# Apa itu Numpy?

> NumPy (Numerical Python) adalah library Python yang fokus pada scientific computing.
>
> NumPy memiliki kemampuan untuk membentuk objek N-dimensional array, yang mirip dengan list pada Python.
>
> Keunggulan NumPy array dibandingkan dengan list pada Python adalah konsumsi memory yang lebih kecil serta runtime yang lebih cepat.
>
> NumPy juga memudahkan kita pada Aljabar Linear, terutama operasi pada Vector (1-d array) dan Matrix (2-d array).

```
In [1]:  # list pada python
         a = [1, 2, 3]
         b = [4, 5, 6]
         a + b
Out[1]:  [1, 2, 3, 4, 5, 6]
```

```
In [2]:  # penjumlahan dilakukan iterasi
         result = []
         for first, second in zip(a, b):
             result.append(first + second)
         result
Out[2]:  [5, 7, 9]
```

## Contoh" Penggunaan Numpy

1. Membuat Array

```
In [3]:  import numpy as np
```

```
In [4]:  # membuat array
         a = np.array([1, 2, 3])
         a
Out[4]:  array([1, 2, 3])
```

1. Cek Tipe

```
In [6]:  # mengecek tipe data array
         type(a)
Out[6]:  numpy.ndarray
```

1. Jumlah Dimensi

```
In [34]:  # cek jumlah dimensi pada array
          a = np.array([1, 2, 3])
          a.ndim
Out[34]:  1
```

2. Array Shape

```
In [38]:  # cek shape dari array
          a = np.array([1, 2, 3])
          a.shape
Out[38]:  (3,)
```

1. Operasi Pada Array

```
In [39]:  a = np.array([1, 2, 3])
          f = np.array([1.1, 2.2, 3.3])
          a + f
Out[39]:  array([2.1, 4.2, 6.3])
```

```
In [40]:  a * f
Out[40]:  array([1.1, 4.4, 9.9])
```

```
In [41]:  a ** f
Out[41]:  array([ 1.        ,  4.59479342, 37.5405076 ])
```

```
In [42]:  # universal function (ufuncs) pada numpy seperti sin
          np.sin(a)
Out[42]:  array([0.84147098, 0.90929743, 0.14112001])
```

1. Element Array

```
In [32]:  # mengakses elemen pada indeks ke-0
          a = np.array([1, 2, 3])
          a[0]
Out[32]:  1
```

```
In [33]:  # melakukan assign pada element array dengan element baru
          a[0] = 10
          a
Out[33]:  array([10,  2,  3])
```

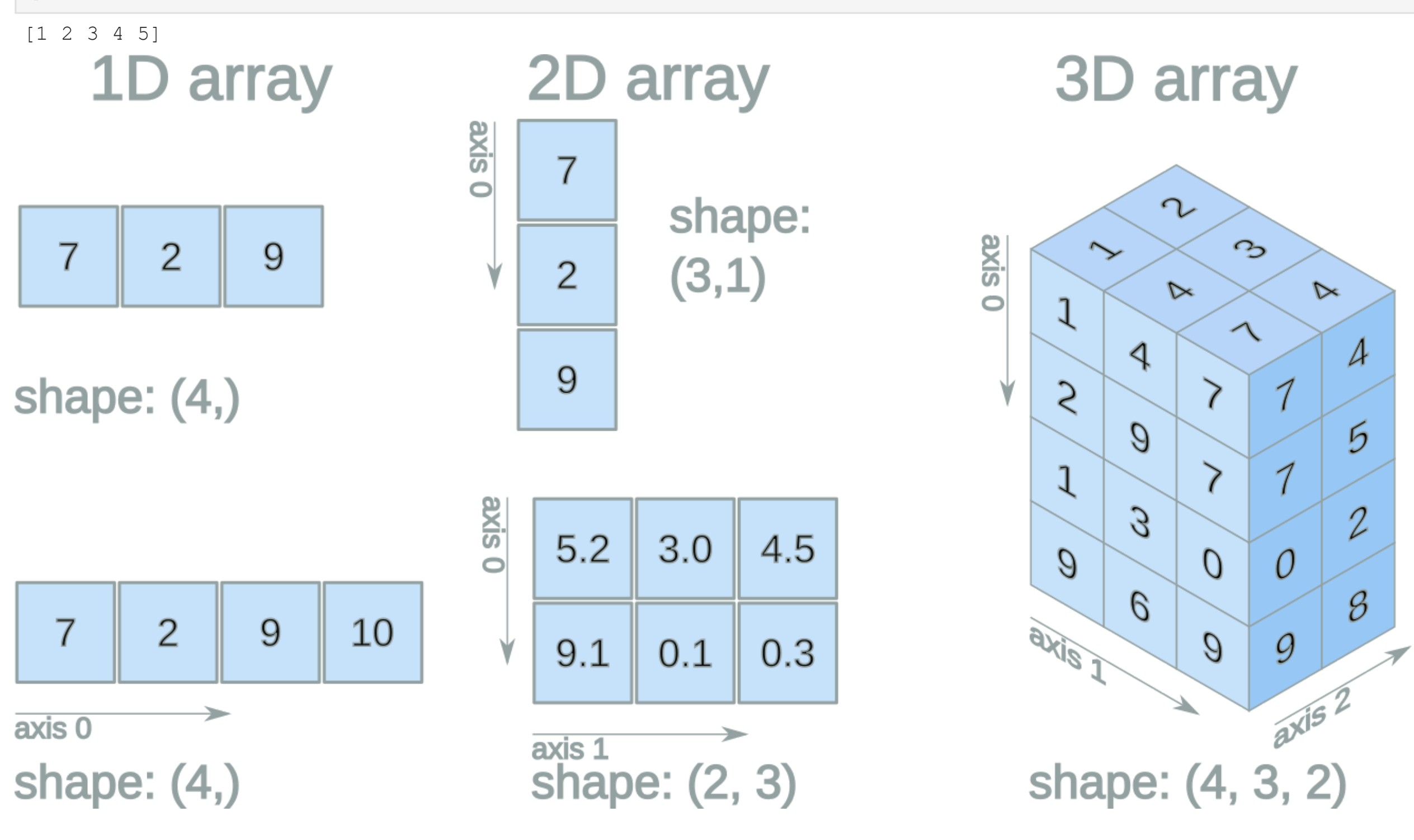
1. Multidimensional Arrays

```
In [45]:  # membentuk array 2-dimensi
          a = np.array([[ 0,  1,  2,  3],
                        [10, 11, 12, 13]])
          a
Out[45]:  array([[ 0,  1,  2,  3],
                 [10, 11, 12, 13]])
```

