Autodesk

Autodesk Developer Network

Autodesk Maya Python API Training Agenda and Workshops Handout

Instructor: Kristine Middlemiss, Autodesk Developer Network
(kristine.middlemiss@autodesk.com)

Training Description

Maya has been used by the world's best animation studios and companies ever since it came into existence over ten years ago. As one of the most powerful 3d animation applications, it provides quite flexible architecture so that you are free to mold and change Maya to fit your particular production environment and workflow, through Maya's Application Programming Interface (API). Maya API increase Maya's power, customizing and extending Maya in many ways that you never thought possible. This training will be an intensive journey to explore Python and the Maya API through a top-down approach. We will first look at Maya's fundamental concepts and internal design architecture then goes step-by-step into details on Maya API key components individually. There will also be hands-on labs for you to play around with sample plug-ins and get your feet wet in Maya plug-in development field.

Level of expectation

This course is a step-by-step training transitioning from novice level to intermediate level. The target audience preferably would be people who are becoming familiar with programming in Python, but are new to Maya plug-in development. People with basic knowledge of programming language such as Python and want to develop Python Maya plug-ins can also benefit from this API training.

Agenda

Week 1: We will be focusing on the Maya's own particular design philosophy to give you a better understanding of what's going on under the hood. We will cover basics of getting you started to build a plug-in from scratch, and then we will dwell into Maya's most fundamental concepts, Dependency Graph Nodes and Commands.

Week 2: Will be a reinforcement and extension of what we have learned in week 1. We will also introduce Maya Dependency Graph's push-pull mechanism. In addition, we will be covering some of the caveats and gotcha for dependency graph programming in Maya. We will also be looking at more advanced topics such as Maya callback systems, and miscellaneous tools to help improving your workflow.

Time Specific Agenda

Week 1:
Day 1 – Maya Programming Introduction
Day 2 – API Introduction
Day 3 – API Classes and Command Introduction
Day 4 – Maya Commands In-depth
Day 5 – Maya Nodes

Week 2:
Day 6 – Maya Node Attributes
Day 7 – Closer Look at Dependency Graph
Day 8 – Miscellaneous Tools
Day 9 – Locators
Day 10 – Manipulators (tentative) – Python 2.0 and Some Python Gotchas

Autodesk

Autodesk Developer Network

Python Workshop Handout

For every plug-in, we will provide an "Exercise" folder and a "Solution" folder with the corresponding Python files.

Solution folder includes the complete code for you to finish the plug-in; it is for you to reference when you are stuck at problems when adding code into "Exercise" folder.

In the "Exercise" folder, all the code you need to finish is specified with comments "#- TODO" you need to search for all the "TODO" comments and add your code there.

Module 1

Topics Covered

- How to find API Docs
- Creating Your Plug-in Folder
- Safely reloading a plug-in without restarting Maya

Background

- Download the API Documentation from here:
  - http://www.autodesk.com/me-sdk-docs

  or

- View the API Documentation online here (sub in the xx for the version you are using)
  - http://www.autodesk.com/maya-sdkdoc-20xx-enu

- Creating Your Plug-in Folder

  1. Browse to the folder:
     C:\Users\middlek\Documents\maya\20xx-x64
  2. Create a folder called 'plug-ins', this is where you will put all your exercise Python plug-ins, this way the location will automatically get picked up in Maya path.
  3. Optionally you could create a Maya.env file and insert the line:
     MAYA_PLUG_IN_PATH=C:\MayaAPITraining\plug-ins

- **Safely reloading a plug-in without restarting Maya**

  1. Clear the scene:
     File > New Scene, or

     ```
     import maya.cmds as cmds
     cmds.file(f=True,  new=True)
     ```

  2. Clear the undo queue:

     ```
     import maya.cmds as cmds
     cmds.flushUndo()
     ```

  3. Unload the plug-in:
     Window > Settings/Preferences > Plug-in Manager, uncheck 'Loaded', or

     ```
     import maya.cmds as cmds
     cmds.unloadPlugin("<plugin_name.py>")
     ```

