Week 02  MEL basic syntax I

- The Basics of MEL syntax (part 2): array, random, and loops

The Basics of MEL syntax (part 2)

- **Arrays:** Array puts a series of variables of same data type in an ordered fashion. You can access individual variable using a common variable name and an index value. For example:
  
  ```mel
  int $numbers[5];
  ```
  
  declares 5 variables with a common name `$numbers`, and the members of this array are numbered 0,1,2,3 and 4, i.e. the third member can be accessed by `$numbers[2]`.

Examples of using array in MEL:

```mel
int $a[27];
int $b;
$a[9] = 6;
$b = $a[0] + $a[10] * 28;
```

**Initialize an array:** you can create and initialize an array in a single statement. For example:

```mel
int $numbers[5] = { 2, 5, 4, 7, 8 };
```

You can use the initialization form only when declaring the array. You cannot use it after declaration. For example:

```mel
int $a[4] = {2, 5, 6, 3}; // this is okay
int $a[4]; // this is also okay
$a[4] = {2, 5, 6, 3}; // this is not allowed
$a[3] = 17; // but this is okay.
```

**Size of an array:** in MEL, you can declare an array without specifying the size. Maya will increase the size of the array when necessary. For example:

```mel
int $a[]; // an integer array without size
$a[0] = 4; // this is okay;
$a[50] = 25; // the size of the array increase automatically
```

To get the size of an array, you can use `size($a)`; To clear an array, you can use `clear($a);`
Class exercise: changing torus to cone

You are given a scene J:\SM3122\week02-start.mb. In the scene there are many torus, placed in different location, and have different size and orientation. Now assume that you want to change some of the torus into cone, but keeping the position, size and orientation unchanged.

Try to make a shelf button that can do the following: whenever I select a torus and then click the button, the selected torus will be changed to a cone, but the position, size and orientation will not change. You can assume that only one object is selected at a time.

Hints: your script may contain the following steps:
1. Get the name of the selected object using selectedNodes (see the example above).
2. Get the attributes translateX, translateY, translateZ, rotateX, rotateY, rotateZ, scaleX, scaleY, scaleZ of the select object. Store these values into variables. Use the getAttr command (see the example above).
3. Delete the selected object using the command delete;
4. Create a cone using the command cone –axis 0 1 0;
5. Move, rotate and scale the cone using the command move, rotate and scale.

The following example is quite useful: it gets the name of the current selected nodes (using the MEL command selectedNodes) and stores them into an array of string. Finally, it prints the name of the first selected object:

```
string $selected[];
$selected = selectedNodes();
print ( $selected[0] );
```
Random number

One thing that a MEL writer usually does is to use “random” to generate something. For example:

- Create a sphere and put it in a random position.
- Create a shape and randomly move its vertices a little bit.
- Create particles moving in a random speed and direction.

The MEL command to generate a random number is `rand`. It can be used in two ways:

```
rand( 10 );
```
will generate a floating-point number between 0 to 10 (not including 10); and

```
rand( 3, 98 );
```
will generate a floating-point number between 3 and 98.

For example, guess what will happen:

```
select "myCone1";
move -y ( rand( -0.5, 0.5 ) );
```

Class exercise: random torus

Write a MEL script to generate a torus at a random position, with random scale and random rotation. (Decide yourself what is the “reasonable” range for each random number.)
Repetitive tasks

In most situations, a script needs to perform repetitive tasks. To perform repetitive tasks, we can use the following statements:

- For-loop
- For-in loop
- While-loop
- Do-while-loop

**For-loop**

You can use a **for-loop to execute a block of statements in a fixed number of times.**

A for-loop has the following syntax:

```plaintext
for (initialization; condition; update) {
    statements_1;
    statements_2;
}
statements_3;
```

**Meaning:** When a script reaches a for-loop, it will perform the following:

1. Execute the statement in the *initialization* part.
2. Check the *condition*.
3. If the *condition* is false, jump to *statements_3* and continue.
4. If the *condition* is true, execute *statements_1* and *statements_2*.
5. Execute the statement in the *update* part.
6. Repeat step (2) to (5), until the *condition* is false.

For example, guess what will happen:

```plaintext
int $i;
for ($i = 1; $i < 5; $i = $i + 1) {
    sphere -radius 0.2;
    move -x $i;
    move -y (rand( -0.5, 0.5 ));
}
```

You can think that the variable $i$ is a counter. It is initialized to 1, and increased by 1 every time after the block of statements is executed. The block will be executed until $i$ is greater than or equal to 5. **So, many spheres are created?**

(Note that the variable $i$ can also be used inside the block of the for-statement.)

