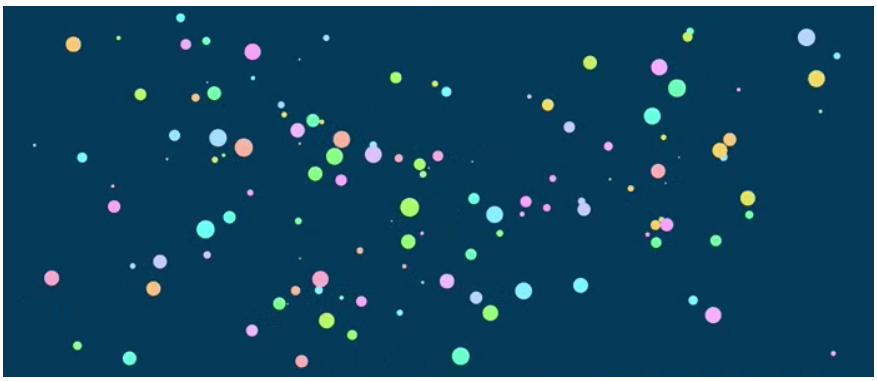
Tess Gadd Follow
UI Designer & a Student of DO.
Sep 9 · 9 min read

Framer Cheatsheet: The versatility of Utils

Utils (utilities) is the Swiss-Army knife of Framer. Learn how to wield it and you'll have graduated to an intermediate level.



`Utils` was designed to allow us mere mortals to do complicated shit. YAY! In this cheat sheet I am only going to go into detail about the most useful features. So apologies in advance for leaving things out. If you want to read more about the other features of `Utils` check out the Framer docs.

> *TLDR:* `Utils.` *allows you to do awesome shit.*

Like the other Cheat Sheets in this series, we will look at the very basics: simple properties and commonly used patterns. This was written for people like me—who aren't great at writing code, but are pretty darn good at copy and pasting. If you don't have Framer yet, download a free 2 week trial.

In this Framer Cheat sheet, we will look at the following:

1. What is Utils?
2. Utils.random[]
   a) color()
   b) image()
   c) number()
   d) choice()

3. Utils.cycle()
4. Utils.round()
5. Utils.interval(interval , handler)
6. Utils.delay
7. Utils.modulate()
8. Utils platform detection

. . .

1. What is Utils?

Utils or utilities are a set of functions in Framer that make our lives easier (yay! Lazy high-fives all around). So basically, as opposed to some long ass formula that I wouldn't understand, we can now use simple syntax to do complicated tasks.

What are these tasks, you ask? Well, see below.

. . .

2. Utils.random-

I love Utils.random.
Too lazy to search for images? `Utils.randomImage()` .
Too lazy to write a whole lot of random numbers?
`Utils.randomNumber()` .
Need to shuffle between random choices? `Utils.randomChoice()`

Basically, using the random part of Utils creates random choices, numbers, images, or colors. Pretty neat right?

This function is particularly powerful when used along with a loop so you can generate lots of random data.

a) Color

`Utils.randomColor()` allows you to randomize different rgb colours.
To use it, just do the following:

```
layer = new Layer
   backgroundColor: Utils.randomColor()
```

If you want to make your layer have an opacity, just add a value between 0–1, in the brackets. If you don't add anything in the brackets, it will default to opacity: 1.

```
layer = new Layer
   backgroundColor: Utils.randomColor(0.5)
```

Now let's try it in action. This is a simple example of how easy it is to use `Utils.randomColor()` .

```
for i in [0...6]
  layer = new Layer
     backgroundColor : Utils.randomColor()
     size: 100
     y: Align.center
     x: 110 * i + 25

  layer.onClick ->
     @backgroundColor = Utils.randomColor()
```

b) Image

I hate looking for images. But thanks to the lovely folks at Framer, we can import random images directly from unSplash—Holla! This is particularly useful if you want to make a newsfeed or contact list that needs random pictures.

To import a random image, use the following:

```
layer = new Layer
   image: Utils.randomImage()
```

Boom! How easy is that? Obviously, every time you refresh or save the images will change, because, well, they are random.

Try out this little interaction to see how easy it is to use random images.

```
for i in [0...6]
  pic = new Layer
    image: Utils.randomImage()
    size: 100
    y: Align.center
    x: 110 * i + 25

  pic.onClick ->
    @image = Utils.randomImage()
```

c) Number

Imagine a world where you could generate random numbers in one line of syntax. The future is here, my friends.

`Utils.randomNumber()` allows you to generate random numbers with ease. You can use the randomly generate numbers for size, position, time, html or anything else you can think of.

