Chapter-21 Introduction to NumPy

NumPy သည် Numerical Python ၏ အတိုခေါက် ဖြစ်သည်။ High performance scientific computing လုပ်ရန် နှင့် data analysis လုပ်ရန်အတွက် မရှိမဖြစ် အသုံးပြုရမည့် Python package တစ်ခု ဖြစ်သည်။ Higher-level tool များ အားလုံးနီးပါးသည် NumPy ပေါ်တွင် အခြေခံ၍ တည်ဆောက်ထား ကြသည်။

21.1.1 NumPy ကို အသုံးပြုနိုင်သည့် နယ်ပယ်များ

ကွန်ပျူတာ memory ပေါ်သို့ ဒေတာများ တင်ခြင်း(loading လုပ်ခြင်း)၊ သိမ်းဆည်းခြင်း(storing) နှင့် ပေါင်း, နှုတ်, မြှောက်, စားလုပ်ခြင်း(manipulating) စသည့်ကိစ္စများကို NumPy package က effective ဖြစ်အောင် လုပ်ပေးသည်။ NumPy ကြောင့် ဒေတာတွေကို ထုတ်ရတာ, သွင်းရတာ, ကိုင်တွယ် ရတာ ပိုချောမွေ့ အဆင်ပြေ လာသည်။ ကွန်ပျူတာသည် 0 နှင့် 1 ကိုသာ နားလည်သောကြောင့် ဓာတ်ပုံများ(digital images)များကို pixel လေးများအဖြစ် ခွဲပြီးတော့ 2D array(two dimensional arrays of numbers representing pixel brightness across the area)အဖြစ် ပြောင်းလဲပြီး မှတ်သားသိမ်းဆည်း ရသည်။

အသံဖိုင်များ(audio file)ကိုလည်း အချိန်နဲ့လိုက်ပြီး ပြောင်းလဲနေသည့် တုန်နှုန်း အမြင့်အနိမ့်(one-dimensional arrays of intensity versus time)အဖြစ် ပြောင်းလဲပြီးတော့ မှတ်သားသိမ်းဆည်းသည်။ စာရွက် စာတမ်းများ ကိန်းဂဏန်း၊ အက္ခရာများကိုလည်း 0 နှင့် 1 (binary digit) များအဖြစ် ပြောင်းလဲပြီးတော့ မှတ်သားသိမ်းဆည်းပါသည်။(Text can be converted in various ways into numerical representations, perhaps binary digits representing.)

ဒေတာများ အားလုံးသည် array ပုံစံဖြင့် ဖွဲ့စည်းထားသည့် ကိန်းဂဏန်းများသာ ဖြစ်သည်။ (data fundamentally as arrays of numbers)။ ထို့ကြောင့် ဓာတ်ပုံများ၊ အသံဖိုင်များ၊ ဗွီဒီယိုဖိုင်များ၊ စာရွက် စာတမ်းများ၊ မြေပုံများ စသည်တို့အားလုံးကို analyze လုပ်ရန် သို့မဟုတ် analyze လုပ်နိုင်သည့် အခြေအနေ ရောက်ရန်(analyzable ဖြစ်ရန်)အတွက် ပထမဦးစွာ array ပုံစံဖြင့် ဖွဲ့စည်းထားသည့် ကိန်းဂဏန်းများအဖြစ် ပြောင်းလဲ(transform them into arrays of numbers)ရပါသည်။

ထို့ကြောင့် numerical array များကို သိမ်းဆည်းရန်(efficient storage)နှင့် manipulation လုပ်ရန် အတွက် NumPy package သည် အဓိကကြသည့် နေရာမှ ပါဝင်လာသည်။ NumPy နှင့် ပတ်သက်သည့် သင်ခန်းစာများကို မလေ့လာခင် array များအကြောင်း၊ array များ ပေါင်းခြင်း, မြှောက်ခြင်း စသည်တို့ကို ဦးစွာ ရင်းနှီးကျွမ်းဝင်အောင် ပြန်လည် လေ့လာထားသင့်သည်။

NumPy array များသည် Python ရှိ list အမျိုးအစား(built-in) ကဲ့သို့ပင် ဖြစ်သည်။ သို့သော် NumPy package နှင့် Pandas package တို့သည် အလွန်ကြီးမားသည့် ဒေတာများကို အဆင်ပြေပြေ လွယ်ကူစွာ ကိုင်တွယ် သိမ်းဆည်း လုပ်ကိုင်ရန် အထူးတီထွင်ထားသည့် package များ ဖြစ်သောကြောင့် အလွန် အစွမ်း ထက်သည်။

အလွယ်ကူဆုံး ဥပမာဖြင့် ပြောရလျှင် Python's built-in list များဖြင့် အလွန်ကြီးမားသည့် ဒေတာ များကို ကိုင်တွယ်ခြင်းသည် ကိန်းများစွာကို စာရွက်ပေါ်တွင် လက်ဖြင့် ပေါင်း, နှုတ်, မြှောက်, စား လုပ်သကဲ့သို့

ဖြစ်ပြီး NumPy package နှင့် Pandas package တို့ကို အသုံးပြုပါက calculator ဖြင့် အလွယ်တကူ ပေါင်း, နှုတ်, မြောက်, စား လုပ်နိုင်သကဲ့သို့ ဖြစ်လိမ့်မည်။

21.1.2 NumPy Basics: Arrays and Vectorized Computation

- **`ndarray`**(nth dimenstional array)သည် multidimensional array ဖြစ်သည်။ Vector နည်းဖြင့် ပေါင်း၊ နှုတ်၊ မြောက်၊ စား လုပ်သည့် ကိစ္စရပ်(vectorized arithmetic operation)များကို လုပ်ပေးနိုင်သည်။ ထူးခြားစွာ broadcasting လုပ်နိုင်သည့် စွမ်းရည် ရှိသည်။
- **`ndarray`** တွင် array များကို loop ပတ်ရန် မလိုဘဲ အလွန်လျှင်မြန်စွာ ပေါင်း၊ နှုတ်၊ မြောက်၊ စား လုပ်နိုင်သည့် standard mathematical function များ ပါရှိသည်။
- Memory အပေါ်သို့ ဒေတာများ ရေးခြင်း(writing on disk) ၊ disk ပေါ်မှာ ဒေတာများကို ဖတ်ယူခြင်းတို့ ပြုလုပ်နိုင်သည့် tool များစွာ ရှိသည်။ Memory-mapped file နှင့် တွဲ၍ အလုပ်လုပ်နိုင်သည်။
- အက္ခရာသင်္ချာ(linear algebra) ၊ random number ထုတ်ခြင်း(generation) နှင့် Fourier transform စသည့် အဆင်မြင့် သင်္ချာများကို တွက်ချက်နိုင်စွမ်းရှိသည်။
- C, C++ နှင့် Fortran စသည် တခြားသော programming language များနှင့် တွဲ၍ သုံးနိုင်သည့် (code integrating) tool များ ရှိသည်။ NumPy က low-level language ဖြင့် ရေးထားသည့် ပြင်ပ(external libraries)မှ ဒေတာများကို Python ထဲသို့ ထည့်ပေးနိုင်သည်။ 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 လူကြိုက်များခြင်း ဖြစ်သည်။
- NumPy တွင် high-level data analytical function များ မပါသော်လည်း NumPy arrays နှင့် array-oriented computing ကို နားလည်အောင် ကြိုးစားသင်ယူခြင်းဖြင့် high-level data analytical အလုပ်များ လုပ်နိုင်သည့် pandas ကို ကျွမ်းကျင်စွာ အသုံးပြုနိုင်လိမ့်မည်။ Python ကို စတင် လေ့လာသူများ၊ data analysis နှင့် machine learning ကို စတင်လေ့လာသူများသည် NumPy ကို အချိန်ပေး လေ့လာရန် တိုက်တွန်းလိုသည်။

