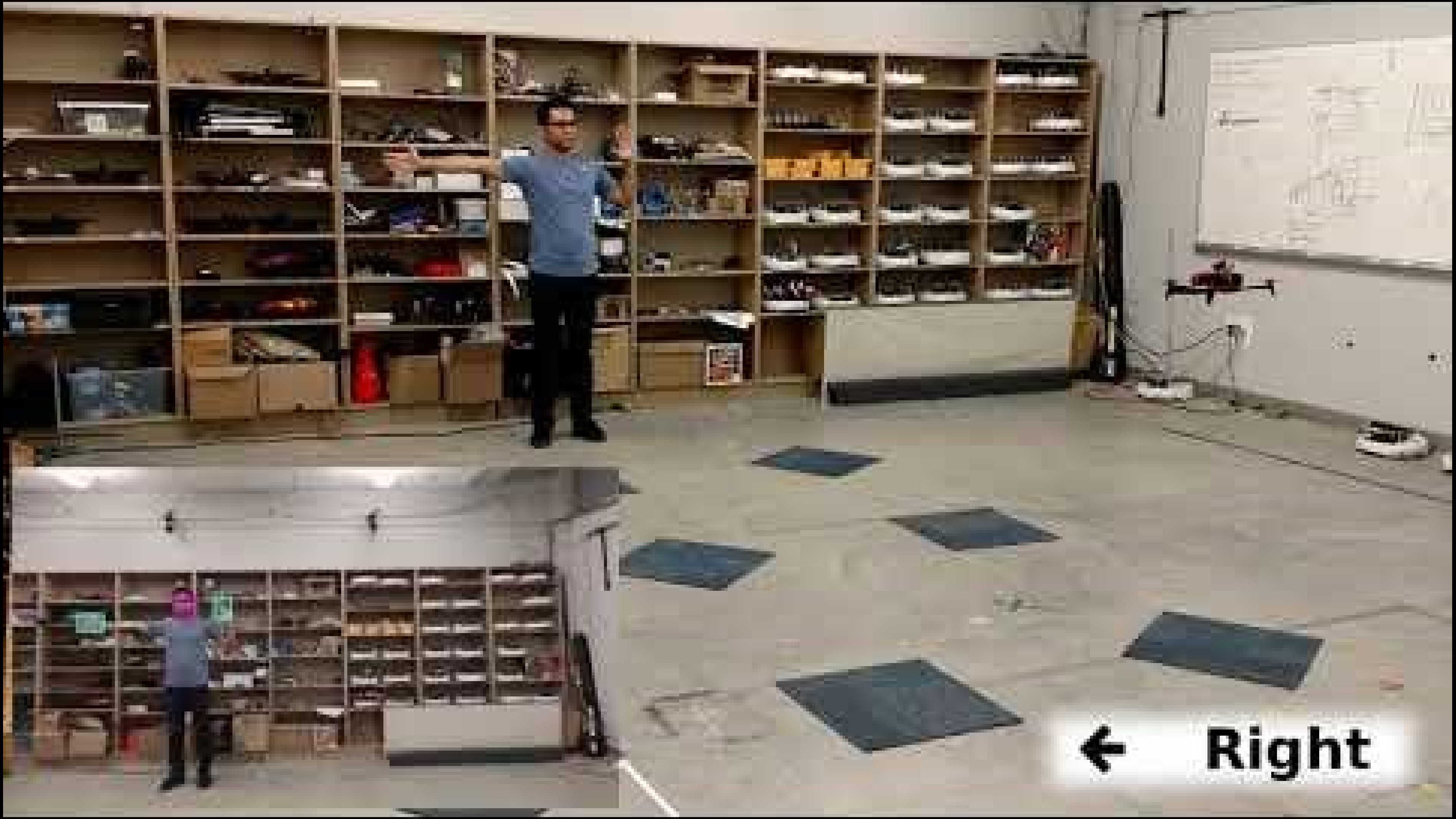
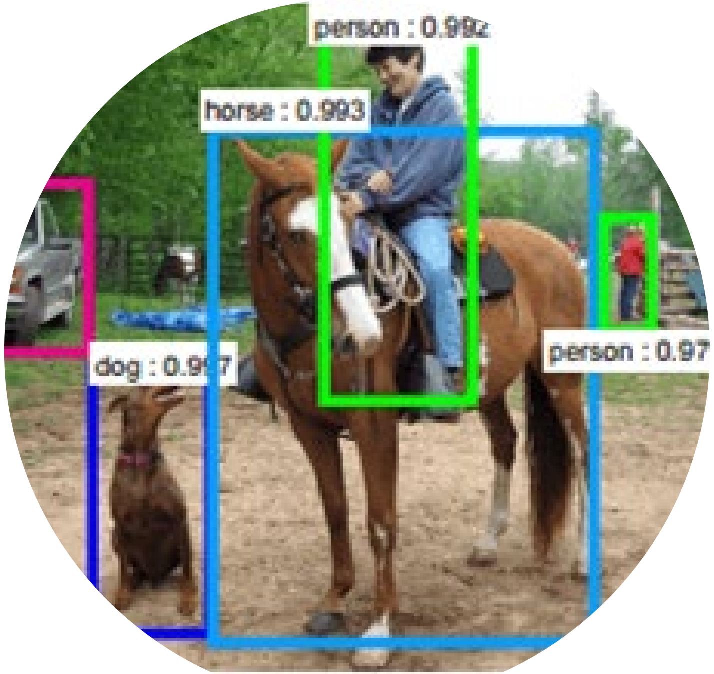
Machines that can see


← Right

Images and videos

We can make **machines** understand our world **visually**

- Facebook's automatic photo captions for the blind
- iPhone X recognizes faces
- Instagram filters
- Medical imaging
- Gesture recognition
- SFU's Visual Computing M.Sc.



Unit 5 Computer Vision

1/APPLICATIONS

In this unit, we'll learn about the computing science field of **computer vision**, that allows machines to **process** images and **understand** them.

2/ALGORITHMS

We'll learn about more advanced functions, tuples, embedded loops and more.

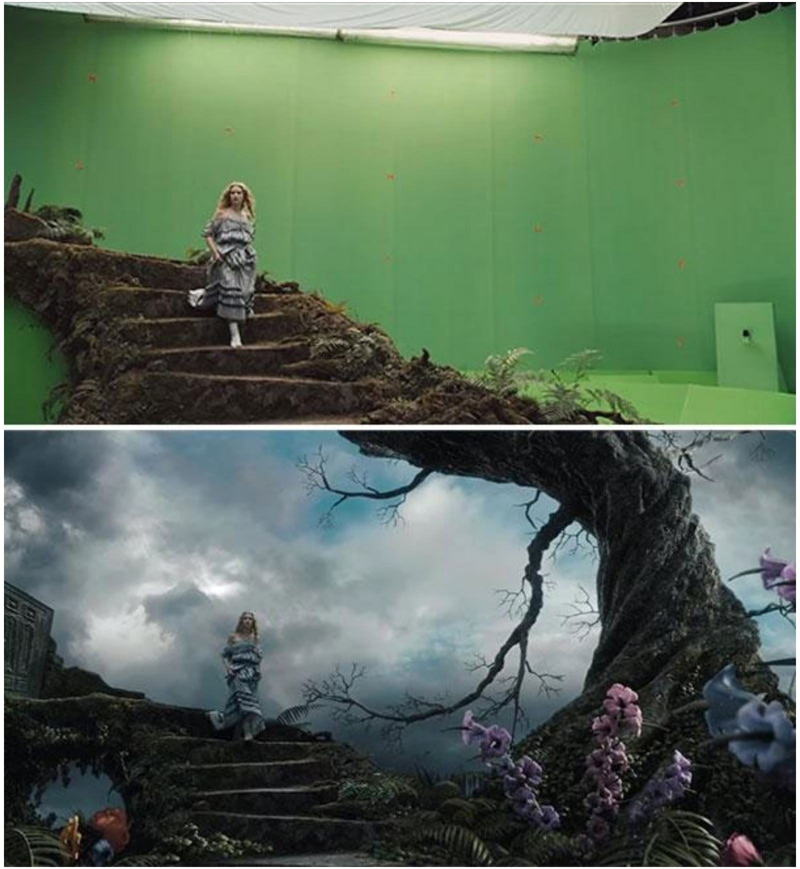
3/PROGRAMMING LANGUAGE

In Python 3, we'll be learning the syntax and keywords to implement our algorithms.

