Python for Data Scientists
L13 : Data science libraries

Part 1

Python or R for Data Science?

Python or R

- Different community : statisticians and computer science communities
- Similar functionalities

Read more:
https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis

Popularity Rankings, Python vs. R

	Python	R
2016	#3	#5
2017	#1	#6
2018	#1	#7
2019	#1	#5

Source: IEEE Spectrum, Sep 2019

Python or R

Python	R
General use	Best tool for visualization and graphs
Code readability, speed	Specific to data analysis
Several libraries	Great for statistical analysis
Great mathematical computation	Easy to learn if you have programming background

Python or R

Python	R
R has more libraries	Finding the right package to use is time consuming
Requires rigorous testing as errors show up in runtime	There are many dependencies between R libraries
Visualizations are more convoluted in python that in R	Not popular for deep learning and NLP

Python Libraries for Data Science

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

Visualization libraries

- matplotlib
- Seaborn

and many more …

Part 1: Numerical Python
NumPy

Numpy?

- Core library for scientific computing in Python.

- Provides a high-performance multidimensional array object, and tools for working with these arrays.

Why NumPy?

- Internally stores data in a contiguous block of memory, independent of other built-in Python objects

- NumPy’s library of algorithms written in C can operate on this memory without any type checking or other overhead

Why NumPy?

- NumPy arrays use much less memory than built-in Python sequences

- NumPy operations perform complex computations on entire arrays without the need for python loops

Why NumPy?

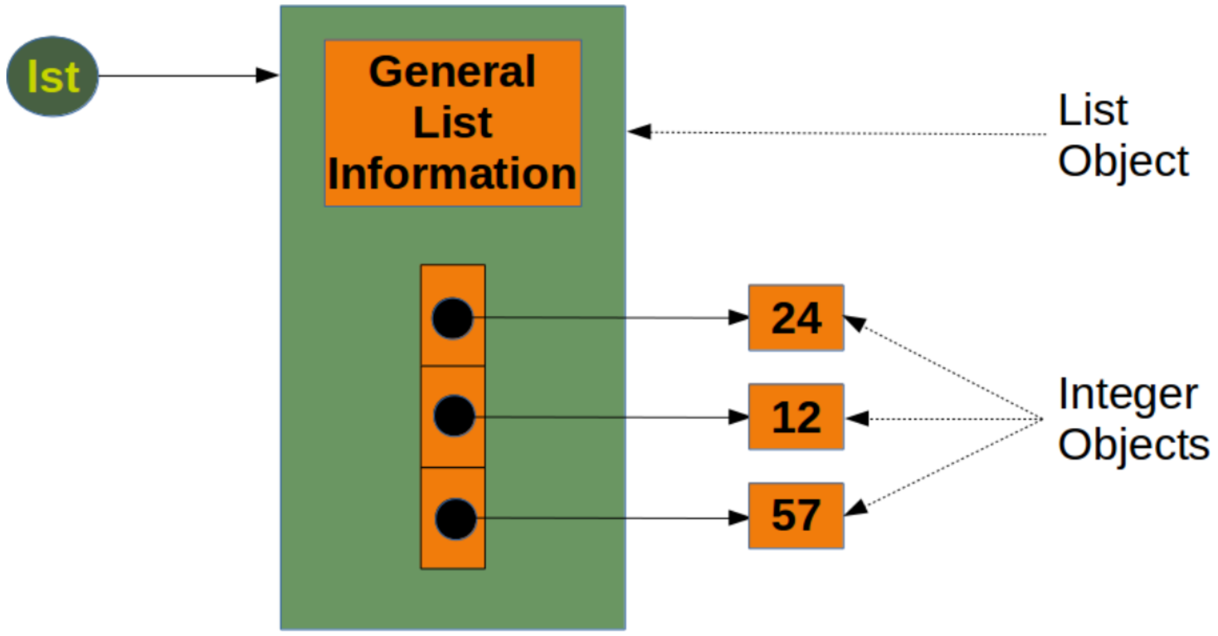
Numpy- based algorithms are generally 10 to 100 times (or more) faster than pure Python counterparts and uses stigmatically less memory

```
import time
import numpy as np
arr = np.arange(1000000)
l = list(range(1000000))
start = time.time()
for _ in range (100):
    arr2 = arr * 2
end = time.time()
print(f"Runtime of numPy array : {end - start}")

start = time.time()
for _ in range (100):
    l2 = [x * 2 for x in l]
end = time.time()
print(f"Runtime of Python List : {end - start}")
```

Runtime of numPy array : 0.12903070449829102
Runtime of Python List : 7.34744930267334

Why NumPy?

Memory usage of numpy arrays compared to the memory consumption of Python lists:



Why NumPy?

