Present

```
br =
```

Welcome to 18.S191 – Fall 2020!

Introduction to Computational Thinking for Real-World Problems

https://github.com/mitmath/18S191

Alan Edelman, David P. Sanders, Grant Sanderson, James Schloss

& Philip the Corgi

Class outline

Data and computation

- Module 1: Analyzing images
- Module 2: Particles and ray tracing
- Module 3: Epidemic spread
- Module 4: Climate change

Tools

- Julia programming language: **http://www.julialang.org/learning**

- Pluto notebook environment

Module 1: Images

8

```
4 + 4
```

Data takes many forms

- Time series:
  - Number of infections per day
  - Stock price each minute
  - A piece for violin broadcast over the radio
- Video:
  - The view from a window of a self-driving car
  - A hurricane monitoring station
- Images:
  - Diseased versus healthy tissue in a scan
  - Deep space via the Hubble telescope
  - Can your social media account recognise your friends?

Capture your own image!

Enable webcam

```
@bind raw_camera_data camera_input(;max_size=2000)
```

```
UndefVarError: grant not defined

  1. top-level scope @ Local: 1
```

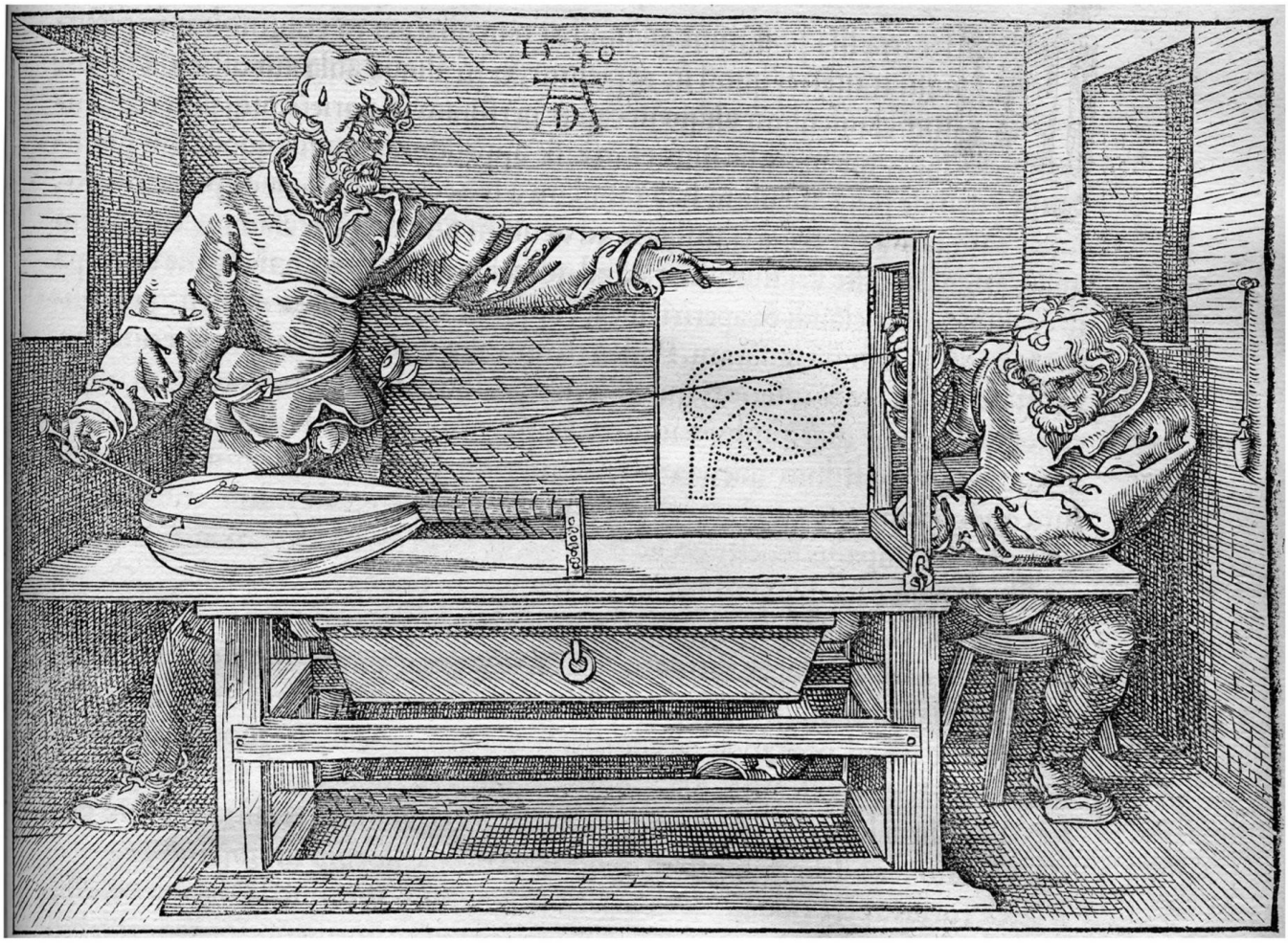
```
[
    grant             grant[:,end:-1:1]
    grant[end:-1:1,:] grant[end:-1:1,end:-1:1]
]
```

```
grant =
MethodError: no method matching getindex(::Missing, ::String)

  1. process_raw_camera_data(::Missing) @ Other: 13
  2. top-level scope @ Local: 1
```

```
grant = decimate(process_raw_camera_data(raw_camera_data), 2)
```

What is an image?

Albrecht Dürer:



An image is:

- A 2D representation of a 3D world
- An approximation

What *is* an image, though?

