EBook Gratis

APRENDIZAJE maya

Free unaffiliated eBook created from
Stack Overflow contributors.

#maya

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: maya

It is an unofficial and free maya ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official maya.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con maya

Observaciones

Esta documentación cubre la codificación de Autodesk Maya . **No** es la intención de los usuarios finales del software Maya. (Para saber cómo modelar o animar en Maya, pruebe los videos introductorios de Autodesk o un sitio de usuario final como CGSociety).

Idiomas

Maya admite 3 lenguajes de programación: MEL, su lenguaje de secuencias de comandos incorporado; C ++, que se utiliza para complementos; y Python, que es común para los trabajos de integración, pero también puede crear complementos utilizando una versión envuelta de la API de C ++

Examples

Instalación

Maya soporta 3 entornos de programación principales. Cada uno tiene diferentes requisitos de configuración.

MEL

El lenguaje de scripting **MEL** se incluye con la aplicación Maya. Habilitado de forma predeterminada, los usuarios pueden probar MEL en la ventana de escucha del script en una copia en ejecución de Maya.

Los archivos MEL son archivos de texto con la extensión `.mel` . Se pueden cargar en una sesión Maya en ejecución usando el comando `source` en el escucha o en otro script MEL. Maya mantiene una lista de directorios de origen y buscará un script MEL solicitado en todos los directorios hasta que encuentre un archivo con el nombre apropiado.

Hay varios métodos para configurar la ruta del script; Consulte la documentación de Autodesk para más detalles.

Pitón

Maya incluye un intérprete **Python** incrustado. Los comandos MEL están disponibles desde Python en el módulo `maya.cmds` Python, por lo que un comando como `polyCube -n "new_cube"` está disponible en Python como `maya.cmds.polyCube(n='new_cube')` . La ventana de escucha incluye una pestaña de Python que permite a los usuarios ingresar comandos de Python de forma interactiva.

Maya Python puede importar módulos usando la directiva de `import` Python. Maya buscará

archivos de Python en varias ubicaciones, configuradas en la aplicación Maya, usando una variable de entorno o un archivo `maya.env` . La documentación de Autodesk cubre los aspectos básicos de la colocación de archivos de Python donde Maya puede verlos e importarlos.

C ++

Maya expone su API a **C ++** . Los desarrolladores pueden compilar complementos que Maya reconocerá en el inicio.

El desarrollo de complementos de C ++ para Maya requiere el Maya Devkit . Descargue la versión adecuada para su plataforma y siga las instrucciones incluidas para configurar el entorno de compilación.

Ejemplo de Python simple

Abra el oyente Maya con el botón en la esquina inferior derecha de la línea de ayuda. Esto abre la escucha del script.

Crea una pestaña de `Python` desde la barra de pestañas.

Aquí hay un script muy básico que imprimirá las posiciones de las cámaras en una escena predeterminada. Ingrese esto en el oyente:

```
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
```

Seleccione el script y ejecútelo con `CTRL+enter` ;

Aquí hay otro ejemplo simple que genera una colección aleatoria de cubos. Utiliza el módulo `random` python para generar valores aleatorios.

```
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))
```

Hola Mundo

Imprimiendo "hola mundo" en varios idiomas en Maya en la consola (Script Editor).

MEL

En una pestaña MEL en el Editor de secuencias de comandos, o en la barra de línea de comando, seleccionando MEL:

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

Y pulsa play en el editor de scripts o introduce la tecla en la línea de comandos.

PITÓN

En una pestaña de Python en el Editor de secuencias de comandos, o en la barra de línea de comando, seleccionando Python:

```
print "hello world"
```

Y pulsa play en el editor de scripts o introduce la tecla en la línea de comandos.

Lea Empezando con maya en línea: https://riptutorial.com/es/maya/topic/7423/empezando-con-maya

Capítulo 2: Caminos de Python Maya

Observaciones

Esta página debe cubrir varias formas de configurar rutas de Python Maya: userSetup, maya.env, variables de entorno, etc.

Examples

Utilizando userSetup.py

Agregue rutas arbitrarias al entorno Maya Python en el archivo `userSetup.py` . `userSetup.py` es un archivo Python (**no** un módulo) que se ejecuta automáticamente en el inicio de Maya. `userSetup.py` puede vivir en varias ubicaciones, dependiendo de la os y las variables de entorno.

Cuando Maya se inicie, ejecutará el contenido del archivo de configuración de usuario. Agregar rutas de Python aquí le permitirá encontrar módulos:

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

Esto hará que los archivos del módulo Python en '/ path / to / my / modules' estén disponibles para importarlos usando la directiva de `import` estándar.

Para configuraciones más avanzadas, el módulo del `site` puede hacer lo mismo usando la función `addsitedir()` . `site.addsitedir()` admite archivos .pth que configuran múltiples rutas de una sola vez.

