MAX PLANCK INSTITUTE
FOR MOLECULAR GENETICS

Python Basics

Scripting Imaging Workshop

Clemens Kohl
Max Planck Institute for Molecular Genetics
25.09.2023

Outline

Jupyter Notebook

- Open Jupyter Notebook from Link
- File > "Save a Copy in Drive"
- Execute Cells with "Ctrl + Enter" or "Shift + Enter"

www.tinyurl.com/sva6zv98

Get the slides (or don't)

You can download the slides (and the code to generate them) from github:

www.github.com/ClemensKohl/python_nanocourse

DATA TYPES

Data Types

- What are data types and why should you care?

```
2 + 3 = ?
"hello" + "world" = ?
True + False = ?
True + 1 = ?
["version"] + [2] = ?
```

Data Types

- What are data types and why should you care?

```
>>> 2 + 3
Out: 5
>>> "hello" + "world"
Out: "helloworld"
>>> True + False
Out: 1
>>> True + 1
Out: 2
>>> ["version"] + [2]
Out: ['version', 2]
```

Different types can do different things!

Data Types

Built in Python data types:

Data Type	Name	Example
Numbers	`int`, `float`, `complex`	`1234`, `3.1415`, `3+4j`
Text	`str`	`'spam'`, `"Bob's"`
Sequences	`list`, `tuple`, `range`	`[1, 'two', 3.0]`, `("a", "b", "c")`, `range(3)`
Mapping Type	`dict`	`{'food': 'spam', 'taste': 'yum'}`
Sets	`set`	`set('abc')`, `{'a', 'b', 'c'}`
Boolean	`bool`	`True`, `False`
None Types	`NoneType`	`None`

You can get the data type of an object by using `type()`:

```
>>> type("word")
Out: <class 'str'>
```

Python Lists

- Python lists can contain elements of any type.
- Python lists are 1 dimensional (they can be nested though).
- Element-wise operations require a loop.

```
x_list = ["a", 1, 2.3, True, False, 3]
```

2nd to 4th element of the list:

```
>>> x_list[1:4]
Out: [1, 2.3, True]
```

Note

Python is 0 indexed! The slicing start point is inclusive and the end point is exclusive.

dictionaries

- Dictionaries are ”key” - ”value” pairs.
- Keys need to be unique, values can be anything.
- dictionaries are created with `{}` or `dict()`

```
microscope = {
  "brand": "Zeiss",
  "model": "LSM 980",
  "year": 2020,
  "price": 999999
}
```

Access elements, add new ones or change existing ones:

```
>>>  print(microscope["brand"])
Out: "Zeiss"
>>> microscope["broken"] = True
>>> microscope["price"] = 200
```

Keys and values are accessed through `dict.keys()` and `dict.values()`

```
>>> microscope.keys()
Out: dict_keys(['brand', 'model', 'year', 'price', 'broken'])
>>> microscope.values()
Out: dict_values(['Zeiss', 'LSM 980', 2020, 200, True])
```

Numpy Arrays

- Numpy arrays contain elements of similar type.
- Numpy arrays are N-dimensional.
- Allow for element-wise operations.
- Usually more efficient.

```python
import numpy as np
x = np.array([[1,2],          # 2-dimensional
              [3,4],
              [5,6]])

y = np.array([[[1,2,3],       # 3-dimensional
               [4,5,6]],
              [[7,8,9],
               [9,8,7]]])
```

PYTHON AS A CALCULATOR

Basic Math

- Basic Math operations:

```
a = 1+2-10       # add/subtracting numbers up
b = (7*4)/2      # multiplication and division
c = a**2         # raise to the power of x
```

- Results can be either ”integers” or ”floats”:

```
>>> 50 - 5*6
Out: 20
>>> 17 / 3
Out: 5.666666666666667
```

- Division always returns a float!

