

What Do You See?



You See a Cow. Python Sees...



- » Python has never met a cow. It doesn't know what one looks like.
- » It doesn't even know what a "photo" is.
- » All Python ever sees is **numbers**.
- » So how does a program work with pictures at all?
- » Today you will find out what you can do with those numbers. There is a puzzle waiting at the end.

// think, pair, share: how many numbers is one photo?

Predict: What Numbers?

- ☐ If a picture is just numbers... how many numbers?
- ☐ Which numbers? Big ones? Small ones?

$320 \times 320 = 102,400$ tiny squares (pixels)

3 numbers per pixel: red, green, blue

each number is between 0 and 255

// 307,200 numbers for one small photo. 1 minute alone, 1 with your partner, then the reveal.

IMAGE ANALYSIS

Image Analysis

```
$ image_explore.py
```

What a picture is to a computer, what you can do with it, and a puzzle to solve.

— Dr. Hughes

PART 1

Explore a Picture

open

one pixel

crop

colour

channels

switch off

grey

count

histogram

// 8 short tasks in Spyder, then Part 2: the puzzle

Nobody Writes Everything Themselves

variables

functions

if / else

loops

A **library** is a bunch of functions someone else already wrote, ready for you to use.

```
>>> import math
```

```
>>> math.sqrt(16)
```

```
4.0
```

A Library for Pictures

» **PIL** (short for *Pillow*) opens, edits and reads images.

```
>>> from PIL import Image
```

from X import Y = "just give me Y out of library X".

Open It, Check It

```
>>> img = Image.open("cow-demo.jpg")  
  
>>> img.size  
(320, 320)  
  
>>> img.mode  
'RGB'
```

// one line opens the whole picture: 320 pixels wide, 320 tall

A Picture Is a Grid of Numbers

```
# One small group of numbers per pixel. NumPy does maths on huge grids, fast.
```

```
>>> import numpy as np
```

```
>>> pixels = np.array(img)
```

// `pixels` is now the whole picture as a grid

// TASK 1: open the picture in Spyder and read its shape

Open a Picture

- ☐ Open `image_explore.py` in Spyder. `cow-demo.jpg` must be in the same folder.
- ☐ Predict first: how many rows? how many columns?
- ☐ Run the file (green triangle or F5). Read `shape` in the console.
- ☐ Want it on paper? **Worksheet 06: The explore code** is this whole file

// F9 runs just the lines you have selected. Check: `(320, 320, 3)` = rows × columns × 3 colour numbers.

Rows and Columns



- » Rows count **down**. Columns count **across**.
- » Counting starts at **0**, top left.
- » Row first, then column.

```
>>> print(pixels[0, 0])  
      [16  0  0]
```

// red 16, green 0, blue 0: almost black

// TASK 2: find a pixel of your own

Find a Pixel

- ☐ Run the `print(pixels[0, 0])` line. Does it match?
- ☐ Print the pixel in row 100, column 200
- ☐ Hint: `pixels[row, column]`

// predict first: which of the three numbers will be biggest? Check: `[94 50 37]`, a reddish brown.

Crop: a Block of Pixels

```
>>> corner = pixels[0:100, 0:100]
```



rows 0 to 99, columns 0 to 99

show(corner): just that box

// **[start:stop]** for rows, a comma, **[start:stop]** for columns. Like **range**, it stops before the last number.

// TASK 3: crop a square of your own

Crop a Corner

- ❑ Run TASK 3. See the top-left corner.
- ❑ Crop a different 100×100 square, then show it
- ❑ Hint: change the numbers, keep the colon and the comma
- ❑ On paper: **Worksheet 01: Pixel coordinates**

// use the axis numbers on the last picture. Can you crop just the cow's head?

Every Pixel: 3 Numbers

R = red

G = green

B = blue

Each one is a brightness: **0** (none of that colour) to **255** (as much as possible).