4/DOCUMENTATION AND TESTING

We will show how to tell if our program is any good or not!

Image Processing



This week

We're going to make a program that can merge a green screen image with another background, using the following concepts:

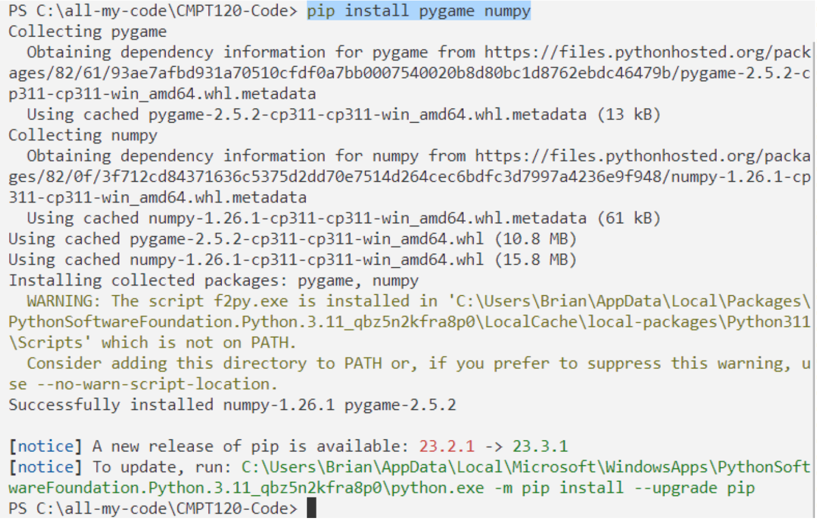
- Colours in RGB space
- Pixel representations of images in 2D arrays, e.g. **image[row][col]**
- Review functions that **return** values
- Looking up and understanding module documentation

Working with images

- Install the Pygame and Numpy module
  - In VS Code, go to Terminal > New Terminal
  - Install modules:
    `pip install pygame numpy`

Pygame module has some image processing modules we are going to use.

Note: Runestone code doesn't work in replit.com, so the image processing syntax from the readings will differ here.



```
PS C:\all-my-code\CMPT120-Code> pip install pygame numpy
Collecting pygame
  Obtaining dependency information for pygame from https://files.pythonhosted.org/pack
ages/82/61/93ae7afbd931a70510cfdf0a7bb0007540020b8d80bc1d8762ebdc46479b/pygame-2.5.2-c
p311-cp311-win_amd64.whl.metadata
  Using cached pygame-2.5.2-cp311-cp311-win_amd64.whl.metadata (13 kB)
Collecting numpy
  Obtaining dependency information for numpy from https://files.pythonhosted.org/packa
ges/82/0f/3f712cd84371636c5375d2dd70e7514d264cec6bdfc3d7997a4236e9f948/numpy-1.26.1-cp
311-cp311-win_amd64.whl.metadata
  Using cached numpy-1.26.1-cp311-cp311-win_amd64.whl.metadata (61 kB)
Using cached pygame-2.5.2-cp311-cp311-win_amd64.whl (10.8 MB)
Using cached numpy-1.26.1-cp311-cp311-win_amd64.whl (15.8 MB)
Installing collected packages: pygame, numpy
  WARNING: The script f2py.exe is installed in 'C:\Users\Brian\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311
\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, u
se --no-warn-script-location.
Successfully installed numpy-1.26.1 pygame-2.5.2

[notice] A new release of pip is available: 23.2.1 -> 23.3.1
[notice] To update, run: C:\Users\Brian\AppData\Local\Microsoft\WindowsApps\PythonSoft
wareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip
PS C:\all-my-code\CMPT120-Code> █
```

If you have troubles with this (possible "TK version" problem), then see last point on **Resources** page of website.

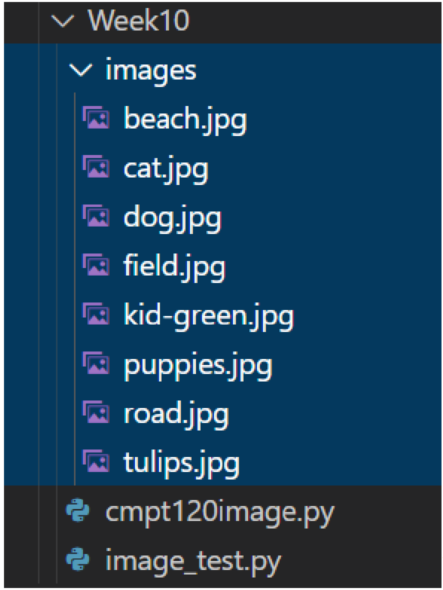
Download now:

Save files into your VS Code folder for this week.
Put all .jpg images into a "**images**"

- **Notes**
  - Starting Code and Files for Images
    - `cmpt120image.py` : Image manipulation module (custom).
    - **Images**: download ZIP file of all images, or download one at a time in this folder of images.
    - **Code**: sample Python code for working with images (shown in notes)
  - If you have troubles installing `pygame` and `numpy` using the command

    ```
    pip install pygame numpy
    ```

    (perhaps a "TK version" error), then see the last point in the resources page.

