LEARNING maya

#maya

Table of Contents

About

Chapter 1: Getting started with maya

Remarks

This documentation covers coding for Autodesk Maya. It's **not** meant for end-users of the Maya software. (To find out how to model or animate in Maya, try Autodesk's introductory videos or an end-user site like CGSociety.)

Languages

Maya supports 3 programming languages: MEL, its built-in scripting language; C++, which is used for plugins; and Python which is common for integration jobs but can also create plugins using a wrapped version of the C++ API

Examples

Installation

Maya supports 3 main programming environments. Each has different setup requirements.

MEL

MEL scripting language is included with the Maya application. Enabled by default, users can test MEL in the script listener window in a running copy of Maya.

MEL files are text files with the extension `.mel`. They can be loaded into a running Maya session using the `source` command in the listener or in another MEL script. Maya maintains a list of source directories and will search for a requested MEL script in all directories until it finds an appropriately named file.

There are number of methods for setting up the script path; see the Autodesk documentation for more details.

Python

Maya includes an embedded **Python** intepreter. MEL commands are available from Python in the `maya.cmds` Python module, so a command like `polyCube -n "new_cube"` is available in Python as `maya.cmds.polyCube(n='new_cube')`. The listener window includes a Python tab which allows users to enter Python commands interactively.

Maya python can import modules using the python `import` directive. Maya will look for Python files in a number of locations, configured in the Maya application, using environment variable or a `maya.env` file. The Autodesk documentation covers the basics of placing python files where Maya can see and import them.

C++

Maya exposes its API to **C++**. Developers can compile plugins which Maya will recognize at startup.

Developing C++ plugins for Maya requires the Maya Devkit. Download the version appropriate to your platform and follow the included instructions to set up the build environment.

Simple Python example

Open the Maya listener with the button at the lower right corner of the help line. This opens the script listener.

Create a `Python` tab from the tab bar.

Here's a very basic script that will print out the positions of the cameras in a default scene. Enter this into the listener:

```
import maya.cmds as cmds
cameras = cmds.ls(type ='camera')
for each_camera in cameras:
    parent = cmds.listRelatives(each_camera, parent=True)
    position = cmds.xform(parent, q=True, translation=True)
    print each_camera, "is at", position
```

Select the script an execute it with `CTRL+enter`;

Here's another simple example that generates a random collection of cubes. It uses the python `random` module to generate random values.

```
import maya.cmds as cmds
import random

for n in range(25):
    cube, cubeShape = cmds.polyCube()
    x = random.randrange(-50, 50)
    y = random.randrange(-50, 50)
    z = random.randrange(-50, 50)
    cmds.xform(cube, t = (x,y,z))
```

Hello world

Printing "hello world" on several languages on Maya on the console (Script Editor).

MEL

On a MEL tab on the Script Editor, or the command line bar, selecting MEL:

```
print ("hello world");
```

And hit play on the script editor or enter key on the command line.

PYTHON

On a Python tab on the Script Editor, or the command line bar, selecting Python:

```
print "hello world"
```

And hit play on the script editor or enter key on the command line.

Read Getting started with maya online: https://riptutorial.com/maya/topic/7423/getting-started-with-maya

Chapter 2: Basic Maya Commands Explained

Examples

What Is set/get Attr

setAttr

Basically as any other language setAttr can set a value for a specified attribute of a node or any context. And it support very wide range of options. For detailed instructions please visit the official documentation from maya itself here.

Here is a very minimal example of setAttr

```
nodeName = "pSphere1"
cmds.setAttr("%s.tx" % nodeName, 10)
```

getAttr Same as setAttr here it will give back the value from a specific attribute from a node. And it can return multiple types of dataTypes also. Autodesk has well documented the command here

Here is a very minimal example of getAttr

```
nodeName = "pSphere1"
txValue = cmds.getAttr("%s.tx" % nodeName)
```

Basic maya command syntax

Maya commands come in a very small range of forms. Recognizing the form that a command takes is useful for working with new commands.

Simple commands

The most basic form is simply `<command>(<object>)` where is the function you're calling and is the string name of an object you are working with:

```
  cmds.hide('pCube1')
  cmds.delete('nurbsCurve8')
```

Many commands can accept multiple targets. You can pass these individually or as iterables (lists, tuples)

```
  cmds.select("top", "side")
  cameras = ['top', 'side']
  cmds.select(cams)
```

You can Python's star *args to pass an iterable object like a generator to a command:

```
cmds.select(*a_generator_function())
```

A lot of commands take flags which control their behavior. for example

```
cmds.ls(type='mesh')
```

will return a list of meshes, and

```
cmds.ls(type='nurbsCurve')
```

returns a list of nurbs curves.