// on paper: Worksheet 02: What Python Sees

A 4 × 4 Crop, and Its Real Numbers



```
// pixels[100:104,  
100:104]
```

113, 73, 63

123, 80, 71

136, 92, 81

141, 95, 80

125, 83, 71

139, 95, 84

150, 104, 91

147, 99, 85

132, 88, 77

140, 93, 83

145, 97, 85

142, 93, 78

125, 81, 68

130, 82, 70

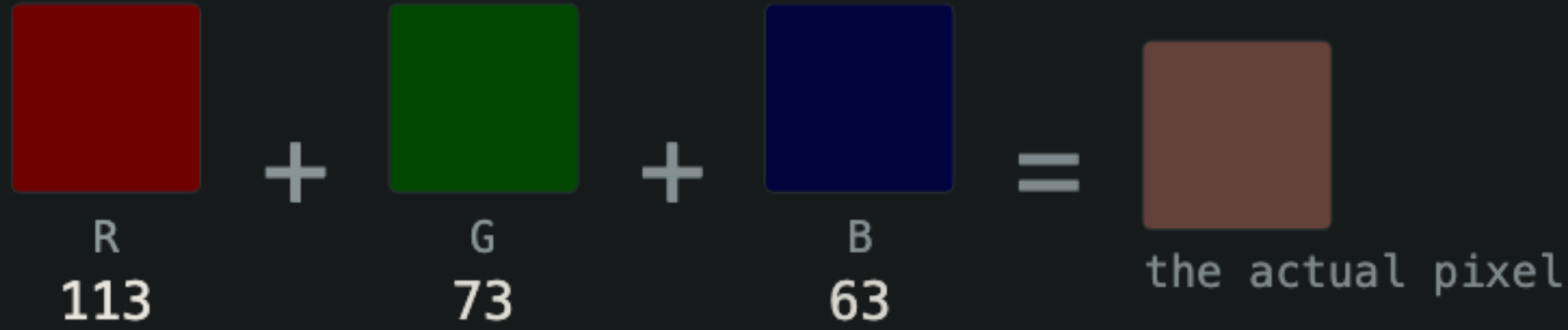
138, 89, 75

144, 92, 78

Worksheet 02: What Python Sees

- ☐ Questions about the 4×4 patch you just saw
- ☐ Pencil, not Python. Go back a slide if you need the numbers.

Take One Pixel: (113, 73, 63)



Change the numbers, change the picture. That is *all* a photo is.