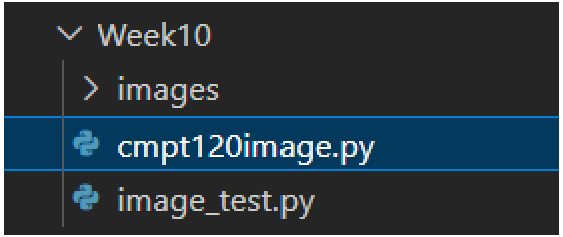


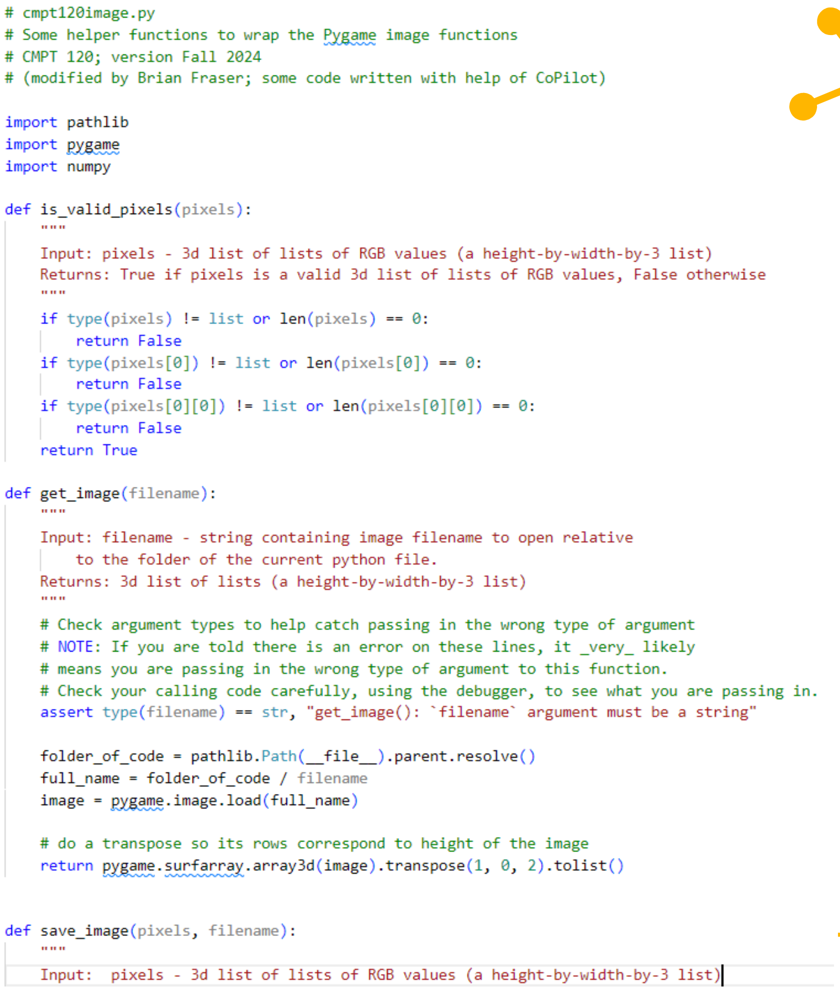
```
Week10
  images
    beach.jpg
    cat.jpg
    dog.jpg
    field.jpg
    kid-green.jpg
    puppies.jpg
    road.jpg
    tulips.jpg
  cmpt120image.py
  image_test.py
```

Cmpt120image.py

Loads, shows, and saves images

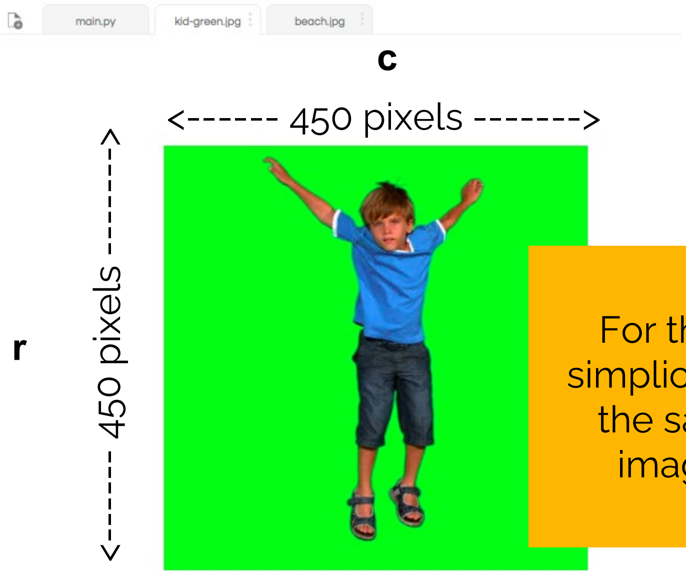
Save this into your folder for this week inside VS Code

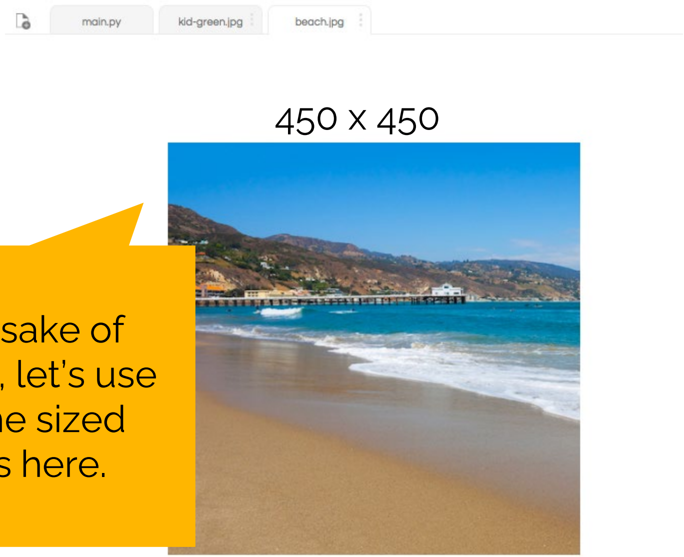




```
# cmpt120image.py
# Some helper functions to wrap the Pygame image functions
# CMPT 120; version Fall 2024
# (modified by Brian Fraser; some code written with help of CoPilot)

import pathlib
import pygame
import numpy

def is_valid_pixels(pixels):
    """
    Input: pixels - 3d list of lists of RGB values (a height-by-width-by-3 list)
    Returns: True if pixels is a valid 3d list of lists of RGB values, False otherwise
    """
    if type(pixels) != list or len(pixels) == 0:
        return False
    if type(pixels[0]) != list or len(pixels[0]) == 0:
        return False
    if type(pixels[0][0]) != list or len(pixels[0][0]) == 0:
        return False
    return True

def get_image(filename):
    """
    Input: filename - string containing image filename to open relative
        to the folder of the current python file.
    Returns: 3d list of lists (a height-by-width-by-3 list)
    """
    # Check argument types to help catch passing in the wrong type of argument
    # NOTE: If you are told there is an error on these lines, it _very_ likely
    # means you are passing in the wrong type of argument to this function.
    # Check your calling code carefully, using the debugger, to see what you are passing in.
    assert type(filename) == str, "get_image(): `filename` argument must be a string"

    folder_of_code = pathlib.Path(__file__).parent.resolve()
    full_name = folder_of_code / filename
    image = pygame.image.load(full_name)

    # do a transpose so its rows correspond to height of the image
    return pygame.surfarray.array3d(image).transpose(1, 0, 2).tolist()


def save_image(pixels, filename):
    """
    Input:  pixels - 3d list of lists of RGB values (a height-by-width-by-3 list)
```

