Mengenal Numpy Untuk Pembelajar Pemula

Library NumPy adalah modul kaya yang membantu kita bekerja dengan objek array multidimensi. Hal ini memungkinkan pengguna untuk membangun matriks dimensi yang lebih tinggi untuk melakukan perhitungan matematis yang kompleks. NumPy hadir dengan banyak operasi matematika bawaan, aljabar Linear, dan fungsi terkait Transformasi Fourier.

Array Numpy menunjukkan matematika tingkat lanjut dan berbagai jenis operasi pada data dalam jumlah besar.

Setiap kali kamu melihat “array”, “NumPy array” atau “ndarray” dalam teks, dengan beberapa pengecualian, semuanya mengacu pada hal yang sama: objek ndarray.

Mengapa Numpy?

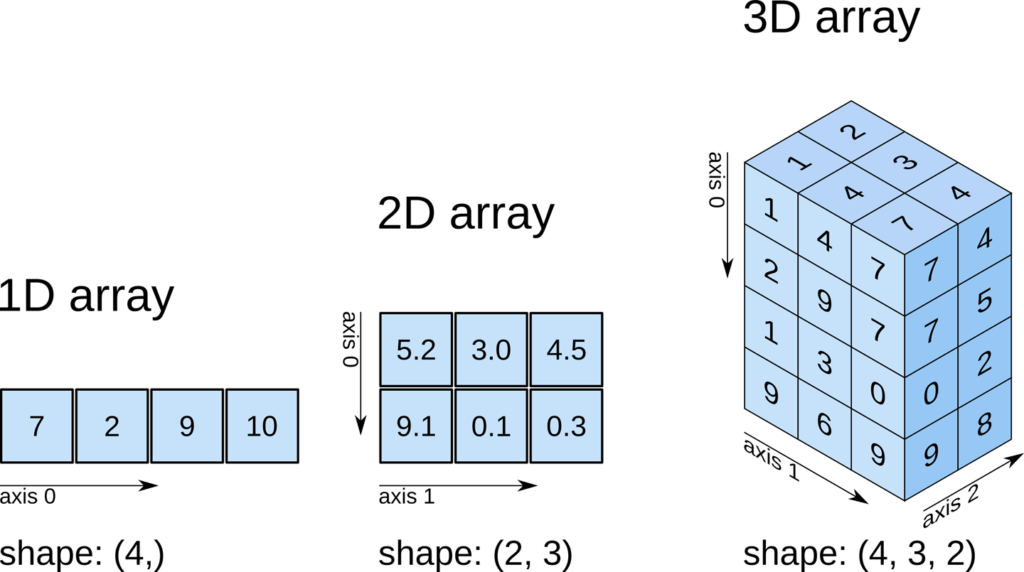
Numpy adalah library untuk menangani data numerik besar. Tidak seperti list python, NumPy memiliki tipe data yang homogen karena dapat dikemas dengan padat ke dalam memori. Numpy secara internal menggunakan bahasa seperti Fortran, C, dan C++ yang memiliki waktu eksekusi lebih cepat dibandingkan dengan python.

Tujuan NumPy

* operasi matematika dasar, perhitungan statistik, dan Aljabar linier dasar.
* membangun array n-dimensi dan matriks
* Mengakses, Memperbaharui elemen dalam array.
* Menata ulang atau mereformasi indeks matriks.
* Mengiris matriks.

Manfaat NumPy

* Konsumsi memori lebih sedikit
* Daya komputasi yang cepat

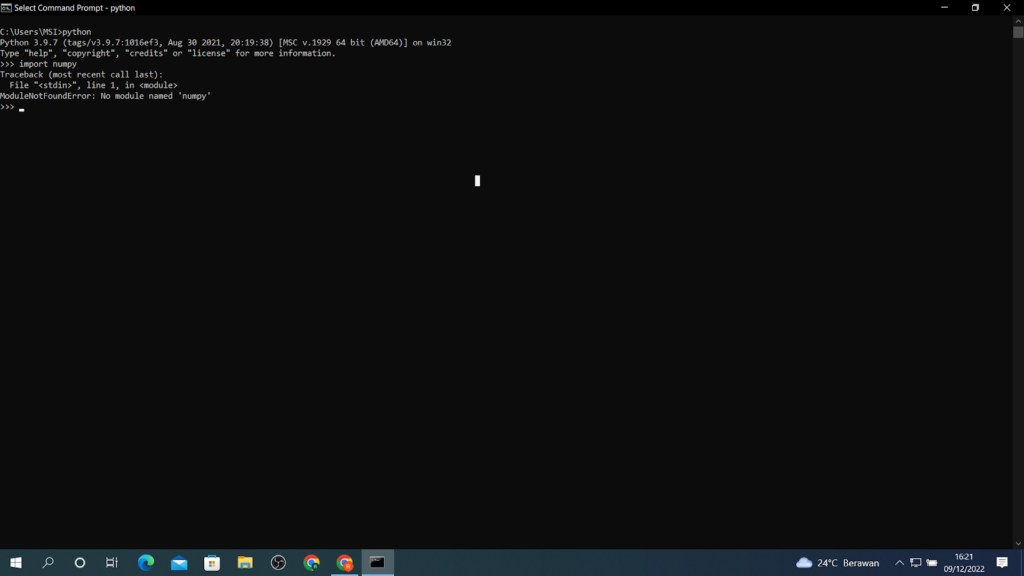


Source: analyticsvidhya.

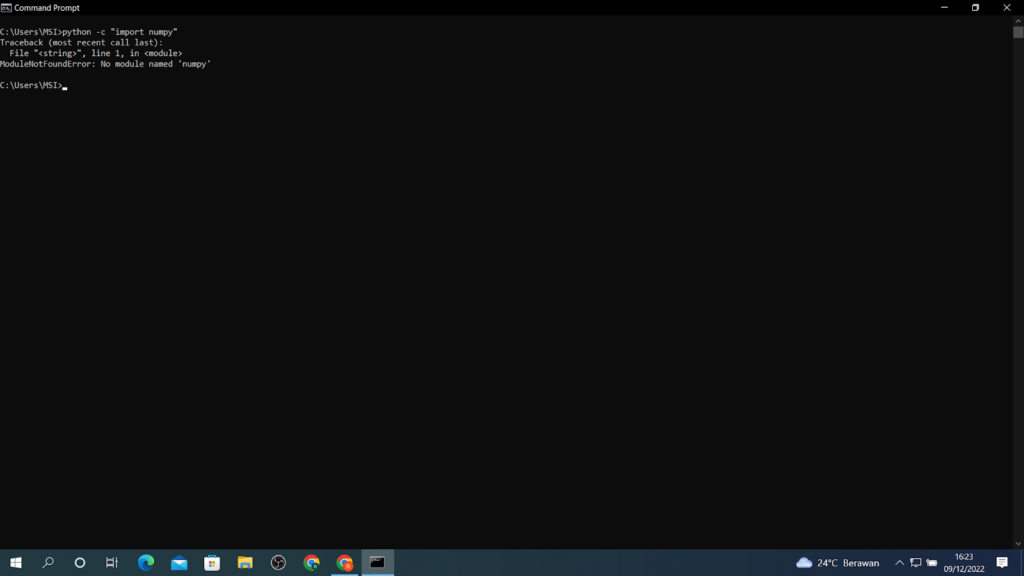
**Dasar-dasar Numpy**

**Install python library numpy**

Cek dulu apakah numpy sudah terinstall atau belum

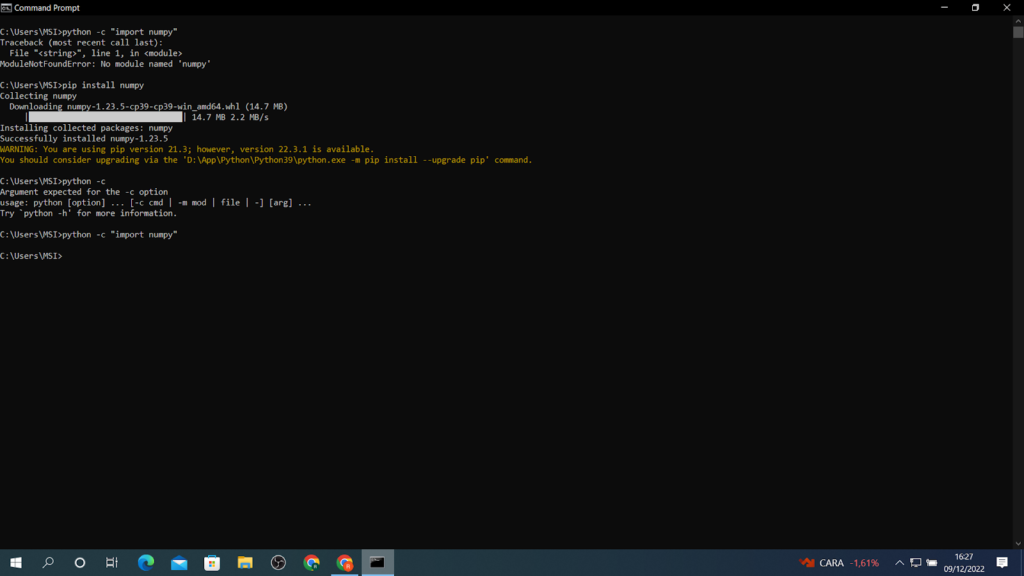


Source : dokumentasi pribadi penulis

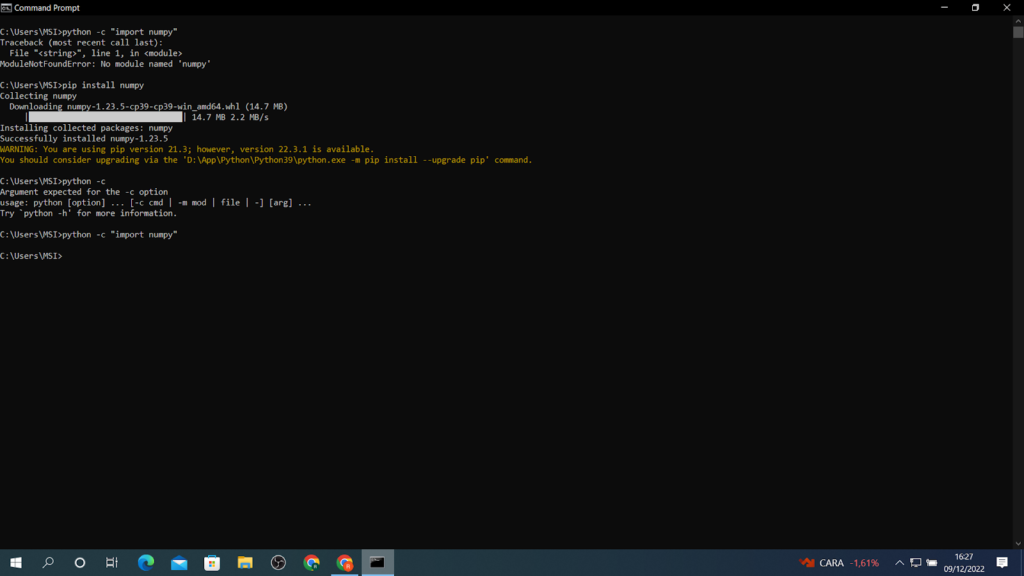


Source : dokumentasi pribadi penulis

**Cara install numpy**



Source : dokumentasi pribadi penulis



Source : dokumentasi pribadi penulis

Untuk menginstall library Numpy di Python, kamu dapat menggunakan perintah pip (pip installer). Perintah ini dapat dijalankan di Command Prompt atau Terminal, tergantung sistem operasi yang kamu gunakan.

