Python for Nuke 101

Lesson 02 || Course Notes

INTRODUCTION

Last week, we learned about the necessary Python files, *menu.py and init.py* that nuke needs to run, and also started building some good habits in terms of commenting and version-controlling our code.

In this lesson, we're going to be:

- Setting knob defaults for individual nodes in Nuke
- Adding custom gizmos to a menu
- Adding custom keyboard shortcuts

CHAPTER 1: KNOB DEFAULTS

Timecode: *00:40*

Nuke is built with Python at its core, which means it's also very customizable with Python! Rather than submitting to working the way Nuke dictates you to work, let's start making Nuke work the way *we* want it to work!

First up, we want to enable a checkbox in *Edit* > *Preferences* > *Script Editor* called *"echo python commands to output window*. This will allow us to see the code as it's run, as well as any errors that might pop up along the way...

Changing the default values in nodes is one of the easiest and most effective ways to customize Nuke to your liking! Let's start with the Tracker node as an example, by running the following code in Nuke's script editor:

Timecode: *01:23*

```
nuke.knobDefault('Tracker4.shutteroffset', "centered")
```

Now create a *Tracker* node, go to the transform tab, and look at what's changed!

Notice how we start with `nuke.`? We're using the "nuke" Python module, which we already imported into our *menu.py* in Lesson 1!

`knobDefault()` is one of the many functions that are a part of the **nuke** module, and it expects two parameters, or arguments, inside the parentheses. The first is the node name + the knob name, separated by a dot, and the second is what we want the new value to be.

In our above example, we're setting the *shutter offset* knob in our *Tracker* node to be *centered* by default. This is a good default to add for any node with a *shutter offset* knob, as it mimics the way camera shutters & motion blur works in the real world.

But if we're changing knobs on a *Tracker* node, why are we calling *Tracker4*? This is an important distinction to make. In Nuke, things are "called", or referenced with Python, with a different name than the label, which is what you see things are called.

When we're calling a node, we're actually calling it's *Class* rather than it's *name*. Additionally, when we're calling a node's knob, we're calling it's *name*, instead of it's *label*. Confusing right? I promise with practice, you'll get used to differentiating them. The easiest way to find a node's *Class name* & *knob name* is by selecting it, and hitting "i". Let's do this on our Tracker.

Notice the second line starts with `Class=Tracker4`? That's what we're calling when we're writing our code. The reason its *Tracker4* instead of just *Tracker* is that, over the years as Nuke has been developed, the *Tracker* node has improved and gone through many iterations! We can actually still use the older versions by creating them with Python, but let's stay focused and leave that until next week…

Timecode: *05:11*

Let's try another example.

```
nuke.knobDefault('Tracker4.label', "Motion: [value transform]\nRef
Frame: [value reference_frame]")
```

This one looks more complex, but it really isn't! Every time we create a *Tracker* node, we'll now label it with the value of the Transform type & the Reference Frame, with the help of some simple TCL. The **\n** will be replaced with a line-break.

Timecode: *07:13*

What if we could set the *reference frame* knob to be the frame we're currently looking at, so we never have to click the "set to current frame" button? We can do that!

```
nuke.addOnUserCreate(lambda:nuke.thisNode()['reference_frame'].set
Value(nuke.frame()), nodeClass='Tracker4')
```

Now here's some new, much more confusing code!

The `addOnUserCreate()` function runs any time a User creates a node. This function also has two arguments: The first being the code we want to run, and the second is defining which node Class triggers the code to run. Let's break this down into smaller chunks.

`lambda` is basically a nameless, one-time use function. It's useful in situations like this one, where we just have 1 line of code that needs to be a function to run.

`nuke.thisNode()` is a function from the nuke module that returns the current node.

`['reference_frame']` is the knob we want to change.

`.setValue(nuke.frame())` sets the value of the affixed knob to whatever is inside the brackets. In this case, it's `nuke.frame()`, which is a function from the nuke module that returns the value of the current frame.

So all together,
`nuke.thisNode()['reference_frame'].setValue(nuke.frame())` is a 1-line

function that sets a *reference_frame* knob's value to the value of the current frame in the viewer.

Phwoah, this is only the first parameter in the `addOnUserCreate()` function! The second is `nodeClass='Tracker4'`. Meaning, every time a node with a *Class* of *Tracker4* is created, we run the code from the first parameter.

