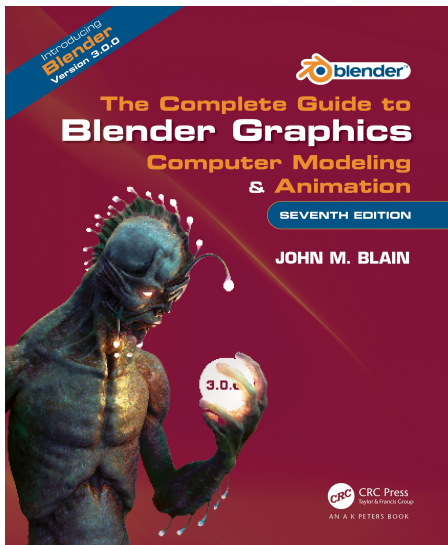


Drivers in Blender

Drivers in Blender is offered as a **FREE** supplement to:

The Complete Guide to Blender Graphics, 7th Edition



Blender Drivers are Functions or Scripts which use properties (values) to affect other properties. This control is particularly applicable when animating. Drivers are used to control the Animation of one property based on the value of another.

For example the rotation of one Object may be used to control the translation of another Object. This means that the Objects animated value is not controlled by the frame number interpolated from Keyframes, but rather by the data in a specified Animation Channel. Drivers can take their effects from single properties, differences in rotation, or scripted Python expressions which can be edited inside the User Interface controls.

Drivers are not limited to simple animated movements. You may use the X Location of a Driver of an Object to control the Material (color RGB curves) of another

Objects Material or use the Rotation of a Driver to control the Scale of an Object or use the Scale of a Driver to control the shape (through shape keys) of a mesh/curve/etc., use a Python function to control a constraint's influence, and much much more.

One key usage of Drivers is in Character Animation: for example, you can add Object Drivers to the Relative Shape Keys of a Face. Then, you manipulate the expressions of your Character just by moving the Driver Objects.

Examples and instruction provided on the internet often make the assumption that the viewer is conversant with Blender and has a reasonably advanced knowledge. It is, therefore, felt that a beginner may find a simplified instruction beneficial.

Blender is continually being developed with new features being incorporated and improvements made. In writing an instruction book it is difficult, if not impossible, to keep pace with the developments. The lead time from completing a manuscript to publication to release, prohibits some features from being included. **Drivers** were not included in the book and are, therefore, offered as a supplement to the book and free to those who are interested.

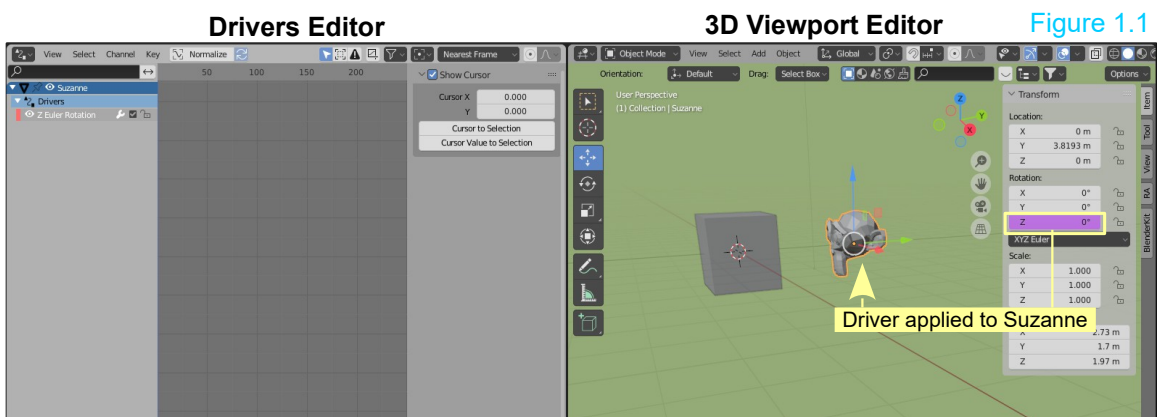
Blender Drivers

To understand the basic concept of using Drivers work through the following example.

A Driver will be set, causing a Monkey to rotate on its Z axis when a Cube is translated on its Y axis. The Driver will be applied to the Monkey Object (Suzanne).

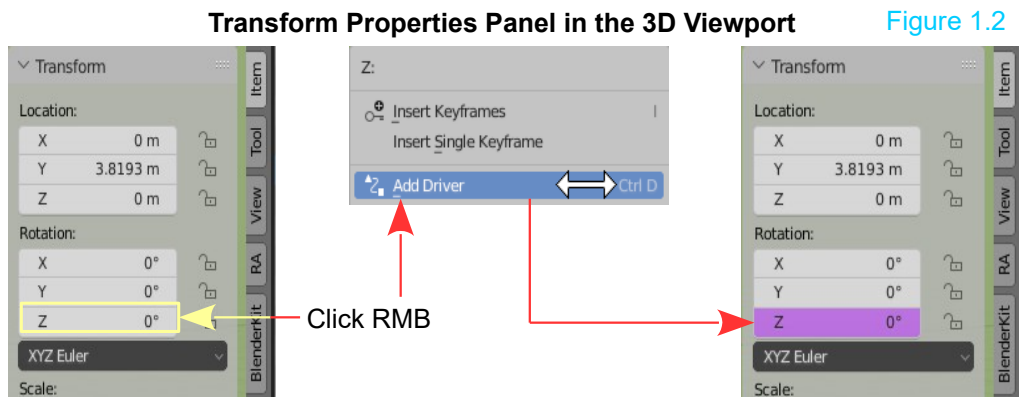
Open Blender with the default Scene containing the Cube object. Deselect the Cube and add a Monkey object. Move the Monkey to one side as shown in Figure 1.1.

Split the 3D Viewport Editor in two and make one part the **Driver Editor**. With the cursor in the Driver Editor press **N key** to display the **Drivers Properties panel**.

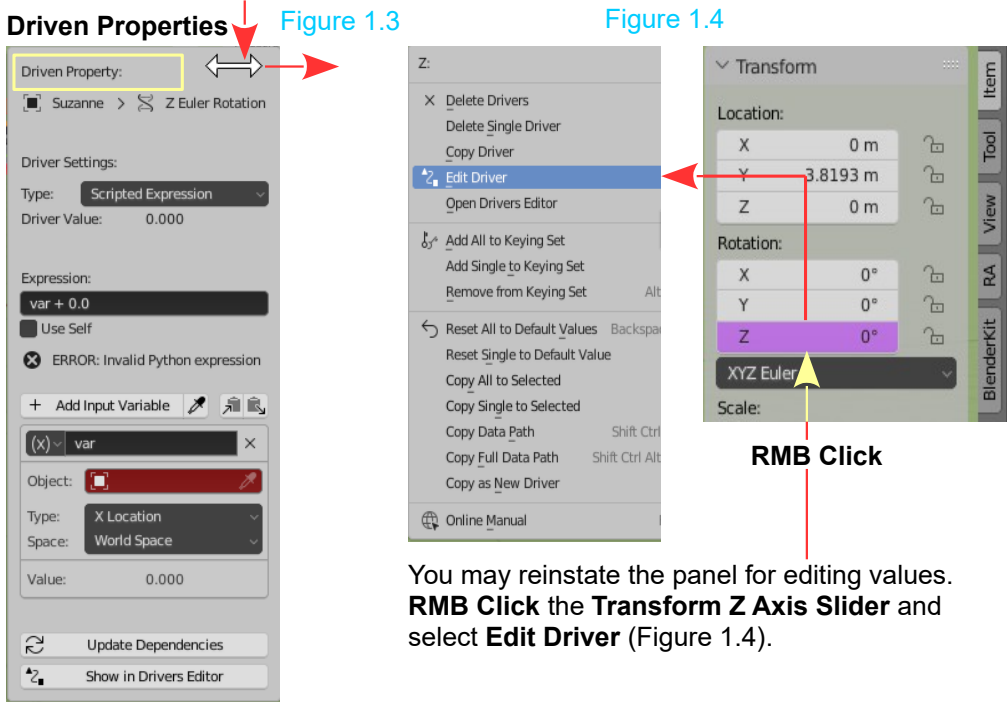


With the cursor in the 3D Viewport Editor press the **N key** to display the **Transform Properties Panel**. You now have a properties panels in the 3D Viewport and in the Drivers Editor. With the Mouse Cursor In the 3D Viewport Editor press the **T Key** to close the Tools panel and remove clutter from the left hand side of the Scene.

With the **Monkey Object** selected right click on the **Z Axis Rotation Slider** in the **Transform Properties Panel** (press N key). Select (click) **Add Driver**. The slider will turn purple showing that a driver has been added and the **Driven Property** panel displays (Figure 1.2).



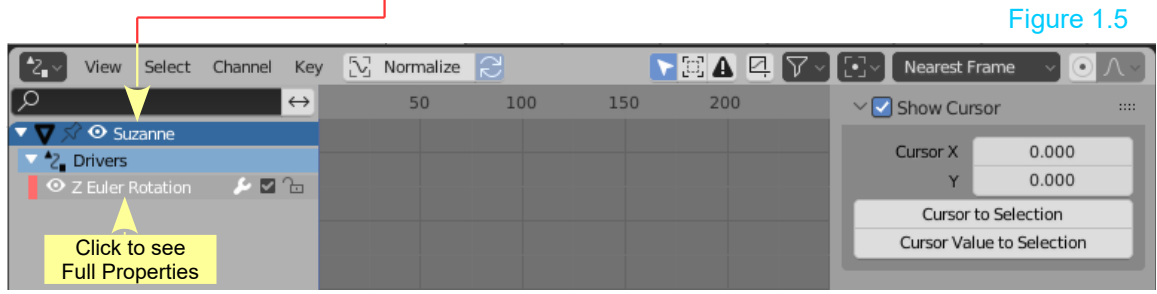
Note: Moving the **Mouse Cursor** out of the panel, causes the panel to disappear from view.



Adding a **Z Axis Driver** with the **Monkey** selected means that a change in the property values of a secondary Object, yet to be specified, will control the Z Axis Rotation of Suzanne. Specifying the secondary Object takes place in the **Drivers Editor**.

