Pemrograman Sains Data

Library Numpy

Oleh :

Fakhrian Fadlia Adiwijaya

Library **Numpy**

Numpy merupakan library penting dalam Python yang khsusu diperntukkan untuk komputasi saintifik. Library ini menyediakan fitur yang dapat mendukung **komputasi data bertipe array multidimensi dengan performa sangat baik.**

Untuk menggunakan module numpy, pastikan module ini sudah dipasang di lingkungan Python yang digunakan. Perintah instalasinya sebagai berikut :

Installasi via pip

```
pip install numpy
```

Installasi via conda

```
conda install numpy
```

Menggunakan **Numpy**

Untuk bisa mendefinisikan suatu array dan pengolahannya menggunakan Numpy, terlebih dahulu harus dilakukan import library dengan perintah

```
import numpy as np
```

Numpy vs **List**

Secara karakteristik, tipe data array sangat mirip dengan list namun perbedaannya pada tipe datanya. Jika dalam list, elemen-elemennya dapat tidak sama tipe datanya, namun pada array semuanya harus sama tipe datanya.

Selain itu, perbedaannya juga terletak pada fitur komputasi yang mendukungnya. Sebagai contoh misalnya ada sebuah data list [2, 4, 6, 8]. Kemudian misalkan kita menginginkan diperoleh list baru [1, 2, 3, 4] yang elemennya merupakan hasil pembagian dengan 2 dari data awalnya. Hal ini tidak bisa dilakukan melalui perintah sederhana seperti berikut :

```
a = [2, 4, 6, 8]
b = a/2
print(b)
```

Perintah di atas akan menyebabkan error seperti berikut

```
Traceback (most recent call last):
  File "c:\Users\Fakhrian Fadlia A\Desktop\Test.py", line 2, in <module>
    b = a/2
TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

Numpy vs **List**

Untuk menghasilkan list baru yang di inginkan tersebut, maka langkah yang dilakukan tidak sesederhana itu, melainkan harus melalui perintah yang agak panjang, yaitu :

```
a = [2, 4, 6, 8]
b = []
for x in a :
    b.append(x/2)
print(b)
```

Outputnya : **[1.0, 2.0, 3.0, 4.0]**

Namun dengan menggunakan tipe data array, komputasi di atas cukup dilakukan dengan perintah yang sederhana, yaitu :

```
a = np.array([2, 4, 6, 8])
b = a/2
print(b)
```

Membuat **Array**

Pendefinisian **Array**

Untuk membuat array dengan dimensi tertentu menggunakan Numpy digunakan perintah **np.array()**. Berikut beberapa contoh cara membuat array

```
a = np.array([1, 2, 3, 4])
print(a)
```

```
Outputnya :
[1 2 3 4]
```

```
a = np.array([1, 2.1, 3, 4])
print(a)
```

```
Outputnya :
[1. 2.1 3. 4.]
```

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

```
Outputnya :
[1. 2. 3. 4.]
```

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

```
Outputnya :
[[1 2]
 [3 4]]
```

Fungsi **Array Khusus**

Di dalam NumPy juga terdapat beberapa *function* untuk membuat array khusus. Berikut ini beberapa perintah untuk membuat array khusus.

np.zeros()

Perintah **np.zeros()** digunakan untuk membuat array nol dua dimensi. Secara default tipe data dari array nol adalah **float**. Namun bisa juga nol di sini dibuat dalam tipe ***integer*** dengan menambahkan parameter **dtype=int**

```
a = np.zeros((3, 4))
print(a)
```

```
Outputnya :
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Fungsi **Array Khusus**

np.ones()

Untuk mendefinisikan sebuah array dengan semua elemennya bernilai 1 juga bisa dibuat dengan perintah **np.ones**().

Berikut ini adalah contoh untuk membuat array dua dimensi dengan elemen 1 bertime *integer* berukuran 3x3.

```
a = np.ones((3, 3), dtype=int)
print(a)
```

```
Outputnya :
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

Fungsi **Array Khusus**

np.linspace()

Perintah **linspace()** digunakan untk men-generate sebuah deret array dengan jumlah interval yang sama. Sebagai contoh di bawah ini kita akan definisikan 5 buah bilangan dari 0 sampai dengan 5. jarak antar bilangan dalam array ini adalah sama.

