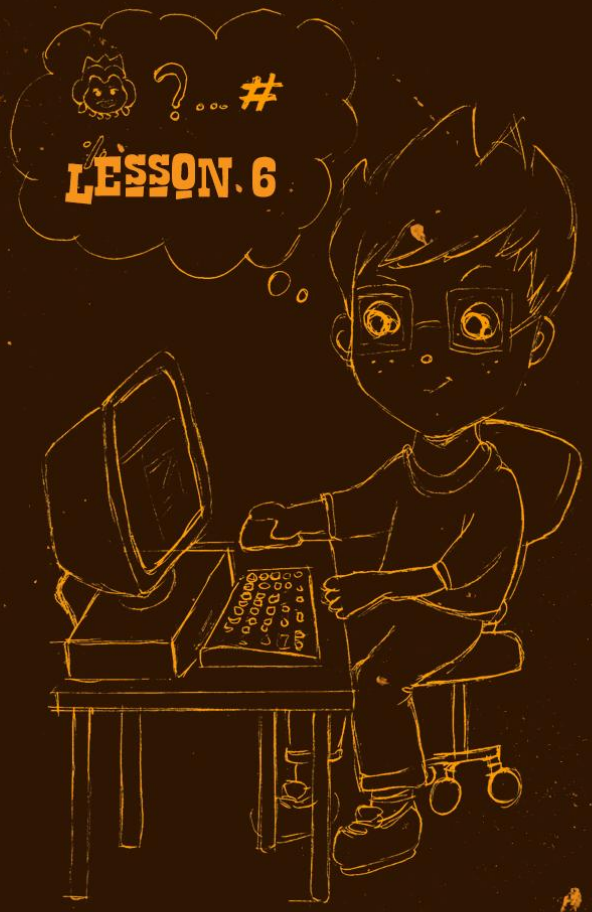
PYTHON SCRIPTING FOR SMART AND CURIOUS COMPOSITORS



Gianluca Dentici
www.gianlucadentici.com

LESSON 6

Hi VFX-Nuke-Python gang! Welcome back to a new hilarious Python for Nuke lesson!
This time the lesson is going to be very rich and demanding… just warning you ahaahha ;)
How was your Christmas period ? I really hope you all had a very pleasing time !
So first of all happy new year to you all !! I celebrated in a fantastic restaurant on the south side of Italy ! I spent the entire night eating very posh and well crafted food and fresh fishes!
Find here the picture of the tasty appetizer I had, just to cheer up our Python lesson:

Ok now before starting, I know that some of you might have found issues with the indentation when copy and paste codes from these pdf documents straight to the nuke script editor, it is looking misaligned comparing with the pdf, so if this is the case (and I know that might be boring to align all the things from scratch), please feel free to text me and I could send you the original Open Office file. Ok now let's start with our beloved assignment solutions!

Assignment 1:

"*Let's assume we have Group nodes into our script and we want our Python code to expand just those with a particular node within. Let's say a Blur or grade node*".
Well there are many ways to get to it, in fact someone might be thinking on doing something crazy like this:

```
grNames=[]
targetNodes=[]
for gn in nuke.allNodes('Group'):
   grNames.append(gn)

kkk=str(len(grNames))

for uu in range (0,int(kkk)):
   sw=grNames[(uu)]
   sw.setSelected(True)
   for n in sw.nodes():
     if n.Class() == "Grade":
        targetNodes.append(sw)

nuke.selectAll()
nuke.invertSelection()

for xx in targetNodes:
   xx.setSelected(True)

for yuppy in nuke.selectedNodes():
   nuke.selectAll()
   nuke.invertSelection()
   yuppy.expand()
```

...which is working well actually but it is very mental! It's too long and laborious, I won't go deep into it because it is really wacky, but as you can see it is basically entering groups and running actions into them, yet it is using the **range** command to count off the total number of groups...etc...
But we should ease the things up! we need to consider everything we learnt so far, put all of that together and make the most of our beloved code.
So let's try to simplify this beast, shall we :

Let's try this:

```
import nuke
lis=[]
for a in nuke.allNodes('Group'):
      xx=a.nodes()
      for k in xx:
           if 'Grade' in k.Class():
                lis.append(a)

for ww in lis:
```

```
    if 'Group' in ww.Class():
        ww.setSelected(True)
        ra=nuke.selectedNodes()

for ee in ra:
        nuke.selectAll()
        nuke.invertSelection()
        ee.expand()
```

Wow! Way better! That's all we really need !

Ok let's go through the lines now.
Basically on the first line I'm looking for all Groups nodes in the script. The second line is very important and really makes the difference comparing with the first crazy-long approach! - it adds a new variable **xx** that represents all nodes sitting inside the selected groups. That's the trick !
In fact the **nodes()** construction is, in this case, actually looking for nodes sitting within groups. Then the loop starts and the first condition is telling Python whether there are any nodes belonging to the Class of 'Grade'. If this is true it stores them into my **lis.**
Now on the following line I'm checking whether the class of the elements in the list are actually Groups and if this is true it selects them and add a new variable '**ra**'.

Then before exploding groups I'm deselecting all script's nodes just in case something is accidentally selected and might return to unexpected results, so better do it than don't.

Awesome ! it should be working like a charm !

We might improve this code by adding some backdropNodes behind the exploded groups, just to make them more obvious and the artists can easily find them all.

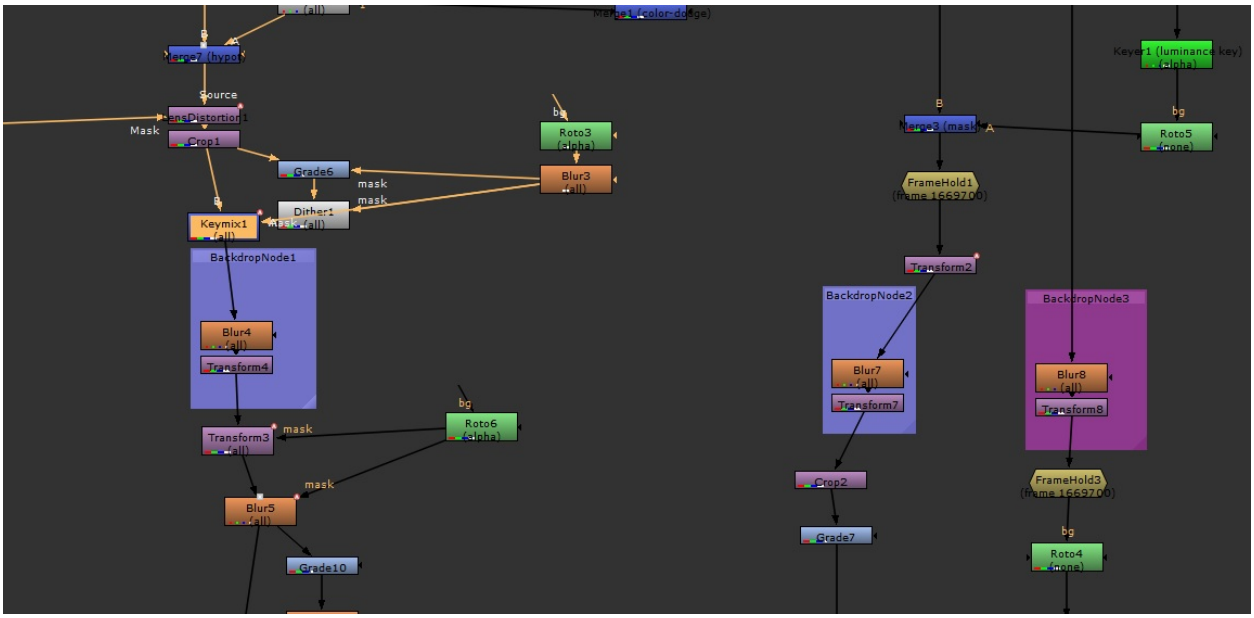
```
lis=[]
for a in nuke.allNodes('Group'):
    xx=a.nodes()
    for k in xx:
        if 'Grade' in k.Class():
            k.setSelected(True)
            a.begin()
            nukescripts.autoBackdrop()
            a.end()
            a.setSelected(True)
            lis.append(a)

for ww in lis:
```

```
    if 'Group' in ww.Class():
        ww.setSelected(True)
        ra=nuke.selectedNodes()

for ee in ra:
        nuke.selectAll()
        nuke.invertSelection()
        ee.expand()
```