  4. Reload the plug-in:
     Window > Settings/Preferences > Plug-in Manager, check 'Loaded', or

     ```
     import maya.cmds as cmds
     cmds.loadPlugin("<plugin_name.py>")
     ```

Module 2

Topics Covered

- Implement a simple custom Python command "spHelloWorld".
- It demonstrates the skeleton code implementation for a Python Plug-ins; in this case we will inherit from the class MPxCommand.
- In this plug-in, we will create a custom command "spHelloWorld" in python, which prints out "Hello World!" in the Script Editor window.

helloWorldCmd Plug-in

- Exercises
  1. Go to "1_helloWorldCmd\Exercise - py" folder, open helloWorldCmd.py, the skeleton of the command is already there.
  2. Add corresponding code into the skeleton to make it work, relevant classes and commands:

     ```
     import sys
     import maya.OpenMaya as OpenMaya
     import maya.OpenMayaMPx as OpenMayaMPx
     def __init__(self)
     ```

```
def doIt(self,argList)
OpenMayaMPx.asMPxPtr()
```

- Result

  In the Maya Script Editor > Python tab, execute the following line:

  ```
  import maya,cmds as cmds
  cmds.loadPlugin("helloWorldCmd.py")
  cmds.spHelloWorld()
  ```

  You should see "Hello World!" printed out in the Script Editor

Module 3

Topics Covered

- Write a custom command "dagInfoCmd" with MPxCommand class
- Retrieve instancing information and dag path with MFnDagNode
- Have a better understanding on DAG hierarchy by getting inclusive and exclusive matrix of dag node

dagInfoCmd Plug-in

- Overview

  In this exercise, we will implement a custom command dagInfoCmd. For all the selected DAG nodes in the scene, it will print out the instance information, dag path and also inclusive and exclusive matrix.

- Exercises
  1. Go to "1_dagInfoCmd\Exercise - Py" folder, open dagInfoCmd.py, the skeleton of the command is already there.
  2. Fill in the TODOs, relevant classes and methods:

     ```
     MGlobal::getActiveSelectionList()
     MItSelectionList::getDependNode()
     MFnDagNode::instanceCount()
     MDagPathArray:: getAllPaths()
     MDagPath:: fullPathName()
     MDagPath:: exclusiveMatrix(), MDagPath:: inclusiveMatrix()
     MObject::hasFn()
     MFnTransform::transformation()
     ```

  3. Implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the dagInfoCmd command, relevant classes and methods:

     ```
     MFnPlugin::registerCommand()
     MFnPlugin::deregisterCommand()
     ```

- Result
  Open a scene file with multiple instances, for example, multiInstance.ma (provided), select the shape node which has multiple instances, and execute:

  ```
  import maya.cmds as cmds
  cmds.dagInfoCmd()
  ```

  The number of instances, exclusive matrix and inclusive matrix of this shape node will be printed out into the Script Editor. If you select a transform node, it will also print out in to the Script Editor the local transformation matrix of this transform node.

Module 4

Topics Covered

- Write a custom command "nodeInfoCmd" with MPxCommand class from scratch
- Implement flag argument for MPxCommand with MSyntax class
- Retrieve node type information and plug information with MFnDependencyNode and MPlug

nodeInfoCmd Plug-in

- Overview

  In this exercise, we will implement a custom command nodeInfoCmd, with an option to specify a flag "-quiet". By default, for all the selected nodes in the scene, it will print out all the node types and connected plugs information and also the node which is connecting to this selected node as a source. If "-quiet" is provided by user, then nothing will be printed out.