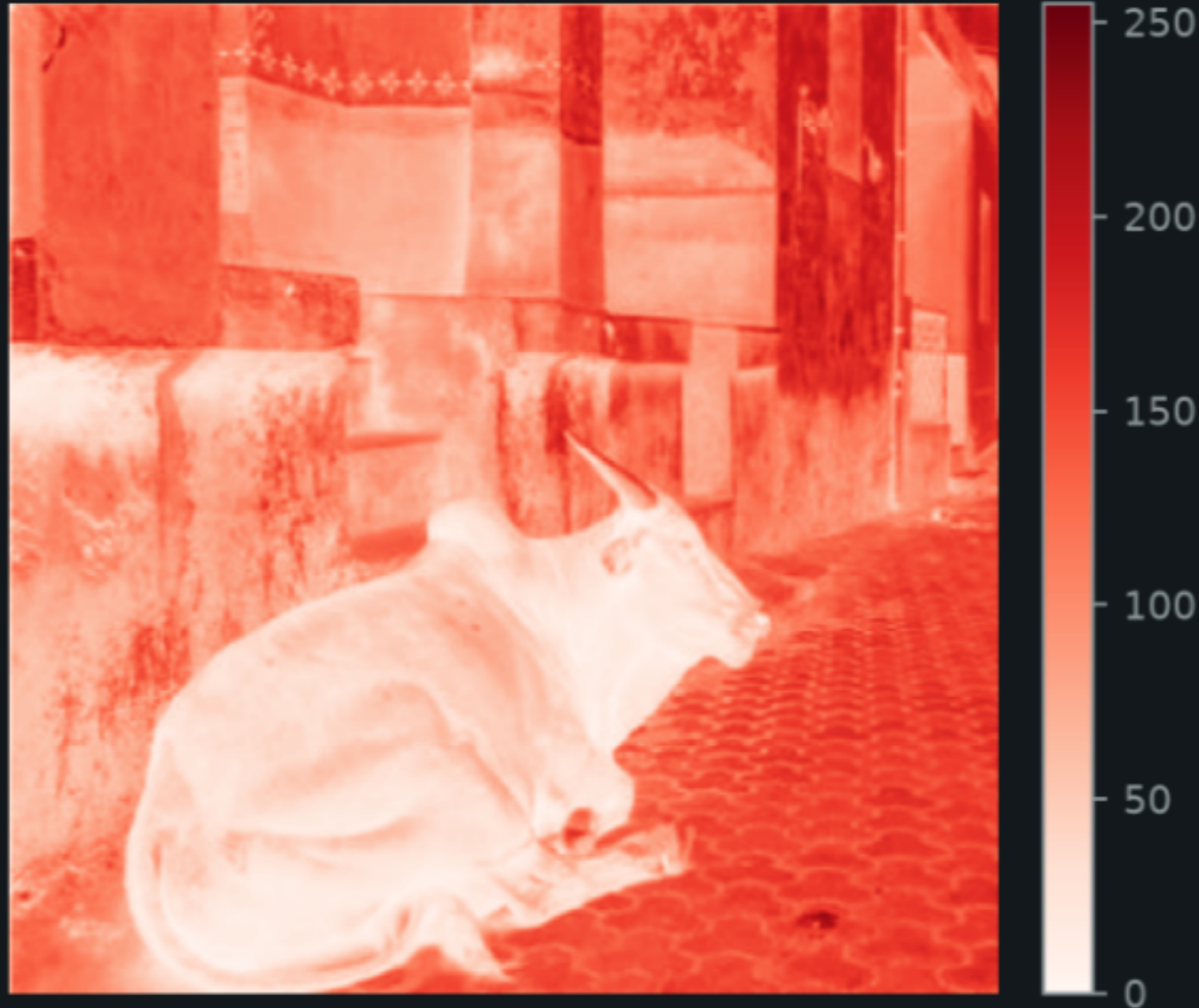
// TRY IT: mix your own colours in `image_explore.py`

Mix a Colour

- ☐ In `image_explore.py`, find `show_colour(230, 30, 30)`
- ☐ Change the three numbers (red, green, blue). Run it.
- ☐ Make yellow, then make a grey
- ☐ On paper: **Worksheet 03: Mix a colour**

// each number is 0 to 255. `show_colour` is given: no need to read inside it.

Split Into Three Pictures



```
>>> red = pixels[:, :, 0]
```

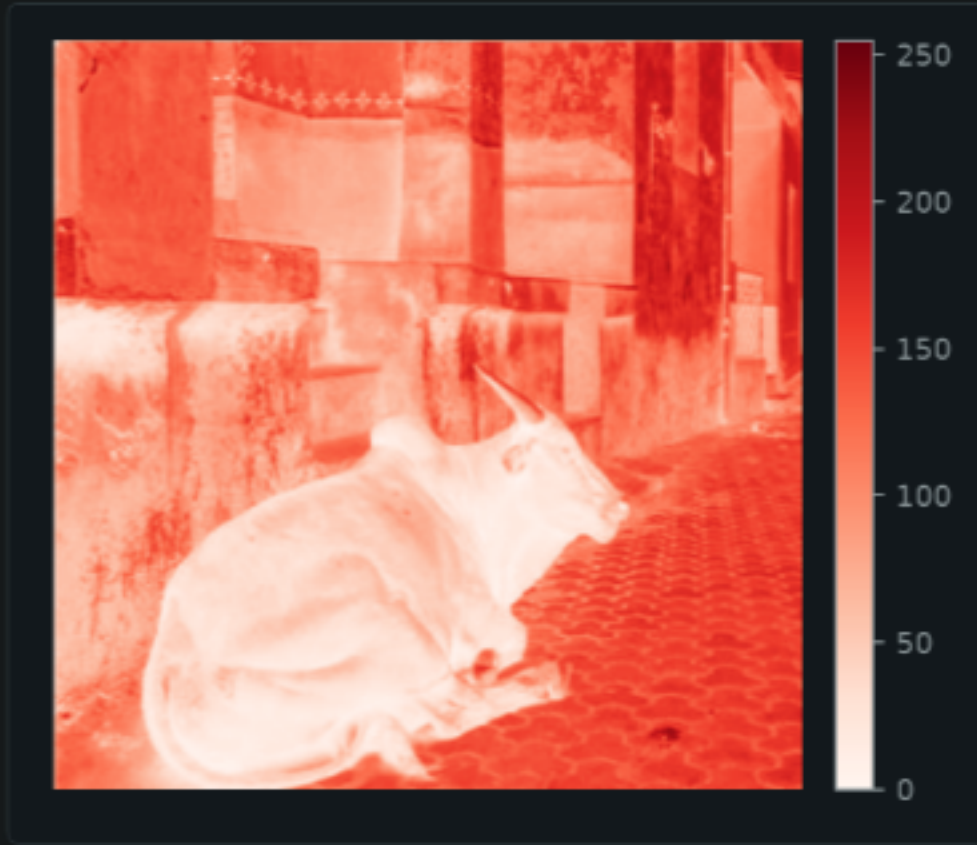
- » Every row, every column, just channel 0: red.
- » Deep red = a **big** number. Pale = a **small** one.
- » Wall: lots of red. Cow: almost none.