- A grid of coloured squares called **pixels**
- A colour for each pair (i, j) of indices
- A **discretization**

How can we store an image in the computer?

- Is it a 1D array (`Vector`)?
- A 2D array (`Matrix`)?

- A 3D array (`tensor`)?

If in doubt: Ask Julia!

- Let's use the `Images.jl` package to load an image and see what we get

```
begin
    import Pkg
    Pkg.activate(mktempdir())
end
```

```
begin
    Pkg.add(["Images", "ImageIO", "ImageMagick"])
    using Images
end
```

```
url = "https://i.imgur.com/VGPeJ6s.jpg"
```

```
# defines a variable called `url`
# whose value is a string (written inside `"`):

url = "https://i.imgur.com/VGPeJ6s.jpg"
```

```
philip_file = "philip.jpg"
```

```
philip_file = download(url, "philip.jpg")  # download to a local file
```

```
philip =
```

```
philip = load(philip_file)
```

- philip

```
[
    philip             philip[:,end:-1:1]
    philip[end:-1:1,:] philip[end:-1:1,end:-1:1]
]
```

```
Array{RGB{Normed{UInt8,8}},2}
```

```
typeof(philip)
```

```
RGBX(0.1, 0.2, 0.5)
```

```
Array{RGB{Normed{UInt8,8}},2}
```

```
typeof(philip)
```

- According to Julia / Pluto, the variable `philip` *is* an image
- Julia always returns output
- The output can be displayed in a "rich" way

- Arthur C. Clarke:

> Any sufficiently advanced technology is indistinguishable from magic.

```
(3675, 2988)
```

```
size(philip)
```

- philip

```
philip[1:1000, 1:400]
```

How big is Philip?

- He's pretty big:

```
(3675, 2988)
```

```
size(philip)
```

- Which number is which?

- philip

So, what *is* an image?

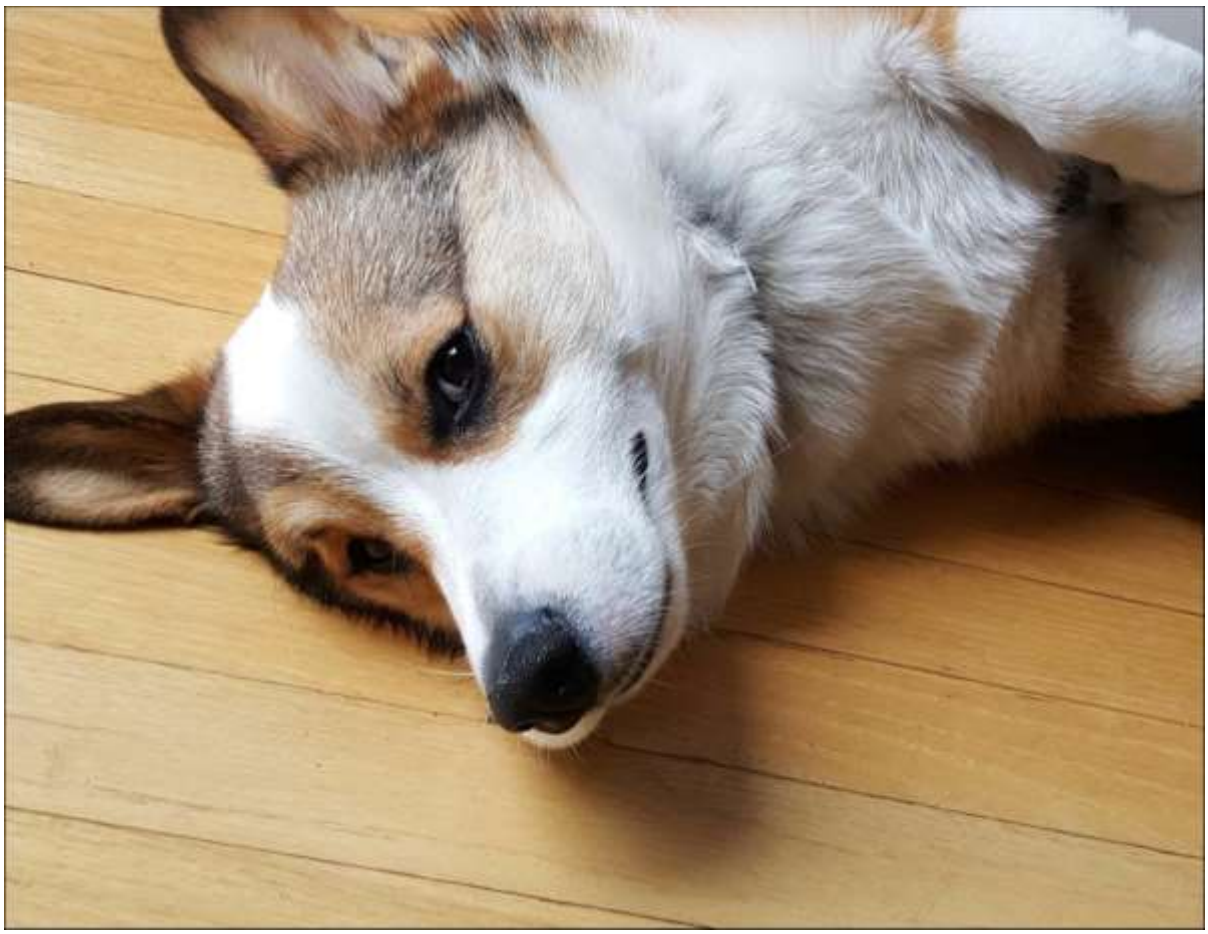
```
Array{RGB{Normed{UInt8,8}},2}
```

```
typeof(philip)
```

