NumPy Basics: Arrays and Vectorized Computation

NumPy သည် Numerical Python ၏ အတိုခေါက်ဖြစ်သည်။ high performance scientific computing လုပ်ရန်နှင့်data analysis လုပ်ရန်အတွက် မရှိမဖြစ်လိုအပ်သည် Python package တစ်ခုဖြစ်သည်။ higher-level tool များအားလုံးနီးပါးသည် NumPy ပေါ်တွင် အခြေခံ၍ တည်ဆောက်ထားကြသည်။

- ndarray သည် multidimensional array ဖြစ်သည်။ vectorနည်းဖြင့် ပေါင်း နှုတ် မြောက် စား ကိစ္စ (vectorized arithmetic operation)များကို လုပ်ပေးနိုင်သည်။ ထူးခြားစွာ broadcasting လုပ်နိုင်သည့် စွမ်းရည်ရှိသည်။
- loop များကို သုံးရန် မလိုဘဲ အလွန်လျင်မြန်စွာ ပေါင်း နှုတ် မြောက် စားလုပ်နိုင်သည့် Standard mathematical function များရှိသည်။
- disk အပေါ်သို့ ဒေတာများ ရေးခြင်း၊ disk ပေါ်မှာ ဒေတာများကို ဖတ်ယူနိုင်သည့် Tool များစွာ ရှိသည်။ memory-mapped file နှင့် တွဲ၍ အလုပ်လုပ်နိုင်သည်။
- အက္ခရာသင်္ချာ(Linear algebra) ၊ random number ထုတ်ခြင်း(generation) နှင့် , Fourier transform စသည့် အဆင်မြင့် သင်္ချာများကို တွက်ချက်နိုင်စွမ်းရှိသည်။
- Tools for integrating code written in C, C++, and Fortran စသည် တခြားသော programming language များနှင့် တွဲ၍ သုံးနိုင်သည့်(integrating code)Tool များရှိသည်။ NumPy က low-level language ဖြင့် ရေးထားသည့် ပြင်ပ(external libraries)မှ ဒေတာများကို Python သို့ ထည့်ပေးနိုင်သည်။ Python NumPy မှ ဒေတာများကို တခြားသော low-level language ဖြင့် ရေးထားသည့် ပြင်ပ(external libraries)ဆီသို့ ပြန်ပို့ပေးနိုင်သည်။ Python သည် C ၊ C++ နှင့် Fortran စသည် တခြားသော programming language များနှင့် တွဲ၍ သုံးနိုင်သော interface(dynamic and easy-to-use)များရှိသောကြောင့် လူကြိုက်များခြင်းဖြစ်သည်။
- NumPy တွင် high-level data analytical function များ မပါသော်လည်း NumPy arrays နှင့် array-oriented computing ကို သေသေချာနားလည်းခြင်းကြောင့် high-level data analytical အလုပ်များလုပ်နိုင်သည့် pandas ကို ကျွမ်းကျင်စွာအသုံးပြုနိုင်လိမ့်မည်။ Python ကို စတင်လေ့လာသူများ၊ Data analysis နှင့် machine learning ကို စတင်လေ့လာသူများသည် NumPy ကို အချိန်ပေးလေ့လာရန် တိုက်တွန်းလိုသည်။

Data analysis application လုပ်ငန်းများအတွက် NumPy ၏ လုပ်ဆောင်ချက်များမှာ

- Fast vectorized array operations for data munging and cleaning, subsetting and filtering, transformation, and any other kinds of computations
- Common array algorithms like sorting, unique, and set operations
- Efficient descriptive statistics and aggregating/summarizing data
- Data alignment and relational data manipulations for merging and joining together heterogeneous data sets
- Expressing conditional logic as array expressions instead of loops with if-elif¬else branches
- Group-wise data manipulations (aggregation, transformation, function application). Much more on this in Chapter 5

In [1]:
```
import numpy as np
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
np.set_printoptions(precision=4, suppress=True)
```

```
In [2]: import numpy as np
        my_arr = np.arange(1000000)
        my_list = list(range(1000000))
```