- Exercises
  1. Go to "1_nodeInfoCmd\Exercise - Py" folder, open nodeInfoCmd.py, the skeleton of the command is already there.
  2. Implement nodeInfoCmd.py, add necessary function declaration, relevant classes and methods:

     ```
     MPxCommand::doIt()
     MSyntax::addFlag(), MArgDatabase::isFlagSet()
     MGlobal::getActiveSelectionList()
     MItSelectionList::getDependNode()
     MFnDependencyNode::MFnDependencyNode()
     MFnDependencyNode::getConnections()
     MPlug::connectedTo()
     ```

  3. In nodeInfoCmd.py, implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the nodeInfoCmd command, relevant classes and methods:

MFnPlugin:: registerCommand()
MFnPlugin:: deregisterCommand()

- Result

  In any Maya scene, select a node that has connections and in Script Editor, execute:

  ```
  import maya.cmds as cmds
  cmds.nodeInfoCmd ()
  ```

  In the Script Editor the node type and all the connected plugs information will be printed out and also the nodes which is the source of the connection. Now test with the quiet flag:

  ```
  import maya.cmds as cmds
  cmds.nodeInfoCmd (quiet=True)
  ```

  You should now see nothing printed out in the Script Editor.

instanceRotateCmd Plug-in

- Overview

  In this exercise, we will implement a custom command nodeInfoCmd, with an option to specify a flag "-quiet". By default, for all the selected nodes in the scene, it will print out all the node types and connected plugs information and also the node which is connecting to this selected node as a source. If "-quiet" is provided by user, then nothing will be printed out.

- Exercises

4. Go to "1_nodeInfoCmd\Exercise - Py" folder, open nodeInfoCmd.py, the skeleton of the command is already there.
5. Implement nodeInfoCmd.py, add necessary function declaration, relevant classes and methods:

   MPxCommand::doIt()
   MSyntax::addFlag(), MArgDatabase::isFlagSet()
   MGlobal::getActiveSelectionList()
   MItSelectionList::getDependNode()
   MFnDependencyNode::MFnDependencyNode()
   MFnDependencyNode::getConnections()
   MPlug::connectedTo()

6. In nodeInfoCmd.py, implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the nodeInfoCmd command, relevant classes and methods:

MFnPlugin:: registerCommand()
MFnPlugin:: deregisterCommand()

- Result

  In any Maya scene, select a node that has connections and in Script Editor, execute:

  ```
  import maya.cmds as cmds
  cmds.nodeInfoCmd ()
  ```

  In the Script Editor the node type and all the connected plugs information will be printed out and also the nodes which is the source of the connection. Now test with the quiet flag:

  ```
  import maya.cmds as cmds
  cmds.nodeInfoCmd (quiet=True)
  ```

  You should now see nothing printed out in the Script Editor.

Module 5

Topics Covered

- Build Maya Custom Node with MPxNode

simpleNode Plug-in

- Topics Covered
  - Write a skeleton of a custom node "simpleNode" with MPxNode class from scratch
  - Add simple attribute using MFnNumericAttribute class
- Overview
  - In this exercise, we will implement a custom node simpleNode, it has two attributes: "input", "output"
    Whenever the "input" attribute changes value, the "output" attribute will always be the "input" attribute value multiplied by 2.
- Exercises
  a. 1.Assuming you already copied the whole Plug-in under C:\MayaAPITraining, go to C:\MayaAPITraining\simpleNode\Exercise\
  b. 2. Double click on simpleNode.py to open the Plug-in, the skeleton of the simpleNode has already been provided.
  c. 3. In simpleNode.py, adding declaration of "output" attributes and also declare your unique node ID

d. 4. In simpleNode.py, implement functions that are declared in simpleNode.py.

Relevant classes and methods:

MFnNumericAttribute::create()
MPxNode::attributeAffects(), MPxNode::addAttribute()
MDataBlock::outputValue(), MDataBlock::setClean()
MDataHandle::set()

e. 5. In simpleNode.py, implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the simpleNode node.

