MEL

Для начала давайте вспомним как выглядит типичная mel команда.

```
Команда -атрибуты имяОбъекта;

move -relative 2 0 0 pSphere1;
```

В отличие от питона, если мы хотим применить команду к какому-то объекту в сцене, то этот объект пишется в конце.

MEL как язык программирования, был основан на языке C (Си), и перенял в себя много черт (которые я не считаю преимуществом). Например в конце команды ставится точка с запятой. А когда мы создаем функцию, цикл или условие - внутренний код мы объявляем в фигурных скобках.

Очень важным моментом в MEL является типизация данных. В отличие от питона, в MEL мы для каждой переменной должны явно указать, к какому типу она принадлежит

Переменные

Для хранения информации используются переменные. В MEL они начинаются со знака доллара и далее следует название (абсолютно любое). Например:

```
string $myName = "Roman";
int $myAge = 30;
float $PI = 3.14;
$someNumber  = 13;
```

В MEL программист должен указывать тип данных, с которыми он работает. Это позволяет Maya правильно понимать как работать с нашим кодом. MEL также позволяет объявлять переменные без типа (Майа может автоматически распознать тип данных в переменной), однако это считается плохим подходом в языке MEL, поскольку может привести к непредвиденным ошибкам и добавит сложности в изучении этого языка.

В MEL существуют следующие типы данных:

- int (целочисленные, например 2, 44, 23)
- float (с плавающей точкой, например 2.34, 77.5)
- string (текст, например “hello”, “mel is fun”)
- vector (вектор, например <<2 ,5, 0>>)
- matrix (матрицы, например <<1,2,3; 4,5,7; 7,8,9>>)

Рекомендуется давать интуитивно понятные названия переменным, так как это очень сильно упрощает чтение кода. Когда вы напишите сложный скрипт, и вернетесь к нему через месяц чтобы продолжить его улучшать - вы прочувствуете насколько правильное наименование переменных важно и как это облегчает жизнь.

Так-же мы можем назначить одной переменной значение другой переменной, однако тип данных у них должен совпадать.

```
string $myName = “Roman”;
string $BBB = $myName; // $BBB теперь “Roman”
```

Некоторые команды MEL возвращают определенный результат (например проверка существования объекта в сцене) и чтобы сохранить этот результат в переменную, достаточно этой переменной присвоить саму команду, заключенную в одинарных обратных кавычках (` - под знаком ~), например:

```
//Проверяет существует ли объект в сцене. Если да - возвращает 1, иначе 0
//Результат команды objExists (1 или 0) мы сохраняем в переменную $inScene
int $inScene = `objExists pSphere1`;
```

Для того, чтобы проверить, какое значение в переменной - мы можем это значение вывести в верхней части Script Editor. Для этого используется команда print

```
//Напечатает: 1
print($inScene);
```

Мы не можем использовать одну и ту же переменную для сохранения в нее различных типов данных. Однако мы можем использовать одно и то же имя переменной внутри разных функций.

Комментарии в коде

Любой код нужно комментировать, чтобы в дальнейшем можно было легко вспомнить, что там написано. В MEL любая строка, начинающаяся с // считается комментарием и не обрабатывается Maya. Для превращения целого блока текста в комментарий - достаточно поставить перед ним /* и после него */. Многие языки программирования имеют проблемы с восприятием кириллицы, поэтому писать комментарии нужно на английском.

```
//Comments one line

/*
All text
here is commented
*/
```

Массивы

Переменные могут хранить не только одно значение, но так же и целый список значений. Однако все эти значения должны быть одного типа. Такая переменная зовется массивом, и пишется так:

```
string $name[2] = {"Roman","Volodin"};
print $name[1]; // напечатает “Volodin”
```

Следует помнить, что нумерация элементов массивов начинается с НУЛЯ. Печать $name[0] выдаст "Roman".