NumPy array သည် python list ထက် ပို၍ မြန်ဆန်စွာ အလုပ်လုပ်ပေးနိုင်သည်။ အောက်တွင် အရွယ်အစားတူညီသည့် NumPy array နှင့် python list တို့ကို ၂ နှင့်မြှောက် ရန်အတွက် ကြာချိန်ကို နှိုင်းယှဉ်ပြထားသည်။

```
In [3]: %time for _ in range(10): my_arr2 = my_arr * 2
        %time for _ in range(10): my_list2 = [x * 2 for x in my_list]
```

```
Wall time: 32.5 ms
Wall time: 1.23 s
```

The NumPy ndarray: A Multidimensional Array Object

NumPy ၏ ထူးခြားသည့် လုပ်ဆောင်နိုင်စွမ်းတစ်ခုမှာ ndarray ဟူခေါ်သည့် N-dimensional array object တစ်ခုဖြစ်သည်။ Python တွင် အလွန်ကြီးမားသည့် ဒေတာ(large data sets)များကို လွယ်ကူ လျှင်မြန်စွာ အဆင်ပြေပြေ ဆောင်ရွက်ပေးနိုင်ခြင်း ဖြစ်သည်။ အလွန်ကြီးမားသည့် ဒေတာများကို Arrays အဖြစ်သိမ်းဆည်းပြီး ကိန်းတစ်လုံးကဲ့သို့ ပေါင်း နှုတ် မြှောက် စား လုပ်ပေးနိုင်ခြင်းဖြစ်သည်။

အလွန်ကြီးမားသည့် ဒေတာများကို Arrays အဖြစ်သိမ်းဆည်းပြီး ကိန်းတစ်လုံးကဲ့သို့ ပေါင်းနှုတ်မြှောက်စားကိစ္စများ(mathematical operations)ကို လုပ်ပေးနိုင်ခြင်းဖြစ်သည်။ ndarray သည် အမျိုးတူသည့်ဒေတာများ(homogeneous data)ကို သိမ်းဆည်းထားနိုင်သည့် generic multidimensional container တစ်ခုဖြစ်သည်။ homogeneous data ဆိုသည်မှာ အမျိုးတူသည့်ဒေတာများ(all of the elements must be the same type)ကို ဆိုလိုသည်။

array များအားလုံးတွင် `shape` နှင့် `dtype` နှစ်မျိုး ရှိကြသည်။ `shape` သည် row အရေအတွက်နှင့် ကော်လံ အရေအတွက်ဖြစ်သည်။ row အရေအတွက်နှင့် ကော်လံ အရေအတွက်ကို dimension ဟူလည်းခေါ်သည်။ ထို့ကြောင့် shape ဆိုသည်မှာ row အရေအတွက်နှင့် ကော်လံ အရေအတွက်ကို ဖော်ပြထားသည့် tuple တစ်ခုဖြစ်သည်။ (a tuple indicating the size of each dimension)။ `dtype` ဆိုသည်မှာ ထဲရှိ ဒေတာများ၏ အမျိုးအစားကို ဖော်ပြသည့် object တစ်ခုဖြစ်သည်။ (an object describing the *datatype* of the array)

အောက်တွင် ဥပမာဖြင့် ရှင်းပြရန် import numpy ကို import လုပ်သည်။ import numpy ကို import လုပ်သည့်အခါတိုင်း np ဟု အတိုခေါက် အသုံးပြုမည်။ np.random.randn(2, 3) ဖြင့် 2x3 array ဖြစ်အောင် random နံပါတ်များဖြင့် တစ်ခုတည်ဆောက်သည်။

```
In [4]: import numpy as np
        # Generate some random data
        data = np.random.randn(2, 3)
        data
```

```
Out[4]: array([[-0.2047,  0.4789, -0.5194],
               [-0.5557,  1.9658,  1.3934]])
```

data ဆိုသည့် array ၏ shape ကို `.shape` ကြည့်ရန် (2x3 array ဖြစ်သည်။)

```
In [5]: data.shape
```

```
Out[5]: (2, 3)
```

data ဆိုသည့် array ၏ ဒေတာအမျိုးအစားများကို `.dtype`` ကြည့်ရန် (float64 ဖြစ်သည်။)

```
In [6]: data.dtype
```

```
Out[6]: dtype('float64')
```