Relevant classes and methods:

MFnPlugin:: registerNode()
MFnPlugin:: deregisterNode()

- Result

  In script editor, execute:
  createNode simpleNode;
  Open "Attribute Editor", you will see there are two attributes listed, "input", "output". If you change "input" value, "output" value will be 2* input.

sineNode Plug-in

- Topics Covered

  Implement a custom python node "spSineNode". It demonstrates the skeleton code implementation for a python node plug-in with MPxNode.

- Overview

  In this Plug-in, we will create a custom node "spSineNode" in python, this node has an input attribute "input" and an output attribute "output", the "output" is a sine calculation of "input".

- Exercises

  1. Go to "sineNode python Plug-in\Exercise" folder, open sineNode.py, the skeleton of the node is already there.
  2. Add corresponding code into the skeleton to make it work.

     Relevant Classes and commands:

```
import sys,math
import maya.OpenMaya as OpenMaya
import maya.OpenMayaMPx as OpenMayaMPx
OpenMaya.MObject()
OpenMayaMPx.asMPxPtr()
def compute(self,plug,dataBlock)
```

- Result

  In Maya, load the plug-in "sineNode.py", execute the following script:

```
polySphere;
createNode spSineNode -n sine1;
connectAttr time1.outTime sine1.input;
connectAttr time1.outTime pSphere1.translateX;
connectAttr sine1.output pSphere1.translateY;
```

  You will find a polySphere created and move along X axis as time goes on.

transCircleNode Plug-in

- Topics Covered
  - Write a custom node “transCircleNode” with MPxNode class
  - Add compound attribute using MFnNumericAttribute
  - Implement functions of transCircleNode to achieve the functionality so that the output attribute’s value is the value of inputTranslate plus the value of a circular movement based on current time frame.
- Overview
  - In this exercise, we will implement a custom node transCircleNode, it takes in two input attributes and one output:

    A compound input translate attribute “inputTranslate”, composes of three elements: translateX, translateY, and translateZ
    An input attribute “input”: current time
    An input attribute “frames”: rotating speed (frames per circle)
    An input attribute “scale”: decides the radius of the circle
    An output a translate attribute “outputTranslate”, the value of outputTranslate is the value of inputTranslate plus the value of a circular movement based on current time frame.
- Exercises
  1. Double click on “transCircleNode.py” to open the Plug-in, the skeleton of the transCircleNode has already been provided.
  2. Implement transCircleNode.py, declare output attributes
  3. Implement transCircleNode.py
     i. Relevant classes and methods:
        1. MFnCompoundAttribute::create()
           MFnCompoundAttribute::addChild()
        2. MDataBlock::inputValue(), MDataBlock::outputValue()
  4. Put AEtransCircleTemplate.mel to
     i. C:\My Documents\maya\20xx\prefs\scripts

- Result:

  Open a new scene, execute the following script in Script Editor:

```
createNode transCircle -n circleNode1;
sphere -n sphere1 -r 1;
sphere -n sphere2 -r 2;
connectAttr sphere2.translate circleNode1.inputTranslate;
connectAttr circleNode1.outputTranslate sphere1.translate;
connectAttr time1.outTime circleNode1.input;
```

  You will see two nurbs sphere created and once you hit the “play” button, one sphere is rotating around the bigger sphere; you can also set the radius of the circle, and rotating speed by setting the values in attribute editor of circleNode1.