More Math operations

- Floor division and the modulo operation:

```
>>> 17 // 3  # floor division discards the fractional part
Out: 5

>>> 17 % 3  # the % operator returns the remainder of the division
Out: 2

```

Precision: Integers and Floats

- Computers can only calculate with limited precision.
- This can lead to unexpected behaviour:

```
>>> 0.1 + 0.2
Out: 0.30000000000000004
>>> 0.1 + 0.2 == 0.3
Out: False
```

- **Explanation:** 0.1 in binary is a infinitely repeating number similar to ⅓ in decimal base.
- This phenomenon called *Representation Error* is very common and needs to be taken into account.

```
>>> format(0.1, ".20g")
Out: '0.10000000000000000555'
```

Comparing Floats

Solution:

```
>>> import math
>>> math.isclose(0.1 + 0.2, 0.3)
Out: True

>>> import numpy as np
>>> np.allclose([1e10, 1e-8], [1.00001e10, 1e-9])
Out: True
>>> np.isclose([1e10, 1e-7], [1.00001e10, 1e-8])
Out: array([ True, False])
```

- Tolerances of the functions can be set as needed.
- **Don't compare floats directly!**

Basic NumPy Operations

```
img = np.random.randint(0, 255, size=(500,500,3)) # random matrix

img[i,:,:] = img[j,:]     # set the values of row i with values from row j
img[:,i] = 100           # set all values in column i to 100
img[:100, :50, :].sum()     # the sum of the values of the first 100 rows and 50 columns
img[50:100, 50:100, :]      # rows 50-100, columns 50-100 (100th not included)
img[i].mean()            # average of row i
img[:,-1]                # last column
img[-2,:]                # second to last row
```

Vectorization with numpy

```python
# Average RGB value.
mimg = img.mean(axis=2)
plt.imshow(mimg, cmap='gray')
plt.show()

# show max value per channel
plt.imshow(img.max(axis = 2), cmap='gray')
plt.show()

print(img.max())
print(img.min())

```

Channel Order

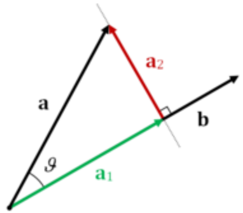
The order of channels is RGB for matplotlib and PIL, but BGR for cv2!

More vectorization

```
a = np.random.randint(0, 255, size=100)
b = np.random.randint(0, 255, size=100)

c = a - b            # element-wise subtraction.
a_transpose = a.T    # transpose

# index of max value
np.argmax(a)
np.where(a == np.max(a))
```

Vector and matrix multiplication

```python
# scalar projection of a onto b.
a1_length = np.dot(a,b)/np.linalg.norm(b)
```



```python
# multiply two 2D matrices
A = np.random.randint(0, 255, size=(500,250))
B = np.random.randint(0, 255, size=(500,250))
AxB = np.dot(A, B.T)

# multiply two 3D matrices
# matmul assumes 2D matrices in last two indices!
img = img.reshape(3, img.shape[0], img.shape[1])
mult_img = np.matmul(img, img.transpose((0,2,1)))
```

Tensor contraction

- Generalization of Matrix Multiplication for >2 dimensions.
- If you are mad enough to do this, you clearly don't need me.
- Look into `torch.tensordot()` from `pytorch`.

The easy to understand formula tensordot computes:

$$r_{i_0,\ldots,i_{m-d},i_d,\ldots,i_n} = \sum_{k_0,\ldots,k_{d-1}} a_{i_0,\ldots,i_{m-d},k_0,\ldots,k_{d-1}} \times b_{k_0,\ldots,k_{d-1},i_d,\ldots,i_n}$$

LOOPS AND CONDITIONALS

if clauses

- Executes a set of statements if a condition is true.
- It can have zero or more `elif` (else if) statements and/or an optional single `else` statement.

```python
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Wherever possible, avoid nested conditions!

for loops

- A for loop iterates over a sequence (called iterator, e.g. a list, tuple, dictionary, ...).
- Technically, the iterator must have a `__iter__()` and `__next__()` methods.
- Within the loop we can execute a set of statements.

Simple example:

```python
fruits = ["apple", "banana", "cherry"]

for x in fruits:
    print(x)
    if x == "banana":
        print("I don't like banana.")
```

continue and break

- The `continue` statement skips the current iteration.
- The `break` statement exits the the loop.

Examples:

```python
for i in range(1,10):
    if i == 5:
        continue
    print("Number", i)

for i in range(1,10):
    if i == 5:
        break
    print("Number", i)
```

Loop tips

- Pre-allocate the memory for the output of a loop.
- If possible **do not append to a list** in each iteration.

```python
results = [None] * 1000         # Make a list of 1000 None's.
for i in range(1000):
    is_even = (i % 2) == 0
    results[i] = is_even        # Save result in spot in list.
```

Loop tips - 2

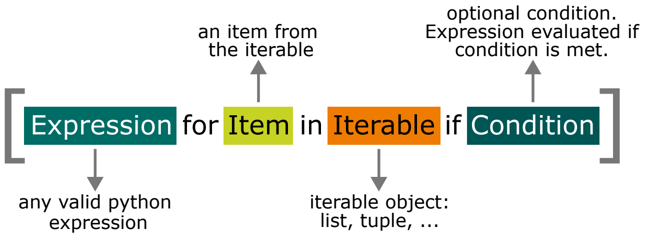
- use `enumerate()` and `zip()`

```python
nums = [1, 2, 3]
letters = ["a", "b", "c"]

for idx, val in enumerate(letters):     # get the index and value.
    print(idx, val)

for val1, val2 in zip(nums, letters):   # Loop over two lists.
    print(val1, val2)
```

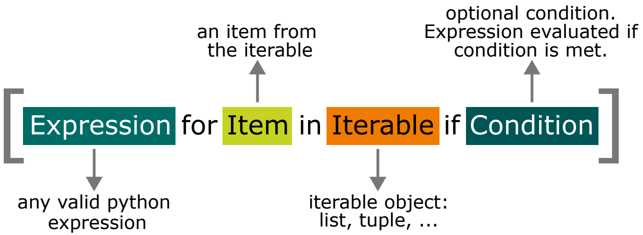
- If you can, **avoid for loops!**
- Use built-in or numpy functions instead.

List comprehension

- Shorter syntax than for loop.
- No need to preallocate memory.
- Often faster than for loops.
- Street cred. More ”pythonic”.


an item from the iterable
optional condition. Expression evaluated if condition is met.
[Expression for Item in Iterable if Condition]
any valid python expression
iterable object: list, tuple, ...

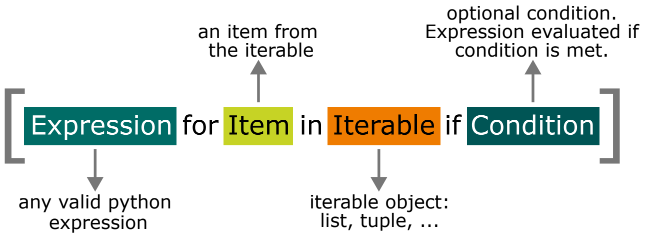
List comprehension - Example


an item from the iterable
optional condition. Expression evaluated if condition is met.
[Expression for Item in Iterable if Condition]
any valid python expression
iterable object: list, tuple, ...

with for loop:

```python
squared_nums = []
for number in range(10):
    if number % 2 == 0:
        squared = number ** 2
        squared_nums.append(squared)
```

List comprehension- Example



[Expression for Item in Iterable if Condition]
an item from the iterable
optional condition. Expression evaluated if condition is met.
any valid python expression
iterable object: list, tuple, ...

with list comprehension:

```python
squared_numbers = [i**2 for i in range(10) if i % 2 == 0]
```

while loops

- Executes a set of statement as long as a condition is true.
- `else` statement gets executed when the condition is no longer true.
- Works with `break` and `continue` statements.
- Can be thought of as a fusion between if clauses and for loops.