“array”၊ “NumPy array” သို့မဟုတ် “ndarray” စသည့် စကားလုံးများသည် ndarray object(N dimension array) ကို ဆိုလိုသည်။

data ဆိုသည့် array ကို ၁၀ ဖြင့် မြှောက်သည်။

```
In [7]: data * 10
```

```
Out[7]: array([[-2.0471,  4.7894, -5.1944],
               [-5.5573, 19.6578, 13.9341]])
```

array အချင်းချင်း ပေါင်းသည်။

```
In [8]: data + data
```

```
Out[8]: array([[-0.4094,  0.9579, -1.0389],
               [-1.1115,  3.9316,  2.7868]])
```

Creating ndarrays

array တစ်ခုပြုလုပ်ရန်အတွက် အလွယ်ကူဆုံးနည်းမှာ array function ကို အသုံးပြုခြင်း ဖြစ်သည်။

```
In [9]: data1 = [6, 7.5, 8, 0, 1]
        type(data1)
```

```
Out[9]: list
```

data1 သည် list တစ်ခုသာဖြစ်သည်။ `np.array(data1)` ဖြင့် data1 ကို array တစ်ခု ဖြစ်အောင် ပြုလုပ်သည်။

```
In [10]: arr1 = np.array(data1)
         arr1
```

```
Out[10]: array([6. , 7.5, 8. , 0. , 1. ])
```

အတိုင်းအတာတူညီသည့် list များပါဝင်သည့် list (a list of equal-length lists)ကို multidimensional array အဖြစ် တည်ဆောက်နိုင်သည်။ Nested sequences ဖြစ်သည်။

```
In [11]: data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
         arr2 = np.array(data2)
         arr2
```

```
Out[11]: array([[1, 2, 3, 4],
                [5, 6, 7, 8]])
```

`.ndim` ဖြင့် array ၏ dimensions အရေအတွက်ကို စစ်သည်။ (Number of array dimensions.)

```
In [12]: arr2.ndim
```

```
Out[12]: 2
```

```
In [13]: arr2.shape
```

```
Out[13]: (2, 4)
```

Array ထဲတွင် ပါရှိသည့် element များ၏ ဒေတာမျိုးအစားများကို dtype object ထဲတွင် သိမ်းဆည်းထားသည်။ (The data type is stored in a special dtype object.)

```
In [14]: arr1.dtype
```

```
Out[14]: dtype('float64')
```

```
In [15]: arr2.dtype
```

```
Out[15]: dtype('int32')
```

np.array function ထဲတွင် တခြားသော နည်းများဖြင့် array အသစ်များကို ပြုလုပ်နိုင်သည့်နည်းများလည်းပါဝင်သည်။ ဥပမာ array ထဲရှိ element များအားလုံး 0's သာ ဖြစ်သည့် array ကို `np.zeros()` ဖြင့် ပြုလုပ်နိုင်သည်။ array ထဲရှိ element များအားလုံး 1's သာဖြစ်သည့် array များကို `np.ones()` ဖြင့် ပြုလုပ်နိုင်သည်။

array အသစ်များ ပြုလုပ်သည့်အချိန်တွင် မိမိအလိုရှိသည့် အရွယ်အစား (length or shape) ကို row အရေအတွက်နှင့် ကော်လံ အရေအတွက် ထည့်သွင်း၍ ပြုလုပ်နိုင်သည်။

```
In [16]: np.zeros(10)
         np.zeros((3, 6))
```

```
Out[16]: array([[0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0., 0.]])
```

```
In [17]: np.ones((3, 2))
```

```
Out[17]: array([[1., 1.],
                [1., 1.],
                [1., 1.]])
```

np.empty() ဖြင့် empty array တစ်ခု ပြုလုပ်နိုင်သည်။ array ထဲရှိ element များ၏ တန်ဖိုးကို initializing မလုပ်လိုသည့်အခါတွင် empty array ကို ပြုလုပ်ကြသည်။ np.empty ဖြင့် empty array တစ်ခုသည် အခါ array ထဲရှိ element များ၏ တန်ဖိုး(value)များအားလုံးသည် သုည(zero)ဖြစ်လိမ့်မည်ဟု အာမ, မခံနိုင်ပါ။ သေချာပေါက် မယူဆသင့်ပါ။

```
In [18]: np.empty((2, 3, 2))
```

```
Out[18]: array([[[0., 0.],
                 [0., 0.],
                 [0., 0.]],

                [[0., 0.],
                 [0., 0.],
                 [0., 0.]]])
```

