```
import maya.cmds as mc
from functools import partial
'''

'''

class attrFunctionsUI(object):


    def __init__(self, parentLayout):
        self.parentLayout=parentLayout

        self.userDefinedStatus = False
        self.keyableStatus = False
        self.connectableStatus = False

        self.nodeName = None
        self.nodeAttr = None
        self.attrValue = None

        self.buildUI()

    def buildUI(self):
        clm1 = 125
        clm2 = 75
        clm3 = 100
        separation = 15

        self.radioCollection = mc.radioCollection()
        RBlayout = mc.rowLayout(numberOfColumns=3,
parent=self.parentLayout)
        self.userDefinedRB = mc.radioButton(label='User defined',
parent=RBlayout, onCommand=(partial(self.setUserDefinedStatus, True)))
        self.keyableRB = mc.radioButton(label='Keyable', parent=RBlayout,
onCommand=(partial(self.setKeyableStatus, True)))
        self.connectableRB = mc.radioButton(label='Connectable',
parent=RBlayout, onCommand=(partial(self.setConnectableStatus, True)))

        self.nodeInputLayout = mc.rowLayout(numberOfColumns=4,
parent=self.parentLayout)
        mc.text('     Node name :', parent=self.nodeInputLayout)
        self.nodeNameTF = mc.textField('nodeNameTF',
parent=self.nodeInputLayout, width=clm2, \
                text='nodeName', enterCommand=self.populateAttrOptionBox,
alwaysInvokeEnterCommandOnReturn=True)

        mc.button(label='getAttr', w=clm2, command=self.getNodeAttr,
parent=self.nodeInputLayout)
        self.attrMenuGrp = mc.optionMenuGrp(label='Node attribute :',
changeCommand =self.getNodeAttr, parent=self.nodeInputLayout)

        separator = mc.separator(style ="none", w=300, h=separation,
parent=self.parentLayout)
        setAttrlayout = mc.rowLayout(numberOfColumns=3,
parent=self.parentLayout)
        mc.text('New attr value :')
        self.newAttrValueTF = mc.textField('newAttrValueTF',
parent=setAttrlayout, width=clm2)
```

```
        mc.button(label='setAttr', w=clm2, command=self.setNodeAttr,
parent=setAttrlayout)

        separator = mc.separator(style ="none", w=300, h=separation,
parent=self.parentLayout)
        self.nodeValueFeedback = mc.text('>>>>    Node :         Attr :
Value : ', parent= self.parentLayout)

    def setUserDefinedStatus(self, status, *args):
        self.userDefinedStatus = status
        self.keyableStatus = False
        self.connectableStatus = False
        print '>>>> setUserDefinedStatus == ', self.userDefinedStatus

    def setKeyableStatus(self, status, *args):
        self.keyableStatus = status
        self.connectableStatus = False
        self.userDefinedStatus = False
        print '>>>> setKeyableStatus == ', self.keyableStatus

    def setConnectableStatus(self, status, *args):
        self.connectableStatus = status
        self.userDefinedStatus = False
        self.keyableStatus = False
        print '>>>> connectableStatus == ', self.connectableStatus


    def populateAttrOptionBox(self, *args):
        nodeName = mc.textField(self.nodeNameTF, query=True, text=True)
        print 'nodeName == ', nodeName

        # need to clear the box
        if mc.optionMenuGrp(self.attrMenuGrp, query=True, exists=True):
            mc.deleteUI(self.attrMenuGrp)
        self.attrMenuGrp = mc.optionMenuGrp(label='Node attribute :',
parent=self.nodeInputLayout)


        if mc.objExists(nodeName):

                if self.userDefinedStatus == True:
                        attrs = mc.listAttr(nodeName, userDefined=True)
                        print '==== userDefinedStatus == ',
self.userDefinedStatus

                if self.keyableStatus == True:
                        attrs = mc.listAttr(nodeName, keyable=True)
                        print '==== keyableStatus == ',
self.keyableStatus

                if self.connectableStatus == True:
                        attrs = mc.listAttr(nodeName, connectable=True)
                        print '==== connectableStatus == ',
self.connectableStatus


                if self.userDefinedStatus == False and self.keyableStatus
== False and self.connectableStatus == False:
```

```
                        attrs = mc.listAttr(nodeName)


                print 'attrs == ', attrs
                for attr in attrs:

mc.menuItem(parent=('{attrMenuGrp}|OptionMenu'.format(attrMenuGrp=self.at
trMenuGrp)), label=attr)
        else:
                print '>>>> {nodeName} nodeName doesnt exist >>>> no
attrs found'

        #### set nodeName
        self.nodeName = nodeName

    def printFeedback(self, nodeName = '', nodeAttr = '', attrValue =
''):
        feedback = '>>>>    Node :   {nodeName}             Attr :
{nodeAttr}             Value :   {attrValue}'\
        .format(nodeName = nodeName, nodeAttr=nodeAttr,
attrValue=attrValue)

        mc.text(self.nodeValueFeedback, edit=True, label=feedback)


    def getNodeAttr(self, *args):
        nodeName = self.nodeName

        attr = mc.optionMenuGrp(self.attrMenuGrp, query=True, value=True)
        ###sett attr
        self.nodeAttr = attr

        ### get attrValue
        attrValue =
mc.getAttr('{nodeName}.{attr}'.format(nodeName=nodeName, attr=attr))
        ##set self.attrValue
        self.attrValue = attrValue

        self.printFeedback(nodeName, attr, attrValue)

    def setNodeAttr(self, *args):
        nodeName = self.nodeName
        attr = self.nodeAttr
        ## get the user entered attr
        newValue = mc.textField(self.newAttrValueTF, query=True,
text=True)

        # get the attr type
        attrType =
mc.getAttr('{nodeName}.{attr}'.format(nodeName=nodeName, attr=attr),
type=True)
        print '>>>>>>> nodeName, attr,  type  ',nodeName, attr, attrType

        if attrType == 'string':
            print '>>>>>>> attribute type string'
            mc.setAttr('{nodeName}.{attr}'.format(nodeName=nodeName,
attr=attr), newValue, type='string')
```

```
        elif attrType == 'bool' or attrType == 'byte':
            intNewValue = int(newValue)
            mc.setAttr('{nodeName}.{attr}'.format(nodeName=nodeName,
attr=attr), intNewValue)
            print '>>>>>>> attribute type is bool or byte'

        else:
            floatNewValue = float(newValue)
            print '>>>>>>> attribute type is other'
            mc.setAttr('{nodeName}.{attr}'.format(nodeName=nodeName,
attr=attr), floatNewValue)

        ## reset attrValue
        self.attrValue = newValue

        self.printFeedback(nodeName, attr, newValue)
```