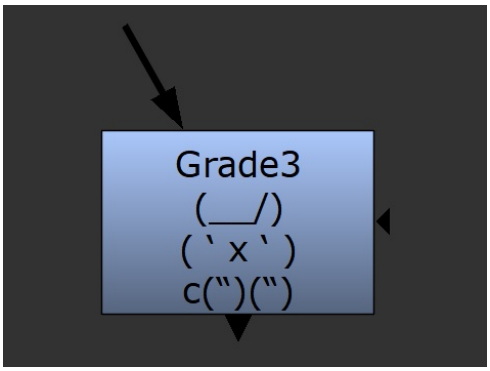
...and this is how it should be looking like:



Brilliant ! As you can see this time I went for another method, I've just jumped inside groups to create backdropNodes! This is something we have already covered, so I'm quite sure it will make sense to you !

but now… have you noticed the backdrop colors are all different ? now what if you want them being one same color, let's say a black or a red ? I'm challenging you know, show who you really are with your Python !! This will be another add for your next homeworks's list !

Now on our usual daily practice we are not supposed to explode Groups just because we are worried about Grade nodes, how in the world a Grade node could be so harmful?



Hahah ;)
but this is something that might be true for any non-standard gizmo nodes for instance.
In fact for instance many big companies' pipelines don't allow you to work with any external or unknown nodes or gizmos! There are many reasons for that, one is they might erroring the farm on rendering time because those Gizmos don't sit into the local pipeline. On other hands you could have picked someone else Nuke's script and you couldn't be aware he has been using something non standard.
Yet there might be circumstances where you are asked to create simpler versions of your scripts because of a stereo conversion work that has to be done from another vendor out there and they want to simplify the script by using standard Nuke's nodes only - in this case it means exploding all gizmo nodes, also those proprietary. However all these circumstances will require you to explode all non-standard nodes.
So let's figure out how to do it !

This is the basic construction:

```
for kk in nuke.selectedNodes():
    gizmo=(kk.Class() + ".gizmo")
    if gizmo in nuke.plugins(nuke.ALL | nuke.NODIR):
        print 'hello'
```

Ok we are learning something new here! Not a big deal actually but it worth spending a few words.
Basically on the second line I'm creating a variable called **gizmo** and it stands for the 'w*hatever class node you select with the gizmo extension*..'. Then the third lines is performing a check into the **nuke.plugins** main folders...ok don't freak out with it, it is very simple!

In order to figure out what this **nuke.plugins** command is doing just clear up your script editor, type and execute: **nuke.plugins(nuke.ALL | nuke.NODIR)** .. you'll see that it returns with a listing of all plugins into the nuke folder (not only those loaded), whereas the second construction **nuke.NODIR** simply gets rid of the file full path, which is a more relaxing way to look at them ;)
On other hands if you don't specify any switched between your parenthesis and just type and

execute: **nuke.plugins()** it returns with the full paths of all loaded plugins.

..there is even more you can do ! If you just want to see a list of the gizmo you are currently using into your script just execute this:

```
ss=nuke.plugins(nuke.NODIR)
for xx in ss:
    if '.gizmo' in xx:
        print xx
```

Ok after this long rant let's go back to the 'checking for gizmo' coding ! Now if you try to put some non standard gizmo into your script and execute the code we have seen before it will print out the word "hello" into your script viewer.
So now that you know how it works why don't you try and squeeze it into the previous script? I mean the one who looks for specific nodes into groups ? This time It should be looking for gizmo nodes in Groups and if this condition is true it has to explode them all. This is another homework for you !!! ;) Get in !

Now let's take a look at the solution for the **Assignment 2:**
"Grab the projection rig we have covered in the previous lesson and improve it by creating a sphere and connecting Project3D nodes to each camera".

This is the solution:

```
from __future__ import division
import math
FL=nuke.getInput('focal length', 'type your focal length')
sen_width=nuke.getInput('Sensor Width', 'type your sensor width')
FOV=2*math.atan(float(sen_width)/(2*int(FL)))*(180/math.pi)
print math.ceil(FOV) #approximates the result to the returning value

cam_num=math.floor(360/FOV)

for s in range(int(cam_num)):
    a= nuke.nodes.Camera()
    rot= s * float(FOV)
    a['rotate'].setValue([0,rot,0])
    a['selected'].setValue(True)
    a['haperture'].setValue(float(sen_width))
    a['focal'].setValue(int(FL))
    k=nuke.selectedNodes()
    pr=nuke.nodes.Project3D()
    pr.setInput(1,a)
```

```
coun=len(k)+1
b = nuke.createNode('Scene',inpanel=False)
x=0
for i in k:
    b.setInput(x,i)
sph=nuke.createNode('Sphere', inpanel=False)
sph['uniform_scale'].setValue(50)
sph['display'].setValue(1)
b.setSelected(True)
b.setInput(coun,sph)
nuke.selectConnectedNodes()
for n in nuke.selectedNodes():
   nuke.autoplace(n)

nuke.message('I have created '+ str(cam_num) +' cameras' + ' with a rotation factor of ' +
str(FOV) + ' degrees')
```

Please notice that here I'm connecting all cameras to a Scene nodes just for viewing purposes, but in order to create projections you would need to unlink them and use '**Mergemat**' nodes to merge your textures together.
So I know that this is going to be a little tricky programming-wise but let's give it a try ! it is worth doing it ! ..please consider it as a good starting point to learn concatenating multiple things together. So Don't let you down !
this would be the code:

```
from __future__ import division
import math

FL=nuke.getInput('focal length', 'type your focal length')
sen_width=nuke.getInput('Sensor Width', 'type your sensor width')
FOV=2*math.atan(float(sen_width)/(2*int(FL)))*(180/math.pi)
print math.ceil(FOV) #approximates the result to the returning value

camList=[]
projList=[]

cam_num=math.floor(360/FOV)

for s in range(int(cam_num)):
    a= nuke.nodes.Camera()
    rot= s * float(FOV)
    a['rotate'].setValue([0,rot,0])
    a['selected'].setValue(True)
    a['haperture'].setValue(float(sen_width))
```

```
    a['focal'].setValue(int(FL))
    k=nuke.selectedNodes()
    pr=nuke.nodes.Project3D()
    pr.setInput(1,a)
    projList.append(pr)

nuke.selectAll()
nuke.invertSelection()

asa=projList[0].setSelected(True)

prjNum=len(projList)

for syt in range(1,prjNum):
      sr=nuke.createNode('MergeMat', inpanel=False)
      sr.setInput(0,projList[(syt)])

sr.setSelected(True)

sph=nuke.createNode('Sphere', inpanel=False)
sph['uniform_scale'].setValue(50)
sph['display'].setValue(1)

nuke.message('I have created '+ str(cam_num) +' cameras' + ' with a rotation factor of ' +
str(FOV) + ' degrees')
```

