# COMP 204

## Introduction to image analysis with scikit-image (part one)

Mathieu Blanchette, based on slides from Christopher J.F. Cameron and Carlos G. Oliver

# Until the end of the semester...

We will learn how to use Python modules that are very useful in life science applications:

- Scikit-image: Analysis of images (3 lectures)
- BioPython: Bioinformatics applications (2 lectures)
- Scikit-learn: Machine learning (2-3 lectures)

Our goal is not to learn everything about those packages (they contain hundreds of functions), but to learn the key ideas about them, and let you more easily use them in the future.

# Image processing and analysis in Python

Goal: Process and analyze digital images.

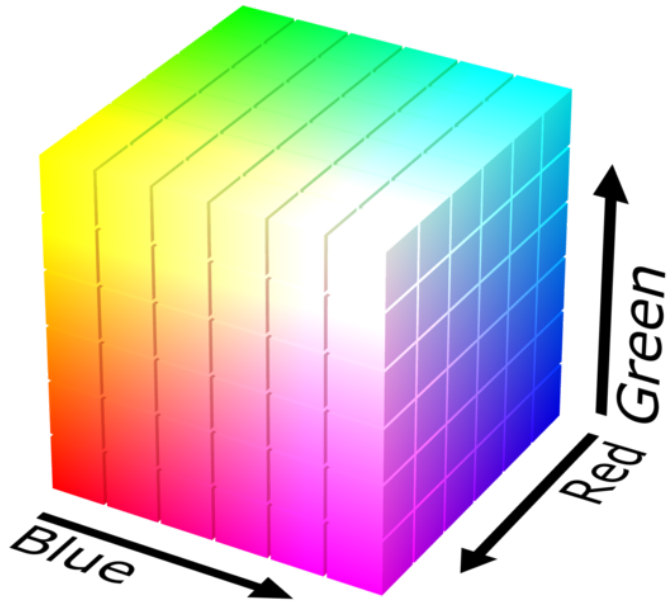
- Very useful for processing microscopy images, medical imaging, etc.
- Closely linked with machine learning for image analysis

**scikit-image module or (skimage)**

- image processing module in Python
- holds a wide library of image processing algorithms: filters, transforms, point detection, etc.
- Documentation (Application Program Interface - API)
  - `http://scikit-image.org/docs/dev/api/api.html`

# RGB Colors

The RGB color cube (source: Wikipedia)




Each pixel's color is represented using 3 integers, each between 0 and 255 (inclusively): (R, G, B), where R= red intensity, G = green intensity, B = blue intensity. All colors can be expressed as RGB:

- black = (0,0,0)
- white = (255,255,255)
- red = (255,0,0)
- purple = (255,0,255)
- dark purple = (120,0,120)
- yellow = (255, 255, 0)

# Our mascot for today

# Reading an image into memory

Skimage's **io** submodule allows reading images into memory and writing them out to file.

API: `http://scikit-image.org/docs/dev/api/skimage.io.html`

- `image = io.imread(filename)` reads the image stored in filename
- `io.imsave(filename, image)` saves image to filename

read_write.py program:

```
1  import skimage.io as io
2  import matplotlib.pyplot as plt
3
4  # read image into memory
5  image = io.imread("monkey.jpg")
6
7  # show the image on screen
8  plt.imshow(image)
9
10 # write image to disk
11 io.imsave("monkey_copy.jpg",image)
```

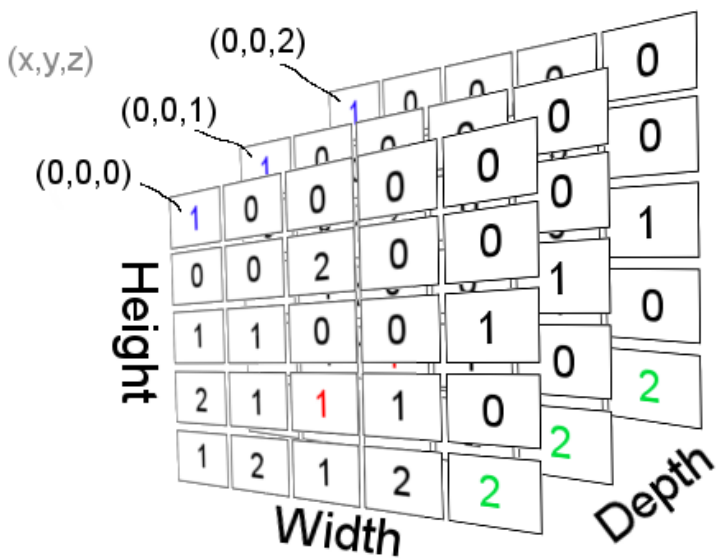
# What's an image in Python?

