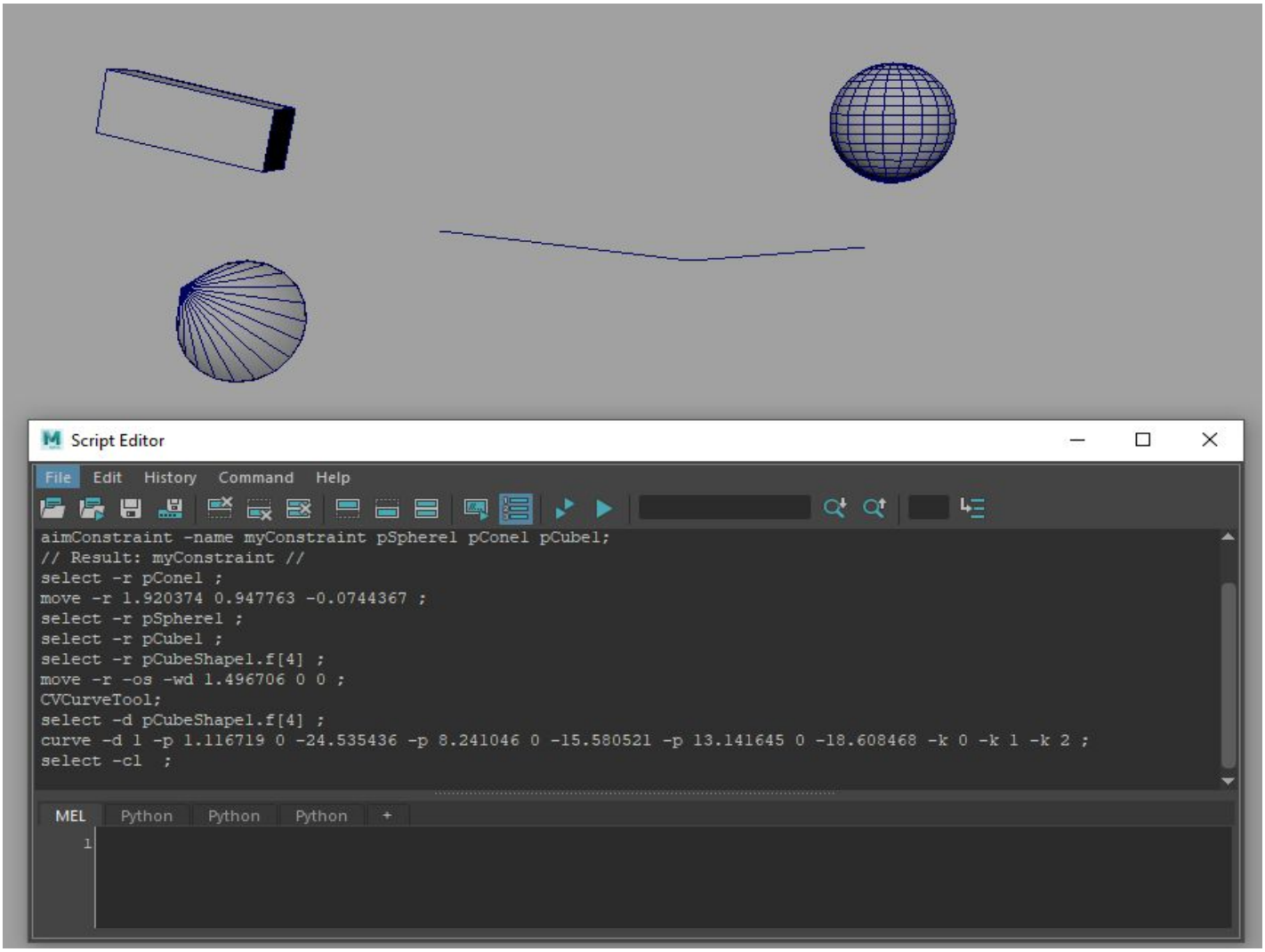
Анатомия команды Maya

На каждое действие, которое вы совершаете в Maya - есть соответствующая команда. Команда Maya - это некое атомарное действие, выполнив которое, мы сможем добиться единичного эффекта в сцене. Например, команда **extrude**, команда **move**, **rotate**, **scale** или **parentConstraint** и т.д.