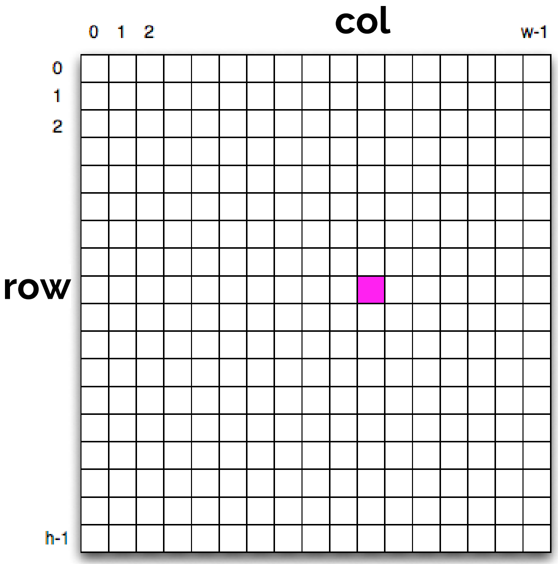
Working with images





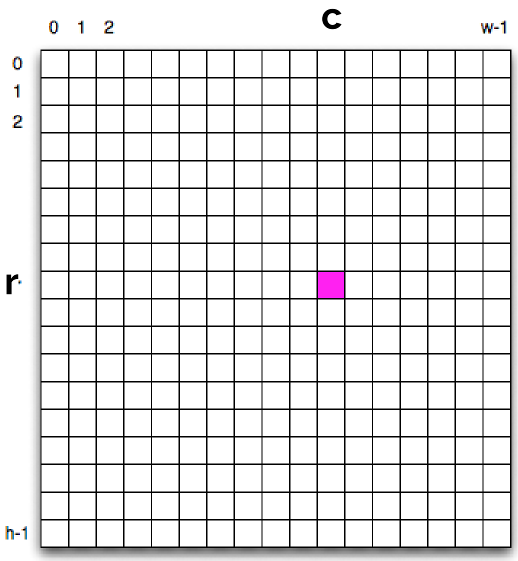
For the sake of simplicity, let's use the same sized images here.

Image primer

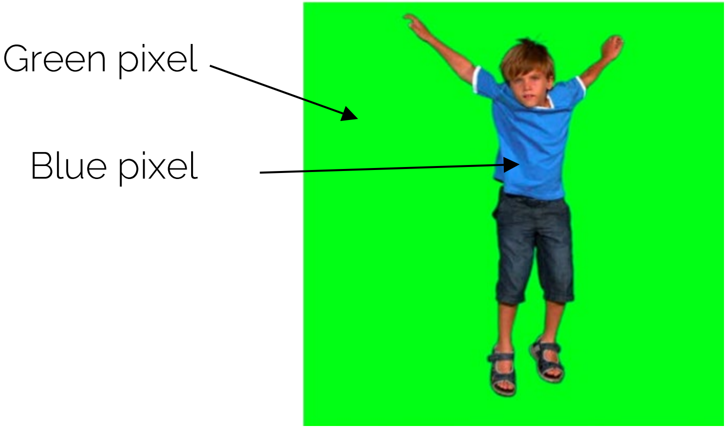


Images are represented by a 2-dimensional matrix or **array** of pixels.

Image primer

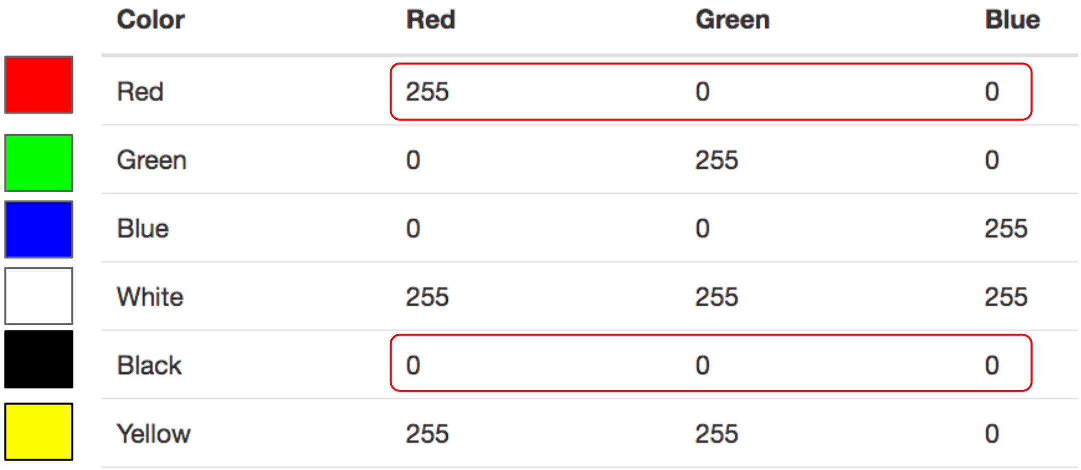


Images are represented by a 2-dimensional matrix or **array** of pixels. Each **pixel** is represented by a **colour**. A colour is represented by **3 values (RGB)**.


Green pixel

Blue pixel

Colors as Red, Green, Blue (RGB) values



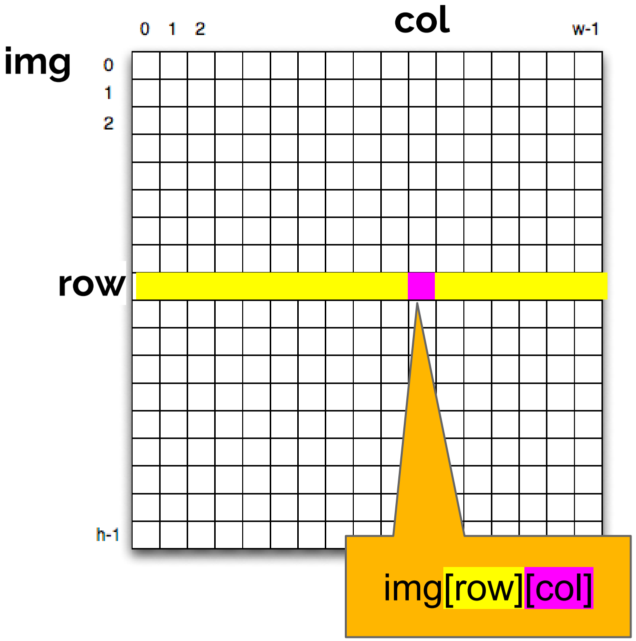
	Color	Red	Green	Blue
🟥	Red	255	0	0
🟩	Green	0	255	0
🟦	Blue	0	0	255
⬜	White	255	255	255
⬛	Black	0	0	0
🟨	Yellow	255	255	0

A red pixel
`[255,0,0]`

A black pixel
`[0,0,0]`

Try playing around with RGB values here:
https://www.w3schools.com/colors/colors_rgb.asp

Image primer



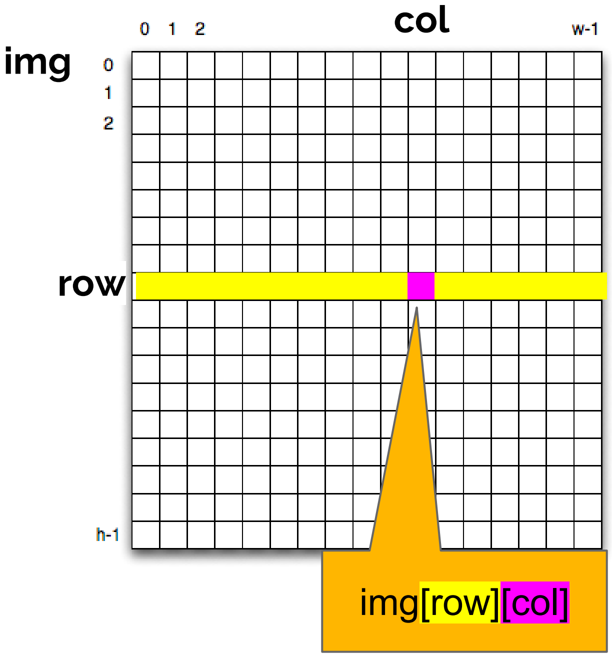
In this class, we will use an array representation of an image that looks like:

```
[
      [ [ ], [ ], …, [ ] ],
      [ [ ], [ ], …, [ ] ],
      ...
      [ [ ], [ ], …, [ ] ],
      ...
      [ [ ], [ ], …, [ ] ]
]
```

img[0] is the first row

img[row]