```
a = np.linspace(0, 2, 5)
print(a)
```

```
Outputnya :
[0. 0.5 1. 1.5 2. ]
```

Fungsi **Array Khusus**

np.full()

Sebuah array berisi bilangan konstan yang sama dapat pula dibuat, yaitu menggunakan perintah **np.full()**. Berikut adalah contoh untuk membuat array dua dimensi berukuran 3x3 dengan elemennya berupa bilangan 2 semuanya.

```
a = np.full((3, 3), 2)
print(a)
```

```
Outputnya :
[[2 2 2]
 [2 2 2]
 [2 2 2]]
```

Fungsi **Array Khusus**

np.eye()

Di dalam aljabar linear, sering menggunakan matriks identitas dalam komputasinya. Matriks identitas berukuran m x n ini dapat dibuat dengan menggunakan perintah **np.eye().** Berikut ini contoh perintah untuk membuat matriks identitas berukuran 3x3

```
a = np.eye(3, dtype=float)
print(a)
```

```
Outputnya :
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Fungsi **Array Khusus**

np.random()

NumPy juga bisa digunakan untuk men-generate sebuah array yang berisikan bilangan random antara 0 dan 1. adapun caranya adalah dengan menggunakan perintah **np.random.random().** Sebagai contoh kita buat matriks random 2 x 3

```
a = np.random.random((2,3))
print(a)
```

```
Outputnya :
[[0.71631299 0.76856787 0.80889282]
 [0.59964015 0.15309833 0.06768213]]
```

Menyimpan Array

Menyimpan **Array ke File**

Sebuah array yang sudah dibuat dapat disimpan secara permanen di dalam komputer. Sehingga suatu saat bisa dipanggil kembali untuk proses komputasi apabila diperlukan. Cara untuk menyimpan suatu array ke dalam file adalan menggunakan perintah **np.save(nama file, nama variabel array).** Contoh :

```
np.save('E:\myarray',a)
```

Perintah di atas digunakan untuk menyimpa array a yang sudah didefinisikan sebelumnya ke dalam file dengan nama myarray.npy. Dalam hal ini, secara default ekstensi nama file hasil penyimpanan array adalah *.npy

Selanjutnya, untuk membaca file berisi array yang sebelumnya sudah tersimpan kita gunakan perintah **np.load(nama file).** Contoh :

```
b = np.load('E:\myarray.npy')
print(b)
```

```
Outputnya :
[[0.71631299 0.76856787 0.80889282]
 [0.59964015 0.15309833 0.06768213]]
```

Karakteristik **Array**

Melihat **Karakteristik Array**

Untuk keperluan komputasi, terkadang perlu untuk mengetahui karakteristik suatu array khususnya array berukuran besar. Misalnya adalah berapa dimensinya, berapa ukurannya, jumlah semua elemen, apa tipe datanya dan sebagainya. Dengan NumPy, karakteristik suatu array dapat diketahui dengan mudah.

Sebagai contoh kita buat array **a** yang memiliki ukuran 3x3, dimana elemennya di generate secara random dengan *function* **random()**

```
a = np.random.random((3,3))
print(a)
```

```
Outputnya :
[[0.60154286 0.65034401 0.60490856]
 [0.37281955 0.63523491 0.2091052 ]
 [0.30058646 0.32027647 0.22594052]]
```

Melihat **Karakteristik Array**

shape : Untuk mengetahui ukuran dari array

```
print(a.shape)
```

```
Outputnya :
(3, 3)
```

ndim : Untuk melihat dimensi dari array

```
print(a.ndim)
```

```
Outputnya :
2
```

size : Untuk mengetahui total banyaknya seluruh elemen dari suatu array berapapun ukuran dan dimensinya

```
print(a.size)
```

```
Outputnya :
9
```

len : Alternatif mengetahui total banyaknya seluruh elemen dari suatu array 1 dimensi selain menggunakan **size**

```
b = np.linspace(0, 1, 6)
print(b)
```

```
Outputnya :
[0.  0.2 0.4 0.6 0.8 1. ]
```

```
print(len(b))
```

```
Outputnya :
6
```