- It's an `Array`
- The `2` means that it has **2 dimensions** (a **matrix**)

- `RGBX{Normed{UInt8,8}}` is the type of object stored in the array
- A Julia object representing a colour
- RGB = Red, Green, Blue

Getting pieces of an image



```
begin
    (h, w) = size(philip)
    head = philip[(h ÷ 2):h, (w ÷ 10): (9w ÷ 10)]
    # '÷' is typed as \div <TAB>  -- integer division
end
```

```
(1839, 2392)
```

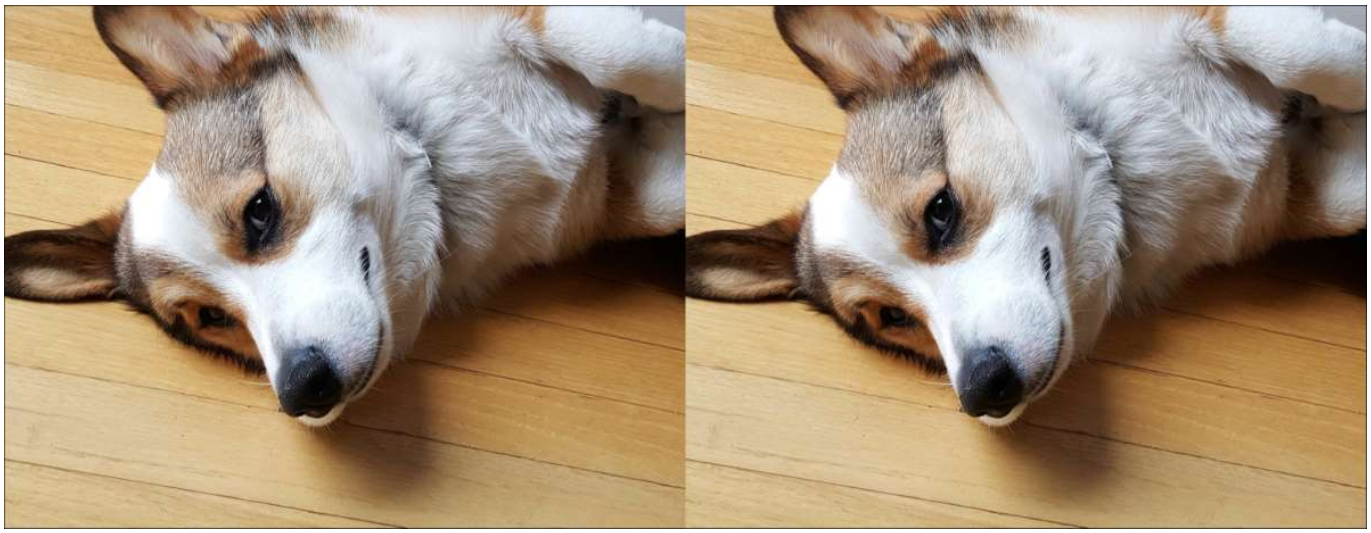
```
size(head)
```

```
(3675, 2988)
```

```
size(philip)
```

Manipulating matrices

- An image is just a matrix, so we can manipulate *matrices* to manipulate the *image*



```
[head head]
```

```
[
 head                   reverse(head, dims=2)
 reverse(head, dims=1)  reverse(reverse(head, dims=1), dims=2)
]
```