Практически любая команда доступна на языках **MEL** и **Python**. Несмотря на то, что Python сейчас пользуется огромной популярностью, и в Maya практически все скрипты пишутся на этом языке, MEL все еще остается важной частью Maya. Большинство опций Maya, конфигураций, надстроек интерфейса и т.д. описано на языке MEL, и с ним вам придется столкнуться в будущем, но в контексте изучения программирования в Maya - MEL это до сих пор единственный язык, которым Maya умеет общаться с пользователем.

Чтобы понять, о чем речь, откройте **Script Editor**. Это окно имеет два текстовых поля - верхнее (**Command History**) и нижнее (**Script Input)**. В нижнем поле мы вводим команды на языках **MEL/Python**. Верхнее поле отображает результат выполнения наших команд, и помимо этого, показывает MEL команды, соответствующие любым действиям, которые пользователь выполняет в среде Maya, будь то выделение объекта, его перемещение, анимация и т.д. Это очень полезная опция, позволяющая новичкам видеть MEL команды, чтобы затем использовать их в своих программах.



Рассмотрим пример команды на языке **MEL**. Схематично ее можно описать следующим образом.

```
команда  -имяПараметра значениеПараметра  -имяПараметра значениеПараметра  объект;
```

Синтаксис Python слегка отличается

```
команда(объект, имяПараметра=значениеПараметра, имяПараметра=значениеПараметра)
```

Основные нюансы, о которых следует помнить:

- В MEL любая команда заканчивается точкой с запятой. В Python такой необходимости нет.
- В Python текст всегда пишется в кавычках, в MEL можно и с ними с без. Единственным правилом в MEL считается следующее - если текст, который вы хотите использовать в команде, имеет пробелы - кавычки обязательны.
- Mel больше похож на команду из терминала Linux, чем на команду языка программирования. Это связано с тем, что Maya в древние времена была анимационной программой для UNIX систем, в которых команды очень похожи на MEL.
- Есть команды, которые влияют на конкретный объект в сцене. Например, чтобы установить анимационный ключ для объекта **pSphere1**, мы должны в команде **setKeyframe** в конце после всех параметров задать имя объекта. В Python имя объекта задается вначале. Сравним два варианта:

```
setKeyframe -attribute rotateX -value 30 pSphere1; // MEL
```

```
import maya.cmds as cmds
cmds.setKeyframe("pSphere1", attribute="rotateX", value=30)  # Python
```

Работа с документацией

Если перейти в документацию, где нам представлен список всех команд Maya, и открыть любую команду, например **aimConstraint**, мы увидим много полезной информации, которая позволит понять, что делает команда, как ее нужно писать чтобы добиться нужного эффекта, а также многое другое. Разберем подробнее команду **aimConstraint**.

Synopsis

В пункте **Synopsis** мы видим краткое описание команды. В желтом поле представлена сама команда и ее возможные параметры. Параметры, помеченные синим цветом и записанные в квадратных скобках, являются необязательными, т.е. если мы их не напишем - команда все еще будет работать. Последние два параметра **[target…]** и **object** являются обязательными, поскольку для **aimConstraint** важно, чтобы вы указали **цель** и **объект**, который будет следить за этой целью. Параметр **[target...]** в квадратных скобках и многоточием означает, что вы можете указать несколько целей.