```python
i = 0
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Make sure that the exit condition will (eventually) happen!

FUNCTIONS AND CLASSES

Defining functions

- Functions are short modules that accomplish a specific task.
- They are defined with the `def` keyword and the parameters in the parantheses.
- Functions always return something. If undefined the function returns `None`.

```
def fib(n, a=0, b=1):
    """Print a Fibonacci series up to n."""
    result = []
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

```
>>> fib(500)    # == fib(n=500, a=0, b=1) or fib(500,0,1)
Out: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

Functions - tips

- Default values for arguments can be specified:

```python
def square_x(x, y = 2):
    return x ** y
```

- Variables defined inside a function are **only valid within that function**.
- To use variables from the global namespace use `global`. Use it only if **absolutely necessary!**

```python
def f():
    global x
    x = 40
    print(x)
```

```
>>> x = 20
>>> f()
Out: 40
>>> x
Out: 40
```

Classes

- Classes are abstract blueprints from which objects are created.
- Instances are realizations of the abstract blueprint and have actual values.
- Classes often bundle data and functions together.
- New instances of a class can have different properties.

Simple class definition:

```python
class MyClass:
  x = 5
```

Class example

Defining a new class `Dog`:

```python
class Dog:
    kind = 'canine'         # class variable shared by all instances

    def __init__(self, name, breed, age):
        self.name = name    # instance variable unique to each instance
        self.breed = breed
        self.age = age
        self.tricks = []

    def describe(self): # class method
        print(self.name, "is a(n)", self.breed, "and", self.age, "years old.")

    def add_trick(self, trick):
        self.tricks.append(trick) # We can change instance variables.
```

Class instances

Create a new instance of this class and call its method function:

```
>>> d = Dog(name = "Rex", breed = "German Shepherd", age = 4)
>>> d.kind
Out: 'canine'
>>> d.describe()
Out: "Rex is a German Shepherd and 4 years old."
```

We can create a second instance:

```
>>> d2 = Dog(name = "Laika", breed = "unknown breed", age = 3)
>>> d2.add_trick("roll over")
>>> d2.add_trick("fly to space")
>>> d2.tricks
Out: ['roll over', 'fly to space']
>>> d.tricks
Out: []
```

Class inheritance

- Classes can inherit properties from a parent Class
- The parent class can also be from an external library.
- Derived classes may override methods of their base classes.

```python
class DerivedClassName(BaseClassName):
    pass
```

- Check inheritance with `isinstance()`:

```
>>> isinstance(d, Dog)
Out: True
```

LIBRARIES

What are libraries?

- A python library is a collection of related modules or functions.
- Libraries allow us to reuse code for different projects.
- The python standard library consists of 200 modules for basic system functionality (e.g. I/O).
  - The python standard library comes pre-installed with python.
  - **https://docs.python.org/3/library/index.html**
- Making a library out of your own code is easy!

Useful libraries - General

For general use:

- **os:** For accessing the operating system.
- **Numpy:** Numeric Python. For working with Matrices.
- **Pandas:** For working and manipulating with tabulated data.
- **SciPy:** Scientific Python. Contains functions for common scientific computations.
- **PyTorch:** Machine Learning, Neural Nets, ...
- **Matplotlib:** For plotting.
- **Seaborn:** Built on Matplotlib but easier to use, esp. with Pandas.

Useful libraries

Image manipulation:

- **Python Imaging Library (PIL):** General image handling.
- **OpenCV/cv2:** Computer vision library with many useful functions (e.g. edge detection etc.).
- **PyTorch:** (again) machine learning tools for images.