The Drivers Editor

The secondary action is defined in the **Drivers Editor** (Figure 1.5). The Drivers Editor (at this point) is showing the **Z Euler Rotation Driver** applied to **Suzanne** (the Monkey Object).

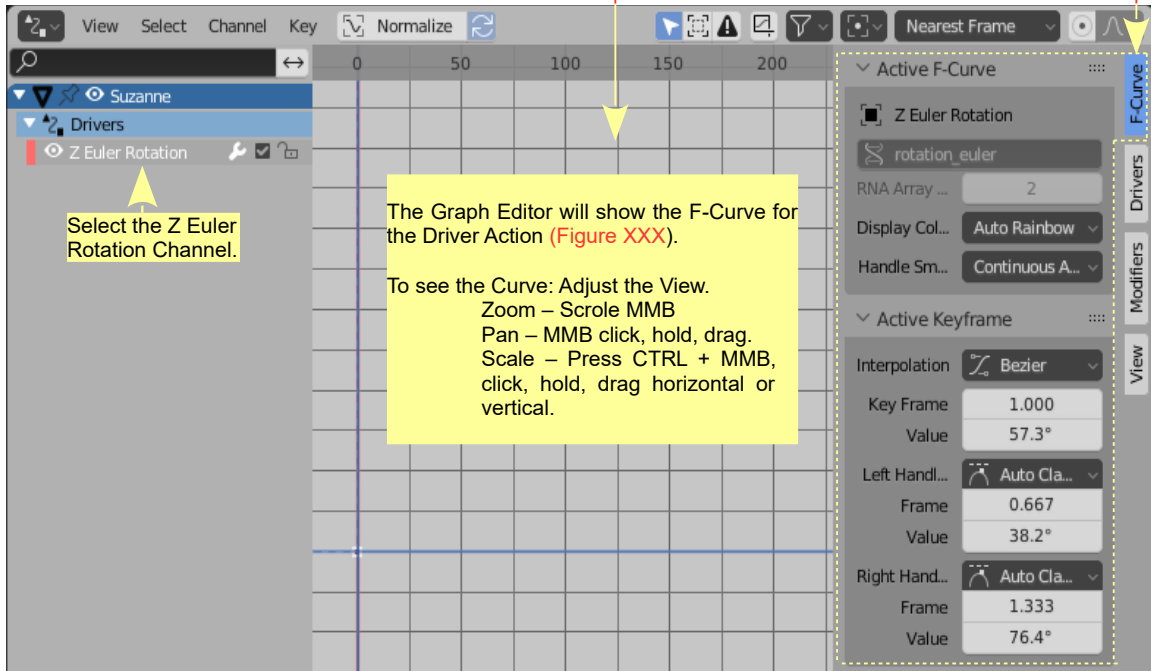


Where you see **Show Cursor** checked at the right hand side of the Editor is a copy of the **Driven Property View Tab**. To see the full Properties Panel for Driver Properties , click on **Z Euler Rotation**.



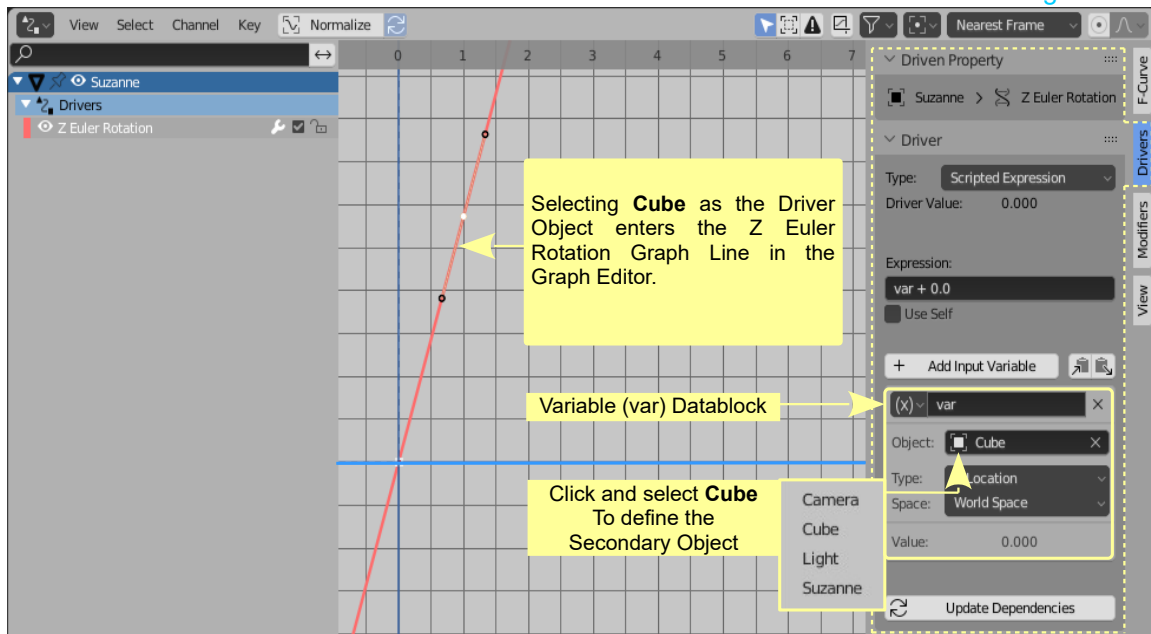
Clicking **Z Euler Rotation** expands the view showing Driver **Properties Tabs** with **F-Curve** selected. The center panel is the **Graph Editor**.

Figure 1.7



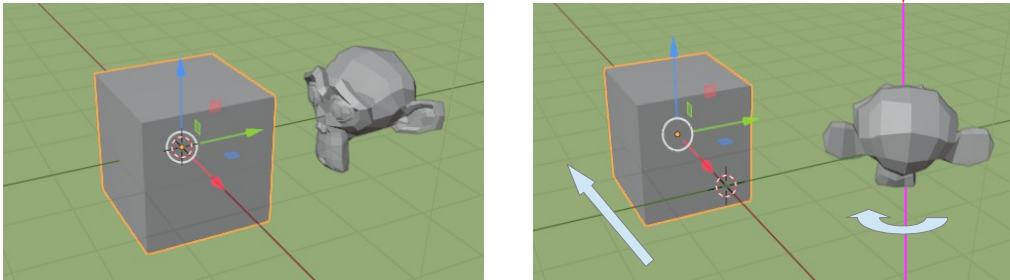
To set **Z Euler Rotation Drivers** for **Suzanne**, defining the secondary Object, change from the **F-Curve** Tab to the **Drivers** Tab (Figure 1.8).

Figure 1.8



By selecting the Cube in the 3D Viewport Editor and using the Move Tool, Translating the Cube along the X Axis of the Scene, Rotates Suzanne about her Z Axis.

Figure 1.9



This simple demonstration is the tip of the iceberg in understanding Drivers and merely shows that Drivers are a way to control values of properties by means of a function, or a mathematical expression.

The Blender Manual: The Blender 3.0.0 Manual provides an in depth discussion of Drivers with some excellent examples.

<https://docs.blender.org/manual/en/latest/animation/drivers/introduction.html>

Excellent Tutorial: One of many excellent tutorials on the internet describing the basic use of Drivers is:

<https://www.youtube.com/watch?v=N8GR9icb51w>

Expanding the Demonstration

Go back to the Transform Properties Panel in the 3D Viewport Editor with Suzanne selected and add a Driver for the Z Location Channel. In the Driver Editor set Cube as the driver Object.

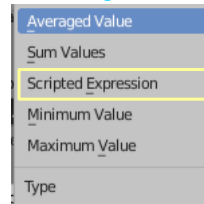
With the second driver added to Suzanne, Translating the Cube Object along the X Axis causes Suzanne to move up or down on the Z Axis while at the same time Rotating on about the Z Axis.

Figure 1.10

A Z Location Graph Line is inserted in the Graph Editor.



Note: In the **Driven Property** the **Driver Type** by default is; **Scripted Expression**. [Figure 1.11](#)



The Driver Type determines how the variables are used. The type can be:

- a built-in function: for example, the **sum of the variables'** values, or
- a **scripted expression**: an arbitrary Python expression that refers to the variables by their names.

Without going into a whole heap of Python scripting, for the moment, consider that when **var** is entered as the name of the variable value the Expression **var + 0.0** is executed by the driver causing Suzanne to Rotate when the Cube is moved on the X Axis of the Scene.

To gain an understanding of how the different settings and values affect the movement of Objects study the following which has been copied from the Blender 3.0 Manual.

Workflow & Examples

Simple Drivers can be configured from the pop-over that appears when adding a new Driver. When adding multiple Drivers or for more advanced configurations, it is useful to have open the Drivers Editor.

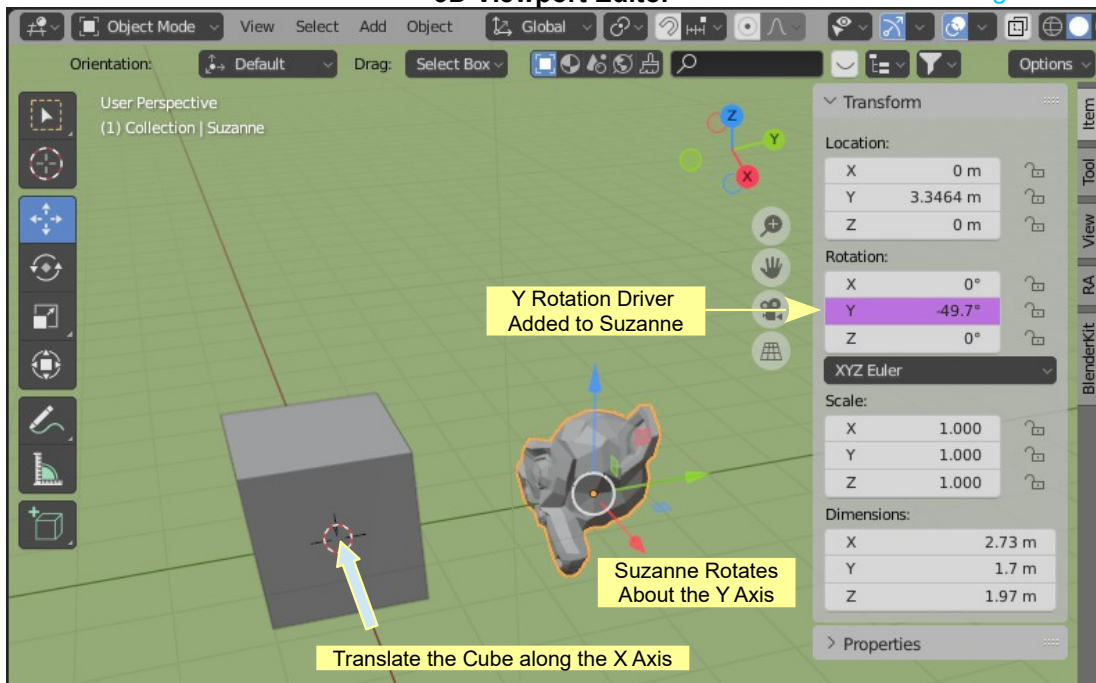
Transform Driver

Control a property with an object's transform. In this example, the Y rotation of Object 2 will be driven by the X position of Object 1. Starting from a simple setup with two objects:

Add a Driver to the Rotation Y property of the second object via the context menu or with Ctrl-D.

