# PYTHON SCRIPTING FOR SMART AND CURIOUS COMPOSITORS

Gianluca Dentici
www.gianlucadentici.com

## INTRODUCTION

Everybody welcome to this crazy “Python for smart and curious compositors” course. As already mentioned on previous messages on socials the final goal of these lessons won’t be to make a nerd programmer out of you, neither a pipeline managers, nor a compositor TD but taking you through some knowledge over useful code lines that can make our life easier on our daily basis work and hopefully giving you sufficient skills to write your own tools.

We will kick off  basic concepts that most of you might already know and will get to much more interesting stuff (hopefully) , by creating exciting coding for our outstanding comp scripts, therefore I won’t expect you falling from your chair with this very first lesson.

I have to add that this is my personal approach to Python for nuke but obviously there might be many other programming methods out there to get to same outcomes, simply the one I’m using is the easiest to me without overcomplicating with too many programming knowledge.

These lessons are totally free as I’m quite happy to share with our fantastic community, however the only thing I’m asking you is to keep sharing it to other colleagues all around the world if you reckon that it worth the case and just mentioning my name as the author and your best friend :-)

As many of you already know Python is a fair,easy-to-learn and effective programming language nowadays very popular and incorporated within the core of many graphics softwares and tools either for CG and Compositing and its power is really the blending of a quite simple syntax and the great integration with third party softwares, libraries and API. And it is exactly this latter feature who allows us dealing with customization and automation within nuke; “Automation” is maybe the most appropriate word for the kind of job Python is very good for and that’s the reason for many Python scripts for nuke being written just about to automate processes and/or repeating a specific function or procedure to all or some of the nodes of our scripts.
Just imagine how much time you could save by creating codes that could act on vast compositing scripts with more than 1000 nodes.

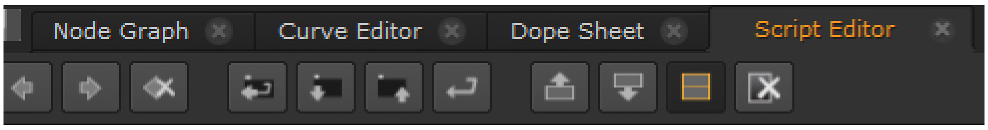
So now let’s start by going over the great history of Python from the beginning of the universe to present...ahahah just kidding ! If there is anybody out there particularly interested on it I would advice to peek into this:
https://en.wikipedia.org/wiki/Python_(programming_language)

I just wanted to add that during a lovely stroll into the colorful and vibrant Neil’s Yard in London I came across to a building where it is said Monty Python lived for some time. What a great buzz !

That said let's dip into Python for nuke !  yayyy !!

## LESSON 1

First of all I couldn't take for granted that everybody knows how and where to write python code within Nuke, well this is easy, we just need to open up the script editor within your Viewer interface. Personally I rather split the window so I can keep both the viewer and the script editor, in order to do this, you know,  just simply click on top-left icon and do it.
As soon as the script interface pops up you will notice that it comes with a few useful buttons to clean the history, execute the code, clearing the output window, saving script etc.



For more complex scripts I would advice to look for third-party tools that may help you even checking the syntax through entire scripts and being consistent with the API and libraries for the software you are scripting for, but for now we will get away with the Nuke standard scripting interface and honestly it works out for the majority of the script you may want to create for Nuke.

Let's start to learn how to create a simple node. First of all it is crucial to say that the python syntax used within nuke is obviously influenced, as already mentioned, by specific functions, constructs and modules of the software, therefore a good understanding of this stuff is really important for becoming more proficient with the code scripting. But we will go through it later on.

