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?
- That's today's problem — and it's the key to a challenge waiting for you at the end.

IMAGE ANALYSIS

Image Analysis Challenge

$ image_analysis.py

What a picture actually is, to a computer — and a hidden message to go find.

— Dr. Hughes

Quick Recap

`variables` `functions` `if / elif / else` `for & while loops`

> Everything so far, we wrote ourselves, from scratch.

Nobody Writes Everything Themselves

> # A **library** is just a bunch of functions someone else already wrote — ready for you to use.

Bringing One In: `import`

```
>>>  import math
>>>  math.sqrt(16)
   4.0
```

// math is built into Python — someone already wrote sqrt() for us

Only Need One Part?

```
>>>  from math import sqrt
>>>  sqrt(16)
   4.0
```

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

A Library for Pictures

> » **PIL** (short for *Pillow*) is a library for opening, editing, and reading images.

```
// it's not built in — but it's one line to bring in
```

Importing It

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

// we only need the Image part of the PIL library

Opening a Picture

```
>>>  img = Image.open("cow.jpg")
```

> One line. Now `img` holds the whole picture.

What Do We Actually Have?

```
>>>  img.size
   (640, 640)
>>>  img.mode
   'RGB'
```

// 640 pixels wide, 640 pixels tall, 3 colour values per pixel

```
// this is what img.show() would put on screen
```

What Is a Picture, Really?

> # To Python, a picture is a big grid of numbers — one small group of numbers per pixel.

Another Library: NumPy

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

// numpy is built for doing maths on huge grids of numbers, fast

Making the Grid

```
>>>  pixels = np.array(img)
>>>  pixels.shape
   (640, 640, 3)
```

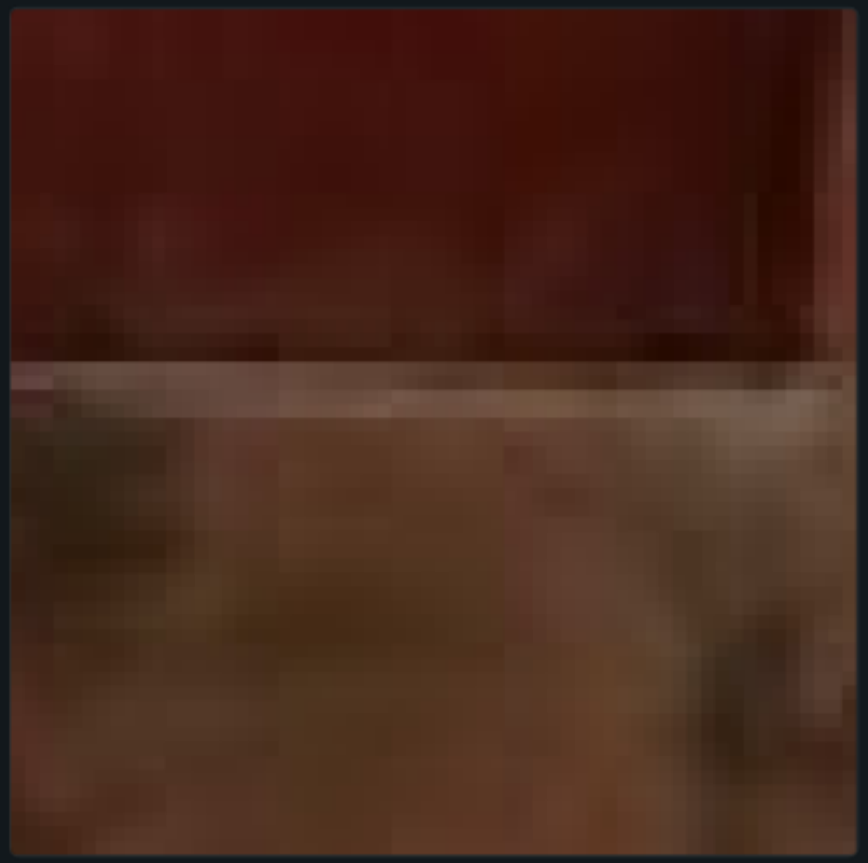
> height × width × 3 colour numbers per pixel.

409,600 Pixels

```
// 640 × 640
```

> Every single one of those pixels is just **three numbers**: red, green, blue.

Zooming Into One Tiny Corner



» This is 4 × 4 pixels — a tiny square from the wall behind the cow.

» Out of 409,600 pixels total.

» Let's look at the actual numbers behind it.

The Real Numbers

115, 86, 72	113, 84, 70	110, 81, 67	108, 79, 65
111, 79, 64	109, 80, 64	110, 78, 63	108, 79, 63
93, 60, 43	93, 62, 44	97, 64, 47	97, 66, 48
92, 59, 40	94, 61, 42	96, 63, 44	97, 64, 45

```
// straight out of pixels[100:104, 100:104] — every value 0-255
```

Neighbours Look Similar

> Real photos change gradually — next-door pixels are usually close in value, not random.