An image is stored as a NumPy ndarray (n-dimensional array).

- ndarrays are easier and more efficient than using 2-dimensional lists as we've seen before.

A color image with $R$ rows and $C$ columns is

- represented as a 3-dimensional ndarray of dimensions $R \times C \times 3$
- element at position $(i, j)$ of the array corresponds to the pixel's RGB value at row $i$ and column $j$ of the image
- each pixel is represented by 3 numbers, each between 0 and 255: Red, Green, Blue



# NumPy's ndarray

When loading an image with `image=io.imread("monkey.jpg")`, you get a object of type ndarray, which contains the pixel values of the entire image.
Things to know about ndarrays:

- Get their dimensions:
  ```
  n_row, n_col, n_colours = image.shape
  ```
- Get a particular element at row $r$, column $c$, and depth $d$
  ```
  value = image[r,c,d]
  ```
- Get an RGB pixel value at row $r$ and column $c$:
  ```
  pixel_RBG = image[r,c]
  ```
- Change the color at row $r$ and column $c$:
  ```
  image[r,c] = [120,134, 231]
  ```

# Playing with an image - modify.py

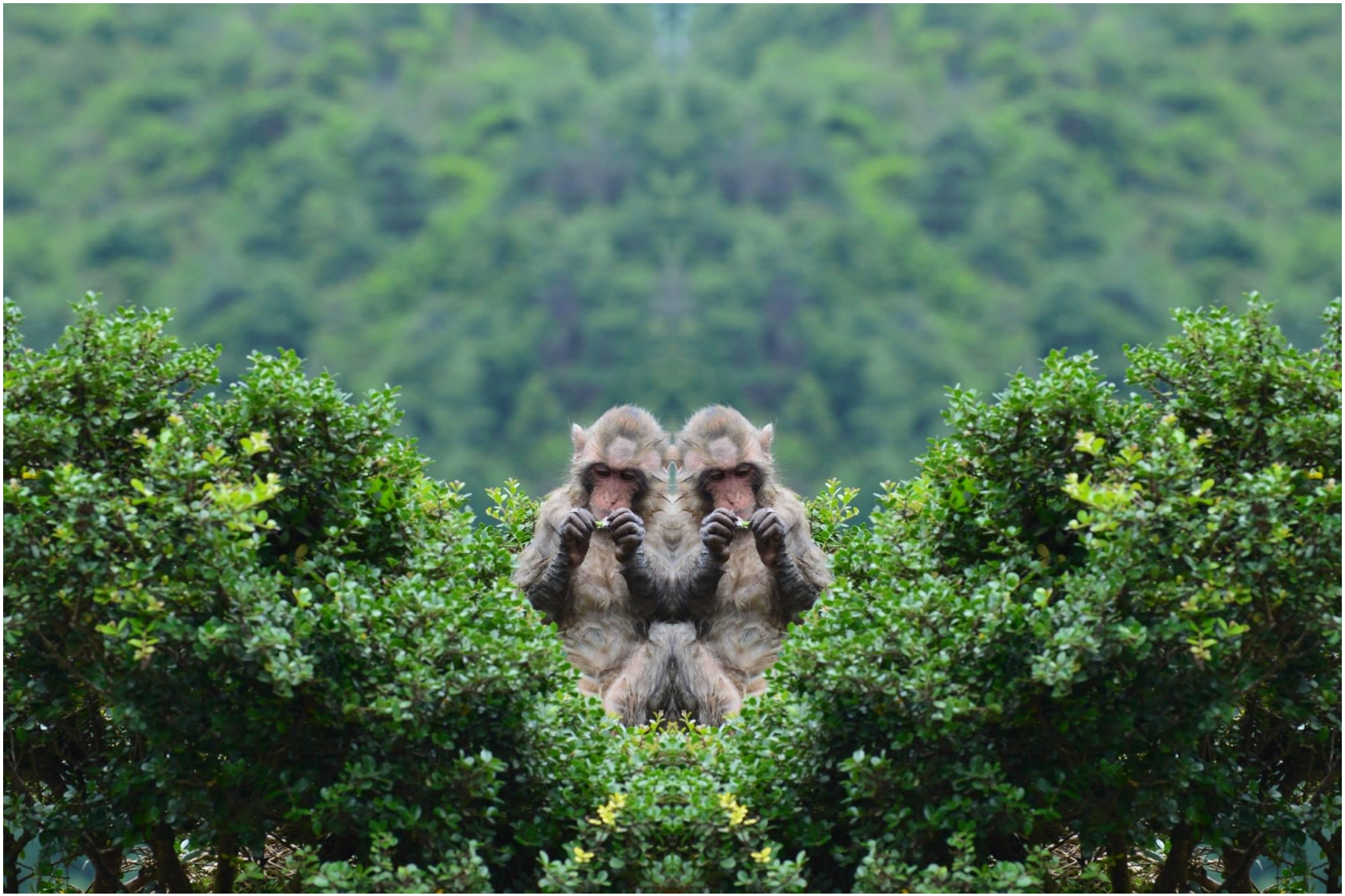
```
1  import skimage.io as io
2  import matplotlib.pyplot as plt
3
4  # read image into memory
5  image = io.imread("monkey.jpg")
6
7  n_row, n_col, n_colours = image.shape
8  print(n_row, n_col, n_colours)  # prints 1362 2048 3
9
10 # print pixel value at row 156 and column 293
11 pixel = image[156,292]
12 print(pixel)  # prints [112 158 147]
13
14 # change pixel value to red
15 image[156,292]=[255,0,0]
16
17 # Create a purple rectangle between rows 1000-1200
18 # and column 500-900
19 for i in range(1000,1200):
20     for j in range(500,900):
21         image[i,j] = (255,0,255)
22
23 plt.imshow(image)
24 plt.show()
25 io.imsave("monkey_bar.jpg",image)
```