Untuk menginstal numpy, kamu dapat mengetikkan perintah berikut di Command Prompt atau Terminal:

|  |
| --- |
| pip install numpy |

Atau

|  |
| --- |
| python -m pip install numpy |

Jika kamu menggunakan python versi 3.x, kamu dapat menggunakan perintah:

|  |
| --- |
| pip3 install numpy |

Sebagai alternatif, kamu juga dapat menginstall library Numpy melalui Anaconda Navigator atau menambahkan library numpy pada file requirements.txt.

Check version

Mari kita mulai dengan melihat versi Python dan versi package NumPy. Ini untuk hampir semua paket — selalu periksa versinya.

Setelah library Numpy berhasil diinstall, kamu dapat mengecek versi Numpy yang terinstall dengan mengetikkan perintah:

|  |
| --- |
| **import** numpy print(numpy.\_\_version\_\_) |

Jika kamu sudah berhasil menginstall dan mengecek versi numpy, kamu dapat mulai menggunakan library ini dalam proyek kamu.

## Import Numpy

|  |
| --- |
| **import** numpy **as** np |

Baris di atas akan mengimpor NumPy dan alias ke np(alias standar yang diikuti dalam komunitas ilmu data untuk NumPy).

# Membuat array dengan numpy

Kamu dapat membuat array dengan Numpy menggunakan fungsi numpy.array(). Fungsi ini mengambil satu parameter, yaitu data yang akan ditambahkan ke dalam array. Data tersebut dapat berupa list, tuple, atau array lainnya.

Contoh:

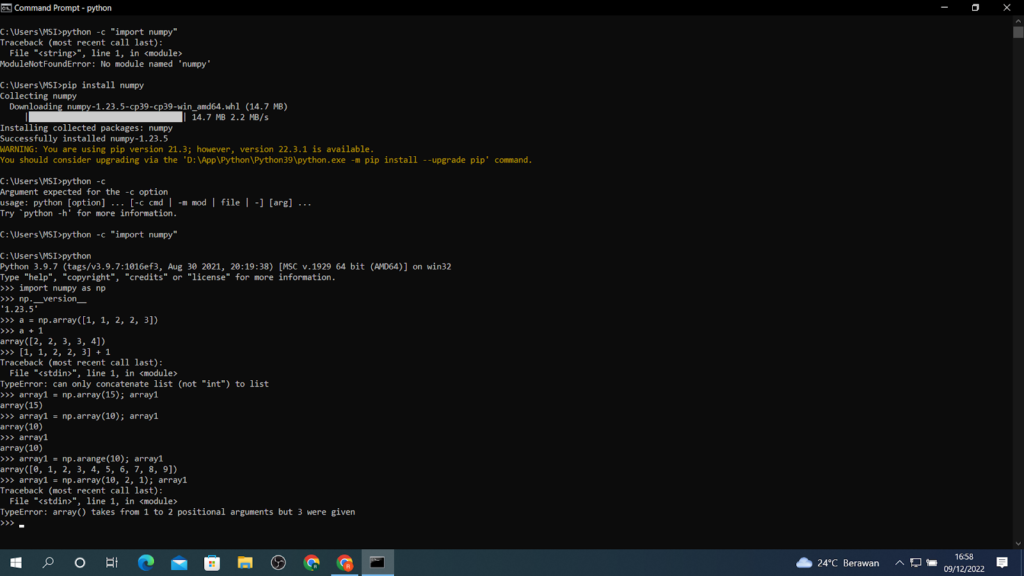
|  |
| --- |
| **import** numpy **as** np  # Membuat array dari list arr = np.array([1, 2, 3, 4, 5]) print(arr) # Output: array([1, 2, 3, 4, 5])  # Membuat array dari tuple arr = np.array((6, 7, 8, 9, 10)) print(arr) # Output: array([6, 7, 8, 9, 10]) |

### **Inisialisasi Array**

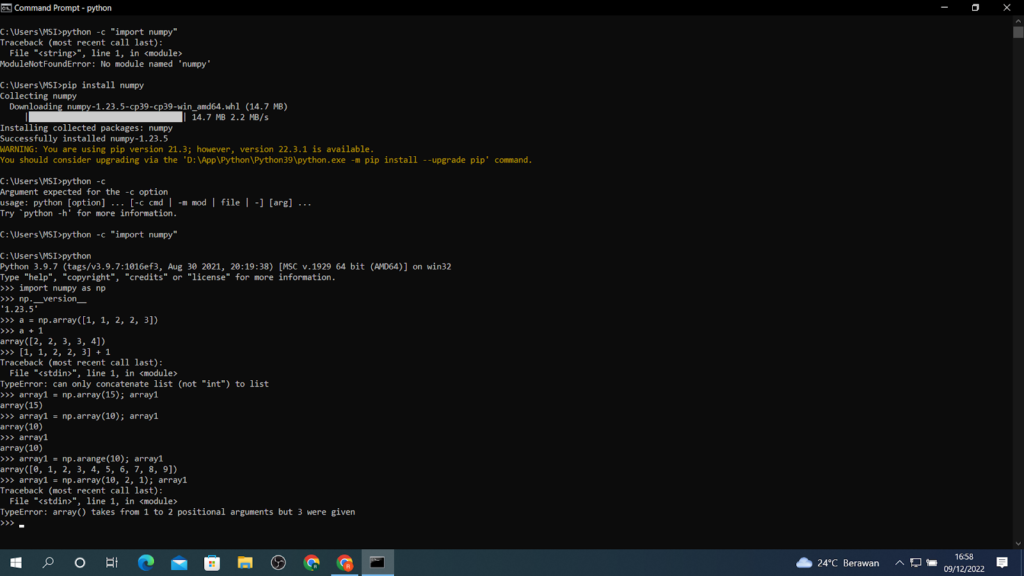
np.array: Metode numpy ini membuat array numpy.

Menggunakan fungsi array ()

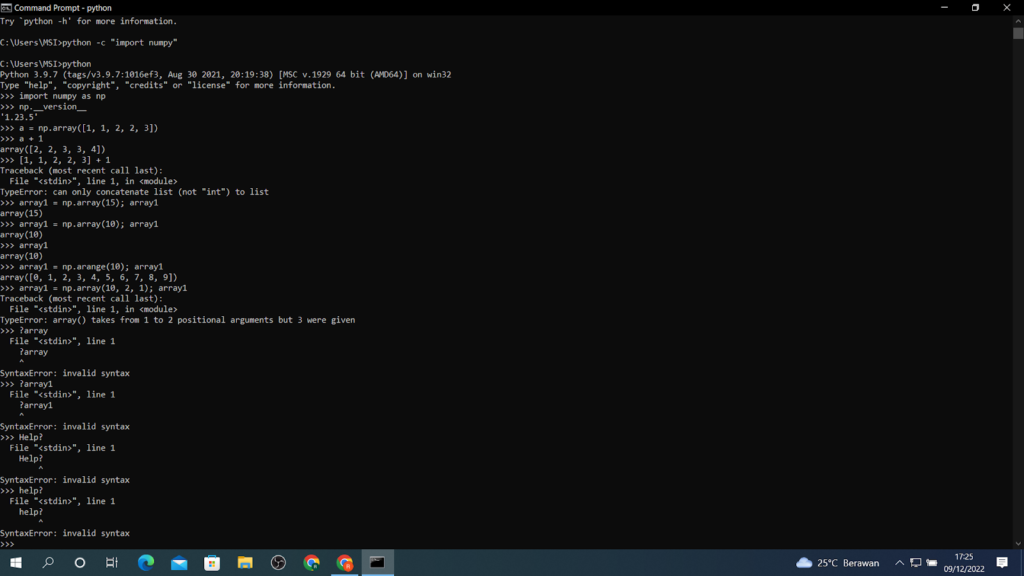
Source : dokumentasi pribadi penulis



Source: dokumentasi pribadi penulis



Source: dokumentasi pribadi penulis



Source: dokumentasi pribadi penulis

Kita dapat menginisialisasi array menggunakan fungsi np.array(). Array dengan ukuran berapa pun dapat dibuat menggunakan fungsi. Selain itu, tipe data dari array yang dibuat dapat ditentukan selama pembuatan array menggunakan parameter dtype.

Kita dapat mengirimkan daftar 1-D, 2-D atau N-D dan itu akan mengubahnya menjadi array numpy. Kita juga bisa mengirimkan tipe data dari elemen array. Kode di bawah ini akan membuat array numpy 2-D dengan dtype sebagai float32).

|  |
| --- |
| np.array([[1,2],[3,4]],dtype=np.float32) |

**Latihan praktek 1:**

|  |
| --- |
| a = np.array([10,20,30,40,50]) a |

**Latihan praktek 2:**

|  |
| --- |
| a1 = np.array([0,5],[1,5],[2,5],3,5]],dtype=np.float64) a1 |

**Latihan praktek 3 (Non-graded/evaluasi)**

Untuk melakukan latihan praktek dengan fungsi np.array() kamu dapat mencoba beberapa latihan berikut:

1. Buat array dengan np.array() dari list angka 1-5, cetak array tersebut
2. Buat array dengan np.array() dari list string, cetak array tersebut
3. Buat array dengan np.array() dari tuple angka 1-5, cetak array tersebut
4. Buat array 2×3 dengan np.array() dan isi dengan angka acak, cetak array tersebut
5. Buat array 1 dimensi dengan np.array() dan isi dengan angka 0-10 dengan jarak 2, cetak array tersebut.

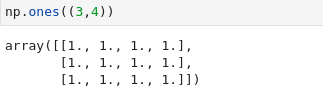
|  |
| --- |
| **import** numpy **as** np  # Latihan 1 list\_angka = [1,2,3,4,5] arr\_angka = np.array(list\_angka) print(arr\_angka)  # Latihan 2 list\_string = [‘satu’,’dua’,’tiga’] arr\_string = np.array(list\_string) print(arr\_string)  # Latihan 3 tuple\_angka = (1,2,3,4,5) arr\_tuple = np.array(tuple\_angka) print(arr\_tuple)  # Latihan 4 arr\_random = np.random.rand(2,3) print(arr\_random)  # Latihan 5 arr\_0\_10 = np.arange(0,11,2) print(arr\_0\_10) |

### **Membuat array dengan Fungsi numpy.zeros(), numpy.ones(), dan numpy.empty()**

Kamu juga dapat membuat array dengan ukuran dan tipe data yang ditentukan. Fungsi numpy.zeros() dan numpy.ones() dapat digunakan untuk membuat array dengan nilai 0 atau 1. Fungsi numpy.empty() digunakan untuk membuat array dengan nilai acak. Contohnya:

|  |
| --- |
| **import** numpy **as** np  # Membuat array 2×3 dengan nilai 0 arr = np.zeros((2, 3)) print(arr) # Output: # [[0. 0. 0.] #  [0. 0. 0.]]  # Membuat array 2×3 dengan nilai 1 arr = np.ones((2, 3)) print(arr) # Output: # [[1. 1. 1.] #  [1. 1. 1.]]  # Membuat array 2×3 dengan nilai acak arr = np.empty((2, 3)) print(arr) # Output: # [[0. 0. 0.] #  [0. 0. 0.]] |

np.ones: Metode ini akan membuat larik satu, kamu hanya perlu memasukkan bentuk larik. Dalam bentuk blok kode di bawah ini (3,4) berarti 3 baris dan 4 kolom untuk matriks. Jika kamu melewatkan satu bilangan bulat di np.ones, itu akan membuat array dengan banyak elemen.