I didn't comment over the lines intentionally because it could be messy and harder to understand, but I will shortly go through the lines.
Basically the lines I have highlighted are those where relevant changings occur, otherwise the code would look like quite the same as the previous one, but let's see what it is actually changing.
First thing to notice is I've created 2 lists called **projList** and **camList** those are the 'containers' where Project3Ds and Cameras will be stored to - and in fact going down the code we run into the line where we are storing our Project3D's nodes to this list.
Then we make sure to deselect everything..

Now in order to understand the remaining lines I would advice to jump onto the line where the second loop starts and then proceed backwards.
Basically there I'm creating a new loop with a **range** function, it means that everything inside it will be repeated a certain number of times. So the question is: how many times? As you know the minimum and max values for the **range** loop is defined between parenthesis; now min is set to **1** whereas the maximum number is set by the variable **prjNum**.. So what this variable stands for ? well if you look a few lines upwards you'll find out that it simply counts the number of the nodes into the **projList**, hence the Project3D's nodes. This means that the range routine will be

repeated for the max number coming out this counting.
The remaining lines simply creates the mergemat node.

Then comes the tricky part:

sr.setInput(0,projList[(syt)])

Well, on this line we are setting up the input for each **Mergemat**.. so how ? normally when it comes to set an input name we have to specify the actual name of the node we want to link to, but we are inside a loop now! hence we need to specify a range of names!!
So the solution is calling each specific Project3D node in the **projList** by using its indexes, but which indexes ? this is also an evolving number! Fucking hell !
So we need to use the number that comes out the range loop itself by using its variable **syt** - you know this variable will become a specific number accordingly to the amount of project3D nodes in the list and bla bla bla.. Therefore we can write it between the square bracket as it was an index number - and it definitely stands for an index number!
Please just don't forget the round parenthesis because it is a variable, if you miss it Python will returns and error because he would look for a proper integer number.

Ok guys if you got through it then the worst is over !!!
Now what is remaining to do is the sphere creation, so let's select the last created MergeMat node (which is usually the last we have created, thus you don't need to do anything crazy), by simply writing **sr.setSelected(True)**
The rest of the code you already know. So all done guys !

ok if you get through it you really deserve a break…. So let's work off the stress with some crazy instruments. Look at this youtube video:

https://www.youtube.com/watch?v=lY-7cYMVQ2A

nice stuff uh ?

Now the **Assignment 3** was:

“Let’s assume we have a big script where for some reason we have been using a few strong defocuses or something similar like a PGBokeh. We want to create a script asking the user what’s the maximum amount of defocusing value after which it sets the $Gui disable expression for all these nodes”
This is sometimes useful when you are dealing with someone else’s shot and they told you to make it lighter.

```
uin=nuke.getInput('type your maximum tolerable value','value')
for n in nuke.allNodes('Defocus'):
    if n['defocus'].value() > int(uin):
        n['disable'].setExpression("$gui")
```

It is easy like that !! basically it is picking a user defined value and using it back again on the third line to specify the max tolerable value before $gui-disabling ;) so if the defocus value is higher than the user defined value, it sets the $gui expression to your defocus node(hence it is disabled). Please note that I’ve been using the variable **uin** within the condition and please don’t forget that it is looking for an integer number!

Now let’s move onto another topic.
On the last lesson we have seen how cloning nodes, but how about making just simple copies of nodes ?
So we now need to get familiar with a new library import, that is **copy.**
It is quite easy actually, just select the node you want the copies for and execute the following code:

```
import copy
for ex in range(10):
   copy.copy(nuke.selectedNode())
```

We have copied our node ten times! Obviously you could re-organize your Python code so it could get a value from a **getInput** window, hence the user would be free to input the number of copies he likes, just like we have done on the previous example.

We will be coming back to ‘copy’ and cloning on multiple occasion but for now I thought it was important to mention since it was neglected on the previous lesson.

Now remember our past excursus over backdrops ? well looks like I’m in love with them ahaha, well the true reason is sometimes it is easier to cook simple Python examples by using them! Yet they can be very helpful on making some order into your scripts as well as messing them up even more ahahha, so learning how to do something specific with it is truly important.
So now we are going to write kind of useful stuff with some help from conditions, with the goal in mind of making our scripts more neat and efficient !

I would begin by saying that sometimes when we work on scripts we might need to clear some stuff out, like removing any crazy Backdrops that ourselves or someone else spreaded all over the script. So by writing 3 simple Python lines we can get rid of all backdrops all at once. Obviously all nodes will remain where they are.
In order to test it feel free to open up one of your busy scripts and try this:

```
for a in nuke.allNodes():
   if a.Class()=='BackdropNode':
      nuke.delete(a)
```

Good! It works like a charm ! In fact this very ridiculous piece of code is part of my tools in Nuke, among many similar ones.
There is more! Sometimes comp. Leads or sups love to add sticky notes for your convenience and flagging what is going on over specific areas of their script just in case you have to pick it up and use part or all of it into your shot.. And this is good for you! but by the time your script gets more busy they become quite annoying, so why not using the same code to perform a deletion of this stuff too ?
Well well well, this is offering us the opportunity to improve our Python notions! !!
Let's do it !

```
for a in nuke.allNodes():
   if a.Class()=='BackdropNode' or a.Class()=='StickyNote':
     nuke.delete(a)
```

Wowwwwwwwwwwww !! see what you've got up to by just adding a simple '**or**' ?? that's very exciting ! and there is much more you could do ! we will go through other lovely stuff very soon !

So far we have been playing with basic conditions operators, so now has come the time to carry on and try to build a concatenation of conditions, or so called "nested conditions" ….sounds scary but it isn't indeed ! This is very useful whenever you need to check for, let's say, two or more conditions prior executing your code.
Now we assume that we want to turn the PostageStamp gadget on for a specific class of nodes, a Blur for instance.

```
myNodes=nuke.allNodes()
for ex in myNodes:
 if ex.Class()=='Blur':
   if ex.knob('mix').value()==0:
         ex.knob('postage_stamp').setValue(True)
```

Here it is interesting, in fact as we've seen also on other examples we want to check not only whether Blur nodes are living into our current selection but if they actually have a knob called "**mix**" that has zero value. This procedure is a very good habit and it is called '**Debugging**'; It means specifying what happens with the code if one or multiple conditions are not True or if you are doing something wrong. Obviously you can change the first line at your convenience by

telling Python to act on selectedNodes instead.

