```
import maya.cmds as mc
from functools import partial
from operator import itemgetter

'''
NB. dupe mesh function puts the first key at frame 0
this needs to be changed for blend etc to work
18_07_14
- allow for the selecction list not to be in order.........

17_07_14
- create blends according to vis keys on transform node or shape nodeList
start ==> number of frames from first vis key
duration ==> number of frames that it lasts for (Gina should this be a
percentage???)
NB. I would like to use this format for other functions;
    shader keys
    transform component
'''
def orderList(nodeList):
    extendedNodeList = []

    for node in nodeList:
        nodenameList = node.split('_')

        if nodenameList[-1] == '':
            nodeNumber = 0
        else:
            nodeNumber = int(nodenameList[-1])

        extendedNodeList.append([node,nodeNumber])

    newExtendedNodeList = sorted(extendedNodeList, key=itemgetter(1))
    print 'newExtendedNodeList == ', newExtendedNodeList

    newNodeList = []
    for item in newExtendedNodeList:
        newNodeList.append(item[0])

    print 'newNodeList == ', newNodeList
    return newNodeList



class CreateBlendTweenNodes(object):

    def __init__(self, parentLayout, workOnTransformNodeStatus):
        self.parentLayout=parentLayout
        self.workOnTransformNodeStatus=workOnTransformNodeStatus
        print 'TESTING TESTING BLEND workOnTransformNodeStatus == ',
workOnTransformNodeStatus

        self.workOnChildNodes = True
        self.blendOriginIsWorld = True

        self.buildUI()

    def buildUI(self):
```

```
        clm1 = 125
        clm2 = 75
        clm3 = 100
        separation = 15

        if self.workOnTransformNodeStatus == True:
            mc.text('- Create blend nodes between transform nodes',
parent=self.parentLayout)
        else:
            mc.text('- Create blend nodes between shape nodes',
parent=self.parentLayout)

        self.radioCollection = mc.radioCollection()
        RBlayout = mc.rowLayout(numberOfColumns=2,
parent=self.parentLayout)
        self.childrenRB = mc.radioButton(label='Work on child nodes',
parent=RBlayout, select=True,
onCommand=(partial(self.setWorkOnChildNodeStatus, True)))
        self.selectedNodesRB = mc.radioButton(label='Work on selected
nodes', parent=RBlayout,
onCommand=(partial(self.setWorkOnChildNodeStatus, False)))

        separator = mc.separator(style ="none", w=300, h=10,
parent=self.parentLayout)

        ### blend origin controls
        self.BlendRadioCollection = mc.radioCollection()
        RBblendLayout = mc.rowLayout(numberOfColumns=2,
parent=self.parentLayout)
        self.worldRB = mc.radioButton(label='Blend origin = world',
parent=RBblendLayout, select=True,
onCommand=(partial(self.setBlendOriginToWorld, True)))
        self.selectedNodesRB = mc.radioButton(label='Blend origin =
relative', parent=RBblendLayout,
onCommand=(partial(self.setBlendOriginToWorld, False)))

        blendAlignLayout = mc.rowLayout(numberOfColumns=4,
parent=self.parentLayout)
        mc.text('Start')
        self.blendStartTF = mc.textField('blendStartTF',
parent=blendAlignLayout, width=25, text='10')
        mc.text('Duration')
        self.blendDurationTF = mc.textField('blendDurationTF',
parent=blendAlignLayout, width=25, text='10')


        blendLayout = mc.rowLayout(numberOfColumns=4,
parent=self.parentLayout)
        mc.button(label = 'Create blend nodes', width=clm1,
command=self.createBlends,\
            parent=blendLayout)
        mc.button(label = 'Delete history', width=clm2,
command=self.deleteHistory,\
            parent=blendLayout)

    def setWorkOnChildNodeStatus(self, status, *args):
        self.workOnChildNodes = status
```

```
    def setBlendOriginToWorld(self, status, *args):
        self.blendOriginIsWorld = status
        print 'blendOriginIsWorld status = ', self.blendOriginIsWorld

    def getNodeList(self):
        if self.workOnChildNodes == True:
            groupNode = mc.ls(selection=True)
            tranformNodeList = mc.listRelatives(groupNode, children=True,
type = 'transform')
            shapeNodeList = mc.listRelatives(groupNode, children=True,
type = 'shape')
            # print 'tranformNodeList, shapeNodeList', tranformNodeList,
shapeNodeList

            if self.workOnTransformNodeStatus == True:
                unorderedNodeList = tranformNodeList
            else:
                unorderedNodeList = shapeNodeList
        else:
            unorderedNodeList = mc.ls(selection=True)

        #### get rid of mesh_Orig and make new list ##########
        newList = []
        checkString1 = "Orig"
        checkString2 = "polySurfaceShape"
        for node in unorderedNodeList:
            # print node
            if checkString1 in node or checkString2 in node:
                print '>>> removing ', node
            else:
                newList.append(node)