np.ones method in numpy

np.zeros: Mirip dengan np.ones, itu akan membuat array nol.

|  |
| --- |
| >>>np.zeros(4) >>>array([0.,0.,0.]) |

**Latihan Praktek (Non-graded/evaluasi)**

Untuk melakukan latihan praktek dengan fungsi numpy.zeros(), numpy.ones(), dan numpy.empty() kamu dapat mencoba beberapa latihan berikut:

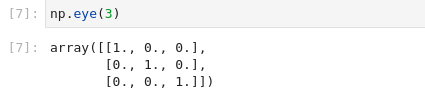
1. Buat array 2×3 dengan numpy.zeros() dan cetak array tersebut
2. Buat array 2×3 dengan numpy.ones() dan cetak array tersebut
3. Buat array 2×3 dengan numpy.empty() dan cetak array tersebut
4. Buat array 1 dimensi dengan numpy.zeros() dengan panjang 5 dan cetak array tersebut
5. Buat array 1 dimensi dengan numpy.ones() dengan panjang 5 dan cetak array tersebut
6. Buat array 1 dimensi dengan numpy.empty() dengan panjang 5 dan cetak array tersebut

Itulah beberapa latihan praktek yang dapat kamu coba untuk membuat array dengan fungsi numpy.zeros(), numpy.ones(), dan numpy.empty(). Ingat untuk selalu mengecek hasil array yang dibuat dengan fungsi-fungsi tersebut.

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_zeros = np.zeros((2,3)) print(arr\_zeros)  # Latihan 2 arr\_ones = np.ones((2,3)) print(arr\_ones)  # Latihan 3 arr\_empty = np.empty((2,3)) print(arr\_empty)  # Latihan 4 arr\_zeros\_5 = np.zeros(5) print(arr\_zeros\_5)  # Latihan 5 arr\_ones\_5 = np.ones(5) print(arr\_ones\_5)  # Latihan 6 arr\_empty\_5 = np.empty(5) print(arr\_empty\_5) |

### **Fungsi np.eye**

np.eye: Metode ini membuat matriks identitas m\*m (karena matriks identitas adalah matriks persegi dengan elemen diagonal sebagai 1 dan elemen lainnya sebagai 0). Kita hanya perlu meneruskan nilai m.



np.eye method with m=3

**Latihan praktek**

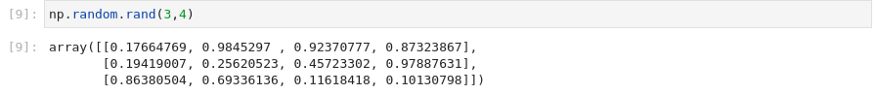
Untuk melakukan latihan praktek dengan fungsi np.eye() kamu dapat mencoba beberapa latihan berikut:

1. Buat array 2×2 dengan np.eye() dan cetak array tersebut
2. Buat array 3×3 dengan np.eye() dan cetak array tersebut
3. Buat array 5×5 dengan np.eye() dan cetak array tersebut
4. Buat array 2×4 dengan np.eye() dan cetak array tersebut
5. Buat array 4×3 dengan np.eye() dan cetak array tersebut
6. Buat array 3×5 dengan np.eye() dan cetak array tersebut

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_eye\_2 = np.eye(2) print(arr\_eye\_2)  # Latihan 2 arr\_eye\_3 = np.eye(3) print(arr\_eye\_3)  # Latihan 3 arr\_eye\_5 = np.eye(5) print(arr\_eye\_5)  # Latihan 4 arr\_eye\_2\_4 = np.eye(2,4) print(arr\_eye\_2\_4) |

### **Fungsi np.random.rand.**

np.random.rand: Metode ini menghasilkan elemen acak menggunakan distribusi seragam. Kita hanya perlu mengirimkan dalam bentuk array.



np.random.rand with a shape of (3,4) meaning 3 rows and 4 columns

**Latihan praktek**

Untuk melakukan latihan praktek dengan fungsi np.random.rand() kamu dapat mencoba beberapa latihan berikut:

1. Buat array 2×3 dengan np.random.rand() dan cetak array tersebut
2. Buat array 4×4 dengan np.random.rand() dan cetak array tersebut
3. Buat array 3×2 dengan np.random.rand() dan cetak array tersebut
4. Buat array 1 dimensi dengan panjang 5 dan isi dengan angka acak menggunakan np.random.rand()
5. Buat array 1 dimensi dengan panjang 10 dan isi dengan angka acak menggunakan np.random.rand()
6. Buat array 2 dimensi dengan ukuran (3,4) dan isi dengan angka acak menggunakan np.random.rand()

Contoh:

import numpy as np

# Latihan 1

arr\_rand\_2\_3 = np.random.rand(2,3)

print(arr\_rand\_2\_3)

# Latihan 2

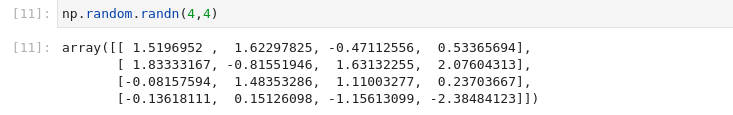
arr\_rand\_4\_4 = np.random.rand(4,4)

print(arr\_rand\_4\_4)

Selanjutnya coba kamu kerjakan sendiri ya.

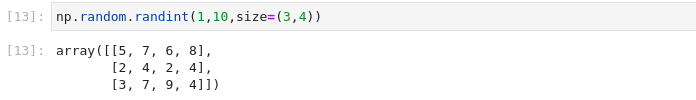
### **Fungsi np.random.randn.**

np.random.randn: Ini mirip dengan elemen np.random.rand tetapi elemen dipilih dari distribusi gaussian atau normal.



### **Fungsi np.random.randint**

np.random.randint: Metode ini mengembalikan bilangan bulat acak dari rendah(inklusif) ke tinggi(eksklusif) dengan ukuran array keluaran tertentu.



**Latihan praktek**

Untuk melakukan latihan praktek dengan fungsi np.random.rand(), np.random.randn, dan np.random.randint() kamu dapat mencoba beberapa latihan berikut:

1. Buat array 2×3 dengan np.random.rand() dan cetak array tersebut
2. Buat array 4×4 dengan np.random.randn() dan cetak array tersebut
3. Buat array 1 dimensi dengan panjang 5 dan isi dengan angka random menggunakan np.random.randint(0,10)
4. Buat array 1 dimensi dengan panjang 10 dan isi dengan angka random menggunakan np.random.randint(5,15)
5. Buat array 2 dimensi dengan ukuran (3,4) dan isi dengan angka random menggunakan np.random.randn()
6. Buat array 2 dimensi dengan ukuran (4,5) dan isi dengan angka random menggunakan np.random.randint(0,100)

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_rand\_2\_3 = np.random.rand(2,3) print(arr\_rand\_2\_3)  # Latihan 2 arr\_randn\_4\_4 = np.random.randn(4,4) print(arr\_randn\_4\_4)  # Latihan 3 arr\_randint\_5 = np.random.randint(0, 10, 5) print(arr\_randint\_5)  # Latihan 4 arr\_randint\_10 = np.random.randint(5, 15, 10) print(arr\_randint\_10)  # Latihan 5 arr\_randn\_3\_4 = np.random.randn(3,4) print(arr\_randn\_3\_4)  # Latihan 6 arr\_randint\_4\_5 = np.random.randint(0, 100, (4,5)) print(arr\_randint\_4\_5) |

Itulah beberapa latihan praktek yang dapat kamu coba untuk mengenal dan menggunakan fungsi np.random.rand(), np.random.randn, dan np.random.randint(). Ingatlah untuk selalu mengecek hasil array yang dibuat dengan fungsi-fungsi tersebut dan memahami perbedaan masing-masing fungsi.

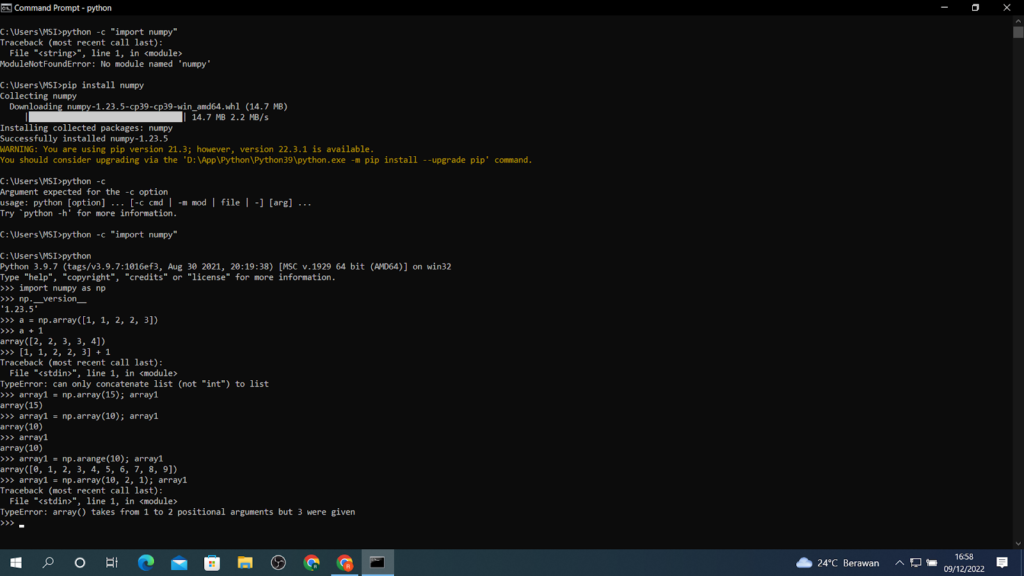
### **Fungsi numpy.arange () atau numpy.linspace ()**

Kamu juga dapat membuat array dengan rentang nilai tertentu menggunakan numpy.arange() atau numpy.linspace(). Contohnya:

|  |
| --- |
| **import** numpy **as** np  # Membuat array dengan rentang nilai dari 0 sampai 10 arr = np.arange(0, 11) print(arr) # Output: [ 0  1  2  3  4  5  6  7  8  9 10]  # Membuat array dengan rentang nilai dari 0 sampai 10 dengan jarak 2 arr = np.arange(0, 11, 2) print(arr) # Output: [ 0  2  4  6  8 10]  # Membuat array dengan rentang nilai dari 0 sampai 10 dengan jumlah elemen 5 arr = np.linspace(0, 10, 5) print(arr) # Output: [ 0.   2.5  5.   7.5 10. ] |

np.arange: Fungsi ini mirip dengan metode jangkauan python. Kamu hanya perlu memasukkan nilai awal (inklusif), nilai akhir (eksklusif), dan nilai interval.

Sekarang kita membuat array dengan**fungsi arange** dari numpy dengan menentukan indeks awal dan akhir dalam parameter dan bukan range built-in.