Pandas

- Provides the 2D `DataFrame` and 1D `Series` classes for handling data.

```python
import numpy as np
import pandas as pd

df = pd.DataFrame(np.random.randn(6, 4),
                  index = ["R1", "R2", "R3", "R4", "R5", "R6"],
                  columns=["C1", "C2", "C3", "C4"])
```

```
>>> df
```

	C1	C2	C3	C4
R1	−0.297 058	−1.201 026	0.270 988	−0.213 413
R2	0.090 518	0.038 817	−0.306 152	−0.415 315
R3	0.700 081	0.476 054	0.558 491	0.358 557
R4	0.535 402	−0.094 973	2.247 575	−0.210 451
R5	−1.407 642	0.135 530	0.062 964	0.474 207
R6	−1.111 652	0.877 221	0.427 484	0.360 299

Viewing data

- Use `df.head()` and `df.tail()` to view the top and bottom rows of the data frame.
- Use `df.to_numpy()` to convert back to numpy array.
- `df.index` and `df.columns` return the row and column names.
- Slicing can be done either with `df.iloc` or by label `df.loc`.

```
>>> df.iloc[0:2, 0:2]
>>> df.loc["R1":"R2", ["C1", "C2"]]
```

Reading in text files

- There are many ways of reading in a text file in python.
- The `with` statement makes sure all files are closed in the end.
- For unstructured files one can use the following:

```python
# read from a file without with.
file = open("lorem.txt", "r+")
lines = file.read().splitlines()
file.close()

# write to a file with "with"
with open("newfile.txt", "w") as file2:
    newline = "text to add to file"
    file2.write(newline)
```

Reading in .csv files

- For .csv files either use numpy or pandas.
- Pandas `read_csv` creates a data frame, whereas `genfromtxt` creates a numpy array.

Pandas:

```python
import pandas as pd
df = pd.read_csv('iris.csv',
                 sep=',')
```

Numpy:

```python
import numpy as np
mat = np.genfromtxt('iris.csv',
                    delimiter=',',
                    skip_header = 1,
                    usecols = range(4))
```

Note

For data of mixed types use pandas! Numpy expects all data to be of the same type!

Writing .csv files

- Writing .csv files can be done with both packages again.
- Pandas `to_csv` or numpy's `savetxt`.

Pandas:

```python
# if you want to keep
# rownames: index=True
df.to_csv("pd_iris.csv",
          sep=',',
          index=False)
```

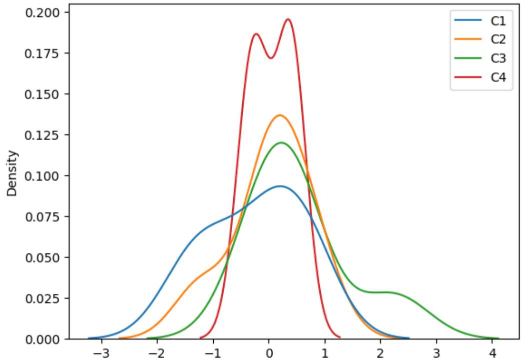
Numpy:

```python
np.savetxt("np_iris.csv",
           mat,
           delimiter=",")
```

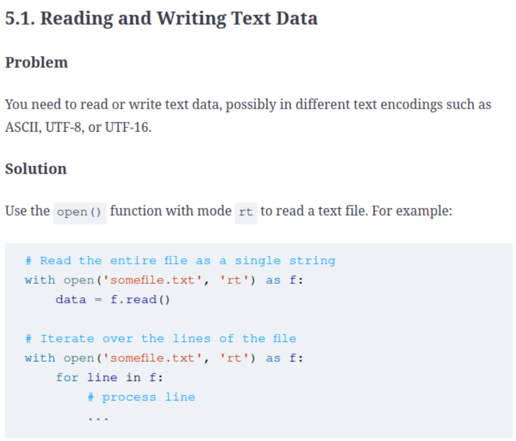
Plotting with seaborn

- The Seaborn library allows for easy plotting.
- Takes pandas data frame as input.

```python
import seaborn as sns
sns.kdeplot(data=df)
```



Python Cookbook



5.1. Reading and Writing Text Data

Problem

You need to read or write text data, possibly in different text encodings such as ASCII, UTF-8, or UTF-16.