21.1.3 NumPy ၏ လုပ်ဆောင်ချက်များ

Data analysis application လုပ်ငန်းများအတွက် NumPy ၏ လုပ်ဆောင်ချက်များမှာ

- Data များကို အသုံးပြုနိုင်အောင် ပြင်ဆင်ခြင်း(munging)၊ သန့်စင်ခြင်း(cleaning)၊ subsetting လုပ်ခြင်း၊ filtering လုပ်ခြင်း၊ transformation လုပ်ခြင်း စသည့် အလုပ်များကို အလွန်လျှင်မြန်သည့် vectorized array operation ဖြင့် NumPy က လုပ်ဆောင်ပေးနိုင်သည်။
- Data များကို array ပုံစံဖြင့် ရွေးချယ်ခြင်း၊ ရှေ့နောက် စဉ်ခြင်း၊ စုစုပေါင်း တန်ဖိုးရှာခြင်း၊ ပျမ်းမျှတန်ဖိုး ရှာခြင်း စသည်တို့ ကို NumPy က ပြုလုပ်ပေးနိုင်သည်။
- Data များကို ပေါင်းခြင်း(merging)၊ ဆက်ခြင်း(joining) စသည်တို့ကိုလည်း ပြုလုပ်ပေးနိုင်သည်။

ထို့ကြောင့် NumPy နှင့် Pandas package ကို ကျွမ်းကျင်စွာ အသုံးပြုနိုင်ရန် အချိန်ပေး ကြိုးစားခြင်း သည် ခက်ခဲသည့် အလုပ်များကို လွယ်ကူအောင် အချိန်ပေးခြင်း ဖြစ်သည်။

Anaconda install လုပ်ထားလျှင် NumPy package ပါဝင်ပြီး ဖြစ်သည်။ NumPy package ကို install မလုပ်ရသေးလျှင် https://www.numpy.org တွင် ဒေါင်းလုပ် လုပ်နိုင်သည်။

Terminal တွင် pip command ဖြင့် numpy package ကို အောက်ပါတိုင်း install လုပ်နိုင်သည်။

```
pip install numpy
```

Install လုပ်ပြီးပါက သို့မဟုတ် မိမိ၏ NumPy package version ကို သိလိုပါက အောက်ပါတိုင်း စစ်နိုင်သည်။

```
import numpy
numpy.__version__
```

```
'1.11.1'
```

NumPy package ကို အသုံးပြုရန် လိုအပ်သည် အခါတိုင်း import လုပ်ရသည်။ NumPy ကို np ဟု အတိုခေါက် နာမည်ပေးသည်။ ထိုသို့ np အတိုခေါက် နာမည် မပေးပါ numpy ဟု မူလအတိုင်း ရေးရလိမ့်မည်။

```
import numpy as np
```

ထိုသို့ import လုပ်ပြီးပါက စတင် အသုံးပြုနိုင်ပြီ ဖြစ်သည်။ လိုအပ်သည့် အချက်များကို ပြင်ဆင်သည်။

NumPy array နှင့် python list တို့၏ လုပ်နိုင်သည့် အလုပ် တူညီသော်လည်း အလုပ်လုပ်ပုံ ကွာခြားသောကြောင့် အလုပ်လုပ်ရန်ကြာချိန် ကွာခြားသည်။ NumPy array သည် python list ထက် ပို၍ efficient ဖြစ်ကို အောက်တွင် နှိုင်းယှဉ်ပြထားသည်။

my_arr ဆိုသည့် array တစ်ခု နှင့် my_list ဆိုသည် list တစ်ခု တည်ဆောက်သည်။

```
import numpy as np
my_arr = np.arange(1000000)     # NumPy array တစ်ခု တည်ဆောက်သည်။
my_arr
```

```
array([     0,      1,      2, ..., 999997, 999998, 999999])
```

```
my_list = list(range(1000000)) # list တစ်ခု တည်ဆောက်သည်။
my_list
```

```
[0, 1, 2, 3, 4, 5,  . . . . . 999997, 999998, 999999])
```

my_arr နှင့် my_list ထဲတွင် ပါဝင်သည့် element များ တူညီကြသည်။ NumPy array သည် python list ထက် ပို၍ မြန်ဆန်စွာ အလုပ် လုပ်ပေးနိုင်ပုံ အောက်တွင် ဥပမာဖြင့် ဖော်ထားသည်။ အောက်တွင် အရွယ်အစား တူညီသည့် NumPy array နှင့် python list တို့ကို (၂)နှင့် မြှောက်ရန်အတွက် ကြာချိန်ကို နှိုင်းယှဉ် ပြထားသည်။

```
%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: 35 ms
```

```
Wall time: 1.96 s
```

NumPy array သည် python list ထက် အဆများစွာ ပိုမြန်သည်။

21.1.4 The NumPy ndarray: A Multidimensional Array Object

NumPy ၏ ထူးခြားသည့် အချက်တစ်ခုမှာ **ndarray** ဟုခေါ်သည့် N-dimensional array object တစ်ခု ဖြစ်သည်။ **ndarray** သည် အလွန်ကြီးမားသည့် ဒေတာ(large data sets)များကို လွယ်ကူ လျှင်မြန်စွာ အဆင်ပြေပြေ ဆောင်ရွက်ပေးနိုင်သည်။ အဘယ်ကြောင့်ဆိုသော် အလွန်ကြီးမားသည့် ဒေတာများကို arrays အဖြစ် သိမ်းဆည်းပြီး ကိန်းတစ်လုံးကဲ့သို့ loop မပတ်ဘဲ ပေါင်း၊ နှုတ်၊ မြောက်၊ စား(mathematical operations) လုပ်ပေးနိုင်သောကြောင့် ဖြစ်သည်။

ndarray သည် အမျိုးတူသည့် ဒေတာများ(homogeneous data)ကိုသာ သိမ်းဆည်းထားနိုင်သည့် ကွန်တိန်နာ(generic multi-dimensional container)တစ်ခု ဖြစ်သည်။ Homogeneous data ဆိုသည်မှာ အမျိုးအစား တူညီသည့် ဒေတာများ (all of the elements must be the same type)ကို ဆိုလိုသည်။

21.2 Array တစ်ခု၏ Shape, dtype, Dimension နှင့် Size

Array များ၏ အချက်အလက်များကို သိလိုပါက `.shape` `.dtype` , `.ndim` , `.size` စသည်တို့ကို အသုံးပြုနိုင်သည်။

- `.shape` ဆိုသည်မှာ row အရေအတွက်နှင့် ကော်လံ အရေအတွက် ဖြစ်သည်။
- `.dtype` ကို array ၏ data type ကို သိလိုသည့်အခါ အသုံးပြုသည်။
- `.ndim` ကို array ၏ dimenstion ကို သိလိုသည့်အခါ အသုံးပြုသည်။
- `.size` သည် array အရွယ်အစား(size of the arrays)ကို ဖတ်ရန် ဖြစ်သည်။

Array တစ်ခု ပြုလုပ်ရန်အတွက် အလွယ်ကူဆုံးနည်းမှာ **np.array()** function ကို အသုံးပြုခြင်း ဖြစ်သည်။

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

```
list
```

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

```
arr1 = np.array(data1)
arr1
```

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

အတိုင်းအတာ တူညီသည့် list များ ပါဝင်သည့် list(a list of equal-length lists)ကို multi-dimensional array အဖြစ် တည်ဆောက်နိုင်သည်။ Nested sequences ဖြစ်သည်။ `np.array(data2)` ဖြင့် data2 ကို array တစ်ခု ဖြစ်အောင် ပြုလုပ်သည်။

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)      # two dimensional array ဖြစ်အောင် ပြုလုပ်သည်။
```

```
arr2
```

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

21.2.1 Shape of Array

Array တစ်ခု၏ shape ဆိုသည်မှာ row အရေအတွက်နှင့် ကော်လံ အရေအတွက် ဖြစ်သည်။ တစ်နည်းအားဖြင့် shape ဆိုသည်မှာ row အရေအတွက်နှင့် ကော်လံ အရေအတွက်ကို ဖော်ပြထားသည့် tuple တစ်ခု ဖြစ်သည်။ (a tuple indicating the size of each dimension)။

data2 array သည် two dimensional array ဖြစ်သည်။ `.shape` ဖြင့် array ၏ shape ကို စစ်သည်။

