What is NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Why use NumPy

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

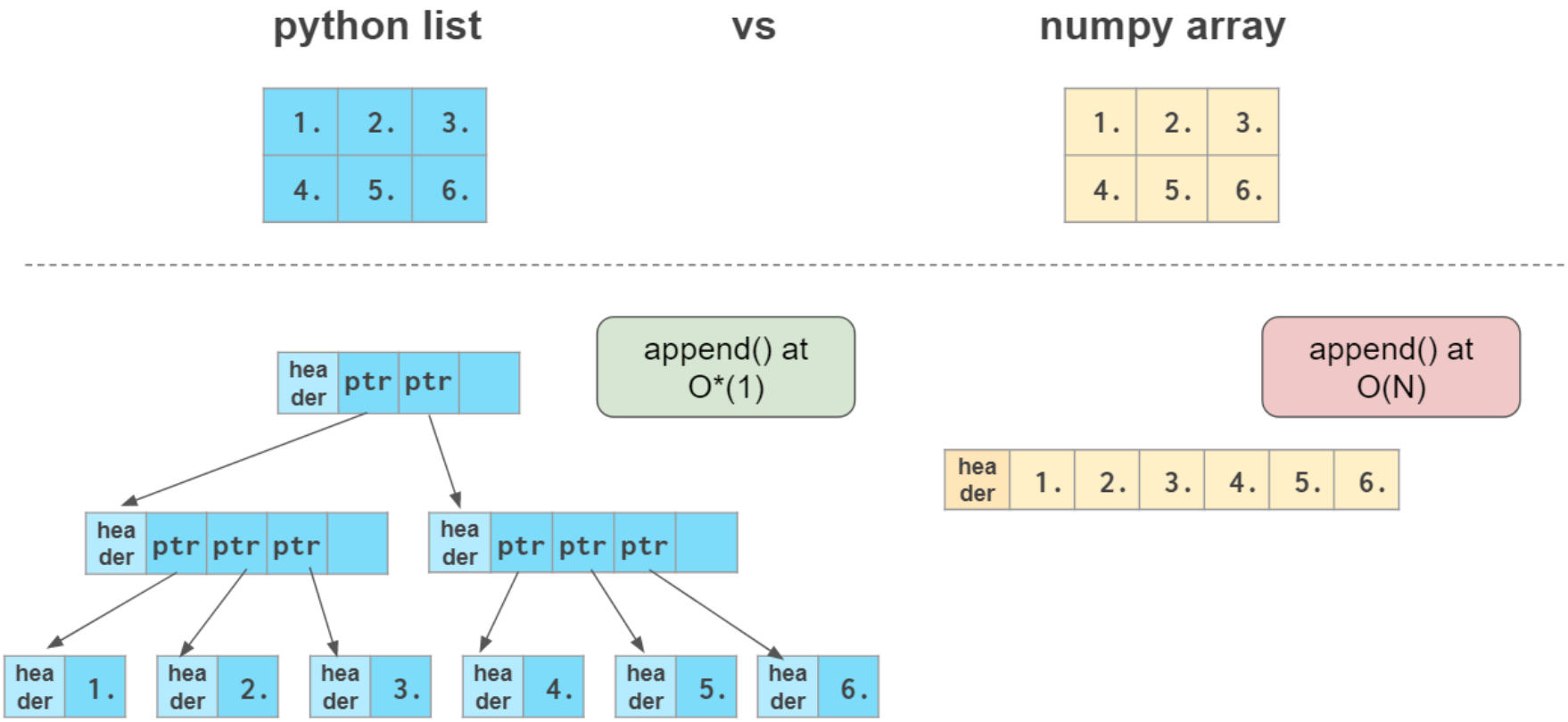
Arrays are very frequently used in data science, where speed and resources are very important.

Why is NumPy Faster Than Lists?

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

This behavior is called locality of reference in computer science.

This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.