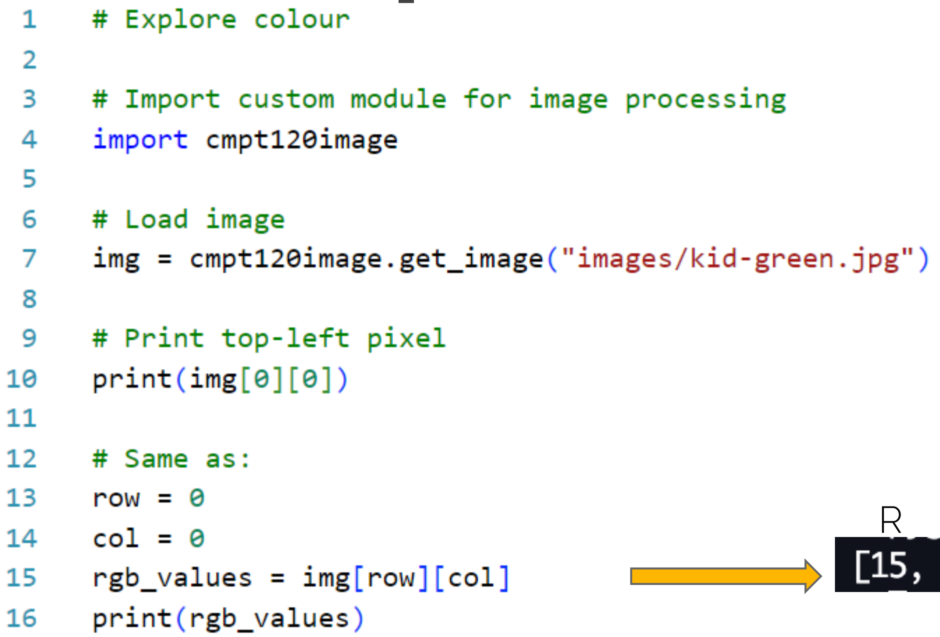
Image primer



How do we calculate the **height** of the image?
→ Length of the whole img list, e.g. `len(img)`
How do we calculate the **width** of the image?
→ Length of one row, e.g. `len(img[0])`

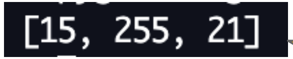
```
[
     [ [ ], [ ], …, [ ] ],
     [ [ ], [ ], …, [ ] ],
     ...
     [ [ ], [ ], …, [ ] ],
     ...
     [ [ ], [ ], …, [ ] ]
]
```

Let's explore an image

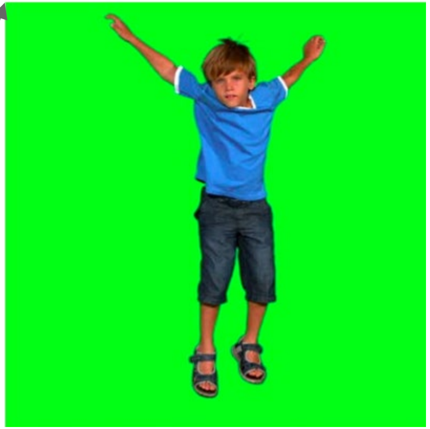


```python
# Explore colour

# Import custom module for image processing
import cmpt120image

# Load image
img = cmpt120image.get_image("images/kid-green.jpg")

# Print top-left pixel
print(img[0][0])

# Same as:
row = 0
col = 0
rgb_values = img[row][col]
print(rgb_values)
```

R G B



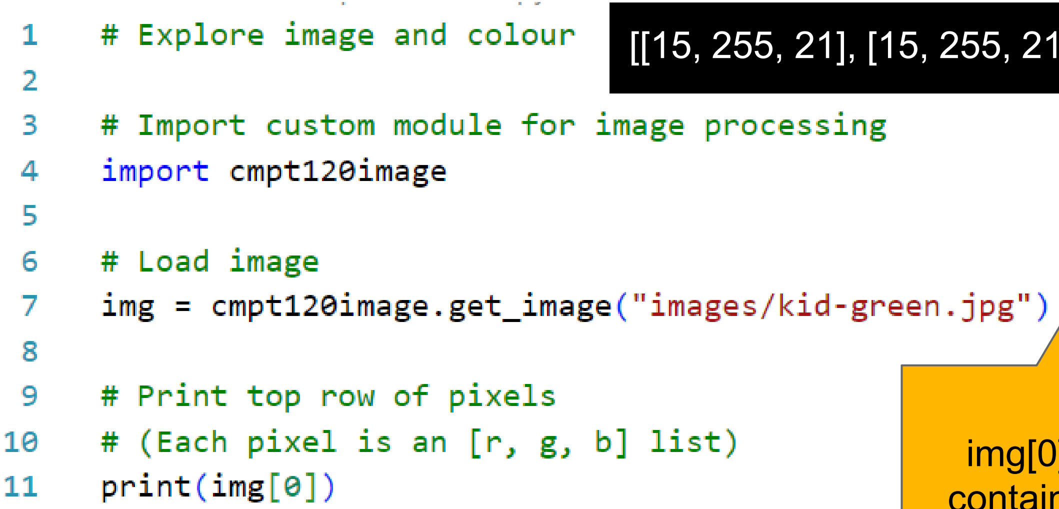
```
[15, 255, 21]
```



Interesting! Not exactly 0 for R & B… we'll come back to this later.

Try playing around with RGB values here:
https://www.w3schools.com/colors/colors_rgb.asp

Let's explore an image



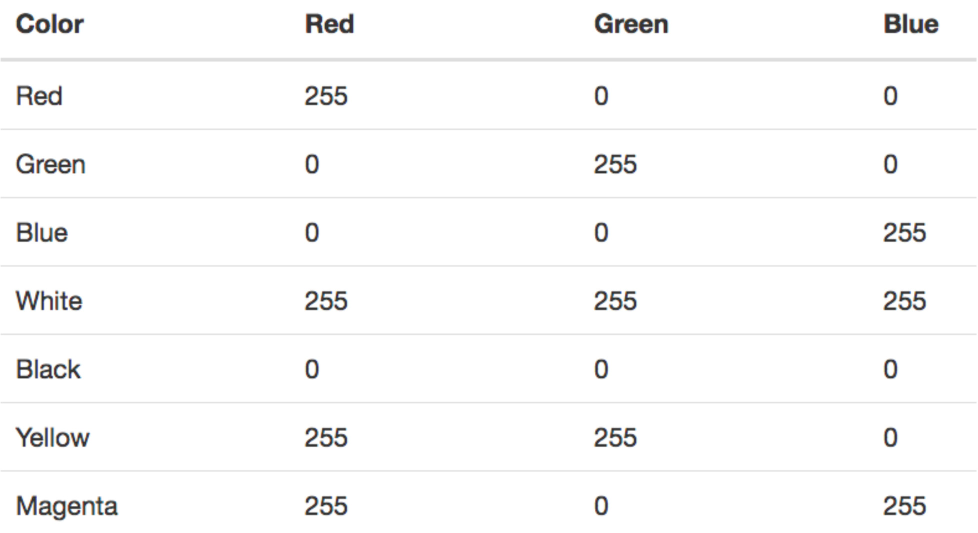
```python
# Explore image and colour

# Import custom module for image processing
import cmpt120image

# Load image
img = cmpt120image.get_image("images/kid-green.jpg")

# Print top row of pixels
# (Each pixel is an [r, g, b] list)
print(img[0])
```

```
[[15, 255, 21], [15, 255, 21], [15, 255, 21],... ]
```

img[0] contains the first row of pixels!