Aritmatika Array

Aritmatika **Array**

Sebuah array atau beberapa array dapat diberikan suatu operasi aritmatika, seperti : penjumlahan, pengurangna, perkalian, pembagian dan sebagainya. Kita akan coba beberapa operasi aritmatika pada array, pertama kita akan buat array 2 dimensi yang masing-masing berukuran 3x3 yaitu a dan b sebagai berikut :

```
a = np.array([[1, 2, 3],[3, 4, 5],[6, 7, 8]])
b = np.array([[1, 1, 2],[1, 2, 1],[2, 1, 1]])
print(a)
print(b)
```

```
Outputnya :
[[1 2 3]
 [3 4 5]
 [6 7 8]]
[[1 1 2]
 [1 2 1]
 [2 1 1]]
```

Selanjutnya dua array tersebut yang akan kita gunakan pada pembahasan aritmatika array

Pengurangan **Array**

Dua buah array dapat dilakukan operasi pengurangan pada setiap elemen yang bersesuaian. Pengurangan ini dapat dilakukan dengan syarat ukuran dari dua array tersebut adalah sama. Sebagai contoh, misalnya akan dilakukan pengurangan dua array a dan b yang sudah diketahui sebelumnya. Perintah untuk melakukannya adalah sebagai berikut

```
print(a-b)
```

```
Outputnya :
[[0 1 1]
 [2 2 4]
 [4 6 7]]
```

Konsep pengurangan secara aritmatika pada array ini sama seperti pada pengurangna vector atau matriks, yaitu pengurangan dilakukan pada setiap elemen yang bersesuaian letaknya.

Penjumlahan **Array**

Sama seperti pada operasi pengurangan, dimana dua buah array yang dapat dilakukan penjumlahan (pada setiap elemen) harus memiliki ukuran yang sama / bersesuaian. Perintah untuk melakukan penjumlahan adalah sebagai berikut :

```
print(a+b)
```

```
Outputnya :
[[2 3 5]
 [4 6 6]
 [8 8 9]]
```

Pembagian **Array**

Pembagian dua array untuk setiap elemennya, untuk perintahnya sebagai berikut

```
print(a/b)
```

```
Outputnya :
[[1.  2.  1.5]
 [3.  2.  5. ]
 [3.  7.  8. ]]
```

Suatu array juga bisa dilakukan pembagian dengan suatu scalar atau bilangan. Misalnya kita bagi dengan 2

```
print(a/2)
```

```
Outputnya :
[[0.5 1.  1.5]
 [1.5 2.  2.5]
 [3.  3.5 4. ]]
```

Perkalian **Array**

Perkallian dua buah array dapat dilakukan dengan NumPy. Namun, perkalian yang dimaksud disini bukan seperti perkalian antara 2 matriks seperti yang dikelnal dalam aljabar linear.

Perkalian yang dimaksud adalah perkalian antar elemen yang bersesuaian dalam dua array. Secara konsep perkalian disini sama seperti penjumlahan, pengurangan, dan pembagian antar dua array.

```
print(a*b)
```

```
Outputnya :
[[ 1  2  6]
 [ 3  8  5]
 [12  7  8]]
```

Perkalian **Array**

Jika kita ingin melakukan perkalian dua array seperti pada perkalian dua buah matriks dalam aljabar linear, perintah yang digunakan adalah sebagai berikut :

```
print(a.dot(b))
```

```
Outputnya :
[[ 9  8  7]
 [17 16 15]
 [29 28 27]]
```

Akar Kuadrat **Array**

Untuk menghitung nilai akar kuadrat pada elemen array, perintah yang digunakan adalah sebagai berikut :

```
print(np.sqrt(a))
```

```
Outputnya :
[[1.         1.41421356 1.73205081]
 [1.73205081 2.         2.23606798]
 [2.44948974 2.64575131 2.82842712]]
```

Nilai Trigonometri **Array**

Untuk menghitung nilai trigonometri, perintah yang digunakan sebagai berikut :

```
print(np.sin(a))
```

```
Outputnya :
[[ 0.84147098  0.90929743  0.14112001]
 [ 0.14112001 -0.7568025  -0.95892427]
 [-0.2794155   0.6569866   0.98935825]]
```

Selain **sin()**, *function* trigonometri lainnya yang bisa digunakan adalah **cos(),** dan **tan()**