```
arr2.shape
```

```
(2, 4)
```

21.2.2 Dimension of Array

Row နှင့် ကော်လံ ပေါင်းနှစ်ခုကို ရည်ညွှန်း၍ two dimensional array ဟု ခေါ်သည်။

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

```
arr2.ndim
```

```
2
```

21.2.3 Array Data Type

dtype ဆိုသည်မှာ NumPy array ထဲရှိ ဒေတာများ၏ အမျိုးအစားကို ဖော်ပြသည့် object တစ်ခု ဖြစ်သည်။ (an object describing the data type of the array)။ Array ထဲတွင် ပါရှိသည့် element များ၏ ဒေတာ အမျိုးအစားများကို dtype object ထဲတွင် သိမ်းဆည်းထားသည်။ (The data type is stored in a special dtype object.)

အောက်တွင် ဥပမာဖြင့် ရှင်းပြရန် numpy ကို import လုပ်သည်။ Numpy ကို import လုပ်သည့် အခါတိုင်း np ဟု အတိုခေါက် အသုံးပြုမည်။ `np.random.randn(2, 3)` ဖြင့် random နံပါတ်များ ပါဝင်သည့် 2 x 3 array တစ်ခု ဖြစ်အောင် တည်ဆောက်သည်။

```
import numpy as np
data = np.random.randn(2, 3)           # Generate some random data
data
```

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

data ဆိုသည့် array ၏ shape ကို **.shape** ကြည့်သည်။ (2x3 array ဖြစ်သည်။)

```
data.shape  # .shape ဖြင့် row အရေအတွက်နှင့် ကော်လံ အရေအတွက်ကို ဖတ်သည်။
```

```
(2, 3)
```

data ဆိုသည့် array ၏ ဒေတာ အမျိုးအစားများကို `.dtype` ကြည့်သည်။ (float64 ဖြစ်သည်။)

```
data.dtype  # array ၏ ဒေတာအမျိုးအစား(data type)ကို စစ်ရန်
```

```
dtype('float64')
```

arr1 သည် float64 ဒေတာ အမျိုးအစားများ ပါရှိသည့် array တစ်ခု ဖြစ်သည်။ arr2 သည် interger ဒေတာ အမျိုးအစားများ ပါရှိသည့် array တစ်ခု ဖြစ်သည်။

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

```
dtype('float64')
```

```
arr2.dtype
```

```
dtype('int32')
```

21.3 Array တစ်ခု တည်ဆောက်ခြင်း

21.3.1 Fixed-Type Arrays in Python

Python တွင် data များကို ကောင်းစွာ သိမ်းဆည်းပြီး fixed-type data buffer များဖြင့် လျှင်မြန်စွာ ပေါင်း၊ နုတ်၊ မြှောက်၊ စား လုပ်နိုင်သည့် နည်းများစွာ ရှိသည်။ ဒေတာများသည် အမျိုးအစားတူညီ(uniform type)လျှင် built-in `array` module ကို အသုံးပြုပြီး array များ တည်ဆောက်နိုင်သည်။ အောက်တွင် `array` module ဖြင့် array တစ်ခုတည်ဆောက်ပုံ ဖော်ပြထားသည်။

```
import array
L = list(range(10))
A = array.array('i', L)
A
```

```
array('i', [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

`'i'` သည် array အတွင်းရှိကိန်းများသည် integer များ ဖြစ်ကြောင်းကို ဖော်ပြသည်။ NumPy package မှ `ndarray` object သည် ပို အသုံးကျသည်။

21.3.2 NumPy ဖြင့် Array တစ်ခု တည်ဆောက်ခြင်း

NumPy မှ `ndarray` ဖြင့်လည်း array များ တည်ဆောက်နိုင်သည်။ NumPy array များသည် ပေါင်း၊ နုတ်၊ မြှောက်၊ စား operation များကို ပိုလွယ်ကူစွာ ပိုလျှင်မြန်စွာ လုပ်ပေးနိုင်သည်။ အောက်တွင် NumPy array တည်ဆောက်ပုံ နည်းအမျိုးမျိုးကိုသာ ဖော်ပြသွားမည်။

ပထမဦးစွာ NumPy ကို အသုံးပြုရန်အတွက် import လုပ်ရပါမည်။ np အတိုခေါက် ခေါ်မည်။ np မဟုတ်ဘဲ တခြား မိမိ ကြိုက်သည့် နာမည်ပေးနိုင်သည်။

```
import numpy as np
```

21.3.3 Python List မှ Arrays တစ်ခု တည်ဆောက်ခြင်း

အောက်တွင် `np.array` ကို အသုံးပြု၍ Python list တစ်ခု ကို NumPy array တစ်ခု ဖြစ်အောင် တည်ဆောက်မည်။

```
np.array([1, 4, 2, 5, 3])  # integer array တစ်ခု တည်ဆောက်သည်။
```

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

```
one = np.random.randint(10, size = 6) # Creating a one dimensional array.
print(one)
```

```
array([5 0 3 3 7 9])
```

Python list နှင့် မတူသည့်အချက်မှာ NumPy array ထဲတွင်ရှိသည့် ဒေတာများ အားလုံးသည် အမျိုးအစား တူညီသည့် ဒေတာများ(same data type)သာ ဖြစ်ရမည်။

Python list ထဲတွင် အမျိုးအစား မတူညီသည့် ဒေတာများ(different data type) ထည့်ထားနိုင်သည်။ NumPy array ထဲတွင်ရှိသည့် အမျိုးအစားတူညီသည့် ဒေတာများ(same data type)သာ ဖြစ်ရမည်။

Python list မှ NumPy array အဖြစ်သို့ ပြောင်းသည့်အခါ အမျိုးအစား မတူညီသည့် ဒေတာများ (different data type)ကို အမျိုးအစားတူညီသည့် ဒေတာများ(same data type) ဖြစ်အောင် **up-cast** လုပ်သည်။ ဥပမာ- integer နှင့် floating point များ ရောနှောပါဝင်နေသည့် list ကို NumPy array အဖြစ်သို့ ပြောင်းသည့်အခါ အားလုံးကို floating point များဖြစ်အောင် up-cast လုပ်သည်။

```
np.array([3.14, 4, 2, 3])  # integer 4 နှင့် 2 ကို up-cast လုပ်သည်။
```

```
array([3.14, 4.  , 2.  , 3.  ])
```

NumPy array ကို မိမိအလိုရှိသည့် ဒေတာအမျိုးအစား(data type) ဖြစ်စေရန်အတွက် **dtype** keyword ကို အသုံးပြုနိုင်သည်။ ဥပမာ- NumPy array တွင်ရှိ ဒေတာ များ အားလုံး float32 ဖြစ်စေရန် `dtype='float32'` ဟု ရေး၍ မိမိအလိုရှိသည့် ဒေတာအမျိုးအစား(data type) ဖြစ်အောင် ပြောင်းနိုင်သည်။ အောက်တွင် 'float32' သို့ ပြောင်းပုံကို ဖော်ပြထားသည်။

```
np.array([1, 2, 3, 4], dtype='float32')
```

```
array([1., 2., 3., 4.], dtype=float32)
```

21.3.4 NumPy Multidimensional Array

Python list နှင့် မတူသည့်အချက်မှာ NumPy array များကို multi-dimensional ဖြစ်အောင် လုပ်ပေးနိုင်သည်။ အောက်တွင် list of list ကို multidimensional array တစ်ခုအဖြစ် initializing လုပ်ပုံကို ဖော်ပြထားသည်။ random များ ပါဝင်သည့် 2 x 5 array တစ်ခုအဖြစ်သည်။

```
# Creating a two dimensional array.
two = np.random.randint(10, size = (2, 5))
print(two)
```

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

အပေါ်မှ two-dimensional array တွင် inner list များသည် row များ ဖြစ်ကြသည်။

```
# nested lists result in multi-dimensional arrays
np.array([range(i, i + 3) for i in [2, 4, 6]])
```