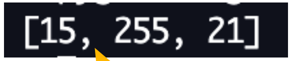
Try playing around with RGB values here:
https://www.w3schools.com/colors/colors_rgb.asp

Is this pixel green?



Color	Red	Green	Blue
Red	255	0	0
Green	0	255	0
Blue	0	0	255
White	255	255	255
Black	0	0	0
Yellow	255	255	0
Magenta	255	0	255

Note: 255 is the max here. We'll learn where this number comes from next week.

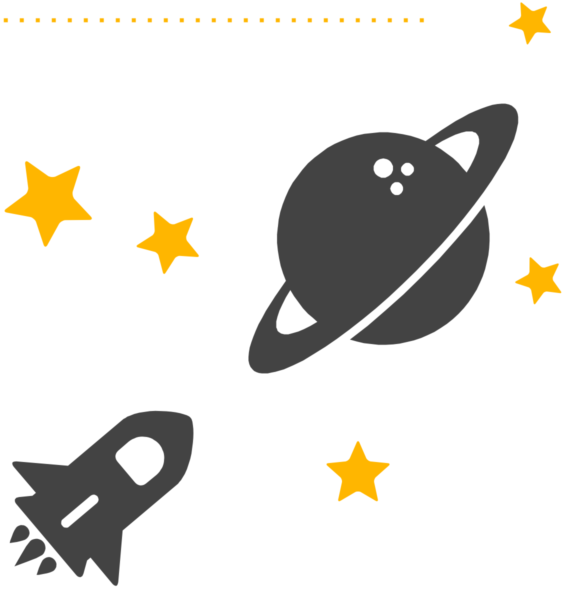


```
[15, 255, 21]
```

Do the values here make sense?

Try playing around with RGB values here: https://www.w3schools.com/colors/colors_rgb.asp

Functions



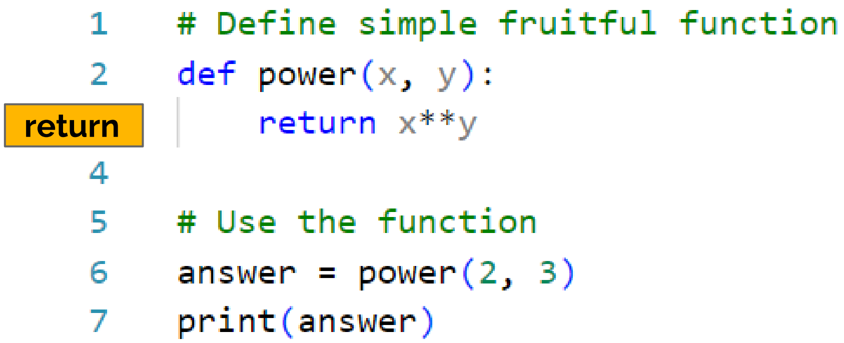
Functions that **return** something

Fruitful functions review

Now you can define your own fruitful function! Just use the keyword **return** in your function definition.

This does 2 things:

- Return a value
- Exit the function immediately



return

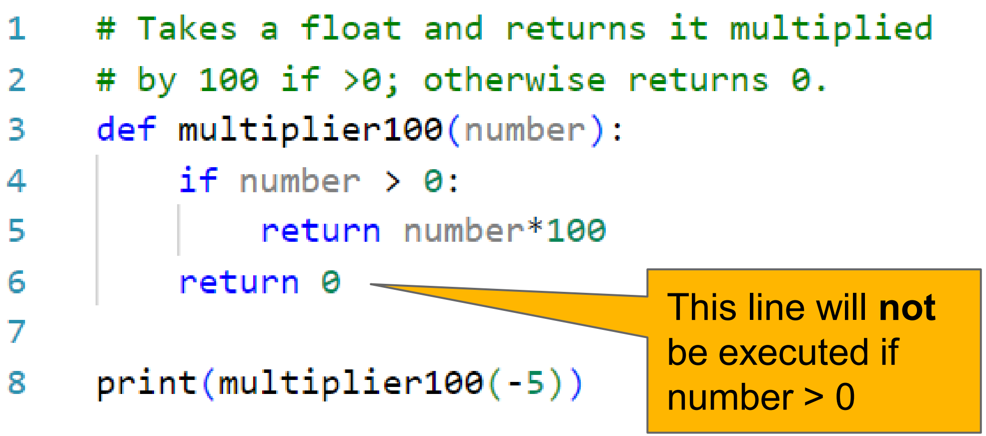
```python
# Define simple fruitful function
def power(x, y):
    return x**y

# Use the function
answer = power(2, 3)
print(answer)
```

http://interactivepython.org/runestone/static/thinkcspy/Functions/Functionsthatreturnvalues.html

Fruitful functions

In the case to the right, what will be printed if the argument to **multiplier100** is **5**?

What if the argument is **-5**?



```
# Takes a float and returns it multiplied
# by 100 if >0; otherwise returns 0.
def multiplier100(number):
    if number > 0:
        return number*100
    return 0

print(multiplier100(-5))
```

This line will **not** be executed if number > 0

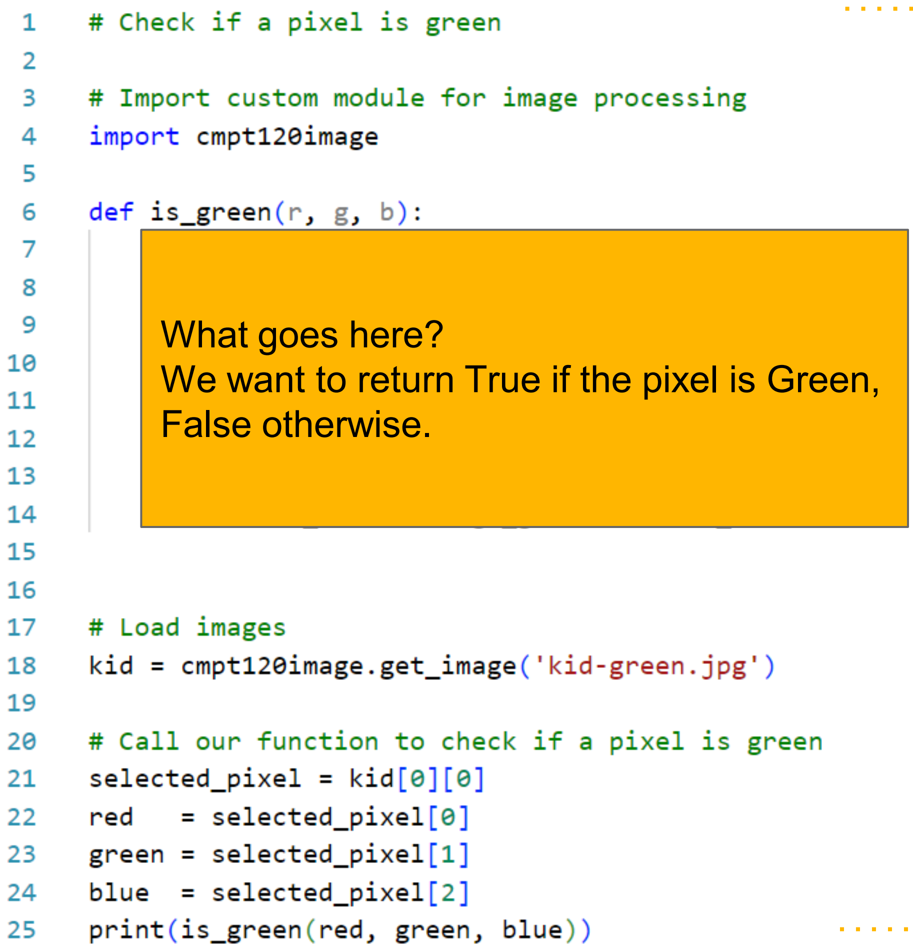
http://interactivepython.org/runestone/static/thinkcspy/Functions/Functionsthatreturnvalues.html

Fruitful functions!