Commands which take flag can use the Python **kwargs syntax, allowing you to create dictionary of flag-value pairs and pass that to the command:

```
 options = {type: 'mesh'}
 cmds.ls(**options)
```

is the same as

```
 cmds.ls(type='mesh')
```

This can be very useful when assembling a command from a list of options supplied by a user or by script logic

Read Basic Maya Commands Explained online: https://riptutorial.com/maya/topic/7630/basic-maya-commands-explained

Chapter 3: Creating Maya UI

Parameters

parameter	details
e / edit	tells Maya you want to change the value of an existing property
q / query	tells Maya you want to get the value of an existing property

Remarks

Maya comes with a fairly complete UI toolkit that includes windows, layouts, and a variety of controls. This is implemented using the QT framework in C++, but exposed to MEL and Python users via the default Maya command set.

QT

Advanced users can extend the Maya UI using either C++ or Python. Maya versions from 2012 to 2016 use Pyside and QT4; Maya 2017 uses Pyside2 and QT5. More details here

Note: Older reference on the web refers to the Maya GUI toolkit as "ELF"; that's still the correct name but it's rarely used.

Examples

Basic UI example [Python]

The Maya GUI toolkit creates a variety of UI elements in a simple, imperative form. There are basic commands to create and edit GUI widgets; the widgets are identified by a unique string name.

All gui commands take the same basic form: you supply a command type and the string name of the object you want to work on or create, along with flags that specify the look or behavior of the widget. So, for example, to create a button you'd use:

```
cmds.button('my_button', label = 'my label')
```

This will create a new gui button. To edit the button you'd use the same command with the `edit` flag (the short version is just `e`). So you could change the label of the button like this:

```
cmds.button('my_button', e=True, label = 'a different label')
```

and you can query the current value of a property with the `query` or `q` flag:

```
cmds.button(`my button`, q=True, label=True)
# 'a different label'
```

Widget naming

When you create a new widget with a UI command you can supply the name you'd like the new widget to get. However, its **not** guaranteed: Maya will give the button the name you asked for -- if you've given it a character it doesn't recognize or if there is already a widget with the same name you may get back a different name. It *always* a good practice to capture the name of a new widget when it's created to avoid surprises:

```
 my_button = cmds.button('my_button')
 cmds.button(my_button, e=True, label = "a new label")
```

Callback functions

Many widgets include events which can fire callback functions when the user interacts with the widget. For example when a button is pressed, a checkbox checked, or a dropdown chosen you can fire a function.

The exact flag which is associated with these event depends on the widget, but a typical callback would look like this:

```
 def callback_fn(_ignore):
     print "button pressed"

 button = cmds.button(label='press me', command = callback_fn)
```

Pressing the button will print "button pressed" to the listener window. Most widget's fire some arguments when their callbacks activate -- the `button` for example always includes a boolean value -- so you'll need to make sure that the callback handler has the right signature to go with the widget you're using. That's why `callback_fn()` takes an argument even though it doesn't need it.

Callback Assignment

Maya supports two different ways of attaching callback functions:

```
   # this works, but is not a great idea
   cmds.button(label = 'string reference', command = 'string_name_of_function')
   # use this form whenever possible
   cmds.button(label = 'direct function reference', command = callback_fn)
```

In the first example the callback is assigned by a string value. Maya will find the callback in the global Python scope -- which is usually hard to access when writing properly organized code. String-name callbacks are also slower to resolve. The second example passes the actual Python function to the callback -- this form is preferred because it is faster and, if you've failed to provide a valid function to the callback function, you'll know when the UI is created instead of when the UI widgets are actually used.

If you want to pass a argument value to a call back function, you can use a lambda, a closure or a functools.partial bind arguments to the callback.