Source: dokumentasi pribadi penulis

**Latihan Praktek 1**

|  |
| --- |
| a2 = np.arange(5,15) a2 |

**Latihan praktek 2**

Untuk melakukan latihan praktek dengan fungsi np.arange() dan np.linspace() kamu dapat mencoba beberapa latihan berikut:

1. Buat array dengan np.arange() dari 0 sampai 10 dengan jarak 1 dan cetak array tersebut
2. Buat array dengan np.arange() dari 0 sampai 10 dengan jarak 2 dan cetak array tersebut
3. Buat array dengan np.linspace() dari 0 sampai 10 dengan jumlah elemen 5 dan cetak array tersebut
4. Buat array dengan np.linspace() dari 0 sampai 10 dengan jumlah elemen 10 dan cetak array tersebut
5. Buat array dengan np.linspace() dari 0 sampai 10 dengan jumlah elemen 20 dan cetak array tersebut
6. Buat array dengan np.arange() dari 0 sampai 10 dengan jarak 0.5 dan cetak array tersebut.

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_arange\_1 = np.arange(0,11,1) print(arr\_arange\_1)  # Latihan 2 arr\_arange\_2 = np.arange(0,11,2) print(arr\_arange\_2)  # Latihan 3 arr\_linspace\_3 = np.linspace(0,10,5) print(arr\_linspace\_3)  # Latihan 4 arr\_linspace\_4 = np.linspace(0,10,10) print(arr\_linspace\_4)  # Latihan 5 arr\_linspace\_5 = np.linspace(0,10,20) print(arr\_linspace\_5)  # Latihan 6 arr\_arange\_6 = np.ar |

Itulah beberapa **cara untuk membuat array dengan Numpy.** Kamu dapat memilih metode yang sesuai dengan kebutuhan kamu untuk membuat array.

# Membentuk kembali array numpy(Reshaping numpy arrays)

Membentuk kembali seperti namanya berarti mengubah bentuk array numpy mengingat bentuk baru kompatibel dengan bentuk lama.

Kamu dapat membentuk kembali array Numpy menggunakan fungsi numpy.reshape(). Fungsi ini mengambil dua parameter, yaitu jumlah baris dan jumlah kolom yang diinginkan. Contohnya:

|  |
| --- |
| **import** numpy **as** np  # Membuat array 1-9 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])  # Membentuk kembali menjadi array 3×3 arr = np.reshape(arr, (3, 3)) print(arr) # Output: # [[1 2 3] #  [4 5 6] #  [7 8 9]] |

Contoh lainnya: misalkan awalnya bentuk array numpy adalah (3,4) yang berarti ia memiliki 12 elemen (3\*4), sekarang bentuk yang diizinkan harus sedemikian rupa sehingga memiliki hasil kali 12 yaitu bentuk yang diizinkan adalah (1 ,12),(2,6),(3,4),(4,3),(6,2),(12,1).

Reshaping numpy array

Kamu  juga dapat **menggunakan atribut shape untuk mengubah ukuran array tanpa mengubah array asli.**

|  |
| --- |
| **import** numpy **as** np  # Membuat array 1-9 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])  # Membentuk kembali menjadi array 3×3 arr = np.reshape(arr, (3, 3)) print(arr) # Output: # [[1 2 3] #  [4 5 6] #  [7 8 9]] |

**Latihan Praktek**

Untuk melakukan latihan praktek membentuk kembali array numpy kamu dapat mencoba beberapa latihan berikut:

1. Buat array 1 dimensi dengan panjang 9 dan isi dengan angka 1-9, kemudian ubah menjadi array 2×4
2. Buat array 1 dimensi dengan panjang 12 dan isi dengan angka 1-12, kemudian ubah menjadi array 3×4
3. Buat array 1 dimensi dengan panjang 16 dan isi dengan angka 1-16, kemudian ubah menjadi array 4×4
4. Buat array 1 dimensi dengan panjang 10 dan isi dengan angka 1-10, kemudian ubah menjadi array 2×5
5. Buat array 1 dimensi dengan panjang 15 dan isi dengan angka 1-15, kemudian ubah menjadi array 5×3

Contoh:

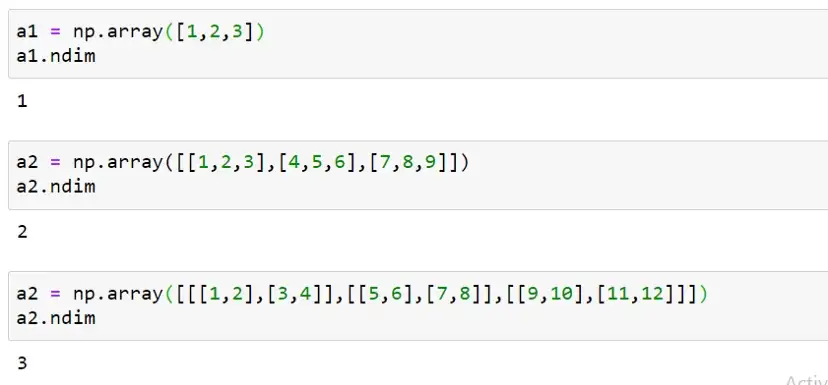
|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_1 = np.array(range(1,10)) arr\_1 = np.reshape(arr\_1, (2,4)) print(arr\_1)  # Latihan 2 arr\_2 = np.array(range(1,13)) arr\_2 = np.reshape(arr\_2, (3,4)) print(arr\_2)  # Latihan 3 arr\_3 = np.array(range(1,17)) arr\_3 = np.reshape(arr\_3, (4,4)) print(arr\_3)  # Latihan 4 arr\_4 = np.array(range(1,11)) arr\_4 = np.reshape(arr\_4, (2,5)) print(arr\_4)  # Latihan 5 arr\_5 = np.array(range(1,16)) arr\_5 = np.reshape(arr\_5, (5,3)) print(arr\_5) |

Itulah beberapa latihan praktek yang dapat kamu coba untuk membentuk kembali array numpy. Ingatlah untuk selalu memeriksa ukuran array yang diinginkan sebelum membentuk kembali array agar hasil sesuai dengan yang diharapkan.

# Mengakses Array

ndim

ndim digunakan untuk menentukan jumlah dimensi dalam array yang diberikan. Kesalahpahaman mengenai ndim dianggap sebagai baris dan kolom. Tapi sebenarnya, itu menunjukkan kedalaman array.



Fungsi numpy.ndim dapat digunakan untuk mengetahui dimensi dari suatu array. Fungsi ini akan mengembalikan jumlah dimensi dari array yang diberikan sebagai input.

Contohnya:

|  |
| --- |
| **import** numpy **as** np  # Membuat array 2 dimensi arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(np.ndim(arr)) # Output: 2  # Membuat array 1 dimensi arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9]) print(np.ndim(arr)) # Output: 1 |

Kamu dapat menggunakan ndim untuk mengecek dimensi array sebelum melakukan operasi pada array tersebut, sehingga kamu dapat memastikan bahwa operasi yang kamu lakukan sesuai dengan dimensi array yang diinginkan.

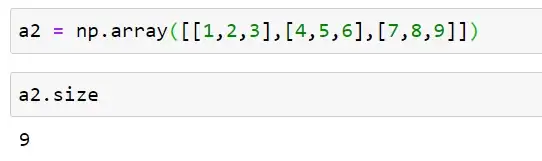
Itemsize

Menentukan ukuran setiap elemen dalam byte yang dikonsumsi saat menyimpan data. Item pada gambar di bawah ini adalah bilangan bulat, sehingga byte yang dikonsumsi perangkat adalah 4 byte per item.



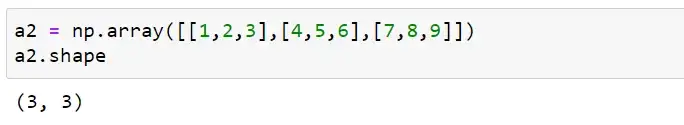
Size

Menentukan jumlah elemen yang ada dalam array. Pada gambar di bawah, ada total 9 elemen dan setiap array dalam masing-masing terdiri dari 3 elemen.

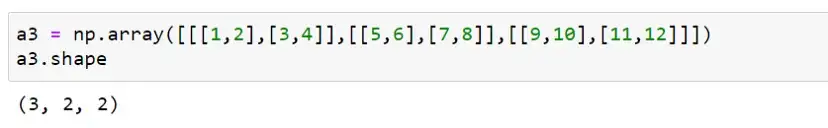


Shape

Menentukan jumlah baris dan kolom dari array yang ditentukan.



Array berikut juga berisi array dalam, jadi menentukan shape sebagai jumlah array, jumlah baris, dan jumlah kolom.



**Latihan praktek**

Untuk melakukan latihan praktek mengakses array dengan ndim, itemsize, size, shape Anda dapat mencoba beberapa latihan berikut:

1. Buat array 2 dimensi dengan ukuran 3×3 dan cetak jumlah dimensi menggunakan np.ndim()
2. Buat array 1 dimensi dengan panjang 8 dan cetak jumlah dimensi menggunakan np.ndim()
3. Buat array 2 dimensi dengan ukuran 4×5 dan cetak ukuran satu elemen menggunakan np.itemsize()
4. Buat array 1 dimensi dengan panjang 10 dan cetak ukuran satu elemen menggunakan np.itemsize()
5. Buat array 2 dimensi dengan ukuran 3×4 dan cetak jumlah elemen dalam array menggunakan np.size()
6. Buat array 1 dimensi dengan panjang 6 dan cetak jumlah elemen dalam array menggunakan np.size()
7. Buat array 2 dimensi dengan ukuran 3×3 dan cetak ukuran array menggunakan np.shape()
8. Buat array 1 dimensi dengan panjang 7 dan cetak ukuran array menggunakan np.shape()

Contoh:

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(np.ndim(arr\_2d))  # Latihan 2 arr\_1d = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9]) print(np.ndim(arr\_1d))  # Latihan 3 arr\_2d\_b = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19, 20]]) print(np.itemsize(arr\_2d\_b))  # Latihan 4 arr\_1d\_b = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) print(np.itemsize(arr\_1d\_b))  # Latihan 5 arr\_2d\_c = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]) print(np.size(arr\_2d\_c))  # Latihan 6 arr\_1d\_c = np.array([1, 2, 3, 4, 5, 6]) print(np.size(arr\_1d\_c))  # Latihan 7 arr\_2d\_d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print( |

**Mengakses elemen tertentu dalam baris dan kolom**

Di dalam library Python Numpy, kamu dapat mengakses elemen tertentu di baris dan kolom dengan menggunakan indexing array. Pengindeksan dimulai dari 0, sehingga untuk mengakses elemen pertama dalam baris atau kolom, kamu harus menggunakan indeks 0.

Untuk mengakses elemen tertentu dalam array Numpy, kamu dapat menggunakan tanda kurung siku dan indeks yang sesuai. Misalnya, jika kamu memiliki sebuah array Numpy yang disebut “a”, kamu dapat mengakses elemen pada baris ke-i dan kolom ke-j dengan menggunakan a[i, j].

Contoh:

|  |
| --- |
| **import** numpy **as** np  a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  # Mengakses elemen pada baris ke-1 dan kolom ke-2 element = a[1, 2] print(element) # Output: 6 |

Kamu juga dapat menggunakan slicing untuk mengakses beberapa elemen pada baris atau kolom. Contohnya, jika kamu ingin mengakses elemen dari baris ke-1 sampai ke-2, kamu dapat menggunakan a[1:3, :], yang akan mengambil semua elemen dari baris ke-1 sampai ke-2. Atau jika kamu ingin mengambil semua elemen dari kolom ke-2, kamu dapat menggunakan a[:, 1].

