Manuel does Motion

Expressions Cheat Sheet

V3.0
03/04/25

Content:

A few **Basics**

Expressions are small pieces of code, based in the Java Script coding language.
Expressions are read from left to right, top to bottom.

Semicolon

A semicolon at the end of a line indicates the end of a statement.

```
var x = time*50;
```

(Number) Value

All properties in After Effects are defined by values. There are 1-, 2- and 3-dimensional values.

Opacity	**100%**		
Anchor Point	**960,0,540,0**		
Position	**960,0,540,0,0,0**		
Index Number	**[0]**	**[1]**	**[2]**
Dimension	**x**	**y**	**z**

You can specify which value to use by refering to their index numbers.

position[0]

Refers to the x-value of the position.

A few **Basics**

Variable

A variable is a container for a value. The value can change. They help to keep your expressions organized and easier to understand. Adding “var” in front of a varibale makes the code easier to read. Use a letter or name that makes sense.

```
var x = time*50;
var newheight = random(65);
```

String Value

A string value is a text value, written between single or double quotation marks.

```
“this is a text”
```

Boolean Value

A boolean value is a logical value. True or false.

```
seedRandom(19, timeless = false);
```

Array (Value)

An array is a group of values or variables or even arrays between two square brackets.

```
[960,540,0]
[x,width]
[960,[x,width]]
```

Basic Math

You can manipulate values with basic math:

- **+** **Addition**
- **-** **Subtraction**
- ***** **Multiplication**
- **/** **Division**
- ***-1** **Perform opposite of origonal**

transform.position+50

transform.scale-time*30

45+time*50

wiggle(1,30)/2

time*-1

Expression Controls

Using expression controls makes it easier to update your expressions. Instead of updating them inside the expression editor, you can link them to an Expression Control in the Effect Controls panel.

- **3D Point COntrol**
- **Angle Control**
- **Checkbox Control**
- **Color Control**
- **Dropdown Menu Control**
- **Layer Control**
- **Point Control**
- **Slider Control**

It is useful to apply them to a Null Object (for some unknown reason I use Adjustment Layers) which you use as a control layer.
You can then link expression controls to properties of multiple layers. It will probably look like this:

thisComp.layer(“CONTROL”).effect(“Slider Control”)(“Slider”)

thisComp.layer(“CONTROL”).effect(“Slider Control”)(“Slider”)

We access the currently open composition.

thisComp**.layer(“CONTROL”)**.effect(“Slider Control”)(“Slider”)

In this comp, we access the layer named “CONTROL”

thisComp.layer(“CONTROL”)**.effect(“Slider Control”)**(“Slider”)

In this layer, we access the effect properties, the effect “Slider Control”.

thisComp.layer(“CONTROL”).effect(“Slider Control”)**(“Slider”)**

In the Slider Control effect, we select the “Slider” value.

Round Numbers

Values in After Effects come with lots of decimals. In some cases you need whole numbers, or reduce the number of decimals.

```
var t = time*50;
[t]
```

t = 14,57931567341

Math.round()

“Math.round” is the most common option.

```
var t = time*50;
Math.round(t)
```

t = 15

Math.floor()

Always rounds the numbers down.

```
var t = time*50;
Math.floor(t)
```

t = 14

Math.ceil()

Always rounds the numbers up.

```
var t = time*50;
Math.ceil(t)
```

t = 15

Round Numbers

toFixed()

The number inside the parentheses specifies the number of decimals you want to keep.

```
var t = time*50;
t.toFixed(2)
```

t = 14,57

```
var t = time*50;
t.toFixed(4)
```

t = 14,5793

The **“time”** Expression

The time expression returns a composition’s current time in seconds. Meaning it’s value increases by 1 per second.

time

If you add the time expression to a property you can create linear and continouus movements.

```
time
```

To speed the movement up, you need to multiply it.

```
time*50
```

By default, the start value is 0. If you want to start at a different value you need to add it.

```
45+time*50
```

The time expression can only be added to a 1-dimensional value, like the rotation value.

```
var x = time*10;
```

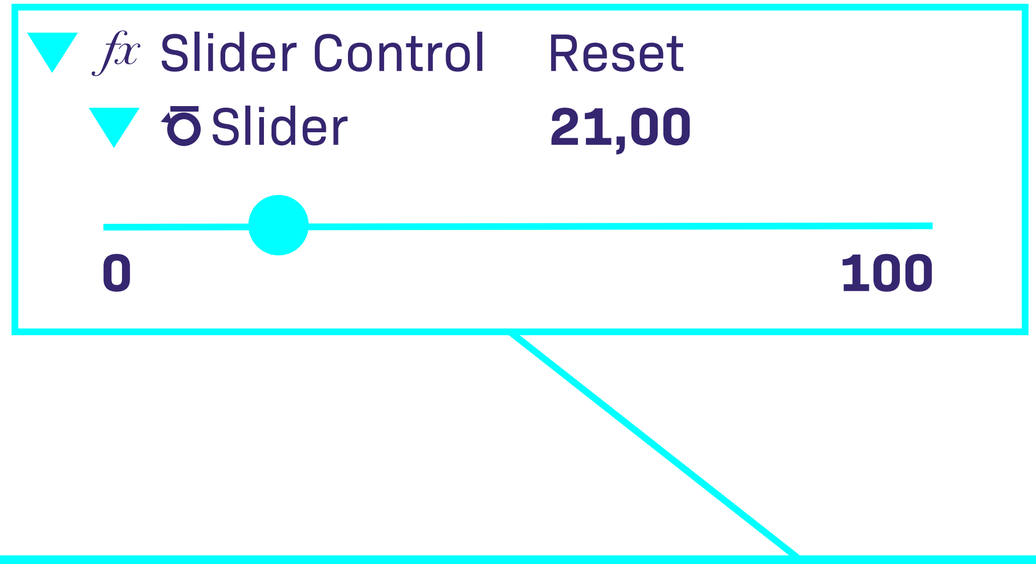
If you want to add it to a 2-dimensional value like the scale value, you need to define a variable for each dimension.

```
var y = 30+time*25;
[x,y]
```

Linking values to **Expression Controls**

time***50**

Instead of adding numbers, you can link values to slider controls to manipulate them and animate them.



time***thisComp.layer(“CONTROL”).effect(“Slider Control”)(“Slider”)**

The “time” expression is multipiled with whatever value the slider control indicates. If you animate the slider value, it changes over time.