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

21.3.5 Size of Array

အပေါ်တွင် တည်ဆောက်ခဲ့သည့် array `"one"` နှင့် `"two"` တို့၏ size ကို သိရန် `.size` ကို သုံးသည်။

```
# Using the attribute to determine the size of the arrays.
print("one ==>", one.size)
print("two ==>", two.size)          # array “two” တွင် entry (၁၀)ခု ပါဝင်သည်။
```

```
one ==> 6
two ==> 10
```

21.3.6 Creating Arrays from Scratch

အောက်တွင် NumPy array တည်ဆောက်ပုံကို ဖော်ပြထားသည်။ `np.zeros(10, dtype=int)` ဖြင့် သုည(၁၀)ခု ပါဝင်သည့် NumPy array တစ်ခု တည်ဆောက်သည်။

```
# Create a length-10 integer array filled with zeros
np.zeros(10, dtype=int)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

`np.ones()` ဖြင့် (၁)များသာ ပါသည့် 3x5 floating-point array တစ်ခု တည်ဆောက်သည်။

```
# Create a 3x5 floating-point array filled with ones
np.ones((3, 5), dtype=float)  # 3 နှင့် 5 သည် row နှင့် ကော်လံ အရေအတွက် ဖြစ်သည်။
```

```
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])
```

`np.full()` ဖြင့် 3.14 တန်ဖိုးများသာပါသည့် 3x5 floating-point array တစ်ခုတည်ဆောက်သည်။

```
# Create a 3x5 array filled with 3.14
np.full((3, 5), 3.14)     # 3 နှင့် 5 သည် row နှင့် ကော်လံ အရေအတွက် ဖြစ်သည်။
```

```
array([[3.14, 3.14, 3.14, 3.14, 3.14],
       [3.14, 3.14, 3.14, 3.14, 3.14],
```

```
       [3.14, 3.14, 3.14, 3.14, 3.14]])
```

၀ မှ ၂၀ အထိ နှစ်ခုခြားစီ ဖြစ်သည့် ကိန်းများဖြင့် တည်ဆောက်ထားသည့် array ပြုလုပ်သည်။

```
# Create an array filled with a linear sequence
# Starting at 0, ending at 20, stepping by 2
# (this is similar to the built-in range() function)
np.arange(0, 20, 2)
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

`np.linspace()` ကို အသုံးပြု၍ အစ(၀)မှ အဆုံး(၁)အတွင်း ညီမျှစွာ ခြားထားသည့် ကိန်း(၅)ခု ဖြင့် တည်ဆောက်ထားသည့် array တစ်ခု ပြုလုပ်သည်။

```
# Create an array of five values evenly spaced between 0 and 1
np.linspace(0, 1, 5)
```

```
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ])
```

သုည(၀)မှ (၁)အတွင်း random value များ(uniformly distributed) ပါသည့် 3x3 array တည်ဆောက်ပုံကို အောက်တွင် ဖော်ပြထားသည်။

```
# Create a 3x3 array of uniformly distributed
# random values between 0 and 1
np.random.random((3, 3))
```

```
array([[0.58569043, 0.18114868, 0.01690071],
       [0.83503977, 0.74005786, 0.89401027],
       [0.67090455, 0.59241442, 0.23957754]])
```

သုည(၀)မှ (၁)အတွင်း random value များ(normally distributed) ပါသည့် 3x3 array တည်ဆောက်ပုံကို အောက်တွင် ဖော်ပြထားသည်။ (3, 3) သည် array ၏ shape ဖြစ်သည်။

```
# Create a 3x3 array of normally distributed random values
# with mean 0 and standard deviation 1
np.random.normal(0, 1, (3, 3))    # 3 x 3 array
```

```
array([[ 1.51772646,  0.39614948, -0.10634696],
       [ 0.25671348,  0.00732722,  0.37783601],
       [ 0.68446945,  0.15926039, -0.70744073]])
```

သုည(၀)မှ (၁)အတွင်း random ကိန်းပြည့်များ(integer)သာပါသည့် 3x3 array တည်ဆောက်ပုံ အောက်တွင် ဖော်ပြထားသည်။

```
# Create a 3x3 array of random integers in the interval [0, 10)
np.random.randint(0, 10, (3, 3))
```

```
array([[2, 3, 4],
       [5, 7, 8],
```

```
       [0, 5, 0]])
```

21.3.7 Identity Matrix တည်ဆောက်ခြင်း

`np.eye()` သည် 2-D identity matrix တည်ဆောက်ပေးသည်။ Identity matrix တွင် diagonal ရှိ ကိန်းများအားလုံးသည် 1 ဖြစ်ပြီး ကျန်ကိန်းများ အားလုံးသည် zeros ဖြစ်သည်။

```
np.eye(3)    # Create a 3x3 identity matrix
```

```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

```
# Create an uninitialized array of three integers
np.empty(3)
```

```
array([ 1.,  1.,  1.])
```

21.3.8 ndarrays

np.array function ထဲတွင် တခြားသော နည်းများဖြင့် array အသစ်များကို ပြုလုပ်နိုင်သည့် နည်းများလည်း ပါဝင်သည်။ ဥပမာ- array ထဲရှိ element များအားလုံး သုည(0's)များသာ ပါဝင်သည့် array တစ်ခု ကို `np.zeros()` ဖြင့် ပြုလုပ်နိုင်သည်။ Array ထဲရှိ element များအားလုံး 1's သာ ပါဝင်သည့် array တစ်ခုကို `np.ones()` ဖြင့် ပြုလုပ်နိုင်သည်။

Array အသစ်များ ပြုလုပ်သည့်အချိန်တွင် မိမိအလိုရှိသည့် အရွယ်အစား(length or shape) ကို row အရေအတွက်နှင့် ကော်လံ အရေအတွက် ထည့်သွင်း၍ ပြုလုပ်နိုင်သည်။ မိမိ အလိုရှိသည့် array ၏ shape အတိုင်းလည်း တည်ဆောက်နိုင်သည်။

```
np.zeros(10)      # သုည(0's)များသာ ပါဝင်သည့် array တစ်ခု တည်ဆောက်သည်။
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
np.zeros((3, 6))  # သုည(0's)များသာ ပါဝင်သည့် 3x6 array တစ်ခု တည်ဆောက်သည်။
```

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

```
np.ones((3, 2))    # 1 များသာ ပါဝင်သည့် 3x2 array တစ်ခု တည်ဆောက်သည်။
```

```
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
```

np.empty() ဖြင့် empty array တစ်ခု ပြုလုပ်နိုင်သည်။ Array ထဲရှိ element များ၏ တန်ဖိုးကို initializing မလုပ်လိုသည့်အခါတွင် empty array ပြုလုပ်ကြသည်။ **np.empty** ဖြင့် ပြုလုပ်ထားသည့် empty

array တစ်ခုသည် ထို array ထဲရှိ element များ၏ တန်ဖိုး(value)များအားလုံးသည် သုည(zero) ဖြစ်လိမ့်မည်ဟု အာမ, မခံနိုင်ပါ။ သေချာပေါက် မယူဆသင့်ပါ။

```
np.empty((2, 3, 2))  # 2 x 3 x 2 empty array တစ်ခု ပြုလုပ်သည်။
```

```
array([[[0., 0.],
        [0., 0.],
        [0., 0.]],

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

အပေါ်မှ array ၏ data type သည် dtype('float64') ဖြစ်သည်။ 3 dimenasion array ဖြစ်သည်။ Array ၏ shape မှာ (2, 3, 2)ဖြစ်သည်။

`arange` သည် **range()** function ဖြစ်သည်။ **range()** function သည် ကိန်းတစ်လုံးချင်းစီ အတွက်သာ အသုံးပြုနိုင်သည်။ **`arange`** သည် array အတွက် **range()** function ဖြစ်သည်။ (array-valued version)။ 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)

21.4 Data Types for ndarrays

`Data type` သို့မဟုတ် **`dtype`** သည် ndarray ၏ ဒေတာများနှင့် သက်ဆိုင်သည့် အချက်အလက်များကို သိမ်းဆည်းထားသည့် special object တစ်ခု ဖြစ်သည်။