Nilai Log **Array**

Untuk menghitung nilai log, perintah yang digunakan sebagai berikut :

```
print(np.log(a))
```

```
Outputnya :
[[0.         0.69314718 1.09861229]
 [1.09861229 1.38629436 1.60943791]
 [1.79175947 1.94591015 2.07944154]]
```

Untuk perintah yang lain, kalian bisa explore di https://numpy.org/doc/stable/reference/index.html#reference

Perbandingan **Array**

Perbandingan **Array**

Dua buah array dapat dibandingkan antar elemen yang bersesuaian di dalam keduanya. Selain itu dapat pula dibandingkan apakah dua array itu sama atau tidak. Seperti yang sudah diketahui, bahwa setiap perbandingan akan menghasilkan nilai berupa Boolean (**true** atau **false**). Demikian juga untuk perbandingan array ini.

Dari kedua array yang sudah kita definisikan sebelumnya, kita akan melakukan perbandingan setiap elemen yang bersesuaian letaknya. Perintah untuk melakukannya adalah sebagai berikut :

```
a == b
```

```
Outputnya :
array[[ True, False, False]
      [False, False, False]
      [False, False, False]]
```

Perbandingan **Array**

Perintah pada slide sebelumnya dimaksudkan untuk membandingan apakah setiap elemen dari array **a** dan elemen yang bersesuaian di array b keduanya sama. Jika sama, maka hasilnya True, sedangkan jika tidak sama hasilnya False.

Operator perbandingan lainnya yang bisa digunakan adalah >, <, >=, <=.

Sedangkan untuk membandingkan sama tidaknya dua array secara keseluruhan dapat menggunakan perintah **array_equal()** sebagai berikut :

```
np.array_equal(a,b)
```

```
Outputnya :
False
```

Agregat Array

Agregat **Array**

Di dalam NumPy juga tersedia *function-function* yang digunakan untuk melakukan perhitungan secara agregat terhadap elemen array, seperti *function* untuk mencari nilai minimum dari seluruh elemen array, demikian pula maksimumnya, menghitung total jumlahan seluruh nilai dalam array, dan lain-lain.

sum : untuk menjumlahkan seluruh data dalam array

```
print(a.sum())
```

```
Outputnya :
39
```

min : untuk menentukan nilai minimum dari seluruh elemen array

```
print(a.min())
```

```
Outputnya :
1
```

Agregat **Array**

Adaun perintah berikut ini adalah mencari nilai maksimum pada setiap **barisnya** dengan menggunakan **max()** namun dengan tambahan parameter **axis=1.** Apabila akan mencari maksimum pada setiap kolom, maka digunakan parameter **axis=0.** Penambahan parameter ini dapat pula diberikan pada **min().**

```
print(a.max(axis=1))
```

```
Outputnya :
[3 5 8]
```

Agregat **Array**

Secara lengkap, *function-function* agregat utama yang tersedia dalam NumPy adalah sebagai berikut :

Nama Function	Keterangan
sum()	Menghitung jumlah smua elemen array
prod()	Menghitung hasil perkalian semua elemen array
mean()	Menghitung rata-rata semua elemen array
std()	Menghitung standar deviasi semua elemen array
var()	Menghitung variasi semua elemen array
min()	Mencari nilai minimum dari elemen array
max()	Mencari nilai maksimum dari elemen array
argmin()	Mencari indeks array yang memiliki nilai minimum
argmax()	Mencari indeks array yang memiliki nilai maksimum
median()	Mencari nilai tengah elemen array

Sorting Array

Sorting **Array**

Array bersifat *mutable,* sama seperti llist. Sehingga dalam hal ini elemen dari array dapat di ubah susunannya, misalnya melalui sorting. Oleh karena itu, di dalam NumPy terdapat *function* untuk melakukan sorting dengan menggunakan *function* **sort().**

Berikut ini adalah contoh untuk array 1 dimensi

```
a = np.array([8,4,2])
print (a)
a.sort()
print(a)
```

```
Outputnya :
[8 4 2]
[2 4 8]
```

Untuk array 2 dimensi, sorting juga bisa dilakukan per kolom maupun per baris.

```
a = np.array([8,4,2], [3,2,6])
print (a)
```

```
Outputnya :
[[8 4 2]
 [3 2 6]]
```