Now let's create a Grade node, so let's go to the script editor and type:

**nuke.nodes.Grade()**

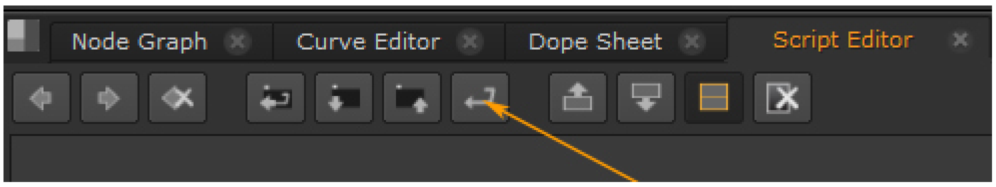
What does this construct mean ? so, nuke is the module we are calling in, whilst nodes point out that we are using a specific part of the module responsible for node creation, so it allows us to choose and build any nuke node we want, in this case a Grade. One important thing is: don't forget the capitol letter for each node's name. In general keep an eye on the spelling and capitol letters on scripting, this is very vital for a script to work.

My mom didn't gave me birth with the knowledge of all Nuke classes and when I was gently sleeping into my cot I was unaware that one day someone would have written nuke, neither that my life would have taken this direction, so honestly in order to know more about classes and modules I woke up, got rid of the soother and browsed this link:

https://learn.foundry.com/nuke/developers/70/pythonreference/

But for now don't mess with it, let's move on our code.

You can see parenthesis with the line of code I've just written, those are mandatory in the standard construct but they can be filled with other parameters that we will see in a min.
So now let's execute this line. In order to do so you can hit the provided button



or simply put your pointer at the end of the code line and ctrl+enter.

Once done a new Grade node pops up in all his glory into the node graph! Yay! Our first python generated node ! we can celebrate with some blue cheese but rest assured the things will get more complicated very soon.
So now let's see that we could create the same node by using a different syntax:

**nuke.createNode("Grade")**

The difference is that on the first case our node is simply created and it isn't attached to anything else... it could be even dispersed somewhere on your node graph and not selected. On the second case if we had a previous node selected it will be linked to it and selected.
We will use the first syntax for a few other examples, then we will switch to the other one once we got more familiar with the syntax.
So both construct can be used to create whatever nuke node you need ! so let's make other

nice examples:
So let’s assume I need a saturation node, I’ll write this:

```
nuke.createNode("Saturation")
```

Or a blur node

```
nuke.createNode("Blur")
```

*A*nd now it comes the time for a curiosity..let’s try to create a merge node. One could think of creating it this way:

```
nuke.createNode("Merge")
```

And if you execute the line it will create a merge node...right...but if you click on it to reveal its properties you will soon realize that it looks different with the standard merge node you are familiar with. This is happening because The Foundry has left the old merge node from previous releases, so in order to create the new one you have to write Merge2 instead of Merge and execute it - this will give you the right merge node.

```
nuke.createNode("Merge2")
```

There are other nodes who have a different python call comparing with the name they have on the toolbar, like the Exposure; in this case if we try to type nuke.createNode("Exposure") and execute it we will be given an error saying that it doesn’t exist.
His real python name is in fact EXPTool , therefore the right syntax will be:

```
nuke.createNode("EXPTool")
```

I decided to highlight this immediately before moving on because I know of some nice colleagues who freaked out with it and I’ve seen many of them wandering around Thames bank without a goal while looking for a plausible solution.

So now the smart question would be: how can I be sure to call the nodes with their proper names without a guessing game ? Fortunately there is a way out of this, let's figure it out: let's create a merge node in a traditional way by simply hitting the shortcut "m" on our keyboard, then by keeping it selected let's move to the script editor and type:

```
print nuke.selectedNode().Class()
```

What does it mean ?
Print is a specific command that actually prints something on screen, to be more precise it prints something within the script editor, then (nuke.selectedNode()) is a specific construct that performs an operation on a specific selected node, as it clearly suggests. Just try to memorize it as we are going to use it extensively very soon.
Class() is used to return the class to whom the node belongs.

In fact when executed this returns the class name for the selected node, in this case is Merge2.
Mystery sorted !

Now let's go back to our grade node.
So now why don't we create a new one but this time setting up a specific parameter of the node ? let's say the Gain ?
How to do that ? this is leading us to learn something new.

Some time ago we built a construct with 2 empty parentheses, so now has come the time to put something in the middle with specific parameters !