Dtypes ၏ ထူးခြားသည့် စွမ်းရည်ကြောင့် NumPy သည် စွမ်းအားပိုကောင်း(powerful and flexible)ခြင်း ဖြစ်သည်။ တိုက်ရိုက်(direct) mapping လုပ်နိုင်ခြင်းကြောင့် ဒေတာများအလွယ်တကူ ဖတ်ခြင်း၊ ရေးခြင်း ပြုလုပ်နိုင်သည်။ (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။

Data type သို့မဟုတ် dtype သည် ndarray ၏ ဒေတာများနှင့် သက်ဆိုင်သည့် အချက်အလက်များကို သိမ်းဆည်းထားသည့် special object တစ်ခုဖြစ်သည်။ Dtypes ၏ ထူးခြားသည့် စွမ်းရည်ကြောင့် NumPy သည် စွမ်းအားပိုကောင်း(powerful and flexible)ခြင်း ဖြစ်သည်။ တိုက်ရိုက်mapping လုပ်နိုင်ခြင်းကြောင့် (map directly) ဒေတာများ အလွယ်တကူ ဖတ်ခြင်း၊ ရေးခြင်း ပြုလုပ်နိုင်သည်။ (easy to read and write binary streams of data to disk)။

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	O	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'`.

`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။)

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

```
dtype('int32')
```

```
float_arr = arr.astype(np.float64)    # arr array ကို float64 အဖြစ်ပြောင်းသည်။
float_arr.dtype
```

```
dtype('float64')
```

```
float_arr
```

```
array([1., 2., 3., 4., 5.])
```

Floating point ကို integer dtype အဖြစ်သို့ ပြောင်းလျှင် ဒဿမကိန်းများကို truncate လုပ်သည်။ (decimal part will be truncated:)

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

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

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

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

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

Casting လုပ်၍ မရသည့် dtype များ ရှိသည်။ ဥပမာ- string များကို float64 အဖြစ်သို့ ပြောင်း၍ (convert လုပ်၍) မရပါ။ TypeError ပေးလိမ့်မည်။

NumPy သည် Python type များကို အလားတူ ဖြစ်သည့် dtypes(equivalent dtypes)သို့ ပြောင်းပေးနိုင်သည်။ Array's dtype attribute ကိုလည်း အသုံးပြုနိုင်သည်။

```
int_array = np.arange(10)
calibers=np.array([.22, .270, .357, .380, .44, .50],dtype=np.float64)
int_array.astype(calibers.dtype)
```

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

astype ကို ခေါ်သုံးသည့်အခါတိုင်း array အသစ်တစ်ခု ပြုလုပ်ပေးသည်။ (ကော်ပီကူးယူထားသည့် array အသစ် တစ်ခု ပြုလုပ်ပေးသည်။) dtype အဟောင်းနှင့် အသစ်တို့သည် တူညီနေသည့် အခါမျိုးတွင်လည်း astype ကို ခေါ်သုံးသည့်အခါတိုင်း array အသစ်တစ်ခု ပြုလုပ်ပေးသည်။

21.5 Arithmetic with NumPy Arrays

21.5.1 Operations Between Arrays and Scalars

Array များကို ပေါင်း၊ နှုတ်၊ မြှောက်၊ စား(operations)လုပ်သည့်အခါ array ထဲရှိ element တစ်ခု ချင်းစီကို for loop ဖြင့် မပတ်ဘဲ array ကို ကိန်းတစ်ခုအနေဖြင့် batch operation ပြုလုပ်သည်။ အရွယ်အစား တူညီသည့် array (equal-size arrays)များကို arithmetic operation ပြုလုပ်သည့်အခါ element wise operation အဖြစ် ပြုလုပ်သည်။

Loop မပတ်ဘဲ array အချင်းချင်း ပေါင်း၊ နှုတ်၊ မြှောက်၊ စား(batch operations) လုပ်နိုင်သည်။ Loop မပတ်ဘဲ ထိုသို့ batch operation လုပ်ခြင်းကို **vectorization** ဟုခေါ်သည်။

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

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

```
arr * arr                # array အချင်းချင်း မြှောက်သည်။
```

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

```
arr - arr                # နုတ်သည်။
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

"array" ၊ "NumPy array" သို့မဟုတ် "ndarray" စသည့် စကားလုံးများသည် ndarray object (N dimension array)ကို ဆိုလိုသည်။ အောက်တွင် array ကို ကိန်းတစ်ခုကဲ့သို့ ပေါင်း, နုတ်, မြှောက်, စား လုပ်နိုင်ပုံကို ဖော်ပြထားသည်။ data ဆိုသည့် array ကို (၁၀)ဖြင့် မြှောက်သည်။

```
data * 10
```

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

array နှစ်ခု ပေါင်းသည်။

```
data + data
```

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

Array များကို ကိန်းတစ်လုံး(scalar)ဖြင့် ပေါင်း၊ နှုတ်၊ မြှောက်၊ စား(arithmetic operation) လုပ်နိုင်သည်။ Array တစ်ခု၏ reciprocal ကို ရှာသည်။

```
1 / arr                # reciprocal
```

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

```
arr ** 0.5             # square root of 2
```

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

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

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

```
arr2 > arr                          # boolean
```

```
array([[False,  True, False],
       [ True, False, True]])
```

အရွယ်အစား မတူညီသည့် array များ ပေါင်း၊ နှုတ်၊ မြှောက်၊ စား လုပ်ခြင်း(operations between differently sized arrays)ကို **broadcasting** လုပ်သည် ဟု ခေါ်သည်။

21.5.2 Splitting of Arrays

```
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 0])
print(a)
print("------------------------")
b, c, d = np.split(a, [2, 5])    # 2 သည် ပထမ array ၏ အဆုံးကိန်း ဖြစ်သည်။
print(b, c, d)
```

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

```
b, c, d = np.split(a, [4, 8])    # 8 သည် ဒုတိယ array ၏ အဆုံးကိန်း ဖြစ်သည်။
print(b, c, d)
```

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

အောက်တွင် array များကို မိမိလိုသလို vertical split နှင့် horizontal split ကို သုံး၍ ပိုင်းခြားနိုင်ပုံကို ဖော်ပြထားသည်။

```
grid = np.arange(16).reshape(4, 4) # နမူနာ ပြရန် array တစ်ခု ဆောက်သည်။
print(grid)
```

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

Vertical Split (`vsplit`)

ဒေါင်လိုက်အတိုင်း အပေါ်ပိုင်း array တစ်ခု နှင့် အောက်ပိုင်း array ဖြစ်အောင် ခွဲခြင်းကို vertical split လုပ်သည်ဟု ခေါ်သည်။ `upper` array နှင့် `lower` array ဖြစ်အောင် ခွဲထုတ်သည်။

```
upper, lower = np.vsplit(grid, [2])
print(upper)
print("------------------------")
print(lower)
```

```
[[0 1 2 3]
```

```
 [4 5 6 7]]
------------------------
[[ 8  9 10 11]
 [12 13 14 15]]
```

Horizontal Split (hsplit)

```
left, right = np.hsplit(grid,[2])   # left နှင့် right array ဖြစ်အောင် ခွဲထုတ်သည်။
print(left)
print("------------------------")
print(right)
```

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

21.6 Basic Indexing and Slicing

NumPy array များကို နည်းမျိုးစုံဖြင့် indexing လုပ်နိုင်သည်။ ဒေတာအားလုံးထဲမှ ဒေတာတချို့ကို ရွေးချယ်ခြင်း(select a subset of data)နှင့် ကိန်းတစ်ခုတည်းကိုသာ ရွေးချယ်ခြင်း(to select a individual elements)စသည့်ဖြင့် နည်းမျိုးစုံဖြင့် ပြုလုပ်နိုင်သည်။ One-dimensional array သည် အလွယ်ကူဆုံး ဖြစ်သည်။ Python list ကို indexing သို့မဟုတ် slicing လုပ်သည့် ပုံစံနှင့် တူညီသည်။

```
arr_2 = np.arange(10)
arr_2
```

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