## Membuat Array

```
In [1]:  import numpy as np

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

         print(arr)

         print(type(arr))

[1 2 3 4 5]
<class 'numpy.ndarray'>
```

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

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

         print(arr)

[1 2 3 4 5]
```



## 1-D Array

```
In [3]:  import numpy as np

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

         print(arr)

[1 2 3 4 5]
```

## 2-D Array

```
In [4]:  import numpy as np

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

         print(arr)

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

```
In [5]:  # Using above first method to create a
         # 2D array
         rows, cols = (5, 5)
         arr = [[0]*cols]*rows
         print(arr)

[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

```
In [6]:  # Using above second method to create a
         # 2D array
         rows, cols = (5, 5)
         arr=[]
         for i in range(cols):
             col = []
             for j in range(rows):
                 col.append(0)
             arr.append(col)
         print(arr)

[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

```
In [7]:  import numpy as np

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

         for x in arr:
           print(x)

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

## 3-D Array

```
In [8]:  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]]]
```

## Reshape

```
In [9]:  # Reshape From 1-D to 2-D
         import numpy as np

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

         newarr = arr.reshape(4, 3) # 4 baris, 3 kolom

         print(newarr)

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

```
In [10]:  # Reshape From 1-D to 3-D
          import numpy as np

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

          newarr = arr.reshape(2, 3, 2)

          print(newarr)

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

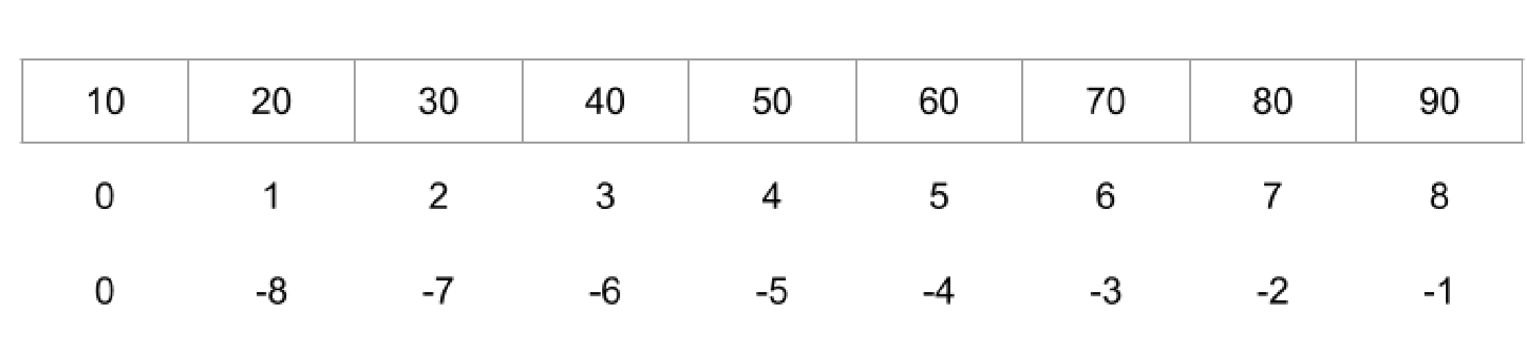
## Check Dimension