// TASK 4: split the channels yourself

Split the Channels

- ☐ Run TASK 4 to see the red channel
- ☐ Do the same for green (channel 1) and blue (channel 2)
- ☐ Hint: `show_channel(green, "Greens")`
- ☐ Which channel is brightest on the wall? Which on the cow?
- ☐ On paper: **Worksheet 04: Channel detective**

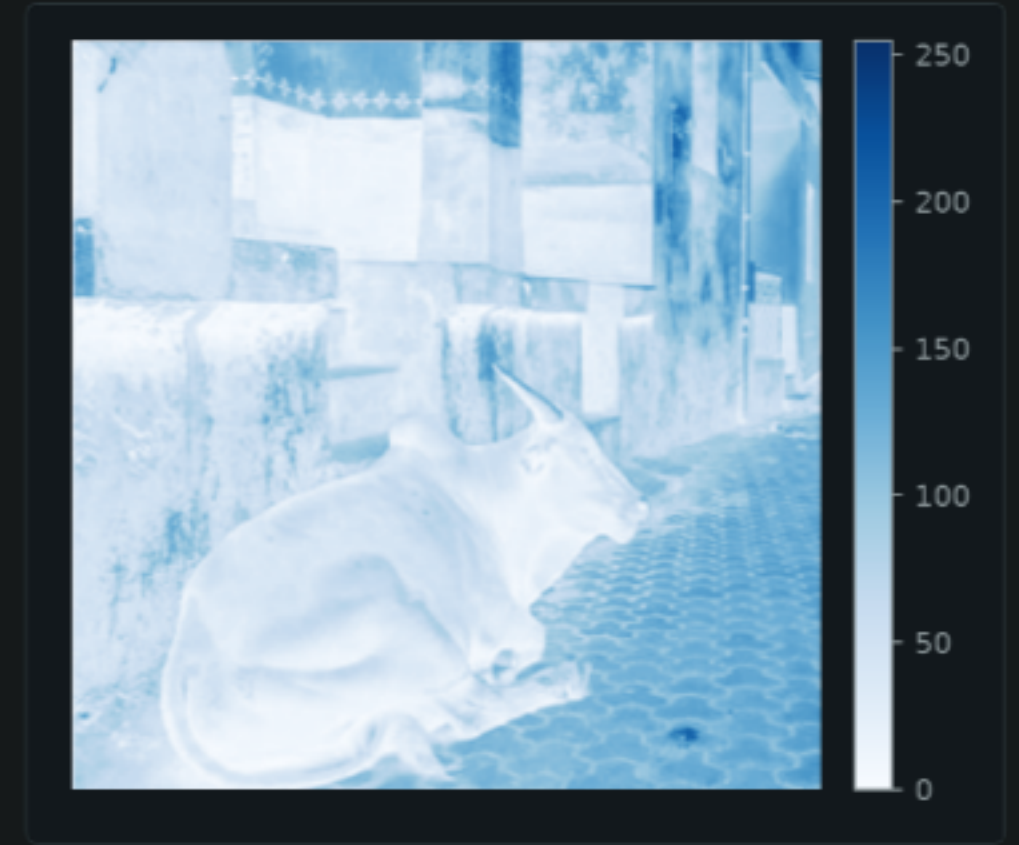
All Three Channels



red



green



blue

Wall = bright in red. Cow = dark in every channel.