Using `partial`:

```
from functools import partial
....
def callback_fn(myValue, _ignore):  # _ignore swallows the original button argument
    print myValue

button = cmds.button(label='press me', command = partial(callback_fn, "fooo"))
```

Using `lambda`:

```
def callback_fn(myValue):
    print myValue

button = cmds.button(label='press me', command = lambda _ignore: callback_fn("fooo"))
# here the lambda needs to handle the button argument
```

Using Closures

```
b = cmds.button(label = 'press me')
# by defining this function when `b` exists, we can use it later
# without storing it explicitly
def get_button_label(*_):
    print "label of button", b, " is ", cmds.button(b, q=True, l=True)
cmds.button(b, e=True, c=get_button_label)
```

There' more about string callback names vs. callback function here

Creating a window

```
# create a window with a button that closes the window when clicked
window = cmds.window(title='example window')       # create the window
layout = cmds.columnLayout(adjustableColumn=True)  # add a vertical layout

def close_window(*_):
    cmds.deleteUI(window)                          # deletes the window above

button = cmds.button(label= 'press to close", command = close_window)

# show the window
cmds.showWindow(window)
```

Lambdas and loops

Lambdas are a useful shortcut for hooking up behaviors to GUI elements.

```
b = cmds.button("make a cube", command = lambda _: cmds.polyCube())
```

However, due to the way Python captures variables inside of lambdas, you can get unexpected results if you bind commands using lambdas inside a loop. For example this *looks* like it should produce buttons that create spheres of different sizes:

```
# warning: doesn't work like it looks!
for n in range(5):
    b = cmds.button("sphere size %i" % n, command = lambda _: cmds.polySphere(radius=n))
```

The buttons will be labelled correctly but will all use the same radius (4) because the lambdas will all capture that value when the loop closes. *TLDR:* If you're generating callbacks inside of a loop, use `functools.partial` or another method for capturing values - lambdas don't work for this application. See here for more details

Read Creating Maya UI online: https://riptutorial.com/maya/topic/7627/creating-maya-ui

Chapter 4: Creating PyQt GUI With Maya

Examples

Creating PyQt Window

This is a very basic example how to load a pyqt ui file to maya with pyqt libs. In this solution you really don't need to convert your pyqt ui file a python file. You can simply load your pyqt ui.

```
from PyQt4 import QtCore, QtGui, uic
import maya.OpenMayaUI as mui
import sip

baseUI                      = "/user/foo/bar/basic.ui"
baseUIClass, baseUIWidget = uic.loadUiType(baseUI)

class Ui_MainWindow(baseUIWidget, baseUIClass):
    def __init__(self,parent=None):
        super(baseUIWidget, self).__init__(parent)
        self.setupUi(self)


def getMayaWindow():
    ptr = mui.MQtUtil.mainWindow()
    return sip.wrapinstance(long(ptr), QtCore.QObject)

def mayaMain():
    global maya_basicTest_window
    try:
        maya_basicTest_window.close()
    except:
        pass
    maya_basicTest_window = Ui_MainWindow(getMayaWindow())
    maya_basicTest_window.show()

mayaMain()
```

Creating a PyQt Window By Code

In this example, we trying to create a gui with only through code rather than using any ui file. Its a very basic example you need to extend based on your need."

```
from PyQt4 import QtCore, QtGui
import maya.OpenMayaUI as mui
import sip

class Ui_MainWindow(QtGui.QMainWindow):
    def __init__(self,parent=None):
        QtGui.QMainWindow.__init__(self, parent)

        self.centralwidget = QtGui.QWidget(self)
        self.pushButton = QtGui.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(80, 50, 75, 23))
        self.pushButton_2 = QtGui.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(190, 50, 111, 151))
        self.pushButton_3 = QtGui.QPushButton(self.centralwidget)
        self.pushButton_3.setGeometry(QtCore.QRect(350, 60, 75, 101))
        self.setCentralWidget(self.centralwidget)
        self.menubar = QtGui.QMenuBar(self)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 21))
        self.setMenuBar(self.menubar)
        self.statusbar = QtGui.QStatusBar(self)
        self.setStatusBar(self.statusbar)
        self.retranslateUi()

    def retranslateUi(self):
        self.setWindowTitle("MainWindow")
        self.pushButton.setText("test")
        self.pushButton_2.setText( "test")
        self.pushButton_3.setText("test")

def getMayaWindow():
    ptr = mui.MQtUtil.mainWindow()
    return sip.wrapinstance(long(ptr), QtCore.QObject)

def mayaMain():
    global maya_basicTest_window
    try:
        maya_basicTest_window.close()
    except:
        pass
    maya_basicTest_window = Ui_MainWindow(getMayaWindow())
    maya_basicTest_window.show()

mayaMain()
```

Read Creating PyQt GUI With Maya online: https://riptutorial.com/maya/topic/7629/creating-pyqt-gui-with-maya

Chapter 5: Finding scene objects

Examples

Find objects by name

Use the `ls()` commands to find objects by name:

```
freds = cmds.ls("fred")
#finds all objects in the scene named exactly 'fred', ie [u'fred', u'|group1|fred']
```