Por ejemplo, tres carpetas de Python no relacionadas podrían organizarse así:

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

Usando `sys.path` directamente, tendría que agregar `python_files/studio` , `python_files/external/paid` y `python_files/external/paid` manualmente. Sin embargo, podría agregar un archivo .pth a la raíz de los `python_files` que se parecían a esto:

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

y llame a esto en userSetup:

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

y obtendrás todos los caminos de una sola vez.

Uso de variables de entorno

El intérprete de Maya Python funciona como un intérprete regular de Python, por lo que utilizará las mismas variables de entorno para encontrar archivos importantes como cualquier otra instalación de Python 2.6 o 2.7 (se describe con más detalle en la documentación de Python) .

Si no hay otra instalación de Python en su máquina, puede usar las variables de entorno para apuntar a la ubicación de sus archivos de Python para Maya (si tiene otro Python, cambiarlas por el bien de Maya puede interferir con su otra instalación de Python, usted ' Sería mejor utilizar un script de inicio de usuario o de inicio). Establezca la variable `PYTHONPATH` para que incluya sus rutas de búsqueda. Si está editando la variable para incluir varias rutas, recuerde que en los sistemas * NIX, las rutas están separadas por dos puntos:

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

donde en Windows hay punto y coma:

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

Configuraciones múltiples

Una de las ventajas de usar variables de entorno es que puede volver a configurar rápidamente una instalación de maya para cargar herramientas y scripts desde diferentes ubicaciones para diferentes proyectos. La forma más fácil de hacer esto es configurar `PYTHONPATH` justo antes de iniciar Maya para que herede las rutas necesarias para esta sesión de maya. Por ejemplo

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

lanzará Maya (en Windows) con las rutas `C:/users/me/maya` y `//server/shared/maya_python` disponibles para su uso. Podría lanzar una segunda copia de Maya desde una nueva línea de comando usando un comando `set` diferente y la segunda Maya usaría diferentes rutas.

Debido a que es difícil para la mayoría de los usuarios finales escribir este tipo de cosas, es una buena idea automatizar el proceso con un archivo por lotes o shell que establece las variables de entorno locales y lanza maya. *nota: necesitamos ejemplos de esto para los archivos .bat y .sh* En

este sistema, distribuiría un archivo .bat o .sh para cada proyecto que estaba respaldando y sus usuarios iniciarían maya utilizando esos; el lanzamiento de maya sin el archivo bat los revertiría a la configuración predeterminada de Maya sin ningún script personalizado.

Lea Caminos de Python Maya en línea: https://riptutorial.com/es/maya/topic/7437/caminos-de-python-maya

Capítulo 3: Comandos básicos mayas explicados

Examples

Lo que se establece / obtener Attr

setAttr

Básicamente, como cualquier otro idioma, setAttr puede establecer un valor para un atributo específico de un nodo o cualquier contexto. Y soporta muy amplia gama de opciones. Para obtener instrucciones detalladas, visite la documentación oficial de maya aquí .

Aquí hay un ejemplo muy mínimo de setAttr

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

getAttr Igual que setAttr aquí devolverá el valor de un atributo específico de un nodo. Y puede devolver múltiples tipos de tipos de datos también. Autodesk ha documentado bien el comando aquí.

Aquí hay un ejemplo muy mínimo de getAttr

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

Sintaxis básica del comando maya

Los comandos mayas vienen en un rango muy pequeño de formas. Reconocer la forma que toma un comando es útil para trabajar con nuevos comandos.

Comandos simples

La forma más básica es simplemente `<command>(<object>)` donde está la función a la que está llamando y es el nombre de la cadena de un objeto con el que está trabajando:

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

Muchos comandos pueden aceptar múltiples objetivos. Puede pasar estos individualmente o como iterables (listas, tuplas)

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

```
cmds.select(cams)
```

Puede Python's star * args para pasar un objeto iterable como un generador a un comando:

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

Una gran cantidad de comandos toman banderas que controlan su comportamiento. por ejemplo

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

devolverá una lista de mallas, y

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

Devuelve una lista de curvas de nurbs.

Los comandos que toman la bandera pueden usar la sintaxis de kwargs de Python **, lo que le permite crear un diccionario de pares de valores de bandera y pasarlo al comando:

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

es lo mismo que

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

Esto puede ser muy útil cuando se ensambla un comando a partir de una lista de opciones proporcionadas por un usuario o por una lógica de script

Lea Comandos básicos mayas explicados en línea:
https://riptutorial.com/es/maya/topic/7630/comandos-basicos-mayas-explicados

Capítulo 4: Creando PyQt GUI con Maya

Examples

Creando la ventana de PyQt

Este es un ejemplo muy básico de cómo cargar un archivo ui pyqt a maya con libs pyqt. En esta solución, realmente no necesita convertir su archivo pyqt ui en un archivo python. Simplemente puede cargar su 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()
```

Creando una ventana de PyQt por código

En este ejemplo, intentamos crear una interfaz gráfica de usuario con solo código a través de un archivo ui. Es un ejemplo muy básico que necesita ampliar según su necesidad ".