Synopsis

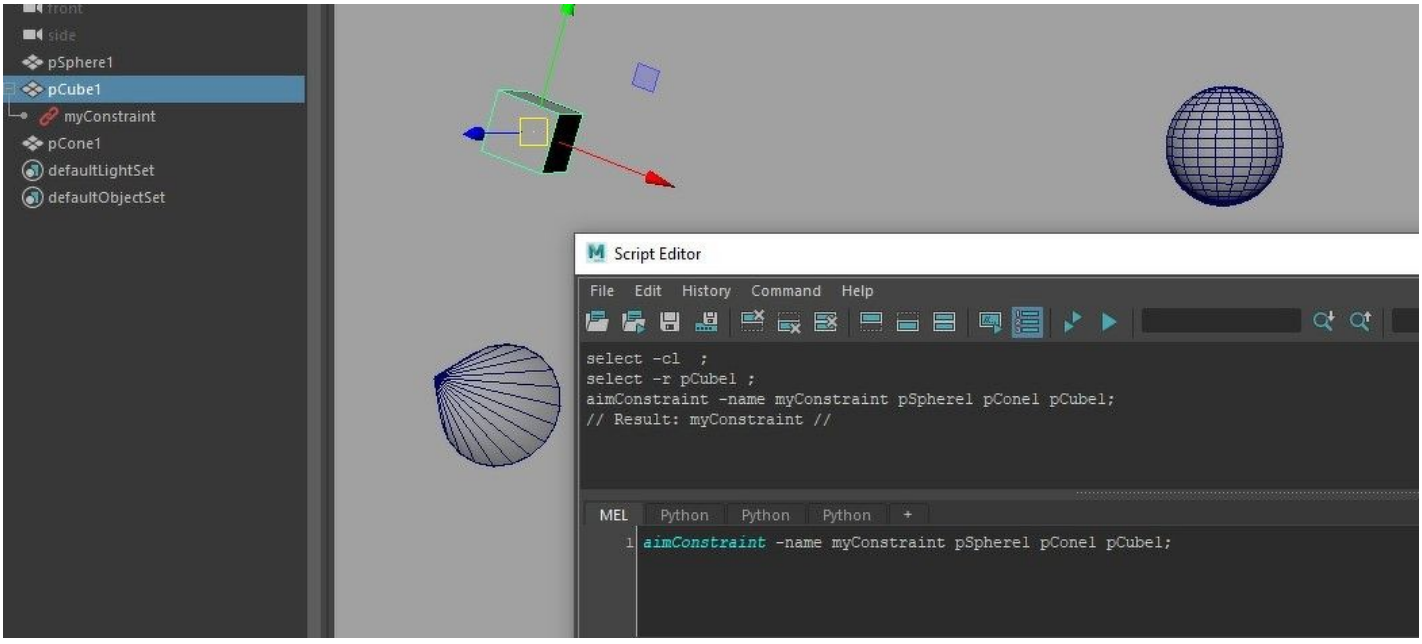
```
aimConstraint [-aimVector float float float] [-layer string] [-maintainOffset] [-name string] [-
offset float float float] [-remove] [-skip string] [-targetList] [-upVector float float float] [-
weight float] [-weightAliasList] [-worldUpObject name] [-worldUpType string] [-worldUpVector float
float float] [target...] object
```

*Справки команд не всегда следуют единому формату (видимо их все в свое время писали разные люди). Например, если мы откроем команду **bindSkin**, в конце мы увидим параметр **[objects]** без многоточия. Но суть остается той же - квадратные скобки и слово objects в множественном числе сигнализируют нам о возможности выполнить команду **bindSkin** для множества объектов.*

Параметры команды - это набор опций, который влияет на то, каким будет результат запуска команды. Например, опция **-name** позволяет указать имя ноды **aimConstraint**, которую мы создадим при выполнении команды. Вместо стандартного имени **pCube1_aimConstraint1** мы можем задать имя **myConstraint**.