Importing NumPy

```
import numpy as np
```

In [1]:
```
a = [1,2,3]
b = [4,5,6]


c=[a[0]+b[0], a[1]+b[1], a[2]+b[2]]
c
```

Out[1]:
```
[5, 7, 9]
```

In [2]:
```
import numpy as np

array_a = np.array(a)  #array_a = np.array([1,2,3])
array_b = np.array(b)  #array_b = np.array([4,5,6])

array_a/array_b
```

Out[2]:
```
array([0.25, 0.4 , 0.5 ])
```

In [5]:
```
a+b
```

Out[5]:
```
[1, 2, 3, 4, 5, 6]
```

In [37]:
```
np.concatenate((array_a, array_b))
```

Out[37]:
```
array([1, 2, 3, 4, 5, 6])
```

Arrays

In [38]:
```
#1-D array
import numpy as np

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

```
print(arr)
```

```
[1 2 3 4 5]
```

In [5]:
```
#2-D array
import numpy as np

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

print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

In [7]:
```
np.sum(arr, axis=1)
```

Out[7]:
```
array([ 6, 15])
```

In [2]:
```
#3-D array

import numpy as np

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

print(arr)
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
```

In [3]:
```
arr = ([1,2,3])
arr
```

Out[3]:
```
[1, 2, 3]
```

Fill the blanks to generate the following Matrix

$$\begin{bmatrix} 1 & 2 & 1 \\ 3 & 0 & 1 \\ 0 & 2 & 4 \end{bmatrix}$$

In [4]:
```
mat = np.array([[1,2,1], [3,0,1], [0,2,4]])
print(mat)
```

```
[[1 2 1]
 [3 0 1]
 [0 2 4]]
```

In [7]:
```
#arange function

a = np.arange(0,10,1)

b = np.arange(0,10,2) #(start, end, skip)

c = np.arange(-10,10, 0.5)

c
```

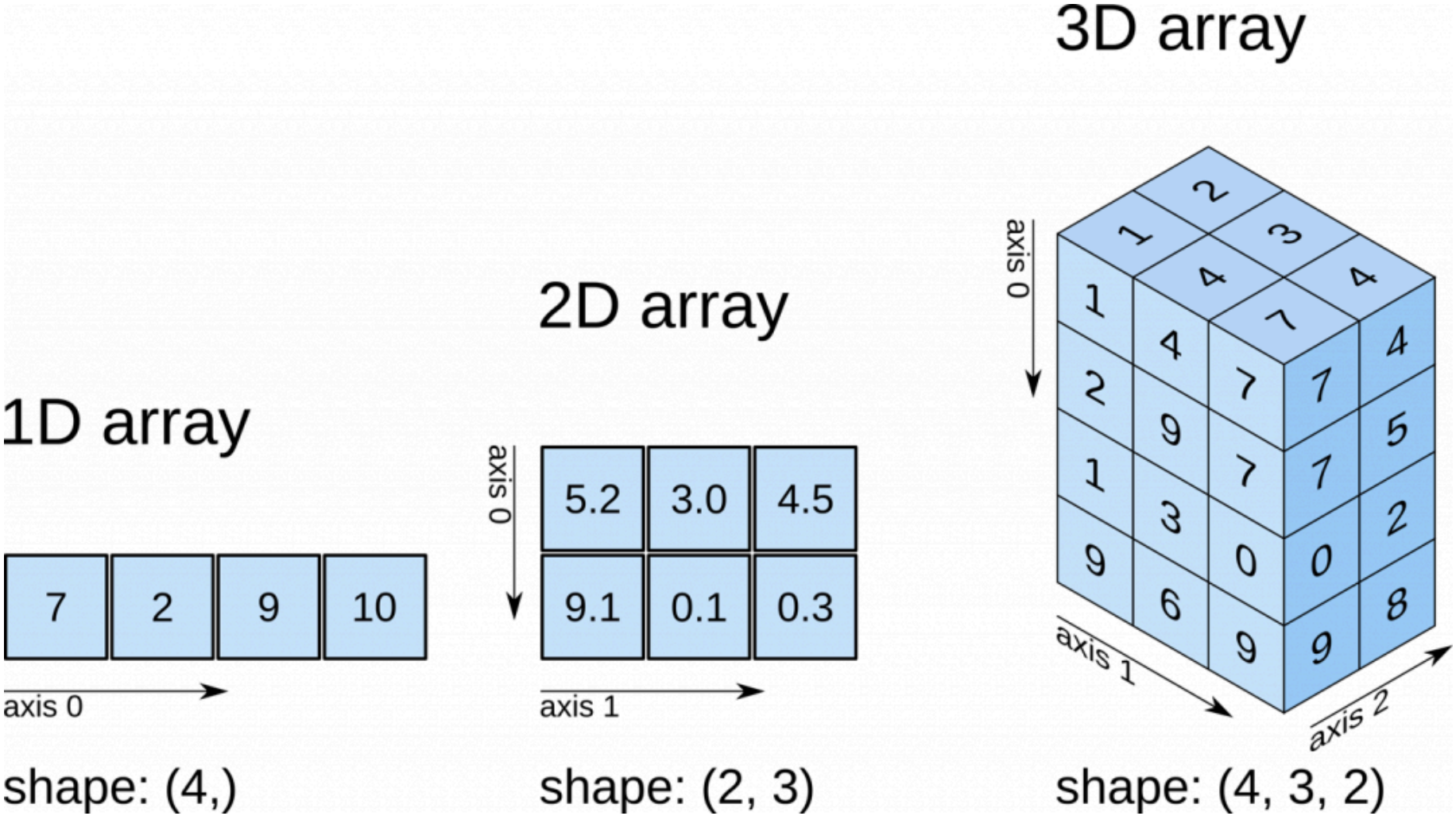
Out[7]:
```
array([-10. ,  -9.5,  -9. ,  -8.5,  -8. ,  -7.5,  -7. ,  -6.5,  -6. ,
       -5.5,  -5. ,  -4.5,  -4. ,  -3.5,  -3. ,  -2.5,  -2. ,  -1.5,
       -1. ,  -0.5,   0. ,   0.5,   1. ,   1.5,   2. ,   2.5,   3. ,
        3.5,   4. ,   4.5,   5. ,   5.5,   6. ,   6.5,   7. ,   7.5,
        8. ,   8.5,   9. ,   9.5])
```