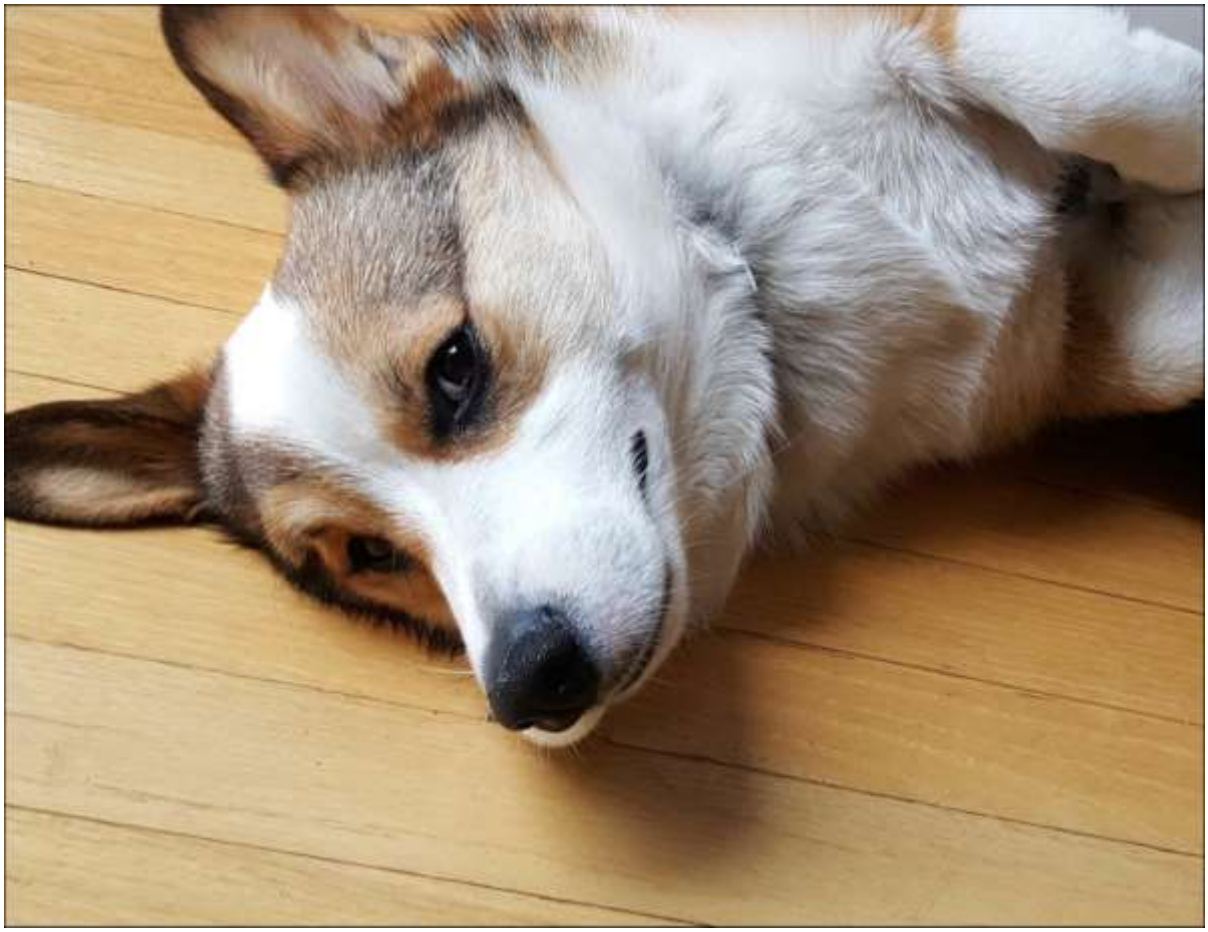
Manipulating an image

- How can we get inside the image and change it?
- There are two possibilities:
  - **Modify** (**mutate**) numbers inside the array – useful to change a small piece
  - Create a new **copy** of the array – useful to alter everything together

Painting a piece of an image

- Let's paint a corner red
- We'll copy the image first so we don't destroy the original

```
new_phil =
```



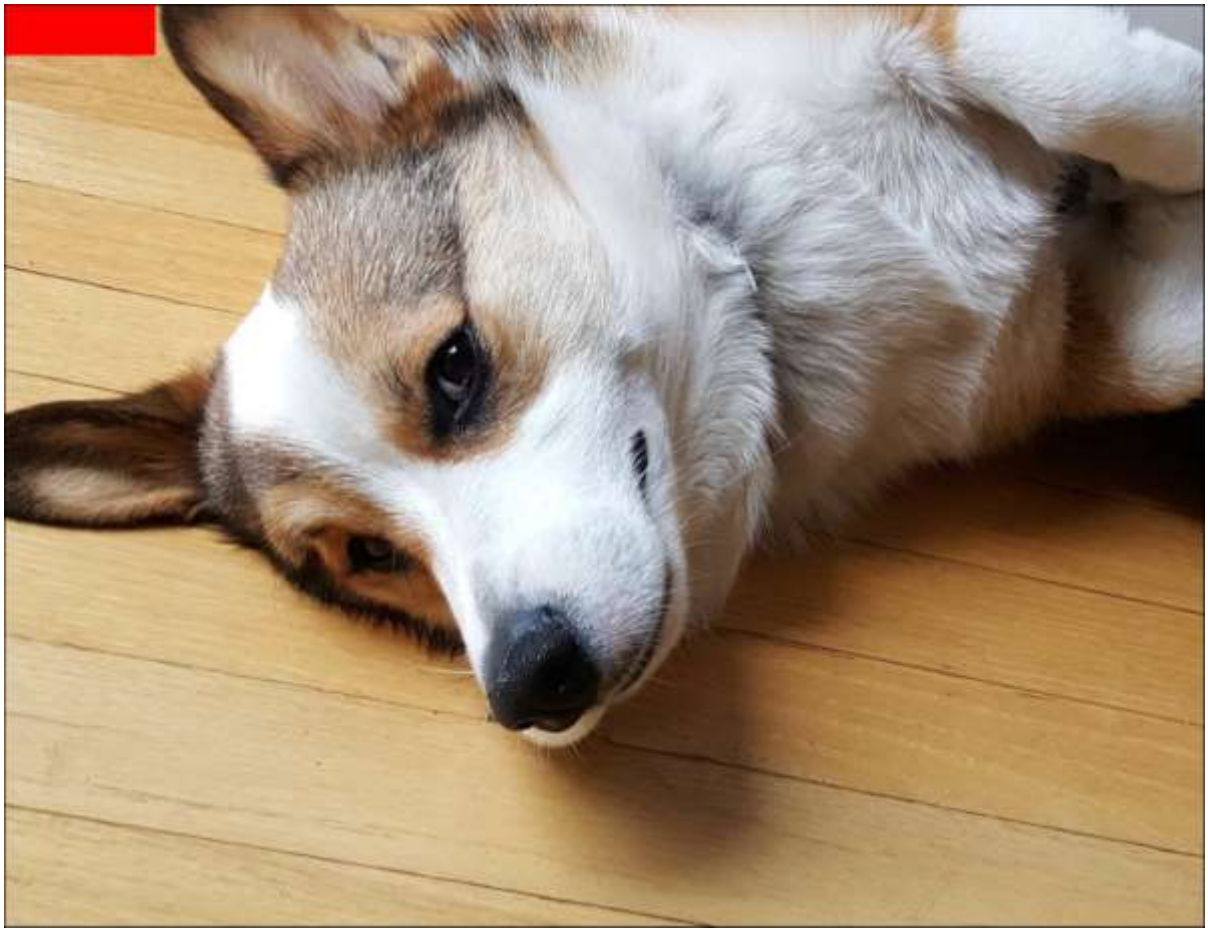
```
new_phil = copy(head)
```

red =

```
red = RGB(1, 0, 0)
```

```
for i in 1:100
    for j in 1:300
        new_phil[i, j] = red
    end
end
```