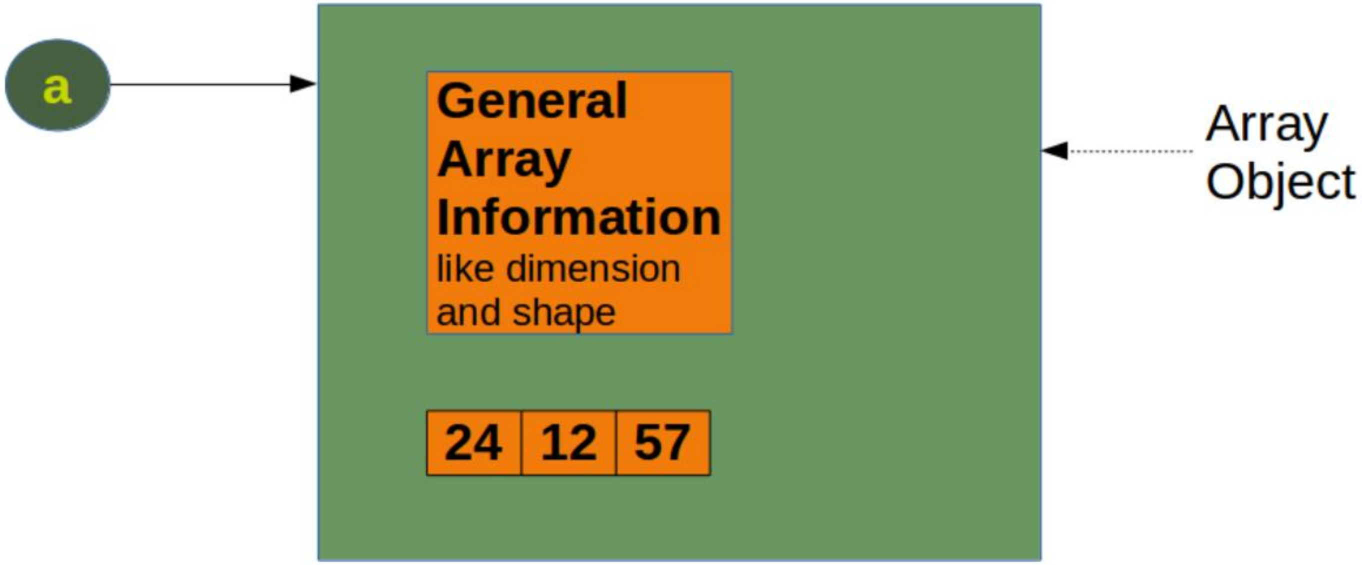
Memory usage of numpy arrays compared to the memory consumption of Python lists:

```
from sys import getsizeof as size

lst = [24, 12, 57]

size_of_list_object = size(lst)   # only green box
size_of_elements = len(lst) * size(lst[0]) # 24, 12, 57

total_list_size = size_of_list_object + size_of_elements
print("Size without the size of the elements: ", size_of_list_object)
print("Size of all the elements: ", size_of_elements)
print("Total size of list, including elements: ", total_list_size)
lst = []
print("Emtpy list size: ", size(lst))
```

Size without the size of the elements: 88
Size of all the elements: 84
Total size of list, including elements: 172
Emtpy list size: 64

Why NumPy?

Memory usage of numpy arrays compared to the memory consumption of Python lists:



Why NumPy?

Memory usage of numpy arrays compared to the memory consumption of Python lists:

```
import numpy as np
from sys import getsizeof as size

a = np.array([24, 12, 57])
print(size(a))

e = np.array([])
print(size(e))
```

```
108
96
```

Creating NumPy arrays

Creating a numpy arrays from sequence-like objects:

```python
import numpy as np

data = [1, 3, 6, 9]
arr = np.array(data)
print(arr)
print(arr.ndim)
print(arr.shape)

data2 = [[1, 3, 6, 9],[4, 7, 8, 9]]
arr2 = np.array(data2)
print(arr2)
print(arr2.ndim)
print(arr2.shape)
```

```
[1 3 6 9]
1
(4,)

[[1 3 6 9]
 [4 7 8 9]]
2
(2, 4)
```

Functions to create Numpy arrays

```
import numpy as np
```

- Create an array of all zeros

```
a = np.zeros((3,3))
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

- Create an array of all ones

```
b = np.ones((3,3))
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

- Create a constant array

```
c = np.full((3,4), 10)
```

```
[[10 10 10 10]
 [10 10 10 10]
 [10 10 10 10]]
```

- Create an identity matrix

```
d = np.eye(3)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

- Create an array with random values

```
e = np.random.random((3,3))
```

```
[[0.69305812 0.29505744 0.34006237]
 [0.6299978  0.29455254 0.37892641]
 [0.82626825 0.11757629 0.09561123]]
```

Functions to create Numpy arrays

Difference between array and asarray?

```
import numpy as np

A = np.ones((3,3))
print(A)

np.array(A)[2]=2
print(A)

np.asarray(A)[2]=2
print(A)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

[[1. 1. 1.]
 [1. 1. 1.]
 [2. 2. 2.]]
```

Documentation:

https://numpy.org/doc/stable/reference/generated/numpy.array.html

https://numpy.org/doc/stable/reference/generated/numpy.asarray.html

NumPy array creation functions

Function	Description
array	Convert input data (list, tuple, array, …) to an ndarray
asarray	Convert input to ndarray (do not copy object)
arange	Like the built-in range but returns an ndarray and not a list
ones,	Create an array of all 1s
ones_like	Take another array and create a 1s array with the same dtype and shape
zeros	Create an array of all 0s
zeros_like	Take another array and create a 0s array with the same dtype and shape
empty,	Create new arrays by allocating new memory but without any values
full,	Create an array with the given dtype and shape and values set to the indicated “fill value”
eye, identity	Create an NxN array with 1s on the diagonal and 0s elsewhere

Poll

What will be the output of this code?