Switch Red Off

```
1 no_red = pixels.copy()
2 no_red[:, :, 0] = 0
3 show(no_red)
```

// `.copy()` makes a second picture, so the original stays safe. Line 2 sets every red number to 0.

// TASK 5: switch channels off yourself

Switch Channels Off

- ☐ Predict first: what will the picture look like with no red? no green? no blue?
- ☐ Run TASK 5 (no red). Then make one with NO green, and one with NO blue
- ☐ Make a picture that is ONLY blue (red and green both 0)

Check Your Pictures



no red



no green



no blue



only blue

// red + blue light = magenta. red + green light = yellow.

Grey and Negative

```
1 grey = np.array(img.convert("L"))
2 show_channel(grey, "gray")
3 negative = 255 - pixels
```

// "L" = one brightness number per pixel: 0 black, 255 white. `255 - pixels` flips every number: 0 becomes 255.

// TASK 6: make a grey and a negative picture

Grey and Negative

- ❑ Predict: the first pixel was `[16 0 0]`. What is it in the negative?
- ❑ Run TASK 6. Look at both pictures. What happened to the black cow?



original



grey



negative: [239
255 255]

Maths on the Whole Picture

```
>>> print(grey.mean())  
81.449951171875  
  
>>> print((grey < 50).sum())  
28501
```

// the average brightness: a fairly dark grey (0 is black, 255 is white)

28,501 pixels are darker than 50, out of 102,400.

// `grey < 50` is True or False for every pixel. `.sum()` counts the Trues.

// TASK 7: count pixels and find an average yourself

Count and Average

- ☐ Run TASK 7. Do you get the same numbers?
- ☐ How many pixels are brighter than 100? Hint: change the `<` and the number
- ☐ What fraction of the whole picture is that? Hint: divide by `grey.size`

// predict first: more than half, or less than half?

// check: 34315 pixels, grey.size is 102400, fraction 0.335107421875. About a third.

A Different Question

Not "what is the average?" but: **how many pixels are dark, how many bright, how many in between?**