У любого параметра есть длинное имя и короткое. Это выбор программиста, какой из вариантов использовать, но я рекомендую всегда писать длинное имя, поскольку **aimConstraint -n myC** менее понятно чем **aimConstraint -name MyC**.

Теперь, выполним команду aimConstraint так, как это описано в документации.



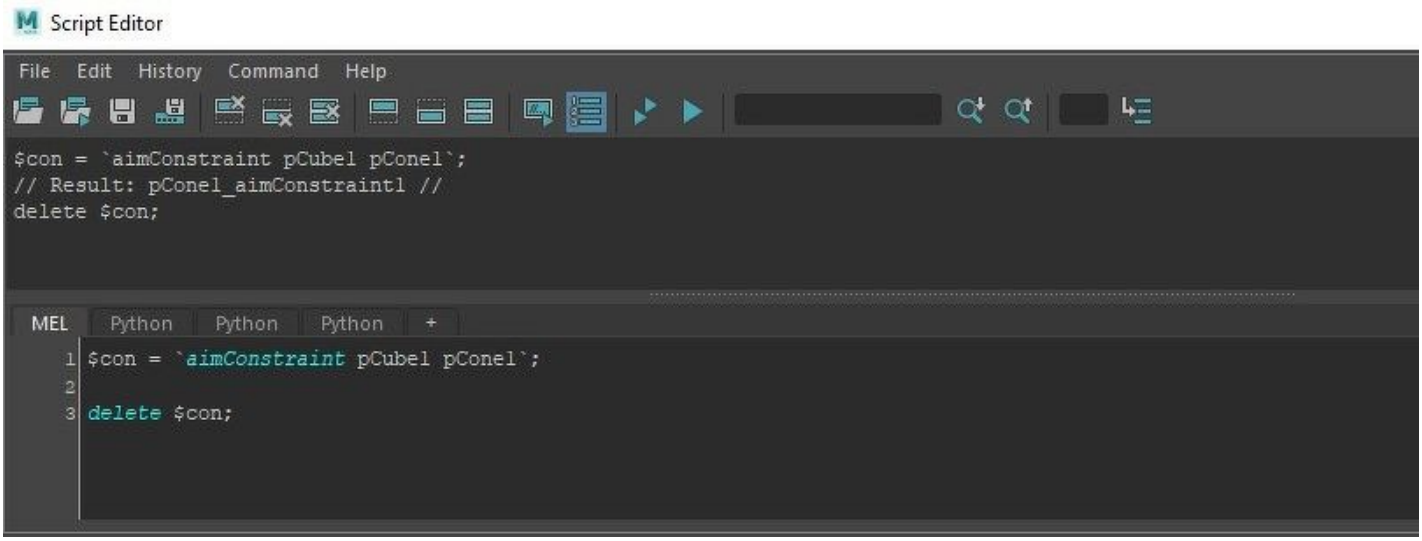
Return Value

Если мы посмотрим в документацию чуть ниже, мы увидим пункт **Return Value**. Этот пункт является очень важным, поскольку он дает нам понять, что команда, когда мы ее выполним, в качестве результата, помимо основного действия (создания ноды **aimConstraint**) возвращает нам полезную информацию, которую мы потом сможем использовать в нашем коде в дальнейшем.