Let’s write and execute this:

```
nuke.nodes.Grade(white=1.5)
```

A new grade node pops up to the node graph with a gain value of 1.5……but...I can see your puzzled faces..the thing that upsets you is why that “white” instead of “gain”, right ?
Well the reason is easy, let’s investigate it. If you open up the node’s properties and hover the pointer on the gain value box, just right on top of the numeric value, you will see a small yellow-ish window popping up, and it says in bold: “white”

This means that the true python name assigned to this parameter and thus the one to be used in the scripting is “white”; this rule applies to all nuke node’s parameters !! therefore when you are about to write a script that is looking for a specific parameter, take a look at their names at first, just to make sure it is written like its real name or something different. There is an alternative way to get away with it by printing a list of all parameters’ names that come with a node. So now let’s learn how to do it.
First of all we need to select our grade node and then type this into the script editor:

```
print (nuke.selectedNode())
```

On a first view you may notice that it is quite similar with the previous code we used for printing node classes, but this time we simply omit the Class() statement. So now by executing it we will be presented a list of all parameters that come with the grade node, but there’s more! It prints also their current values.

```
channels rgb
blackpoint 0
blackpoint_panelDropped false
whitepoint 1
whitepoint_panelDropped false
black 0
black_panelDropped false
white 1.5
white_panelDropped false
multiply 1
multiply_panelDropped false
add 0
add_panelDropped false
gamma 1
gamma_panelDropped false
reverse false
black_clamp true
white_clamp false
maskChannelMask alpha
```

```
maskChannelInput none
inject false
invert_mask false
fringe false
process_mask false
unpremult none
invert_unpremult false
enable_mix_luminance true
mix_luminance 0
mix 1
```

This is how it should look like within your script editor.
Don't get scared with those "panelDropped false", they are just telling you that the current viewing status of the color tabs are collapsed. But if you try to click on the small color wheel icon and execute the code again it will say "panelDropped true", or if you click on the "4" to reveal the multiple value visualization and execute again the code it will return:

```
white {1.5 1.5 1.5 1.5}
```

For other parameters it simply means whether they are ticked or not.(true or false)

So now that we know all these little tricks let's go back to the node creation.
We learned how to put values for specific parameters, so now let's have some fun by adding some more. In order to do it we just need to separate them with commas.

Let's say:

```
nuke.nodes.Grade(white=1.5, black=0.01, gamma=2 )
```

And now let's try this out:

```
nuke.nodes.Grade(add=0.05, invert_mask='true')
```

What does it mean ? well, up to now we only edited numeric parameters but now we wanted to tick  a specific check box, in this case the one that inverts the input mask.
So we just need to put 'true' between quotes or 'false' otherwise. It is just simple like that !

Now let's see how selecting a value from a dropdown menu:
Let's assume we want a grade node and change the channels value from standard rgb to alpha. This is quite straightforward:

```
nuke.nodes.Grade(channels='alpha')
```

It's easy isn't it ?

So now that we learned how to create nodes with python we may experiment it on further kind of nodes. So let's create an emboss node for instance with Angle set to 50 and the Threshold to 0.3 - I would write it this way:

**nuke.nodes.Emboss(Angle=50, Threshold=0.3)**

So now why don't we name the node differently ?

**nuke.nodes.Emboss(Angle=50, Threshold=0.3, name='Best Emboss in the world')**

Giving a node's name is very important because it affects the way you will recall it from expressions for instance.

So now we could also change the font:
**nuke.nodes.Emboss(Angle=50, Threshold=0.3, name='Best emboss in the world', note_font=("Arial Black"))**

Don't forget to properly open and close parenthesis!

And now a particle emitter node ! let's do it with "start at" set to 50, "maxlifetime" to 100 and the parameter "emit from" set to "faces":

**nuke.nodes.ParticleEmitter(start_frame=50, lifetime=50, emit_from='faces')**

Awesome ! looks working pretty well!  but now let's try to create it with a little syntax variation:

**nuke.nodes.ParticleEmitter(start_frame=50, lifetime=50, emit_from=2)**

I'm sure you are wondering why I have created it by typing a value of 2 for the emit from parameter ? well if you take a look at the dropdown menu there are 4 choices and nuke counts them from 0, thus the one we are interested to is the number 2.