3D Viewport Editor

[Figure 1.12](#)



Driver Editor

Figure 1.13

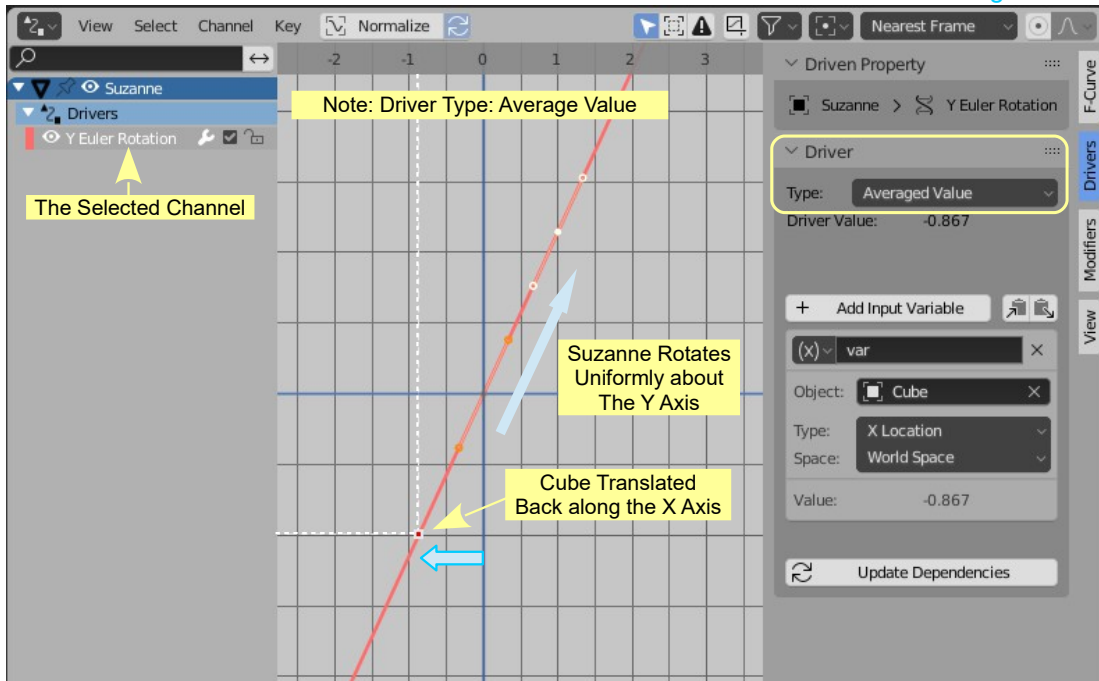
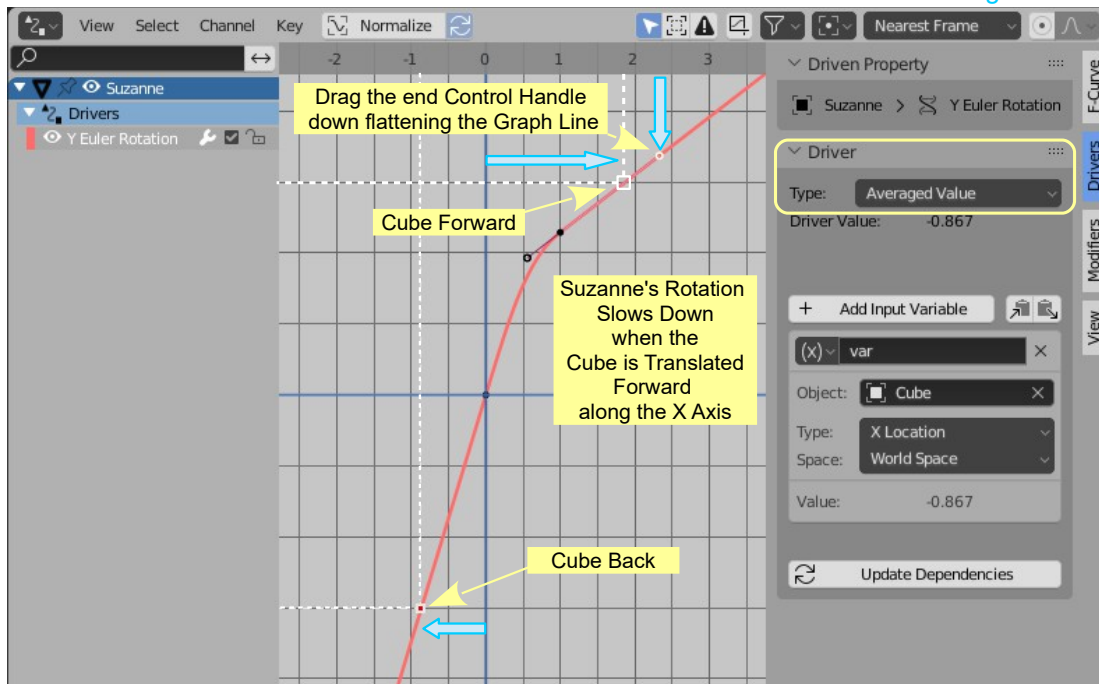
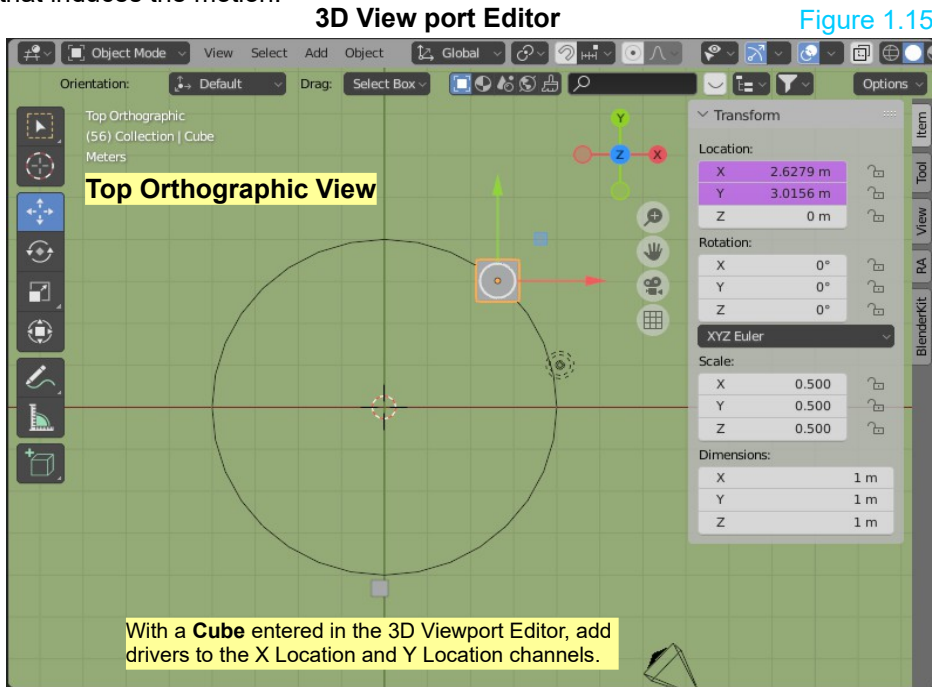


Figure 1.14



Scripted Expression – Orbit a Point

Orbit an Object's position (a Cube Object) around a point with a custom **Scripted Expression**. The Object's position will change when **scrubbing the timeline**. Circular motion can be defined in 2D using the **sine** and **cosine** functions. In this example, the current frame is used as the variable that induces the motion.



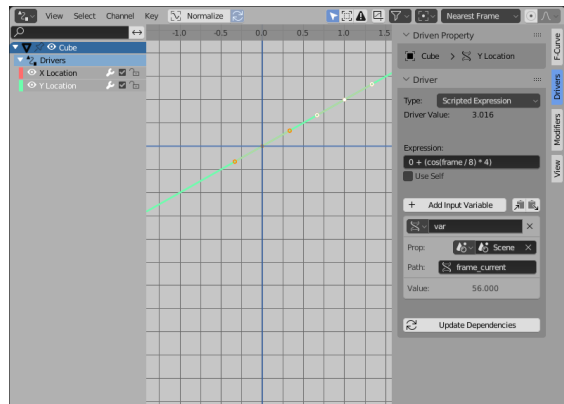
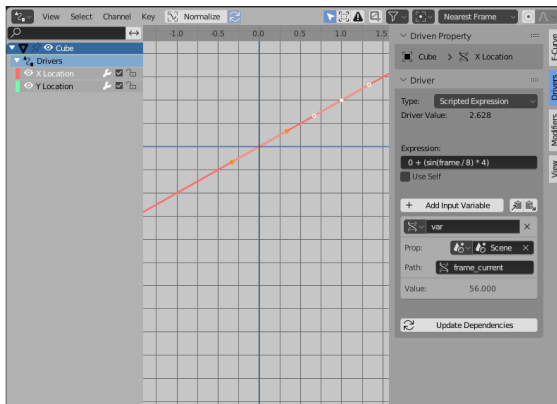
Adding Drivers in the 3D Viewport Editor for the X and Y Location Transform Properties of the Cube creates Drivers Driven Properties for for the X Location and the Y Location of the Cube. You see these Properties displayed in the Drivers Editor.