Module 6

Topics Covered

- Dynamic attribute on MPxNode

dynNode Plug-in

- Topics Covered
  - Create dynamic attribute using MFnNumericAttribute and add it in MPxNode::postConstructor()
  - Set up affecting relationship between dynamic attribute and general attribute
- Overview
  - In this exercise, custom node “dynNode” has two attributes: “input” and “output”. We will add a dynamic attribute “dynAttr” on this class, also set up the affecting relationship so that the value of “output” is the sum of “input” and “dynAttr”.
- Exercises
  - 1. Double click on “dynNode.py” to open the Plug-in, the skeleton of the dynNode has already been provided.
  - 2. Implement dynNode.py, add necessary function declaration
    Relevant classes and methods:
    MPxNode::postConstructor()
    MPxNode::setDependentDirty()
  - 3. Implement dynNode.py, create a dynamic attribute, set up affecting relationship between it and output attribute, also in compute(), set up so that output = input + dynAttr
    Relevant classes and methods:
    MFnNumericAttr::create()
    MPxNode:: thisMObject()
    MFnDependencyNode::addAttribute()
    MDataBlock::inputValue()
    MDataHandle::asFloat()
- Result
  In script editor, execute:
  createNode dynNode;
  Open “Attribute Editor”, you will see there are three attributes listed, “input”, “output” and “dynAttr”. If you change “dynAttr” value, and refresh the attribute editor, you will see “output” value will be the value of “input” plus the value of “dynAttr”.

Module 7

retrieveWeight Plug-in

- Topics Covered

Implement a custom command, which retrieves information from multi attribute "weight" of a blendShape node with class MPlug. It demonstrates how to traverse the element plugs and get value from array plug outside of a node.

- Overview

  In this Plug-in, we will create a custom command "retrieveWeight", it searches attribute "weight" on blendShape node and since it is a multi-attribute, it prints out the number of elements in this array attribute and traverse the array to print out plug data on every element.
- Exercises

  f. 1. Double click on retrieveWeight.py, the skeleton of the retrieveWeightCmd has already been provided.

  g. 2. Implement retrieveWeightCmd.py, add necessary function declaration

  h. 3. Implement retrieveWeightCmd.py, in doIt() function, get a hold of "blendShape" in the scene and find out "weight" on this node and print out all multi attribute information on this node.

  Relevant classes and methods:

  MFnDependencyNode::findPlug()
  MPlug::isArray(), MPlug::numElements()
  MPlug::elementByPhysicalIndex(), MPlug::logicalIndex()
  MPlug::getValue()

  i. 4. In retrieveWeightCmd.py, implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the retrieveWeight command.

  Relevant classes and methods:

  MFnPlugin::registerCommand()
  MFnPlugin::deregisterCommand()
- Result

  Open scene file blendShape.mb. Find the blendShape1 node and select it, then execute "retrieveWeight". It will print out the array attribute name, how many elements this array has, the physical and logical index of all the elements in the array and the weight value of each target shape stored in every element of the array.

Module 8

Topics Covered

- Manipulate Dependency Graph through MDGModifier class
- Implement scene callback functions with MSceneMessage class

setUpTransCircle Plug-in

- Topics Covered
  - Implement a custom command and use MDGModifier to manipulate Dependency Graph (such as adding nodes, building connections)
  - Support undo/redo
- Overview

- In this exercise, we will create a custom command, and simulate the same functionality of the MEL operations we used in Lab 1 "transCircleNode" Plug-in, which set up the transCircle node.
  Here are the commands you need to simluate:

  ```
  createNode transCircle -n circleNode1;
  sphere -n sphere1 -r 1;
  sphere -n sphere2 -r 2;
  connectAttr sphere2.translate circleNode1.inputTranslate;
  connectAttr circleNode1.outputTranslate sphere1.translate;
  connectAttr time1.outTime circleNode1.input;
  ```

- Exercises

  j. 1. Double click on setUpTransCircle.py, the skeleton of the setUpTransCircle has already been provided. Note that this Plug-in also includes the source code for transCircleNode.

  k. 2. Implement setUpTransCircleCmd.py, add necessary function declaration and member variable declaration.

  Relevant classes and methods:

  MPxCommand:: isUndoable(), MPxCommmand::undoIt(),
  MPxCommand::redoIt()

  Since you want to support undo/redo, you may want to declare an MDGModifier object as a member variable.

  l. 3. Implement setUpTransCircleCmd.py, in doIt() functions, simulate the MEL commands using your member variable MDGModifier object.