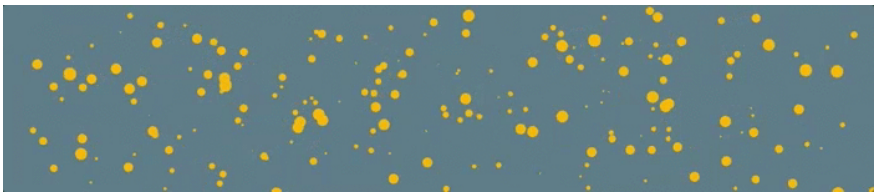
By default, `Utils.randomNumber()` gives you a number between 0–1, which means it will probably be a decimal. To try it, input the following into Framer:

```
layer = new Layer
   html : Utils.randomNumber()
```

To make a randomly sized layer that is bigger than 5 px and smaller than 500px, just do the following:

```
layer = new Layer
   size: Utils.ranomNumber(5,500)
```

Once you have understood the above, try the example below to see where else you can use `Utils.random()` .



```
for i in [0...200]
  dot = new Layer
     size: Utils.randomNumber(1,10)
     borderRadius: "100%"
     backgroundColor: "#FFB502"
     x: Utils.randomNumber(1,Screen.width)
     y: Utils.randomNumber(1,Screen.height)

 dot.animate
   properties:
      x: Utils.randomNumber(1,Screen.width)
      y: Utils.randomNumber(1,Screen.height)
   time:  Utils.randomNumber(0.3,5)
   repeat: 100
```

d) choice

`Utils.randomChoice()` makes a random choice for you, picked from an array. So, in essence, you give Framer a mini set of data and it makes a random selection out of that.

There are two ways to do this.

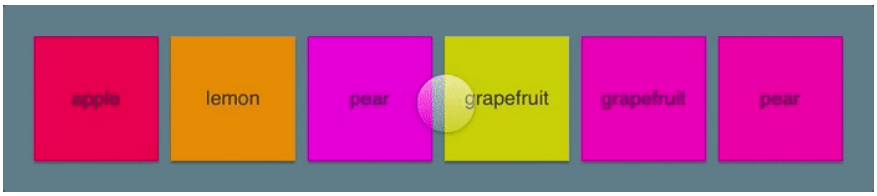
Option one:

```
layer = new Layer
   html: Utils.randomChoice(['Mary', 'Ben', 'Sam'])
```

Option two:

```
arrayNames = ['Mary', 'Ben', 'Sam']

layer = new Layer
   html: Utils.randomChoice(arrayNames)
```

The example below incorporates both of these methods.




```
fruit = ['apple', 'banana', 'orange', 'watermelon', 'kiwi',
'grapefruit', 'pear', 'lemon']

for i in [0...6]
   layer = new Layer
       backgroundColor : Utils.randomChoice(['#FF8800',
'#FFB502', '#FF0051', '#DD00FF', '#006EFF'])
      size: 100
      x: 110 * i + 25
      html: Utils.randomChoice(fruit)

   layer.onClick ->
      @html = Utils.randomChoice(fruit)
      @backgroundColor = Utils.randomChoice(['#FF8800',
'#FFB502', '#FF0051', '#DD00FF', '#006EFF'])
```

. . .

3. Utils.cycle()

`Utils.cycle()` allows you to cycle through values. It's a similar concept to states when you cycle through them. But the main difference is that you have to declare it outside the event.

So, if you look at the example below, the *toggle* has become the `Utils.cycle()` . If you move the `Utils.cycle()` itself into the event —it will break.

…and don't ask me why, I don't know either.

Try out the example below.


```
layer = new Layer
  html: "zero"

toggle = Utils.cycle("zero", "one", "two ", "three", "four",
"five")
```

```
layer.onClick ->
  layer.html = toggle()
```

. . .

4. Utils.round()

Learning to round numbers in math was pretty cool. Only problem is now as an adult (I use the word “adult” very loosely), I tend to round-down as opposed to up, when I am shopping.

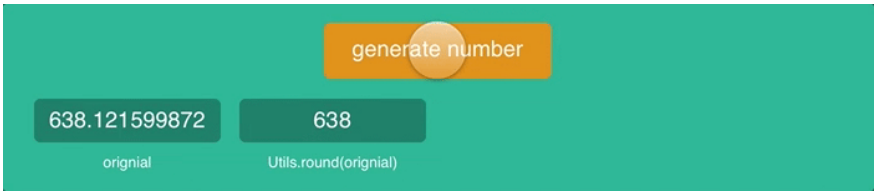
Anyway.

`Utils.round()` allows you to round a number up to the closest integer, or decimal or interval. To do this correctly, you almost work in degrees of complexity.