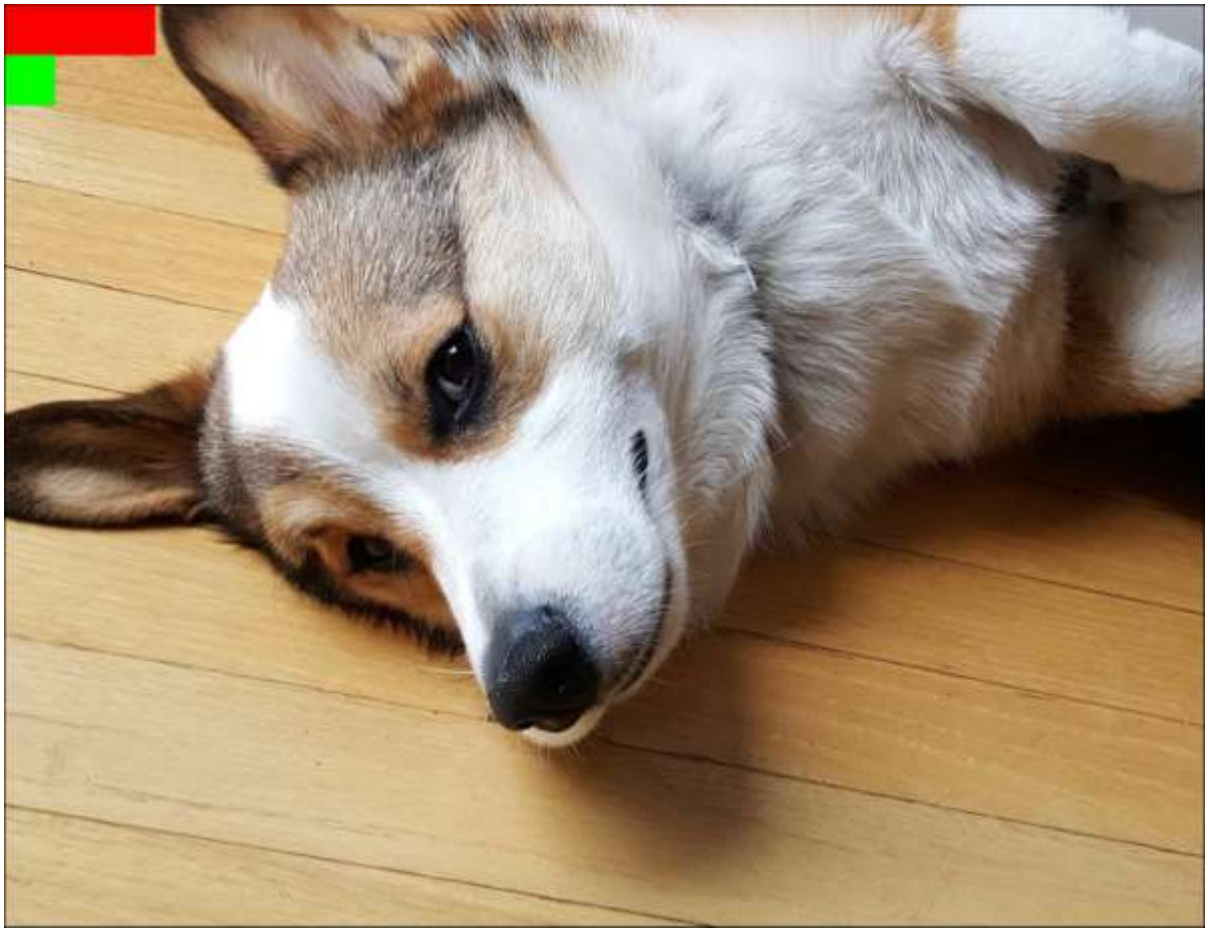
Note that `for` loops *do not return anything* (or, rather, they return `nothing`)