The **“wiggle”** Expression

wiggle()

The “wiggle” expression creates random movement.

wiggle(frequency, amplitude)

You need to add two values to make it work:

frequency: How often you want your value to change per second

amplitude: How much your value is allowed to change above and below the starting value.

You can add it to any property like the rotation, opacity, scale or position.

If you want to apply different behaviours for different dimensions, you need to define a variable for each dimension.

Or you might want to wiggle only 1 dimension.
Link the dimension you don’t want to wiggle to the corresponding value of the property.

```
wiggle(2,50)
```

The value changes **2** times per second by **50** in each direction.

The wiggle expression generates a two-dimensional value for the x and y variables.

```
var x = wiggle(2,50);
var y = wiggle(1,300);
[x[0],y[1]]
```

You can refer to only one dimension of each value.

```
var x = wiggle(2,50);
var y = transform.position[1];
[x[0],y]
```

The **“wiggle”** Expression

The “wiggle” expression is actually much more complicated though.

wiggle(frequency, amplitude, octaves=1, amplitude_multiplier=0.5, t=time**)**

Most of these parameters are left out by default because we usually don’t change them.

octaves: Defines how detailed the wiggle is. Low octaves will get you a smoother result, higher octave values will make the wiggle more detailed and jittery. “=1” means it is set to 1 by default.

amplitude_multiplier: Defines by which value each additional octave’s amplitude should be multiplied by. “=0.5” means the default value is 0.5. If you had 3 octaves for example, each of these octaves’ amplitude will get smaller.
If the multiplier is bigger than one, each additional octave will be much more pronounced and add a strong influence onto the wiggle curve.

time: Returns the current composition’s time in seconds as a value. “=time” the actual time is set as default. If you change this value, you could slow down the wiggle expression for example.

Why do you need to know this?

To understand how to loop a wiggle expression!

The **“wiggle”** Loop

A “wiggle” loop is based on two expressions:

wiggle()

linear ()

Check out page 29 to learn more about the “linear” expression.

This is the expression that loops a “wiggle” animation:

```
var freq = 0.5;
var amp = 500;
var looptime = 5;
var t = time % looptime;
var wiggle1 = wiggle(freq, amp, 1, 0.5, t);
var wiggle2 = wiggle(freq, amp, 1, 0.5, t - looptime);
linear(t, 0, looptime, wiggle1, wiggle2)
```

var freq = 0.5;

Sets the frequency to 0.5 times per second.

var amp = 500;

Sets the amplitude to 500.

var looptime = 5;

Defines the length of your loop: 5 seconds.

var t = time % looptime;

Defines how far we are into the current loop cycle. We get a time value that resets to 0 once it reaches 5 seconds.
The “%” sign is called the “remainder operator” in Java Script. It obtains the remainder between two numbers.

var wiggle1 = wiggle(freq, amp, 1, 0.5, t);

The “wiggle” expression. The “octave” and “amplitude multiplier” parameters are set to its default values. The time parameter is manipulated. “t” equals the remaining looptime.

The **“wiggle”** Loop

var wiggle2 = wiggle(freq, amp, 1, 0.5, t - looptime);

The same “wiggle” expression. Except for the time parameter. The time is inverted. It runs from -5 to 0. The wiggle animation runs backwards.

linear(t, 0, looptime, wiggle1, wiggle2)

The “linear” expression blends the two wiggles. As “t” goes from 0 to 5, the output starts at the new minimum output value of “wiggle1” and ends at the new maximum output value of “wiggle 2”.

The **“posterizeTime”** Expression

posterizeTime()

The “posterizeTime” expression reduces the frame rate of a property by returning a value every X number of frames.

posterizeTime(12)

The frame rate is reduced to 12 frames per second.

posterizeTime(fps)

In contrast to the “PosterizeTime” effect which applies to a whole layer, you can reduce the frame rate of a single property with the “posterizeTime” expression.

You need to kow: The expression needs a value it refers to. And it can’t return a value directly, so you need to add **.value** at the end.

posterizeTime(12);

transform.scale.value

The “posterizeTime” expression in combination with the “wiggle” expression is great to create a scribbled style.

posterizeTime(8);

wiggle(2,100)

The **“random”** Expression

The “random” expression generates random values for the property it’s applied to.

random()

The number in parentheses is the maximum value.

random(max value)

0 is the minimum value by default. If you want to add a different min value, add it inside the parentheses.

random(min value,max value)

You can add the random expression to a 2-dimensional property like the scale property.

```
random(25)
```

After Effects returns random values between 0 and 25.
The value changes every frame.

```
random(5,75)
```

After Effects returns random values between 5 and 75.
The value changes every frame.

```
var s = random(5,75);
[s,s]
```

Or define a variable for each dimension:

```
var x = random(5,75);
var y = random(15,30);
[x,y]
```

The **“random”** Expression

Instead of a value, you can add arrays to define a range of values.

random(max array**)**

```
random([350,700])
```

x y

The expression picks values within a defined area that goes from 0 to 350 pixels horizontally and from 0 to 700 pixels vertically.

random(min array,max array**)**

```
random([10,50], [350,700])
```

x y x y

The expression picks values within a defined area that goes from 10 to 350 pixels horizontally and from 50 to 700 pixels vertically.

To prevent the “random” expression from returnong a value every frame, you can combine it with the “posterizeTime” expression.

```
posterizeTime(4);
random(5,75)
```

The **“gaussRandom”** Expression

gaussRandom()

The “gaussRandom” expression is interchangeable with the “random” expression.

gaussRandom(max value)

The minimum value is 0 by default.

gaussRandom(min value,max value)

Instead of values, you can add one or two arrays to define a range of random values.