```
arr_2[5]            # index နံပါတ် (၅)ကို ဖတ်သည်။
```

```
5
```

```
arr_2[5:8]           # index နံပါတ် (၅) မှ (၈)အထိကို ဖတ်သည်။
```

```
array([5, 6, 7])
```

```
arr_2[5:8] = 12      # index နံပါတ် (၅) မှ (၈)အထိကို 12 ဖြင့် assign လုပ်သည်။
arr_2
```

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

arr[5:8] = 12 သည် index နံပါတ်(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။ တစ်နည်းအားဖြင့် ဒေတာများကို မကူးယူဘဲ မူလ (source) array တွင် modification ပြုလုပ်သောကြောင့် မူလ(source) array သည် ပြောင်းလဲသွား လိမ့်မည်။

```
arr_slice = arr_2[5:8]  # arr_slice အဖြစ် ကူးယူသည်။
arr_slice
```

```
array([12, 12, 12])
```

arr_slice သည် array အသစ်တစ်ခု ဖြစ်သည်။

```
arr_slice[1] = 12345   # arr_slice မှ index နံပါတ် 1 ကို ပြောင်းသည်။
arr_slice
```

```
array([   12, 12345,    12])
```

```
arr_2       # မူလ array ဖြစ်သည့် arr_2 တွင်လည်း ပြောင်းသွားသည်ကို တွေ့ရမည်။
```

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

```
arr_slice[:] = 64  # arr_slice မှ အာလုံးကို တန်ဖိုး 64 အဖြစ် ပြောင်းသည်။
arr_2           # မူလ array ဖြစ်သည့် arr_2 တွင်လည်း ပြောင်းသွားသည်ကို တွေ့ရမည်။
```

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

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

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

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

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

```
arr2d[2]          # row 2 ကို ဖတ်သည်။  [ rows, columns]ပုံစံ ဖြစ်သည်။
```

```
array([7, 8, 9])
```

```
arr2d[0][2]          # row 0 နှင့် ကော်လံ 2 ကို ဖတ်သည်။
```

3

```
arr2d[0, 2]          # row 0 နှင့် ကော်လံ 2 ကို ဖတ်သည်။ row နှင့် ကော်လံ တွဲရေးသည်။
```

3

`d[:,:]` row အားလုံးနှင့် ကော်လံအားလုံး `d[:,1:4]` row အားလုံးနှင့် ကော်လံ ၁ မှ ၃

`d[:,1]` row အားလုံးနှင့် ကော်လံ ၁ တစ်ခုတည်းသာ

21.7 Boolean Indexing

np.array() ဖြင့် နာမည်များသာ ပါသည့် array တစ်ခု တည်ဆောက်သည်။

```
names=np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe'])
names
```

```
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe'], dtype='<U4')
```

Random ကိန်းများပါဝင်သည့် data ဆိုသည့် array တစ်ခု တည်ဆောက်သည်။

```
data = np.random.randn(6, 4)
data
```

```
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]])
```

names ဆိုသည့် array မှ 'Bob' ဖြစ်သည့် item များကို စစ်ယူသည်။ 'Bob' ဖြစ်လျှင် True ပေးသည်။

```
names == 'Bob'
```

```
array([ True, False, False,  True, False, False, False])
```

`data` array မှ True ဖြစ်သည့် row များကိုသာ ထုတ်ယူသည်။ True ဖြစ်သည့် row များကို `names == 'Bob'` မှ ပေးသည်။

```
data[names == 'Bob']
```

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

`names == 'Bob'` ဖြစ်သည့် row များထဲမှ item တစ်ခုချင်းစီကို ထုတ်ယူသည်။

```
data[names == 'Bob', 2:]
```

```
array([1.2464, 0.477 ])
```

```
data[names == 'Bob', 3]
```

```
array([1.2464, 0.477 ])
```

21.8 Transposing Arrays

```
arr = np.arange(15).reshape((3, 5))
arr
```

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

```
arr.T          # arr array ကို transpose လုပ်သည်။
```

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

21.9 Comparison Operators as Universal Functions (ufunc)

ufunc သည် Universal functions ၏ အတိုခေါက် ဖြစ်သည်။ **ndarrays** များကို array broadcasting, type casting စသည့် အလုပ်များ လုပ်နိုင်သည်။ ufunc သည် “vectorized” wrapper ဖြစ်သည်။

`np.array()` ဖြင့် [1, 2, 3, 4, 5] ဒေတာများထည့်ပြီး NumPy array တစ်ခုတည်ဆောက်သည်။

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

NumPy array ဖြစ်သည့် x ထဲမှာ (၃)ထက် ငယ်သည့် ကိန်းများကို True ဖြင့် ဖော်ပြခိုင်းသည်။ (၃)ထက် ငယ်လျှင် True ဖြစ်ပြီး ကြီးလျှင် False ဖြစ်သည်။

```
x < 3       # less than
```

```
array([ True,  True, False, False, False])
```

NumPy array ဖြစ်သည့် x ထဲမှာ (၃)ထက် ကြီးသည့် ကိန်းများကို ဖော်ပြခိုင်းသည်။ (၃)ထက် ကြီးလျှင် True ဖြစ်ပြီး ငယ်လျှင် False ဖြစ်သည်။

```
x > 3       # greater than
```

```
array([False, False, False,  True,  True])
```

(၃) နှင့် ညီပြီး (၃) ငယ်လျှင် True ဖြစ်ပြီး (၃)ထက် ကြီးလျှင် False ဖြစ်သည်။

```
x <= 3      # less than or equal
```

```
array([ True,  True,  True, False, False])
```

(၃)နှင့် ညီပြီး (၃)ထက် ကြီးလျှင် True ဖြစ်ပြီး (၃)ငယ်လျှင် False ဖြစ်သည်။

```
x >= 3      # greater than or equal
```

```
array([False, False,  True,  True,  True])
```

(၃)နှင့် မညီလျှင် True ဖြစ်သည်။

```
x != 3      # not equal
```

```
array([ True,  True, False,  True,  True])
```

(၃)နှင့် ညီလျှင် True ဖြစ်သည်။

```
x == 3       # equal
```

```
array([False, False,  True, False, False])
```

21.9.1 Element-wise Comparison of Two Arrays

NumPy array ဖြစ်သည့် x ကို (၂)နှင့် မြှောက်သည်။ x ကို နှစ်ထပ်ကိန်းတင်သည်။ ဖြစ်ပေါ်လာသည့် ထို array နှစ်ခု ရှိ element များကို အချင်းချင်းကို comparison လုပ်သည်။ တူညီလျှင် True ဖြစ်ပြီး မတူညီလျှင် False ဖြစ်သည်။

```
(2 * x) == (x ** 2)
```

```
array([False,  True, False, False, False])
```

Comparison operators and their equivalent ufunc

Operator	Equivalent ufunc	Operator	Equivalent ufunc
==	np.equal	!=	np.not_equal
<	np.less	<=	np.less_equal
>	np.greater	>=	np.greater_equal

Random NumPy array တစ်ခုတည်ဆောက်သည်။

```
rng = np.random.RandomState(1)
x = rng.randint(10, size=(3, 4))
x
```

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

x ထဲမှ element များ တစ်ခုချင်းစီကို (ဖ)ထက် ငယ် မငယ် စစ်သည်။ ရလဒ်ကို Boolean array အဖြစ် ပြန်ထုတ်ပေးသည်။

```
x < 6
```

```
array([[ True, False, False,  True],
       [ True,  True,  True, False],
       [False, False,  True,  True]])
```

21.9.2 Counting Entries

x ထဲမှ (ဖ)ထက် ငယ်သည့် element အရေအတွက်ကို ဖော်ပြရန် `np.count_nonzero()` ကို သုံးသည်။

```
# how many values less than 6?
np.count_nonzero(x < 6)
```

```
7
```

x ဆိုသည့် array ထဲတွင် (ဖ)ထက် ငယ်သည့် entry (၇)ခု ပါရှိသည်။ x ထဲမှ element များ (ဖ)ထက် ငယ်လျှင် True ဖြစ်သည်။ True ၏ တန်ဖိုးကို (၁)အဖြစ် ယူ၍ `np.sum()` ဖြင့်လည်း ဖော်ပြနိုင်သည်။