To reiterate, with only 3 lines of code, every time we create a *Tracker* node, it's going to automatically set it's reference frame to the current frame, it will label the node with some useful information, and will correct the silly *shutteroffset* value to *centered* -- a much more sensible default value!

Timecode: *10:14*

We don't want to type these 3 lines of code every time we start nuke -- we want it to be a default. To have it load up with Nuke, we have to copy/paste the code into our *menu.py*.

If you're still with me, congratulations! Given it's the first time being introduced to these concepts, I expect you to be scratching your head in confused curiosity, because we've quickly brushed over some examples of what *can* be done. Everything we've talked about so far will be taught again and reinforced in the coming lessons.

You're probably noticing that this is becoming a pattern? My aim throughout this course is to introduce you to wild new concepts, without overwhelming you from all the information flying at you at the same time. If you're struggling to follow along, you're only human! You've just learned something new, but there are many puzzle pieces towards understanding Python as a whole. Again, everything I've just taught

in the above `addOnUserCreate()` example will be explained more-thoroughly in time when we start using those bits and pieces of code more!

Let's go back to the basics, and set another knob default so you can see the simple side of things once again.

```
nuke.knobDefault('Blur.size', "2")
```

As you can see, this code is setting the default value of our Blur node's size knob to a value of 2!

Moving on…

CHAPTER 2: CREATING CUSTOM MENUS & ADDING ITEMS

Timecode: *11:15*

It's great to develop python scripts and handy gizmos, but what use are they if we don't have a place to hold & access them! Let's create two menus. First, we should create a "Utilities" menu to hold all our Python scripts.

```
utilitiesMenu = nuke.menu('Nuke').addMenu('Utilities')
```

Let's break this down. First, we're going to be adding a bunch of things to this menu in the future, so let's create it as a variable to make it easier to use going forward.

`nuke.menu('Nuke')` is referencing the top toolbar, where the file, edit, etc. menus are. The `menu()` function can reference any of the menus in Nuke, just by changing the argument. We'll come back to this when we create our next menu item.

`.addMenu('Utilities')` is running the `addMenu()` function, which to nobody's surprise, is adding a menu which we're naming, `'Utilities'`.

Run this in Nuke's script editor, and watch the magic happen!

Now, let's add something to our menu. There's this nifty python script in the **nukescripts** module called `autocrop()`. After selecting a node, it runs through your script frame by frame using the *CurveTool*, calculates the minimum bbox required, and outputs the data in a crop node. Super useful for speeding up your Nuke scripts!

In Nuke's script editor, you can run it by typing `nukescripts.autocrop()`, and hitting ctrl+enter to run. But we want to create a menu item in our new Utilities menu to do this!

```
utilitiesMenu.addCommand('Autocrop', 'nukescripts.autocrop()')
```

Remember how we assigned our new menu to a variable when we created it? This is where it comes in handy -- `utilitiesMenu` is referencing it, so we can run another function on it.

The `addCommand()` function adds a command to the aforementioned menu. Notice that it contains two arguments. The first, `'Autocrop'`, is the label we'll see in our Utilities menu. The second, `'nukescripts.autocrop()'`, is the code we want our menu item to run. In this case, it's the `autocrop()` function from the **nukescripts** module.

But wait, I hear you ask. Don't we have to import modules to use them? You're Correct! Since we're running all this in Nuke, *nukescripts* has already loaded. But we *do* need to `import nukescripts` it into our menu.py, as it's an external file.

`import nukescripts` just underneath the other imports in our *menu.py*. Then let's copy/paste the menu creation & autocrop item underneath. It's important to stay organized, so let's add some comments to all our new code whilst we're here.

Timecode: *16:02*

What if we have some custom gizmos to add into nuke? Let's create a new menu.

```
myGizmosMenu = nuke.menu('Nodes').addMenu('myGizmos',
icon=dir+"/icons/myGizmos_icon.png")
```

As we did before, we're creating our menu inside a variable for easy access later. We're creating a menu called `'myGizmos'` in the `'Nodes'` menu, with the same syntax as before!

Although this time, we're going to be adding a second argument, which adds an icon to our menu! Nuke's icons are all 24x24, so I recommend you to create yours the same going forward.

