The Basics of NumPy Arrays

Array နှင့် ပတ်သက်သည့် အောက်ပါအချက်များကို ရှင်းပြထားသည်။

- (၁) Attributes of arrays*: Determining the size, shape, memory consumption, and data types of arrays
- (၂) Indexing of arrays*: Getting and setting the value of individual array elements
- (၃) Slicing of arrays*: Getting and setting smaller subarrays within a larger array
- (၄) Reshaping of arrays*: Changing the shape of a given array
- (၅) Joining and splitting of arrays*: Combining multiple arrays into one, and splitting one array into many

(၁) NumPy Array Attributes

First let's discuss some useful array attributes. We'll start by defining three random arrays, a one-dimensional, two-dimensional, and three-dimensional array. We'll use NumPy's random number generator, which we will *seed* with a set value in order to ensure that the same random arrays are generated each time this code is run:

In [1]:
```
import numpy as np
np.random.seed(0)  # seed for reproducibility
```

element (၆)ခု ပါသည့် One-dimensional array တည်ဆောက်ရန်

In [2]:
```
x1 = np.random.randint(10, size=6)  # One-dimensional array
x1
```

Out[2]:
```
array([5, 0, 3, 3, 7, 9])
```

In [3]:
```
x2 = np.random.randint(10, size=(3, 4))  # Two-dimensional array
x2
```

Out[3]:
```
array([[3, 5, 2, 4],
       [7, 6, 8, 8],
       [1, 6, 7, 7]])
```

```
In [4]: x3 = np.random.randint(10, size=(3, 4, 5))  # Three-dimensional array
        x3
```

```
Out[4]: array([[[8, 1, 5, 9, 8],
                [9, 4, 3, 0, 3],
                [5, 0, 2, 3, 8],
                [1, 3, 3, 3, 7]],

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

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

array များတွင် dimensions , size of each dimension, total size of the array, data type စသည့် array နှင့် သက်ဆိုင်သည့် attribute များ ရှိကြသည်။

array တစ်ခု၏ ကို number of dimensions သိရန် .ndim ကို အသုံးပြုသည်။ `ndim` (the number of dimensions)

```
In [5]: print("x3 ndim: ", x3.ndim)
```

```
x3 ndim:  3
```

array တစ်ခု၏ size of each dimension ကို သိရန် .shape ကို အသုံးပြုသည်။ `shape` (the size of each dimension)

```
In [6]: print("x3 shape:", x3.shape)
```

```
x3 shape: (3, 4, 5)
```

array တစ်ခု၏ ကို total size of the array သိရန် .size ကို အသုံးပြုသည်။ `size` (the total size of the array)

```
In [7]: print("x3 size: ", x3.size)
```

```
x3 size:  60
```

array တစ်ခု၏ ကို data type သိရန် .dtype ကို အသုံးပြုသည်။ `dtype` , the data type of the array

```
In [8]: print("dtype:", x3.dtype)
```

```
dtype: int32
```

array တစ်ခု၏ ကို itemsize သိရန် .itemsize ကို အသုံးပြုသည်။ `itemsize` , which lists the size (in bytes) of each array element

array တစ်ခု၏ ကို nbytes သိရန် .nbytes ကို အသုံးပြုသည်။ `nbytes` , which lists the total size (in bytes) of the array

```
In [9]: print("itemsize:", x3.itemsize, "bytes")
        print("nbytes:", x3.nbytes, "bytes")
```

```
itemsize: 4 bytes
nbytes: 240 bytes
```

ယေဘုယျအားဖြင့် `nbytes` သည် `itemsize` နှင့် `size` တို့ မြှောက်လဒ် ဖြစ်သည်။ (In general, we expect that `nbytes` is equal to `itemsize` times `size`.)

(၂) Array Indexing: Accessing Single Elements

Python's standard list indexing နှင့် NumPy indexing တို့သည် ခပ်ဆင်ဆင် တူညီကြသည်။ one-dimensional array တွင် i^{th} value (counting from zero) ဖြင့် indexing လုပ်သည်။ သုညမှ စတင်ရေတွက်သည်။ ထောင့်ကွင်း [ ] ထဲတွင် ဖော်ပြသည်။

In a one-dimensional array, the i^{th} value (counting from zero) can be accessed by specifying the desired index in square brackets, just as with Python lists:

```
In [10]: x1
```

```
Out[10]: array([5, 0, 3, 3, 7, 9])
```

```
In [11]: x1[0] # ပထမဆုံး element
```

```
Out[11]: 5
```

```
In [12]: x1[4] # ပဉ္စမကိန်း
```

```
Out[12]: 7
```

နောက်ဆုံးမှလည်း ပြန်လည် ညွှန်ပြနိုင်သည်။ နောက်ဆုံးမှ ညွှန်ပြသည့်အခါ အနှုတ်လက္ခဏာဖြင့် ပြသည်။ (negative indices) နောက်ဆုံးကိန်းသည် [-1] ဖြစ်သည်။ To index from the end of the array, you can use negative indices:

```
In [13]: x1[-1]
```

```
Out[13]: 9
```

' အနှုတ်လက္ခဏာ (-ve) သည် နောက်ဆုံးမှ ပြန်ညွှန်ပြသည်ကို ဆိုလိုသည်။ 2 သည် ဒုတိယကိန်းဖြစ်သည်။ -2 သည် ဒုတိယ နောက်ဆုံးကိန်းဖြစ်သည်။

```
In [14]: x1[-2]
```

```
Out[14]: 7
```

multi-dimensional array များတွင် item များကို ကော်မာဖြင့်ခွဲရေးသ်။ comma-separated tuple of indices ဖြစ်သည်။ (In a multi-dimensional array, items can be accessed using a comma-separated tuple of indices)

```
In [15]: x2
```

```
Out[15]: array([[3, 5, 2, 4],
                [7, 6, 8, 8],
                [1, 6, 7, 7]])
```

row = 0 နှင့် column = 0 နေရာမှ ကိန်း(item)ကို ရည်ညွှန်းသည်။

In [16]:
```
x2[0, 0]
```

Out[16]:
```
3
```

In [17]:
```
x2[2, 0]
```

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

In [18]:
```
x2[2, -1]
```

Out[18]:
```
7
```

ကိန်းများ(items) များ၏ နေရာကို ညွှန်ပြ၍ ထိုကိန်းများ(items) များ၏ တန်ဖိုးကို ပြောင်းလဲနိုင်သည်။ (Values can also be modified using any of the above index notation:)

In [19]:
```
x2[0, 0] = 12
x2
```

Out[19]:
```
array([[12,  5,  2,  4],
       [ 7,  6,  8,  8],
       [ 1,  6,  7,  7]])
```

Keep in mind that, unlike Python lists, NumPy arrays have a fixed type. This means, for example, that if you attempt to insert a floating-point value to an integer array, the value will be silently truncated. Don't be caught unaware by this behavior!

In [20]:
```
x1[0] = 3.14159  # this will be truncated!
x1
```

Out[20]:
```
array([3, 0, 3, 3, 7, 9])
```

(၃) Array Slicing: Accessing Subarrays

တစ်ခုထက်ပိုများသည့် ကိန်းများကို ရွေးချယ်လိုသည့်အခါ

```
x[start:stop:step]
```

ကဲ့သို့ ရွေးချယ်နိုင်သည်။

If any of these are unspecified, they default to the values `start=0`, `stop=` *`size of dimension`*, `step=1`. We'll take a look at accessing sub-arrays in one dimension and in multiple dimensions.

One-dimensional subarrays

In [21]:
```
x = np.arange(10)
x
```

Out[21]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[:5]= အစမှ (၅)ခုမြောက်ကိန်းအထိ