Now let's assume we have a nuke's script with Transform nodes that we labeled to 'STEREO' because we are using them to offset stereo elements on our shot. Now let's say we need to get rid of them. We might need a routine that could act on all of them at once; let's try this:

```
for a in nuke.allNodes():
    if a.Class()=='Transform':
        if "STEREO" in a['label'].value():
            nuke.delete(a)
```

As obvious in these lines I've used the conditions concatenation one more time. As anticipated I'm telling him to check on all nodes and **if** there are any Transform nodes with it and **if** they have a STEREO label, then it runs the deletion. Looks like working pretty well !

...or we might want to disable them instead of deleting at all:

```
for a in nuke.allNodes():
    if a.Class()=='Transform':
        if "STEREO" in a['label'].value():
            a['disable'].setValue(True)
```

Another Python tool that I'm very keen on using when it comes the time to tidy up scripts is the one that get rid of all disable nodes. It is quite easy, it just checks on any 'disable' knob across all nodes in the script, and if so, it deletes them all.

```
for a in nuke.allNodes():
    if 'disable' in a.knobs():
        if a['disable'].value():
            nuke.delete(a)
```

Obviously you have to be very careful on using this tool and I strongly recommend you to double check you have the same outcome at the very end of your tree as it was before the deletion! In fact you might have used any weird gizmos into your script and some of them might have non-standard linking-inputs with their slots and occasionally they could break unexpectedly.
Or for instance you could have **$gui** expressions that you might have set to temporary disable heavy nodes like Zdefocus, Bokeh etc, In this case if you are using those python lines they will be removed from your script too!!!! Holy shit ! this is not what we want ! So some sort of action is required to prevent that !
well well well...how about adding just one more condition to check for any possible expressions within your nodes?
here you are:

```
for a in nuke.allNodes():
   if 'disable' in a.knobs():
      if a['disable'].value():
         if not a['disable'].hasExpression():
            nuke.delete(a)
```

Remember this guy? We have already covered it some time ago ! it cames useful once again!
However this writing might sounds awkward because over the 4th line we are kind of negating that **'a'** has an expression ;) it sounds funny ! Actually it should be read llke: "*if the disable knob doesn't have an expression….*" do something.
However now the code will be getting rid of all disabled nodes but those with expressions ! Spot-on !! we are safe !

Now let's go back to our beloved Backdrops friends and let's play with conditions;
Let's assume we want to color all backdrops in grey whereas any other overlapping backdrop you might have into your script turn to a brighter grey... Ok this is now a good chance for introducing another important command used in loops and conditions: '**else**'

```
for a in nuke.allNodes():
   if a.Class()=='BackdropNode':
      if a['z_order'].value()==0:
         a['tile_color'].setValue(1128481791)
      else:
         a['tile_color'].setValue(1701144063)
```

As you can see these lines are very straightforward, in fact they are just performing a check to the BackdropNode's **z_order** knobs; If they are zero it assigns a grey to Backdrops otherwise a brighter shade of gray.
Obviously this makes sense as long as you haven't increased the z_order on your lowest level backdrop ! In fact I would expect you have increased the number of your overlapping backdrops instesad! This makes much more sense.
Another variation would be coloring all BackdropNodes to grey and leaving those overlapping as they are:

```
for a in nuke.allNodes():
   if a.Class()=='BackdropNode':
     if a['z_order'].value()==0:
        a['tile_color'].setValue(1128481791)
     else:
         pass
```

Ok here we are also learning something new..
We are telling him that if the **z_order** knob is zero, then change the tile color, otherwise **pass**…..
This '**pass**' command is just a null operation; nothing happens when it executes but it is useful

in this specific case as the termination of our code, basically it is like saying "*ok if the second condition is false don't do anything*"
It sounds weird but it is easy as it is !

..or on the following example we could tell him to come up with a message for the user:

```
list=[]
for a in nuke.allNodes():
    if 'BackdropNode' in a.Class():
        if 'z_order' in a.knobs():
            if a['z_order'].value()==0:
                list.append(a)

if len(list)==0:
    nuke.message("you don't have any BackDropNodes in your script")

else:
  for kk in list:
      kk['tile_color'].setValue(1128481791)
```

Wo wo wo...hold on a minute...what the hell is going on here ? it is looking like I'm using more conditions and the code layout is quite different! What's the reason for that ?
Well first of all has come the time to add some more lines to make our code safer on any circumstances, furthermore this is one of the safer ways to get to a good outcome, in fact if you try to make the things easier you might run into unexpected outcomes.

Now let me explain what I'm doing:
First of all I've introduced one more condition on top of the others and it is basically checking whether I really have any Backdrop nodes into my comp script, the second condition is checking for the **z_order** knob;
Moving downwards we ran into the first critical condition that is verifying if the **z_order** value is 0 and if this is true it appends the nodes to the **list**.

Then we get out of the loop just for writing another condition: this time it is checking whether the amount of nodes into our list is zero, meaning that if we don't have any backdrop nodes in the script (hence the list is empty), then returns a message.

Moving down, by using the **else** command I'm telling Python that if the list isn't empty we want it to execute something else. And in fact it is finally changing the color of all your backdrops.

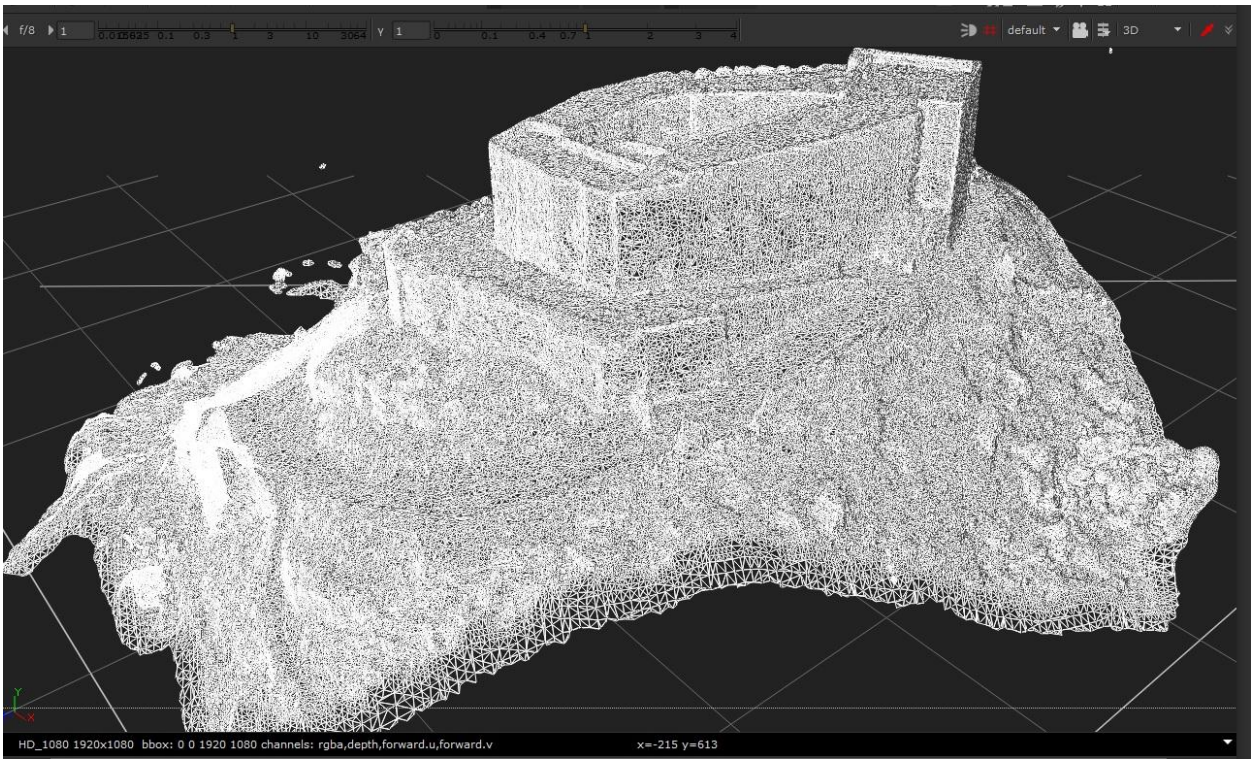
AAAMMMMAAAAAZINGGGG !!!