Drivers Editor

X Location Channel

Figure 1.16

Y Location Channel



To set the Driven Properties to have the Cube Orbit around the center of the Scene when the Timeline Editor Cursor is scrubbed you set values in the Driven Property Panels for both X and Y Location Channels.

Pay careful attention to the values. The slightest error, especially to the Expression value, will result in a failure.

Figure 1.17

Default Driven Property

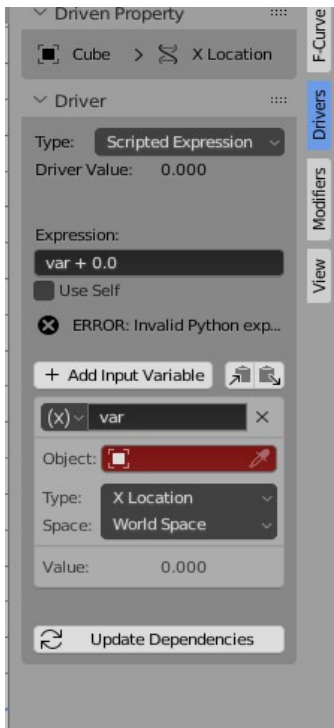


Figure 1.19

Modified X Location

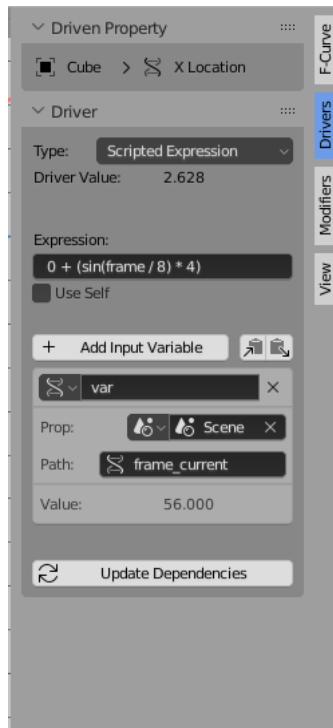


Figure 1.20

Modified Y Location

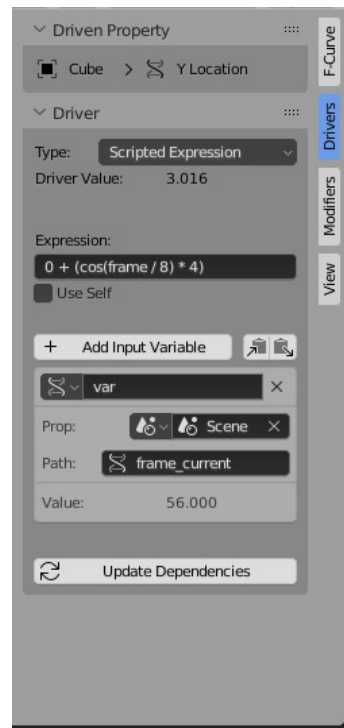


Figure XXX shows the default Driven Property when a Transform Driver is added to the Cube.

The procedure fore editing values for the X Location Driver and the Y Location Driver is identical except for the Expression. For X Location Driver change the Expression: var + 0.0 to:

$$0 + (\sin(\text{frame} / 8) * 4)$$

Important: Type the Expression exactly as shown. **Type sin NOT sine.**

For the Y Location Driver the Expression is: $0 + (\cos(\text{frame} / 8) * 4)$

When the Expression is entered the Error message in the default Driver display remains in the panel.

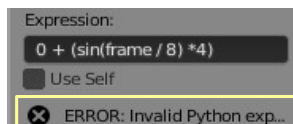


Figure 1.21

There is a little more setup to do before the expression will take effect.

Figure 1.22

Click-Select- **Single Property**

Remember
The procedure is identical
For the X and Y
Drivers

Click-Select- **Scene**

Click-Select- **Scene**

Type: **frame_current**

Value: 1.000

3D Viewport Editor

Scrubbing the Timeline
(Dragging the Cursor)
Orbits the Cube in the
3D Viewport Editor.

Playing the Animation
sees the Cube
continually Orbit orbit
the center of the Scene



The Scripted Expression: $0 + (\sin(\text{frame} / 8) * 4)$

$(\text{frame} / 8)$ is the current frame of the animation, divided by 8 to slow the orbit down.

$(\sin(\quad) * 4)$ multiplies the result of $\sin(\text{frame} / 8)$ by 4 for a larger orbit circle.

$0 +$ controls the offset to the orbit center point.

The forgoing is intended to wet your appetite for using Drivers. With an understanding of Drivers and perhaps a little bit of Python the results are limitless. To entice you further the following exercise will show how a **Python Script** is used.

Randomise Object Properties

The objective in this exercise is to set up a Blender file which may be used to create a Random Array of Objects, that is to create a number of similar objects which vary in their characteristics. To be specific, say you want to create a bunch of cubes with each cube slightly different in size, rotated and located in a different position in a scene. You do not require and specific differences, in fact you want a **Random Array of Objects**.

There are several ways in which you could create the array. You could simply duplicate the Cube in the 3D Viewport Editor then Scale, Rotate and Translate and change the Material. Keep on doing this until you have sufficient Cubes. To help with changing the Objects size you could set up **Shapes Keys**. To help with duplicating the Cubes you could use an **Array Modifier** or multiple **Array Modifiers**. Needless to say these methods are somewhat tedious.

The following instructions will demonstrate how to set up a **Blender File** including a **Python Script** which may be used with **Drivers** to create an **Array**.

Create the Blender File

In creating the Blender File you register a **Python Script** in the File. This means that this particular Blender File, when saved, will contain the Python Script. The File will then be available to be used for creating Random Arrays at a later date.

Create a Python Script

A Python Script is a piece of code written in the **Python Computer Language** and is simply a text file. In this exercise the script shown in [Figure 29.5](#) will be used.

Type the text shown in the diagram into a Text Editor such as Notepad or Wordpad and **save the text file**. You will copy and paste this into a Blender file. You can type it directly into the Blender text editor but having it saved as a text file gives you a back up. Make sure the text is copied **exactly** as shown. The slightest error will cause an error when running the script.

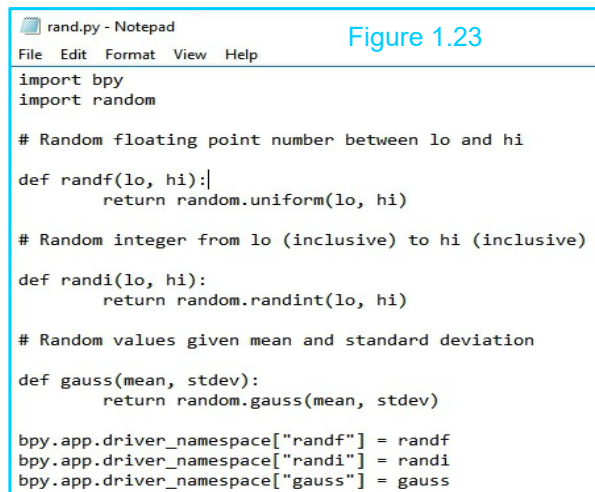


Figure 1.23

```
import bpy
import random

# Random floating point number between lo and hi
def randf(lo, hi):
    return random.uniform(lo, hi)

# Random integer from lo (inclusive) to hi (inclusive)
def randi(lo, hi):
    return random.randint(lo, hi)

# Random values given mean and standard deviation
def gauss(mean, stdev):
    return random.gauss(mean, stdev)

bpy.app.driver_namespace["randf"] = randf
bpy.app.driver_namespace["randi"] = randi
bpy.app.driver_namespace["gauss"] = gauss
```

The Python Script will be used in conjunction with the **Blender Driver Functions**. Remember where you have saved the Text File and the name of the File. The name of the File in this exercise is **rand.py**.

Blender Driver Functions

In simple terms **Drivers** are functions which affect the attributes or properties of an Object. Refer to the demonstration at the beginning of this paper where the translation of one Object controls the rotation of another. In that instance the position of one Object in the Scene controlled the rotation of the other. Instead of using the translation of an object the **Python Script**, will be introduced to the **Driver** to control properties of Objects.

Registering the Script

The first step is to **Enter** and **Register** the **Python Script** in a Blender file. This will create a Blender file which you can save for future use.

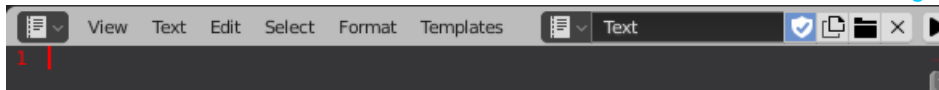
Open a new Blender file and open the **Text Editor**. Create a new **Text Block** by clicking **New** in the window header or by clicking **Text - Create New Text Block**.

Figure 1.24



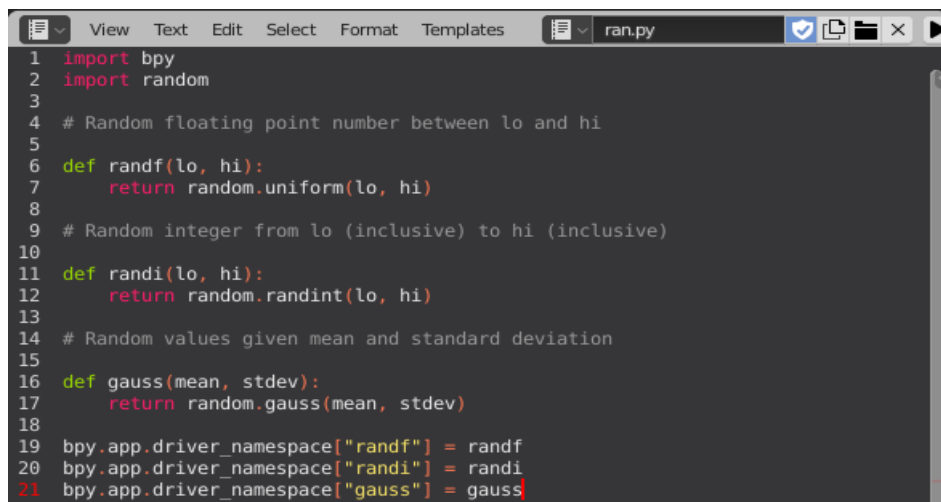
The default name of the new text block is show simply as **Text**. Rename this to something more significant. Since you are about to work with random properties and a Python Script, **ran.py**, is appropriate. Note the suffix **.py** is very important, therefore make sure you include this in the name.