```
import numpy as np

a = np.array([1,2,3,4,5])
b = np.arange(0,10,2)
c = a + b
print (c[4])
```

- 4
- 5
- 13
- None of the above

Data types for ndarrays

The data type (dtype) contains information that the ndarray needs to interpret a chunk of memory as a particular type of data

```
import numpy as np

arr1 = np.array([1,3,5], dtype=np.float)
arr2 = np.array([1,3,5], dtype=np.int)

print(arr1.dtype)
print(arr2.dtype)
```

```
float64
int32
```

Data types for ndarrays

Numpy type	C type	Description
np.int8	`int8_t`	Byte (-128 to 127)
np.int16	`int16_t`	Integer (-32768 to 32767)
np.int32	`int32_t`	Integer (-2147483648 to 2147483647)
np.int64	`int64_t`	Integer (-9223372036854775808 to 9223372036854775807)
np.uint8	`uint8_t`	Unsigned integer (0 to 255)
np.uint16	`uint16_t`	Unsigned integer (0 to 65535)
np.uint32	`uint32_t`	Unsigned integer (0 to 4294967295)
np.uint64	`uint64_t`	Unsigned integer (0 to 18446744073709551615)
np.intp	`intptr_t`	Integer used for indexing, typically the same as `ssize_t`
np.uintp	`uintptr_t`	Integer large enough to hold a pointer
np.float32	`float`	
np.float64 / *np.float_*	`double`	Note that this matches the precision of the builtin python *float*.
np.complex64	`float complex`	Complex number, represented by two 32-bit floats (real and imaginary components)
np.complex128 / *np.complex_*	`double complex`	Note that this matches the precision of the builtin python *complex*.

https://numpy.org/devdocs/user/basics.types.html

Data types for ndarrays: conversions between types

```
import numpy as np

arr1 = np.array([1, 3, 5])
print(arr1)
print(arr1.dtype)

arr2 = arr1.astype(np.float)
print(arr2)
print(arr2.dtype)
```

[1 3 5]
int32
[1. 3. 5.]
float64

→ Integers were cast to floating values with astype function

Arithmetic with NumPy

Any arithmetic operations between equal-size arrays applies the operation element-wise

```
import numpy as np

arr = np.array([[1,2,3],[4,5,6]])

arr1= arr * arr
print(arr1)

arr2= arr + arr
print(arr2)
```

```
[[ 1  4  9]
 [16 25 36]]

[[ 2  4  6]
 [ 8 10 12]]
```

Arithmetic with NumPy

Arithmetic operations with scalars propagate the scalar argument to each element in the array:

```
import numpy as np

arr = np.array([[1,2,3],[4,5,6]])

arr1= 1 / arr
print(arr1)

arr2= arr ** 2
print(arr2)
```

```
[[1.         0.5        0.33333333]
 [0.25       0.2        0.16666667]]

[[ 1  4  9]
 [16 25 36]]
```

Arithmetic with NumPy: Poll

What will be the output of this code?

```
import numpy as np

ary = np.array([1,2,3,5,8])
ary = ary + 1
print (ary[1])
```

- 1
- 2
- 3
- 4

Arithmetic with NumPy: Poll

What will be the output of this code?

```
import numpy as np

a = np.zeros(6)
b = np.arange(0,10,2)
c = a + b
print (c[4])
```

- 0
- 8
- 9
- None of the above

c = a + b
ValueError: operands could not be broadcast together with shapes (6,) (5,)

Arithmetic with NumPy

Comparisons operations:

```
import numpy as np

arr1 = np.array([[1,2,3],[4,5,6]])
arr2 = np.array([[5,2,4],[1,9,0]])


print(arr1 > arr2)
```

```
[[False False False]
 [ True False  True]]
```

Array indexing and slicing

```python
import numpy as np

# Create array with shape (3, 4)
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a)

# Use slicing to pull out the subarray consisting of the
#first 2 rows
# and columns 1 and 2;
b = a[:2, 1:3]
print(b)

# A slice of an array is a view into the same data, so
#modifying it
# will modify the original array.
print(a[0, 1])
b[0, 0] = 0
print(a[0, 1])
print(a)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

[[2 3]
 [6 7]]

2

0

[[ 1  0  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

Array indexing and slicing

```
import numpy as np


arr1 =
np.array([1,2,3,4,5,6,7,8,9])
print(arr1)

arr2 = arr1[4:7]
print(arr2)

arr2[1]=0
print(arr1)

arr2[:]=100
print(arr1)
```

```
[1 2 3 4 5 6 7 8 9]

[5 6 7]

[1 2 3 4 5 0 7 8 9]

[  1   2   3   4 100 100 100   8   9]
```

→ Data is not copied, any modification of the view will be propagated to the source array