Beberapa contoh slicing di Numpy :

|  |
| --- |
| **import** numpy **as** np  a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  # Mengambil semua elemen dari baris ke-1 sampai ke-2 row\_elements = a[1:3, :] print(row\_elements) # Output: # [[4 5 6] #  [7 8 9]]  # Mengambil semua elemen dari kolom ke-2 col\_elements = a[:, 1] print(col\_elements) # Output: [2 5 8] |

Itulah cara mengakses elemen tertentu dalam baris dan kolom di library Python Numpy. Ingatlah bahwa indeksing dimulai dari 0, dan kamu dapat menggunakan slicing untuk mengakses beberapa elemen dalam baris atau kolom.

**Latihan praktek**

Untuk melakukan latihan praktek mengakses elemen tertentu dalam baris dan kolom dalam array Numpy, kamu dapat mencoba beberapa latihan berikut:

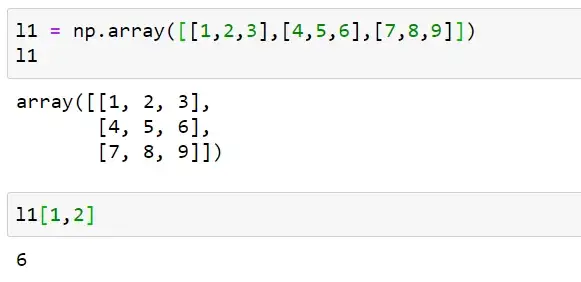
1. Buat array 2 dimensi dengan ukuran 3×3 dan cetak elemen pada baris ke-2 dan kolom ke-3
2. Buat array 2 dimensi dengan ukuran 4×4 dan cetak elemen pada baris ke-1 dan kolom ke-4
3. Buat array 2 dimensi dengan ukuran 5×5 dan cetak elemen pada baris ke-3 dan kolom ke-2
4. Buat array 2 dimensi dengan ukuran 3×3 dan cetak elemen pada baris ke-1 dan kolom ke-1 sampai ke-3
5. Buat array 2 dimensi dengan ukuran 4×4 dan cetak elemen pada baris ke-2 sampai ke-4 dan kolom ke-1

Contoh:

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(arr\_2d[1][2])  # Latihan 2 arr\_2d\_b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]) print(arr\_2d\_b[0][3])  # Latihan 3 arr\_2d\_c = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19, 20], [21, 22, 23, 24, 25]]) print(arr\_2d\_c[2][1])  # Latihan 4 arr\_2d\_d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(arr\_2d\_d[0, 0:3])  # Latihan 5 arr\_2d\_e = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, |

**Mendapatkan elemen tertentu: [r,c]**

Kamu bisa mendapatkan elemen tertentu dalam array tertentu dengan menentukan baris dan kolom yang diwakili oleh elemen tersebut. Pada gambar di bawah, lokasi elemen ditunjukkan dengan nomor baris dan kolom.



**Latihan praktek**

Untuk melakukan latihan praktek mendapatkan elemen tertentu dalam array Numpy menggunakan notasi [r,c], kamu dapat mencoba beberapa latihan berikut:

1. Buat array 2 dimensi dengan ukuran 3×3 dan cetak elemen pada baris ke-2 dan kolom ke-3 menggunakan notasi [r,c]
2. Buat array 2 dimensi dengan ukuran 4×4 dan cetak elemen pada baris ke-1 dan kolom ke-4 menggunakan notasi [r,c]
3. Buat array 2 dimensi dengan ukuran 5×5 dan cetak elemen pada baris ke-3 dan kolom ke-2 menggunakan notasi [r,c]
4. Buat array 2 dimensi dengan ukuran 3×3 dan cetak elemen pada baris ke-1 dan kolom ke-1 sampai ke-3 menggunakan notasi [r,c]
5. Buat array 2 dimensi dengan ukuran 4×4 dan cetak elemen pada baris ke-2 sampai ke-4 dan kolom ke-1 menggunakan notasi [r,c]

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(arr\_2d[1, 2])  # Latihan 2 arr\_2d\_b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]) print(arr\_2d\_b[0, 3])  # Latihan 3 arr\_2d\_c = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19, 20], [21, 22, 23, 24, 25]]) print(arr\_2d\_c[2, 1])  # Latihan 4 arr\_2d\_d = np.array([[1, 2, 3], |

**Dapatkan elemen di baris tertentu: a[r,:]**

Kamu bisa mendapatkan elemen di baris tertentu dengan memasukkan nomor baris dan menyetel parameter kolom sebagai di argumen kedua.



**Latihan praktek**

Untuk melakukan latihan praktek mendapatkan elemen di baris tertentu dalam array Numpy menggunakan notasi a[r,:], kamu dapat mencoba beberapa latihan berikut:

1. Buat array 2 dimensi dengan ukuran 3×3 dan cetak elemen pada baris ke-2 menggunakan notasi a[r,:]
2. Buat array 2 dimensi dengan ukuran 4×4 dan cetak elemen pada baris ke-1 menggunakan notasi a[r,:]
3. Buat array 2 dimensi dengan ukuran 5×5 dan cetak elemen pada baris ke-3 menggunakan notasi a[r,:]
4. Buat array 2 dimensi dengan ukuran 3×3 dan cetak elemen pada baris ke-1 dan ke-3 menggunakan notasi a[r,:]
5. Buat array 2 dimensi dengan ukuran 4×4 dan cetak elemen pada baris ke-2 sampai ke-4 menggunakan notasi a[r,:]

Contoh:

|  |
| --- |
| import numpy as np  # Latihan 1 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(arr\_2d[1, :])  # Latihan 2 arr\_2d\_b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]) print(arr\_2d\_b[0, :])  # Latihan 3 arr |

**Mendapatkan elemen berdasarkan kolom  tertentu: a[:,c]**

Kita bisa memperoleh elemen berdasarkan baris  tertentu dengan cara memasukkan nomor kolom dan menyetel parameter baris sebagai: argumen pertama.



**Latihan praktek**

Untuk melakukan latihan praktek mendapatkan elemen di kolom tertentu dalam array Numpy menggunakan notasi a[:,c], kamu dapat mencoba beberapa latihan berikut:

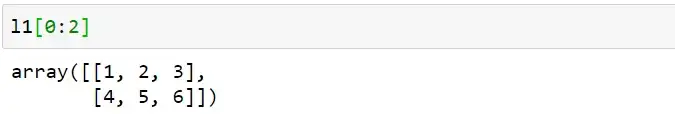
1. Buat array 2 dimensi dengan ukuran 3×3 dan cetak elemen pada kolom ke-2 menggunakan notasi a[:,c]
2. Buat array 2 dimensi dengan ukuran 4×4 dan cetak elemen pada kolom ke-1 menggunakan notasi a[:,c]
3. Buat array 2 dimensi dengan ukuran 5×5 dan cetak elemen pada kolom ke-3 menggunakan notasi a[:,c]
4. Buat array 2 dimensi dengan ukuran 3×3 dan cetak elemen pada kolom ke-1 dan ke-3 menggunakan notasi a[:,c]
5. Buat array 2 dimensi dengan ukuran 4×4 dan cetak elemen pada kolom ke-2 sampai ke-4 menggunakan notasi a[:,c]

Contoh:

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) result = np.where(arr > 5, ‘big’, ‘small’) print(result)  # Latihan 2 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) result\_2d = np.where(arr\_2d > 5, 0, 1) print(result\_2d)  # Latihan 3 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) result = np.where(arr < 5) print(result)  # Latihan 4 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) result\_2d = np.where(arr\_2d[:,1] > 5) print(result\_2d)  # Latihan 5 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) result = arr[np.where(arr > 5)] print(result) |

**Dapatkan jumlah elemen dalam array yang ditentukan: a[start:end]**

Masukkan indeks awal dan indeks akhir untuk mendapatkan jumlah baris. Elemen terindeks terakhir tidak disertakan dalam output yang dihasilkan. Misalnya, pada gambar di bawah ini, nomor baris dimulai dari 0 dan diakhiri dengan 1. Perhatikan bahwa indeks akhir (yaitu nomor baris 2 dalam contoh ini) diabaikan.



**Latihan praktek**

Untuk melakukan latihan praktek mendapatkan jumlah elemen dalam array yang ditentukan dalam array Numpy menggunakan notasi a[start:end], kamu dapat mencoba beberapa latihan berikut:

1. Buat array 1 dimensi dengan panjang 10 dan cetak elemen ke-3 sampai ke-7 menggunakan notasi a[mulai:akhir]
2. Buat array 2 dimensi dengan ukuran 3×3 dan cetak elemen pada baris ke-1 dan kolom ke-2 sampai ke-3 menggunakan notasi a[start:end]
3. Buat array 1 dimensi dengan panjang 15 dan cetak elemen ke-5 sampai ke-10 menggunakan notasi a[start:end]
4. Buat array 2 dimensi dengan ukuran 4×4 dan cetak elemen pada baris ke-2 dan kolom ke-1 sampai ke-3 menggunakan notasia[start:end]
5. Buat array 1 dimensi dengan panjang 20 dan cetak elemen ke-10 sampai ke-15 menggunakan notasi a[start:end]

Contoh:

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_1d = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) print(arr\_1d[2:7])  # Latihan 2 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(arr\_2d[0, 1:3])  # Latihan 3 arr\_1d\_b = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]) print(arr\_1d\_b[4:10])  # Latihan 4 arr\_2d\_b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]) print(arr\_2d\_b[1, 0:3])  # Latihan 5 arr\_1d\_c = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]) print(arr\_1d\_c[9:15]) |

Kamu dapat mencoba mengubah ukuran array serta mengubah rentang elemen yang diambil.

**Dapatkan hanya baris terakhir dari sebuah array: a[-1]**

Kamu bisa mendapatkan baris terakhir dari array tertentu dengan meneruskan -1 sebagai input ke array yang kamu buat.



**Latihan praktek**

Untuk melakukan latihan praktek mendapatkan baris terakhir dari sebuah array Numpy, kamu dapat mencoba beberapa latihan berikut:

1. Buat array 2 dimensi dengan uku ran 3×3 dan cetak baris terakhir menggunakan notasi a[-1]
2. Buat array 2 dimensi dengan ukuran 4×4 dan cetak baris terakhir menggunakan notasi a[-1]
3. Buat array 2 dimensi dengan ukuran 5×5 dan cetak baris terakhir menggunakan notasi a[-1]
4. Buat array 1 dimensi dengan panjang 10 dan cetak elemen terakhir menggunakan notasi a[-1]
5. Buat array 2 dimensi dengan ukuran 4×4 dan cetak baris terakhir dari kolom ke-2 sampai ke-4 menggunakan notasi a[-1]

Contoh:

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(arr\_2d[-1])  # Latihan 2 arr\_2d\_b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]) print(arr\_2d\_b[-1])  # Latihan 3 arr\_2d\_c = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19, 20], [21, 22, 23, 24, 25]]) print(arr\_2d\_c[-1])  # Latihan 4 arr\_1d = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) print(arr\_1d[-1])  # Latihan 5 arr\_2d\_d = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]) print(arr\_2d\_d[-1, 1:4]) |

Kamu dapat mencoba mengubah ukuran array dan juga mengubah notasi yang digunakan untuk mendapatkan baris terakhir dari array.