Sorting **Array**

Selanjutnya dengan menambahkan parameter **axis=0** pada **sort(),** maka akan dihasilkan array baru yang merupakan hasil sorting berdasarkan kolom.

```
a.sort(axis=0)
print(a)
```

```
Outputnya :
[[3 2 2]
 [8 4 6]]
```

Demikian pula untuk parameter **axis=1,** proses sorting dilakukan berdasarkan per baris. (hal ini sama hasilnya untuk **sort()** tanpa parameter **axis** pada array 2 dimensi)

```
a.sort(axis=1)
print(a)
```

```
Outputnya :
[[2 4 8]
 [2 3 6]]
```

Manipulasi Array

Mentranspose **Array**

Dalam aljabar linear, mentranspose suatu array adalah proses mengubah posisi elemen-elemen vektor baris menjadi vektor kolom, demikian pula sebaliknya. Terdapat dua cara untuk melakukan hal ini, yaitu menggunakan *function* **transpose()** atau memanggil property **T** dari suatu array.

```
a = np.array([3,5,7], [4,1,8], [8,5,2])
print (a)
```

```
Outputnya :
[[3 5 7]
 [4 1 8]
 [8 5 2]]
```

Cara mentranspose array **a** tersebut adalah dengan menggunakan **transpose()**

```
print (a.transpose())
```

```
Outputnya :
[[3 4 8]
 [5 1 5]
 [7 8 2]]
```

Mengubah Dimensi **Array**

Dimensi suatu array dapat diubah sesuai dengan kebutuhan. Sebagai contoh suatu array berdimensi dua, maka dengan menggunakan suatu *function* array tersebut dapat ditransformasikan ke dimensi satu.

Untuk melakukan al ini terdapat tiga cara yang harus dilakukan, yaitu menggunakan perintah **ravel(), flatten(),** dan **reshape()**

```
print (a.ravel())
```

```
Outputnya :
[[3 5 7 4 1 8 8 5 2]]
```

```
print (a.flatten())
```

```
Outputnya :
[[3 5 7 4 1 8 8 5 2]]
```

Mengubah Dimensi **Array**

Selanjutnya, apabila menggunakan **reshape()** harus ditentukan terlebih dahulu ukuran arraynya. Berikut contoh yang menunjukkan proses mengubah struktur array a dari yang sebelumnya 3x3, menjadi 9x1.

Alternatif selain menggunakan **reshape(),** dapat pula digunakan **resize().** Beda antara **reshape()** dengan **resize()** adalah bahwa **reshape()** tidak mengubah struktur dari array aslinya. Sedangkan **resize()** mengubah struktur array aslinya.

```
print (a.reshape(9,1))
```

```
a.resize((9,1))
print (a)
```

```
Outputnya :
[[3]
 [5]
 [7]
 [4]
 [1]
 [8]
 [8]
 [5]
 [2]]
```

Operasi Dasar **Array**

Menambahkan Data **Array**

Untuk melakukan penambahan data array di NumPy adalah dengan memanggil method **append()** dan **insert(),** untuk detail perbedaannya sebagai berikut :

append()

Array 1 Dimensi

```
b = np.array([1, 2, 3, 4])
print('Data Sebelum Penambahan :',b)
b = np.append(b, [10])
print('Data Setelah Penambahan :',b)
```

```
Outputnya :
Data Sebelum Penambahan : [1 2 3 4]
Data Setelah Penambahan : [ 1  2  3  4
10]
```

Menambahkan Data **Array**

Array 2 Dimensi

```
a = np.array([[3,5,7],[4,1,8],[8,5,2]])
print(a)
```

```
Outputnya :
[[3 5 7]
 [4 1 8]
 [8 5 2]]
```

Selanjutnya, akan ditambahkan elemen [1,1,1] ke dalam array a menjadi baris terakhir. Untuk melakukan hal ini, perlu ditambahkan parameter **axis=0**

```
a = np.append(a, [[1,1,1]], axis=0)
print(a)
```

```
Outputnya :
[[3 5 7]
 [4 1 8]
 [8 5 2]
 [1 1 1]]
```

Menambahkan Data **Array**

Demikian pula jika ingin ditambahkan pada kolom terakhir, maka perlu dituliskan parameter **axis=1.** Namun perlu diingat bahwa format data harus disesuaikan dengan posisinya. Dalam hal ini, bentuk format data harus dibuat dalam vector kolom

```
a = np.append(a, [[1],[1],[1]], axis=1)
print(a)
```

```
Outputnya :
[[3 5 7 1]
 [4 1 8 1]
 [8 5 2 1]]
```