# Creating the negative of an image

```
1  import skimage.io as io
2  import matplotlib.pyplot as plt
3
4  # read image into memory
5  image = io.imread("monkey.jpg")
6  n_row, n_col, n_colours = image.shape
7
8  # Create the negative of an image
9  for i in range(n_row):
10     for j in range(n_col):
11         for c in range(n_colours):
12             image[i,j,c] = 255-image[i,j,c]
13
14 # we could just have written:
15 #image = 255 - image
16
17 plt.imshow(image)
18 io.imsave("monkey_negative.jpg",image)
```

# Flipping the image horizontally (incorrect!)

```
1  import skimage.io as io
2  import matplotlib.pyplot as plt
3
4  # read image into memory
5  image = io.imread("monkey.jpg")
6  n_row, n_col, colours = image.shape
7
8  # Flip the image horizontally
9  for i in range(0,n_row):
10     for j in range(0,n_col):
11         image[i,j] = image[i, n_col-j-1]
12
13 plt.imshow(image)
14 io.imsave("monkey_flipped_wrong.jpg",image)
```



Problem: For each row i, this mirrors the right half of the image into the left half (as it should), but by the time it reaches the right half (j¿ n_col/2), the left half of the image has already been changed, so we can no longer recover the original pixel values.

# Flipping the image horizontally (correct)

```
1  import skimage.io as io
2  import matplotlib.pyplot as plt
3
4  # read image into memory
5  image = io.imread("monkey.jpg")
6  n_row, n_col, colours = image.shape
7
8  # Flip the image horizontally
9  for i in range(n_row):
10     for j in range(int(n_col/2)):
11         colour = image[i,j].copy()
12         opposite_colour = image[i, n_col-j-1].copy()
13         image[i,j] = opposite_colour
14         image[i, n_col-j-1] = colour
15
16 plt.imshow(image)
17 plt.show()
18 io.imsave("monkey_flipped_right.jpg",image)
```

# Combining images

Since images are just arrays are numbers, we can easily combine them.

Example: Create an image that is the average of monkey and tiger.