```
np.sum(x < 6)
```

```
7
```

Row တစ်ခုစီတွင် ရှိသည့် (ဖ) ထက်ငယ်သည် element ကိုလည်း ယခုကဲ့သို့ np.sum(x < 6, axis=1) ဖော်ပြနိုင်သည်။ ပထမ row တွင် (ဖ)ထက်ငယ်သည့် element ပထမ row တွင် (၂)ခု ၊ ဒုတိယ row တွင် (၃)ခု ၊ တတိယ row တွင် (၂)ခု စီ ရှိကြသည်။

```
# how many values less than 6 in each row?
np.sum(x < 6, axis=1)
```

```
array([2, 3, 2])
```

(၈) ထက်ကြီးသည့် element ရှိ မရှိ `np.any(x > 8)` ဖြင့် စစ်သည်။

```
np.any(x > 8)        # are there any values greater than 8?
```

```
True
```

သုည(၀) ထက် ငယ်သည့် element ရှိ မရှိ `np.any(x < 0)` ဖြင့် စစ်သည်။

```
np.any(x < 0)       # are there any values less than zero?
```

```
False
```

Element များ အားလုံးကို (၁၀)ထက် ငယ် မငယ် စစ်သည်။

```
np.all(x < 10)      # are all values less than 10?
```

```
True
```

Element များ အားလုံးကို (၆)နှင့် ညီ မညီ စစ်သည်။

```
np.all(x == 6)      # are all values equal to 6?
```

```
False
```

`np.all` နှင့် `np.any` တို့ ဖြင့် row အလိုက်, column အလိုက် စစ်နိုင်သည်။ Row တစ်ခုလုံးတွင် ရှိသည့် element အားလုံး (၈)ထက် ငယ် မငယ် စစ်သည်။

```
# are all values in each row less than 8?
np.all(x < 8, axis=1)              # axis=1 သည် row အတိုင်း ဖြစ်သည်။
```

```
array([ True, False,  True])
```

```
np.all(x < 8, axis=0)              # axis=0 သည် ကော်လံအတိုင်း ဖြစ်သည်။
```

```
array([ True, False, False,  True])
```

21.10 Numpy မှ Fancy Indexing

Array များကို indexing လုပ်သည့် နောက်စတိုင် တမျိုးမှာ **fancy indexing** ဖြစ်သည်။

Fancy indexing ဆိုသည်မှာ ပြီးခဲ့သည့် အခန်းတွင် ဖော်ပြပြီး ဖြစ်သည့် ရိုးရိုး indexing များကဲ့သို့ပင် ဖြစ်သည်။ သို့သော် ရိုးရိုး indexing တွင် ကိန်းတစ်ခု သို့မဟုတ် ဂဏန်းတစ်ခု(single scalar)ကိုသာ ထည့်ပြီး indexing လုပ်သည်။ Fancy indexing လုပ်သည့်အခါ တစ်ခုထက် ပိုများသည့် ကိန်းများကို array ပုံစံဖြင့် (arrays of indices) indexing လုပ်သည်။

ထို့သို့ ပြုလုပ်နိုင်ခြင်းကြောင့် လျှင်မြန်စွာ access လုပ်နိုင်သည်။ Array အကြီးကြီးထဲမှ array ငယ်များကို လျှင်မြန်လွယ်ကူစွာ ခွဲထုတ်နိုင်သည်။ လိုသလို ပြောင်းလဲနိုင်သည်။ (very quickly access and modify complicated subsets of an array's values.)

21.10.1 Fancy Indexing ဆိုသည်မှာ

Fancy indexing လုပ်ရန် တစ်ခုထက်ပိုများသည့် ကိန်းများကို access လုပ်ရန် index ကိန်းများကို array ပုံစံဖြင့် ထည့်ပေးခြင်း ဖြစ်သည်။ (passing an array of indices to access multiple array elements at once.)

ဥပမာဖြင့် ရှင်းပြရန် x ဟုသည့် array ကို random ကိန်းများဖြင့် တည်ဆောက်သည်။ RandomState method ကို random နံပါတ်များ ထုတ်ပေးရန် အသုံးပြုသည်။ (methods for generating random numbers drawn from a variety of probability distributions)

`rand.randint(100, size=10)` သည် (၁၀၀)ထက်နည်းသည့် random နံပါတ် (၁၀) ထုတ်ပေးရန် ဖြစ်သည်။

```
import numpy as np
rand = np.random.RandomState(42)

x = rand.randint(100, size=10)
```

```
print(x)
```

```
[51 92 14 71 60 20 82 86 74 74]
```

ဥပမာအဖြစ် အောက်တွင် index နံပါတ် 3(စတုတ္ထ နေရာမှကိန်း) ၊ index နံပါတ် 7(အဋ္ဌမ နေရာမှကိန်း)နှင့် index နံပါတ် 2 (တတိယနေရာမှ ကိန်း)တို့ကို access လုပ်ပုံကို နည်းသုံးမျိုးဖြင့် ဖော်ပြထားသည်။

x[3] = စတုတ္ထနေရာမှ ကိန်း , x[7] = အဋ္ဌမနေရာမှကိန်း၊ x[2] = တတိယနေရာမှ ကိန်း

```
[x[3], x[7], x[2]]
```

```
[71, 86, 14]
```

Indexing လုပ်သည့်အခါ ကိန်းတစ်ခုချင်းစီ မထည့်ပေးဘဲ (pass မလုပ်ဘဲ) ကိန်းသုံးခုစလုံးတွက် index များကို list တစ်ခု သို့မဟုတ် array တစ်ခု ပုံစံမျိုးဖြင့် ထည့်ပေးပြီး access လုပ်နိုင်သည်။ (pass a single list or array of indices to obtain the same result)

`ind = [3, 7, 4]` သည် access လုပ်လိုသည့် နေရာ(index)များကို ပေါင်း၍ list တစ်ခု ပြုလုပ်သည်။ ထို `ind` ဆိုသည့် list ကို သုံး၍ array x ကို access လုပ်သည်။

```
ind = [3, 7, 4]                                  # index array
x[ind]
```

```
array([71, 86, 60])
```

Fancy indexing ပြုလုပ်သည့်အခါ ရလဒ်(result)အဖြစ် ပြန်ပေးသည့် array ၏ shape သည် index array ၏ shape အတိုင်း ဖြစ်သည်။ index လုပ်ခံရသည့် နှင့် မူလ array ၏ shape အတိုင်း ပြန်မပေးပါ။ (the shape of the result reflects the shape of the index arrays rather than the shape of the array being indexed)။ တစ်နည်းအားဖြင့် ထွက်လာသည့် array ၏ shape သည် index array အတိုင်း ဖြစ်သည်။

```
ind = np.array([[3, 7],                           # index array
                [4, 5]])
x[ind]
```

```
array([[71, 86],
       [60, 20]])
```

Dimension များစွာ ပါသည့် array များအတွက်လည်း fancy indexing ကို အသုံးပြုနိုင်သည်။ (Fancy indexing also works in multiple dimensions.)