```
In [22]: x[:5]  # first five elements
Out[22]: array([0, 1, 2, 3, 4])
```

[5:] = (၅)ခုမြောက်ကိန်းမှ နောက် ရှိသမျှကိန်းများအားလုံး ((၅)ခုမြောက်ကိန်းမပါ)

```
In [23]: x[5:]  # elements after index 5
Out[23]: array([5, 6, 7, 8, 9])
```

[4:7] = (၄)ခုမြောက်ကိန်းမှ နေ (၇) မြောက်ကိန်းအထိ((၄)ခုမြောက်ကိန်းမပါ)

```
In [24]: x[4:7]  # middle sub-array
Out[24]: array([4, 5, 6])
```

အစ မှ အဆုံးအထိ : နှစ်ခုမြောက်ကိန်းများသာ

```
In [25]: x[::2]  # every other element
Out[25]: array([0, 2, 4, 6, 8])
```

x[1::2] (၁)ခုမြောက်ကိန်းမှ နောက် ရှိသမျှကိန်းများအားလုံး ((၁)ခုမြောက်ကိန်းမပါ) : နှစ်ခုမြောက်ကိန်းများသာ

```
In [26]: x[1::2]  # every other element, starting at index 1
Out[26]: array([1, 3, 5, 7, 9])
```

`step` value `step` value ကို အနှုတ်ဖြင့် ပြသည့်အခါ ရောထွေးသွားနိုင်သည်။ `step` value ကို အနှုတ်ဖြင့် ပြသည့်အခါ `start` နှင့် `stop` ရှေ့နေရာမှ နောက်နေရာ ၊ နောက်နေရာမှ ရှေ့နေရာသို့ ပြောင်းလဲလိုက်ပါ။

A potentially confusing case is when the `step` value is negative. In this case, the defaults for `start` and `stop` are swapped. This becomes a convenient way to reverse an array:

```
In [27]: x[::-1]  # all elements, reversed
Out[27]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
In [28]: x[5::-2]  # reversed every other from index 5
Out[28]: array([5, 3, 1])
```

Multi-dimensional subarrays

Multi-dimensional array ကို slicing လုပ်သည့်အခါ လေးထောင့်ကွင်းထဲတွင် ကော်မာ ခံ၍ ရေးသည်။ [rows , columns]

Multi-dimensional slices work in the same way, with multiple slices separated by commas. For example:

```
In [29]: x2
```

```
Out[29]: array([[12,  5,  2,  4],
                [ 7,  6,  8,  8],
                [ 1,  6,  7,  7]])
```

[:2, = အစ row မှ ဒုတိယ row အထိ ဖြစ်သည်။ :3] သည် အစ column မှ နောက်ဆုံး row အထိ ဖြစ်သည်။

```
In [30]: x2[:2, :3]  # two rows, three columns
```

```
Out[30]: array([[12,  5,  2],
                [ 7,  6,  8]])
```

```
In [31]: x2[:3, ::2]  # all rows, every other column
```

```
Out[31]: array([[12,  2],
                [ 7,  8],
                [ 1,  7]])
```

subarray dimensions များကိုလည်း အနှုတ်လက္ခဏာ ထည့်၍ ပြောင်းပြန်ယူရန် ရေးနိုင်သည်။ Finally, subarray dimensions can even be reversed together:

```
In [32]: x2[::-1, ::-1]
```

```
Out[32]: array([[ 7,  7,  6,  1],
                [ 8,  8,  6,  7],
                [ 4,  2,  5, 12]])
```

Accessing array rows and columns

One commonly needed routine is accessing of single rows or columns of an array. This can be done by combining indexing and slicing, using an empty slice marked by a single colon (:):

x2 array ပထမ ကော်လံ တစ်ခုတည်းကိုသာ ရွေးရန်

```
In [33]: print(x2[:, 0])  # first column of x2
```

```
[12  7  1]
```

ပထမ row တစ်ခုတည်းကိုသာ ရွေးရန်

```
In [34]: print(x2[0, :])  # first row of x2
```

```
[12  5  2  4]
```

In the case of row access, the empty slice can be omitted for a more compact syntax:

```
In [35]: print(x2[0])  # equivalent to x2[0, :]
```

```
[12  5  2  4]
```

Subarrays as no-copy views

array ကို slicing လုပ်သည့်အခါ ထိုarray ထဲမှာ ကိန်းများကို copy ကူးယူခြင်း မဟုတ်ဘဲ။ ကြည့်ရှုရန်အတွက် သာဖြစ်သည်။ ဤ အချက်သည် NumPy array slicing နှင့် Python list slicing တို့၏ ကွာခြားချက်ဖြစ်သည်။ Python list slicing လုပ်သည်အခါ ကိန်းများ(ဒေတာ)များကို copy ကူးယူခြင်း ဖြစ်သည်။

One important–and extremely useful–thing to know about array slices is that they return *views* rather than *copies* of the array data. This is one area in which NumPy array slicing differs from Python list slicing: in lists, slices will be copies. Consider our two-dimensional array from before:

In [36]:
```
print(x2)
```

```
[[12  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

မူလ X2 array မှ 2×2 subarray တစ်ခုကို ထုတ်ယူပြီး ထို subarray ထဲမှာ ကိန်းတစ်ခုကို ပြောင်လိုက်လျှင် မူလ X2 array မှ ကိန်းတန်ဖိုးများလိုက်ပြောင်းပုံကို အောက်တွင် ဖော်ပြထားသည်။

အဘယ်ကြောင့်ဆိုသော် 2×2 subarray သည် မူလ X2 မှ copy ကူးယူထားသည့် subarray မဟုတ်သောကြောင့် ဖြစ်သည်။

In [37]:
```
x2_sub = x2[:2, :2]
print(x2_sub)
```

```
[[12  5]
 [ 7  6]]
```

Now if we modify this subarray, we'll see that the original array is changed! Observe:

In [38]:
```
x2_sub[0, 0] = 99
print(x2_sub)
```

```
[[99  5]
 [ 7  6]]
```

In [39]:
```
print(x2)
```

```
[[99  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

This default behavior is actually quite useful: it means that when we work with large datasets, we can access and process pieces of these datasets without the need to copy the underlying data buffer.

Creating copies of arrays

array ကို slicing လုပ်ခြင်းသည် ရွေးချယ်ကြည့်ရှုခြင်းသာဖြစ်သည်။ ကော်ပီကူးလိုပါက `copy()` method ကို အသုံးပြုရသည်။

Despite the nice features of array views, it is sometimes useful to instead explicitly copy the data within an array or a subarray. This can be most easily done with the `copy()` method:

အောက်တွင် 2×2 subarray ကို copy ကူးယူ၍ တန်ဖိုးများ ပြောင်းသည့်အခါ မူလ x2 array မှ ကိန်းတန်ဖိုးများ မပြောင်းလဲပုံကို ဖော်ပြထာသည်။

In [40]:
```
x2
```

Out[40]:
```
array([[99,  5,  2,  4],
       [ 7,  6,  8,  8],
       [ 1,  6,  7,  7]])
```

In [41]:
```
x2_sub_copy = x2[:2, :2].copy()
print(x2_sub_copy)
```

```
[[99  5]
 [ 7  6]]
```

If we now modify this subarray, the original array is not touched:

In [42]:
```
x2_sub_copy[0, 0] = 42
print(x2_sub_copy)
```

```
[[42  5]
 [ 7  6]]
```

In [43]:
```
print(x2)
```

```
[[99  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```

(၄) Reshaping of Arrays (Dimensions ပြောင်းခြင်း)

array များကို reshaping လုပ်ခြင်း အလွန်အသုံးဝင်သည့် နည်းတစ်ခု ဖြစ်သည်။ `reshape` method ဖြင့် reshaping လုပ်သည်။ ၁ မှ ၉ အထိ ကိန်း(၉)လုံးကို 3×3 array ဖြင့် ရေးလိုလျှင်