# Combining images

```
1  import skimage.io as io
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from skimage.transform import resize
5
6  monkey = io.imread("monkey.jpg")
7  tiger = io.imread("tiger.jpg")
8
9  #resize images to 500x1000 pixels
10 monkey_resized = resize(monkey, (500, 1000))
11 tiger_resized = resize(tiger, (500, 1000))
12
13 combined = np.zeros([500,1000,3])
14 for i in range(500):
15     for j in range(1000):
16         for c in range(3):
17             combined[i,j,c]=monkey_resized[i,j,c]/2 +\
18                             tiger_resized[i,j,c]/2
19
20 # we could also have replaced lines 13-19 with:
21 #combined = monkey_resized/2 + tiger_resized/2
22
23 plt.imshow(combined)
24 io.imsave("combined.jpg",combined)
```

# Combining images

# Color separation

Goal: Produce three images, one for each colors (see next slides)

```
1  import skimage.io as io
2  image = io.imread("monkey.jpg")
3  n_row, n_col, colours = image.shape
4
5  # create three copies of the image
6  red = image.copy()
7  green = image.copy()
8  blue = image.copy()
9
10 # set to zero the B and G channels of the red image
11 # set to zero the R and B channels of the green image
12 # set to zero the R and G channels of the blue image
13 for i in range(n_row):
14     for j in range(n_col):
15         red[i,j,1]=red[i,j,2]=0
16         green[i,j,0]=green[i,j,2]=0
17         blue[i,j,0]=blue[i,j,1]=0
18 # We could have replaced the 5 lines above with:
19 #red[:,:,(1,2)] = 0
20 #green[:,:,(0,2)] = 0
21 #blue[:,:,(0,1)] = 0
22
23 io.imsave("monkey_red.jpg",red)
24 io.imsave("monkey_green.jpg",green)
```

red intensity

green intensity

blue intensity

# Shifting colors

Goal: Produce an image where the three colour channels are shifted (see next slide)

```
1  import skimage.io as io
2  import matplotlib.pyplot as plt
3  import numpy as np
4  image = io.imread("monkey.jpg")
5  n_row, n_col, colors = image.shape
6
7  # create a blank image
8  new_image = np.zeros( (n_row, n_col, 3), dtype=np.uint8)
9
10 # assemble a new image made of shifted colors
11 # blue is shifted right by 100 pixels
12 # green is shifted up by 100 pixels
13
14 for i in range(n_row):
15     for j in range(n_col):
16         new_image[i,j,0] = image[i,j,0]  # keep red
17         if i>=100:
18             new_image[i,j,1]=image[i-100,j,1]  # move green
19         if j>=100:
20             new_image[i,j,2]=image[i,j-100,2]  # move blue
21
22 plt.imshow(new_image)
23 io.imsave("monkey_shifted.jpg",new_image)
```

# Grayscaling

Many image processing algorithms assume a 2D matrix

- not an image with a third dimension of color

To bring the image into two dimensions

- we need to summarize the three colors into a single value
- this process is more commonly know as **grayscaling**
- where the resulting image only holds intensities of gray
  - with values between 0 and 1

skimage submodule **color** has useful functions for this task

- API
  `http://scikit-image.org/docs/dev/api/skimage.color.html`

# Grayscaling

Goal: Create a grayscale version of a color image (see next slide)

```
1  import skimage.io as io
2  import matplotlib.pyplot as plt
3  from skimage.color import rgb2gray
4
5  # read image into memory
6  image = io.imread("monkey.jpg")
7  # convert to grayscale
8  gray_image = rgb2gray(image)
9
10 print(image[0,0])  # prints [255,255,255]
11 print(gray_image[0,0])  # prints 1.0
12 plt.imshow(gray_image)
13 io.imsave("monkey_grayscale.jpg",gray_image)
```

# Binary image

Goal: Produce a black-and-white version of a color image (see next slide).

```
1  import skimage.io as io
2  import matplotlib.pyplot as plt
3  from skimage.color import rgb2gray
4  import numpy as np
5
6  image = io.imread("monkey.jpg")
7  n_row, n_col, c_colours = image.shape
8
9  gray_image = rgb2gray(image)
10
11 black_and_white=gray_image.copy()
12 for i in range(n_row):
13     for j in range(n_col):
14         if gray_image[i,j]>0.5:
15             black_and_white[i,j]=1.0
16         else:
17             black_and_white[i,j]=0
18
19 # We could have replaced the 7 lines above with:
20 # black_and_white = np.where(gray_image>0.5, 1, 0)
21
22 plt.imshow(black_and_white)
23 io.imsave("monkey_bw.jpg",black_and_white)
```