```
X = np.arange(12)
X
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

`np.arange(12)` သည် သုည(၀) မှ (၁၁) အထိ ကိန်း (၁၂) လုံးထုတ်ပေးရန် ဖြစ်သည်။ ထိုအစဉ်လိုက် ကိန်း (၁၂)လုံးကို `.reshape((3, 4))` ဖြင့် 3 x 4 array ဖြစ်အောင် ပြောင်းသည်။

```
X = np.arange(12).reshape((3, 4))
X
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

အထက်ပါ array X ကို standard indexing ဖြစ်သည့် row, col ဖြင့် access လုပ်သည်။ `X[row, col]` row index များကို `row = np.array([0, 1, 2])` ဖြင့် row array တစ်ခု ဖြစ်အောင် ပြုလုပ်သည်။ ကော်လံ index များကို `col = np.array([2, 1, 3])` ဖြင့် col array တစ်ခု ဖြစ်အောင် ပြုလုပ်သည်။

```
row = np.array([0, 1, 2])
row
```

```
array([0, 1, 2])
```

```
col = np.array([2, 1, 3])
col
```

```
array([2, 1, 3])
```

```
X[row, col]                    # X[0, 2], X[1, 1], X[2, 3]
```

```
array([ 2,  5, 11])
```

2 သည် `X[0, 2]` ဖြစ်ပြီး , 5 သည် `X[1, 1]` နှင့် , 11 သည် `X[2, 3]` ဖြစ်သည်။

Fancy indexing တွင် index များကို တွဲခြင်း(pairing) လုပ်သည့်အခါ **broadcasting rules** ကို လိုက်နာသည်။ ဥပမာ index များ၏ column vector တစ်ခုနှင့် row vector တစ်ခုကို ပေါင်းသည့်အခါ two-dimensional index array တစ်ခု ရသည်။

```
X[row[:, np.newaxis], col]
```

```
array([[ 2,  1,  3],
       [ 6,  5,  7],
       [10,  9, 11]])
```

row value တစ်ခုချင်းစီဖြင့် column vector တစ်ခုချင်းစီကို match လုပ်ခြင်းသည် broadcasting of arithmetic operations ဖြစ်သည်။

```
row[:, np.newaxis] * col
```

```
array([[0, 0, 0],
       [2, 1, 3],
       [4, 2, 6]])
```

Fancy indexing တွင် return value သည် broadcast လုပ်သည့် indices ၏ shape အတိုင်း ဖြစ်သည်။ Index လုပ်ခံရသည့် array ၏ shape အတိုင်း ဖြစ်လိမ့်မည် မဟုတ်ပေ။

21.10.2 Combined Indexing

Fancy indexing ကို တခြားသော indexing scheme များဖြင့် တွဲသုံးပါက ပိုအစွမ်းထက်သည်။ (more powerful operations)

```
print(X)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

Fancy indexing နှင့် simple indexing တို့ တွဲသုံးပုံကို အောက်တွင် ဥပမာဖြင့် ပြထားသည်။ `[2, [2, 0, 1]]` သည် တတိယ row နှင့် col index array ဖြစ်သည်။ 2,2 2,0 နှင့် 2,1 တို့ဖြစ်သည်။

```
X[2, [2, 0, 1]]      # row 2 မှ element များကို indexing လုပ်ပြီး ပြန်စီသည်။
```

```
array([10,  8,  9])
```

Fancy indexing ကို slicing လုပ်ခြင်းနှင့် လည်း တွဲ၍ အသုံးပြုနိုင်သည်။

```
X[1:, [2, 0, 1]]
```

```
array([[ 6,  4,  5],
       [10,  8,  9]])
```

Fancy indexing ကို masking နှင့်လည်း တွဲ၍ အသုံးပြုနိုင်သည်။

```
mask = np.array([1, 0, 1, 0], dtype=bool)
X[row[:, np.newaxis], mask]
```

```
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10]])
```

Array တစ်ခုအတွင်းရှိ value များအား access လုပ်သည့်အခါ၊ modify လုပ်သည့်အခါများတွင် fancy indexing ကို တခြားသော နည်းများဖြင့် တွဲ၍ အသုံးပြုခြင်းကြောင့် ပို၍ လွယ်ကူ လျှင်မြန်သည်။

21.11 NumPy Standard Data Types

NumPy array များထဲတွင် အမျိုးအစားတူညီသည့် ဒေတာများသာ ရှိနိုင်သည့်အတွက် ဒေတာ အမျိုးအစားနှင့် ကန့်သတ်ချက်များ(types and their limitations)ကို သိထားသင့်သည်။ NumPy သည် built in C programming ဖြစ်သောကြောင့် C နှင့် Fortran ကို လေ့လာထားသူများ data type များနှင့် ရင်းနှီး ကြလိမ့်မည်။

Array များကို တည်ဆောက်(construct)သည်အခါ အသုံးပြုရမည့် Standard NumPy data type များကို အောက်တွင် ဇယားဖြင့် ဖော်ပြထားသည်။

Data type	Description
`bool_`	Boolean (True or False) stored as a byte
`int_`	Default integer type (same as C long; normally either int64 or int32)
`intc`	Identical to C int (normally int32 or int64)
`intp`	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
`int8`	Byte (-128 to 127)
`int16`	Integer (-32768 to 32767)
`int32`	Integer (-2147483648 to 2147483647)
`int64`	Integer (-9223372036854775808 to 9223372036854775807)
`uint8`	Unsigned integer (0 to 255)
`uint16`	Unsigned integer (0 to 65535)
`uint32`	Unsigned integer (0 to 4294967295)
`uint64`	Unsigned integer (0 to 18446744073709551615)
`float_`	Shorthand for float64.
`float16`	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
`float32`	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
`float64`	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
`complex_`	Shorthand for complex128.
`complex64`	Complex number, represented by two 32-bit floats
`complex128`	Complex number, represented by two 64-bit floats

Reference: https://numpy.org/doc/stable/reference/

Python Data Science Handbook by Jake VanderPlas

Array Conversion

ndarray.item(*args)	Copy an element of an array to a standard Python scalar and return it.
ndarray.tolist()	Return the array as an a.ndim-levels deep nested list of Python scalars.
ndarray.itemset(*args)	Insert scalar into an array (scalar is cast to array's dtype, if possible)
ndarray.tostring([order])	A compatibility alias for tobytes, with exactly the same behavior.
ndarray.tobytes([order])	Construct Python bytes containing the raw data bytes in the array.
ndarray.tofile(fid[, sep, format])	Write array to a file as text or binary (default).
ndarray.dump(file)	Dump a pickle of the array to the specified file.
ndarray.dumps()	Returns the pickle of the array as a string.
ndarray.astype(dtype[, order, casting, …])	Copy of the array, cast to a specified type.
ndarray.byteswap([inplace])	Swap the bytes of the array elements
ndarray.copy([order])	Return a copy of the array.
ndarray.view([dtype][, type])	New view of array with the same data.
ndarray.getfield(dtype[, offset])	Returns a field of the given array as a certain type.
ndarray.setflags([write, align, uic])	Set array flags WRITEABLE, ALIGNED, (WRITEBACKIFCOPY and UPDATEIFCOPY), respectively.
ndarray.fill(value)	Fill the array with a scalar value.

Array Methods

ndarray.max([axis, out, keepdims, initial, …])	Return the maximum along a given axis.
ndarray.argmax([axis, out])	Return indices of the maximum values along the given axis.
ndarray.min([axis, out, keepdims, initial, …])	Return the minimum along a given axis.
ndarray.argmin([axis, out])	Return indices of the minimum values along the given axis of *a*.

ndarray.ptp([axis, out, keepdims])	Peak to peak (maximum - minimum) value along a given axis.
ndarray.clip([min, max, out])	Return an array whose values are limited to `[min, max]`.
ndarray.conj()	Complex-conjugate all elements.
ndarray.round([decimals, out])	Return *a* with each element rounded to the given number of decimals.
ndarray.trace([offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.
ndarray.sum([axis, dtype, out, keepdims, ...])	Return the sum of the array elements over the given axis.
ndarray.cumsum([axis, dtype, out])	Return the cumulative sum of the elements along the given axis.
ndarray.mean([axis, dtype, out, keepdims])	Returns the average of the array elements along given axis.
ndarray.var([axis, dtype, out, ddof, keepdims])	Returns the variance of the array elements, along given axis.
ndarray.std([axis, dtype, out, ddof, keepdims])	Returns the standard deviation of the array elements along given axis.
ndarray.prod([axis, dtype, out, keepdims, ...])	Return the product of the array elements over the given axis
ndarray.cumprod([axis, dtype, out])	Return the cumulative product of the elements along the given axis.
ndarray.all([axis, out, keepdims])	Returns True if all elements evaluate to True.
ndarray.any([axis, out, keepdims])	Returns True if any of the elements of *a* evaluate to True.

-**End**-