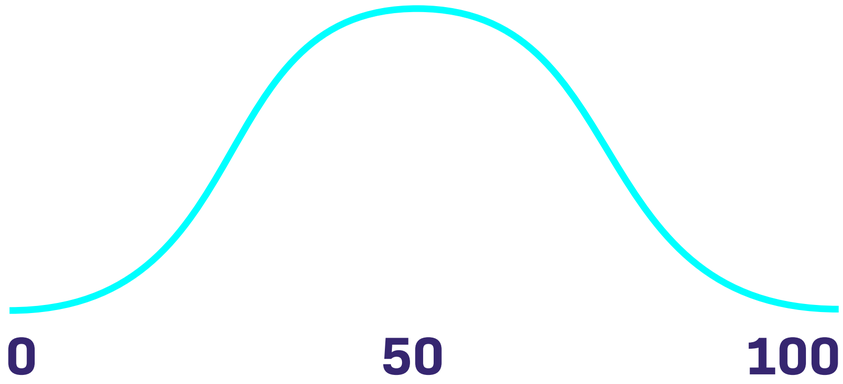
gaussRandom(max array)

gaussRandom(min array,max array)

random(100)

The “random” expression picks values between 0 and 100. All values have the exact same chance to be picked.

gaussRandom(100)



The “gaussRandom” expression picks values between 0 and 100. The middle values have a much higher chance to be picked than the values near the min and max value.
This gaussian distribution creates a more natural look.

The **“seedRandom”** Expression

The random values that are returned by the “random” expression follow a certain animation pattern.

```
random(100)
```

seedRandom()

The “seedRandom” expression makes sure, the animation pattern is the same, everytime you render it.

```
seedRandom(19)
```

Any “seed” value you add to the expression is linked to a unique animation pattern.
You can add the same “random” expression to position properties of different layers. If you use the “seedRandom” expression and add the same “seed” number, they will use the exact same animation pattern. By using different “seed” numbers, you make sure they behave differently.

```
seedRandom(19);
random(100)
```

Refering to the “index” number of a layer is an easy way to assign different “seed” numbers for each layer.

```
seedRandom(index);
random(100)
```

The **“seedRandom”** Expression

There is a second part, the timeless argument.

seedRandom(value, timeless = false)

“timeless = false” means, the value changes each frame.
The timelss argument is set to false by default.

```
seedRandom(19, timeless = false);
random(100)
```

To slow down the speed, add the posterizeTime expression.

```
seedRandom(19, timeless = false);
posterizeTime(12);
random(100)
```

seedRandom(value, timeless = true)

“timeless = true” means, it will choose a random value at the beginning. The value then stays the same and behaves like a freeze frame.

```
seedRandom(19, timeless = true);
random(100)
```

The **“noise”** Expression

noise()

The noise() expression is similar to the wiggle() expression.
It’s not based on the seed value or the Index number,
but on Perlin noise.
Perlin noise produces values between -1 and 1

noise(time)

This expression produces values between -1 and 1.

noise(time)*1000

Multiply the values by 1000 for example to increase
the movement.

Add the “noise” expression to the Position property of a layer.
The Composition has a resolution of 1920x1080 pixels.

```
var x = 960+noise(time)*500;
var y = 540+noise(time)*500;
[x,y]
```

The layer moves randomly around the centre in a diagonal direction between the top left corner and the bottom right corner.

The **“key”** Expression

Keyframes are numbered in After Effects.
Starting with number 1.

key(4)

“key(4)” is the fourth keyframe in the timeline. If you add this expression to a property, it will jump to the keyframe in that property with the index number 4.

numkeys

“numkeys” is the max keyframe number, no matter how many keyframes you have set.

You can use these two expressions to randomly jump between keyframes:

```
var rankeys = random(1, numkeys);
```

Define a variable. It equals a random value between 1 and the max number of keyframes.

```
var rankeys = Math.round(random(1, numkeys));
```

To get whole numbers, add the “Math.round” expression.

```
key(rankeys)
```

The result is a different key number between 1 and the max key number. Each frame.

To slow it down, you can add the “posterizeTime” expression on top.
To assign a specific seed number, add the “seedRandom” expression before the “random” expression.

The **“key”** Expression

With the “key” expression you can access the time, index and value properties of a keyframe.

key.time

key.index

key.value

The Position of “Shape Layer 1” is animated with 2 Positon keyframes.

“Shape Layer 2” has the Position of the second keyframe.

The **“loop”** Expression

The “loop” expression loops a series of keyframes.

loopIn()

There are two different versions.

loopOut()

You’ll mostly use the “loopOut” expression.

loopIn()

Loops before the first keyframe.

loopOut()

Loops beyond the last keyframe.

loopOut(“type”,modifier)

The loop expression has three diffferent parameters.

loopIn(“cycle”,1)

loopOut(“pingpong”,1)

Loop Types:

There are 4 different types of loops. Each type has a different way of how your keyframes are looped. The type is added inside the parentheses between quotation marks.

loopOut(“cycle”)

The “cycle” loop repeats your keyframes over and over again. After the last keyframe it jumps back to the beginning.

All loop expressions are set to “cycle” by default.

loopIn(“cycle”)

loopOut(“cycle”)

loopOut(“pingpong”)

The “pingpong” loop plays your keyframes back and forth. After the last keyframe it plays your keyframes backwards, then starts again from the beginning.

loopIn(“pingpong”)

loopOut(“pingpong”)

loopOut(“continue”)

The “continue” loop takes the value and speed of the last key-frame and continues it. If it is a position keyframe, the loop will pick up the direction and the speed of the movement at the time of the last keyframe and continues it.

loopIn(“continue”)

loopOut(“continue”)

loopOut(“offset”)

If your loop ends on a different position that it starts, you can use the “offset” loop. It takes the last position value as new start value and adds or subtracts the difference to all the other values.