For this example, we're going to use this **Hand** icon I created, called **myGizmos_icon.png**. Why the hand? Well we're going to be adding Handy Gizmos to the menu. *(crickets chirping)*

This argument states `icon=dir+"/icons/myGizmos_icon.png`

We're specifying that we want to add an `icon`, and then we're giving it a filepath to where it's supposed to be. This starts with `dir` -- remember the automatic OS **dir** variable we created in Lesson 01? That's what we're referencing! We want to tell Nuke where our *.nuke* directory lives, and then add the subsequent directory.

As you can see, we're asking Nuke to look inside an `icons` directory, which we currently don't have… Let's go to our *.nuke* folder and add it, and put our icon image **myGizmos_icon.png** in there.

Let's restart Nuke and see what happens…

CHAPTER 3: KEYBOARD SHORTCUTS

Timecode: *18:54*

Searching for nodes through the menus, or even just hitting tab to search for nodes with long names can be unnecessarily cumbersome. Why not set some shortcuts for common nodes?

Compositors use the *Tracker* node all the time. Let's set a shortcut: **ctrl+alt+t**.

```
nuke.menu('Nodes').addCommand("Transform/Tracker4",
"nuke.createNode('Tracker4')", "ctrl+alt+t", icon="Tracker.png",
shortcutContext=2)
```

That's a long line of code to tell nuke "hey assign this hotkey to create a specific node". Oddly enough, the way to do it is by replacing a menu item with an identical one, that then creates a node programmatically. We'll get into this more in-depth next week, but for now, let's break down this code.

`nuke.menu('Nodes')` is referencing the sidebar holding all the nodes in Nuke. The `menu()` function can reference any of the menus in Nuke. Take a look at this lesson's cheat sheet to help add more.

The `.addCommand()` function adds a command to the aforementioned menu. There are many parameters we can add to this, with each being separated by a comma.

`"Transform/Tracker"` this parameter is looking for the *Tracker* menu item in the *Transform* nodes menu. We're going to be adding it again with our `addCommand()` function, which overwrites the default menu item.

`"nuke.createNode('Tracker4')"` is the function that gets executed when we choose the `Transform/Tracker` menu item, or choose it via the Tab menu. We're using the `createNode()` function to create a *Tracker4* node. Remember, we're referencing the Class, not the node's label...

`"Ctrl+alt+t"` adds the keyboard shortcut! If you want to use the "windows" key on the keyboard, type `meta`.

`icon="Tracker.png"` there are a bunch of icons that come with Nuke. Let's just use the one that's already there. If you want to check out any of the other icons, they live in the */plugins/icons* subdirectory, inside Nuke's installation directory. In windows, that's here: *C:\Program Files\Nuke11.3v3\plugins\icons*

`shortcutContext=2` Nuke changes context depending on where your cursor is. It seems pedantic to have to tell Nuke this, and it's not necessary to add, but it's actually quite helpful! It's what makes the "o" key create a *Roto* node in the *Node Graph*, and set the *Out Point* in the *Viewer*!

0 = Window
1 = Application
2 = Node Graph

You've probably noticed that we downloaded **Sublime** at the start of last week's lesson, but we're barely using it! When we're testing single lines of code or short snippets, it's a good idea to test-run them in Nuke's script editor first, in case it errors. If you make a mistake in your code inside your *menu.py*, Nuke will fail to open! So it's critical to test-run your code before committing it to your *menu.py*.

As we dive into more in-depth tools in the coming weeks, we'll be starting to make more use of **Sublime**, it's fancy syntax highlighting, and code shortcuts.

That’s it for this week!

WEEK 2 CHALLENGE

Timecode: *23:51*

- Find all nodes with a *shutter offset* knob, and set their defaults to *centered*.
- Find a way to set the FrameHold's first frame knob to the current frame when it's created
- Change the way you define the icon for your **myGizmos** menu, so it's not reliant on a specific file path. If you don't know where to begin, rewatch **Chapter 1** from **Lesson 01**
- Add a custom gizmo to your new **myGizmos** menu -- you can download one from my website if you don't have any handy: https://www.benmcewan.com/nukeTools.html
- Update your working code to a new version in Github.

The solutions will come with next week's lesson!