Ok now take a cappuccino but please don't drink it with a pizza or pasta like most of the Brits are capable on doing, it's abominable ! Forgive me if I'm italian and I can't stand it!

That's the correct way to drink a Cappuccino:

Now while you drinking it take a look at this :

This is the nice Castle of Arechi (SA) – Italy.
I took many pictures from a drone and I did some photogrammetry work to build up a 3D mesh and I managed to import the geometry into Maya and Nuke. Lots of fun guys ! I'm doing some experimentation with all this stuff to build automations etc..



Ok now let's go back to our Python !

Let’s pick up the previous example and make the things a little more complicated and exhaustive !
Now the question is: what if I’ve got any BackdropsNodes’ **z_order** set to 1 whereas others are set to zero ? well the code we have just written will not work! The reason is you need to feed your Python code with more infos on what doing when such conditions come up.
So let’s take a look at these lines here:

```
list=[]
great=[]
a=nuke.allNodes('BackdropNode')
if a:
   for xx in a:
      xx.setSelected(True)
      if xx['z_order'].value()==0:
            list.append(xx)
            xx['tile_color'].setValue(1128481791)
      if xx['z_order'].value()>0:
            great.append(xx)

   if len(great)>0:
      nuke.message("some of your backdrop have a value higher than zero <BR> - they are now turned to black ")
```

```
else:
    if not a:
        nuke.message("you don't have any BackDropNodes in your script !")

for uu in great:
            uu['tile_color'].setValue(1)
```

Well if you compare this coding with the previous one it is obviously a little more sophisticated because it works more consistently on any unexpected. In fact now the program reacts differently:

1. If your z_order's backdropNodes are all set to zero (their default value), it turns them to a grey color - so far so good;
2. If there are any backdropNodes with z_order values=0 whereas others have different values, then those at zero will turn to grey but then then a popping message will warn the user there are nodes with higher values in the script and all these backdrops will be turned to black; Wow!
3. If all backdrop nodes have a z_order value greater than 1, again the message will pop up;

4. If you don't have any backdropNodes at all in your script a different warning window will pop up instead.

So this is a classic example of a good debugging! Basically it is always important to ask yourself: "*what happens if conditions are true or false ? ...and what if some of them are true and other false?"* This is always a good exercise on programming and to build good debugging lines.

Now going through the code we could make out 4 different blocks:
On the first one (lines 1 to 11) I'm starting with the first condition and I have set up a couple of lists first.
Then I set off other conditions - the first one is really the same as the one in the previous script; Here I'm storing all those backdropNodes who have a **value=0** into **list**. After that another condition is storing all backdrop nodes with a z_order **value > 0** into another list, called '**great**'.

Now the second block is checking whether you have any nodes with z_order value greater than 0 by looking into the '**great**' list; if this is true it pops up the message.

The third block is used to check whether there are any backdrop at all in the script, so the **else** command is dealing with the **'if a:'** condition and in fact they have the same indentation.
If this is not the case it triggers another message.

Now on the 4th and last block we are calling the '**great**' list once again, this time by creating a new loop that will perform the color change for any possible backdrop nodes who sit into this list. If true all of them turn to black.

Another thing that it is worth noticing is just a nice curiosity. You might have seen that I've put a **
** into the nuke message sentence, well this is another html trick that just wraps the line and start with a second one, so you have 2 text lines instead of a longer one. That's good to know !

The program I've just written might look like a bit awkward to more skilled programmer but the reason is we still haven't covered Functions ! As soon as we get there you will find out there will be many other writing syntax to simplify your scripts. So stay tuned ! ;)

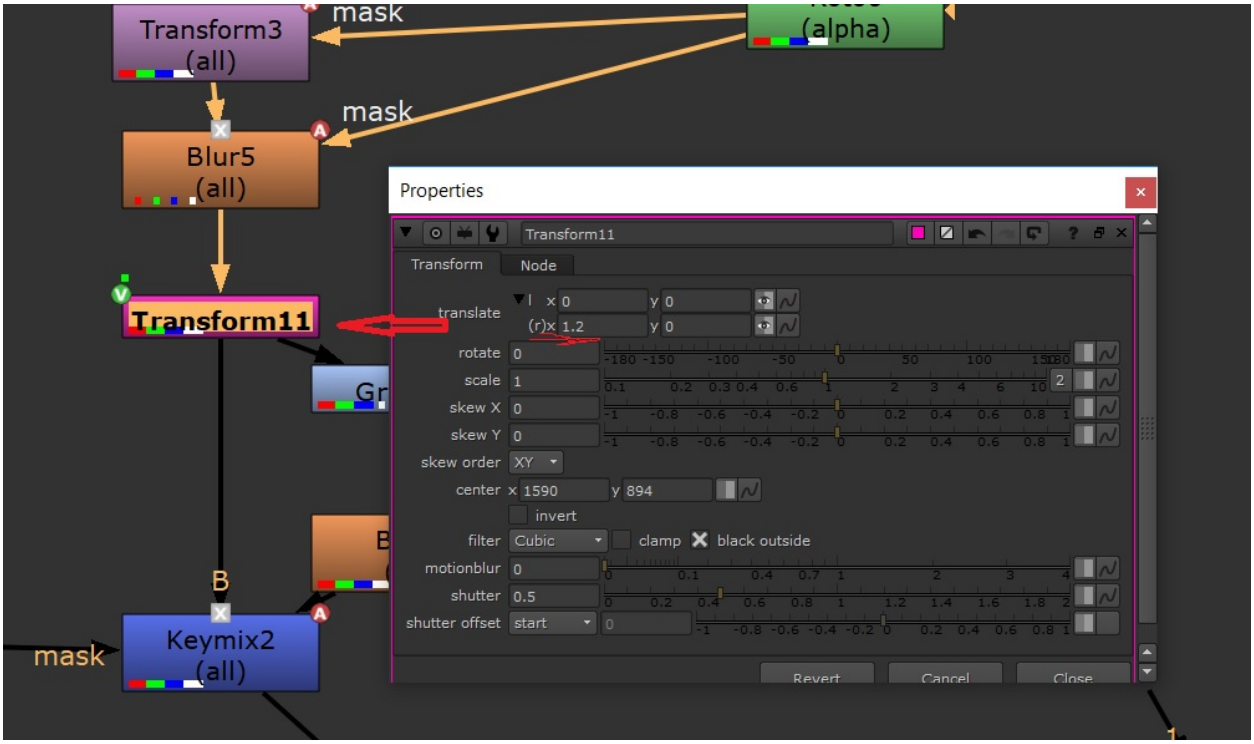
...Now why don't you show me who you are and change this script a bit so the user could choose the color he likes for his backdropNodes ? I'm quite confident you'll figure it out without struggling too much..this is another idea for your homeworks ! put it on the list !