Use `*` as a wildcard:

```
freds = cmds.ls("fred*")
# finds all objects whose name starts with 'fred'
# [u'fred', u'frederick', u'fred2']

has_fred = cmds.ls("*fred*")
# [u'fred', u'alfred', u'fredericka']
```

ls() takes multiple filter string arguments:

```
cmds.ls("fred", "barney")
# [u'fred', u'|group1|barney']
```

It can also accept an iterable argument:

```
look_for = ['fred', 'barney']
# cmds.ls(look_for)
# [u'fred', u'|group1|barney']
```

Dealing with ls() results

Using ls() as a filter can sometimes provide produce odd results. If you accidentally forget to pass a filter argument and call `ls()` with no arguments, you will get a list of **every node in the Maya scene**:

```
 cmds.ls()
 # [u'time1', u'sequenceManager1', u'hardwareRenderingGlobals', u'renderPartition'...] etc
```

A common cause of this is using *args inside `ls()`:

```
cmds.ls(["fred", "barney"]) # OK, returns ['fred', 'barney']
cmds.ls([]) # OK, returns []
cmds.ls(*[]) # not ok: returns all nodes!
```

Maya 2015 and and earlier

In Maya 2015 and earlier, an `ls()` query which finds nothing will return `None` instead of an empty list. In case of using the result, it can result in an exception:

```
for item in cmds.ls("don't_exist"):
    print item
# Error: TypeError: file <maya console> line 1: 'NoneType' object is not iterable
```

The cleanest idiom for working around this is always adding an alternative output when None is returned adding `or []` after an `ls()` operation. That will ensure that the return is an empty list rather than `None`:

```
for item in cmds.ls("don't_exist") or []:
    print item
# prints nothing since there's no result -- but no exception
```

Finding objects by type

`ls()` includes a `type` flag, which lets you find scene nodes of a particular type. For example:

```
cameras = cmds.ls(type='camera')
// [u'topShape', u'sideShape', u'perspShape', u'frontShape']
```

You can search for multiple types in the same call:

```
geometry = cmds.ls(type=('mesh', 'nurbsCurve', 'nurbsSurface'))
```

You can also search for 'abstract' types, which correspond to Maya's internal class hierarchy. These to find out what node types a particular object represents, use the `nodeType` command:

```
cmds.nodeType('pCubeShape1', i=True)  # 'i' includes the inherited object types
// Result: [u'containerBase',
 u'entity',
 u'dagNode',
 u'shape',
 u'geometryShape',
 u'deformableShape',
 u'controlPoint',
 u'surfaceShape',
 u'mesh'] //
 # in this example, ls with any of the above types will return `pCubeShape1`
```

Using ls() to see if an object exists

Since `ls()` finds objects by names, it's a handy way to find out if an object is present in the scene. `ls()` with a list of objects will only return the ones which are in the scene.

```
available_characters = cmds.ls('fred', 'barney', 'wilma', 'dino')
# available_characters will contain only the named characters that are present
```

Working with component selections

When working with components, such as vertices or uv points, Maya defaults to returning a colon-separated range rather than individual items:

```
print cmds.ls('pCube1.vtx[*]')  # get all the vertices in the cube
# [u'pCube1.vtx[0:7]']
```

You can use `ls` with the `flatten` option to force Maya to expand the range notation into individual component entries:

```
expanded = cmds.ls('pCube1.vtx[*]', flatten=True)
print expanded
# [u'pCube1.vtx[0]', u'pCube1.vtx[1]', u'pCube1.vtx[2]', u'pCube1.vtx[3]', u'pCube1.vtx[4]',
u'pCube1.vtx[5]', u'pCube1.vtx[6]', u'pCube1.vtx[7]']
```

This form is usually better when looping, since you don't have write any code to turn a string like `pCube1.vtx[0:7]` into multiple individual entries.

You can also get the same result using the `filterExpand` command.

Safely getting a single object from ls

Many `ls()` queries are intended to find a single object, but `ls` always returns a list (or, in older Mayas, a single `None`). This creates complicated error checking for a simple question.

The easiest way to get a single value from an `ls` under any circumstances is

```
result = (cmds.ls('your query here') or [None])[0]
```

The `or` guarantees that at a minimum you'll get a list containing a single `None` so you can always index into it.

Note that this style won't tell you if you've got more than one result -- it just makes it safe to assume a single result.