A pretty similar case is the DirBlurWrapper node, let's assume to set the  "BlurLength" to 13 and thel BlurType to "linear", we could write:

**nuke.nodes.DirBlurWrapper(BlurLength=13, BlurType='radial')**

or:

**nuke.nodes.DirBlurWrapper(BlurLength=13, BlurType=2)**

So now we got to the point where you should be able to call your nodes and set up your values. We have been using very basic construct and syntax though, so now let's move on to a more advanced method to build our nodes.

I would start again by creating a new grade node, but this time using the following:

**mygrade=nuke.nodes.Grade()**
**mygrade['white'].setValue(1.5)**

Ow wow, we've got 2 lines of code ! exciting !

What do they mean ?
So here we are getting into something new! We have created a variable with an arbitrary name mygrade; this variable has also created a grade node by using the construct we are already familiar with. This means that from now on when we need to edit values for the grade node or need to call it back all the way through the script we just need to use its new name mygrade.
The second line has a new syntax that sets up a white value, hence the gain of our grade node. Square brackets and parentheses are crucial as always so please respect the typing. Basically it is like we are saying: '*for the variable mygrade that now represent the Grade node, for its white parameter, set it to 1.5*'

.setValue is in fact the construct that assign a specific value for the selected parameter; This construct is very much used on python scripts for Nuke, so please make sure you get familiar with it.

Now let's have some fun and start to add other values to parameters. here:

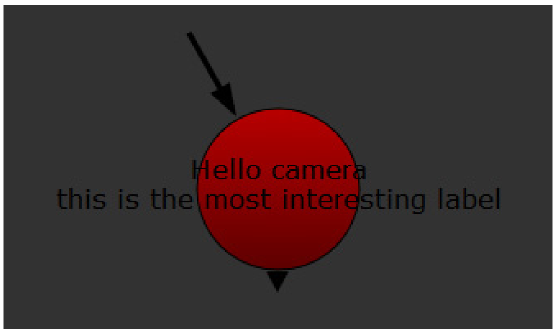
```
mygrade=nuke.nodes.Grade()
mygrade['white'].setValue(1.5)
mygrade['blackpoint'].setValue(0.5)
mygrade['add'].setValue(0.05)
mygrade['gamma'].setValue(1.5)
```

Now let's try to do the same thing but this time creating a Camera node and adding these translation values to x,y,z: 3,5 and 1, then let's change something else like the Display value to "solid+wireframe"

```
mycamera = nuke.nodes.Camera()
mycamera['translate'].setValue([3,5,1])
mycamera['display'].setValue(3)
```

It is relevant noticing how the xyz coordinates are sitting between the square brackets. Once again it is critically important to respect this syntax.
So now we could also add a label that will appear below the node name, hence the entire script would look like this:

```
mycamera = nuke.nodes.Camera()
mycamera['name'].setValue("Hello camera")
mycamera['translate'].setValue([3,2,1])
mycamera['display'].setValue
mycamera['label'].setValue('this is the most interesting label')
```



Another interesting case may be the creation and editing of a shuffle node:

```
w = nuke.nodes.Shuffle()
w['red'].setValue("black")
w['green'].setValue("green")
w['blue'].setValue("blue")
w['alpha'].setValue( "white" )
w['in'].setValue("rgb")
```

So we've just define a variable called w and called a shuffle node, then we decided to map the red channel on black and the alpha over a full white.

And how about writing a constant color node with python ? here you are:

```
w = nuke.nodes.Constant()
w['color'].setValue([0.5,1,0,1])
```

Where those values within the square brackets are RGBA values for the color.

Now would you like a crop node ? let's make it !

```
croppy=nuke.createNode("Crop")
croppy['box'].setValue([0,0,1920,1080])
croppy['reformat'].setValue('true')
```

By writing these lines we are creating a crop node and set up his reformat to full HD resolution of 1920x1080.