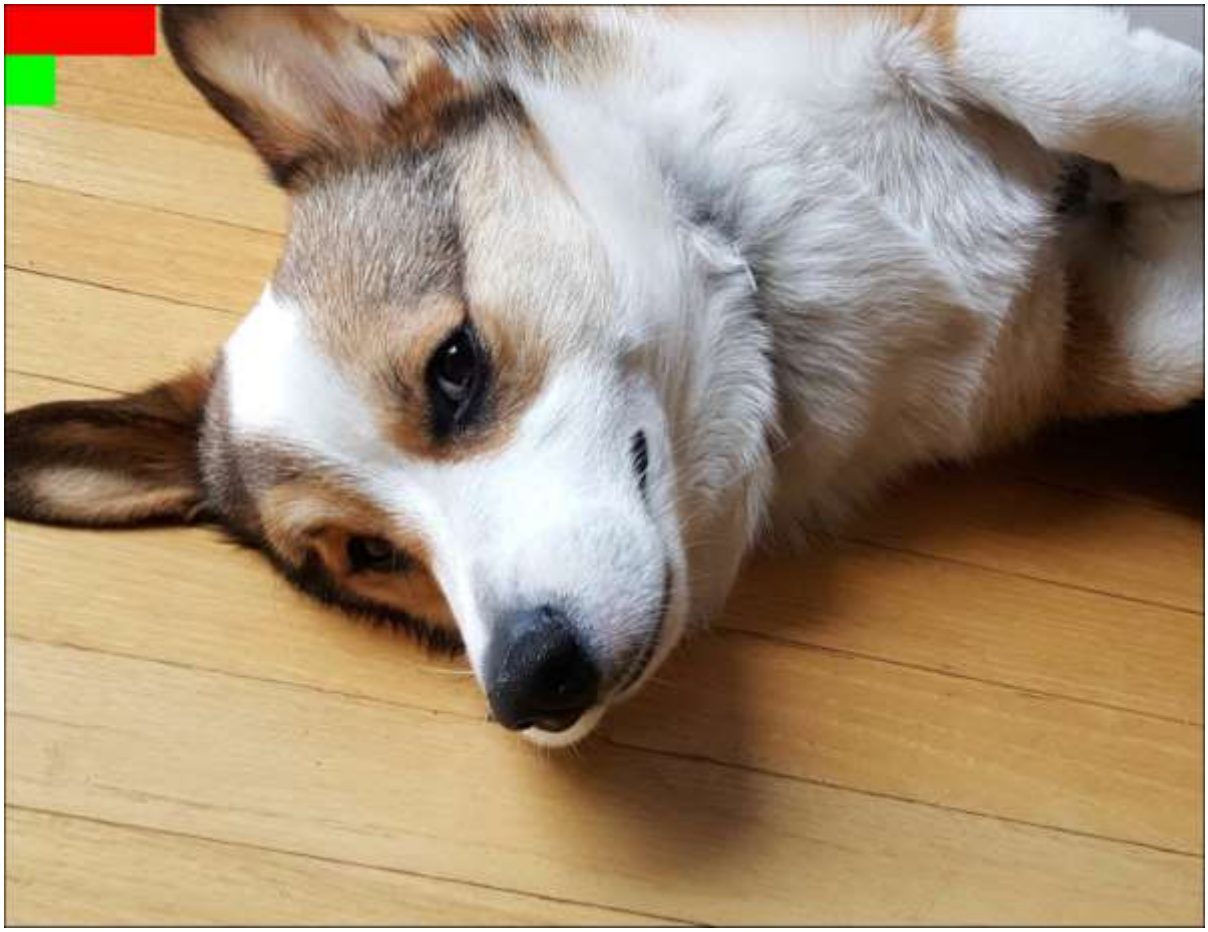
- `new_phil`

Element-wise operations: "Broadcasting"

- Julia provides powerful technology for operating element by element: **broadcasting**
- Adding " `.` " applies an operation element by element



```
begin
    new_phil2 = copy(new_phil)
    new_phil2[100:200, 1:100] .= RGB(0, 1, 0)
    new_phil2
end
```



```
new_phil2
```

Modifying the whole image at once

- We can use the same trick to modify the whole image at once
- Let's **redify** the image
- We define a **function** that turns a colour into just its red component

```
redify (generic function with 1 method)
```

```
function redify(c)
    return RGB(c.r, 0, 0)
end
```

RGB{Float64}[,]

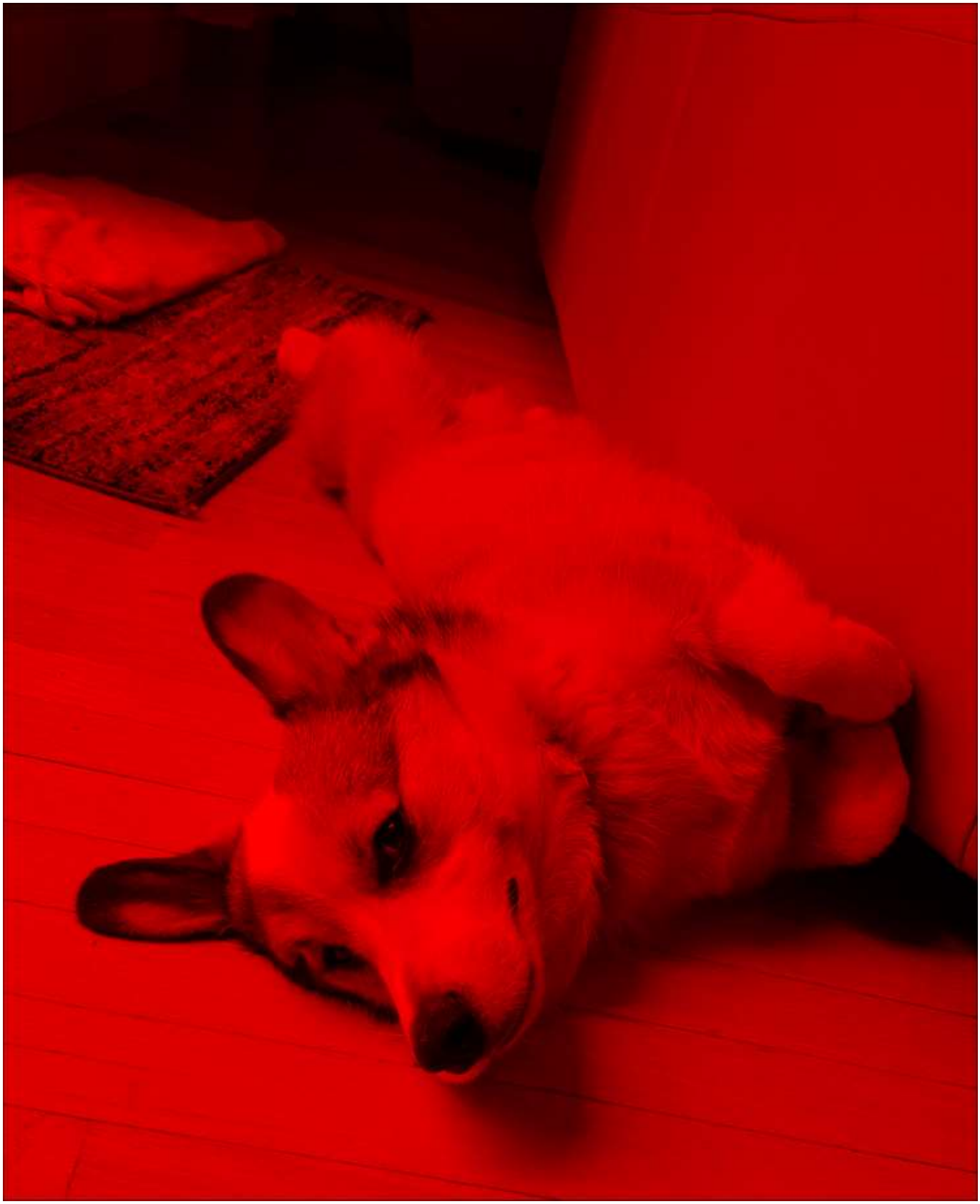
```
begin
    color = RGB(0.9, 0.7, 0.2)

    [color, redify(color)]
end
```

```
blue_bomb (generic function with 1 method)
```