`arange` is an array-valued version of the built-in Python range function:

အောက်တွင် array တည်ဆောက်ရန်အတွက် function များကို ဖော်ပြထားသည်။

Array creation functions

Function	Description
`array`	Convert input data (list, tuple, array, or other sequence type) to an ndarray either by inferring a dtype or explicitly specifying a dtype. Copies the input data by default.
`asarray`	Convert input to ndarray, but do not copy if the input is already an ndarray
`arange`	Like the built-in `range` but returns an ndarray instead of a list.
`ones, ones_like`	Produce an array of all 1's with the given shape and dtype. `ones_like` takes another array and produces a ones array of the same shape and dtype.
`zeros, zeros_like`	Like ones and `ones_like` but producing arrays of 0's instead
`empty, empty_like`	Create new arrays by allocating new memory, but do not populate with any values like ones and `zeros`
`eye, identity`	Create a square N x N identity matrix (1's on the diagonal and 0's elsewhere)

Data Types for ndarrays

data type သို့မဟုတ် `dtype` သည် ndarray ၏ ဒေတာများနှင့် သက်ဆိုင်သည့် အချက်အလက်များကို သိမ်းဆည်းထားသည့် special object တစ်ခုဖြစ်သည်။

Dtypes ၏ ထူးခြားသည့် စွမ်းရည်ကြောင့် NumPy သည် စွမ်းအားပိုကောင်း(powerful and flexible)ခြင်းဖြစ်သည်။ တိုက်ရိုက်mapping လုပ်နိုင်ခြင်းကြောင့် (map directly) ဒေတာများ အလွယ်တကူ ဖတ်ခြင်း၊ ရေးခြင်းပြုလုပ်နိုင်သည်။ (easy to read and write binary streams of data to disk)။ C သို့မဟုတ် Fortran စသည့် low-level language များနှင့်လည်း ချိတ်ဆက်ပြီး အလွယ်တကူ အသုံးပြုနိုင်ခြင်း ဖြစ်သည်။ NumPy တွင် အသုံးပြုနိုင်သည် ဒေတာအမျိုးအစား(NumPy's supported data types)အားလုံးကို အောက်တွင် ဖော်ပြထားသည်။

NumPy data types

Type	Type Code	Description
`int8, uint8`	`i1, u1`	Signed and unsigned 8-bit (1 byte) integer types
`int16, uint16`	`i2, u2`	Signed and unsigned 16-bit integer types
`int32, uint32`	`i4, u4`	Signed and unsigned 32-bit integer types
`int64, uint64`	`i8, u8`	Signed and unsigned 32-bit integer types
`float16`	`f2`	Half-precision floating point
`float32`	`f4 or f`	Standard single-precision floating point. Compatible with C float
`float64, float128`	`f8 or d`	Standard double-precision floating point. Compatible with C double and Python `float` object
`float128`	`f16 or g`	Extended-precision floating point
`complex64, complex128, complex256`	`c8, c16, c32`	Complex numbers represented by two 32, 64, or 128 floats, respectively
`bool`	`?`	Boolean type storing `True` and `False` values
`object`	`0`	Python object type
`string_`	`S`	Fixed-length string type (1 byte per character). For example, to create a string dtype with length 10, use `'S10'`.

It's often only necessary to care about the general kind of data you're dealing with, whether floating point, complex, integer, boolean, string, or general Python object.

```
In [19]: np.arange(15)
```

```
Out[19]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 1
         4])
```

```
In [20]: arr1 = np.array([1, 2, 3], dtype=np.float64)
         arr2 = np.array([1, 2, 3], dtype=np.int32)
         arr1.dtype
```

```
Out[20]: dtype('float64')
```

```
In [21]: arr2.dtype
```

```
Out[21]: dtype('int32')
```

`astype method` ကို သုံး၍ ဒေတာအမျိုးအစား တစ်ခုမှာ တခြားတစ်ခုသို့ ပြောင်းလဲနိုင်သည်။ (You can explicitly convert or cast an array from one dtype to another using ndarray's `astype` method)