Untuk menambahkan dua kolom baru, caranya sama seperti sebelumnya yaitu dengan menambahkan parameter **axis=1** serta format bentuk datanya disesuaikan

```
a = np.append(a, [[1,2],[1,2],[1,2]],
axis=1)
print(a)
```

```
Outputnya :
[[3 5 7 1 2]
 [4 1 8 1 2]
 [8 5 2 1 2]]
```

Menambahkan Data **Array**

Untuk melakukan penambahan data array di NumPy adalah dengan memanggil method **append()** dan **insert(),** untuk detail perbedaannya sebagai berikut :

insert()

Array 1 Dimensi

```
b = np.array([1, 2, 3, 4])
print('Data Sebelum Penambahan :',b)
b = np.insert(b, 2,  [10])
print('Data Setelah Penambahan :',b)
```

```
Outputnya :
Data Sebelum Penambahan : [1 2 3 4]
Data Setelah Penambahan : [ 1  2  10
3  4]
```

Menambahkan Data **Array**

Array 2 Dimensi

```
a = np.array([[3,5,7],[4,1,8],[8,5,2]])
print(a)
```

```
Outputnya :
[[3 5 7]
 [4 1 8]
 [8 5 2]]
```

Khusus untuk array dua dimensi, perlu ada parameter yang digunakan untuk menentukan pada bagian mana baris atau kolom akan ditambahkan. Berikut contoh untuk menambahkan elemen [1,1,1] ke dalam baris indeks no 1

```
a = np.insert(a, 1, [1,1,1], axis=0)
print(a)
```

```
Outputnya :
[[3 5 7]
 [1 1 1]
 [4 1 8]
 [8 5 2]]
```

Menambahkan Data **Array**

Sedangkan perintah untuk menambahkan elemen [1,1,1] ke kolom no 1 adalah sebagai berikut

```
a = np.insert(a, 1, [1,1,1], axis=1)
print(a)
```

```
Outputnya :
[[3 1 5 7]
 [4 1 1 8]
 [8 1 5 2]]
```

Menghapus Data **Array**

Untuk menghasus sebuah elemen atau beberapa elemen di dalam array, dapat dilakukan menggunakan perintah **delete()** dengan menyebutkan nomor indeks dari elemen yang akan dihapus

```
b = np.array([1, 2, 3, 4])
print(b)
b = np.delete(b, 2)
print(b)
```

```
Outputnya :
[1 2 3 4]
[1 2 4]
```

Adapun untuk array dua dimensi, untuk menghapus elemen harus disebutkan kolom atau barisnya. Misalkan diketahui array 2 dimensi sebagai berikut :

```
a = np.array([[3,5,7],[4,1,8],[8,5,2]])
print(a)
```

```
Outputnya :
[[3 5 7]
 [4 1 8]
 [8 5 2]]
```

Menambahkan Data **Array**

Apabila di inginkan untuk menghapus semua elemen yang terletak pada baris indeks ke-1 dari a, maka perintahnya sebagai berikut

```
a = np.delete(a, 1, axis=0)
print(a)
```

```
Outputnya :
[[3 5 7]
 [8 5 2]]
```

Demikian pula untuk penghapusan semua elemen pada kolom indeks tertentu,

```
a = np.delete(a, 1, axis=1)
print(a)
```

```
Outputnya :
[[3 5]
 [4 1]
 [8 5]]
```

Menggabungkan Array

Penggabungan **Array**

Untuk melakukan penggabungan array di NumPy dapat menggunakan **concenate()** dan juga menggunakan model stack

concenate()

```
a = np.array([1,3,5,7])
b = np.array([2,4,6,8])

c = np.concatenate([a,b])
print(c)
```

```
Outputnya :
[1, 3, 5, 7, 2, 4, 6, 8]
```

Adapun untuk array 2 dimensi, dalam proses penggabungan harus diperhatikan posisinya, apakah dalam posisi baris atau kolom.

```
a = np.array([[1,3],[5,7]])
b = np.array([[2,4],[6,8]])

c = np.concatenate([a,b], axis=0)
print(c)
```

```
Outputnya :
[[1 3]
 [5 7]
 [2 4]
 [6 8]]
```