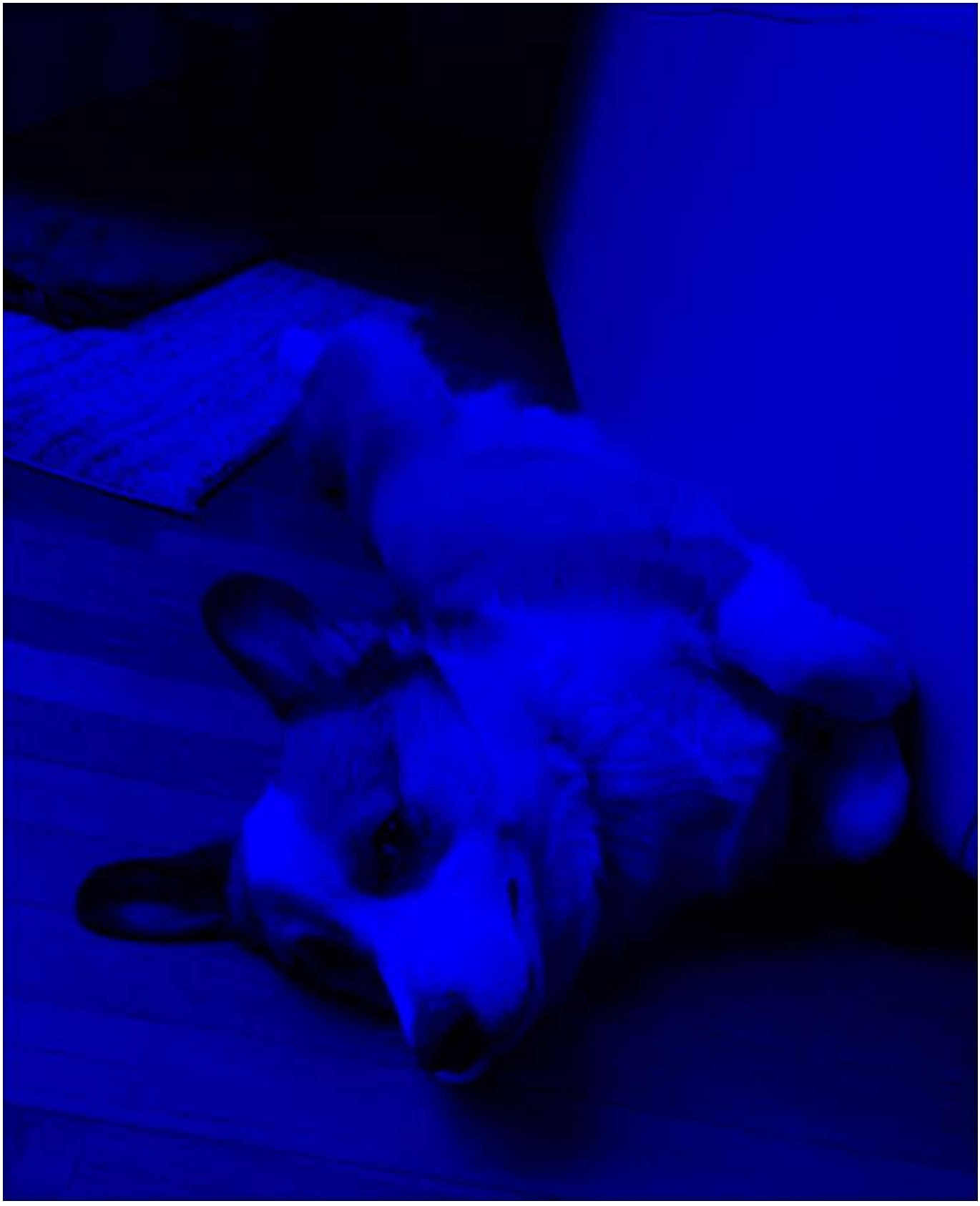
```
function blue_bomb(image)
    return RGB(0, 0, image.b)
end
```

RGB{Float64}[ , ]

```
begin
    temp_pixel = RGB(0.9, 0.1, 0.5)
    [temp_pixel, blue_bomb(temp_pixel)]
end
```