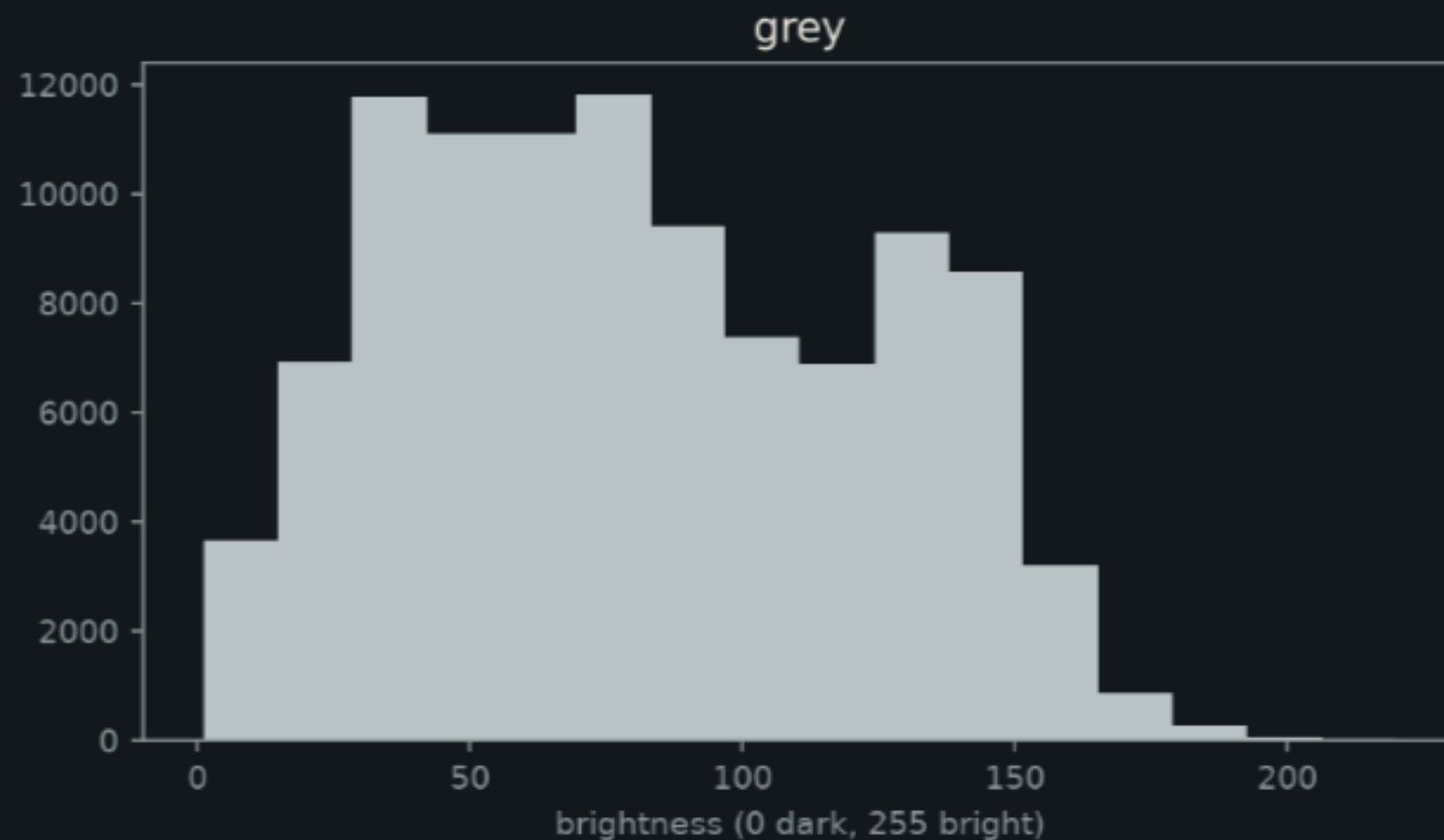
A histogram: split the brightness values into buckets. Count how many pixels land in each one.

// one bar per bucket. Tall bar = lots of pixels with that brightness.

// TASK 8: draw a histogram yourself

Our Photo, in 16 Buckets

```
1 plt.hist(grey.flatten(), bins=16)
2 plt.show()
```