Now the scenario is: you are working on a stereo show and you have just took over someone's else shot because the previous artist left to Veligandu (Maldives).

...and you are down there in the darkest corner of the company trying to figure out what the hell he were doing with the stereo and fixing some stuff for the final stereo delivery. You realised that he was already thinking to Maldives because when you wear your 3D glasses you get motion sickness with the stereo mistakes he has done. So you have to fix it because the show's world premiere is getting closer and if you don't want to risk of being sent around to every cinema projection rooms with an hard disk and stick that shot in it... You gotta rush off!

So first of all I would try to check what the shot would look like without any possible stereo changes. In order to do it you would need to just turn all of them off. Please remember that possibly he could have added some shifting on many other nodes other than the usual Transform and TransformMasked nodes ! it means that on huge scripts it could be very time consuming trying to figure out where and what. So You know what ? let's disable them all

because you don't have such time and, you know,sometimes it is better to start over than swimming into someone's else mistakes !
First of all let's start by 'unsplitting' the views within Transform and TransformMasked nodes so any possible change on the other channel wil be turned off. As you might know the splitted views into your nodes should be looking like this:



So let's take a look at the following lines:

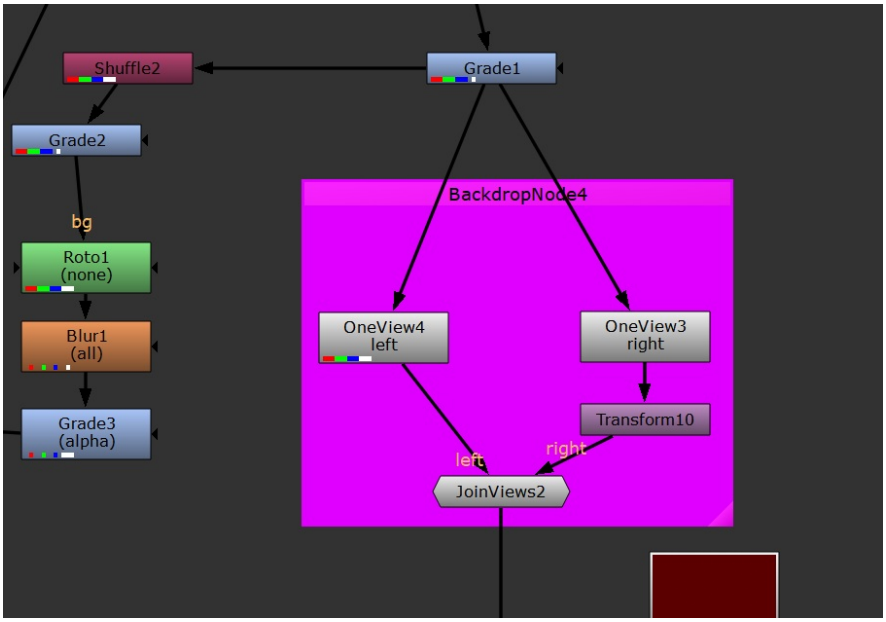
```
sn=nuke.allNodes()
for n in sn:
    if 'translate' in n.knobs():
        n['translate'].unsplitView()
    if 'stereo_offset' in n.knobs():
        n['translate'].unsplitView()
```

Someone could argue this writing is not very elegant and a proper programmer would never write something like this. So please allow me to remember you that the goal of these lessons is making you familiar with python scripting for Nuke, creating your own tools and not raising super nuclear-plants' programming engineers with a posh knowledge of pure Python! I have reached my personal goal when you guys manage to program your own tools improving your daily basis comp work. That's it ! so don't be afraid on writing code wearing a jeans instead of an Oscar Night tight! just make it working properly and quick. You'll have years to improve your programming skills though.

However the same code could be written as follow:

```
sn=nuke.allNodes()
for n in sn:
    if 'translate' in n.knobs() or 'stereo_offset' in n.knobs():
        n['translate'].unsplitView()
```

Now this code will be working nicely for all those nodes with a Translate knob, like Transforms, TransformMasked and DeepTransform and this is fine! but we have to consider that the compositor might have created other transform nodes and he could have placed them under the 'Split and join' template, hence they don't have any split channels and the code we have just covered won't affect them ! In fact that works on nodes with splitted channels only!. Oh yeah ! that is true !! take a look at the image below, this would be the new challenge for us:



So what ? let's go with this:

```
jv=nuke.allNodes('JoinViews')
for kk in jv:
    sel=kk.dependencies()
    for x in sel:
        if x.Class()=="Transform" or x.Class()=="RotoPaint":
            nuke.delete(x)
```

Very interesting indeed! So basically I'm getting rid of all Transform nodes under OneView's nodes… and I'm doing it by using the **dependencies()** command..can you remember it ?

It works amazingly ! Obviously it is up to you whether you want them deleted or simply disabled, feel free to change the last line at your convenience.
Awesome !

Now the problem are the Rotos and RotoPaints that our friend compositor might have spreaded all over the script and that could be acting on the right channel (if we account the left channel is our hero view), so the goal would be getting rid of all that work done on the right channel (obviously please double check first whether the work has to be thrown away!!).
Now in order to sort this out we must use some code we don't know yet ! In fact this time we would need to act on each brush, clone, smear, any stroke in general that is sitting into the paint node, hence we need to access its attributes!
Now find here the entire script to perform all operations we have seen plus the roto/rotopaint solution:

```
import nuke
kn = nuke.allNodes()
ty = ['Roto','RotoPaint']
for s in kn:
    if s.Class() in ty:
          s.knob("selected").setValue(True)

    roto = nuke.selectedNode()
    knob = roto['curves']

    for shp in knob.rootLayer:
       attrs = shp.getAttributes()
       attrs.remove('nv')
       attrs.add('nv', 1)

trNodes=nuke.allNodes()
for n in trNodes:
     if 'translate' in n.knobs():
        n['translate'].unsplitView()
     if 'stereo_offset' in n.knobs():
        n['translate'].unsplitView()

jv=nuke.allNodes('JoinViews')
for kk in jv:
    sel=kk.dependencies()
    for x in sel:
       if x.Class()=="Transform" or x.Class()=="RotoPaint":
          nuke.delete(x)
```

The highlighted lines are those who are actually acting over rotos and rotopaints!

As anticipated this is something we haven't covered yet and I'll put it off until next lessons, but basically I've assigned a variable to the roto's curves, then I've gone through the list of the strokes with the command **knob.rootLayer** and then over the '**nv**' attribute- which deal on views - basically resetting it.

WOW! I think we can stop with it for now! We have done our job, we fixed all our stereo stuff, the shot is safe and the show is playing in cinemas. We are very relieved indeed ! it made our day !

Now let's go back to a piece of code we have already covered and let's add an **else** conditions to make it performing better. The script is the one who deletes any transform nodes that have 'STEREO' as label. First of all make sure you have any Transform's nodes in your script, then execute that:

```
lis=[]
a=nuke.allNodes('Transform')
if a:
    for xx in a:
        xx.setSelected(True)
        if "STEREO" in xx['label'].value():
            lis.append(xx)
            nuke.delete(xx)
else:
   nuke.message("you don't have any Transform node")

if len(lis)==0:
  nuke.message("you are safe ! there aren't any STEREO labeled node in your script!")
```

Ok I can see your puzzled faces but ok don't bring yourself down, we just need to take a closer look to these lines because there are a few things to learn, but I promise this is going to be very easy.
You should be already familiar with the first lines, you might notice that I'm setting a very first condition that is checking if there are any Transform nodes throughout the script and then if this is true it goes through the **for** loop.