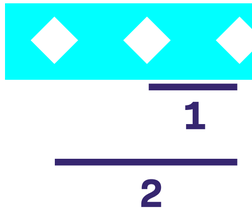
loopIn(“offset”)

loopOut(“offset”)

Loop Modifiers:

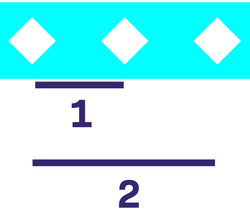
With loop modifiers, you can specify which part of your loop you want to repeat. The numbers refer to the parts beetween keyframes.

loopOut(“pinpong”,1)



1: Loops between the last two keyframes.
2: Loops between the last three keyframes.

loopIn(“pinpong”,1)



1: Loops between the first two keyframes.
2: Loops between the first three keyframes.

The **“loop”** Expression

One more thing!
There are two more variants of the loop expression:

loopOutDuration()

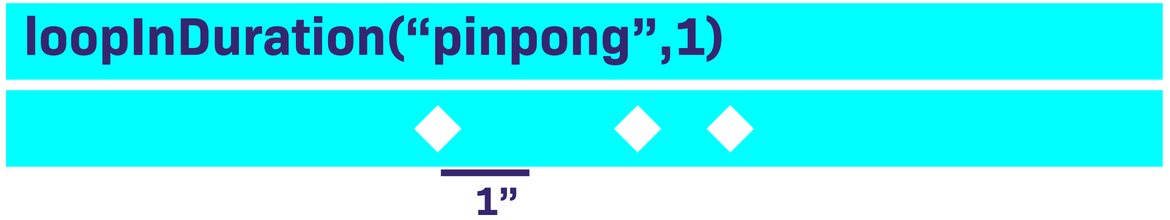
loopInDuration()

The loopDuration expressions loop based on time in seconds.

loopOutDuration(“pinpong”,1)

1: Loops the last second of the loop.

loopInDuration(“pinpong”,1)



1: Loops the first second of the loop.

The **“linear”** Expression

The linear expression relates two different strings of values to each other. It is great to relate properties that don’t use the same value scale.

linear()

linear(variable, varmin, varmax, outmin, outmax**)**

The expression needs a variable, which is the source. Like a Slider Control for example.

linear(variable, **varmin**, **varmax**, outmin, outmax**)**

Then you need to add the variable’s minimum and maximum values.

linear(variable, varmin, varmax, **outmin**, **outmax)**

Lastly you need to add the two new minimum and maximum outcome values.

```
var x = thisComp.layer(“controls”).effect(“Slider Control”)(“Slider”);
```

First, define a variable. It refers to a Slider Control in the effect property of a “controls” layer.

```
linear(x, 0, 100, 50, 360)
```

Then add the variable to the “linear” expression.
The range of the Slider Control is from 0 to 100. The new outcome is from 50 to 360. So when the Slider Control indicates 0, the property value is 50. 100 is 360.

The **“linear”** Expression

With the “linear” expression you can control different properties with 1 Slider Control.

Add an Adjustment Layer or Null Object to your comp and add a Slider Control to it.
Name the layer “controls”.

Add two Shape Layers.

Add an expression to the Rotation property of one shape layer:

```
var r = thisComp.layer(“controls”).effect(“Slider Control”)(“Slider”);
```

First, define a variable. It refers to a Slider Control in the effect property of the “control” layer.

```
linear(r, 0, 100, 0, 360)
```

Then add the variable and the two strings of values to the “linear” expression.

Add an expression to the X-Position property (Seperate the dimensions) of the other shape layer:

```
var x = thisComp.layer(“controls”).effect(“Slider Control”)(“Slider”);
```

First, define a variable. It refers to a Slider Control in the effect property of the “control” layer.

```
linear(x, 0, 100, 750, 1200)
```

Then add the variable and the two strings of values to the “linear” expression.
You now control both properties with one Slider Control.

The **“ease”** Expression

The “ease” expression is interchangeable with the “linear expression.

Add an expression to the X-Position property (Seperate the dimensions) of the other shape layer:

```
var x = thisComp.layer(“controls”).effect(“Slider Control”)(“Slider”);
```

First, define a variable. It refers to a Slider Control in the effect property of the “controls” layer.

```
ease(x, 0, 100, 750, 1200)
```

Instead of a linear interpolation, it adds a standard “Ease In” and “Ease Out” at the beginning and end of the value range.
It behaves like the “Easy Ease” keyframe function.

```
easeIn(x, 0, 100, 750, 1200)
```

The “easeIn” expression adds a “Ease In” at the beginning then switches to linear at the end.

```
easeOut(x, 0, 100, 750, 1200)
```

The “easeOut” expression starts with a linear interpolation and adds a “Ease Out” at the end.

The **“clamp”** Expression

clamp()

The “clamp” expression tells properties to not go below and above a certain minimum and maximum value.

clamp(input, minimum, maximum**)**

time*360

If you add this expression to a rotation property of a layer, it rotates 360 degrees in 1 second.

var r = time*360;

clamp(r, 100,250)

The rotation starts at 100° and ends at 250°.

The **“valueAtTime”** Expression

The “valueAtTime” expression returns the value of another layer at a specific time. It is great to offset animations.

valueAtTime()

Pick-whip to the value of another layer, like the rotation property for example.

thisComp.layer(“White Solid 1”).transform.rotation

Take the rotation value from the layer called “White Solid 1” in this composition. The rotation animation is exactly the same.

thisComp.layer(“White Solid 1”).transform.rotation**.valueAtTime(3)**

Look at the rotation value of the layer at 3 seconds in the timeline. Let’s say it is 30 degrees.
It takes that value und sticks to it like a freeze frame.

thisComp.layer(“White Solid 1”).transform.rotation**.valueAtTime(time-3)**

The animation of the rotation property is delayed by 3 seconds. “time” is the actual time minus three seconds.

thisComp.layer(“White Solid 1”).transform.rotation**.valueAtTime(time+3)**

The animation of the rotation property is 3 seconds early. “time” is the actual time plus 3 seconds.