Now let's learn something really nice ! Now  we want to create a new grade node but we will turn his appearance to a feminine purple-ish color:

```
mygrade=nuke.nodes.Blur()
mygrade['tile_color'].setValue(4278255615)
```

The odd number you see within parenthesis is a specific color defined as Hex colors. So for now don't get crazy with it, we will learn how to convert rgb to hex colors very soon and also dealing with it much more easily. In the mean time if you want to make some practice with colors you could use these following values:

Red= 4278190335
Green= 16711935
Blue= 65535
Cyan= 16777215
Yellow= 4294902015

It has come the time for a further step ahead! So let's go deep some more into variables. Check the following lines:

```
wow=5+2
nuke.nodes.Emboss(Angle=50, Threshold=wow)
```

What does it mean? I've just declared a new variable called wow and put a value that is the sum of 5+2. Then I created an emboss node and instead of using a numeric value for the threshold i just wrote down the name of the variable. Obviously its value is 7.

At this point defining a variable for doing just that might look like useful like a rotten fish,

but try to think about all the possibilities we may have if instead of using the variable that gives a result of 5+2 we could place the value of a further variable! It means that its value would be changing accordingly with other parameters that influence the script. But this is another story and will be getting into this on other lessons to come.

So now let's take a look at this:

```
wow=2*3
mycamera = nuke.nodes.Camera()
mycamera['name'].setValue("Hello camera")
mycamera['translate'].setValue([wow,2,1])
```

Well the construct is quite similar to the previous one except for the fact that I used a variable instead of a number to define the X value for the camera translation.
So now X will have the value that comes out with the multiplication 2x3=6.

So now the following example shows how to use a variable to add some text to the node appearance; let's experiment with the following lines...but please first off all delete your previously created camera or it will turn out with a conflict error.

```
wow=2*3
kk=(' how are you')
mycamera = nuke.nodes.Camera()
mycamera['name'].setValue("Hello camera"+kk)
mycamera['translate'].setValue([wow,2,1])
```

By executing it we will have a new camera node with the text as follow: "hello camera how are you"
Please notice that I left 1 space before the word 'how' on the second line, it will accomodate a space between the two sentences.
However what this script would it be useful for ? Well try to think big on a bigger pipeline, basically you may want to decide to automatically append a specific suffix to a node, like whether the camera is final or wip or other comments. But this is part of a more advanced topic we will go through over the next lessons.

Let's move on ! Until now we just created and edited parameters over a single node, but now let's try to build 2 nodes and giving them values. So let's say we want to build a transform node and a color correct, both with names and specific values:

```
nuke.createNode("Transform", "name Smart_Transform")
desatNode=nuke.nuke.createNode("ColorCorrect", "name amazingColorcorrect")
desatNode['saturation'].setValue(0)
desatNode['gamma'].setValue(2)
```

As you may have noticed, after the transform node creation I put a "name Smart_transform", this is allowing us to assign a name to the node itself. Please note that I used an underscore between the words because leaving spaces inbetween is not allowed for node' names!

Now let's carry on and take a look at the following lines:

```
nuke.createNode("Transform",  "name Smart_Transform")
desatNode=nuke.nuke.createNode("ColorCorrect", "name amazingColorcorrect")
desatNode.knob('saturation').setValue(0)
desatNode.knob('gamma').setValue(2)
```

This script returns the same outcome, this is just an alternative way to write it and defining each parameter's value. By the way if you compare it with the previous one you'll find out that it looks quite similar, we are just describing parameters as Knobs.

For the sake of completeness we may also define a gamma value and at the same time a variable to it.

```
nuke.createNode("Transform",  "name Smart_Transform")
desatNode=nuke.nuke.createNode("ColorCorrect", "name amazingColorcorrect")
desatNode.knob('saturation').setValue(0)
gm= desatNode.knob('gamma').setValue(12)
```

gm is obviously the new name of the variable I've been using.

Doing that is important because for example after the color correct node I could add a blur node whose value is influenced by gamma's value. Look down here:

```
nuke.createNode("Transform",  "name Smart_Transform")
desatNode=nuke.nuke.createNode("ColorCorrect", "name amazingColorcorrect")
desatNode.knob('saturation').setValue(0)
gm= desatNode.knob('gamma').setValue(12)
gval=nuke.selectedNode().knob('gamma').getValue()
nuke.nodes.Blur(size=3+gval)
```

So now at the fifth line we found out a new statement called getValue() , this defines that the new gval variable will get and return the gamma's knob value. In fact on the following line we have created a new blur node and assigned a value that is 3+gval, so giving the gamma value at 12 the final blur node size will be 15. Does it make sense ?