We could **return** *True* or *False* depending if the pixel is green.

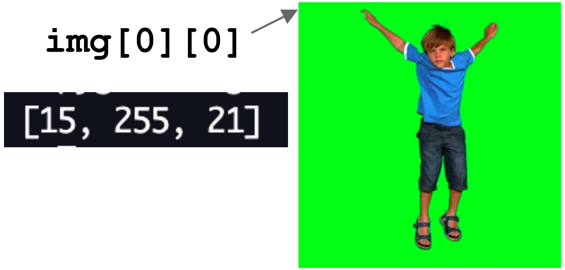
http://interactivepython.org/runestone/static/thinkcspy/Functions/Functionsthatreturnvalues.html

Checking a pixel colour



```
# Check if a pixel is green

# Import custom module for image processing
import cmpt120image

def is_green(r, g, b):



# Load images
kid = cmpt120image.get_image('kid-green.jpg')

# Call our function to check if a pixel is green
selected_pixel = kid[0][0]
red   = selected_pixel[0]
green = selected_pixel[1]
blue  = selected_pixel[2]
print(is_green(red, green, blue))
```

What goes here?
We want to return True if the pixel is Green,
False otherwise.

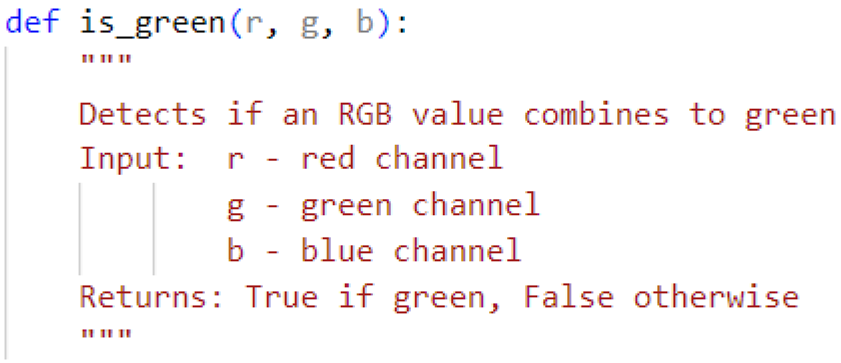


```
[15, 255, 21]
```

Try it!

Complete the **is_green** function in Python.

It should return *True* if the red, green and blue channels are all **within 30** of 0, 255, and 0 respectively.

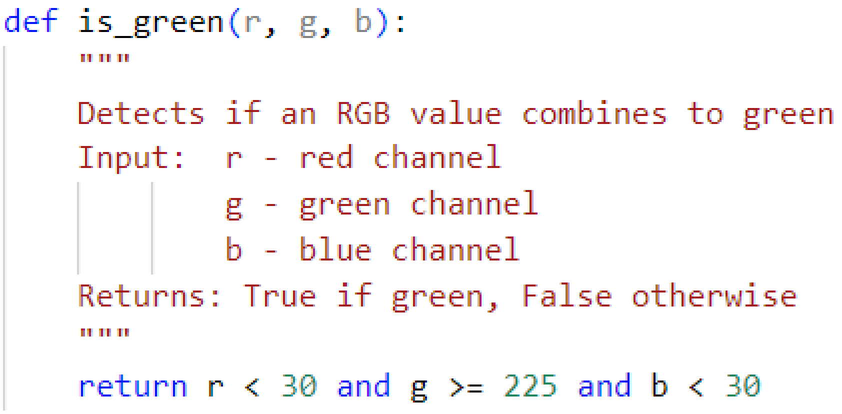


```
def is_green(r, g, b):
    """
    Detects if an RGB value combines to green
    Input:  r - red channel
            g - green channel
            b - blue channel
    Returns: True if green, False otherwise
    """
```

Try it!

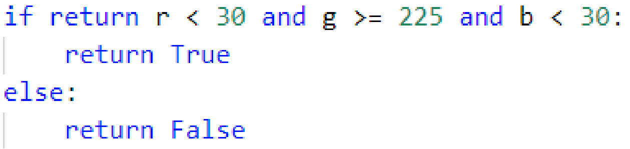
Complete the **is_green** function in Python.

It should return *True* if the red, green and blue channels are all **within 30** of 0, 255, and 0 respectively.



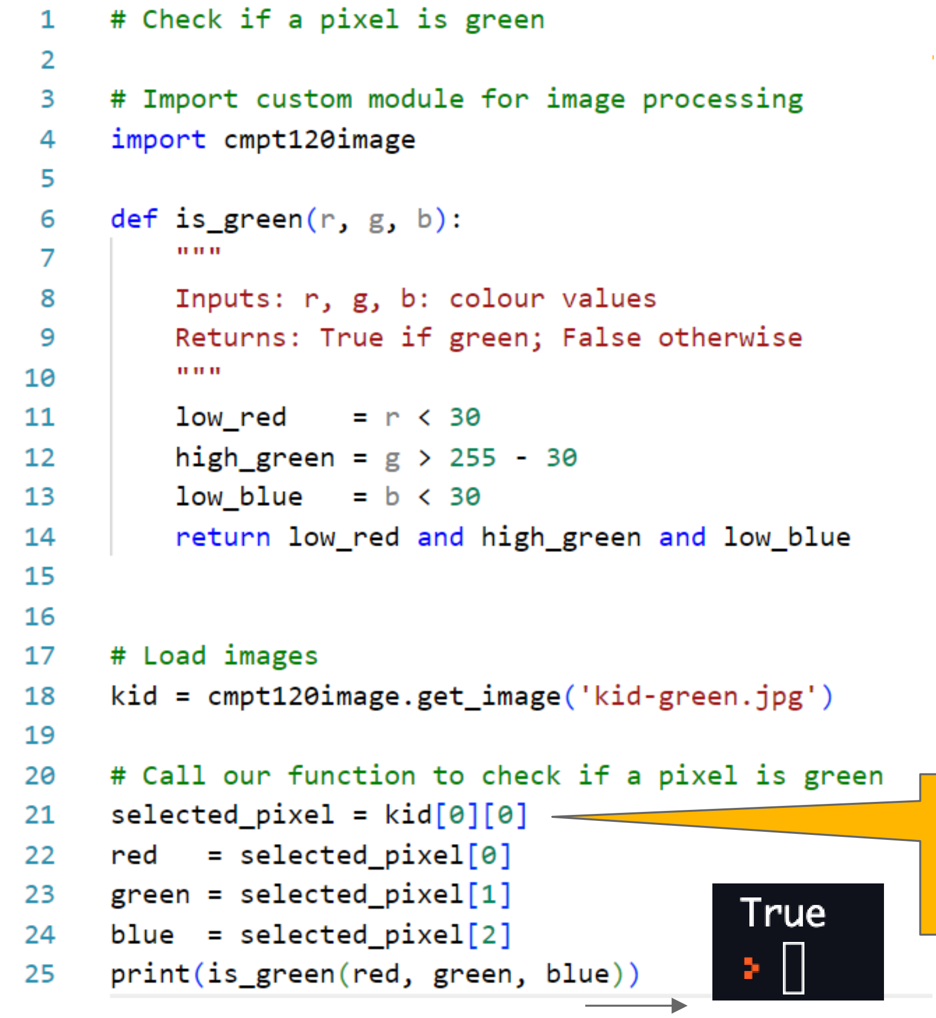
```python
def is_green(r, g, b):
    """
    Detects if an RGB value combines to green
    Input:  r - red channel
            g - green channel
            b - blue channel
    Returns: True if green, False otherwise
    """
    return r < 30 and g >= 225 and b < 30
```