```
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()
```

Lea Creando PyQt GUI con Maya en línea: https://riptutorial.com/es/maya/topic/7629/creando-pyqt-gui-con-maya

Capítulo 5: Crear Maya UI

Parámetros

parámetro	detalles
e / editar	le dice a Maya que desea cambiar el valor de una propiedad existente
q / consulta	le dice a Maya que desea obtener el valor de una propiedad existente

Observaciones

Maya incluye un kit de herramientas de IU bastante completo que incluye ventanas, diseños y una variedad de controles. Esto se implementa utilizando el marco QT en C ++, pero expuesto a los usuarios de MEL y Python a través del conjunto de comandos predeterminado de Maya.

QT

Los usuarios avanzados pueden extender la interfaz de usuario Maya utilizando C ++ o Python. Las versiones mayas de 2012 a 2016 usan Pyside y QT4; Maya 2017 utiliza Pyside2 y QT5. Más detalles aquí

Nota: las referencias más antiguas en la web se refieren al kit de herramientas de GUI Maya como "ELF"; Ese sigue siendo el nombre correcto, pero rara vez se usa.

Examples

Ejemplo de interfaz de usuario básica [Python]

El kit de herramientas de GUI de Maya crea una variedad de elementos de UI en una forma simple e imperativa. Hay comandos básicos para crear y editar widgets GUI; los widgets se identifican por un nombre de cadena único.

Todos los comandos gui adoptan la misma forma básica: usted proporciona un tipo de comando y el nombre de la cadena del objeto en el que desea trabajar o crear, junto con las banderas que especifican el aspecto o el comportamiento del widget. Entonces, por ejemplo, para crear un botón usarías:

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

Esto creará un nuevo botón gui. Para editar el botón, se usaría el mismo comando con la `edit` (la versión corta es solo `e`). Así que podrías cambiar la etiqueta del botón así:

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

y puede consultar el valor actual de una propiedad con la `query` o el indicador `q` :

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

Nombramiento de widgets

Cuando crea un nuevo widget con un comando de UI, puede proporcionar el nombre que desea que obtenga el nuevo widget. Sin embargo, **no está** garantizado: Maya le dará al botón el nombre que solicitó; si le ha dado un carácter que no reconoce o si ya hay un widget con el mismo nombre, puede recuperar un nombre diferente. *Siempre es* una buena práctica capturar el nombre de un nuevo widget cuando se crea para evitar sorpresas:

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

Funciones de devolución de llamada

Muchos widgets incluyen eventos que pueden activar funciones de devolución de llamada cuando el usuario interactúa con el widget. Por ejemplo, cuando se presiona un botón, se marca una casilla de verificación o se selecciona un menú desplegable, puede activar una función.

El indicador exacto que se asocia con estos eventos depende del widget, pero una devolución de llamada típica se vería así:

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

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

Al presionar el botón se imprimirá "botón presionado" en la ventana del oyente. La mayoría de los widgets activan algunos argumentos cuando sus devoluciones de llamada se activan (el `button` por ejemplo, siempre incluye un valor booleano), por lo que deberá asegurarse de que el controlador de devolución de llamada tenga la firma correcta para ir con el widget que está usando. Es por eso que `callback_fn()` toma un argumento a pesar de que no lo necesita.

Asignación de devolución de llamada

Maya admite dos formas diferentes de adjuntar funciones de devolución de llamada:

```
  # 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)
```

En el primer ejemplo, la devolución de llamada se asigna mediante un valor de cadena. Maya

encontrará la devolución de llamada en el ámbito global de Python, que generalmente es difícil de acceder cuando se escribe un código adecuadamente organizado. Las devoluciones de llamada de nombre de cadena también son más lentas de resolver. El segundo ejemplo pasa la función Python real a la devolución de llamada: se prefiere este formulario porque es más rápido y, si no ha proporcionado una función válida a la función de devolución de llamada, sabrá cuándo se creará la interfaz de usuario en lugar de cuándo los widgets de la interfaz de usuario se utilizan realmente.

Si desea pasar un valor de argumento a una función de devolución de llamada, puede usar un lambda , un cierre o un argumento functools.partial vincular a la devolución de llamada.

Usando `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"))
```

Usando `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
```

Utilizando cierres

```
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)
```

Hay más información sobre los nombres de devolución de llamada de cadena frente a la función de devolución de llamada aquí.

Creando una ventana

```
# 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 y bucles

Lambdas es un atajo útil para conectar comportamientos a elementos GUI.