        nodeList = orderList(newList)
        return nodeList


    def deleteHistory(self, *args):
        #deletes visibility keys on selected Node

        ############# replace following with a function to get nodeList
#########
        # if self.workOnChildNodes == True:
        #     groupNode = mc.ls(selection=True)
        #     tranformNodeList = mc.listRelatives(groupNode,
children=True, type = 'transform')
        #     nodeList = tranformNodeList
        # else:
        #     nodeList = mc.ls(selection=True)
        ############
        nodeList = self.getNodeList()

        i = 0
        for index in range(len(nodeList)):
            node = nodeList[i]

            mc.delete('{node}'.format(node=node),
constructionHistory=True)
```

```
            i += 1

    def createBlends(self, *args):
        #### gina it seems that I just have to delete the off keys and
then make them again if shape nodes are selected
        visKeyOffTime = False
        #### ADD radio button to set clipToVisOffTime ####
        clipToVisOffTime = True
        # print 'clipToVisOffTime == ', clipToVisOffTime

        blendStart = int(mc.textField(self.blendStartTF, query=True,
text=True))
        blendDuration = int(mc.textField(self.blendDurationTF,
query=True, text=True))
        if self.blendOriginIsWorld == True:
            blendOrigin = 'world'
        else:
            blendOrigin = 'local'
        if self.workOnChildNodes == True:
            groupNode = mc.ls(selection=True)
           ##################### what if its
shapes??????????????????????????????????????????
            tranformNodeList = mc.listRelatives(groupNode, children=True,
type = 'transform')
            shapeNodeList = mc.listRelatives(groupNode, children=True,
type = 'shape')
            print 'tranformNodeList, shapeNodeList', tranformNodeList,
shapeNodeList

            if self.workOnTransformNodeStatus == True:
                unorderedNodeList = tranformNodeList
            else:
                unorderedNodeList = shapeNodeList

        else:
            unorderedNodeList = mc.ls(selection=True)

        nodeList = orderList(unorderedNodeList)

        #### create a list to keep track of the blend nodes
        blendNodeList = []

        i = 0
        for item in range(len(nodeList)-1):
            blendTarget = nodeList[i+1]
            node = nodeList[i]
            print '>>>> node, blendTarget == ', node, blendTarget

            ### If ITS A SHAPE NODE go to frame -1 and delete first vis
key
            if self.workOnTransformNodeStatus == False:
                print ' making shapes visible '
                mc.currentTime(-1)
                ## delete the first vis key
                mc.cutKey( node, time=(-1,-1), attribute='v',
option="keys")
                mc.cutKey( blendTarget, time=(-1,-1), attribute='v',
option="keys")
```

```
            ###########


            blendNode = mc.blendShape(blendTarget, node,
origin=blendOrigin, tc=0)[0]
            #### add the blend node to the list
            blendNodeList.append(blendNode)
            ### get a list of the vis keys on the node
            visKeys = mc.keyframe('{node}.v'.format(node=node),
query=True)

            index = 0
            for keyTime in visKeys:
                value =  (mc.keyframe('{node}'.format(node=node),
at='v',t=(keyTime, keyTime), q=True, eval=True))[0]
                # print 'value = ', value
                if value == 1.0:
                    visKeyOnTime = keyTime
                    visKeyOffTime = visKeys[index+1]
                index += 1

            # ## set blend keyframes
            if visKeyOffTime != False:
                # blendKey0 = visKeyOffTime - (blendDuration + 1)
                blendKey0 = visKeyOnTime + blendStart


mc.setKeyframe('{blendNode}.{blendTarget}'.format(blendNode=blendNode,
blendTarget=blendTarget),\
                    time = blendKey0, value = 0)

                if clipToVisOffTime == True:
                    print 'clipToVisOffTime == ', clipToVisOffTime

                    if int(blendKey0 + blendDuration) >
int(visKeyOffTime):
                        print 'yep its greater'
                        blendKey1 = visKeyOffTime
                    else:
                        blendKey1 = blendKey0 + blendDuration
                else:
                    blendKey1 = blendKey0 + blendDuration


mc.setKeyframe('{blendNode}.{blendTarget}'.format(blendNode=blendNode,
blendTarget=blendTarget),\
                    time = blendKey1, value = 1)

            ### If ITS A SHAPE NODE put the frame one vis key back
            if self.workOnTransformNodeStatus == False:
                mc.setKeyframe('{node}.v'.format(node=node), time = -1,
value = 0)

mc.setKeyframe('{blendTarget}.v'.format(blendTarget=blendTarget), time =
-1, value = 0)

            i += 1
```

```
print 'blendNodeList is ', blendNodeList
```