  Relevant classes and methods:

  MDGModifier::createNode(), MDGModifier::renameNode()
  MDGModifier::commandToExecute()
  MDGModifier::doIt(), MDGModifier::undoIt(),
  MDGModifier::redoIt() , MDGModifier::connect()
  MGlobal::getSelectionListByName()
  MSelectionList::getDependNode()
  MFnDependencyNode::attribute()

  m. 4. In setUpTransCircleCmd.py, implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the setUpTransCircle command.

  Relevant classes and methods:

  MFnPlugin::registerCommand()
  MFnPlugin::deregisterCommand()

  Note: you will also need to register "transCircle" node to make it work. Relevant classes and methods:

  MFnPlugin::registerNode()
  MFnPlugin::deregisterNode()

- Result

  After you finish and load the plug-in setUpTransCircle.py, execute the follow command:

  ```
  setUpTransCircle;
  ```

You will see two nurbs sphere created and once you hit the "play" button, one sphere is rotating around the bigger sphere, you can also set the radius of the circle, and rotating speed by setting the values in attribute editor of circleNode1.

- Bonus Section

  You can use dgMod.commandToExecute() to simulate MEL command "sphere -n sphere1 -r 1;", however you can also simulate the operations from scratch. Go into the hypergraph and try to find what have happened to the Dependency Graph (for examples, what nodes have been created and what connections have been made) when you execute "sphere -n sphere1 -r 1;" and simulate those changes with MDGModifier.
  Tips: There are mainly two types of nodes created, "nurbsSurface" and "makeNurbSphere", they are connected:
  makeNurbSphere1.outputSurface→sphere1Shape.create

sceneMsgCmd Plug-in

- Topics Covered
  - How to use MSceneMessage to register callbacks watching scene operations
- Overview
  - In this exercise, we will create a custom command to watch specific scene operation messages:
    MSceneMessage::kBeforeOpen,
    MSceneMessage::kAfterNew,
    MSceneMessage::kBeforeSaveCheck
    We will implement callback functions which get trigger by those messages. We will also experiment how to abort current scene operation by using checkCallback functions.
- Exercises
  - 1. Go to "sceneMsgCmd\Exercise" folder, double click on sceneMsgCmd.py, the skeleton for this command is already implemented.

  n. 2. Implement sceneMsgCmd.py, add necessary function declaration and member variable declaration.

  Relevant classes and functions:

  MPxCommand::isUndoable(), MPxCommmand::undoIt(),
  MPxCommand::redoIt()

  To keep a list of all the registered callbacks, you may want to declare a member variable to record all the callback IDs.

  - 3. Implement sceneMsgCmd.py, in redoIt() function, register callbacks for MSceneMessage::kBeforeOpen, MSceneMessage::kAfterNew, MSceneMessage::kBeforeSaveCheck

  Relevant classes and functions:

  MSceneMessage::addCallback()
  MSceneMessage::addCheckCallback()

  In the callbacks functions, just print out "The callback registered for MSceneMessage::kXXXX is executed." Also in the callback function for

MSceneMessage::kBeforeSaveCheck, set “retCode” to be pointing to “false” to abort current operation.
In undoIt() function, remove callbacks.
In doIt() function, call redoIt().

o. 4. In sceneMsgCmd, implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the sceneMsgCmd command.

Relevant classes and methods:
MFnPlugin::registerCommand()
MFnPlugin::deregisterCommand()
Note: In uninitializePlugin() , don’t forget to remove all the callbacks you have registered in your plug-in.