Solution

Use the `open()` function with mode `rt` to read a text file. For example:

```
# Read the entire file as a single string
with open('somefile.txt', 'rt') as f:
    data = f.read()

# Iterate over the lines of the file
with open('somefile.txt', 'rt') as f:
    for line in f:
        # process line
        ...
```

Beazley, D. and Jones, B.K., 2013. Python cookbook: Recipes for mastering Python 3. " O'Reilly Media, Inc.".

Conda environments

- Often certain libraries require certain python versions and/or versions of other libraries.
- These requirements can be mutually exclusive between libraries.
- How to manage different versions?
- **Solution:** Conda (**www.conda.io**) is a package and environment management system.
- It allows to install different versions of python in parallel and keep different packages for different tasks.

EXAMPLES

The Value(s) of A Picture

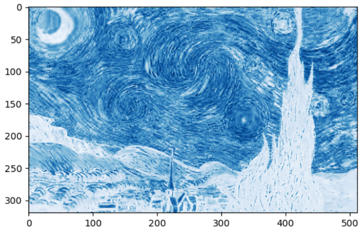
https://tinyurl.com/starrynightjpg

```python
import matplotlib.pyplot as plt
import numpy as np

img = plt.imread('starry_night.jpg')
img.shape # Order of channels: RGB
```

Plotting a single color channel

Task: Plot the blue channel (or any other channel) of the picture in the respective color.

```
# Decide on channel to plot
...
# Plot picture.
# use cmap to set color!
...
```

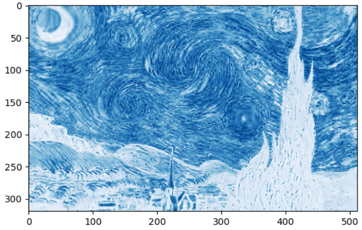


Remember:

You can use `imshow` from `matplotlib.pyplot` to plot an array

Plotting a single color channel

```python
import matplotlib.pyplot as plt

# Show just blue channel
plt.imshow(img[:,:,2], cmap='Blues')
plt.show()
```



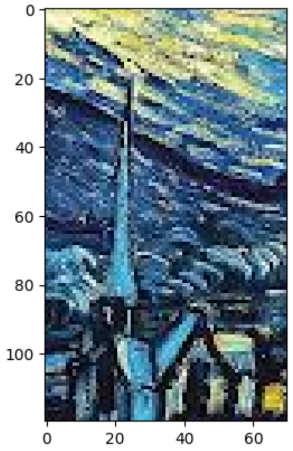
Note:

cmap controls the output colors: **https://matplotlib.org/stable/users/explain/colors/colormaps.html**

Cropping

- To crop just specify the indices of the matrix.

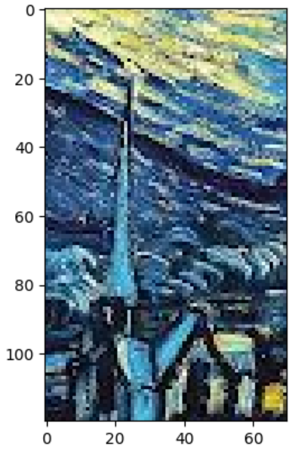
```
# slice and plot image
...
```



Cropping

- To crop just specify the indices of the matrix.

```
cropped = img[200:, 200:270, :]
plt.imshow(cropped)
```

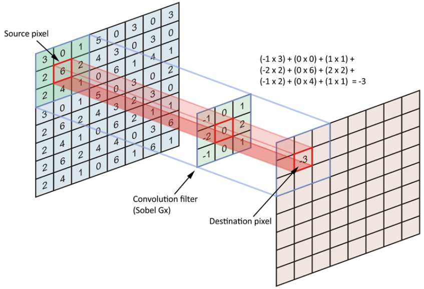


Example: Edge detection

- Edges can be found through the image derivatives I_x and I_y along the x and y axis .
- They describe how image intensity changes over the image.
- In most cases they are approximated with convolutions.
- A popular choice is the Sobel filter:

$$D_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, D_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- The magnitude of the image gradient $\nabla I = [I_x, I_y]^T$ is just its length: $|\nabla I| = \sqrt{I_x^2, I_y^2}$.


Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

https://i.stack.imgur.com/YDusp.png

Convert to grayscale

Step 1: Convert to grayscale image:

```
img = plt.imread('starry_night.jpg')

# Convert to grayscale:
# Red*0.2989 + Green*0.5870 + Blue*0.1140
# Honestly, no clue why these numbers.
...
# Plot
...
```

Tip

If a is an N-D array and b is a 1-D array, `np.dot(a,b)` is a sum product over the last axis of a and b. **What about `np.matmul`?**

Convert to grayscale

Step 1: Convert to grayscale image:

```python
img = plt.imread('starry_night.jpg')

# np.matmul would give the same result! Why?
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

gray_img = rgb2gray(img)

plt.imshow(gray_img, cmap='gray')
plt.show()
```

Convolution

Step 2: Convolution with Sobel Filter.

```
# Make array for Sobel filter Dx
...
# Make array for Sobel filter Dy
...
# Convolve image with filters
# Consider image size when convolving them.
...
# Calculate magnitude
...
```

Tip

You can use the `convolve` function from `scipy.signal`!

Convolution

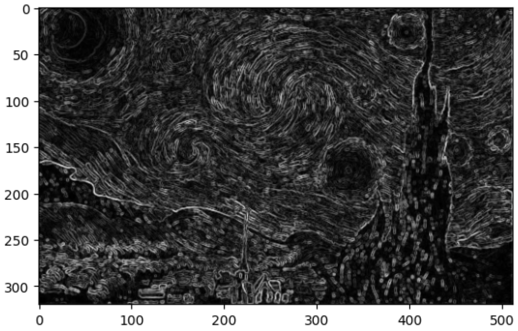
Step 2: Convolution with Sobel Filter.

```
from scipy.signal import convolve

sobel_x= np.array([[-1, 0, 1],
                   [-2, 0, 2],
                   [-1, 0, 1]])

sobel_y= np.array([[-1, -2, -1],
                   [ 0,  0,  0],
                   [ 1,  2,  1]])

Ix = convolve(gray_img, sobel_x, mode='same')
Iy = convolve(gray_img, sobel_y, mode='same')

grad_magnitude = np.sqrt(np.square(Ix) + np.square(Iy))
```

Example: Edge detection - Results

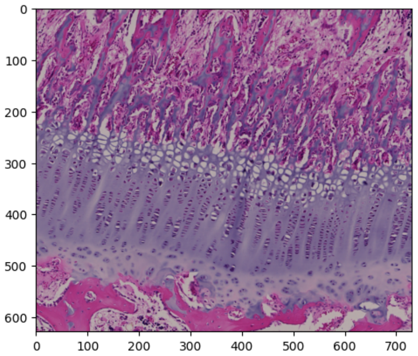
Step 3: Plot results.

```python
plt.imshow(grad_magnitude, cmap='gray')
plt.show()
```



Pig tissue

```python
import cv2
img = cv2.imread('pig_tissue.tif')
# Reduce picture size.
img = cv2.resize(img, None, fx=0.5, fy=0.5)
# convert BGR to RGB colors
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# same as the more cryptic img = img[...,::-1]

plt.imshow(img)
```

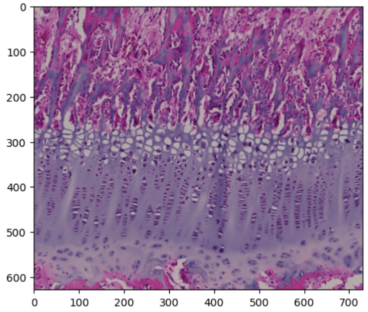


Task: Rotate image around the center by 8° and crop in by 120 %.