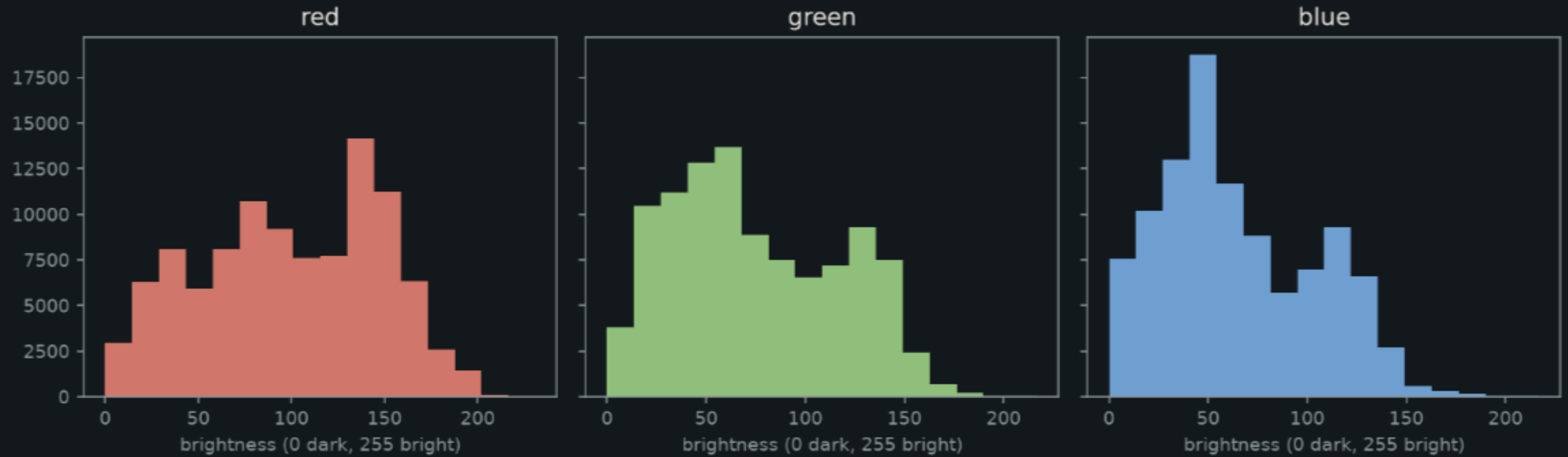


// `.flatten()` turns the grid into one long list. Left = dark, right = bright. Two humps: the dark cow and street, the brighter wall.

Draw a Histogram

- ☐ Run TASK 8. Where are the tallest bars?
- ☐ Draw a histogram of just the red channel: `plt.hist(red.flatten(), bins=16)`
- ☐ Then try green, then blue. What is different?

Red, Green and Blue



Red: a tall bar in the middle-right (the wall). Green and blue: tallest on the left.

Go Backwards: ::-1

```
1 show(pixels[:, ::-1])    # mirror
2 show(pixels[::-1])       # upside down
```



mirror: columns backwards



upside down: rows backwards

// ::-1 means "go backwards"

// STRETCH, then Worksheet 05: Predict and check

Finished Part 1?

- ❑ Try both `::-1` lines from the STRETCH block
- ❑ Brighten it: add 60 to every number. Careful, what happens above 255?
- ❑ On paper: **Worksheet 05: Predict and check**

// predict first, then run.



+ 60 (200 + 60
wraps to 4)

PART 2

\$ cow_message/

The Puzzle

37 photos. One letter is hidden in each. Together they spell a sentence.

The Rule

1. Open a photo.
2. Add up every **red** value in it.
3. Take that total **% 26**.
4. Use the result to pick a letter: A=0, B=1, ... Z=25.

Do it for every photo, in order.

Adding Them All Up

```
>>> red = pixels[:, :, 0]
```

```
>>> red.sum()
```

```
10218857
```

// one number: the total "redness" of all 102,400 pixels of `cow-demo.jpg`

% and Letters

```
>>> 10218857 % 26
```

```
25
```

```
>>> alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

```
>>> alphabet[2]
```

```
'c'
```

Our cow gave 25. Which letter is that?

```
>>> alphabet[25]
```

```
'z'
```

Worksheet 07: Be the Computer

- ☐ Three tiny 3×3 images
- ☐ For each one: add the red numbers
- ☐ Take the total $\% 26$, then look up the letter

// watch how long it takes you. We need that number in a minute.

Oh no.

How long did a tiny 3×3 image take you?

3×3 : 9 red numbers

puzzle photo: $640 \times 640 = 409,600$

$\times 37$ photos = 15,155,200

Careful: Two Different Photos

`cow-demo.jpg`: 320 × 320, our practice photo

`cow_message/`: 37 photos, 640 × 640 each

// same code works on both. Practise on the demo, then solve the puzzle.

Let the Computer Do It

You already know how: functions and loops.

STEP 2: `total_red(path)` gives one number

STEP 3: `letter_for(path)` gives one letter

STEP 4: `decode_message()` gives the sentence

// in `image_starter.py`. Each one uses the one before. Test each before you move on.

Open `image_starter.py`

`#` STEP 1 is done for you. Run it: it prints `size`, `shape`, `first pixel`, then draws a colour.

// put it in the SAME folder as `cow-demo.jpg` and the `cow_message` folder

STEP 2: `total_red`

`total_red(path)` gives one number

// in `image_starter.py` . Test it: it should print 10218857

STEP 2: `total_red`