**Dapatkan data dari kolom tertentu dalam beberapa baris: a[0:2,1]**

Dapatkan kolom tertentu dengan menentukan nomor baris sebagai argumen pertama dan kolom yang diharapkan sebagai argumen kedua kamu bisa mendapatkan beberapa baris data untuk sebuah kolom.



**Latihan Praktek**

Untuk melakukan latihan praktek mendapatkan data dari kolom tertentu dalam beberapa baris dalam array Numpy menggunakan notasi a[0:2,1], kamu dapat mencoba beberapa latihan berikut:

1. Buat array 2 dimensi dengan ukuran 3×3 dan cetak data dari kolom ke-2 dari baris ke-1 sampai ke-2 menggunakan notasi a[0:2,1]
2. Buat array 2 dimensi dengan ukuran 4×4 dan cetak data dari kolom ke-1 dari baris ke-3 sampai ke-4 menggunakan notasi a[2:4,0]
3. Buat array 2 dimensi dengan ukuran 5×5 dan cetak data dari kolom ke-3 dari baris ke-1 sampai ke-3 menggunakan notasi a[0:3,2]
4. Buat array 2 dimensi dengan ukuran 3×3 dan cetak data dari kolom ke-1 dari baris ke-2 sampai ke-3 menggunakan notasi a[1:3,0]
5. Buat array 2 dimensi dengan ukuran 4×4 dan cetak data dari kolom ke-2 sampai ke-4 dari baris ke-1 menggunakan notasi a[0,1:4]

Contoh:

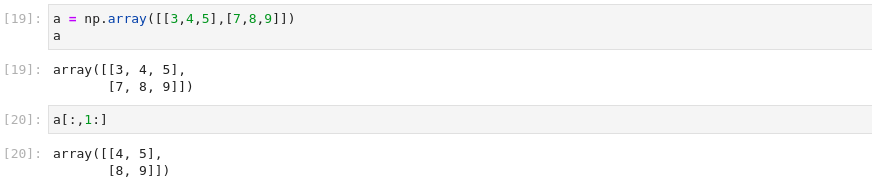
|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(arr\_2d[0:2, 1])  # Latihan 2 arr\_2d\_b = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]) print(arr\_2d\_b[2:4, 0])  # Latihan 3 arr\_2d\_c = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19, 20], [21, 22, 23, 24, 25]]) print(arr\_2d\_c[0:3, 2])  # Latihan 4 arr\_ |

# Pengindeksan dan Seleksi (****Indexing and Selection****)

* Pengindeksan dan Seleksi adalah proses untuk mengambil bagian dari data array atau matriks dalam Python menggunakan library Numpy. Ini dilakukan dengan menggunakan notasi indeks yang sama seperti yang digunakan pada list Python.
* Kamu dapat mengakses elemen tunggal dari array atau matriks dengan menggunakan notasi indeks tunggal, seperti a[i] atau a[i,j] untuk array atau matriks 2D. Anda juga dapat mengambil bagian dari array atau matriks dengan menggunakan notasi indeks slice, seperti a[start:end] atau a[start:end,start:end] untuk array atau matriks 2D.
* Selain itu, kamu juga dapat menggunakan logika seleksi pada array atau matriks untuk mengambil hanya elemen yang memenuhi syarat tertentu. Contohnya, mengambil semua elemen dari array yang lebih besar dari 5 dengan menggunakan a>5.

Ini adalah beberapa teknik yang digunakan dalam pengindeksan dan seleksi dalam Numpy. Latihan praktek yang tepat akan tergantung pada tipe data yang kamu miliki dan analisis yang ingin kamu lakukan.

Mengindeks dan mengiris di numpy berfungsi sama seperti di list python. Array Numpy mengikuti pengindeksan 0.



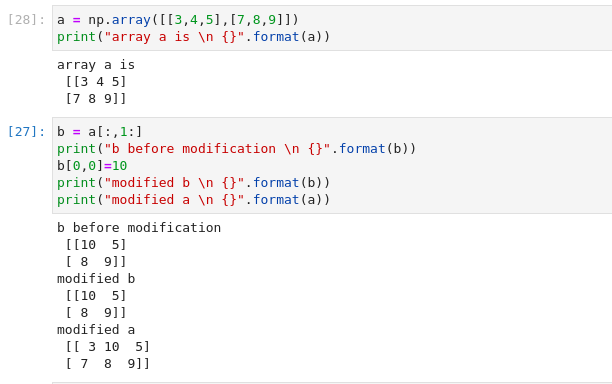
Dalam contoh di atas kita memiliki sebuah array bentuk (2,3). Sekarang param pertama dalam mengiris adalah untuk baris dan yang kedua adalah untuk kolom. Jadi dalam contoh, kita ingin semua baris itu sebabnya kita memiliki [:] yang berarti semua baris dan kita ingin kolom dengan indeks 1 sampai akhir itu sebabnya kita memiliki [1:] menggabungkan dua ini kami memiliki [:, 1:].

Perlu diingat

Pemotongan diterapkan berdasarkan dimensi dalam numpy yaitu untuk dimensi 1 (baris) kita memiliki logika pemotongan dan untuk dimensi 2 (kolom) kita memiliki logika pemotongan yang berbeda. Dengan cara yang sama, kita harus menangani pemotongan untuk array N-d di NumPy.

Pertanyaan jebakan(**Trick question)**

Referensi objek irisan NumPy ke elemen array utama. Jadi jika kita membuat perubahan apa pun pada array yang diiris, itu akan tercermin dalam array utama.



Untuk mengatasi masalah ini kita dapat menggunakan metode array.copy() untuk melakukan salinan mendalam dari array.

# Ekspresi Bersyarat di NumPy (Conditional Expressions)

Ekspresi bersyarat dalam NumPy digunakan untuk mengevaluasi suatu kondisi dan mengembalikan nilai yang sesuai. Ini sering digunakan untuk mengekstrak atau mengubah elemen array yang memenuhi syarat tertentu.

Ekspresi bersyarat dalam NumPy dapat ditulis menggunakan notasi numpy.where(condition, x, y). Kondisi dalam ekspresi bersyarat adalah sebuah boolean array yang menentukan elemen mana yang akan diproses. Jika kondisi terpenuhi pada suatu elemen, maka elemen tersebut akan diisi dengan x, jika tidak maka akan diisi dengan y.

Contoh:

|  |
| --- |
| **import** numpy **as** np  arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) result = np.where(arr > 5, ‘big’, ‘small’) print(result) |

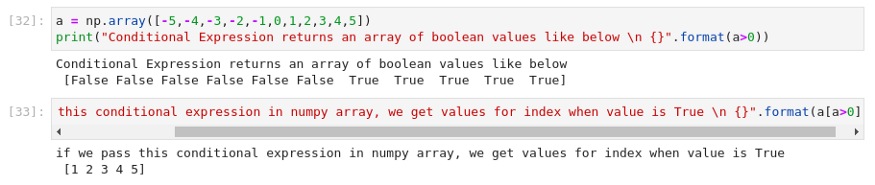
Output

|  |
| --- |
| [‘small’ ‘small’ ‘small’ ‘small’ ‘small’ ‘big’ ‘big’ ‘big’ ‘big’ ‘big’] |

Kamu juga dapat menggunakan ekspresi bersyarat pada array multidimensional, yang akan mengevaluasi setiap elemen dalam array dan mengembalikan hasil yang sesuai.

Ekspresi bersyarat dalam NumPy dapat digunakan untuk melakukan operasi seperti mengubah nilai elemen array yang memenuhi syarat tertentu, mengekstrak elemen yang memenuhi syarat tertentu, atau menghitung jumlah elemen yang memenuhi syarat tertentu.

Misalkan kamu memiliki larik yang berisi elemen negatif dan positif, dan kamu hanya menginginkan elemen positif dari larik. Nah, NumPy menyediakan cara yang elegan untuk melakukan operasi bersyarat ini. Lihat gambar di bawah ini untuk ide yang lebih baik.



Conditional Expressions

**Latihan praktek**

Untuk melakukan latihan praktek ekspresi bersyarat dalam NumPy, kamu dapat mencoba beberapa latihan berikut:

1. Buat sebuah array 1 dimensi dengan panjang 10 dan gantikan semua elemen yang lebih besar dari 5 dengan “big” dan yang lainnya dengan “small” menggunakan ekspresi bersyarat numpy.where()
2. Buat array 2 dimensi dengan ukuran 3×3 dan gantikan semua elemen yang lebih besar dari 5 dengan 0 dan yang lainnya dengan 1 menggunakan ekspresi bersyarat numpy.where()
3. Buat array 1 dimensi dengan panjang 10 dan hitung jumlah elemen yang lebih kecil dari 5 menggunakan ekspresi bersyarat numpy.where()
4. Buat array 2 dimensi dengan ukuran 3×3 dan hitung jumlah elemen yang lebih besar dari 5 pada kolom ke-2 menggunakan ekspresi bersyarat numpy.where()
5. Buat array 1 dimensi dengan panjang 10 dan ekstrak semua elemen yang lebih besar dari 5 menggunakan ekspresi bersyarat numpy.where()

Contoh:

|  |
| --- |
| **import** numpy **as** np  # Latihan 1 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) result = np.where(arr > 5, ‘big’, ‘small’) print(result)  # Latihan 2 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) result\_2d = np.where(arr\_2d > 5, 0, 1) print(result\_2d)  # Latihan 3 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) result = np.where(arr < 5) print(result)  # Latihan 4 arr\_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) result\_2d = np.where(arr\_2d[:,1] > 5) print(result\_2d)  # Latihan 5 arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) result = arr[np.where(arr > 5)] print(result) |

# Memahami Properti dasar numPy

Python Numerik adalah library python open source yang menyediakan fungsi matematika dan numerik umum. Ini memperkenalkan jenis struktur data yang disebut array numPy yang dioptimalkan untuk perhitungan. Pada materi ini, kita akan melihat empat sifat berguna dari numPy yang menjadikannya alat yang ampuh dalam ilmu data.

kamu dapat mengimpor modul dengan berbagai cara seperti yang ditunjukkan di bawah ini. Untuk pembahasan ini, kita menggunakan np mewakili modul numPy dan ini adalah konvensi yang disarankan untuk mengimpor numPy.

|  |
| --- |
| **import** numpy **as** np **import** numpy |

## 1. Homogen

Array NumPy hanya dapat menampung elemen dari satu tipe data, biasanya data numerik seperti integer dan float, tetapi juga dapat menampung string.

Kode di bawah ini membuat array numPy menggunakan np.array(list).

**Contoh**

|  |
| --- |
| array\_1 = np.array([1,2,3,4]) array\_1 ###Results array([1, 2, 3, 4]) |

Perhatikan bahwa jika daftar berisi campuran tipe numerik dan string seperti di bawah ini, numPy akan mengonversi semua elemen menjadi string.

**Contoh**

|  |
| --- |
| array\_2 = np.array([1,2,’three’]) array\_2 ###Results array([‘1’, ‘2’, ‘three’], dtype='<U11′) |

## 2. Multidimensi

Sejauh ini, kita hanya membuat array 1 dimensi, tetapi kamu juga dapat memiliki array numPy lebih dari 1 dimensi. Salah satu caranya adalah memberi membuat list dari list seperti pada np.array(list dari list, tipe data).