Rotate image

Task: Rotate image around the center by 8° and crop in by 120 %.

```
# find center of image
...
# make rotation matrix, f(center, degrees, scale)
...
# rotate image
...
# plot
...
```

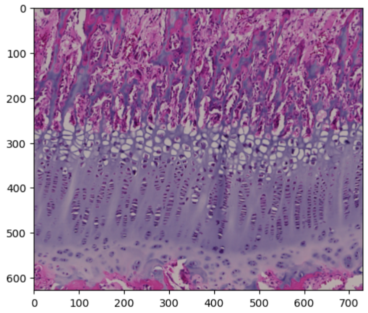


Tip

You can use `cv2.getRotationMatrix2D()` and `cv2.warpAffine()`.

Rotate image

```python
# find center of image
height, width = img.shape[:2]
centerX, centerY = (width // 2, height // 2)

# rotation matrix, f(center, degrees, scale)
M = cv2.getRotationMatrix2D((centerX, centerY), 8, 1.2)

# rotate image
rotated = cv2.warpAffine(img, M, (width, height))
plt.imshow(rotated)
```



Mean RGB

Calculate row-wise mean over each channel seperately:

```
# Calc. row-wise mean
...
# Convert to pandas data frame
...
# Optional: pivot from wide to long format.
...
```

Tip

Checkout the documentation to `pandas.DataFrame`!

Mean RGB

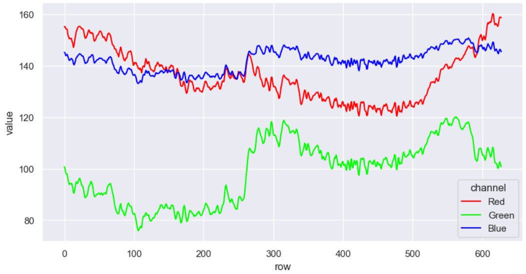
Calculate row-wise mean over each channel seperately:

```python
row_mean = rotated.mean(axis=1) # row-wise mean

# Convert to pandas data frame
rgb = pd.DataFrame(row_mean,
                   columns = ["Red", "Green", "Blue"])

rgb["row"] = range(row_mean.shape[0])
rgb.set_index("row")
rgb_long = pd.melt(rgb, id_vars = "row", var_name="channel", value_vars=["Red", "Green", "Blue"])
```

RGB lineplot

Plot RGB channels over the rows:

```
# Optional: Pick custom colors.
...
# Use seaborn to plot d
# plot pandas dataframe.
...
```



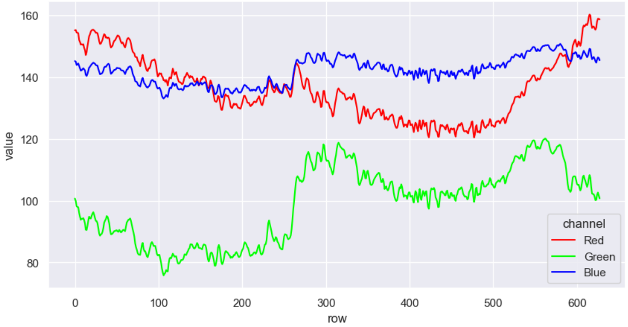
Note

Instead of seaborn and pandas one could use matplotlib to directly plot the numpy array. Because it is arguably much more cryptic and difficult to learn this is skipped here.

RGB lineplot

Plot RGB channels over the rows:

```python
colors = ["#FF0000", "#00FF00", "#0000FF"]
sns.lineplot(rgb_long, x="row", y = "value", hue = "channel", palette = colors)
```



Example solutions

- Open Jupyter Notebook from Link
- File > ”Save a Copy in Drive”
- Execute Cells with ”Ctrl + Enter” or ”Shift + Enter”

www.tinyurl.com/mr3x2se6