Figure 1.25



With the text block created go get your Python Script. That is, go to the text file you previously created. Open the file in the Text Editor you used and select (highlight) the text and copy it to the clipboard. Paste the text into the newly created text block in the Blender **Text Editor**.

Figure 1.26



In the Text Editor Header click **Run Script** then check **Register**.

Run Script

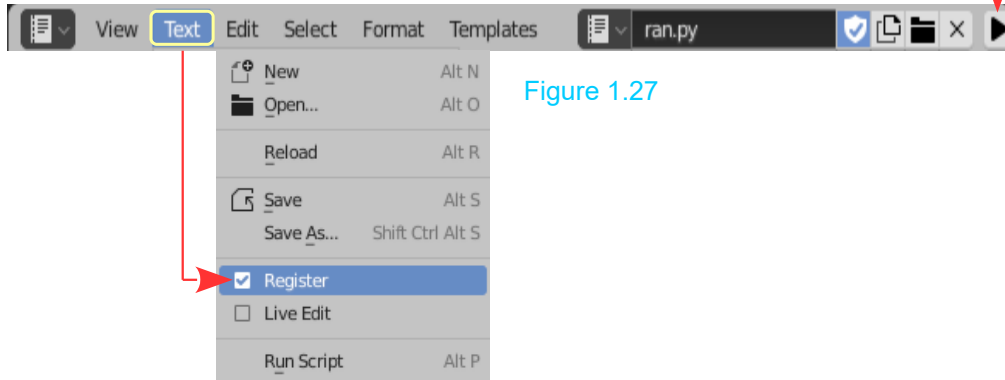


Figure 1.27

Run Script will make the functions contained in the Script available to the Driver. The functions in the Python Script are the **randf**, **randi** and **gauss** bits (functions). Register means that the next time you open the Blender file it will run the script and register automatically.

Save the Blender File with the name **RandomPy.blend**.

Registering the Python Script makes functions in the script (**randf**, **randi** and **gauss**) available for use by the Blender Drivers. Either of these functions can be called (entered into Blender) for use by a driver. Saving the **RandomPy.blend** file means you now have a Blender file available for generating random properties. How the **randf (lo, hi)** part of the Python Script works will be demonstrated. This is the only part being used in this demonstration. Including the **randi (lo, hi)** part and the **gauss (mean, stdev)** parts merely makes them available should you wish to use them in the future. To understand these statements you will have to undertake a study of the Python programming language.

OK! with the **RandomPy.blend** file open you can close the **Text Editor** and return the **3D Viewport Editor**.

Note: If you close the RandomPy.blend file and reopen it at a later date a notice may display in the Info window header stating **For security reasons, automatic execution of scripts in this file was disabled**. This occurs if **Auto Run Python Scripts** is disabled in the **Preferences Editor**.

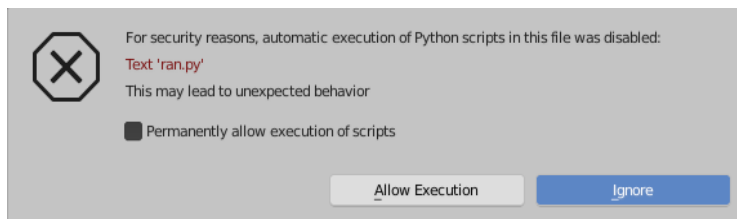
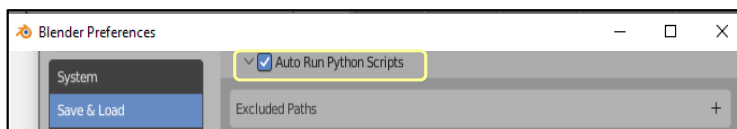


Figure 1.28



At this point it will be assumed that you previously save the Blender File named RandomPy.blend and now wish to use the File to create a Random Display of Objects. With the Array created you can Append the Array into other Blender Files.

Arrange the 3D Viewport Editor and Driver Editor

Have Blender KeyPy.blend opened with the Blender Screen arranged as shown in Figure 1.1 with the Driver Editor and 3D Viewport Editors side by side. Note: This arrangement is purely a matter of choice and will depend on your Screen size and preference.

With the mouse cursor in the 3D Viewport press the **N Key** to display the **Transform Properties Panel**. **Note:** This Properties Panel displays the same information as the **Object Properties** in the **Properties Editor, Object Properties Tab**. Drivers may be added to an Object in either Panel.

Save the File, that is save the current version of RandomPy.blend, as (Save As) a new File with a name of your choice relevant to the Array you are creating, For example; **Random_Cubes.blend**.

Saving a new File leaves the original File (RandomPy.blend) intact, for future use.

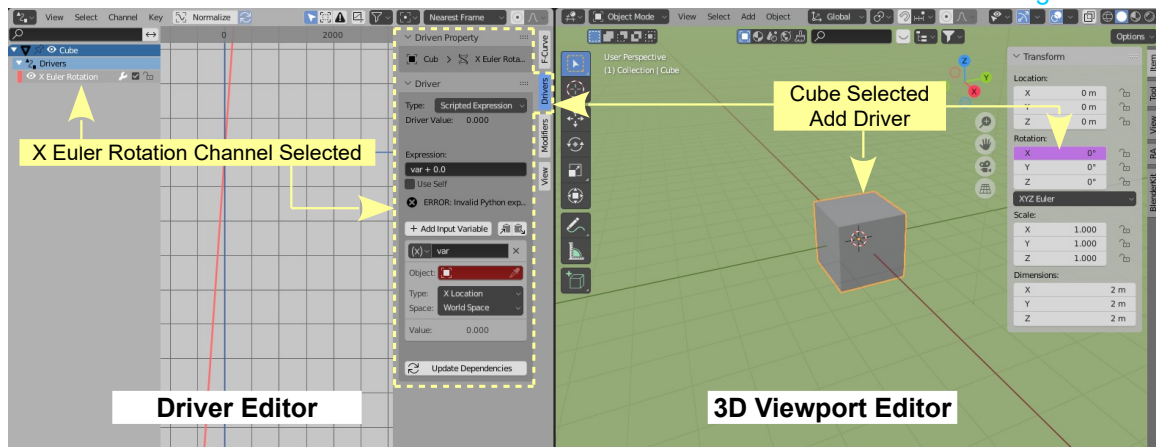
OK you have **Random_Cubes.blend** opened ready to go.

Note: The Array to be created will be constructed in stages by Adding Property Drivers and Expressions for different facets of the Array. To begin a **Rotation Driver** will be engaged.

Rotation Driver Properties

Begin by setting a driver for the X Rotation Property of the default Cube Object in the 3D Viewport. Make sure your Object, the Cube, is selected. In the **Object Properties panel of the 3D Viewport Editor**, right mouse click on **X Rotation** and select **Add Driver** (Figure 1.29).

Figure 1.29



At this point the Driver does nothing since you have to instruct it to call the Python Script function.

Remember: The Blender File named **RandomPy.blend** has been created with the Python Script named **ran.py** entered and registered. You have then copied the File and saved it as **Random_Cubes.blend**. Since it is a copy of RandomPy.blend it also has the Python Script ran.py entered and registered.

Attempting to create an Array in a new Blender File without the Python Script entered and registered will fail.

Enter the Python Script Function

You are using a **Scripted Expression** (Python Script) which has been entered and registered in the Blender File named Random_Cubes.blend (a copy of RandomPy.blend). The Python Script contains several functions.

Figure 1.30

```

1 import bpy
2 import random
3
4 # Random floating point number between lo and hi
5
6 def randf(lo, hi):
7     return random.uniform(lo, hi)
8
9 # Random integer from lo (inclusive) to hi (inclusive)
10
11 def randi(lo, hi):
12     return random.randint(lo, hi)
13
14 # Random values given mean and standard deviation
15
16 def gauss(mean, stdev):
17     return random.gauss(mean, stdev)
18
19 bpy.app.driver_namespace["randf"] = randf
20 bpy.app.driver_namespace["randi"] = randi
21 bpy.app.driver_namespace["gauss"] = gauss
  
```

You have Added an **X Axis Rotation Driver** to the Cube Object. The Driver Type is: **Scripted Expression**. The objective in this first stage of creating an Array is to generate a Random Rotation of the Cube. This means that a change to the status of the Cube, such as, it's location in the Scene will cause the Cube to randomly Rotate.

To have the Driver execute this Random Rotation you call a function in the Python Script, in this instance, **def randf(lo, hi)**.

Read this as, **rand** [random], **f** [function], **(lo, hi)** [lower value to higher value] which you enter in the **Driver Property Panel** as the **Expression: randf(-pi, pi)**

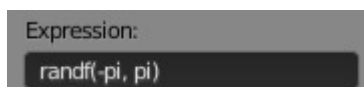
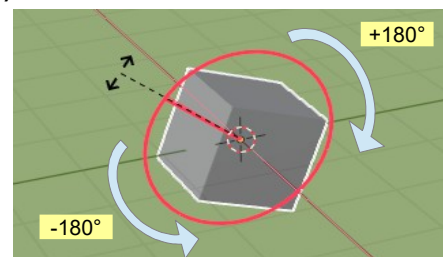


Figure 1.31

pi (in Python) = π = 3.142. There are 2π Radians in 360° , therefore, $(-\pi, \pi) = -180^\circ$ to $+180^\circ$.



Immediately you enter the Expression you will see the Cube Rotate about the X Axis in the 3D Viewport. Each time you click **Update Dependencies** in the **Driver Property** panel the Cube Rotates to a new orientation. With the Cube selected in the 3D Viewport, pressing the G Key and dragging the Mouse sees the Cube Rotate to a new orientation.