အောက်က ဥပမာတွင် integer ကိန်းများကို floating point အဖြစ်ပြောင်းသည်။ (cast လုပ်သည်။)

```
In [22]: arr = np.array([1, 2, 3, 4, 5])
         arr.dtype
         float_arr = arr.astype(np.float64)
         float_arr.dtype
```

```
Out[22]: dtype('float64')
```

floating point ကို integer dtype အဖြစ်သို့ ပြောင်းလျှင် ဒဿမကိန်းများကို ဖြုတ်ပစ်သည်။ (decimal part will be truncated:)

```
In [23]: arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
         arr
         arr.astype(np.int32)
```

```
Out[23]: array([ 3, -1, -2,  0, 12, 10])
```

ကိန်းအက္ခရာ(value မဟုတ်သည့်)များကိုလည်း (array of strings representing numbers) `astype` ကို အသုံးပြု၍ ကိန်းများအဖြစ် ပြောင်းနိုင်သည်။ (to convert them to numeric form)

```
In [24]: numeric_strings = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
         numeric_strings.astype(float)
```

```
Out[24]: array([ 1.25, -9.6 , 42.  ])
```

string များကို float64 အဖြစ်သို့ ပြောင်းမရပါ။ (string that cannot be converted to float64)။TypeError ပေးလိမ့်မည်။

If casting were to fail for some reason (like a string that cannot be converted to float64), a TypeError will be raised. See that I was a bit lazy and wrote float instead of np.float64; NumPy is smart enough to alias the Python types to the equivalent dtypes. You can also use another array's dtype attribute:

NumPy သည် Python type များကို အလားတူဖြစ်သည့် dtypes(equivalent dtypes) သို့ ပြောင်းပေနိုင်သည်။ array's dtype attribute ကိုလည်း အသုံးပြုနိုင်သည်။

```
In [25]: int_array = np.arange(10)
         calibers = np.array([.22, .270, .357, .380, .44, .50], dtype=np.floa
         t64)
         int_array.astype(calibers.dtype)
```

```
Out[25]: array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

There are shorthand type code strings you can also use to refer to a dtype:

```
In [26]: empty_uint32 = np.empty(8, dtype='u4')
         empty_uint32
```

```
Out[26]: array([         0, 1075314688,          0, 1075707904,          0,
                 1075838976,          0, 1072693248], dtype=uint32)
```

- astype ကို ခေါ်သုံးသည့်အခါတိုင်း array အသစ်တစ်ခုပြုလုပ်ပေးသည်။ (ကော်ပီကူးယူထားသည့် array အသစ်တစ်ခုပြုလုပ်ပေးသည်။)။ dtype အဟောင်းနှင့် အသစ်တို့သည် တူညီနေသည့်အခါမျိုးတွင်လည်း astype ကို ခေါ်သုံးသည့်အခါတိုင်း array အသစ်တစ်ခုပြုလုပ်ပေးသည်။

Arithmetic with NumPy Arrays

*Operations between Arrays and Scalars

Array များကို ပေါင်း နှုတ် မြှောက် စား(operations)သည့်အခါ Array ထဲရှိ element တစ်ခုချင်းစီကို for loops ဖြင့် မပတ်ဘဲ Array ကို ကိန်းတစ်ခုအနေဖြင့် batch operation ပြုလုပ်သည်။ အရွယ်အစား တူညီသည့် array (equal-size arrays)များကို arithmetic operation ပြုလုပ်သည့်အခါ elementwise operation အဖြစ် ပြုလုပ်သည်။

Arrays are important because they enable you to express batch operations on data without writing any for loops. This is usually called vectorization. Any arithmetic op¬erations between equal-size arrays applies the operation elementwise:

```
In [27]: arr = np.array([[1., 2., 3.], [4., 5., 6.]])
         arr
```

```
Out[27]: array([[1., 2., 3.],
                [4., 5., 6.]])
```

```
In [28]: arr * arr
```

```
Out[28]: array([[ 1.,  4.,  9.],
                [16., 25., 36.]])
```

```
In [29]: arr - arr
```

```
Out[29]: array([[0., 0., 0.],
                [0., 0., 0.]])
```

Arithmetic operations with scalars are as you would expect, propagating the value to each element:

```
In [30]: 1 / arr
```

```
Out[30]: array([[1.    , 0.5   , 0.3333],
                [0.25  , 0.2   , 0.1667]])
```

```
In [31]: arr ** 0.5
```

```
Out[31]: array([[1.    , 1.4142, 1.7321],
                [2.    , 2.2361, 2.4495]])
```

```
In [32]: arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
```

```
In [33]: arr2
```

```
Out[33]: array([[ 0.,  4.,  1.],
                [ 7.,  2., 12.]])
```

```
In [34]: arr2 > arr
```

```
Out[34]: array([[False,  True, False],
                [ True, False,  True]])
```

အရွယ်အစား မတူညီသည့် array များ ပေါင်း နှုတ် မြှောက် စား လုပ်ခြင်း(Operations between differently sized arrays) ကို broadcasting ဟုခေါ်သည်။

Basic Indexing and Slicing

NumPy array များကို နည်းမျိုးစုံဖြင့် indexing လုပ်နိုင်သည်။ ဒေတာအားလုံးထဲ မှ ဒေတာတချို့ကို ရွေးချယ်ခြင်း(to select a subset of data)နှင့် ကိန်းတစ်ခုကိုသာရွေးချယ်(to select a individual elements)ကို နည်းမျိုးစုံဖြင့် ပြုလုပ်နိုင်သည်။ One-dimensional array သည် အလွယ်ကူဆုံး ဖြစ်သည်။ Python list ကို indexing/Slicing လုပ်သည့် ပုံစံနှင့် တူညီသည် ဟုဆိုနိုင်သည်။

```
In [35]: arr = np.arange(10)
         arr
```

```
Out[35]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [36]: arr[5]
```

```
Out[36]: 5
```

```
In [37]: arr[5:8]
```

```
Out[37]: array([5, 6, 7])
```

```
In [38]: arr[5:8] = 12
         arr
```

```
Out[38]: array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

`arr[5:8] = 12` သည် ၅ မှ ၈ အထိကို scalar value ဖြစ်သည့် 12 ဖြင့် assign လုပ်သည်။ တစ်နည်းအားဖြင့် ၅ နေရာ မှ ၈ နေရာအထိ တန်ဖိုးများကို 12 ဟု သတ်မှတ်ပေးသည်။

မှတ်ရမည့် အဓိက အချက်မှာ array များကို sliceing လုပ်ခြင်းသည် original array မှ ဆွဲထုတ်၍ ကြည့်ခြင်း(view on the original array)သာဖြစ်သည်။ (array slices are views on the original array.)။ တစ်နည်းအားဖြင့် ဒေတာများကို မကူးယူဘဲ တစ်ခုခု ပြောင်းလဲလျှင် (any modifications) မူလ(source) array တွင်လည်း ပြောင်းသွားလိမ့်မည်။ (array slices are views on the original array. This means that the data is not copied, and any modifications to the view will be reflected in the source array)

```
In [39]: arr_slice = arr[5:8]
         arr_slice
```

```
Out[39]: array([12, 12, 12])
```

```
In [40]: arr_slice[1] = 12345
         arr
```

```
Out[40]: array([    0,     1,     2,     3,     4,    12, 12345,    12,
         8,
                    9])
```

```
In [41]: arr_slice[:] = 64
         arr
```

```
Out[41]: array([ 0,  1,  2,  3,  4, 64, 64, 64,  8,  9])
```

NumPy နှင့် မရင်းနှီးသူများ ထိုအချက်ကို သတိပြုရန်လိုသည်။ တခြားသော array programming language များတွင် မူလ(source) array မှ ကူးယူကြသည်။ (copy လုပ်သည်။)

If you are new to NumPy, you might be surprised by this, especially if they have used other array programming languages which copy data more zealously. As NumPy has been designed with large data use cases in mind, you could imagine performance and memory problems if NumPy insisted on copying data left and right.