The **“sampleImage”** Expression

With the “sampleImage” expression, you can access the colour data of another layer.
Add this expression to any property with a colour value, like a fill- or stroke colour property.

sampleImage()

It takes 4 parameters:

```
var target = thisComp.layer(“White Solid 1”);
target.sampleImage(transform.position, [0.5, 0.5], true, time)
```

```
var target = thisComp.layer(“White Solid 1”)
```

The variable “target” defines the target layer. In this case: “White Solid 1” in this composition.

```
target.sampleImage(transform.position, [0.5, 0.5], true, time)
```

Sample the colour and Alpha data from layer “White Solid 1”.

```
target.sampleImage(transform.position, [0.5, 0.5], true, time)
```

Location from which to sample the data. In this case: The layer position inside the “transform” properties.

```
target.sampleImage(transform.position, [0.5, 0.5], true, time)
```

Horizontal und vertical spread of the sample area from the centre point. In this case 1 pixel.

The **“sampleImage”** Expression

target.sampleImage(transform.position, [0.5, 0.5], **true**, time)

Sample the area after Effects and Maskes are applied. If you add a colour gradient for example,
the colour values of the gradient are sampled. This parameter is optional.

target.sampleImage(transform.position, [0.5, 0.5], true, **time**)

Specifes the time of the sampling. “time” means at the current time. This parameter is optional.

The property uses the same colour values as the target layer in the sample area.
If you want to move the sample area, animate the position property of the layer that accesses the colur data.

The **“rgbToHsl”** Expression

The colour values that the sampleImage expression delivers are RGB values. 4-dimensional array values from 0 to 255.

[220,193,149,255]

The expression syntax only accepts values between 0 and 1.
So you need to convert the colour from RGB to HSL.
HSL values range between 0.0 and 1.0

rgbToHsl()

You can refer to a specific value in a colour array by using its index.

RGB:

R	**220**	Index: **[0]**
G	**193**	Index: **[1]**
B	**149**	Index: **[2]**
A	**255**	Index: **[3]**

HSL:

H	**253°**	Index: **[0]**
S	**66%**	Index: **[1]**
B/L	**44%**	Index: **[2]**

The **“rgbToHsl”** Expression

You can use the lightness or brightness channel of another layer to manipulate a property.
Add a solid (“White Solid 1”) and add a gradient ramp to it. Turn it into a radial gradient.
This is the layer you extract the colour data from.

Add a second shape. This is the shape you want to animate, using the colour data.
Add this expression to the property you want to animate.

Let’s say you want the scale of the shape to react to the brightness of “White Solid 1”.
Add an expression to the scale property:

```
var target = thisComp.layer(“White Solid 1”);
```

Define the target layer: “White Solid 1”.

```
var samplergb = target.sampleImage(transform.position, [0.5, 0.5], true, time);
```

Define a second variable “samplergb”: The result is a RGB colour value.

```
var samplehsl = rgbToHsl(samplergb);
```

Define a third variable “samplehsl”: The result is a HSL colour value. The sampled RGB value is converted into an HSL value.

The **“rgbToHsl”** Expression

```
var s = linear(samplehsl[2],0,1,50,200);
```

```
[s,s]
```

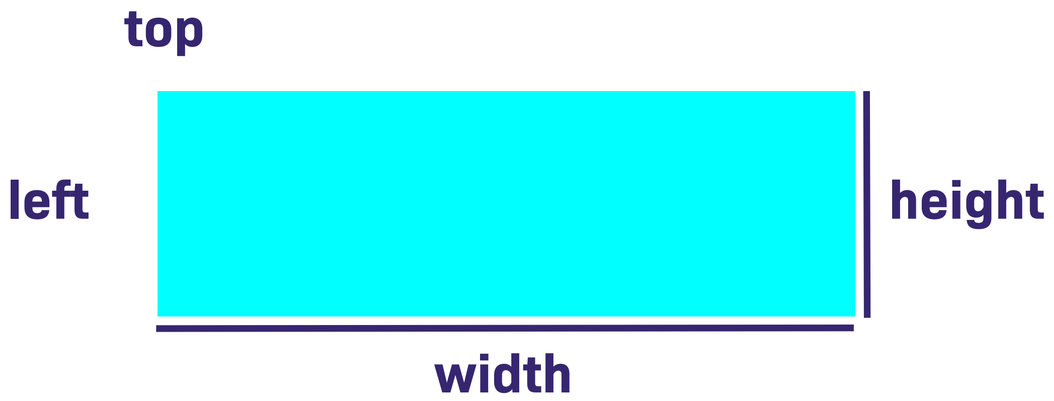
Define a fourth variable “s” like scale. The linear expression has the HSL colour value as input.
The lightness / brightness channel. Index 2. The minimum and maximum values of the HSL colour value are 0 and 1.
Whenever the brightness is 0, the scale of the shape is 50%.
When the brightness is 1, the scale of the shape is 200%.

If you animate the colour gradient, the shape resonds to the brightness of it and changes its scale.

This is how you can create a responsive dock animation or responsive grids.

The **“sourceRectAtTime”** Expression

The sourceRectAtTime() expression can return the dimensions of shape and text layers.



sourceRectAtTime()

4 attributes to be exact:
The top and left plus the width and height.

sourceRectAtTime(2)

By default, it looks at the current time.
You can change the time, by adding numbers. In seconds.

sourceRectAtTime(inPoint)

sourceRectAtTime(outPoint)

You can also point to the in point or out point of a layer.

The **“sourceRectAtTime”** Expression

Let’s say, you created a text layer “text layer” and a shape layer “shape layer”.
You want the shape layer to have the same dimensions as the text layer.

Step 1: Add an expression to the size property of the rectangle path of the shape layer.

Step 2: Establish a variable, which refers to the dimensions of the text layer.

```
var s = thisComp.layer(“text layer”).sourceRectAtTime();
```

“s” equals the dimensions of the “text layer” at the current time.

Step 3: Establish a variable for each dimension. Add an array.