Penggabungan **Array**

Sedangkan jika array **b** digabungkan menjadi kolom baru, maka perlu ditambahkan **axis=1**

```
a = np.array([[1,3],[5,7]])
b = np.array([[2,4],[6,8]])

c = np.concatenate([a,b], axis=1)
print(c)
```

```
Outputnya :
[[1 3 2 4]
 [5 7 6 8]]
```

Selain menggunakan perintah **concenate(),** penggabungan array dapat pula menggukana stack / tumpukan. Stack ini memiliki dua mayam, yaitu stack vertical dan horizontal. Misalkan diketahui sebuah array **a.** kemudian apabila array **b** akan digabung secara vertical dengan **a,** maka array **b** nantinya akan diletakkan di bawah array **a.** sebaliknya, apabila array **b** akan digabung secara horizontal dengan **a,** maka **b** akan diletakkan di sebelah kanan array **a.**

Penggabungan **Array**

Array 1 Dimensi

vstack()

```
a = np.array([1,3,5,7])
b = np.array([2,4,6,8])

c = np.vstack([a,b])
print(c)
```

```
Outputnya :
[[1 3 5 7]
 [2 4 6 8]]
```

hstack()

```
a = np.array([1,3,5,7])
b = np.array([2,4,6,8])

c = np.hstack([a,b])
print(c)
```

```
Outputnya :
[1, 3, 5, 7, 2, 4, 6, 8]
```

Penggabungan **Array**

Array 2 Dimensi

vstack()

```
a = np.array([[1,3],[5,7]])
b = np.array([[2,4],[6,8]])

c = np.vstack([a,b])
print(c)
```

```
Outputnya :
[[1 3]
 [5 7]
 [2 4]
 [6 8]]
```

hstack()

```
a = np.array([[1,3],[5,7]])
b = np.array([[2,4],[6,8]])

c = np.hstack([a,b])
print(c)
```

```
Outputnya :
[[1, 3, 2, 4]
 [5, 7, 6, 8]
```

Memecah **Array**

Memecah **Array**

Selain digabung, array juga bisa dipecah menjadi array-array yang lebih kecil ukurannya dengan ukuran yang sama. Cara pemecahan array ini dapat dilakukan secara vertical maupun horizontal.

Untuk memecah array secara vertical digunakan perintah **vsplit()** dan **hsplit()** jika secara horizontal. Output hasil pecahan ini akan dihasilkan array juga, dengan elemennya adalah array-array hasil pecahan.

Misalkan didefinisikan sebuah array satu dimensi berukuran 1x6 sebagai berikut

```
b = np.array([1, 2, 3, 4, 5, 6])
b = np.hsplit(b, 1)
print(b)
```

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

Jika diberikan perintah **hsplit(a,1)** maka hasilnya akan memecah array **a** secaara horizontal menjadi 1 bagian. Dalam hal ini hasilnya akan sama dengan **a** itu sendiri.

Memecah **Array**

Namun, jika diberikan perintah **hsplit(a,2),** maka akan memecah array **a** menjadi 2 bagian sama yang sama ukuran, yaitu masing-masing berukuran 1x3

```
b = np.array([1, 2, 3, 4, 5, 6])
b = np.hsplit(b, 2)
print(b)
```

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

Jika diperhatikan output **hsplit()** di atas, maka dapat dilihat bahwa output tersebut berupa array juga. Dalam hal ini, khusus array berbentuk vector baris tidak bisa dipecah secara vertical, atau dengan kata lain tidak bisa diberikan perintah **vsplit()**

Selanjutnya, **hsplit()** dan **vsplit()** dapat dilakukan pada array dimensi dua. Pehatikan array dua dimensi berikut

```
a = np.array([[3,5],[1,2],[0,8]])
print(a)
```

```
Outputnya :
[[3 5]
 [1 2]
 [0 8]]
```

Memecah **Array**

Jika diinginkan dipecah menjadi 3 bagian, maka akan didapatkan array-array dalam bentuk vector baris seperti berikut