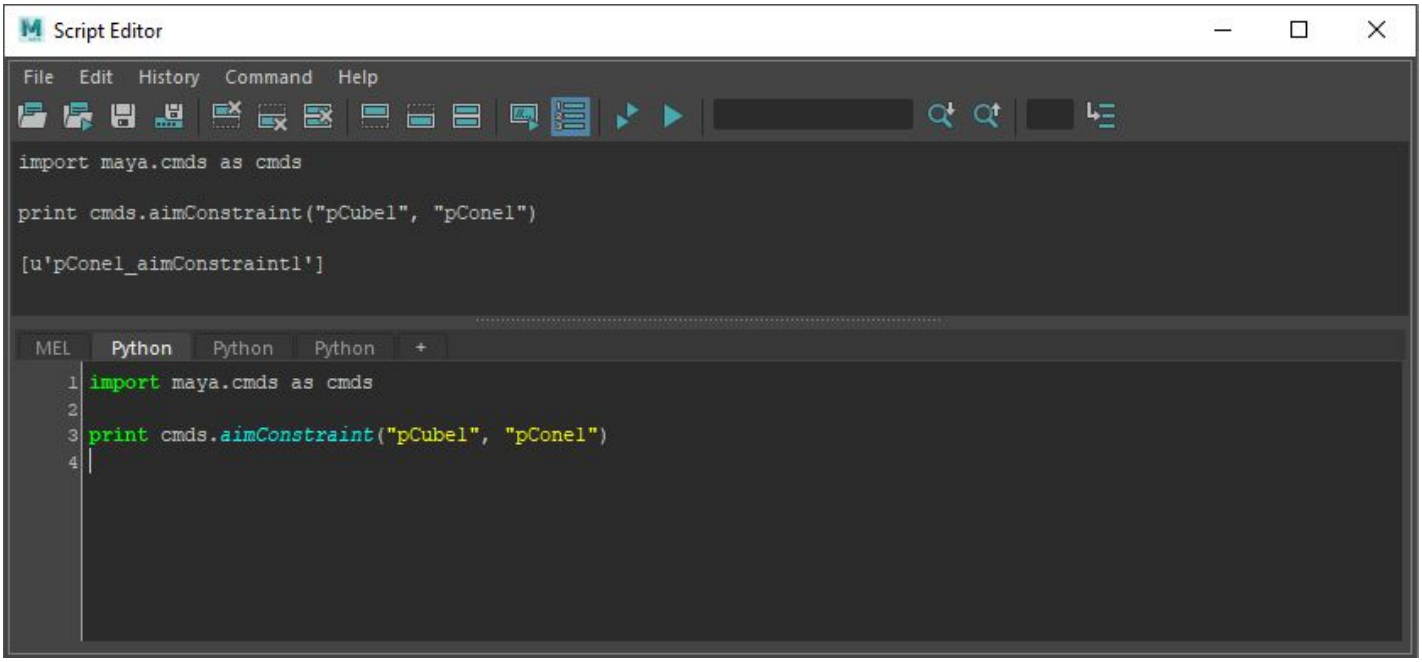
Например, aimConstraint — это команда, которая создает ноду в сцене. Если вы создадите Куб и Конус, выделите их, перейдете в меню [Animation]->Constraint и запустите Aim — в сцене создастся дополнительная нода, которая отвечает за то, чтобы один объект постоянно смотрел на другой.

Теперь, если выполним команду **aimConstraint pCube1 pCone1** - в поле History окна **ScriptEditor** мы увидим новую строку **// Result: pCone1_aimConstraint1 //** . Это значит, что команда вернула нам имя ноды **aimConstraint**. Эту информацию мы можем использовать в дальнейшем. Например, если объект-цель был анимирован, мы можем выполнить bake animation для объекта, который следит за целью, и, поскольку мы знаем имя ноды aimConstraint - удалить ее из сцены.

Если следовать такой задумке - создать aimConstraint, сохранить его имя в некую переменную, а затем, используя эту переменную - удалить constraint из сцены, то код будет выглядеть следующим образом:



В отличие от MEL, Python не печатает нам автоматически результат выполнения команды. Чтобы узнать, какую информацию команда нам вернула, мы должна вставить слово **print** перед самой командой.