To most of you these lines would look like awkward and redundant, and in fact they are, but I'm just playing with it just to make you understand some syntax and the way you could recall variables throughout python scripts and in particular within nuke. But let's come back to the script.
Now you may have noticed a boring issue, in fact by executing the lines the blur node isn't connected to the color correct node and this is happening because we are using a different construct for the node creation. Let's replace it with the other one we have already learned:

```
nuke.createNode("Transform",  "name Smart_Transform")
desatNode=nuke.nuke.createNode("ColorCorrect", "name amazingColorcorrect")
desatNode.knob('saturation').setValue(0)
gm= desatNode.knob('gamma').setValue(12)
gval=nuke.selectedNode().knob('gamma').getValue()
bl=nuke.createNode("Blur", "name myblur")
bl['size'].setValue(3+gval)
```

Ahhh now it works out !

Now let's see how to print the final value for the gamma by typing this into the script editor:

```
print nuke.toNode("myblur").knob('size').getValue()
```

Hold on a minute! What in the world is that nuke.toNode statement ?

Let’s learn something new ! so since we assigned a specific name to the blur node (myblur), we can tell python to get the value out from that specific node and print it on screen. Nice uh ?

Ok now I reckon we over-complicating even so these basic scripts, but I believe this was useful to get more familiar with the syntax. Regarding the complexity, sometimes scripts can be very long and taught to understand, so it is a good practice to add comments over some lines. In order to do it you just need to type a # followed by your text just after each line of code. This is an example:

```
nuke.createNode("Transform",  "name Smart_Transform") #creating the transform and a name
desatNode=nuke.nuke.createNode("ColorCorrect", "name amazingColorcorrect")
desatNode.knob('saturation').setValue(0)
gm= desatNode.knob('gamma').setValue(12)
gval=nuke.selectedNode().knob('gamma').getValue()
bl=nuke.createNode("Blur", "name myblur") #creates the blur, names it and create a variable
bl['size'].setValue(3+gval)
```

Ok now let’s have a break before dipping into the very last part of this lesson. Let’s have a tea with croissant.

Now just the time to wipe up our lips and we can get back to work with something new.

Until now we have been creating nuke standard nodes, so now why don’t we add any additional parameters within these nodes ? Let’s assume we want to create a standard blur node but adding a knob box into the user tab and with a specific value.
Let’s have a look:

```
bb=nuke.createNode("Blur")
k=nuke.Array_Knob ("blurmult", "multiplier")  #this creates the new user knob and names
```

```
bb.addKnob(k)  #adds the new knob k to the Blur node
bb['blurmult'].setValue(12)
```

Once executed let's look for the new node and click on it; you will find out that within the user tab now a new parameter called multiplier has been created, and its python name would be blurmult (as you know, you can check it by hovering the mouse on it), and yet we assigned a value of 12 to it.
So the nuke.Array_knob statement on the second line is responsible for creating the knob box, whilst the third line is responsible to append this box to the blur node the we defined as variable bb.

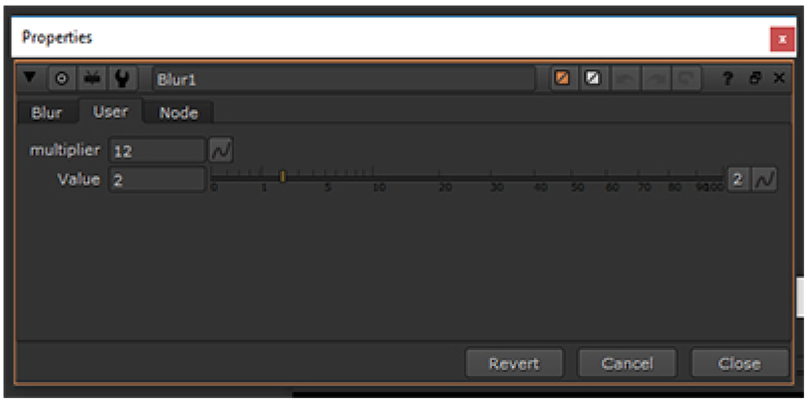
So now let's create a node with a slider within the tab 'user':

```
bb=nuke.createNode("Blur")
k=nuke.WH_Knob ("valueok", "Value") #this creates a user slider and names
bb.addKnob(k)
bb['valueok'].setValue(2)
```

Again the nuke.WH_Knob is this time the command to add a slider.