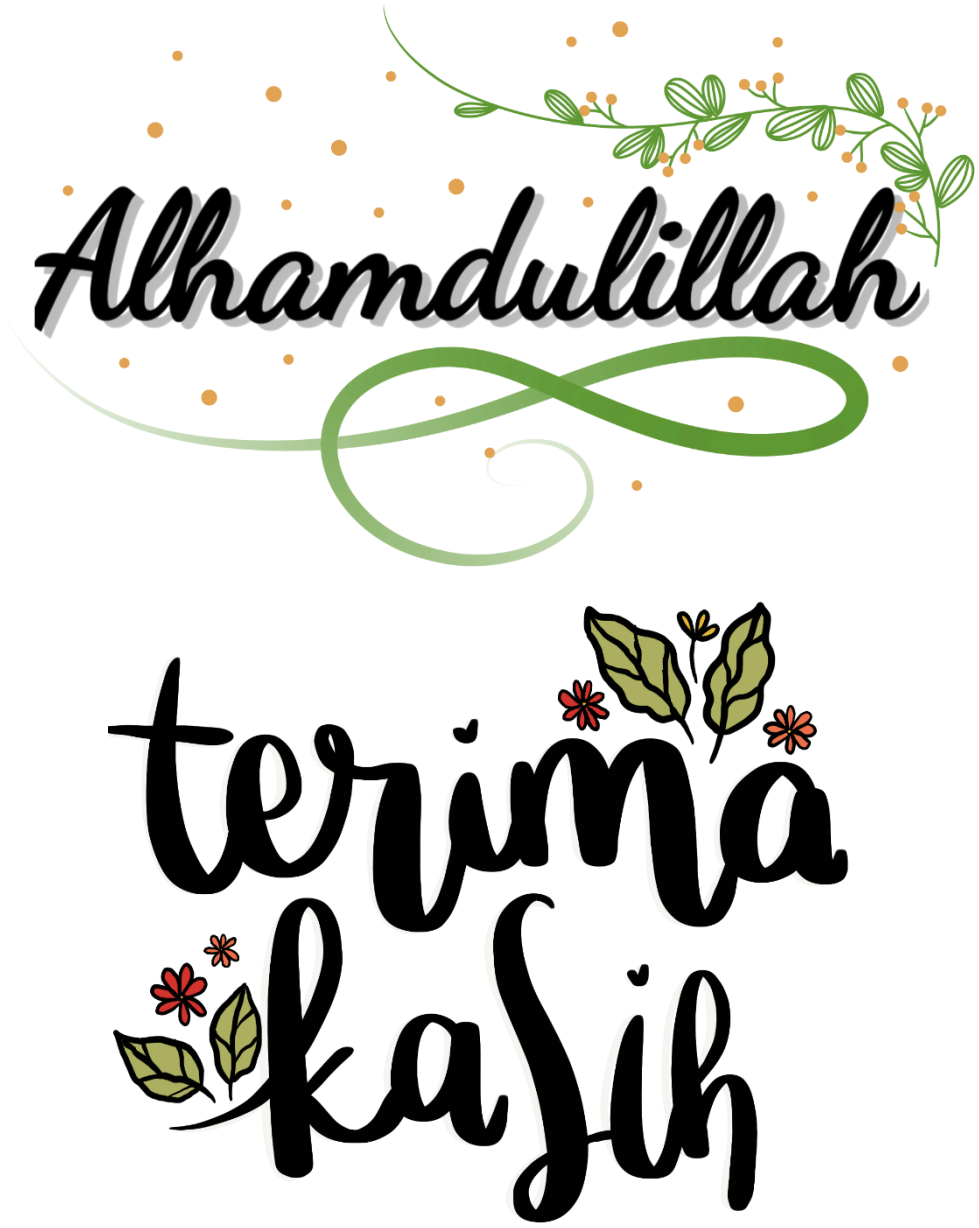
```
a = np.vsplit(a,3)
print(a)
```

```
Outputnya :
[array([[3, 5]]), array([[1, 2]]),
array([[0, 8]])]
```

Sebaliknya, jika dilakukan pemecahan secara horizontal maka akan didapatkan array-array berbentuk vector kolom seperti berikut

```
a = np.hsplit(a,2)
print(a)
```

```
Outputnya :
[array([[3],
        [1],
        [0]]), array([[5],
        [2],
        [8]])]
```


Alhamdulillah
terima kasih