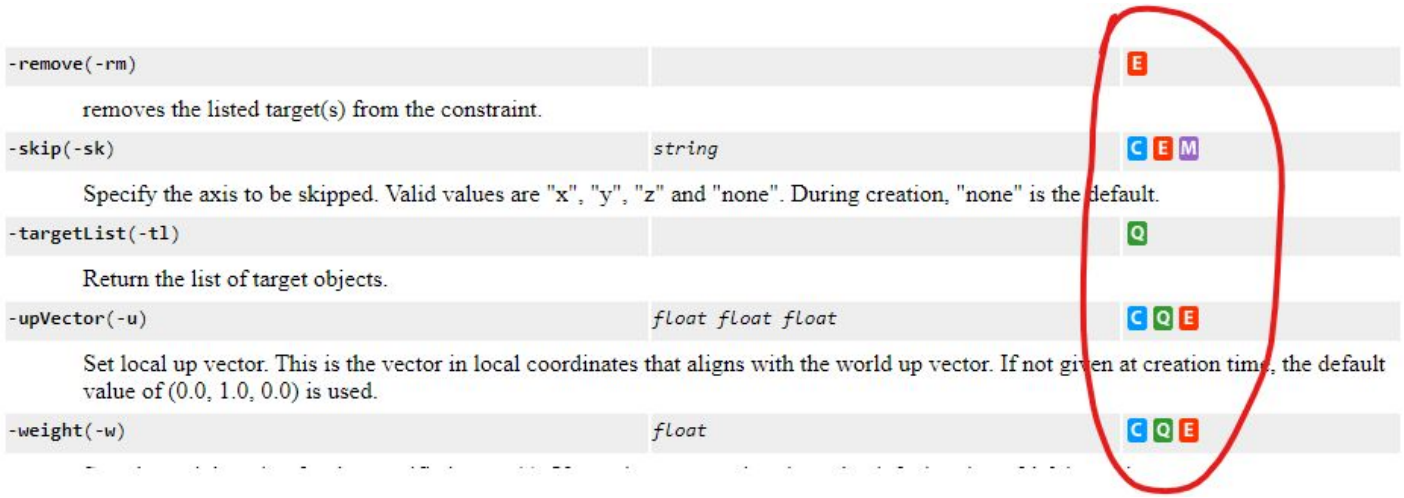


MEL vs Python

Если обратить внимание на верхнюю часть документации команды, мы можем увидеть ссылку **Python version** либо **MEL version**. Данная ссылка очень удобна, т.к. позволяет нам переключиться либо на MEL справку команды, либо на Python версию справки этой же команды.

Режимы команд

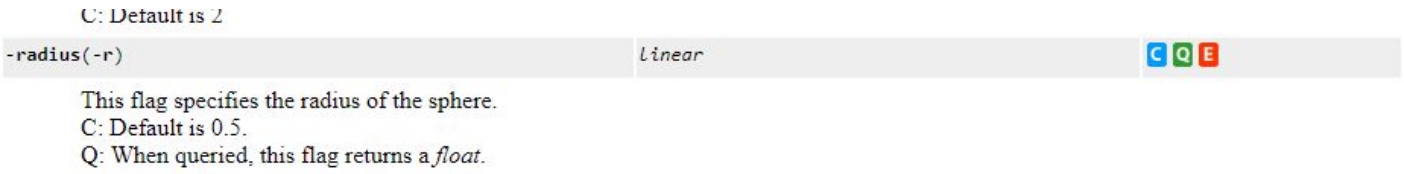
В документации вы наверняка обратили внимание на странные символы напротив каждого параметра.



`-remove(-rm)`		E
removes the listed target(s) from the constraint.		
`-skip(-sk)`	*string*	C E M
Specify the axis to be skipped. Valid values are "x", "y", "z" and "none". During creation, "none" is the default.		
`-targetList(-tl)`		Q
Return the list of target objects.		
`-upVector(-u)`	*float float float*	C Q E
Set local up vector. This is the vector in local coordinates that aligns with the world up vector. If not given at creation time, the default value of (0.0, 1.0, 0.0) is used.		
`-weight(-w)`	*float*	C Q E

Эти символы относятся к режимам команды. Их сложно понять без примеров, поэтому давайте разберем примеры.