Entering the function is in effect telling Blender to use **randf** expression of your Python Script with the arguments **-pi** and **+pi** to recalculate a random value of rotation about the Cubes X Axis within the range minus π to plus π . In other words pick a rotational value about the X Axis between 0° and 360° since there are 2π radians in a circle (Arguments are values that an expression uses in its calculation).

By adding Drivers to the Y Axis and Z Axis Rotation Channels and entering `randf(-pi, pi)` as the Expression in the Driver Property Panel the Cube will Rotate Randomly about all three Axis.

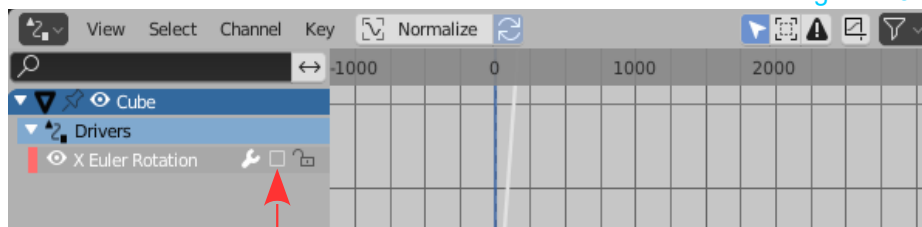
Note: When using the **randf(hi, lo)** Expression the **hi lo** values are not limited to Rotational values. If you are using the Expression to affect the Scale of an Object you would use numeric values.

When entering values in the Expression be careful when pressing the . (period) Key and , (comma) Key. Its too easy to press the wrong Key.

Re-evaluating Drivers

With Drivers set , in the 3D Viewport, pressing G key (Grab) and dragging the Mouse moves the Object in the Scene. Blender constantly re-evaluates the Driver and produces random values for the properties values. Note: This will affect any Driver which has the “randf”, “randi”, or “gauss” Python expression inserted. This function can be negated by unchecking the **F-Curve Contribution button** in the Driver Editor. You will want to negate this function if you require to manually reposition the cube without it Rotating.

Figure 1.32



Uncheck the F-Curve Contribution Button

Scale Properties Driver

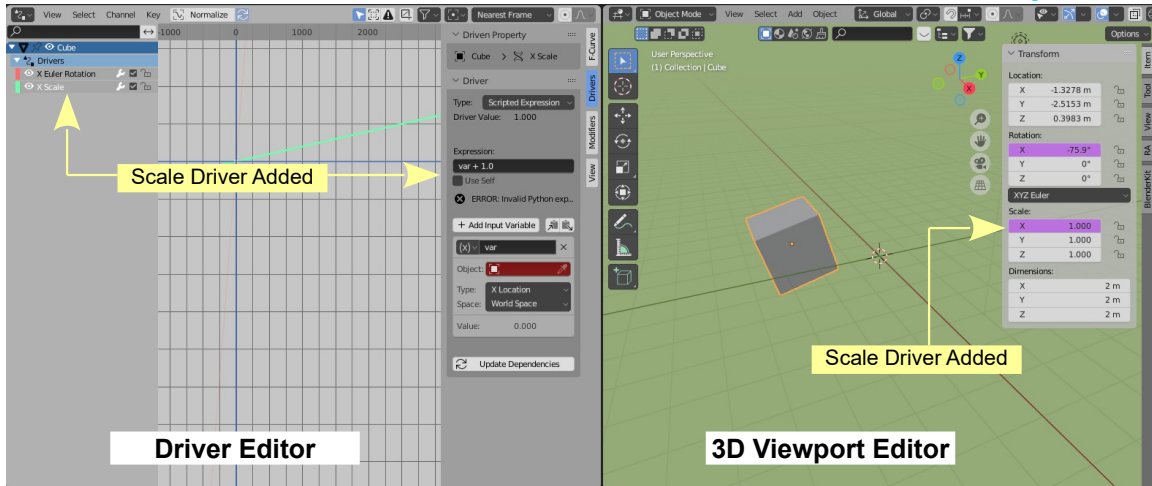
Up to this point Rotation Drivers have been evaluated but, in fact, any Object Property can have a Driver added and use a Python Script Expression.

You can create Scale Drivers and set up a process where the X, Y and Z Axis all Scale in proportion to each other. To show how this works a different process will be demonstrated for each axis. You may however use any one of the processes on each axis.

X Axis Scale Driver (Scripted Expression)

Start by adding the **X Axis Scale Driver** using Scripted Expression. In the 3D Viewport, Object Properties panel, RMB click on **Scale: X Axis**, select **Add Driver** etc. as before. A new Driver Channel is entered in the Driver Editor and again, clicking on the Channel displays the Driven Property Panel. The Tabs displayed are now applicable to the X Axis Scale of the Cube in the 3D Viewport (Figure 1.33).

Figure 1.33



This set up is the same as you used for the X Axis Rotation with the exception that you insert the values 0.8 and 1.8 instead of $-\pi$ and π in the randf Expression

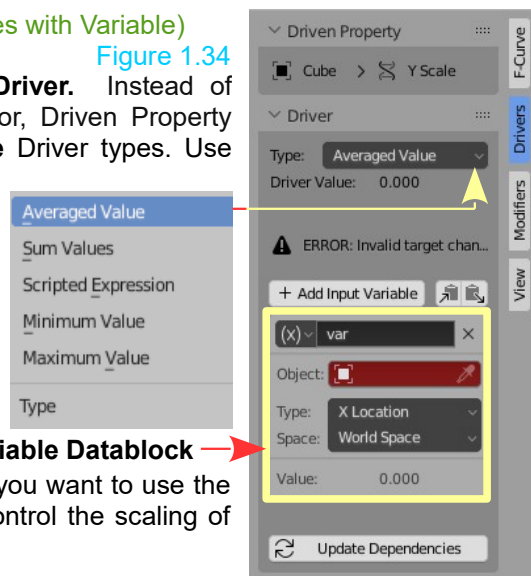
The X Scale value will now be randomly re-evaluated between 0.8 and 1.8 Blender units when you click **Update Dependencies**. You can set any values you like but don't get carried away.

Y Axis Scale Driver (Max, Min, Sum, Average values with Variable)

Figure 1.34

On the **Y Axis Scale** values, Right click **Add Driver**. Instead of selecting **Scripted Expression** in the Driver Editor, Driven Property Panel, choose either **Max**, **Min**, **Sum** or **Average** Driver types. Use **Average Value** (Figure 1.34).

This is where the **Variable Data-block** in the **Driven Properties Panel** is used. **Note:** An error message displays until data is entered in the Variable Datablock and you click Update Dependencies (Figure 1.34).



OK! You are setting up the Y Axis Scale Driver but you want to use the X Axis Scale Data via the Variable Datablock to control the scaling of the Cube on the Y axis.

Note: Since you are referring to the X Axis Scale to control the Y Axis Scale you must have the X Axis Scale Driver in place.

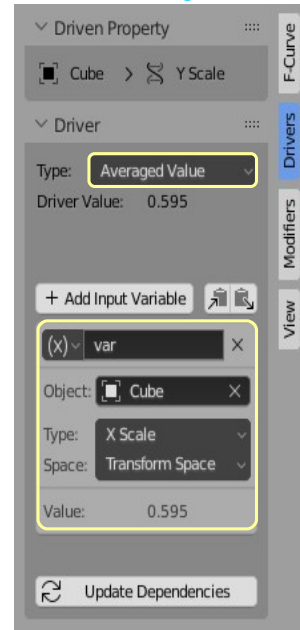
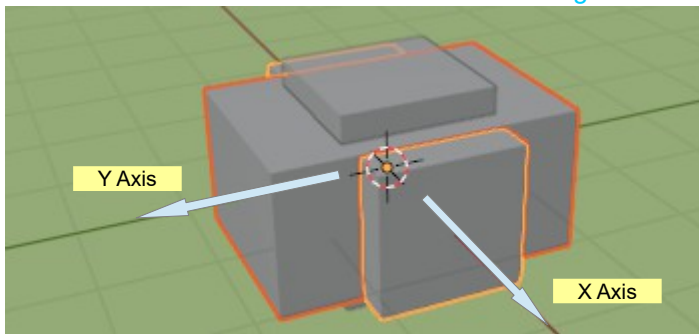
Figure 1.35

Enter the following in the Variable Datablock with the **Cube** selected (The cube object being used) in the Specific Property I.D. block.

Use: **Transform Channel**. Type: **X Scale**. Space: **Transform Space** (Figure 29.14)

At this point clicking **Update Dependencies** produces a random Scale change along the X Axis of the Cube with a corresponding change on the Y Axis.

Figure 1.36



Z Axis Scale Driver

Repeat the identical process for the Z Axis as that used for the Y Axis.

Clicking **Update Dependencies** recalculates random scale values and all Axis Scale in proportion. Translating the Cube in the 3D Viewport will do the same thing.

OK! Take a deep breath. You are not finished.

Note: As previously stated, if you take a break and save and close the program, then come back to it later on, you may find an error message in the Driver Editor, Driven Property Panel, stating, **ERROR: Python auto execution disabled**. There will also be a **Auto-run disabled Driver** notification in the Info Editor Header. If this occurs click on **Reload Trusted** in the Info Editor Header.

Note: You may also find **Invalid Python script ERROR** message in the Driven Property Panel. In this case open the Text Editor window which will have the Python script and click Run Script.

Material Property Drivers

Material properties can be randomized using Python Script just as you did with the Rotation and Scale properties for the X Axis Driver. For the Rotation you used a value range of $-\pi$ to π (2π radians in a circle, therefore $-\pi$ to $+\pi$) and for the Scale the range was 0.8 to 1.8 Blender units).

Remember: For the Cube's X Axis Scale randomizing was performed using Scripted Expression `randf(lo, hi)`. For the Y Axis Scale randomizing was dependent on the Average Value of the X Axis Scale.

When using the `randf(lo, hi)` Expression to generate random colors on the Object, when dependencies are updated, you use the numeric values in the RGB Color Scheme range where 1.000 = White and 0.000 = Black.

The values between these maximum and minimums produce the spectrum of visible color between white and black. To randomize Material within the spectrum all that is required is to use the expression `randf(0.000, 1.000)`.