Same meaning, the above is preferred style!



```python
if return r < 30 and g >= 225 and b < 30:
    return True
else:
    return False
```

Checking a pixel colour

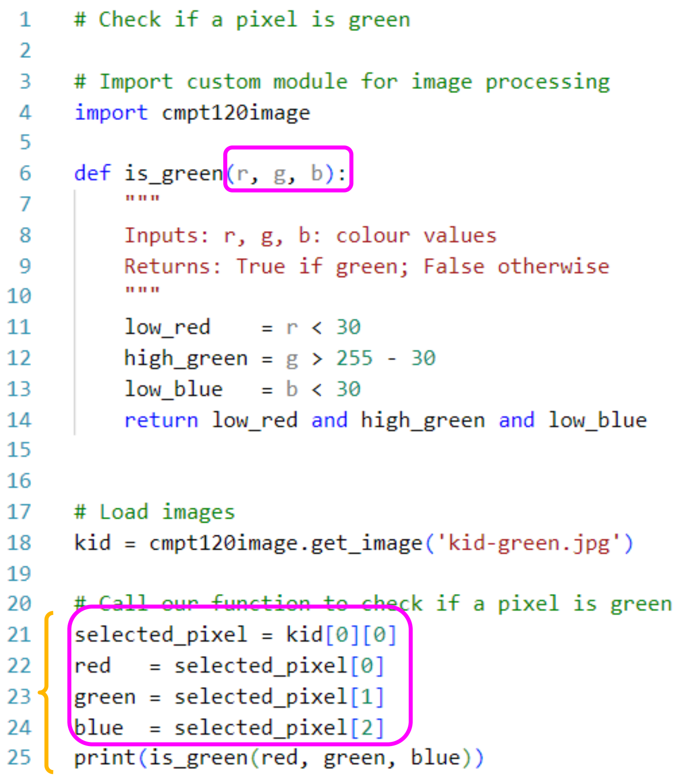


```
1   # Check if a pixel is green
2
3   # Import custom module for image processing
4   import cmpt120image
5
6   def is_green(r, g, b):
7       """
8       Inputs: r, g, b: colour values
9       Returns: True if green; False otherwise
10      """
11      low_red    = r < 30
12      high_green = g > 255 - 30
13      low_blue   = b < 30
14      return low_red and high_green and low_blue
15
16
17  # Load images
18  kid = cmpt120image.get_image('kid-green.jpg')
19
20  # Call our function to check if a pixel is green
21  selected_pixel = kid[0][0]
22  red   = selected_pixel[0]
23  green = selected_pixel[1]
24  blue  = selected_pixel[2]
25  print(is_green(red, green, blue))
```

```
True
>
```

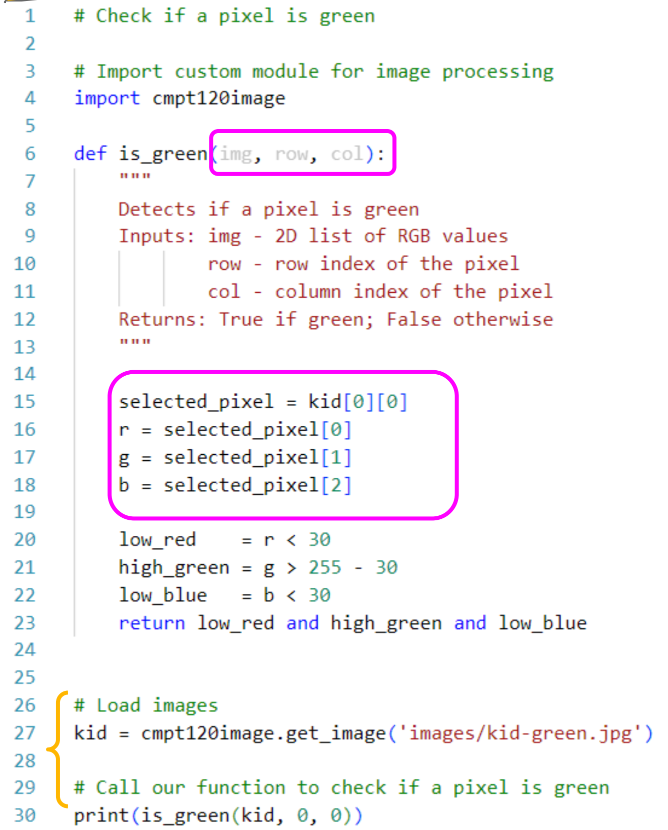
img[0][0]

How might you move lines 21-24 into the is_green() function?



```python
# Check if a pixel is green

# Import custom module for image processing
import cmpt120image

def is_green(r, g, b):
    """
    Inputs: r, g, b: colour values
    Returns: True if green; False otherwise
    """
    low_red    = r < 30
    high_green = g > 255 - 30
    low_blue   = b < 30
    return low_red and high_green and low_blue


# Load images
kid = cmpt120image.get_image('kid-green.jpg')

# Call our function to check if a pixel is green
selected_pixel = kid[0][0]
red   = selected_pixel[0]
green = selected_pixel[1]
blue  = selected_pixel[2]
print(is_green(red, green, blue))
```

You can define functions as you like!



```python
# Check if a pixel is green

# Import custom module for image processing
import cmpt120image

def is_green(img, row, col):
    """
    Detects if a pixel is green
    Inputs: img - 2D list of RGB values
            row - row index of the pixel
            col - column index of the pixel
    Returns: True if green; False otherwise
    """

    selected_pixel = kid[0][0]
    r = selected_pixel[0]
    g = selected_pixel[1]
    b = selected_pixel[2]

    low_red    = r < 30
    high_green = g > 255 - 30
    low_blue   = b < 30
    return low_red and high_green and low_blue


# Load images
kid = cmpt120image.get_image('images/kid-green.jpg')

# Call our function to check if a pixel is green
print(is_green(kid, 0, 0))
```

This formulation makes the main part of your program nice and short.

Let’s review some concepts

What can computer vision be used for?

In an image contained in a 2D array called **awesome_image,** how would you access the pixel located 5 pixels down, and 8 pixels in from the top left?

What would the code below output?

```
import random

def random_animal(animals):
   default = "cat"
   animal = random.choice(animals)
   return animal

pet = random_animal(["dog", "bird"])
print(pet)
print(default)
```