Другие способы объявления массивов:

```
$array1 = {1, 2, 3}; // Maya сама распознает тип данных
float $array2[] = {12.3, 15.7}; //Можно не указывать явное количество элементов
```

Чтобы очистить массив, используется команда clear

```
clear($array1);
```

Условия

Проверка определенных условий в программе является самым главным принципом программирования. Примерно условие на любом языке программирования можно описать так:

```
Команда условия (само условие)
{
    Если условие выполнилось - выполнить участок кода
}
Иначе если (новое условие)
{
    Выполнить этот код
}
Иначе
{
    Если первые два условия не выполняются, выполнить этот код
}
```

Например

```
int $a = 13;

if($a == 16)
{
    print "$a is 16";
}
else if($a > 10)
{
    print "$a is not 16, but bigger than 10";
}
else
{
    print "$a is too small";
}
```

Условие, которое мы написали в скобках if($a == 16), сравнивает значение переменной $a с числом 16. Это не единственный тип условий, существуют и другие типы:

```
if($a == 16)         Если $a равно 16
if($a != 16)         Если $a не равно 16
if($a < 16)      Если $a меньше 16
if($a > 16)      Если $a больше 16
if($a >= 16)         Если $a больше либо равно 16
```

```
if($a <= 16)          Если $a меньше либо равно 16
```

Мы можем комбинировать различные условия с помощью следующих символов:

```
//Условие считается выполненным в том случае, если одно из двух сравнений
выдало положительный результат
||   (ИЛИ)        if($a == 16 || $a > 20)

//Условие считается выполненным, если оба сравнения дали положительный
результат
&&   (И)              if($a == 16 && $a < 20)

//Условие выполнено если $a не равно 20
!    (НЕ)         if($a != 20)
```

Switch .. case - является альтернативой условия if. Его удобно применять когда мы должны проверить много условий и если одно из них выполняется - завершить дальнейшую проверку.

```
int $b = 10;

switch ($b) {
case 4:                  //если $b == 4 - вывести текст и завершить условие
    print "$b is 4";
    break;               //оператор завершения
case 5:
    print "$b is 5";
    break;
case 6:
    print "$b is 6";
    break;
default:
    print "$b is something else";
    break;
}
```

While является еще одним типом условий. Код внутри блока while будет выполняться, пока не будет выполнено определенное условие. While это опасный оператор, если программа начнет выполнять код внутри while, и условия никогда не будет выполнено - вы столкнетесь с таким явлением как бесконечный цикл, и придется перезагружать Maya без возможности сохранить код или сцену.

```
int $a = 1;
while($a < 10)
{
    print $;
    $a = $a + 1; // будет добавлять единицу пока $a не будет равно 10
}
```

Можно использовать альтернативную конструкцию while

```
do {
    код
}
while (условие)
```

Циклы

Циклы являются второй очень важной особенностью программирования и служат для работы с массивами. Простыми словами цикл можно описать так: перебрать все элементы массива, и для каждого элемента выполнить код.

Циклы в MEL бывают разными. Самый распространенный вид цикла (который так же имеется и в других языках программирования) выглядит следующим образом

```
for(инициализация ; условие ; изменение условий)
{
    код
}
```

Данный цикл позволяет выполнить код столько раз, сколько потребуется, пока условие не будет выполнено.

Инициализация это начальное условие цикла. Например переменная $j вначале равна 0.

Условие определяет то, когда цикл должен закончиться. Например, когда значение $j достигло 10.

Изменение условий определяет то, как $j должна изменяться с каждой итерацией цикла.

Например

```
int $j; //у нас имеется некая переменная, созданная специально для цикла
$array1 = {"red", "green", "blue"}; // У нас есть массив, хранящий цвета

//мы обозначили условия цикла.
//$j++ увеличивает значение $j на единицу.
for ($j = 0; $j < 3; $j++){
    //Мы печатаем элемент массива $array1
    //Как только $j == 3, прекратить работу цикла и продолжить выполнять
остальной код
    print ($array1[$j] + "\n");  // \n это символ перехода на новую строку
}
```

Более удобный способ перебрать элементы массива - это использовать цикл for-in.

```
$array1 = {"red", "green", "blue"}; // У нас есть массив, хранящий цвета

for($color in $array1){
    print ($color + "\n");
}
```