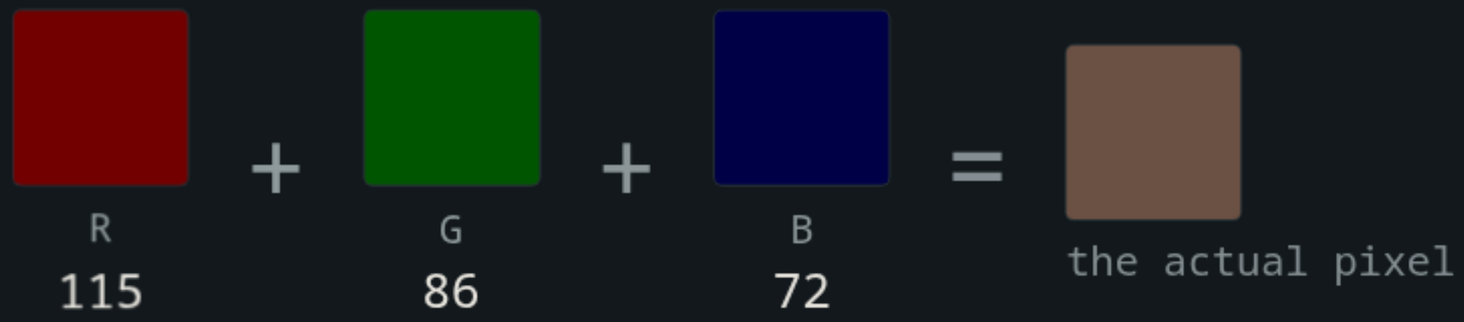
// that's WHY photo compression tricks (like run-length encoding) work so well

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).

Take One Pixel: (115, 86, 72)



// mix red light, green light and blue light — that's a screen pixel

More Red, Less Blue...

230, 30, 30 30, 200, 60 40, 60, 220 20, 20, 20 245, 245, 245

> Every colour you've ever seen on a screen is just three numbers like these.

The Same 16 Numbers, In Colour

115,86,72	113,84,70	110,81,67	108,79,65
111,79,64	109,80,64	110,78,63	108,79,63
93,60,43	93,62,44	97,64,47	97,66,48
92,59,40	94,61,42	96,63,44	97,64,45

> Change the numbers, change the picture. That's *all* a photo

It's Just Numbers Now

» Once a picture is a grid of numbers, Python can ask maths questions about the **whole thing at once**.

Grabbing One Colour Channel

```
>>>  red_values = pixels[:, :, 0]
>>>  red_values.shape
   (640, 640)
```

// [:, :, 0] means "every row, every column, just channel 0 — red"

Adding Them All Up

```
>>>  red_values.sum()
    40862226
```

// one number: the total "redness" of all 409,600 pixels combined

Counting, Not Just Summing

```
>>>  (red_values < 50).sum()
   82046
```

> 82,046 pixels have a red value under 50 — out of 409,600.

Remember `%`?

```
// the leftover / remainder operator, from earlier
```

```
>>>  40862226 % 26
   2
```

// any huge number, squashed down to something between 0 and 25

0-25 Looks a Lot Like A-Z

```
>>>  alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
>>>  alphabet[2]
   'C'
```

> A picture's red total → one letter. That's the whole trick.

A Different Question

> # Not "what's the total?" — instead: **how many pixels are dark, how many are bright, how many are in between?**

What a Histogram Is

> Split 0-255 into buckets. Count how many pixels land in each one.

// e.g. bucket "0-31", bucket "32-63", bucket "64-95" ... up to "224-255"

Our Cow Photo's Histogram

// tall bars on the LEFT = a mostly dark photo. Makes sense — it's a black cow.

One Line, With a Library

```
1 import matplotlib.pyplot as plt
2 plt.hist(red_values.flatten(), bins=8)
```

> Every bar you just watched appear — that's what this one line draws.

$ cow_message/

The Cow Message Challenge

37 images. One hidden letter each. One secret sentence.

The Rule

1. Open an image.
2. Turn it into a NumPy array.
3. Add up every **red** value in the whole image.
4. Take that total `% 26`.
5. Use the result to look up a letter — A=0, B=1, ... Z=25.

> Do it for every image, in order — the letters spell something out.

You Already Know Every Piece

`import / from...import` `Image.open()` `np.array()` `[:, :, 0]`

`.sum()` `%` `a for loop`

// nothing new left — just putting the pieces together

Your Turn

> » Open the `cow_message` folder. 37 images, named `01.png` to `37.png`. Decode them **in order**.

// write your own version first — check it against cow_decoder.py only once you're stuck

What You Learned Today

- ✓ importing libraries (PIL, NumPy)
- ✓ opening an image with Image.open()
- ✓ a picture is a NumPy array — height × width × 3
- ✓ how RGB encodes colour, 0-255 per channel
- ✓ summing and counting pixel values
- ✓ histograms — counting values into buckets

Now Go Decode Some Cows

> Every skill you need is one you already have.

» Challenge: once you've cracked it, try the same trick on the **green** or **blue** channel instead — what happens?