Remember: The Blender File must have the Python Script Entered, Run and Registered. Use the File named **RandomPy.blend** previously created.

Figure 1.37



Figure 1.38

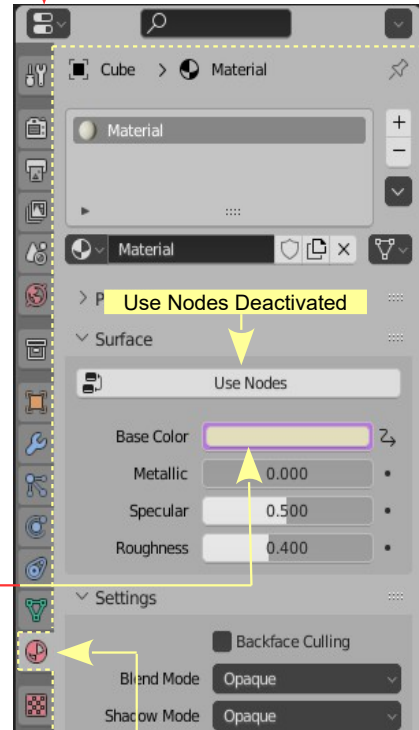
The first step in randomizing Materials (colors) is to Add Drivers. Previously Drivers have been Added in the Transform Properties Panel in the 3D Viewport Editor which displays when you press the N Key. This Panel does not include Material settings. To Add Drivers for Materials you right click on the **Base Color bar** in the **Properties Editor**, **Material Properties** and select **Add Drivers**.

Note: To demonstrate Randomizing Materials by Adding Drivers in the Properties Editor, Material Properties will be employed with Blender Nodes **deactivated**.

RMB Click, Add Drivers

The Drivers Editor will display Driver Channels for the RGB Materials and the A (Alpha or Transparency value).

Properties Editor



Material Properties

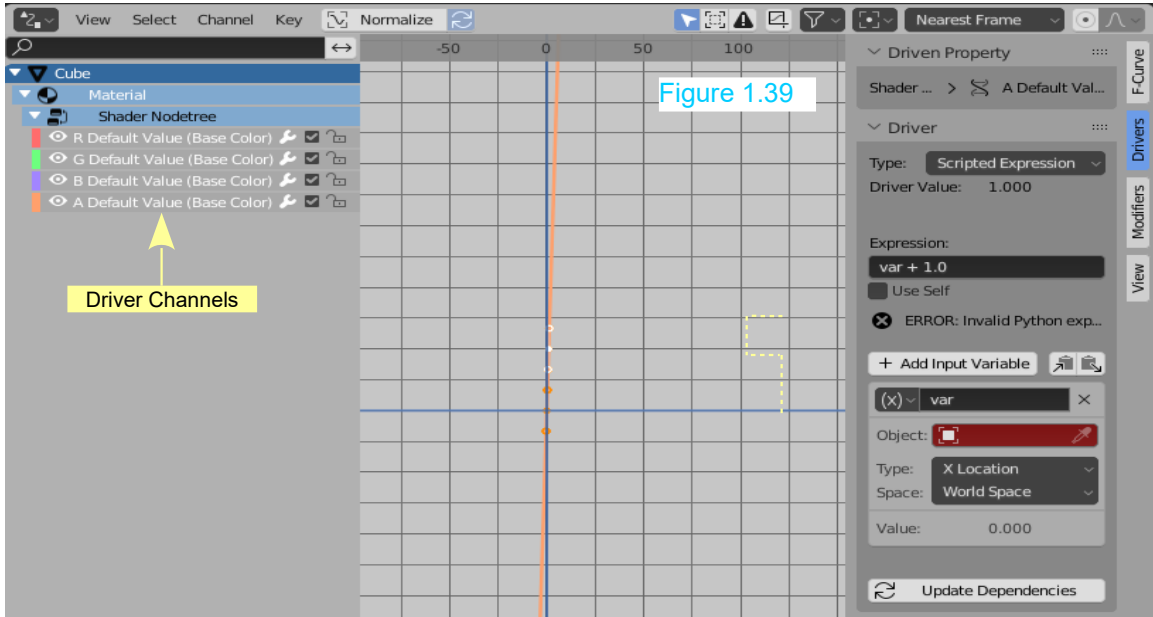


Figure 1.39

Select each **Driver Channel** in turn and in the corresponding **Driven Property Panel** enter the Expression: **ranf(0.000, 1.000)**

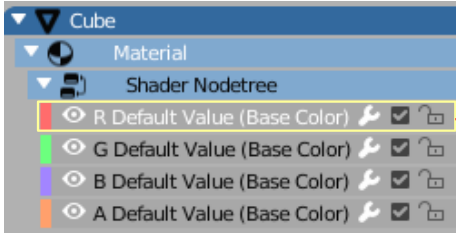
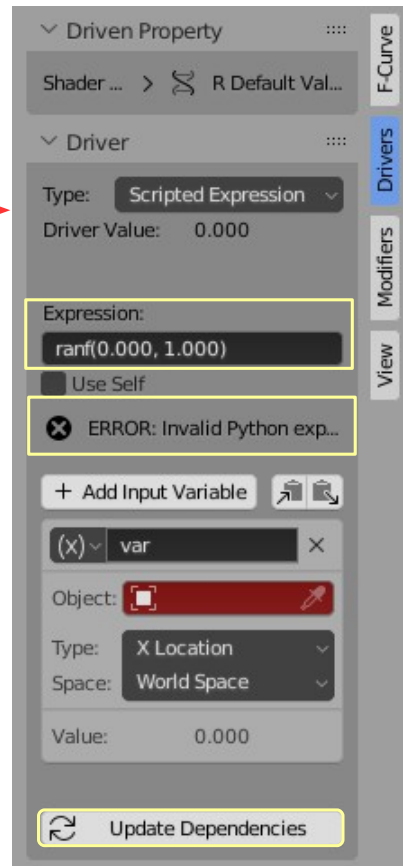


Figure 1.40

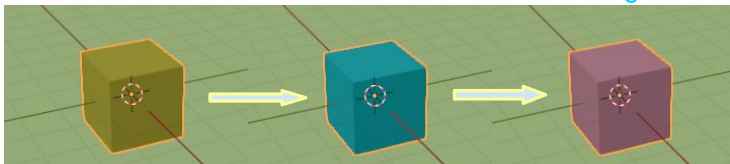


You may disregard the **A Alpha Channel** if you are not concerned with having the selected Object fade in the view.

When Enter is pressed to enter the script the Error Message changes to **Slow Python expression**

Each time Update dependencies is pressed the Material (color) changes randomly.

Figure 1.41



Duplicating the Object

In the 3D Viewport, by default, there is only have a single Cube Object in the Scene. Remember, at the beginning of the chapter, the objective was to create a bunch of Cubes scattered about the Scene.

It will be assumed that **Material Drivers** have Added as well as **Rotation Drivers** (see Figure 1.42)

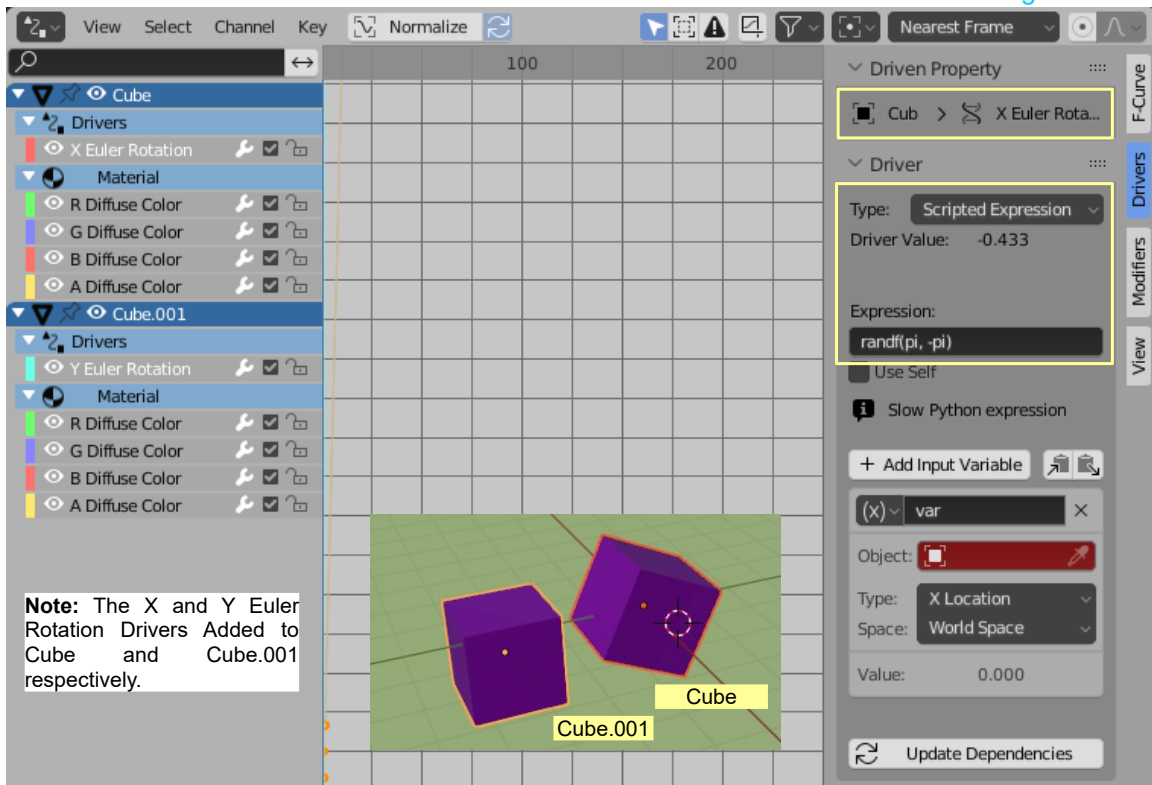
All you do is simply duplicate the Cube. With the Cube selected press **Shift + D key** to duplicate.

A duplicate is created and placed in Grab Mode ready to be Translated. Drag the mouse to relocate and observe that the original and the duplicate are both re-evaluated by the Drivers and the properties change.