Another useful type of operation is reshaping of arrays. The most flexible way of doing this is with the `reshape` method. For example, if you want to put the numbers 1 through 9 in a 3×3 grid, you can do the following:

In [44]:
```
grid = np.arange(1, 10).reshape((3, 3))
print(grid)
```

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

သတိပြုရန်အချက်မှ reshape မလုပ်ခင် array အရွယ်အစား(size)သည် reshape လုပ်ပြီးသည့် array အရွယ်အစား(size) နှင့် တူညီရမည်။ တစ်နည်းအားဖြင့် dimension ကိုက်ညီသည့် array အရွယ်အစား(size) ဖြစ်အောင် reshape လုပ်ရမည်။

ဖြစ်နိုင်လျှင် no-copy view of the initial array ကို အလိုရှိသည့်အခါ `reshape` method ကို အသုံး ပြုသင့်သည်။

အသုံးများသည့် reshaping ပုံစံမှာ one-dimensional array မှ two-dimensional row matrix သို့မဟုတ် two-dimensional column matrix ဖြစ်အောင် ပြောင်းခြင်း ဖြစ်သည်။ within a slice operation တွင် `newaxis` keyword ထည့်၍လည်း ပြုလုပ်နိုင်သည်။

Note that for this to work, the size of the initial array must match the size of the reshaped array. Where possible, the `reshape` method will use a no-copy view of the initial array, but with non-contiguous memory buffers this is not always the case.

Another common reshaping pattern is the conversion of a one-dimensional array into a two-dimensional row or column matrix. This can be done with the `reshape` method, or more easily done by making use of the `newaxis` keyword within a slice operation:

In [45]:
```
x = np.array([1, 2, 3])

# row vector via reshape
x.reshape((1, 3))
```

Out[45]:
```
array([[1, 2, 3]])
```

In [46]:
```
# row vector via newaxis
x[np.newaxis, :]
```

Out[46]:
```
array([[1, 2, 3]])
```

In [47]:
```
# column vector via reshape
x.reshape((3, 1))
```

Out[47]:
```
array([[1],
       [2],
       [3]])
```

In [48]:
```
# column vector via newaxis
x[:, np.newaxis]
```

Out[48]:
```
array([[1],
       [2],
       [3]])
```

We will see this type of transformation often throughout the remainder of the book.

(၅) Array များ ပေါင်းစပ်ခြင်း(Concatenation) နှင့် ပိုင်းခြား ခွဲခြားခြင်း(Splitting)

All of the preceding routines worked on single arrays. It's also possible to combine multiple arrays into one, and to conversely split a single array into multiple arrays. We'll take a look at those operations here.

Array များ ပေါင်းစပ်ခြင်း(Concatenation)

NumPy တွင် array ကို ပေါင်းစပ်(Concatenation or joining)လိုလျှင် `np.concatenate` , `np.vstack` , and `np.hstack` စသည့် routine များကို အသုံးပြုနိုင်သည်။ ဂConcatenation, or joining of two arrays in NumPy, is primarily accomplished using the routines `np.concatenate` , `np.vstack` , and `np.hstack` .)

`np.concatenate` ကို သုံးသည့်အခါ first argument အဖြစ် tuple သို့မဟုတ် list of arrays ကို ထည့်ပေးရသည်။ (`np.concatenate` takes a tuple or list of arrays as its first argument, as we can see here:)

```
In [49]: x = np.array([1, 2, 3])
         y = np.array([3, 2, 1])
         np.concatenate([x, y])
```

```
Out[49]: array([1, 2, 3, 3, 2, 1])
```

array နှစ်ခုကို တပြိုင်နက်ပေါင်းစည်းနိုင်သည်။ (You can also concatenate more than two arrays at once:)

```
In [50]: z = [99, 99, 99]
         print(np.concatenate([x, y, z]))
```

```
[ 1  2  3  3  2  1 99 99 99]
```

two-dimensional arrays များကိုလည်း ပေါင်းစည်းနိုင်သည်။

```
In [51]: grid = np.array([[1, 2, 3],
                          [4, 5, 6]])
```

```
In [52]: # concatenate along the first axis
         np.concatenate([grid, grid])
```

```
Out[52]: array([[1, 2, 3],
                [4, 5, 6],
                [1, 2, 3],
                [4, 5, 6]])
```

```
In [53]: # concatenate along the second axis (zero-indexed)
         np.concatenate([grid, grid], axis=1)
```

```
Out[53]: array([[1, 2, 3, 1, 2, 3],
                [4, 5, 6, 4, 5, 6]])
```