Then the **else** on the the 9th line is acting on the first condition:
So basically starting from the 6th line: - "*if there are any transforms with a STEREO label*", then it appends their names into the list **lis** but deletes them from your script, **else** if the list is empty (hence len==0) it pops up with a message warning the user that there aren't any Transform nodes at all, which is unlikely for the majority of the nuke scripts actually but it was just an example :)
Moving down we ran into another '**if**' , this time it triggers a different nuke message in case you don't have any STEREO labeled node in your script and in fact it says "you are safe..." etc..

Well done ! it works ! Now try yourself on all possible circumstances and check if it is working

properly to you !

We can improve the script a little bit more by creating a popping nuke.message telling us how many nodes have been deleted. You know it is quite easy, we just need to add a line at the very end of the script that counts off the objects in the list and if the number is higher than zero a new message pops up.

```
lis=[]
a=nuke.allNodes('Transform')
if a:
      for xx in a:
            xx.setSelected(True)
            if "STEREO" in xx['label'].value():
                lis.append(xx)
                nuke.delete(xx)
else:
    nuke.message("you don't have any Transform node")

if len(lis)>0:
  nuke.message(str(len(lis))+ "   Transform Class nodes removed")

if len(lis)==0:
   nuke.message("you are safe ! there aren't any STEREO labeled node in your script!")
```

Now some of you might be wondering why I've put it at the very end of the script and not under the 8th line after the **nuke.delete(xx)**, well the reason is I don't like putting such commands within loops because if you try on doing so your popping message will be repeated many times. Honestly there are many other way around with it, like using functions, but you know we haven't covered it yet, so please try to get away with your current knowledge, which is a very good exercise indeed and you will savour the things even more in the future once we get into more advanced stuff.

But let's park it for now and move to something even more engaging, like: how about setting up a **$gui** expression for all selected nodes automatically ? you know that when we add a $gui Expression on a specific knob or to node's 'disable' knobs for instance it means that these will be working only during the render time, you know it helps on lessening the workload during the comping time.
So let's try this piece of code here:

```
k=nuke.selectedNodes()
if k:
  userInput = nuke.ask("would you like to set your nodes' disable expression to $gui ? ")
  if userInput:
      for x in k:
          x['disable'].setExpression('$gui')
  else:
    nuke.message("no nodes have been modified !!!")
```

```
else:
   nuke.message("there aren't any selected nodes!!")
```

On the third line I'm triggering a **nuke.ask** so the user can choose whether he wants to set the $gui expressions, hence on the 4th line I'm using a condition that literally means: "*if the user hits yes…*" - yeah I know the way it is written sounds really weird to you but this is the way it is.
...and if the user hits 'No' it returns with the sentence after the **else** command. How cool ?
It is quite looking like the last Python constructions we havecovered !
Now it must be a piece of cake for you !
...hey calm down!! what a piece of cake ? it's because you have learned a lot ! ;) ahahah ;)

Ok I'm giving away many Python tricks with this lesson, so do you want some more? :) well ok !
Assuming that you want to clean up your script or you need to clean someone else's, one thing to do might be to set default values for all nodes in the script, like' get back to their original colors and font size; this happens because sometimes when you are doing your comp you might want to change nodes' default colors only to make them standing out for some reason (like if you had temporary solutions or you wanted to highlight critical areas on your script, etc.), but now that you have done with it you want to get back with their default colors.
So basically on the few lines below we are calling back the default color number in hex color for a specific kind of nodes:

```
ss= nuke.defaultNodeColor("Grade")
for x in nuke.allNodes():
   if x.Class()=='Grade':
      x['tile_color'].setValue(ss)
```

Wow ! It works really well but it's currently limited to Grade Nodes and even if it gives us the opportunity to test conditions once more we have to think about making things easier and efficient ! so I could get to the same result by using just a couple of lines instead! Look at this:

```
for x in nuke.allNodes():
    x.knob('tile_color').setValue(0)
```

Please keep in mind that by assigning a zero value we are switching them back to their default color values !!! now let's set the default font size (that is 11) to our nodes:

```
for x in nuke.allNodes():
    x.knob('tile_color').setValue(0); x.knob('note_font_size').setValue(11)
```

Now the question is: what happens if I try to run a special routine on all nodes but some of them don't have a specific parameter ? ...mmm.. it makes me thinking back to a piece of code we saw on lesson 2; the problem was that if we ran that code it did something but it returned with an error:

```
for a in nuke.allNodes():
    a['note_font_size'].setValue(60)
    a['mix'].setValue(0.5)
```

The point is that we are running the code on all nodes but a few nuke's nodes, like the Viewer for instance, don't have the '**mix**' attribute, so this is the reason for it returning an error.
On that lesson we didn't know anything about conditions yet, but now we can sort it out !! Yay !

```
for a in nuke.allNodes():
  if 'mix' in a.knobs():
        a['note_font_size'].setValue(60)
        a['mix'].setValue(0.5)
```

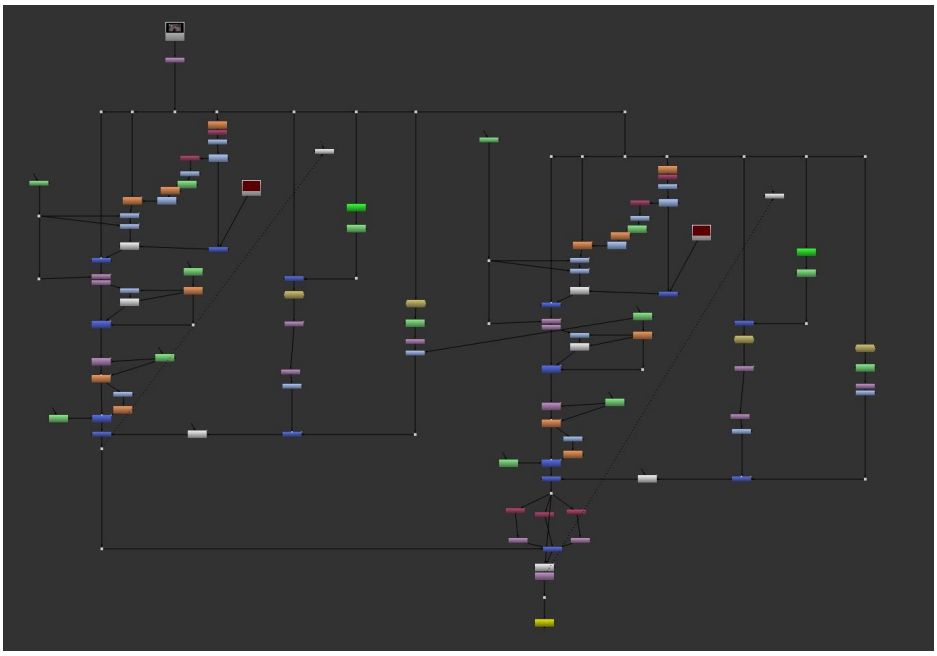
You see ? by adding just one line it now works like a charm ! basically it is checking whether the node has a specific knob, in this case the **'mix**' knob! Therefore if it doesn't the command won't be executed on those nodes without the knob!