- Result

After you finish and load the plug-in sceneMsgCmd.py, execute the follow command:
sceneMsgCmd;
When you open a file, there will be a print out in the output window:
“The callback registered for MSceneMessage::kBeforeOpen is executed.”
After you create a new file, there will be a print out in the output window:
“The callback registered for MSceneMessage::kAfterNew is executed.”
When you try to save current scene file, there will be a print out in the output window:
"The callback registered for MSceneMessage::kBeforeSaveCheck is executed."
There is also an error message popping up in Maya saying “File operation cancelled by user supplied callback” and the file is not saved.

Module 9

arrowLocator Plug-in

- Topics Covered

Implement a custom locator with MPxLocatorNode. It demonstrates working steps you need to follow to draw custom locator in Maya viewport.

- Overview

In this project, we will implement a custom locator; it has a unit attribute “windDirection”. This locator is drawn as a big arrow in the Maya viewport. You can change the direction of the locator by retrieving its “windDirection” attribute and rotating corresponding angles when drawing the locator node with OpenGL calls.

- Exercises

1. Open arrowLocator.py, the skeleton of the arrowLocator has already been provided.
2. In initialize() function, create “windDirection” as a unit attribute and add it onto the node.

In draw() function, get value of the “windDirection” attribute and add necessary calls for drawing.
Relevant classes and methods:
MPxNode::thisMObject()

MPlug::getValue()
M3dView::beginGL()
M3dView::endGL()

3. Implement both initializePlugin() and uninitializePlugin() functions to handle registration and de-registration of the arrowLocator node.
   Relevant classes and methods:
   MFnPlugin::registerNode ()
   MFnPlugin::deregisterNode ()

- Result
  After you built and load the plug-in arrowLocator.mll, execute the follow command:
  createNode arrowLocator;
  you will find a locator created in the viewport, open AE, find "windDirection" attribute and change its value, the locator will be rotating according to the value you set.

Module 10

arrowLocatorManip Plug-in

- Topics Covered
  Implement a custom manipulator with MPxManipContainer and MFnDiscManip. It manipulates the attribute "windDirection" on an arrowLocator node, which is implemented as custom locator from last asssignment. This Plug-in demonstrates how to set up affecting relationship between base manipulators and plugs on nodes.

- Overview
  In this Plug-in, we will create a custom manipulator called "arrowLocatorManip", user can use this manipulator to change the value of the "windDirection" attribute on a arrowLocator node.

- Exercises
  - 1. Double click on arrowLocatorManip.py, the skeleton of the arrowLocatorManip has already been provided.
  - 2. In arrowLocatorManip.py, register your custom manipulator
    Relevant classes and methods:
    MFnPlugin::registerNode()
    Please note that the name of your manipulator has to follow the name of your locator, otherwise Maya will not know how to connect them.
  - 3. Double click on arrowLocatorManip.py, go into arrowLocator::initialize() function, add one line to make connection between your custom node and your custom manipulator
    Relevant classes and methods:
    MPxManipContainer::addToManipConnectTable()

- 4. Implement arrowLocatorManip.py, in arrowLocatorManip::createChildren() , create a base disc manip and add it into this manip container,
  In arrowLocatorManip::connectToDependNode(), connect the "windDirection" plug on the locator node with the disc manip
  Relevant classes and methods:
  MPxManipContainer:: addDiscManip()
  MFnDiscManip::connectToAnglePlug()

- Result

  Load the arrowLocatorManip.py in Maya, in script editor, execute:
  createNode arrowLocator,
  the arrow locator is created in Maya viewport, then click on the "Show Manipulator Tool" on the toolbar, you will see a disc manip created at the center of the locator, rotating the disc manip will be changing the value of "windDirection" and the arrow locator will be rotating correspondingly.

- Bonus section

  In this example, if you move your locator in Maya viewport and then show the manipulator, the manipulator stays at the original location. But if you uncomment the code in connectToDependNode() where it registers a plug to manip conversion callback , and also uncomment the callback function, and build the code and load in Maya again, you will see whenever the locator moves, the center position of the manipulator moves with it. You can remove the callback function code and try to implement it by yourself.