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

Sin embargo, debido a la forma en que Python captura variables dentro de lambdas, puede obtener resultados inesperados si vincula comandos utilizando lambdas dentro de un bucle. Por ejemplo, esto *parece* que debería producir botones que crean esferas de diferentes tamaños:

```
# 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))
```

Los botones se etiquetarán correctamente pero todos usarán el mismo radio (4) porque las lambdas capturarán ese valor cuando se cierre el ciclo. *TLDR:* si está generando devoluciones de llamadas dentro de un bucle, use `functools.partial` u otro método para capturar valores: las lambdas no funcionan para esta aplicación. Vea aquí para más detalles

Lea Crear Maya UI en línea: https://riptutorial.com/es/maya/topic/7627/crear-maya-ui

Capítulo 6: Encontrar objetos de escena

Examples

Encuentra objetos por nombre

Use los comandos `ls()` para encontrar objetos por nombre:

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

Use `*` como un comodín:

```
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 () toma múltiples argumentos de cadena de filtro:

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

También puede aceptar un argumento iterable:

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

Tratando con ls () resultados

El uso de ls () como filtro a veces puede proporcionar resultados impares. Si accidentalmente olvida pasar un argumento de filtro y llama a `ls()` sin argumentos, obtendrá una lista de **cada nodo en la escena Maya** :

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

Una causa común de esto es usar * args dentro de `ls()` :

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

Maya 2015 y anteriores

En Maya 2015 y anteriores, una consulta `ls()` que no encuentra nada devolverá `None` lugar de una lista vacía. En caso de usar el resultado, puede resultar en una excepción:

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

El lenguaje más limpio para solucionar esto siempre es agregar una salida alternativa cuando se devuelve Ninguno agregando `or []` después de una operación `ls()` . Eso asegurará que la devolución sea una lista vacía en lugar de `None` :

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

Encontrar objetos por tipo

`ls()` incluye un indicador de `type` , que le permite encontrar nodos de escena de un tipo en particular. Por ejemplo:

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

Puede buscar varios tipos en la misma llamada:

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

También puede buscar tipos "abstractos", que correspondan a la jerarquía interna de clases de Maya. Para averiguar qué tipos de nodo representa un objeto en particular, use el comando `nodeType` :

```
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`
```

Usando ls () para ver si un objeto existe

Como `ls()` encuentra los objetos por sus nombres, es una forma útil de averiguar si un objeto

está presente en la escena. `ls()` con una lista de objetos solo devolverá los que están en la escena.

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

Trabajar con selecciones de componentes

Cuando se trabaja con componentes, como vértices o puntos uv, Maya devuelve un rango separado por dos puntos en lugar de elementos individuales:

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

Puede usar `ls` con la opción `flatten` para forzar a Maya a expandir la notación de rango en entradas de componentes individuales:

```
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]']
```

Esta forma suele ser mejor cuando se realiza un bucle, ya que no tiene que escribir ningún código para convertir una cadena como `pCube1.vtx[0:7]` en varias entradas individuales.

También puede obtener el mismo resultado utilizando el comando `filterExpand` .

Obtención segura de un solo objeto desde ls.

Muchos `ls()` consultas están destinadas a encontrar un solo objeto, pero `ls` siempre devuelve una lista (o, en mayas más antiguos, un único `None`). Esto crea una comprobación de errores complicada para una pregunta simple.

La forma más fácil de obtener un valor único de un `ls` bajo cualquier circunstancia es

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

El `or` garantiza que, como mínimo, obtendrá una lista que contenga una sola `None` para que siempre pueda indexarla.

Tenga en cuenta que este estilo no le dirá si tiene más de un resultado, solo hace que sea seguro asumir un solo resultado.

Lea Encontrar objetos de escena en línea: https://riptutorial.com/es/maya/topic/7564/encontrar-objetos-de-escena

Capítulo 7: Tutoriales de video en línea de Maya

Examples

Video tutoriales en línea disponibles

Hay muchos videos tutoriales en línea para maya, python y pyqt. Lo que le dará acceso a un conocimiento profundo de la programación de maya y python en maya. Algunos están cubiertos con pyqt también. Aquí hay algunos y siéntase libre de agregar más aquí.

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

Lea Tutoriales de video en línea de Maya en línea:
https://riptutorial.com/es/maya/topic/7910/tutoriales-de-video-en-linea-de-maya

Creditos

S. No	Capítulos	Contributors
1	Empezando con maya	4444, andy, Community, darkgaze, kartikg3, theodox
2	Caminos de Python Maya	4444, mnoronha, theodox
3	Comandos básicos mayas explicados	Achayan, andy, theodox
4	Creando PyQt GUI con Maya	Achayan
5	Crear Maya UI	Achayan, RamenChef, theodox
6	Encontrar objetos de escena	darkgaze, theodox
7	Tutoriales de video en línea de Maya	Achayan