**Question:** how about this?

```plaintext
int $i, $j;
for ($i = 0; $i < 10; $i = $i + 1) {
    for ($j = 0; $j < 10; $j = $j + 1) {
        sphere -radius 0.2 -pivot $i (rand(1)) $j;
    }
}
```
For-in loop
This is a special version of for-loop in MEL.

```
for ( $name in $arrayName ) {
    statements_1;
}
```

**Meaning:** The variable $name will take the value of $arrayName[0], and then statements_1 will be executed. The process repeats for each element $arrayName[1], $arrayName[2], …, until all elements in the array $arrayName has been processed.

For example, the following code prints the name of all selected objects:

```
string $selected[] = selectedNodes();
string $name;
for ( $name in $selected ) {
    print( $name + "\n" );
}
```

---

**Class exercise: for-loop exercises**

Modify your answer in the previous exercise “random torus”, so that it generates 40 torus in random position, with random size and random rotation.

**Class exercise: for-loop exercises**
(Need to hand in at the end of the class)

Modify your answer in the previous exercise “changing torus to cone”, so that it converts all selected torus (can be more than one) into cone at one time. **Hints:** add a for-loop or a for-in loop.

Save the script into a file, and use your student number as the file name. Put the file into the folder J:\SM3122\submit\Week02\ at the end of this class.
Class exercise: Random landscape

Step 1) Create a polygonal plane. Since the default subdivision is 10x10, therefore, your plane should have 11x11 = 121 vertices.

Step 2) Select one vertex, and using the “learn from echoes” technique, **tries to find out what is the MEL statement to select a vertex of a plane**.

Step 3) Write a script that **moves all vertices of the plane along the y direction in a random distance**. Your script should create a shape like this:

![Random Landscape](image)

Class exercise: Tower of square

Step 1) Create a polyCube using Polygons tool. You need to use **“learning by echo”** trick to learn different parameters of controlling the height, width, depth of the polyCube.

Step 2) Move the newly created polyCube along Y direction a bit and learn the corresponding MEL comment.

Step 3) Rotate the polyCube along Y axis a bit and learn the corresponding MEL comment too.

Step 4) Write MEL scripts to create the following:

![Tower of Square](image)
While-loop
When you **don’t know how many times a block should be executed**, you cannot use for-loop. In this situation, you can use a while-statement:

```c
while (condition)
{
    statements_1;
}
statements_2;
```

**Meaning:** When the script reaches a while-loop, the *condition* is checked. If it is true, the block *statements_1* will be executed. Then the *condition* is checked again. If it is still true, the block will be executed again. This situation will continue, until the *condition* is evaluated false. Then the script will go to *statements_2* and continue.

NOTE that inside *statements_1*, it must have some statements which will change the variable in the *condition*. Otherwise, the loop will never end.

For example, guess what will happen:

```c
float $rad = 1;
int $x = 0;
while ( $rad > 0.05 ) {
    sphere -radius $rad -pivot $x (sin($x)) 0;
    $x = $x + 1;
    $rad = $rad * 0.8;
}
```

Do-while-loop
Do-while-loop is similar to while-loop, but the block of statements will be executed **at least once**. Besides this, there is nothing different from a while-loop:

```c
do {
    statements_1;
} while (condition);
statements_2;
```

**Meaning:** The block *statements_1* is executed, and then the *condition* is checked. If it is true, the block *statements_1* will be executed again. Then the *condition* is checked again, and this situation will continue, until the *condition* is evaluated false. Then the script will go to *statements_2* and continue.
Class exercise: spiral

Given the formula of a spiral as:
\[ x = y \cos(\text{angle}) ; \]
\[ z = y \sin(\text{angle}) ; \]
(Note: in MEL, angle is measured in radian, not degree.)

Write a MEL script, **using a while-statement**, to generate the following spiral shape:

**Hint**: your while-loop will looks like this:

```mel
float $x, $y, $z, $angle;
while (???) // what should be put here?
{
    The variables $y and $angle will be increased a little bit.
    The $x and $z position is then calculated from $y and $angle.
    Create a sphere at position ($x, $y, $z).
}
```

** Week 02 End **