There are now two Cube Objects in the Scene which are identical and appear to be separate. The duplicate Cube (Cube.001) is however linked to the original (Cube) in that it is sharing Properties Data-Blocks of the original. If you translate **Cube.001**, **Update Dependencies** is activated and the properties of the Cube change. If you translate **Cube** the properties of both Cubes are changed.

Shift select both Cubes and you will see two sets of Drivers in the **Graph Editor window** (Figure 1.42).

Figure 1.42

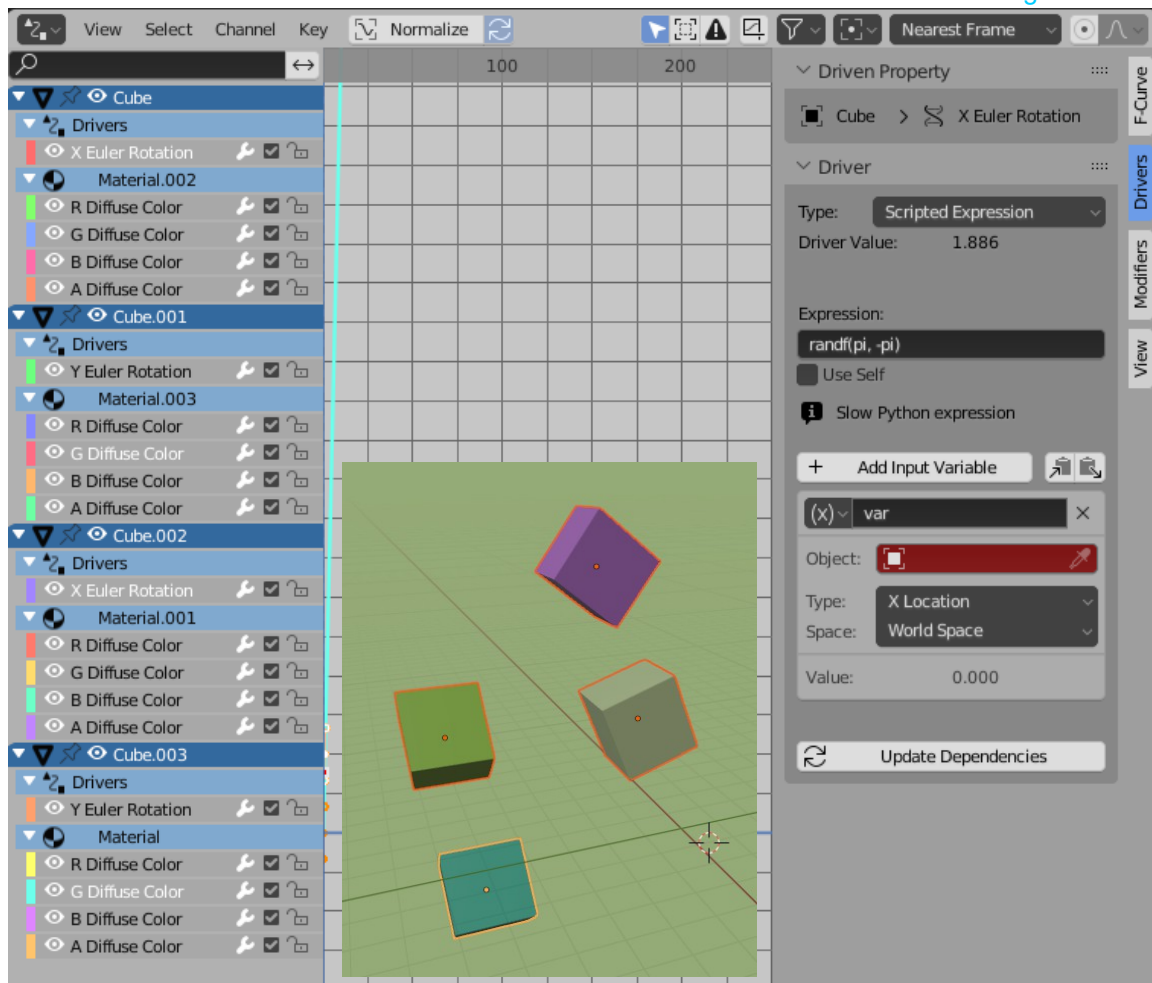


Note: The Euler Rotation Drivers have been Added separately to Cube and Cube.001 following the Cube being duplicated. If the X Euler Rotation Driver were Added to Cube, before duplication, then both Cube and Cube.001 would have an X Euler Rotation Driver.

Anomaly: The above being the case, selecting either Cube and pressing G Key (Grab Mode) and moving the Mouse sees the selected Cube, Translated and randomly Rotated but the Material remains the same. Shift selecting both Cube and Cube.001 and Translating sees both Cubes randomly Rotated with the Material unchanged. Clicking Update Dependencies in any Driver, Driven Property Panel sees both Cubes randomly Rotate with the Material changing but the Material on both Cubes is identical.

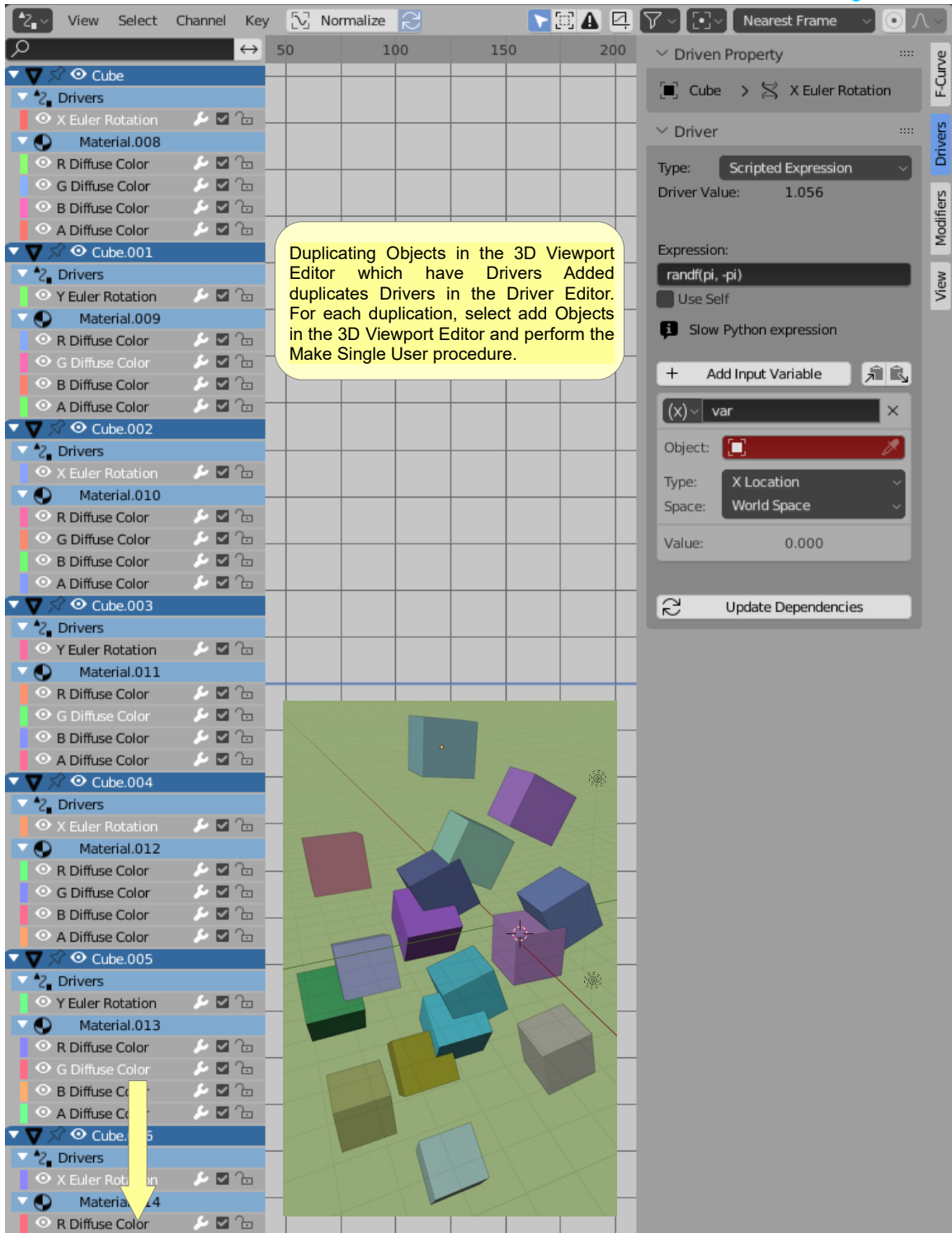
To have the Material and Rotation randomly change when Update Dependencies is pressed, select all Cubes in the 3D Viewport Editor then in the Header click **Object – Relations – Make Single User - Object & Data & Material**.

Figure 1.43



Select all Cubes and continue duplicating as many times as you like (Figure 1.44).

Figure 1.44



Animating the Properties

The simplest way to demonstrate the Animation of the Properties is to select all the Cubes in the 3D Viewport Editor (Press A Key), press the I Key and select Location, Rotation, Scale & Custom Properties. Do this at several Frames in the Timeline Editor. Play the Animation in the Timeline to see the randomly Rotate and change color.

In Conclusion

The forgoing description of **Randomizing Properties** is based on material in a Video Tutorial presented by **David Miller**. For an in depth understanding of this fantastic application in Blender the Tutorial is highly recommended.

Video: <https://vimeo.com/40389198>

David Writes: I found it easy to work with drivers on object and mesh properties. But I found it surprisingly difficult to get drivers to work on properties in material, texture, and node datablocks. This video will show you the secrets to coercing drivers to work anywhere:

- learn multiple ways to create a driver; when one method fails, another will work
- learn how to fix broken drivers after copying an object workflow tips for managing many drivers, objects, and datablocks

There is a multitude of Tutorials available on the Internet specific to the application of Drivers but bare in mind, many have been written and presented as the Blender program continues to develop. You will, therefore, encounter anomalies between the instruction provided and the current Blender release.

You are encouraged to pursue this topic and experiment with new and current information.