**Contoh**

|  |
| --- |
| array\_5 = np.array([[1,2,3],[4,5,6]]) print(array\_5) ###Results [[1, 2, 3], [4, 5, 6]] |

## 3. Operasi adalah Elemen wise

Fungsi numPy diterapkan ke setiap elemen array. Misalnya np.sqrt(array) mengembalikan array baru di mana setiap elemen dikuadratkan. Untuk operasi antar larik, matematika berada di antara elemen yang sesuai dari setiap larik.

Sebagai contoh di bawah ini kita membuat 2 array dan kemudian melakukan operasi di antara keduanya.

**Contoh**

|  |
| --- |
| arr\_1 = np.array([[1,2],[3,4]]) arr\_2 = np.array([[5,6],[7,8]]) print(arr\_1) print(arr\_2) ###Results [[1 2] [3 4]] [[5 6] [7 8]] |

Library numPy dikemas dengan beberapa fungsi pembantu (helper function) untuk melakukan operasi matematika seperti yang ditunjukkan dalam[dokumentasi ini](https://numpy.org/doc/stable/reference/routines.math.html).

Beberapa fungsi melakukan[operasi elemen wise](https://www.pythonprogramming.in/numpy-element-wise-mathematical-operations.html) seperti np.sqrt(), np.abs(), np.power(), np.exp() dan np.log().

**Contoh**

|  |
| --- |
| arr\_5 = np.array([2.4, 7.5, 2.8, 1.9]).reshape(2,2) print(np.power(arr\_5, 2), ‘\n’) print(np.log(arr\_5), ‘\n’) print(np.power(arr\_5, 3), ‘\n’) ###Results [[ 5.76 56.25] [ 7.84  3.61]]  [[0.87546874 2.01490302] [1.02961942 0.64185389]]  [[ 13.824 421.875] [ 21.952   6.859]] |

Fungsi lainnya adalah [fungsi agregasi](https://www.pythonprogramming.in/numpy-aggregate-and-statistical-functions.html)yang meringkas array dan mengembalikan satu angka. kamu juga dapat memilih untuk meringkas menurut baris atau kolom dengan menyertakan masing-masing sumbu=0 atau sumbu=1 sebagai parameter. Fungsi agregasi umum adalah np.mean(), np.median(), np.min(), np.max() dan np.std().

**Contoh**

|  |
| --- |
| print(np.max(arr\_5)) print(np.max(arr\_5, axis=1)) ###Results 7.5 [7.5 2.8] |

## 4. Sangat Acak (Dependably Random)

Numpy hadir dengan modul pengacakan (a **randomization**module) yang disebut numpy.random untuk pengambilan sampel atau menghasilkan sampel data acak.

Untuk membuat larik bilangan bulat acak, gunakan np.random.randint(low, high, size). low adalah batas bawah dan inklusif. High adalah batas yang lebih tinggi dan eksklusif. size adalah jumlah elemen array, dan kamu dapat memberikan hanya satu angka seperti 16 untuk membuat array satu dimensi, atau memberikan bentuk seperti (4,4) untuk membuat array dua dimensi.

[Lihat dokumentasi ini](https://numpy.org/doc/1.16/reference/routines.random.html)untuk semua rutinitas pengambilan sampel acak numPy.

Contoh

|  |
| --- |
| arr\_1 = np.random.randint(0,5,6) arr\_3 = np.random.randint(0,5,(3,2)) print(arr\_1) print(arr\_3) ###Results [0 4 1 3 1 2] [[3 3] [1 1] [3 2]] |

# Persamaan List dan Numpy

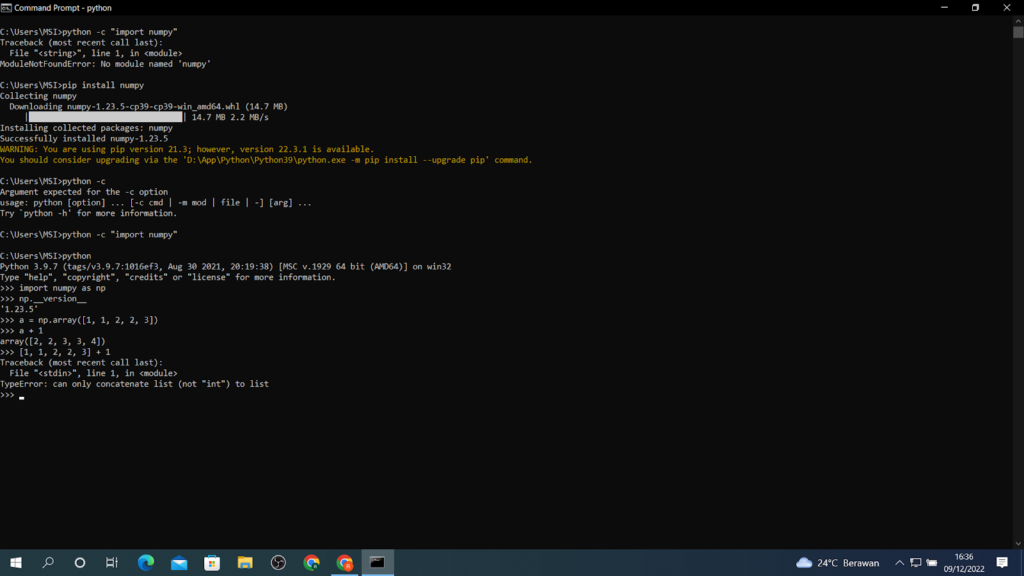
Ini menyiratkan bahwa apa pun yang dapat dilakukan dalam list python juga dapat dilakukan dalam array numpy, termasuk: mendapatkan elemen ke-n dalam list/array dengan tanda kurung siku, mengiris list/array, mengulang melalui list/array dengan start, stop, step, menggunakan operator in untuk menemukan list/array keanggotaan, memeriksa panjang dan membongkar list/array (unpacking list/arrays).

Contoh

|  |
| --- |
| # setup **import** numpy **as** np # membuat pembanding (create comparables) python\_list = [1,2,3,4,5,6,7,8,9] numpy\_array = np.array([1,2,3,4,5,6,7,8,9]) # operasi braket (bracket operations) # dapatkan elemen ke-n dengan braket persegi (get nth element with square bracket) python\_list[0] # 1 numpy\_array[0] # 1 python\_list[8] # 9 numpy\_array[8] # 9 python\_list[-1] # 9 numpy\_array[-1] # 9 # tanda kurung persegi untuk mengiris (square bracket to slice) python\_list[:3] # [1, 2, 3] numpy\_array[:3] # array([1, 2, 3]) python\_list[1:5] # [2, 3, 4, 5] numpy\_array[1:5] # array([2, 3, 4, 5]) # start, stop, step python\_list[1:8:2] # [2, 4, 6, 8] numpy\_array[1:8:2] # array([2, 4, 6, 8]) # gunakan in operator untuk memeriksa keanggotaan (use in operator to check membership) 1 **in** python\_list # true 1 **in** numpy\_array # true 0 **in** python\_list # false 0 **in** numpy\_array # false # menemukan panjang (finding length) len(python\_list) # 9 len(numpy\_array) # 9 # unpacking x,y = [1,2] # sekarang x is 1, y is 2 w,z = np.array([1,2]) #sekarang w is 1, z is 2 |

# Perbedaan List Vs Numpy

Sekarang, inilah perbedaannya.



Tugas-tugas ini dapat dilakukan dalam daftar python, tetapi memerlukan pendekatan yang berbeda untuk larik NumPy termasuk: modifikasi (perpanjang dalam daftar, tambahkan untuk larik). Akhirnya, list dapat menyimpan tipe data campuran, sedangkan larik NumPy akan dikonversi menjadi string.

Hal-hal penting tentang array Numpy:

* Kita bisa membuat array N-dimensi dengan python menggunakan numpy.array().
* Data di dalam array harus dari Tipe Data yang sama yaitu, Homogen.
* Dimungkinkan untuk melakukan elemen operasi dengan bijaksana.
* Untuk perhitungan matriks yang mudah Numpy array memiliki berbagai fungsi, metode, dan variabel.
* Dalam elemen memori array disimpan secara berdekatan.

 Keuntungan menggunakan Numpy Arrays Over Python Lists:

* mengkonsumsi lebih sedikit memori.
* cepat dibandingkan dengan Daftar python.
* nyaman digunakan.

List:

List  adalah kumpulan item yang dipesan dan diubah. Dalam Python, daftar diapit dengan tanda kurung siku.

Hal-hal penting tentang Daftar Python:

* List  itu mungkin homogen atau heterogen.
* Tidak mungkin melakukan operasi berdasarkan elemen pada list.
* Daftar Python adalah 1 dimensi secara default. Namun kita dapat membuat list N-Dimensi.
* Elemen list tidak harus bersebelahan dalam memori.

List dan Numpy adalah dua jenis data yang digunakan dalam pemrograman Python. Keduanya memiliki kelebihan dan kekurangan masing-masing.

List adalah tipe data bawaan Python yang digunakan untuk menyimpan elemen daftar. List dapat menyimpan elemen dengan tipe data yang berbeda dan dapat digunakan untuk melakukan operasi seperti slicing, concatenation, dan iteration. Namun, daftar tersebut tidak efisien untuk melakukan operasi matematika skala besar.

Numpy, atau Numerical Python, adalah library yang menyediakan tipe data array yang digunakan untuk melakukan operasi matematika skala besar. Array Numpy memiliki tipe data yang sama dan memiliki performa yang lebih baik dari list. Numpy juga menyediakan berbagai fungsi matematika seperti penambahan matriks, operasi statistik, dan lainnya. Namun, Numpy tidak sefleksibel dibandingkan dengan List dalam hal penyimpanan elemen dengan tipe data yang berbeda.

**Latihan Praktek**

|  |
| --- |
| # setup **import** numpy **as** np # create comparables python\_list = [1,2,3,4,5,6,7,8,9] numpy\_array = np.array([1,2,3,4,5,6,7,8,9]) # bracket operations # get nth element with square bracket python\_list[0] # 1 numpy\_array[0] # 1 python\_list[8] # 9 numpy\_array[8] # 9 python\_list[-1] # 9 numpy\_array[-1] # 9 # square bracket to slice python\_list[:3] # [1, 2, 3] numpy\_array[:3] # array([1, 2, 3]) python\_list[1:5] # [2, 3, 4, 5] numpy\_array[1:5] # array([2, 3, 4, 5]) # start, stop, step python\_list[1:8:2] # [2, 4, 6, 8] numpy\_array[1:8:2] # array([2, 4, 6, 8]) # use in operator to check membership 1 **in** python\_list # true 1 **in** numpy\_array # true 0 **in** python\_list # false 0 **in** numpy\_array # false # finding length len(python\_list) # 9 len(numpy\_array) # 9 # unpacking x,y = [1,2] # now x is 1, y is 2 w,z = np.array([1,2]) # now w is 1, z is 2 |

**List Python vs Array NumPy: Apa bedanya**

Sekarang kita tahu bahwa ada perbedaan yang berarti, apa yang bisa kita kaitkan dengan perbedaan ini?[Penjelasan dari UCF ini](https://webcourses.ucf.edu/courses/1249560/pages/python-lists-vs-numpy-arrays-what-is-the-difference)menyoroti perbedaan kinerja termasuk:

Ukuran, Kinerja, Kegunaan.