There is something similar that worth a mention. Let's assume we need to turn off only a few nodes among specific classes, let's say those computationally heavy. There are different ways to do it, one of them uses Classes:

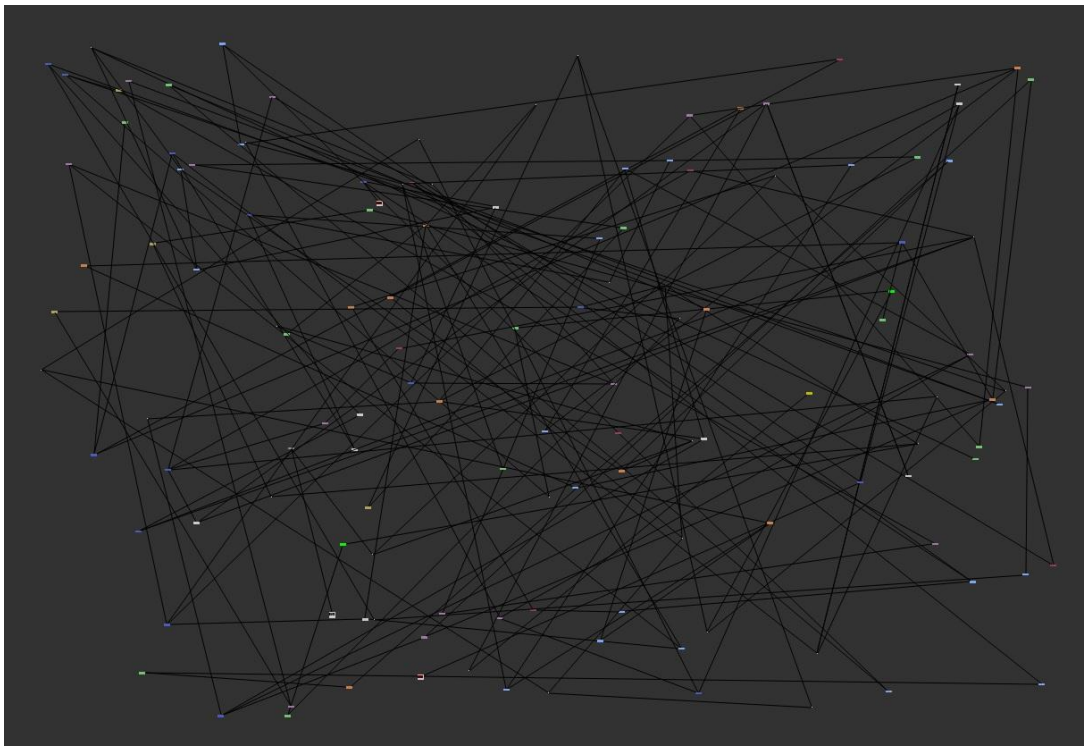
```
for s in nuke.allNodes():
   cct = ['Defocus' , 'VectorBlur', 'Convolve', 'oflow', 'TVIscale']
   for n in cct:
       if n in s.Class():
          s['disable'].setValue(1)
```

It is a quite straightforward and smart solution, I've just created a list and I wrote specific classes into it, just those I will be pointing at. So basically the script checks over all nodes' classes in the script and if it finds them belonging to any class in the list it disables all those nodes at once. Nice uh ?
It might look like a little weird to read because it is a kind of redundant code, but it works really well, in fact on the third line we are creating a variable ' **n** ' which point to the list '**cct**' and then we are telling Python that if this new variable **n** is in the Class of all nodes in the script (**s**) then disable them all.

...and now before wrapping up with this lesson I want to give you something funny to play with, it is a stupid thing and it's somehow nasty ahhaha, bug might be helpful to understand how things work into nuke with Python.
So let's say you have a nuke script and you want to mess up with all nodes' positions in the Graph view in a way it becomes like a spider web ! So we could turn something like this:



Into this:



Ahahaha ;) Well this is really cruel and please don’t even think of doing this in a company or you’ll be fired by end of day for sure!! By the way enjoy the simple code here:

```
import random
for a in nuke.allNodes():
  posx=int( 2* (random.uniform(500,7000)) )
  posy=int(random.uniform(300,9000))
  a.setXYpos( posx,posy)
```

Basically I'm randomizing all nodes positions, thus nuke is spreading them all around. ..Can you remember the **random** command we covered on the previous lesson ? that's it !
...and if you are crazy enough you could create a program to store each node position so you could come back to its original order :) a sort of antidote ahaha :)

...Or you could place all nodes on top all the others on the same position in your graph view:

```
for a in nuke.allNodes():
    a.setXYpos( 100,100)
```

Ahhaa madness !

On doing these things you are kind of protecting your script because nobody would spend so much time on tidying it up...but again, please don't do it! It is a nonsense …. just because there are weapons around it doesn't mean you have to kill people :)

...However there's more you can do, you could also hide all inputs for all your nodes as well as locking all links between nodes - I would also lock all knobs on all your nodes...
...wow I'm quite sure nobody would pick up a script like this!!

```
for a in nuke.allNodes():
  for i in a.allKnobs():
    i.setEnabled(False)

  if 'hide_input' in a.knobs():
      a.setSelected(True)
      a['hide_input'].setValue(1)
  a.setXYpos( 100,100)

nuke.Root().knob('lock_connections').setValue(1)
```

Ahha that was really a "don't do" stuff but it is funny, isn't it ?

And that's all folks ! and now before giving you the homework I just want to tell you something!

SURPRISE !!!!

this was the very last ‘free’ lesson!

The reason for that is that I’ve been asked to write a **proper book**, both printed and ebook/pdf with many other contents and insights!
My solemn pledge is keeping the price for the ebook really low and affordable for everyone, promised !

As you might understand this will take some time to come out as I still need to prepare many contents!! but I’m quite sure you won’t be disappointed with it if you choose to go for it.
Besides I’ll unveil the book index hopefully very soon !

For any information about it or simply for sharing your day thoughts or say hello please feel free to write to your old friend Gianluca here: **info@gianlucadentici.com**

And now here you are your homeworks, just keep in mind the solutions will be on the book !

Assignment1 :
Back to pages 3-4 we have created a script to explode all those groups’ nodes which had a particular node’s class within and we also created backdrop, Now try to figure out a way to color all those backdrops with the same red or black color.

Assignment2 :
On pages 6-7 we covered how to create a simple routine checking for Gizmo nodes. So now try and squeeze this code into the one who is looking for Gizmo nodes all over you script and explodes them all if the condition is true.

Assignment 3:
Take the script on page 17 and change it in a way the user can choose the color he likes for his BackdropNodes.

Assignment 4:
Assign a specific color to all nodes of your script with a particular label, let’s say “STEREO” but only for Transforms and TransformMasked nodes and put a Backdrop behind them, whereas any other node, always with a STEREO label, turns to a different color, let’s say a green or whatever you like the most. Other nodes without the label should be left as they are!

And ….

I really hope you enjoyed my free lessons and hopefully you felt in love with Python, and if you wanna know more about it, my book will help you out !

Take care of you !

I love you all ;)