higher dimensional array များအတွက် တခြားသော option များရှိသည်။ two-dimensional array တွင် index တစ်ခုစီ၏ element များသည် one-dimensional array ဖြစ်သည်။ (NumPy က index တစ်ခုစီ၏ element များကို scalar အဖြစ် မသတ်မှတ်ပါ)

```
In [42]: arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
         arr2d[2]
```

```
Out[42]: array([7, 8, 9])
```

```
In [43]: arr2d
```

```
Out[43]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

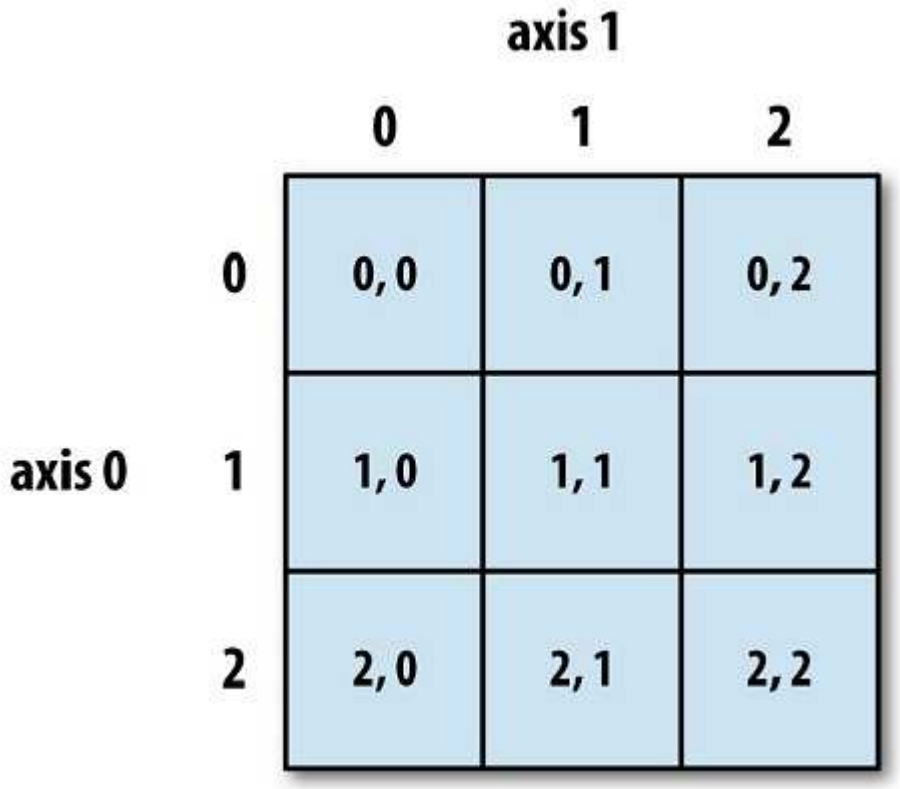
```
In [44]: arr2d[0][2]
```

```
Out[44]: 3
```

Thus, individual elements can be accessed recursively. But that is a bit too much work, so you can pass a comma-separated list of indices to select individual elements. So these are equivalent:

```
In [45]: arr2d[0, 2]
```

```
Out[45]: 3
```



In multidimensional arrays, if you omit later indices, the returned object will be a lower-dimensional ndarray consisting of all the data along the higher dimensions. So in the 2 × 2 × 3 array arr3d

In [46]:
```
arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 1
2]]])
arr3d
```

Out[46]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

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

In [47]:
```
arr3d[0]
```

Out[47]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [48]:
```
arr3d[1]
```

Out[48]:
```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

In [49]:
```
old_values = arr3d[0].copy()
arr3d[0] = 42
arr3d
arr3d[0] = old_values
arr3d
```

Out[49]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

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

In [50]:
```
arr3d[1, 0]
```

Out[50]:
```
array([7, 8, 9])
```

In [51]:
```
x = arr3d[1]
x
x[0]
```

Out[51]:
```
array([7, 8, 9])
```

Note that in all of these cases where subsections of the array have been selected, the returned arrays are views.