Array indexing and slicing

If you need a copy of a slice instead of a view → explicitly copy the array

```
import numpy as np


arr1 = np.array([1,2,3,4,5,6,7,8,9])
print(arr1)

arr2 = arr1[4:7].copy()
print(arr2)

arr2[1]=0
print(arr2)
print(arr1)

arr2[:]=100
print(arr2)
print(arr1)
```

```
[1 2 3 4 5 6 7 8 9]

[5 6 7]

[5 0 7]
[1 2 3 4 5 6 7 8 9]

[100 100 100]
[1 2 3 4 5 6 7 8 9]
```

Array indexing and slicing

Integer array indexing

```
import numpy as np

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])


row_r1 = a[1, :]
row_r2 = a[1:3, :]
print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)
```

```
[5 6 7 8] (4,)

[[ 5  6  7  8]
 [ 9 10 11 12]] (2, 4)
```

Array indexing and slicing

Boolean array indexing

selecting contents from an array based on logical conditions:

```python
import numpy as np

a = np.random.randn(4,7)
print(a)

bool_ind = a > 0
print(bool_ind)

x= a[bool_ind]
print(x)
```

```
[[-2.14853509  1.38715597 -0.92964869  0.25819729 -0.18309206  0.26834034  0.46296071]
 [-0.98666892 -1.92407999 -0.69839165 -0.94686963 -0.58001767  0.73708927  0.29326033]
 [-1.91252571 -0.84641034 -0.30217966  0.58125066 -1.27434801  2.44977101 -0.34783986]
 [-0.23512438 -0.21875003 -0.06464858 -2.91213078  0.44304863  1.24475592 -0.28192733]]

[[False  True False  True False  True  True]
 [False False False False False  True  True]
 [False False False  True False  True False]
 [False False False False  True  True False]]

[1.38715597 0.25819729 0.26834034 0.46296071 0.73708927 0.29326033
 0.58125066 2.44977101 0.44304863 1.24475592]
```

Array indexing and slicing

Boolean array indexing

selecting contents from an array based on logical conditions:

```
import numpy as np

a = np.random.randn(2,2)
print(a)

bool_ind = (a > 0) & (a < 1)
print(bool_ind)

x= a[bool_ind]
print(x)
```

```
[[-0.76355829  1.36696312]
 [ 1.51781204 -0.56326015]]

[[False False]
 [False False]]

[]
```

The python keywords 'and' and 'or' do not work with Boolean array. Use & and | instead.

Universal Functions (ufunc): Fast Element-Wise Array Functions

Is a function that performs element-wise operations on data in ndarrays:

- fast vectorized wrappers for simple functions that take one or more scalar values and produce one or more scalar values.

Universal Functions: Fast Element-Wise Array Functions

Example: unary unfuncs (take one array and return a single array.

```
import numpy as np

a = np.arange(10)
print(a)

a1=np.sqrt(a)
print(a1)

a2=np.exp(a)
print(a2)
```

```
[0 1 2 3 4 5 6 7 8 9]

[0.         1.         1.41421356 1.73205081 2.         2.23606798
 2.44948974 2.64575131 2.82842712 3.        ]

[1.00000000e+00 2.71828183e+00 7.38905610e+00 2.00855369e+01
 5.45981500e+01 1.48413159e+02 4.03428793e+02 1.09663316e+03
 2.98095799e+03 8.10308393e+03]
```

Universal Functions: Fast Element-Wise Array Functions

Example: binary ufuncs (take two arrays and return one single array

```
import numpy as np

a = np.random.randn(4)
print(a)

b = np.random.randn(4)
print(b)

x=np.maximum(a,b)
print(x)
```

```
[ 0.74925885  0.02549931 -0.50154891 -0.60567738]

[ 1.14871075  0.33654484 -0.63097419  0.04487155]

[ 1.14871075  0.33654484 -0.50154891  0.04487155]
```

For more ufuncs : https://numpy.org/doc/stable/reference/ufuncs.html

Array-Oriented Programming with arrays

Mathematical and statistical methods:

- A set of mathematical functions that compute statistics about an entire array or about the data along an axis are accessible as methods of the array class

Array-Oriented Programming with arrays

Example

```
import numpy as np

a = np.random.randn(5,4)
print(a)

x = a.mean()
print(x)

x1 = a.mean(axis=1)
print(x1)

y = a.sum()
print(y)

y = a.sum(axis=0)
print(y)
```

```
[[-1.53513062 -0.05743823 -1.33565165  0.37861204]
 [ 0.1842236   0.04992617 -1.49397836 -1.40422091]
 [-2.29442692 -0.58113661  0.31511513  1.28463518]
 [ 0.33747171  0.46018862 -2.4088402   0.58841714]
 [-0.25327641 -0.06590195 -0.31692971 -0.29659129]]

-0.422246630850141

[-0.63740211 -0.66601237 -0.3189533  -0.25569068 -
0.23317484]

-8.444933261700282
[-3.56113865 -0.19436199 -5.24028479  0.55085217]
```

For more basic array statistical methods : https://numpy.org/doc/stable/reference/routines.statistics.html

Array-Oriented Programming with arrays: Poll

What will be the output of this code?

```
import numpy as np

a = np.array([[0, 1, 2], [3, 4, 5]])
b = a.sum(axis=1)
print (b)
```