```
In [11]:  import numpy as np

          a = np.array(42)
          b = np.array([1, 2, 3, 4, 5])
          c = np.array([[1, 2, 3], [4, 5, 6]])
          d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

          print(a.ndim)
          print(b.ndim)
          print(c.ndim)
          print(d.ndim)

0
1
2
3
```

## Higher Dimensional Arrays

```
In [12]:  import numpy as np

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

          print(arr)
          print('number of dimensions :', arr.ndim)

[[[[[1 2 3 4]]]]]
number of dimensions : 5
```

## Slicing Array



| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

```
In [13]:  # Slice elements from index 1 to index 5 from the following array:
          import numpy as np

          arr = np.array([1, 2, 3, 4, 5, 6, 7])

          print(arr[1:5])

[2 3 4 5]
```

```
In [14]:  # Slice elements from index 4 to the end of the array:
          import numpy as np

          arr = np.array([1, 2, 3, 4, 5, 6, 7])

          print(arr[4:])

[5 6 7]
```

```
In [15]:  # Slice elements from the beginning to index 4 (not included):
          import numpy as np

          arr = np.array([1, 2, 3, 4, 5, 6, 7])

          print(arr[:4])

[1 2 3 4]
```

```
In [16]:  # Slice from the index 3 from the end to index 1 from the end:

          import numpy as np

          arr = np.array([1, 2, 3, 4, 5, 6, 7])

          print(arr[-3:-1])

[5 6]
```

```
In [17]:  # Return every other element from index 1 to index 5:

          import numpy as np

          arr = np.array([1, 2, 3, 4, 5, 6, 7])

          print(arr[1:5:2])

[2 4]
```

## Slicing 2D Array

```
In [18]:  #From the second element, slice elements from index 1 to index 4 (not included):
          import numpy as np

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

          print(arr[1, 1:4])

[7 8 9]
```

```
In [19]:  #From both elements, return index 2:
          import numpy as np

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

          print(arr[0:2, 2])

[3 8]
```

```
In [20]:  #From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:
          import numpy as np

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

          print(arr[0:2, 1:4])

[[2 3 4]
 [7 8 9]]
```

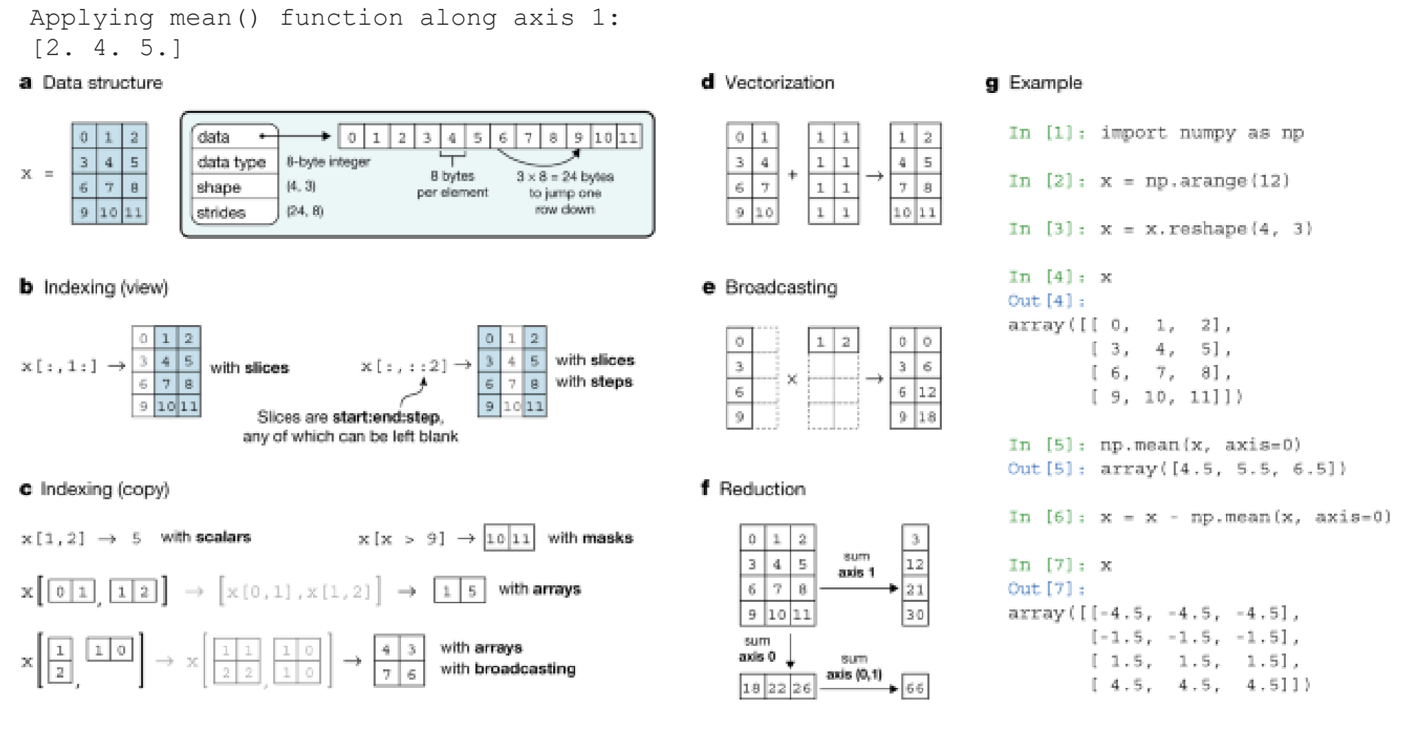
## Array for Statistics