- `redify.(philip)`

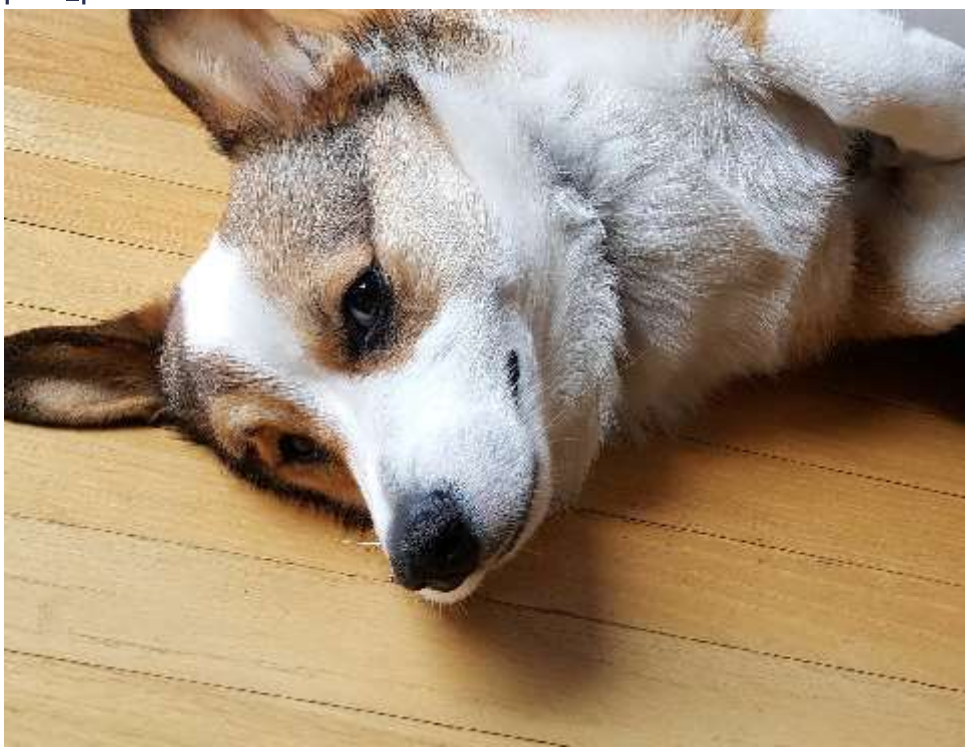


- `blue_bomb.(philip)`

Transforming an image

- The main goal of this week will be to transfrom images in more interesting ways
- First let's **decimate** poor Phil

poor_phil =



```
poor_phil = decimate(head, 5)
```

poor_phil_10 =

```
poor_phil_10 = decimate(head, 10)
```

```
(184, 240)
```

```
size(poor_phil_10)
```

```
(1839, 2392)
```

```
size(head)
```

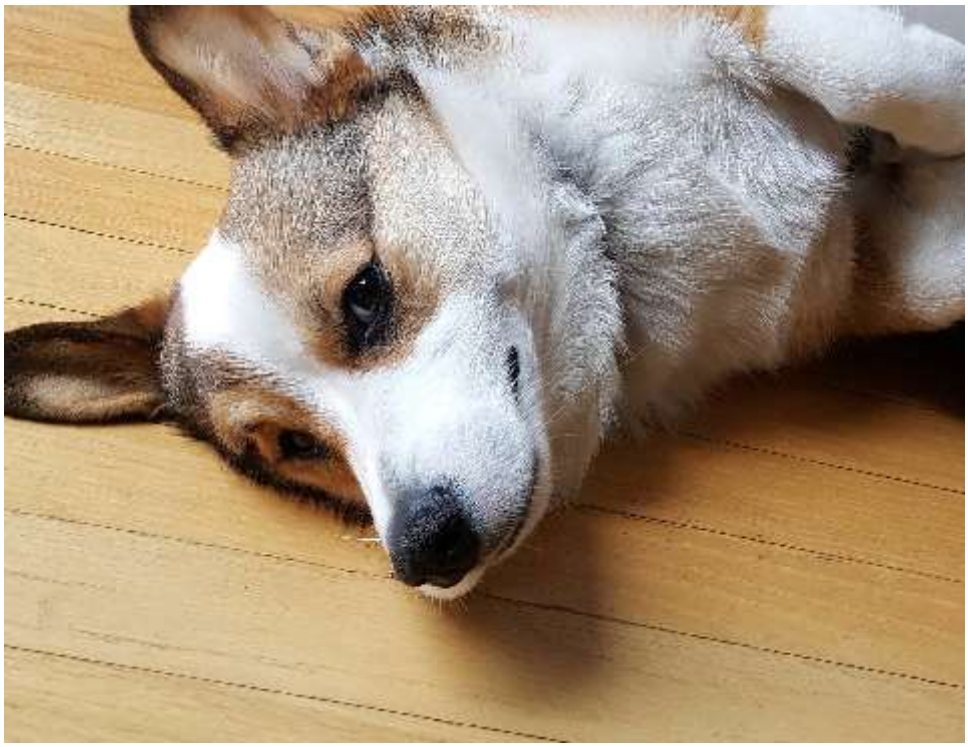
Experiments come alive with interaction

- We start to get a feel for things when we can **experiment**!

```
begin
    Pkg.add("PlutoUI")
    using PlutoUI
end
```

1

```
@bind repeat_count Slider(1:10, show_value=true)
```



```
repeat(poor_phil, repeat_count, repeat_count)
```

Summary

- Images are readily-accessible data about the world
- We want to process them to extract information
- Relatively simple mathematical operations can transform images in useful ways

```
expand (generic function with 2 methods)
```

```
extract_red (generic function with 1 method)
```

```
decimate (generic function with 2 methods)
```

Appendix

Package environment

Camera input

```
camera_input (generic function with 1 method)
```

```
process_raw_camera_data (generic function with 1 method)
```