Simple round

This is the simplest way to round a number to the closest integer (for those of you who haven’t done high school math in the last 5 years, an **integer is a number that doesn’t have a decimal/fraction**. It is a ‘whole’ number).

The basic structure to round a number goes like:
`Utils.round(` *number* `)` .



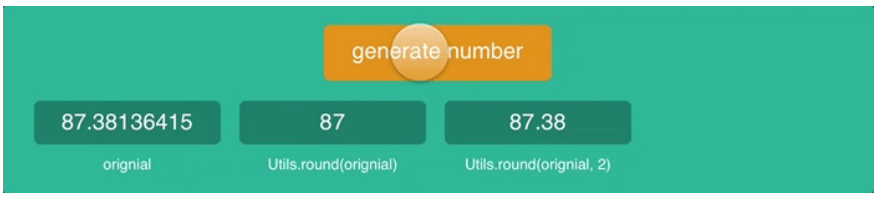
```
Original.html = 638.33546734572234673

RoundedNumber1.html = Utils.round(638.33546734572234673)
# 638
```

Pro tip: Alternatively, you can also use `number.html = Math.round(638.3354)`

Round to a specific decimal place

Now, let's say that you are working with money and need to get your value to have two decimal places. Luckily this is an easy fix with Framer. To do this, just put a comma after the value you need rounded, and the amount of decimal places you want to round to.

The basic structure is:
`Utils.round(` *number* `,` *amount of digits after decimal* `)`




```
Original.html = 638.33546734572234673

RoundedNumber1.html = Utils.round(638.33546734572234673)
# 638

RoundedNumber2.html = Utils.round(638.33546734572234673, 2)
# 638.34
```

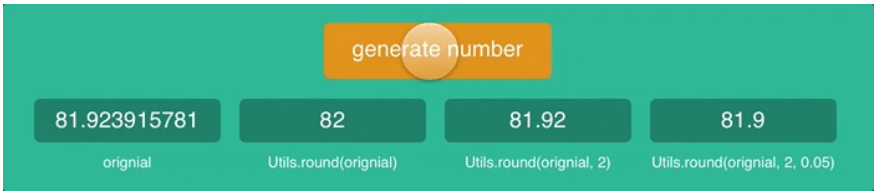
Round to a specific decimal using an interval

Now let's say you want to round to certain intervals—wait, let's have a refresher of what an interval is.

An interval is a value that lies between two or more numbers. For example: 3, 6, 9, 12; the interval is 3.

So, in Framer we can round to the nearest interval, but to do this we also need to round to a specific decimal.

The basic structure:
`Utils.round(` *number* `,` *amount of digits after decimal* `,` *interval* `)`




```
Original.html = 638.33546734572234673
```

```
RoundedNumber1.html = Utils.round(638.33546734572234673)
# 638
```

```
RoundedNumber2.html = Utils.round(638.33546734572234673, 2)
# 638.34
```

```
RoundedNumber3.html = Utils.round(638.33546734572234673, 2,
0.05)
# 638.35
```

. . .

5. Utils.Interval

`Utils.interval` allows you to have intervals of time between different things. This is pretty neat if you want things to feed in slowly.

Side note: Interval carries on forever.

```
ii = 50
hr = 0
```

```
Utils.interval 0.5, ->
```

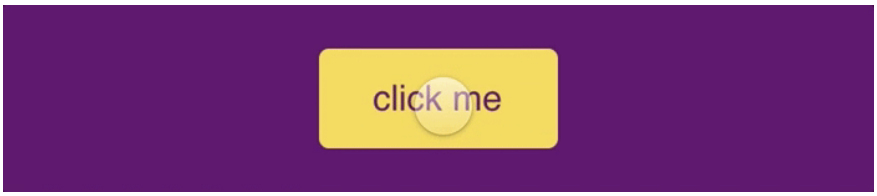
```
   layer = new Layer
      size: 100
```

```
        y: Align.center
        x: ii
        backgroundColor: "#FD81FF"
        hueRotate: hr


    ii = ii + 125
    hr = hr + 20
```

. . .

6. Utils.delay

This is a nifty trick to delay animations or actions a bit after an event. In the example below, you can see that the action happens only 0.6 seconds after I click the button.




```
button.onClick ->

 Utils.delay 0.6, ->
  button.scale = bttn.scale + 0.2
```

. . .

7. Utils.modulate()

`Utils.modulate()` is a tricky concept to explain—BUT basically, it allows you to change something incrementally. Think converting Fahrenheit into Celsius, or miles into kilometers.