List Python dan Array Numpy adalah dua jenis struktur data yang digunakan untuk menyimpan dan mengelola data dalam Python. Namun, terdapat beberapa perbedaan utama antara keduanya:

* **Tipe Data**: List Python dapat menyimpan berbagai tipe data, seperti integer, string, boolean, dan lainnya, sedangkan Array Numpy hanya dapat menyimpan satu tipe data saja, seperti integer atau float.
* **Ukuran**: List Python dapat diubah ukurannya dinamis, sedangkan Array Numpy memiliki ukuran yang tetap setelah didefinisikan.
* **Performa**: Array Numpy lebih cepat dalam melakukan operasi matematika dibandingkan dengan list Python, karena array Numpy menggunakan C dan Fortran dibawahnya sehingga lebih cepat dalam melakukan operasi matematika yang kompleks.
* **Fungsionalitas**: Array Numpy memiliki banyak fungsi dan metode yang dapat digunakan untuk mengelola data, seperti operasi matematika, statistik, dan lainnya. Sedangkan List Python hanya memiliki fungsi dasar seperti append, insert, dan lainnya.

Secara umum, jika kamu ingin melakukan operasi matematika kompleks atau bekerja dengan data numerik, Array Numpy adalah pilihan yang lebih baik. Namun, jika kamu ingin menyimpan data yang bervariasi atau tidak dapat diprediksi, List Python adalah pilihan yang lebih baik.

# Method Numpy arange ()

Pada pembahasan materi ini, kami akan mengajari kamu semua yang perlu kamu ketahui tentang metode np.arange(). Struktur data terpenting dalam NumPy adalah tipe array yang disebut numpy.array. Di akhir pembelajaran, kamu akan merasa nyaman mengintegrasikan metode ini ke dalam aplikasi Python kamu.

**Berbagai Cara Menggunakan Metode np.arange() dapat dilakukan dengan:**

* Argumen Step Metode np.arange()
* Tipe Data dan Metode np.arange()
* Cara Menghitung Mundur Dengan Metode np.arange() NumPy
* Cara Menghasilkan Array NumPy Kosong Dengan np.arange()
* Cara Membuat Tuple Menggunakan NumPy’s np.arange()
* Cara Membuat Set Menggunakan NumPy’s np.arange()
* Cara Menggunakan Metode np.arange() Numpy Untuk Menulis Loop
* Perbedaan Antara Fungsi Range Bawaan Python dan Metode np.arange() NumPy

## Cara Menggunakan Metode np.arange()

Dengan menggunakan bahasa pemrograman Numpy, kamu dapat membuat larik yang sangat mendasar berdasarkan rentang numerik yang diteruskan oleh pengguna.

Keluaran dari metode numpy.arange() adalah larik Numpy yang mengembalikan setiap bilangan bulat yang lebih besar atau sama dengan angka **awal**dan kurang dari angka **akhir**.

Metode np.arange() menerima argumen berikut:

* awal: nilai terendah dalam larik NumPy yang dihasilkan
* akhir: nilai tertinggi (eksklusif) dari larik NumPy yang dihasilkan

Contoh:

|  |
| --- |
| arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])  # Mengganti ukuran array menjadi 3×3 arr.shape = (3, 3) print(arr) # Output: # [[1 2 3] #  [4 5 6] #  [7 8 9]] “ |

Dimungkinkan untuk menjalankan metode yang sama sambil mengirimkan satu argumen. Ini memungkinkan kamu membuat larik dengan panjang yang diinginkan, yang berguna dalam menulis perulangan. Dalam hal ini, mulai dan berhentinya array ditentukan oleh angka yang diteruskan sebagai satu-satunya argumen.

Contoh:

|  |
| --- |
| np.arange(1)  #Returns array([0])  np.arange(5)  #Returns array([0, 1, 2, 3, 4])  np.arange(10)  array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) |

Metode np.arange() juga menerima argumen opsional ketiga dan keempat: **step** dan **dtype**. Kita akan mengeksplorasi kedua parameter ini selanjutnya.

### **Method np.arange() dengan argumen Step**

Bahasa pemrograman NumPy memungkinkan kamu menentukan berapa banyak ruang antara setiap elemen array yang dikembalikannya.

|  |
| --- |
| np.arange(0,10)#Returns array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) |

Jika kita ingin menghasilkan list yang sama tetapi hanya menyertakan elemen yang memiliki spasi 5 bilangan bulat di antaranya, kita dapat menentukan dengan menggunakan step=5:

|  |
| --- |
| np.arange(0,10, step=5)  #Returns array([0, 5]) |

Nyatanya, kamu tidak perlu menentukan komponen step= dari argumen. Kode berikut menghasilkan output yang sama:

|  |
| --- |
| np.arange(0,10, step=5)  #Returns array([0, 5]) |

### **Tipe Data dan Method np.arange()**

Atribut dtype memungkinkan kamu untuk menentukan tipe data yang terkandung dalam array. Nilai default untuk dtype adalah Tidak ada, tetapi dapat diakses melalui operator titik. Sebuah array tidak dapat berisi bilangan titik-mengambang dan bilangan bulat ketika terdiri dari dua tipe data tersebut.

Contoh:

|  |
| --- |
| my\_array = np.arange(0,10)  print(my\_array.dtype) |

Kemudian Python akan mencetak int64, yang merupakan nilai dtype untuk array NumPy.

NumPy memiliki beberapa nilai berbeda untuk dtype yang tersedia untuk pengguna. Berikut adalah beberapa contoh untuk bilangan bulat secara khusus:

* np.int8: integer yang ditandatangani dengan 8 bit
* np.uint8: unsigned integer dengan 8 bit
* np.int16: integer yang ditandatangani dengan 16 bit
* np.uint16: unsigned integer dengan 16 bit

Pelajari dari documentation : [Data type objects (**dtype)**](https://numpy.org/doc/stable/reference/arrays.dtypes.html.dtype.html#numpy.dtype)

### **Cara Menghitung Mundur Dengan Metode np.arange() NumPy**

Dimungkinkan untuk menggunakan metode array NumPy untuk membuat array yang mencantumkan elemen-elemennya dalam urutan menurun, bukan urutan menaik, seperti defaultnya. Bagian ini akan menunjukkan cara membuat array NumPy urutan menurun menggunakan metode numpy.

|  |
| --- |
| np.arange(0,10)#Returns array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) |

Untuk membuat larik NumPy, yang perlu kita lakukan adalah membuat daftar elemen mundur dan menentukan nilai negatif untuk argumen langkah. Berikut adalah contoh tampilannya:.

|  |
| --- |
| np.arange(10,0, -1)#Returns array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1]) |

Kita juga bisa menggunakan strategi ini sambil menentukan nilai untuk argumen dtype. Ini contohnya:

|  |
| --- |
| np.arange(10,0, -1, dtype=’int64′)  #Returns array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1]) |

### **Cara Menghasilkan Array NumPy Kosong Dengan np.arange()**

Metode NumPy.arange() memungkinkan kamu membuat array NumPy kosong dengan mudah. Untuk melakukan ini, kamu hanya perlu mencantumkan nomor yang sama dua kali. Dalam hal ini, kamu ingin memulai dengan array kosong dan menambahkannya dari waktu ke waktu.

Kamu dapat dengan mudah membuat array NumPy kosong menggunakan metode NumPy.arange(). Kamu hanya perlu memasukkan nomor yang sama dua kali. Dalam hal ini, kamu ingin memulai dengan tabel kosong dan memperluasnya dari waktu ke waktu.

Alasan mengembalikan array kosong adalah karena metode np.arange() dirancang untuk mengecualikan titik akhir atas (yang, dalam hal ini, sebenarnya sama dengan titik akhir bawah).

### **Cara Membuat Tuple Menggunakan NumPy’s np.arange()**

Sementara metode np.arange() NumPy terutama digunakan untuk membuat array NumPy, juga dimungkinkan untuk menggunakan np.arange() untuk membuat struktur data lainnya.

|  |
| --- |
| np.arange(10,0, -1, dtype=’int64′)  #Returns array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1]) |

Jika kita membungkus larik NumPy ini dalam fungsi tupel bawaan Python, kita dapat dengan mudah mengubah larik ini menjadi tupel!

|  |
| --- |
| np.arange(0,10)#Returns array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) |

Kamu dapat membungkus struktur data baru ini dalam sebuah fungsi tipe untuk memastikan bahwa itu benar-benar sebuah tuple:

|  |
| --- |
| type(tuple(np.arange(0,10)))  #Returns ‘tuple’ |

### **Cara Membuat Set Menggunakan NumPy’s np.arange()**

Pada bagian ini, kita akan melihat cara membuat set Python menggunakan metode array() NumPy. Strategi untuk membuat set menggunakan metode yang sama sangat mirip dengan membuat tupel di NumPy. Pertama, buat array NumPy menggunakan np.arange() lalu buat set Python dengan sintaks yang sama.

|  |
| --- |
| np.arange(1, 10)  #Returns array([1, 2, 3, 4, 5, 6, 7, 8, 9]) |

Kemudian, seperti yang kita lakukan saat belajar tentang tupel, kita akan membungkus seluruh larik/array NumPy dalam fungsi set() :

|  |
| --- |
| set(np.arange(1, 10))  #Returns {1, 2, 3, 4, 5, 6, 7, 8, 9} |

Seperti sebelumnya, kita bisa membungkus struktur data baru ini dalam sebuah fungsi tipe untuk memastikan bahwa itu benar-benar sebuah himpunan:

|  |
| --- |
| set(np.arange(1, 10))  #Returns {1, 2, 3, 4, 5, 6, 7, 8, 9} |

### **Cara Menggunakan Metode np.arange() Numpy Untuk Menulis Loops**

Dalam perulangan for, kamu dapat menggunakan metode np.arange() untuk membuat larik bilangan bulat untuk diulang. Ini adalah salah satu kasus penggunaan yang lebih umum untuk metode array NumPy. Pada bagian ini, kita akan mendemonstrasikan bagaimana menggunakan metode ini untuk menulis loop menggunakan dua contoh.

|  |
| --- |
| number = 1  **for** x **in** np.arange(10):      number \*= x  print(number)  #Returns 0, since 0 is an element of np.arange(10) and multiplies number variable to 0 on the first loop |

Pada kode di atas, kita mengulang objek array array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) dan mengalikan setiap elemen dengan variabel angka.

Mari pertimbangkan contoh lain:

|  |
| --- |
| **for** x **in** np.arange(0, 100, 5):      print(x)  #Returns every integer that is a multiple of 5 between 0 and 100 (excluding 100) |

### **Perbedaan Antara Fungsi Range Bawaan Python dan Metode np.arange() NumPy**

Fungsi range() adalah **fungsi bawaan Python**, sedangkan metode numpy.arange() adalah bagian dari**library komputasi numerik NumPy**. Perbedaan antara keduanya adalah bahwa yang pertama adalah sebuah**fungsi**, dan yang terakhir hadir dengan**NumPy**.

|  |
| --- |
| range(0, 10)  #Returns range(0,10)  type(range(0, 10))  #Returns ‘range’) |

range() memiliki keterbatasan. Ini karena range() **hanya dapat bekerja dengan bilangan bulat, dan tidak menerima tipe data lainnya.**

Contoh:

|  |
| --- |
| np.arange(1,1)  #Returns array([], dtype=int64)  np.arange(1.5, 1.5)  #Returns array([], dtype=float64) |

# Quiz

## Soal 1

Apa yang akan dicetak oleh kode berikut?

|  |
| --- |
| A = np.array([       [‘a’, ‘b’, ‘c’],       [‘d’, ‘e’, ‘f’],       [‘g’, ‘h’, ‘i’] ]) print(A[:, :2]) |