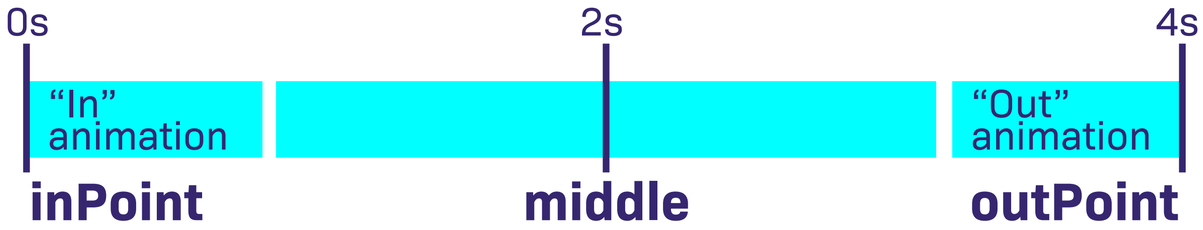
```
var s = thisComp.layer(“text layer”).sourceRectAtTime();
var w = s.width;
var h = s.height;
[w,h]
```

“w” equals the width of the “text layer” at the current time.
“h” equals the height of the “text layer” at the current time.
The array defines the new size of the “shape layer”. The exact same size as the “text layer”.

“inPoint”, “outPoint”

The dimensions of the shape layer are the same at the current time. This means if you animate the text, the size of the text layer changes and jumps.

To avoid that, it is best to look at a later time, when the text is fully animated. Like the midpoint of the layer.
When your composition and layers are 4 seconds long, this would mean: 2 seconds.



Define the midpoint:
We can refer to the inpoint and outpoint of a layer. That’s enough to define the midpoint.

```
var midpoint = (thisLayer.outPoint - thisLayer.inPoint)/2;
var s = thisComp.layer(“text layer”).sourceRectAtTime(midpoint);
var w = s.width;
var h = s.height;
[w,h]
```

“midpoint” equals the middle of the layer. 4 seconds (outpoint) - 0 seconds (inpoint) /2 = 2 seconds
The “sourceRectAtTime” expression now looks at the dimensions of the text layer at 2 seconds.
This allows you to animate the shape layer independently.

Pinning Anchor Points

To make sure, the two shapes match, you need to pin the 2 anchor points to the exact same position.

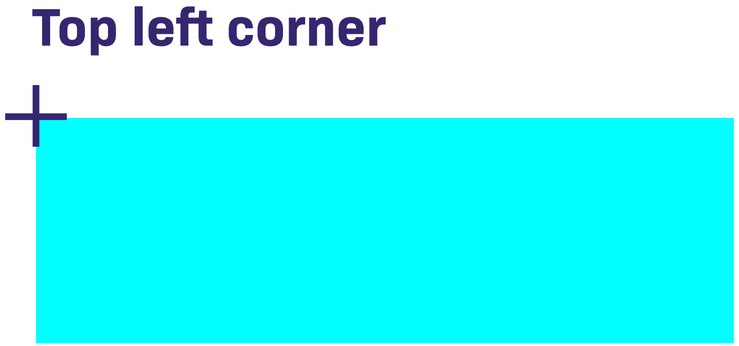
This is possible with the 4 attributes, the “sourceRectAtTime” expression refers to:

top, left, width, height

Add these expressions to the anchor point properties:

```
var a = sourceRectAtTime();
var x = a.left;
var y = a.top;
[x,y]
```

Make sure, both shapes have the same position.
Link the two position properties.



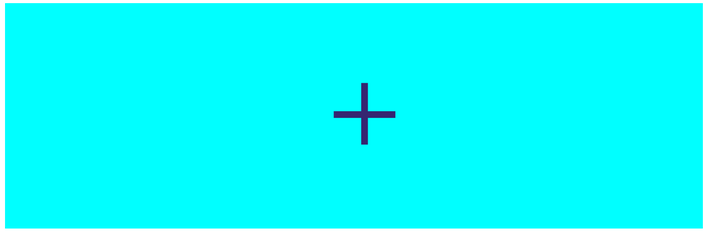
Pinning Anchor Points

```
var a = sourceRectAtTime();
var x = a.left;
var y = a.top + a.height/2;
[x,y]
```

Centre left edge

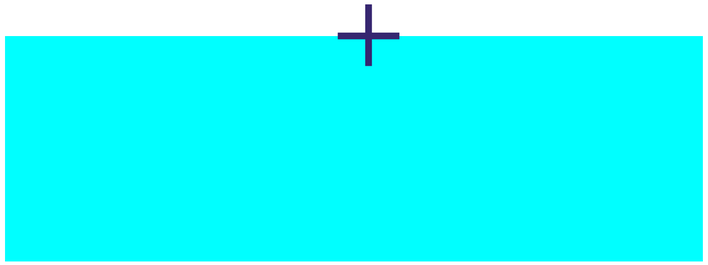
```
var a = sourceRectAtTime();
var x = a.left + a.width/2;
var y = a.top + a.height/2;
[x,y]
```

Shape centre



```
var a = sourceRectAtTime();
var x = a.left + a.width/2;
var y = a.top;
[x,y]
```

Top centre edge



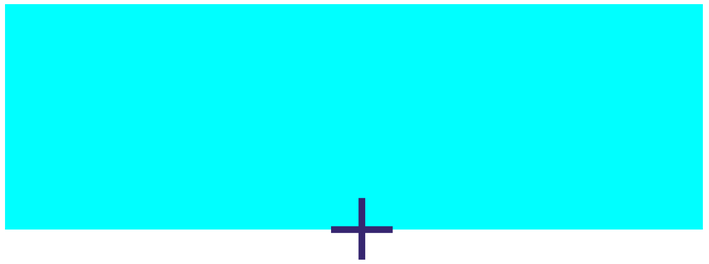
Pinning Anchor Points

```
var a = sourceRectAtTime();
var x = a.left + a.width;
var y = a.top;
[x,y]
```

Top right corner

```
var a = sourceRectAtTime();
var x = a.left + a.width/2;
var y = a.top + a.height;
[x,y]
```

Bottom edge centre



```
var a = sourceRectAtTime();
var x = a.left + a.width;
var y = a.top + a.height;
[x,y]
```

Bottom right corner

The **“index”** Expression

The “index” expression refers to the index number of the current layer. It starts with 1 and goes up with the number of layers you have.