Read Finding scene objects online: https://riptutorial.com/maya/topic/7564/finding-scene-objects

Chapter 6: Maya Online Video Tutorials

Examples

Online Video Tutorials Available

There are many online video tutorials for maya, python and pyqt. Which will give you access to some in depth knowledge of maya and python programming in maya. Some are covered with pyqt also. Here is some and feel free to add more here.

- https://www.udemy.com/python-for-maya/learn/v4/overview By Dhruv Govil
- https://cmivfx.com/products/316-pyqt4-ui-development-for-maya By Justin
- https://cmivfx.com/products/167-python-introduction-vol-01---maya By Justin
- https://cmivfx.com/products/173-python-for-maya-vol-02 By Justin
- https://www.cgcircuit.com/bundle-details.php?val=21 Math and Maya API

Read Maya Online Video Tutorials online: https://riptutorial.com/maya/topic/7910/maya-online-video-tutorials

Chapter 7: Maya Python Paths

Remarks

This page should cover various ways to set up Maya python paths - userSetup, maya.env, environment variables and so on

Examples

Using userSetup.py

Add arbitrary paths to the Maya Python environment in the `userSetup.py` file. `userSetup.py` is a Python file (**not** a module) which gets automatically executed on Maya startup. `userSetup.py` can live in a number of locations, depending on the os and environment variables.

When Maya starts, it will execute the contents of the userSetup file. Adding Python paths here will allow it to find modules:

```
import sys
sys.path.append("/path/to/my/modules")
```

This will make Python module files in '/path/to/my/modules' available for import using the standard `import` directive.

For more advanced setups, the `site` module can do the same using the `addsitedir()` function. `site.addsitedir()` supports .pth files which configures multiple paths in one go.

For example, three folders of unrelated Python could be arranged like this:

```
 python_files
 |
 +---- studio
 |      +  module1.py
 |      +  module2.py
 |
 +---- external
        |
        +---- paid
        |      + paidmodule.py
        |
        +---- foss
               + freemodule.py
```

Using `sys.path` directly you'd have to add `python_files/studio`, `python_files/external/paid` and `python_files/external/paid` manually. However you could add a .pth file to the root of `python_files` that looked like this:

```
 studio
```

```
external/paid
external/foss
```

and call this in userSetup:

```
import site
site.addsitedir("/path/to/python_files")
```

and you'll get all of the paths in one go.

Using Environment Variables

The Maya Python interpreter works like a regular Python intepreter, so it will use the same environment variables to find importable files as any other Python 2.6 or 2.7 installation (described in more detail in the Python documentation.

If there is no other python installation on your machine you can use the environment variables to point at the location of your Python files for Maya (if you do have another Python, changing these for Maya's sake may interfere with your other Python installation - you'd be better off using a userSetup or startup script). Set variable `PYTHONPATH` so it includes your search paths. If you're editing the variable to include multiple paths remember that on *NIX systems the paths are separated by colons:

```
export PYTHONPATH="/usr/me/maya/shared:/usr/me/other_python"
```

where on Windows they are semicolons:

```
setx  PYTHONPATH C:/users/me/maya;//server/shared/maya_python
```

Multiple configurations

One advantage of using environment variables is that you can quickly re-configure a maya install to load tools and scripts from different locations for different projects. The easiest way to do this is to set the `PYTHONPATH` right before launching Maya so that you inherit the necessary paths for this maya session. For example

```
set PYTHONPATH=C:/users/me/maya;//server/shared/maya_python
maya.exe
```

will launch Maya (on Windows) with the paths `C:/users/me/maya` and `//server/shared/maya_python` available for use. You could launch a second copy of Maya from a new commandline using a different `set` command and the second Maya would use different paths.

Because it's hard for most end-users to type these kinds of things, it's a good idea to automate the process with a batch or shell file that sets the local environment variables and launches maya. *note: we need examples of this for .bat and .sh files* In this system you'd distribute a .bat or .sh file for each project you were supporting and your users would launch maya using those; launching maya without the bat file would revert them to the default Maya configuration without any custom scripts.

Read Maya Python Paths online: https://riptutorial.com/maya/topic/7437/maya-python-paths

Credits

S. No	Chapters	Contributors
1	Getting started with maya	4444, andy, Community, darkgaze, kartikg3, theodox
2	Basic Maya Commands Explained	Achayan, andy, theodox
3	Creating Maya UI	Achayan, RamenChef, theodox
4	Creating PyQt GUI With Maya	Achayan
5	Finding scene objects	darkgaze, theodox
6	Maya Online Video Tutorials	Achayan
7	Maya Python Paths	4444, mnoronha, theodox