Fill the blank to generate the following vector

$$\begin{bmatrix} -30 & -20 & -10 & 0 & 10 & 20 & 30 & 40 \end{bmatrix}$$

In [8]:
```
p = np.arange( -30, 50, 10) #fill the blank
p
```

Out[8]:
```
array([-30, -20, -10,   0,  10,  20,  30,  40])
```



In [19]:
```
#reshape function

array_reshaped_1 = np.arange(4,12).reshape(2,4)
array_reshaped_1
```

Out[19]:
```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [17]:
```
array_reshaped_1.ndim
```

Out[17]:
```
1
```

In [21]:
```
array_reshaped_1.shape
```

Out[21]:
```
(2, 4)
```

In [22]:
```
array_reshaped_2 = np.arange(4.,12.).reshape(4,2)
array_reshaped_2
```

Out[22]:
```
array([[ 4.,  5.],
       [ 6.,  7.],
       [ 8.,  9.],
       [10., 11.]])
```

In [23]:
```
# data type of arrays and itemsize

print(array_reshaped_1.dtype)

print(array_reshaped_2.dtype)

print(array_reshaped_1.itemsize)
print(array_reshaped_2.itemsize)
```

```
int64
float64
8
8
```

In [27]:
```
#array_int_item = np.array([1,2,3,4,5,6,7,8,9], dtype='int64')
array_int_item=np.arange(1,10, dtype='int16')

#array_float_item = np.array([1,2,3,4,5,6,7,8,9], dtype='float64')
#fill the blank
array_float_item=np.arange(1,10, dtype='float32')

print(array_int_item)
print(array_float_item)

print(array_int_item.dtype)
print(array_float_item.dtype)
```

```
print(array_float_item.itemsize)
```

```
[1 2 3 4 5 6 7 8 9]
[1. 2. 3. 4. 5. 6. 7. 8. 9.]
int64
float32
4
```

Fill the blanks to generate the following matrix

$$\begin{bmatrix} 1.5 & 2.5 & 3.5 \\ 4.5 & 5.5 & 6.5 \\ 7.5 & 8.5 & 9.5 \end{bmatrix}$$

In [28]:
```
float_array = np.arange(1.5, 10.5, 1, dtype='float64').reshape(3,3)

print(float_array)
```

```
[[1.5 2.5 3.5]
 [4.5 5.5 6.5]
 [7.5 8.5 9.5]]
```

In [31]:
```
#array full of zeros and ones

zero_array = np.zeros((2,3), dtype='int64')

one_array = np.ones((2,3), dtype='float64')

full_array = np.full((2,2), 99, dtype='float64')

full_array
```

Out[31]:
```
array([[99., 99.],
       [99., 99.]])
```

In [32]:
```
#randoms

random_array_1 = np.random.rand(4,2)
print(random_array_1)

random_array_2 = np.random.randint(-4,7, size=(3,3))
```

```
print(random_array_2)

random_array_3 = np.random.randint(0, 8, size=(3,3))

print(random_array_3)
```

```
[[0.58309026 0.28679404]
 [0.81718767 0.51384471]
 [0.05989703 0.25553814]
 [0.75812778 0.00150133]]
[[ 3  6  2]
 [ 2  0 -3]
 [ 5  5 -4]]
[[1 5 1]
 [7 6 3]
 [7 2 6]]
```

In [ ]:

access elements

In [38]:

```
a = np.arange(25).reshape(5,5)

print(a)

print(a[:, :])

print(a[0:5, 0:5])

print(a[0:5:1, 0:5:1])
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

In [40]:

```
a = np.arange(25,50).reshape(5,-1)
print(a)
print(a[1:4:2,1:4:2])  #[start:end:skip, start:end:skip]
```

```
[[25 26 27 28 29]
 [30 31 32 33 34]
 [35 36 37 38 39]
 [40 41 42 43 44]
 [45 46 47 48 49]]
[[31 33]
 [41 43]]
```

In [42]:

```
check_reshape = np.arange(25,50).reshape(5,5)
print(check_reshape)
print(check_reshape.shape)

reshaped = check_reshape.reshape(1,25)

print(reshaped)
print(reshaped.shape)
```

```
[[25 26 27 28 29]
 [30 31 32 33 34]
 [35 36 37 38 39]
 [40 41 42 43 44]
 [45 46 47 48 49]]
(5, 5)
[[25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
  49]]
(1, 25)
```

fill the blanks to print the following matrix

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 9 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

In [46]:

```
a = np.ones((5,5), dtype='int64')
b = np.zeros((3,3))
b[1,1]=9   #fill this blank
a[1:-1, 1:-1]=b   #fill this blank
a
```

Out[46]:

```
array([[1, 1, 1, 1, 1],
       [1, 0, 0, 0, 1],
       [1, 0, 9, 0, 1],
       [1, 0, 0, 0, 1],
       [1, 1, 1, 1, 1]])
```

Mathematics

In [48]:

```
a = np.array([1,2,3,4], dtype='int64')

print(a+2)
print(a*2)
print(a**2)
print(a/2)
```

```
[3 4 5 6]
[2 4 6 8]
[ 1  4  9 16]
[0.5 1.  1.5 2. ]
```

In [49]:
```
angles = np.arange(30,360,30)*np.pi/180 #angles in radians..

sin_values = np.sin(angles)
sin_values
```

Out[49]:
```
array([ 5.00000000e-01,  8.66025404e-01,  1.00000000e+00,  8.66025404e-01,
        5.00000000e-01,  1.22464680e-16, -5.00000000e-01, -8.66025404e-01,
       -1.00000000e+00, -8.66025404e-01, -5.00000000e-01])
```

In [51]:
```
a = np.arange(20,30).reshape(2,5)

b = np.arange(30,40).reshape(5,2)

c = np.matmul(a,b)

print(np.linalg.det(c))
```

```
499.9999999987884
```

In [52]:
```
print(np.linalg.eig(c))

print(np.linalg.norm(c))

print(np.linalg.matrix_rank(c))
```

```
(array([5.87893540e-02, 8.50494121e+03]), array([[-0.7172319 , -0.63204111],
       [ 0.69683456, -0.77493486]]))
8537.073561824333
2
```

In [55]:
```
a = np.array([[1,3], [0,1]])
b = np.array([[1,-1], [0,1]])

print(np.linalg.solve(a,b))  #ax=b => x=b*a^-1

print(np.matmul(np.linalg.inv(a), b))
print(a)
print(b)
```

```
[[ 1. -4.]
 [ 0.  1.]]
[[ 1. -4.]
 [ 0.  1.]]
[[1 3]
 [0 1]]
[[ 1 -1]
 [ 0  1]]
```

In [4]:

```
import numpy as np

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
iris_data = np.genfromtxt(url, delimiter=',', dtype='float', usecols=[0,1,2,3])

# Solution
iris_data[(iris_data[:, 2] > 1.5) & (iris_data[:, 0] < 5.0)]
```

Out[4]:

```
array([[4.8, 3.4, 1.6, 0.2],
       [4.8, 3.4, 1.9, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [4.9, 2.4, 3.3, 1. ],
       [4.9, 2.5, 4.5, 1.7]])
```

In [7]:

```
n = np.array([1,2,3,4])
n = n.reshape((2,-1))
n
```

Out[7]:

```
array([[1, 2],
       [3, 4]])
```