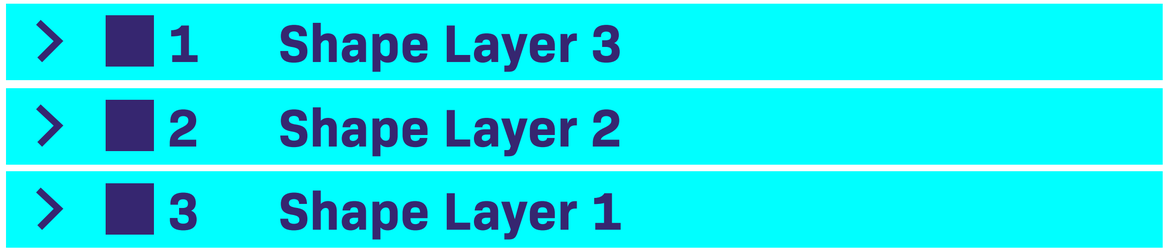
The layer has the index number 1.

index

To refer to another layer, you can add (“index + n”) or subtract (“index - n”) numbers.

(index-1)

“(index-1)” refers to the layer above.

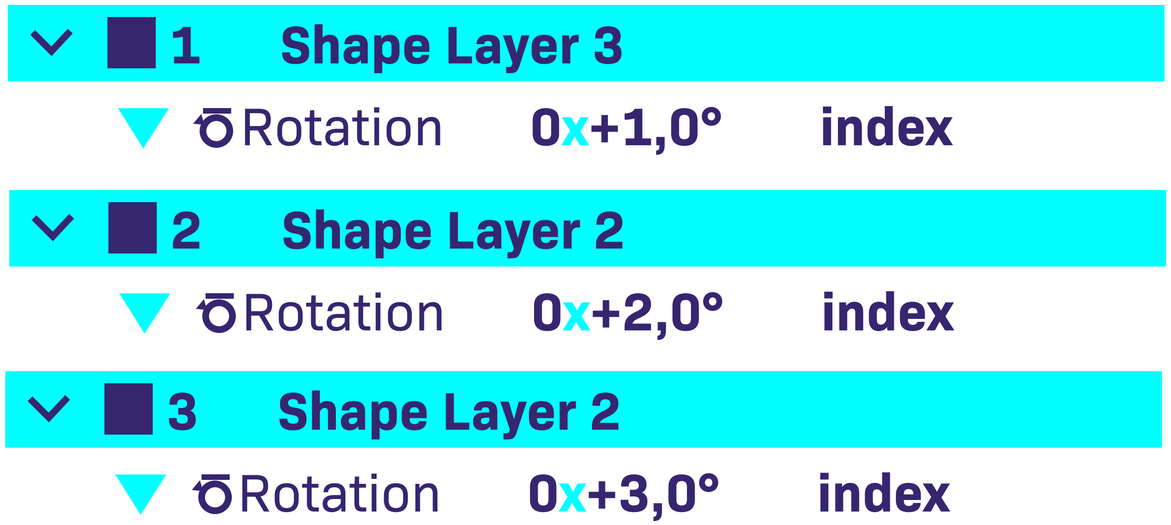


(index+1)

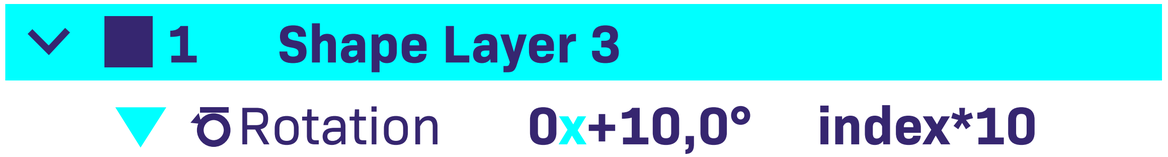
“(index+1)” refers to the layer beneath.

The **“index”** Expression

Refering to the “index” number of a layer saves a lot of time when used to duplicate and offset layers.
Let’s add it to the rotation property of a layer:



The Rotation value is the index number of the layer.



To increase the value, multiply it.
Each time you duplicate the layer , the value increases by 10 degrees.

The **“index”** Expression

You can also add it to two-dimensional properties:

1 Shape Layer 3

Scale **10,00%**

```
var s = index*10;
[s,s]
```

2 Shape Layer 2

Scale **20,00%**

```
var s = index*10;
[s,s]
```

3 Shape Layer 1

Scale **30,00%**

```
var s = index*10;
[s,s]
```

Each time you duplicate the layer , the scale value increases by 10 percent.

The **“index”** Expression

With the “index” expression, you can easily offset animations.

The scale property of “Shape Layer 3” is animated with two keyframes.

“Shape Layer 2” gets the Scale value from the layer above (index-1) and delays the animation by 1 second.

“Shape Layer 3” gets the Scale value from the layer above (index-1) and delays the animation by 1 second.

Each time you duplicate “Shape Layer 1” or “Shape Layer 2”, the animation is reapeted and delayed by 1 second.

The **“if / else”** Conditional Statement

The “if / else” conditional statement is a line of code that will change its output value based on the input value.

```
if(condition){
```

“if” statement. Add the condition in parentheses.
The line ends with open curly brackets.

```
        true value1;
```

If the condition is true, value 1 happens. This value can be a value, an array or a line of code.
This line is indented and ends with a semicolon.

```
}else{
```

“else” statement.

```
        false value2;
```

If the condition is false, value 2 happens. This value can be a value, an array or a line of code.
This line is indented and ends with a semicolon.

```
}
```

The statement ends with a close curly bracket and a semicolon.

Conditions:

a < b	**a is less than b**
a > b	**a is greater than b**
a <= b	**a is less than or equal to b**
a >= b	**a is greater than or equal to b**
a == b	**a equals b**
a && b	**a and b**

The **“if / else”** Conditional Statement

Some examples:
You can link the opacity of a layer to it’s position.

