Flow Control

```
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, Conditions, Subroutines, Jumps, Interrupts, Exit

Control Flow Statement

In general, the flow of code instructions is executed in a top-down order. **Control Flow Statements** are special statements that change the order of code block execution. It introduces a **choice of paths** that the code could follow.

Code execution is like a car on a one-way street and control flow statement is like an intersection on that street. So the code execution car may turn left, turn right or continue straight. This is a powerful tool for scripter and a way to reduce code clutter.

Control flow statements vary between scripting languages but they have the same functionality.

Control Flow Statement

We can categorize control flow statements by their effect on code execution. The following are the main control flow statement categories:

- **Loops**: executing a set of statements zero or more times, until some condition is met such as for, while …
- **Conditions**: executing a set of statements only if some condition is met such as if, switch…
- **Subroutines**: executing a set of distant statements, after which the flow of control usually returns to its place such as functions, procedures…
- **Jumps or go-to**: continuation of code executing at a different statement such as goto, labels…
- **Interrupts**: low-level mechanisms that can alter the flow of control in a response to some external stimulus or event such as events, messages, exception…
- **Exit**: stopping the program and preventing any further code execution such as exit, assert…

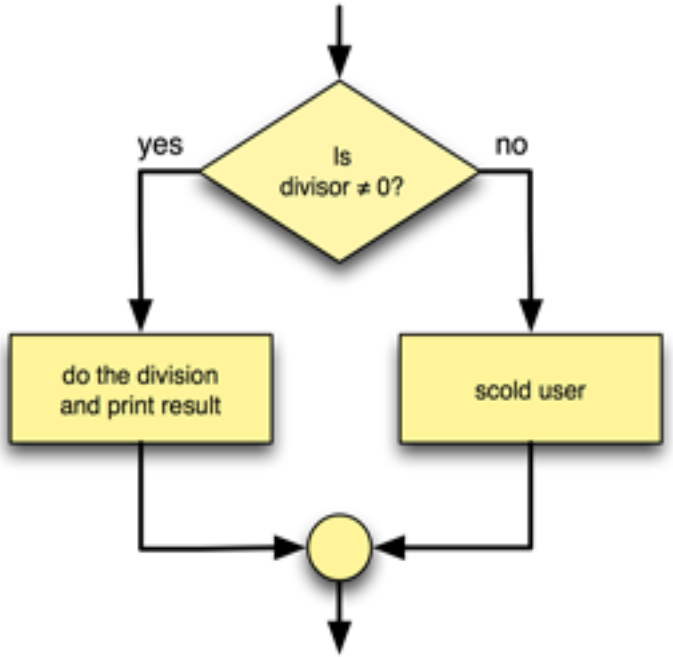
Subroutines

Subroutines are code segments written once and then used many times from various places in the program. They are also refer to as routines, procedures, functions or methods if they belong to classes.

Back when computer memory was very small compared to today, subroutines were used to reduce program size.

More frequently subroutines are used to help make a program more structured by isolating an algorithm, controlling access to a hidden data and adding program modularity.

Conditional statement



Conditional statement performs different actions depending on whether the conditional expression evaluates to true or false.

The if–else construct is common across many programming languages having varying syntax but with the same basic structure.

In English a complete sentence requires a noun and a verb likewise if statement requires a noun (*an expression that evaluate to true/false*) and a verb (*an action or instruction to be executed for true situation and optional instructions for the false situation*).

if else statement

```
if ( x > 0)
{
    print x;
}
else
{
    print “x is negative”;
}
```

1. The statement must start with the key phrase **if**

2. An expression that evaluate to true false like **(x > 0)**

3. An **action** for the true situation

4. An **optional** key phrase **else** and an action for the false situation.

if else statement

We also could chain if statement together or nest them within each other to get further control on our script execution. However, depending on code specifics chaining or nesting if statement may results in slow code execution.

When a condition could occur only in the absence of another condition it is faster to use if-else-if in your script.

For example:

```
if (color == red){
        stop();
}
else if (color == green){
        go();
}
else{
        getReady();
}
```

Here color cannot be red and green at the same time and only when the color is not Red the Green test will take place. Now this will speed code execution only if the sequence of color is Red followed by Green.

Comparison & logical operators

Comparison operators are used to test the relation between to value and always have true (positive digit) or false (zero/Null) results while logical operators are used to compare the results of the comparison operators or to reverse their result.

For example, let m = 2 and w = 4, the following show how logical and comparison operators are used:

- **m>0 && w>0** *will result in true*
- **m==10 || w ==10** *will result in false*
- **!(m==w)** *will result in true*

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

Maya MEL Example:

```
vector $myColor = <<255,0,0>>;
vector $redColor = <<255,0,0>>;
vector $greenColor = <<0,255,0>>;

if( $myColor==$redColor) //notice all in one vector comparison
    print "My color is red\n";
else if ($myColor==$greenColor)//notice all in one vector comparison
    print "My color is green\n";
else
    print "My color is neither red nor green\n";
```

Maxscript Example:

if statement in max is followed by the key phrase 'do' when there is no else or else if where instead of 'do' we use the key phrase 'then'.

if - do example:

```
if x > 0 do
    print "x is positive"
```

if - else - then example:

```
myColor = [0,255,128]
redColor = [255,0,0]

if  myColor.x == redColor.x and
    myColor.y == redColor.y and
    myColor.z == redColor.z then
    print "My color is red\n"

else
    print "My color is not red\n"
```

Python Example:

```
myColor = [0,255,0]
redColor = [255,0,0]
greenColor = [0,255,0]

if myColor[0] == redColor[0] and
   myColor[1] == redColor[1] and
   myColor[2] == redColor[2] : # <-- notice the ':'
print 'My color is red\n'

elif myColor[0]==greenColor[0] and # <-- notice 'elif' not else if
      myColor[1]==greenColor[1] and
      myColor[2]==greenColor[2]: # <-- notice the ':'
print 'My color is green\n'
else: # <-- notice the ':'
print 'My color is neither red nor green\n'
```

Lua Example:

```
myColor = {0,255,128}
redColor = {255,0,0}
greenColor = {0,255,0}

if  myColor[1] == redColor[1]  and
   myColor[2]  == redColor[2]  and
   myColor[3] == redColor[3]  then
   print ("My color is red\n“)

elseif  myColor[1]  == greenColor[1]  and
myColor[2]  == greenColor[2]  and
myColor[3]  == greenColor[3]  then
   print ("My color is green\n")

else
   print ("My color is neither red nor green\n")

end -- notice the ‘end’ phrase to terminate if statement
```

Case - Switch Statement

A switch-case statement (choice statement) executes one of several code segment based on the controlling conditional value which can be an int or string in MEL, and any value in maxscript. When the control value is equal to case statement value then the code segment under it gets executed. Sometime, we use case-switch instead of ‘if - else if’ statement to speed and organize our scripts.

Python and Lua **don’t** have a C-style switch statement. However there are ways to emulate the same effect. We could use an ‘if ... elif ... elif ...’ sequence in Python as a substitute for the switch or case statements. You’ll find many ways to mimic case-switch statement for Python and Lua if you google the term case switch and the language name.

Maya MEL Example:

```
string $myChannel = "TLC";

switch(tolower($myChannel)) //notice the 'tolower' function
{
  case "tlc":
        print "The Learning Channel\n";
        break;
  case "abc":
        print "The American Broadcasting Company Channel\n";
        break;
  case "nbc":
        print "The National Broadcasting Company Channel\n";
        break;
  default:
        print "Unknown channel\n";
}
```

Maxscript Example:

```
myChannel = "ABC"

case tolower(myChannel) of -- notice the ‘tolower’ function & ‘of’
(-- notice ‘(‘ not ‘{‘ to start the code block
   "tlc": print "The Learning Channel\n“ -- notice no case phrase

   "abc": print "The American Broadcasting Company Channel\n"

   "nbc": print "The National Broadcasting Company Channel\n"

   default: print "Unknown channel\n"
)-- notice ‘)’ not ‘}’ to close the code block
```