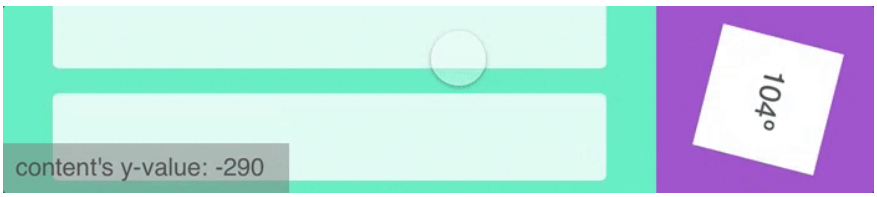
When using `Utils.modulate` there are usually two elements, a **Property that effects change** and **Property that is affected by change**. And both of these need need a limit. In example, let's use miles and kilometers. (♪…because I would walk…♪) 100 miles = 160.934 km and 0 miles = 0 km. So, zero is the beginning limit for km and miles, but 160.934 is the end limit for km and 100 is the end limit for miles.

But `Utils.modulate()` isn't limited to just conversions. It can also be used on layer transformations.

It's basic structure is:
layerB.property `= Utils.modulate(` *layerA.property* `, [` *layerA beginning limit* `,` *layerA end limit* `], [` *layerB beginning limit* `,` *layerB end limit* `], true)`

So, If you understand ^ that, you are very smart.

For the rest of us, try the example below, or try this cheat sheet which explains it significantly better: Framer Cheat Sheets: Utils.modulate




```
ScrollComp = new ScrollComponent

shape = new Layer

ScrollComp.onMove ->
   shape.rotation = Utils.modulate(ScrollComp.content.y, [0,
-1000], [0, 360])
```

. . .

8. Utils. platform detection

If you are making a responsive website or app—your prototype will need to adapt to whatever format it is on. Using platform detection can help you do this. You will, however need to specify what browser / platform you are on.

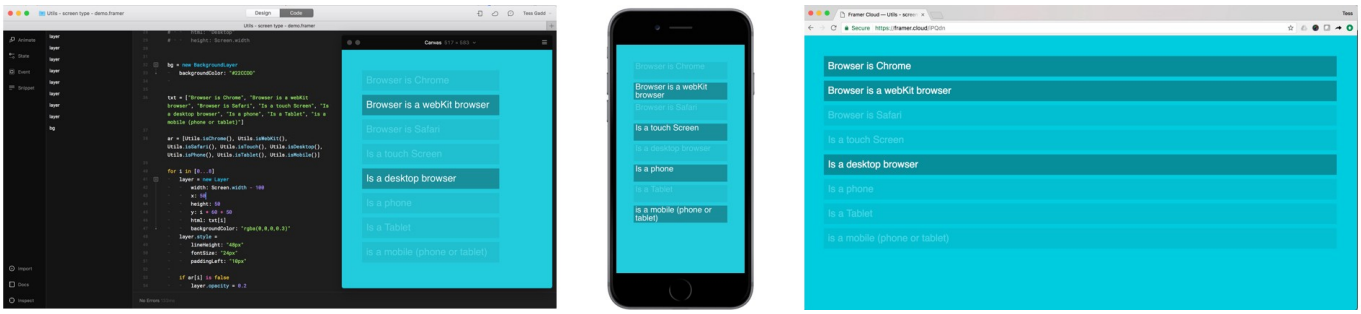
`Utils.` *platform* `()` will return a true or false statement.

Below are the various platform detections that Utils offers.

```
#To see if the browser is Chrome
Utils.isChrome()
```

```
#To see if the browser is a webKit browser
Utils.isWebKit()

#To see if the browser is Safari
Utils.isSafari()

#To see if the device is a touch screen
Utils.isTouch()

#To see if the device is a desktop
Utils.isDesktop()

#To see id the device is a phone
Utils.isPhone()

#To see if the device is a table
Utils.isTablet()

#To see if the device is a phone or a tablet
Utils.isMobile()
```

Give the below prototype a try on different platforms / browsers.



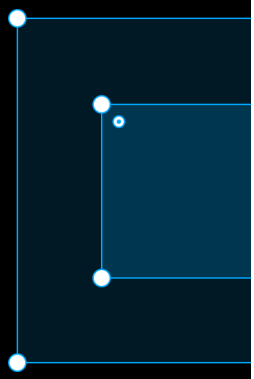
https://framer.cloud/iPQdn

. . .

Look how clever we are! Hopefully this cheat sheet helped you, and if it didn't or you want to learn more about something else, please leave me comment, and I will update :)

Introducing Framer Design

Read More



Join the community

Visit Group

Try Framer for free

Download Trial