Let’s say you animated the x-position property of a shape from 0 to 400. And you want the shape to disappear once it reached 200.
Add an expression to the opacity property of the layer.

```
if(transform.position[0]<200){
```

“if” statement. If the x-position value is smaller than 200.

```
        100;
```

The opacity value is 100.

```
}else{
```

“else” statement. If the value is 200 and bigger.

```
        0;
```

The opacity value is 0.

```
}
```

The statement ends with a close curly bracket and a semicolon.

The **“if / else”** Conditional Statement

Let’s say you want to animate a rotation property with the time expression and you want it to stop at 3 seconds.

```
var end = 3;
```

Define a variable for the end point of the animation. 3 seconds.

```
if(time > end){
```

“if” statement. If the time is bigger than the end point.
In other words: If the time indicator goes beyond 3 seconds.

```
        value;
```

The value of the rotation property is true.

```
}else{
```

“else” statement.

```
        time*50;
```

If the condition is false, the shape rotates with 50° per second.

```
}
```

The statement ends with a close curly bracket and a semicolon.

Let’s say you want to animate a position property with wiggle-expression. You want it to start at 1 second, stop at 3 seconds.

```
var start = 1;
```

Define a variable for the end point of the animation. 3 seconds.

```
var end = 3;
```

Define a variable for the end point of the animation. 3 seconds.

```
if((time > start) && (time < end)){
```

“if” statement. If the time is bigger than the start point and smaller than the end point

```
        wiggle(1,200);
```

Animate the property with a wiggle expression.

```
}else{
```

“else” statement.

```
        value;
```

If the condition is false, use the values of the postion properties.

```
}
```

Checkbox Control

You can use an “if / else” conditional statement and a Checkbox Control to switch layers on and off.

Add the layer you want to switch on and off. Add a Null Object, name it “controls” and add a Checkbox Control to it.

```
var t = thisComp.layer(“controls”).effect(“Checkbox Control”)(“Checkbox”);
```

Add an expression to the Opacity property of the layer and define a variable. Link to the checkbox of the “controls” layer.

```
if(t == 1){
```

“if” statement. If the variable t equals 1

```
    100;
```

the opacity is 100%. 1 means the checkbox is checked. 0 means the checkbox is unchecked.

```
}else{
```

“else” statement.

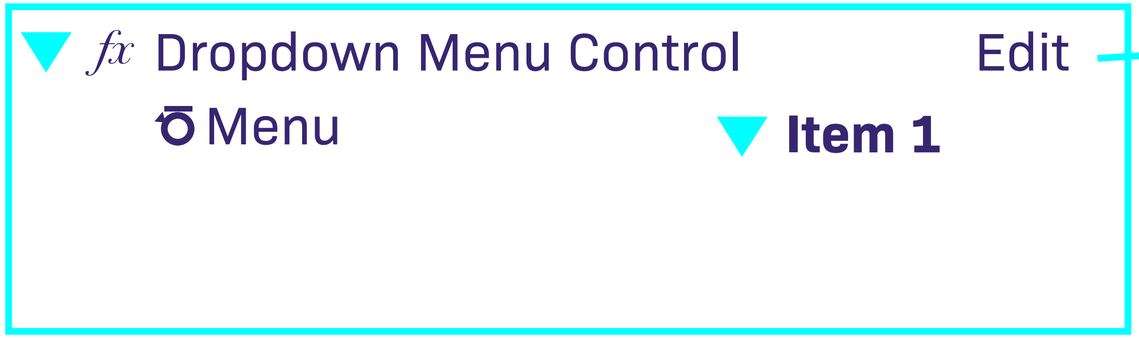
```
    0;
```

If the condition is false / the checkbox is unchecked, the opacity is 0.

```
}
```

Dropdown Menu Control

You can use a Dropdown Menu Control to create a list. With an “if / else” conditional statement you can create a rig out of it.



Edit the list here.
By default it contains “Item 1-3”.
Item 1 has the index number 1.
Item 2 has the index number 2.
Item 3 has the index number 3.
And so on...

Add a text layer. Add a Null Object, name it “controls” and add a Dropdown Menu Control to it.
Add an expression to the Source Text property of the text layer.

```
var m = thisComp.layer(“controls”).effect(“Dropdown Menu Control”)(“Menu”);
```

We define a variable. “m” like Dropdown Menu equals: Link to the Dropwdown Menu Control.
Then use an “if / else” conditional statement to assign text values to each item.

The **“if / else”** Conditional Statement

```
if(m == 1){
```

If the item with the index number 1 is selected:

```
        “Hello”;
```

The Source Text is “Hello”.

```
}else if(m == 2){
```

If the item with the index number 2 is selected:

```
        “Goodbye”;
```

The Source Text is “Goodbye”.

```
}else if(m == 3){
```

If the item with the index number 3 is selected:

```
        “What?”;
```

The Source Text is “What?”.

```
}
```

The **“length”** Expression

With the “length” expression you can get the length between to points.

Let’s say, we created two shape layers and want to get the distance between them:

```
var sphere = thisComp.layer(“sphere”).position;
```

```
var square = thisComp.layer(“square”).position;
```

Two variables. The first one equals the position of the sphere. The second one equals the position of the square.

```
length(sphere, square)
```

The length expression calculates the length between the two positions.

You can manipulate values depending on the length. Add an expression to the scale property of the square and add:

```
var sphere = thisComp.layer(“sphere”).position;
```

```
var d = length(position, sphere);
```

Two variables. The first one equals the position of the sphere. The second one equals the length between the square and the sphere.

```
linear(d,0,500,[500,500],[100,100])
```

The linear expression interpolates the distance.
If the distance (d) between the square is 0, the scale of the square will be 500%.
If the distance is 500 pixels or more, the scale will be 100%.
All values between 0 and 500 are calculated by the linear expression.

Are you missing expressions?

Let me know!

V1.0
07/03/23