Now let's write a snippet that include both of them:

```
bb=nuke.createNode("Blur")
k=nuke.Array_Knob ("blurmult", "multiplier")
t=nuke.WH_Knob ("valueok", "Value")
bb.addKnob(k)
bb.addKnob(t)
bb['blurmult'].setValue(12)
bb['valueok'].setValue(2)
```



multiplier 12
Value 2

Now let's play with a text:

```
bb=nuke.createNode("Blur")
k=nuke.Text_Knob ("mioblur"," ")
```

```
bb.addKnob(k)
bb['mioblur'].setValue('ciao')
```

Wow wow, this time there is something weird with the syntax; you noticed 2 quotes on the second line just before the closing parenthesis. I left the space inbetween intentionally blank because I put my text later on within the 4th line.

And now last but not least how to create a tick box. Here you are:

```
bb=nuke.createNode("Blur")
k=nuke.Boolean_Knob ("mytick", "active!") #this creates the tick box
bb.addKnob(k)
```

So now before leaving you to your deserved rest of the day, let's carry on a bit more and figure out how to put an expression within a knob.
A good example could be the creation of a sphere that moves in the 3D space drawing a sort of "8" in the air. So you want to be sure to do it by using some sin and cos expressions.

Look at this:

```
sph=nuke.createNode("Sphere", "name mysphere")
sph['translate'].setExpression('sin(frame/5)',0)
sph['translate'].setExpression('cos(frame/10)',1)
sph['translate'].setExpression('22',2)
sph['rotate'].setValue(52,2)
```

Execute it, open up your 3d view and click f to center the sphere, hit play and enjoy!

So now let's go through it:
The first line create a 3D sphere in the space and gives a name to it.
The following 3 lines use the construct `setExpression` to set an expression for the translate values. The values we have within parenthesis need some explanation though.

`sin(frame/10)` is the kind of expression we want to use….but then…., this is followed by a number 0. What the heck is that ? well nothing magical, it refers to the first knob box of the translation, hence the X !
Same story on the following line and the third one , we've got a number 1 (refers to Y box) and 2 (means the Z box).
It should make you thinking back to the dropdown menus where the counting is the same.

We can also put the same value for all translation values at once by doing this:

```
sph=nuke.createNode("Sphere", "name mysphere")
sph['translate'].setExpression('sin(frame/5)')
```

Now remember when we have created a crop node with specific values ? Let's say we want to automatically set the resolution to the current project settings. In order to do it we can write an easy expression:

```
cr=nuke.nodes.Crop()
cr['box'].setExpression("format.w", 2)
cr['box'].setExpression("format.h", 3)
```

It is quite obvious that the two statement format.w and format.h will take from the current project settings values. This is very useful! Usually during the comp process we need to crop our final output before the writing node. Then, again, 2 and 3 means that it is setting the 3rd and 4th knob box (as we remember that nuke counting starts from 0).

**And that's all folks for this very first lesson ! we have set first steps with python for nuke syntax and basic constructs. Over the next lesson we will carry on with many other commands by learning how to build our first small pop up window as well as choice windows and obviously how to create loop scripts to automate many processes.**

## HOMEWORKS

Yes homeworks!
I would love you experimenting on your own over the basic topic we covered. Solutions will be unveiled on the following lesson !

So now your assignment is:

Create a constant blue color, connect it to a shuffle with the alpha set to black, then attach its output to a crop format set to your current script resolution and a write node with its font value at 33.
Don't mind the file out path for now, we will learn how to set it later on.

Enjoy!

See ya soon !!