```
In [21]:  import numpy as np
          a = np.array([[3,7,5],[8,4,3],[2,4,9]])

          print('Our array is:')
          print(a)
          print('\n')

          # BARIS
          print('Applying amin() function:')
          print(np.amin(a, 1))
          print('\n')

          # KOLOM
          print('Applying amin() function again:')
          print(np.amin(a, 0))
          print('\n')

          print('Applying amax() function:')
          print(np.amax(a))
          print('\n')

          # KOLOM
          print('Applying amax() function again:')
          print(np.amax(a, axis = 0))

Our array is:
[[3 7 5]
 [8 4 3]
 [2 4 9]]


Applying amin() function:
[3 3 2]


Applying amin() function again:
[2 4 3]


Applying amax() function:
9


Applying amax() function again:
[8 7 9]
```

```
In [22]:  import numpy as np
          a = np.array([[30,40,70],[80,20,10],[50,90,60]])

          print('Our array is:')
          print(a)
          print('\n')

          print('Applying percentile() function:')
          print(np.percentile(a, 50))
          print('\n')

          # BARIS
          print('Applying percentile() function along axis 1:')
          print(np.percentile(a, 50, axis = 1))
          print('\n')

          # KOLOM
          print('Applying percentile() function along axis 0:')
          print(np.percentile(a, 50, axis = 0))

Our array is:
[[30 40 70]
 [80 20 10]
 [50 90 60]]


Applying percentile() function:
50.0


Applying percentile() function along axis 1:
[40. 20. 60.]


Applying percentile() function along axis 0:
[50. 40. 60.]
```

```
In [23]:  import numpy as np
          a = np.array([[30,65,70],[80,95,10],[50,90,60]])

          print('Our array is:')
          print(a)
          print('\n')

          print('Applying median() function:')
          print(np.median(a))
          print('\n')

          # KOLOM
          print('Applying median() function along axis 0:')
          print(np.median(a, axis = 0))
          print('\n')

          # BARIS
          print('Applying median() function along axis 1:')
          print(np.median(a, axis = 1))

Our array is:
[[30 65 70]
 [80 95 10]
 [50 90 60]]


Applying median() function:
65.0


Applying median() function along axis 0:
[50. 90. 60.]


Applying median() function along axis 1:
[65. 80. 60.]
```

```
In [24]:  import numpy as np
          a = np.array([[1,2,3],[3,4,5],[4,5,6]])

          print('Our array is:')
          print(a)
          print('\n')

          print('Applying mean() function:')
          print(np.mean(a))
          print('\n')

          print('Applying mean() function along axis 0:')
          print(np.mean(a, axis = 0))
          print('\n')

          print('Applying mean() function along axis 1:')
          print(np.mean(a, axis = 1))

Our array is:
[[1 2 3]
 [3 4 5]
 [4 5 6]]


Applying mean() function:
3.6666666666666665


Applying mean() function along axis 0:
[2.66666667 3.66666667 4.66666667]


Applying mean() function along axis 1:
[2. 4. 5.]
```



```
In [1]: import numpy as np

In [2]: x = np.arange(12)

In [3]: x = x.reshape(4, 3)

In [4]: x
Out[4]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

In [5]: np.mean(x, axis=0)
Out[5]: array([4.5, 5.5, 6.5])

In [6]: x = x - np.mean(x, axis=0)

In [7]: x
Out[7]:
array([[-4.5, -4.5, -4.5],
       [-1.5, -1.5, -1.5],
       [ 1.5,  1.5,  1.5],
       [ 4.5,  4.5,  4.5]])
```