- [3, 12]
- [3, 5, 7]
- 15
- None of the above

Array-Oriented Programming with arrays

Methods for Boolean Arrays:

```
import numpy as np

a = np.random.randn(5,4)
print(a)

x = (a>0).sum()
print(x)

bools = np.array([True, False,
False, True])
print(bools.any())
print(bools.all())
```

```
[[-0.4710685   0.45318137 -0.26447602  0.85586768]
 [-1.27158324  0.1399954   0.1413527  -1.61994885]
 [-0.88473953  1.4242541   0.65317862  1.06823107]
 [-1.92450076  0.95227345 -0.23874638  1.29757842]
 [ 0.55753215  0.90314486  0.71384226  0.0097927 ]]

13

True

False
```

Array-Oriented Programming with arrays

Sorting:

numpy.sort(a, axis=-1, kind=None, order=None)

→ Return a sorted copy of an array

https://numpy.org/doc/stable/reference/generated/numpy.sort.html

Array-Oriented Programming with arrays

Sorting:

```
import numpy as np

a = np.array([[1,4],[3,1]])
a1=np.sort(a)                   # sort along the last axis
print(a1)

x= np.sort(a, axis=None)        # sort the flattened array
print(x)

y= np.sort(a, axis=0)           # sort along the first axis
print(y)
```

```
[[1 4]
 [1 3]]

[1 1 3 4]

[[1 1]
 [3 4]]
```

Array-Oriented Programming with arrays

Sorting: Use the *order* keyword to specify a field to use when sorting a structured array:

```
import numpy as np

dtype = [('name', 'S10'), ('height', float), ('age', int)]
values = [('Arthur', 1.8, 41), ('Lancelot', 1.9, 38),
          ('Galahad', 1.7, 38)]
a = np.array(values, dtype=dtype)
# create a structured array
x= np.sort(a, order='height')
print(x)
```

[('Galahad', 1.7, 38) ('Arthur', 1.8, 41) ('Lancelot', 1.9, 38)]

Array-Oriented Programming with arrays

Sorting: Sort by age, then height if ages are equal

```
import numpy as np

dtype = [('name', 'S10'), ('height', float), ('age', int)]
values = [('Arthur', 1.8, 41), ('Lancelot', 1.9, 38), ('Galahad', 1.7, 38)]
a = np.array(values, dtype=dtype)
 # create a structured array
x= np.sort(a, order=['age', 'height'])
print(x)
```

[('Galahad', 1.7, 38) ('Lancelot', 1.9, 38) ('Arthur', 1.8, 41)]

File Input and Output with arrays

- NumPy allows saving and loading the data to and from disk either in text or binary format.

- np.save and np.load functions are used to save and load data on disk.

- Arrays are saved by default in an uncompressed raw binary format with file extension .npy

File Input and Output with arrays

Example:

```
import numpy as np

a1 = np.random.randn(4)
print(a1)

np.save('array',a1)

x= np.load('array.npy')
print(x)
```

```
[ 0.61733536  1.00148046 -0.73454978 -0.79836939]
```

```
[ 0.61733536  1.00148046 -0.73454978 -0.79836939]
```

File Input and Output with arrays

You can save multiple arrays in uncompressed archive using np.savez and passing the arrays as keywords arguments.

When you load the npz file, you get back a dict-like object that loads individual arrays

File Input and Output with arrays

Example

```
import numpy as np

a1 = np.random.randn(4)
print(a1)

a2 = np.random.randn(4)
print(a2)

np.savez('arrays.npz', x1=a1, x2=a2)
y=np.load('arrays.npz')
print(y['x2'])
```

```
[-0.24600908  1.48196978 -1.03146327 -0.06491125]

[-0.14792378  0.32089439  0.60722349  1.16275722]

[-0.14792378  0.32089439  0.60722349  1.16275722]
```

File Input and Output with arrays

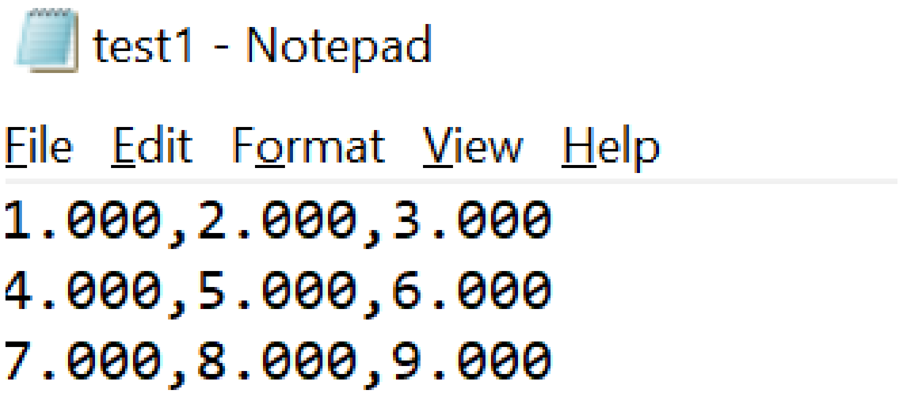
There are other ways for reading from file and writing to data files in numpy :

- savetxt
- loadtxt
- tofile
- fromfile
- …

File Input and Output with arrays

Example:

```
import numpy as np

x = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]], np.int32)

np.savetxt("test1.txt", x, fmt="%2.3f",
delimiter=",")
```


test1 - Notepad

File Edit Format View Help

1.000,2.000,3.000
4.000,5.000,6.000
7.000,8.000,9.000

```
y = np.loadtxt("test1.txt",
delimiter=",")
print(y)
```

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

Linear Algebra

NumPy package contains **numpy.linalg** module that provides all the functionality required for linear algebra.

