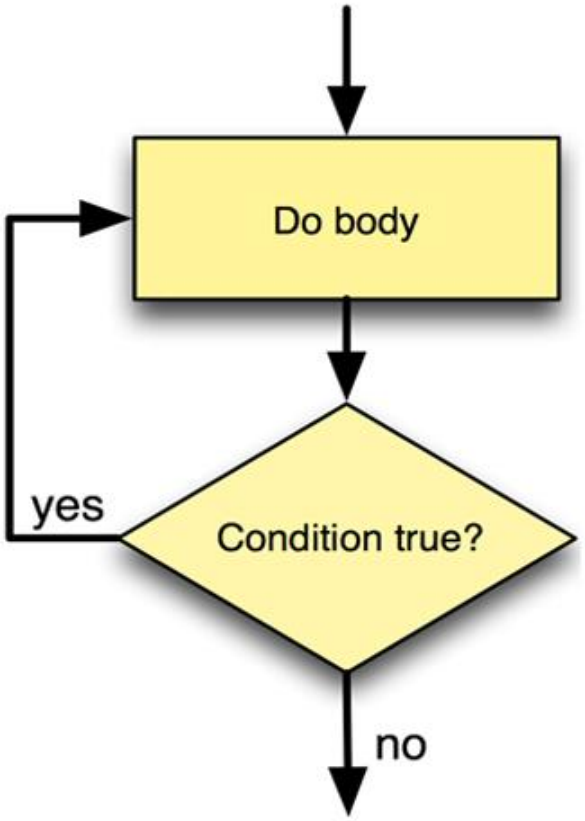
In while statement, we pre-test the controlling condition before we start running the body of the “while statement” and re-test the controlling condition on every iteration.

The “while loop” is best used to process an unknown number of objects and terminate the operation when a condition is no longer true.

Caution:

You may have an **infinite loop** and you would **not be able to interrupt** the script, **If the condition never evaluate to false**.

While loop Statement



do/while flowchart

In do-while statement, we post-test the controlling condition. The body of the "while statement" will execute at least once before the controlling condition is tested and continue re-testing the controlling condition on every iteration.

The "do-while loop" is best used in a situation where we need to execute our script at least once like in showing a window or requesting user feed-back.

Caution:

You may have an **infinite loop** and you would **not be able to interrupt** the script, **If the condition never evaluate to false**.

For Loop, Maya MEL Example:

```
/*
It is best to use a “for loop” statement, when you know how many time you need to repeat a process. Only integers can be reliably used in a count-controlled loop.
 */
int $n;
for($n=0; $n<100; $n++) {
 //int n = 0 is initializing the count, n < 100
//is the terminating condition and n++ (n = n + 1) is the iteration condition
    print (n + "\n");  // the body of the loop
}
```

For loop Maxscript Example:

```
/*
The standard “numbered loop” has a start value, a “to” value as a limiting factor and an optional “by” as an incrementing value. The allowable value types are integer, float and time. If the “by” value isn't given it defaults to 1.
 */
for i = 0 to 100 by 2 do
(
  print i
)

--below is a time sequence given as frames
for t in 0f to 100f by 5f do
(
  sliderTime=t
)
```

For loop Maxscript Example:

```
/*Maxscript has additional flavors of the “for loop” statements. If an optional “while test” is specified, the for loop will terminate if the test evaluates to false. The while test expression must evaluate to a boolean (true/false). */
Local notFound = true
for i = 1 to 10 while notFound do (
   if i== 5 do  (
      notFound = false
      format "found 5\n"
    )
   format "%\n" i
)
/*The optional “where” expression is evaluated at the beginning of the iteration and only executes the loop body if it evaluate to true. The “do” expression simply evaluates the body expression once for each value in the sequence. The collect phrase gathers and stores the expression values from the loop in an array, which is then returned as the overall for loop. */
big_ones =  for obj in $box*  where obj.height > 100 collect obj    -- collect the big boxes into an array
-- “$box*”  sequence pathname
--also the above “for loop” statement is used as a function that return a collection of  objects
```

For loop, Python Example:

```
#The range function creates a list containing numbers defined by
#the input.

for x in range(0,3):
    print 'We\'re on time %d' % (x)

#The xrange function creates a number generator. You will often see
#that xrange is used much more frequently than range.
#xrange(starting value, ending value, steps)
for x in xrange(1,11): #start from 1 to 11
        print '%d * %d = %d' % (x,x,x*x)
```

For loop, Lua Example:

```
-- All three control expressions are evaluated only once, before the
--loop starts. They must all result in numbers. v, limit, and step are
 --invisible variables outside the loop block.

for v = 0, 10, 2 do --0 is initial value, 10 is end value, 2 is incrementing steps
        if v == 6 then
                print "reached 6 and exiting"
                break
        end
        print ( v )
end
```

While Loop, Maya MEL Example:

```
/*
"While loop" checks the truthfulness of the initial condition at the start of each
iterations to continue executing the code body.
*/
string $list[] = `ls –sl`;
int $index = 0;
while($index < size($list))
{
    print ($list[$index] + "\n");
  // if you forget to increment $index, you'll have an infinite loop and
  // the only way to stop it is to exit Maya
    $index++;
}
```

While loop Maxscript Example:

```
/*
In while statement, we pre-test “t” value before we start running the code inside the while statement and re-test “t” value on every iteration.
 */

t = 10;

while t > 0 do
(
  format "t minus % seconds and counting.\n" t
  t = t - 1
)
```

While loop, Python Example:

```
#The break statement in Python terminates the current loop and resumes execution
#at the next statement outside the loop body.
#The continue statement in Python rejects all the remaining statements in the
#current iteration of the loop and moves the control back to the top of the loop.


var = 10                    # initial count
while var > 0:
    print 'Current variable value :', var
   var = var -1             #decrement var
   if var == 6:
      continue              #below skipped when var is 6
   if var == 5:
      break                 #loop will terminate when var is 5
```

While loop, Lua Example:

```
-- The Lua while loop acts pretty much like while loops in all languages,  it tests on top,
--and you can change all and any variables within it because the test gets re-tested on
--every iteration.

n = 1
while n  <= 15 do
        print (n)
        n = n + 1 --increment n to avoid infinite loop
end   -- notice end statement
```

Do-While Loop, Maya MEL Example:

```
/*
In a do while we post-test the condition, which means the code will run at least
once even though the condition was not met.
*/

do
{
    print ($list[$index] + "\n");
    // if you forget to increment $index, you'll have an infinite loop and
    // the only way to stop it is to exit Maya
    $index++;
} while($index < size($list))
```

Do-While loop Maxscript Example:

```
/*
In a do while we post-test the condition, which means the code will run at least once even though the condition was not met. Usually, do-while is used in UI menu selection since it has to show up first for the user select to continue or not.
 */

do
(
    format "You see me only when random is not 0.\n"
 ) while (random 0 1) != 0

--the above loop will continue running until the random function produce a zero
```

Do-While loop, Python Example:

```
#Although there is no do-while statement in python, a properly
#constructed while loop can have the same functionality


var = 1
while True: #the condition is true so execute body
    print (var)
    var = var + 1 #increment var
    if var>10:    #test var to not exceed 10
        break #terminate the loop
```

Repeat loop, Lua Example: (like do-while)

```
--Lua does not have "do-while" but it uses "repeat-until"  to produce the same results
-- In the repeat–until loop, the inner block does not end at the until keyword, but only
--after the condition. So, the condition can refer to local variables declared inside the
--loop block

k = 1
repeat
    print (k)
    k = k + 1 --increment n to avoid infinite loop
until k > 5
```