В данном цикле переменной $color каждый раз присваивается значение очередного элемента массива. Т.е. при первом запуске цикла $color = "red", при втором "green" и т.д. Данный тип цикла является наиболее часто используемым в MEL.

break; - это команда, позволяющая завершить цикл при определенных условиях. В таком случае, имея тысячи элементов в массиве, и найдя нужный нам элемент, команда break избавит Maya от необходимости проверять оставшиеся элементы.. что может занять прилично времени.

```
$array1 = {"red", "green", "blue"}; // У нас есть массив, хранящий цвета

for($color in $array1){

    if($color == "green")
    {
        print ($color + "\n");
        break;      //Завершить цикл и продолжить выполнять остальной код
    }
```

```
}
```

continue; - это команда, позволяющая перейти к следующей итерации цикла, не выполняя оставшийся код тела цикла. Например:

```
$array1 = {"red", "green", "blue"}; // У нас есть массив, хранящий цвета

for($color in $array1){

    if($color == "green")
    {
        print ($color + "\n");
        continue;       //Если $color = "green" - перейти к следующему цвету, минуя сложные вычисления для "green"
    }

    ...
    //Сложные вычисления которые занимают время

}
```

Разница между = и ==

Операторы = и == имеют абсолютно разное назначение.

Оператор = назначает переменной какое то значение, например $a = 10. Теперь, где бы мы не вызвали $a, мы получим 10.

Оператор == сравнивает значения между левым и правым операндом (операнд - переменные, числа и т.д., т.е. то что обрабатывается). В случае если значения совпадают, возвращается значение True или 1, иначе возвращается False или 0.

Процедуры (функции)

Процедуры позволяют взять конкретный участок кода, и занести его в отдельный блок. Таким образом мы делим код на отдельные детали, каждая из которых имеет свое предназначение, и в совокупности они являются программой.

Процедура начинается с ключевых слов global proc, далее следует название процедуры, круглые скобки () - которые дают Maya понять что это процедура, и фигурные скобки, внутри которых находится код.

```
global proc MyProc()
{
    print "test"; // тело функции
}
```

Чтобы вызвать процедуру, достаточно написать ее название где-либо в коде.

```
//где-то в коде нашей программы
MyProc;
```

Процедура, как и любая MEL команда, может возвращать результаты. Однако при этом мы должны в объявлении процедуры указать тип возвращаемого значения. Например, если мы производим всякие расчеты, и нам нужно выдать названия геометрического объекта и присвоить его переменной, делается это следующим образом.

```
global proc string MyObj()
{
    // тело функции, всякие вычисления
    string $obj = "pSphere1";

    return $obj;
}

string $myObject = `MyObj`;
```

У каждой процедуры могут быть 0 или несколько аргументов. Использование аргументов позволяет использовать процедуры многократно, получая различные результаты.

Например

```
global proc string MyName(string $firstName, string $secondName)
```

```
{
    string $fullName = ($firstName + " " + $secondName);
    return $fullName;
}

print `MyName "Roman" "Volodin"`; // Выдаст "Roman Volodin"
```

Глобальные переменные

У программы есть разные уровни доступа к переменным. Переменные бывают локальными и глобальными. Если мы создали какую-либо переменную внутри процедуры - она является локальной и видимой только внутри этой процедуры. Попытка обратиться к этой переменной вне процедуры вызовет ошибку.

Например:

```
global proc MyProc()
{
    int $aaa = 15;
}

print $aaa; // выдаст ошибку, т.к. $aaa существует внутри MyProc.
```

Чтобы переменная стала глобальной, нужно поставить ключевое слово global перед объявлением переменной, и далее объявлять ее на всех уровнях доступа, чтобы дать понять что мы имеем дело с глобальной переменной. Глобальная переменная становится видной всем функциям, и скриптам в текущей сессии Maya.

```
global proc MyProc()
{
    global int $bbb;
    print $bbb;
}

global proc AnotherProc()
{
    global int $bbb;
    print $bbb;
}
global int $bbb = 20;

MyProc;
AnotherProc;
```