Режим C. Представим ситуацию, в которой нам понадобилось создать полигональную сферу. В Python выполним следующую команду, которая создаст сферу с радиусом 3. Режим C (Create) - это режим команды **polySphere**, при котором мы впервые создаем какой-либо объект. “C” напротив параметра radius означает, что мы можем задать радиус, когда мы только впервые создаем сферу.



C: Default is 2

`-radius(-r)`	*linear*	C Q E

This flag specifies the radius of the sphere.
C: Default is 0.5.
Q: When queried, this flag returns a *float*.

```
import maya.cmds as cmds

cmds.polySphere(name="mySphere", radius=3)
```

Режим Q. Это режим запроса информации. Данный режим позволяет узнать значение какого-либо параметра для конкретного объекта. Например, сферу мы уже создали, теперь, если мы хотим узнать ее радиус (для каких-то вычислений в дальнейшем) - мы выполняем следующий код:

```
print cmds.polySphere("mySphere", q=1, radius=1) # 3.0
```

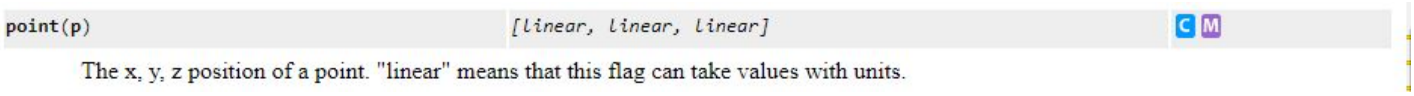
Для запроса информации о каком-либо мы должны использовать параметр q=1, а далее сам параметр, значение которого мы хотим узнать. Причем параметру должно быть присвоено значение 1. Больше одного параметра мы запросить не можем.

Режим E. Это режим редактирования параметра. Сферу мы уже создали, и даже узнали какой у нее радиус. Но чтобы изменить радиус у сферы, используя ту же команду **polySphere**, мы пишем следующее:

```
cmds.polySphere("mySphere", e=1, radius=5)
```

В отличие от режима Q, в этом режиме у параметра мы должны указать его новое значение. И также как и в Q, в этом режиме мы можем отредактировать только один параметр за раз.

Режим M. Самый редкий режим. Означает что мы можем для этого параметра задать сразу множество значений.. Для примера, возьмем другую команду **curve**. Данная команда создает NURBS кривую. Как мы знаем, кривая состоит из множества вершин, через которые эта кривая проходит. Если взглянуть на документацию по этой команде, мы увидим следующее:



Данный параметр отвечает за вершины кривой - их может быть 4, а может быть и 100. Режим M позволяет нам задать в одном параметре сколько угодно координат вершин.

```
cmds.curve( p=[(0, 0, 0), (3, 5, 6), (5, 6, 7), (9, 9, 9)] )
```

Примеры

Одна из причин, почему документация Maya является одной из самых удобных - это наличие примеров практически для каждой команды (как для MEL так и для Python версии). Достаточно прокрутить страницу прямо вниз, и вы сможете увидеть самодостаточные примеры. Самодостаточные - значит что вам не нужно проводить никаких подготовительных действий, достаточно скопировать скрипт в Script Editor, и запустить его.

Python examples

```
import maya.cmds as cmds

# Set a keyframe at the current time on all "keyable"
# attributes of the selected objects.
#
cmds.setKeyframe()

# Set a keyframe so that translateX has a value of 10
# at the current time, regardless of its current position
#
cmds.setKeyframe( v=10, at='translateX' )

# Set keyframes for translateX on two objects at t=0 and
# t=10 seconds.  (Note that if mysteryObject has no
# attribute named translateX, no keyframe is set for mysteryObject.)
#
cmds.setKeyframe( 'nurbsSphere1', 'mysteryObject', attribute='translateX', t=['0sec','10sec'] )
```