```
1 def total_red(path):  
2     img = Image.open(path)  
3     pixels = np.array(img)  
4     # TODO: get just the red channel  
5     # hint: pixels[:, :, 0] (every row, every column, channel 0 = red)  
6     # TODO: add up every red value in it  
7     # hint: .sum()  
8     # TODO: return that total  
9     return 0
```

```
10 print(total_red("cow-demo.jpg"))    # test: it should print 10218857
```

// prints 0? You forgot to change `return 0`.

STEP 3: letter_for

`letter_for(path)` gives one letter

// in `image_starter.py`. It uses `total_red`. Test it: it should print Z

STEP 3: letter_for

```
1 def letter_for(path):  
2     # TODO: 1. get the red total for this picture (you wrote a function for this!)  
3     #         2. take it % 26    (the remainder, a number from 0 to 25)  
4     #         3. use that number to pick a letter from ALPHABET  
5     #         hint: ALPHABET[2] is "C"  
6     #         4. return the letter  
7     return ""
```

```
8 print(letter_for("cow-demo.jpg"))    # test: it should print Z
```

// wrong letter? Test `total_red` again first. `ALPHABET` is already at the top of your file.

STEP 4: `decode_message`

`decode_message()` gives the sentence

// in `image_starter.py`. It uses `letter_for`, once for each of the 37 photos.

```
# Does it read like English? Then it worked.
```

A Function We Give You

```
1 def image_path(number):  
2     return "cow_message/" + str(number).zfill(2) + ".png"
```

```
>>> image_path(1)  
'cow_message/01.png'  
>>> image_path(12)  
'cow_message/12.png'
```

// number in, file name out. You do not need to understand the `zfill` line. Just use it.

Building a String: +=

```
>>> message = ""
>>> message += "C"
>>> message += "A"
>>> message
'CA'
```

```
1 message = ""
2 for letter in ["C", "A", "T"]:
3     message += letter
4 print(message)    # CAT
```

// start empty, stick one letter on the end each time. `range(1, 38)` counts 1 to 37: it stops before 38.

STEP 4: `decode_message`

```
1 def decode_message():  
2     message = "" # the message we are building. It starts empty.  
3     # TODO: for each number from 1 to 37 (hint: for number in range(1, 38))  
4     #     1. find the picture: image_path(number)  
5     #     2. work out its letter (you already wrote a function for this!)  
6     #     3. stick the letter on the end of message: message += letter  
7     return message
```

```
8 print(decode_message()) # remove the # in your file to run this
```

Does it read like **English**? Then it worked. Gibberish? Check STEP 2 and STEP 3.

Solve the Puzzle

- ❑ STEP 1: run `image_starter.py`
- ❑ STEP 2: finish `total_red` . Test: 10218857
- ❑ STEP 3: finish `letter_for` . Test: Z
- ❑ STEP 4: finish `decode_message` . Read the message aloud.
- ❑ Tracker: fill in **Worksheet 11: Message tracker** as you go

// the message should make sense in English: that is how you know it worked

Stuck? Use the Worksheets

Worksheet 08: Plan before you type

Worksheet 09: Fill in the functions

Worksheet 10: The starter code

Worksheet 11: Message tracker

// paper first: work it out, then type it. Random letters? Test `total_red`, then `letter_for`, again.

Stretch

- ❑ Add up the **green** (channel 1) or **blue** (channel 2) instead. Copy your functions, change the 0. What message do you get?
- ❑ Think: if one photo's red total went up by 1, which letter would come next? What if it went up by 26?

What You Learned Today

- ✓ a picture is a grid of numbers: rows × columns × 3
- ✓ one pixel, a crop, the channels, switching one off, grey, negative
- ✓ counting, averages and histograms
- ✓ red total `% 26` gives a letter: you solved a puzzle
- ✓ functions that call functions, and building a string in a loop

Think, Then Share

- ☐ How long did 3 tiny images take by hand? How long did 37 big ones take with Python?
- ☐ Which task was the hardest? Why?
- ☐ Challenge: try the green or blue channel on the puzzle. What happens?