Indexing with slices

Like one-dimensional objects such as Python lists, ndarrays can be sliced using the familiar syntax:

In [52]:
```
arr
arr[1:6]
```

Out[52]:
```
array([ 1,  2,  3,  4, 64])
```

Higher dimensional objects give you more options as you can slice one or more axes and also mix integers. Consider the 2D array above, arr2d. Slicing this array is a bit different:

In [53]:
```
arr2d
arr2d[:2]
```

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

In [54]:
```
arr2d[:2, 1:]
```

Out[54]:
```
array([[2, 3],
       [5, 6]])
```

In [55]:
```
arr2d[1, :2]
```

Out[55]:
```
array([4, 5])
```

In [56]:
```
arr2d[:2, 2]
```

Out[56]:
```
array([3, 6])
```

In [57]:
```
arr2d[:, :1]
```

Out[57]:
```
array([[1],
       [4],
       [7]])
```

In [58]:
```
arr2d[:2, 1:] = 0
arr2d
```

Out[58]:
```
array([[1, 0, 0],
       [4, 0, 0],
       [7, 8, 9]])
```

Boolean Indexing

In [59]:
```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe
'])
data = np.random.randn(7, 4)
names
```

Out[59]:
```
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtyp
e='<U4')
```

In [60]:
```
data
```

Out[60]:
```
array([[ 0.0929,  0.2817,  0.769 ,  1.2464],
       [ 1.0072, -1.2962,  0.275 ,  0.2289],
       [ 1.3529,  0.8864, -2.0016, -0.3718],
       [ 1.669 , -0.4386, -0.5397,  0.477 ],
       [ 3.2489, -1.0212, -0.5771,  0.1241],
       [ 0.3026,  0.5238,  0.0009,  1.3438],
       [-0.7135, -0.8312, -2.3702, -1.8608]])
```

In [61]:
```
names == 'Bob'
```

Out[61]:
```
array([ True, False, False,  True, False, False, False])
```

In [62]:
```
data[names == 'Bob']
```

Out[62]:
```
array([[ 0.0929,  0.2817,  0.769 ,  1.2464],
       [ 1.669 , -0.4386, -0.5397,  0.477 ]])
```

In [63]:
```
data[names == 'Bob', 2:]
data[names == 'Bob', 3]
```

Out[63]:
```
array([1.2464, 0.477 ])
```

In [64]:
```
names != 'Bob'
data[~(names == 'Bob')]
```

Out[64]:
```
array([[ 1.0072, -1.2962,  0.275 ,  0.2289],
       [ 1.3529,  0.8864, -2.0016, -0.3718],
       [ 3.2489, -1.0212, -0.5771,  0.1241],
       [ 0.3026,  0.5238,  0.0009,  1.3438],
       [-0.7135, -0.8312, -2.3702, -1.8608]])
```

In [65]:
```
cond = names == 'Bob'
data[~cond]
```

Out[65]:
```
array([[ 1.0072, -1.2962,  0.275 ,  0.2289],
       [ 1.3529,  0.8864, -2.0016, -0.3718],
       [ 3.2489, -1.0212, -0.5771,  0.1241],
       [ 0.3026,  0.5238,  0.0009,  1.3438],
       [-0.7135, -0.8312, -2.3702, -1.8608]])
```

In [66]:
```
mask = (names == 'Bob') | (names == 'Will')
mask
data[mask]
```

Out[66]:
```
array([[ 0.0929,  0.2817,  0.769 ,  1.2464],
       [ 1.3529,  0.8864, -2.0016, -0.3718],
       [ 1.669 , -0.4386, -0.5397,  0.477 ],
       [ 3.2489, -1.0212, -0.5771,  0.1241]])
```

In [67]:
```
data[data < 0] = 0
data
```

Out[67]:
```
array([[0.0929, 0.2817, 0.769 , 1.2464],
       [1.0072, 0.    , 0.275 , 0.2289],
       [1.3529, 0.8864, 0.    , 0.    ],
       [1.669 , 0.    , 0.    , 0.477 ],
       [3.2489, 0.    , 0.    , 0.1241],
       [0.3026, 0.5238, 0.0009, 1.3438],
       [0.    , 0.    , 0.    , 0.    ]])
```

In [68]:
```
data[names != 'Joe'] = 7
data
```

Out[68]:
```
array([[7.    , 7.    , 7.    , 7.    ],
       [1.0072, 0.    , 0.275 , 0.2289],
       [7.    , 7.    , 7.    , 7.    ],
       [7.    , 7.    , 7.    , 7.    ],
       [7.    , 7.    , 7.    , 7.    ],
       [0.3026, 0.5238, 0.0009, 1.3438],
       [0.    , 0.    , 0.    , 0.    ]])
```

Fancy Indexing