Linear Algebra

Example: Solve the system of equations

$$3 * x0 + x1 = 9 \text{ and } x0 + 2 * x1 = 8$$

```
import numpy as np
from numpy.linalg import solve

a = np.array([[3,1], [1,2]])
b = np.array([9,8])
x = np.linalg.solve(a, b)
print(x)

rslt=np.allclose(np.dot(a, x), b)
print(rslt)
```

```
[2. 3.]

True
```

Linear Algebra

For more functions : https://numpy.org/doc/stable/reference/routines.linalg.html

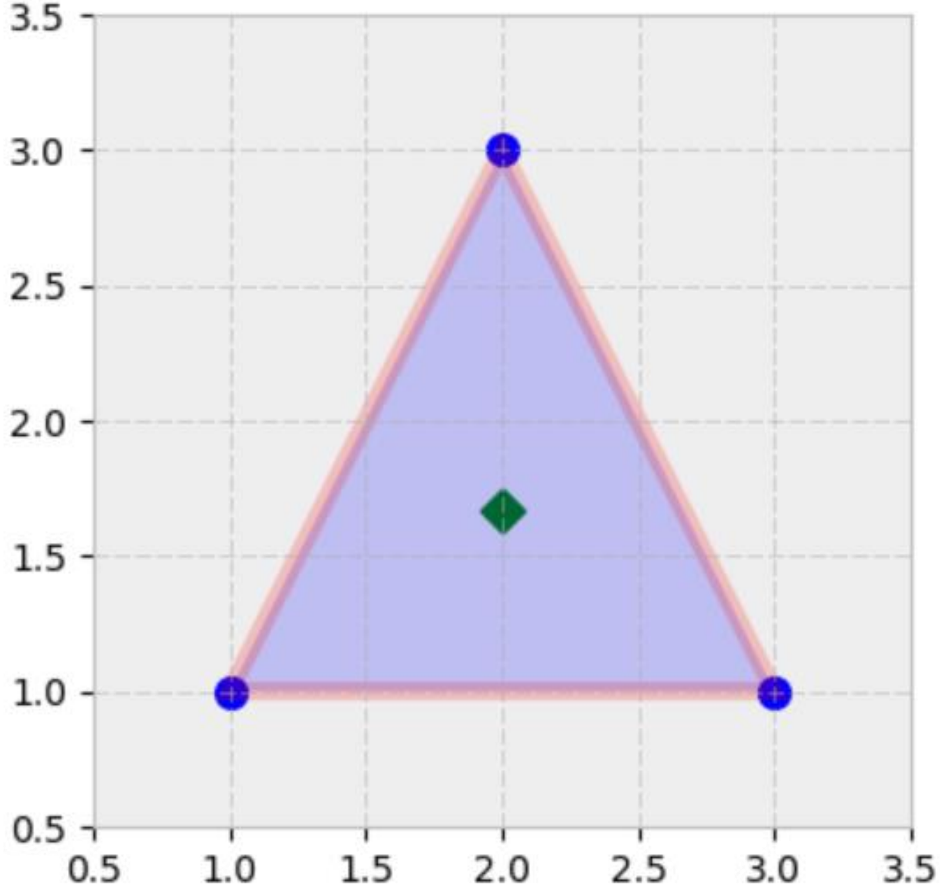
Example of important functions:

dot	Dot product of the two arrays
vdot	Dot product of the two vectors
inner	Inner product of the two arrays
matmul	Matrix product of the two arrays
determinant	Computes the determinant of the array
solve	Solves the linear matrix equation
Inv	Finds the multiplicative inverse of the matrix

Applications: Clustering Algorithms

Example:

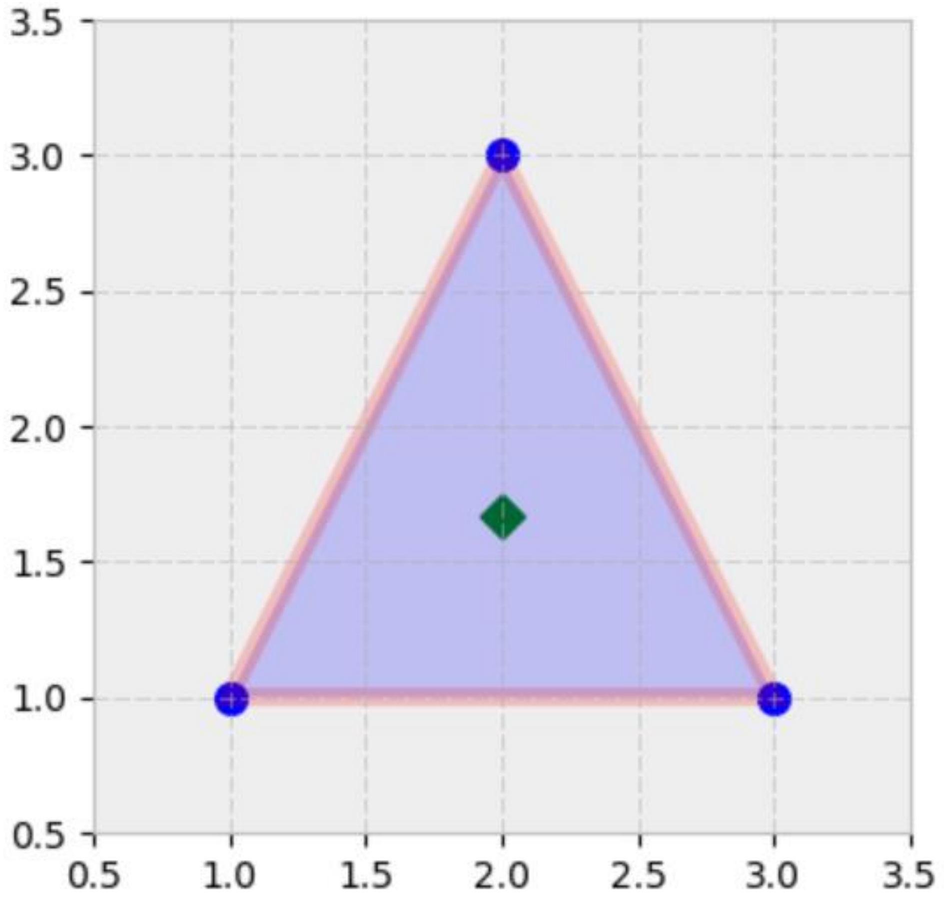
We have the vertices of a triangle (each row is an x,y coordinates) and we want to find the centroid (The centroid of this “cluster” is an *(x, y)* coordinate that is the arithmetic mean of each column)?



Applications: Clustering Algorithms

```python
import numpy as np
import matplotlib as plt
tri = np.array([[1, 1],
                [3, 1],
                [2, 3]])

centroid = tri.mean(axis=0)
print(centroid)
```

[2.   1.66666667]



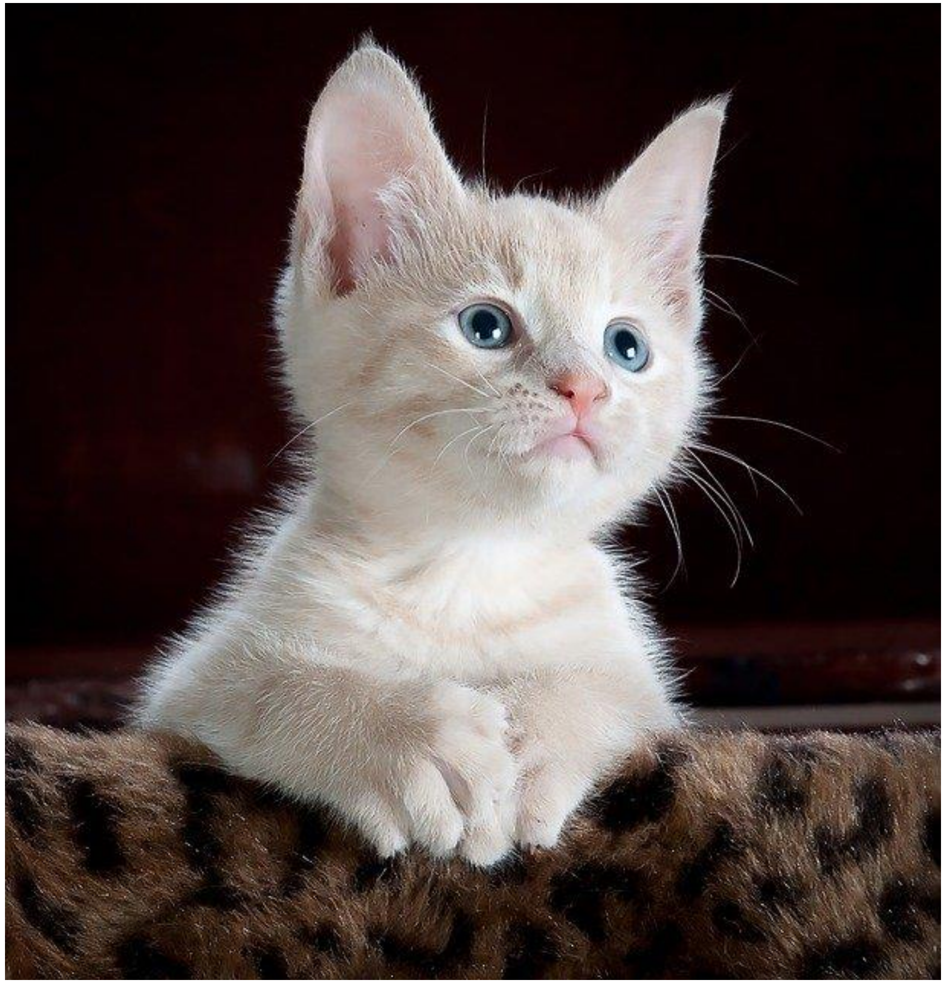
Applications: Image processing

Example:

By storing the images as a numpy arrays, various image processing can be performed using numpy functions:

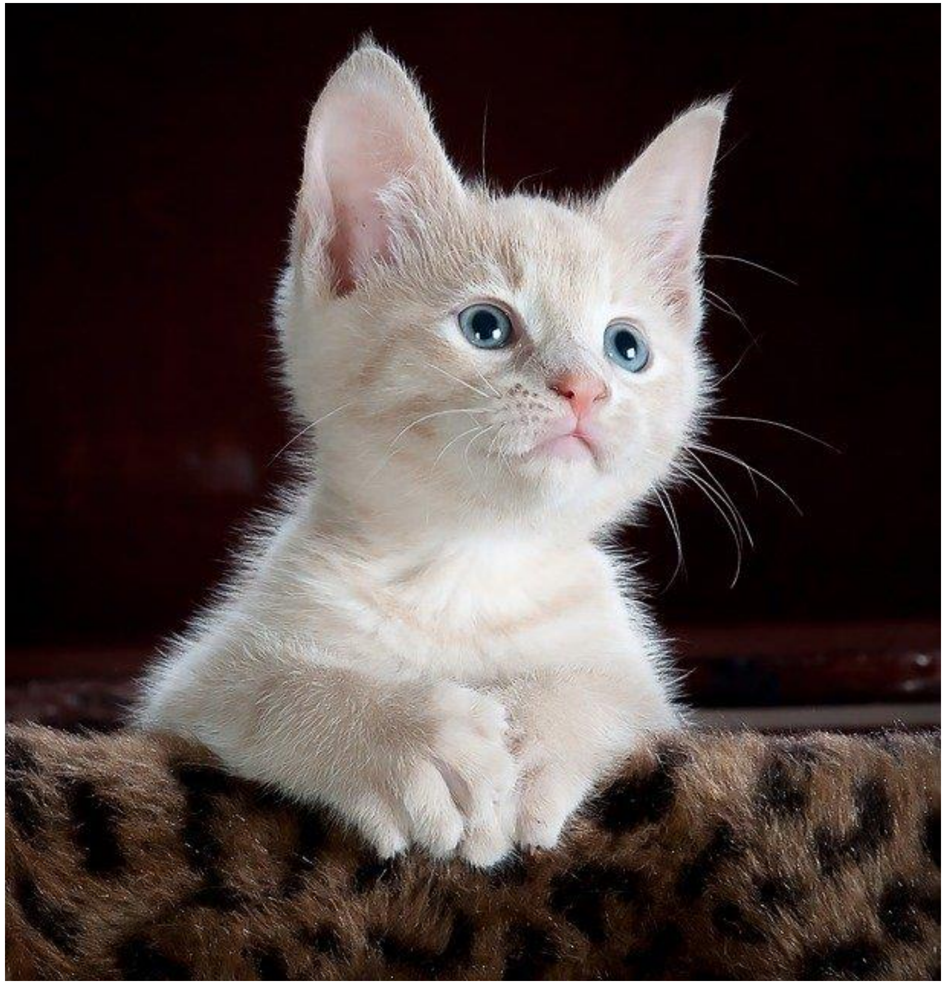
- Generation of single color image and concatenation
- Negative / positive inversion (inversion of pixel value)
- Color reduction
- Binarization
- Gamma correction
- Rotate and flip
- ...

https://note.nkmk.me/en/python-numpy-image-processing/



Applications: Image processing

```
from PIL import Image
import numpy as np

im = np.array(Image.open('img.jpg'))

print(im.dtype)
print(im.ndim)
print(im.shape)
```



```
uint8
3
(640, 613, 3)
```

Applications: Image processing

```
# Generation of single color image and concatenation
im_R = im.copy()
im_R[:, :, (1, 2)] = 0
im_G = im.copy()
im_G[:, :, (0, 2)] = 0
im_B = im.copy()
im_B[:, :, (0, 1)] = 0
im_RGB = np.concatenate((im_R, im_G, im_B), axis=1)
pil_img = Image.fromarray(im_RGB)
pil_img.save('img_color.jpg')
```

Applications: Image processing

```python
# Negative / positive inversion (invert pixel value)
im1 = np.array(Image.open('img.jpg').resize((256, 256)))
im_i = 255 - im1
Image.fromarray(im_i).save('img_inverse.jpg')
```

Applications: Image processing

```
# Color reduction
im_32 = im1 // 32 * 32
im_128 = im1 // 128 * 128
im_dec = np.concatenate((im1, im_32, im_128), axis=1)
Image.fromarray(im_dec).save('img_dec_color.png')
```

Information about the course

MÅN	TIS	ONS	TOR	FRE	LÖR	SÖN
12 10:00 Lectures	13	14 10:00 Lectures 13:15 Labs	15	16 13:15 Labs	17	18 Assignment 7
19 10:00 Lectures	20	21 10:00 Lectures 13:15 Labs	22	23 13:15 Labs	24	25
26	27	28	29	30	31	1 Assignment 8