ပေါင်းစည်းမည့် array များ၏ dimensions မတူညီသည့်အခါ ဒေါင်လိုက်ပေါင်းရန် `np.vstack` (vertical stack) နှင့် အလျားလိုက်ပေါင်းရန် `np.hstack` (horizontal stack) function တို့ကိုအသုံးပြုရမည်။

For working with arrays of mixed dimensions, it can be clearer to use the `np.vstack` (vertical stack) and `np.hstack` (horizontal stack) functions:

```
In [54]: x = np.array([1, 2, 3])
         grid = np.array([[9, 8, 7],
                          [6, 5, 4]])
```

ဒေါင်လိုက်ပေါင်းရန် np.vstack (vertical stack) function အသုံးပြုသည်။

In [55]:
```
# vertically stack the arrays
np.vstack([x, grid])
```

Out[55]:
```
array([[1, 2, 3],
       [9, 8, 7],
       [6, 5, 4]])
```

အလျားလိုက်ပေါင်းရန် np.hstack (horizontal stack) function အသုံးပြုသည်။

In [56]:
```
# horizontally stack the arrays
y = np.array([[99],
              [99]])
np.hstack([grid, y])
```

Out[56]:
```
array([[ 9,  8,  7, 99],
       [ 6,  5,  4, 99]])
```

Similary, `np.dstack` will stack arrays along the third axis.

Array ပိုင်းခြား ခွဲခြားခြင်း(Splitting)

array များကို ပေါင်းစည်း နိုင်သလို `np.split` , `np.hsplit` နှင့် `np.vsplit` စသည့် function များကို သုံး၍ ပိုင်းခြား ခွဲခြားခြင်း(Splitting) ပြုလုပ်နိုင်သည်။

`np.hsplit` သည် အလျားအတိုင်း ပိုင်းခြားခြင်း ဖြစ်သည်။ `np.vsplit` သည် ဒေါင်လိုက် ပိုင်းခြားခြင်းဖြစ်သည်။

The opposite of concatenation is splitting, which is implemented by the functions `np.split` , `np.hsplit` , and `np.vsplit` . For each of these, we can pass a list of indices giving the split points:

In [57]:
```
x = [1, 2, 3, 99, 99, 3, 2, 1]
x1, x2, x3 = np.split(x, [3, 5])
print(x1, x2, x3)
```

```
[1 2 3] [99 99] [3 2 1]
```

Notice that *N* split-points, leads to *N + 1* subarrays. The related functions `np.hsplit` and `np.vsplit` are similar:

In [58]:
```
grid = np.arange(16).reshape((4, 4))
grid
```

Out[58]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

`np.vsplit` သည် ဒေါင်လိုက် ပိုင်းခြားခြင်းဖြစ်သည်။

In [59]:
```
upper, lower = np.vsplit(grid, [2])
print(upper)
print(lower)
```

```
[[0 1 2 3]
 [4 5 6 7]]
[[ 8  9 10 11]
 [12 13 14 15]]
```

`np.hsplit` သည် အလျားအတိုင်း ပိုင်းခြားခြင်း ဖြစ်သည်။

In [60]:
```
left, right = np.hsplit(grid, [2])
print(left)
print(right)
```

```
[[ 0  1]
 [ 4  5]
 [ 8  9]
 [12 13]]
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
```

Similarly, `np.dsplit` will split arrays along the third axis.