Astroinformatics 2022
Session 03: Using Numpy

Kinoshita Daisuke

26 September 2022
publicly accessible version

About this file...

- Important information about this file
  - The author of this file is Kinoshita Daisuke.
  - The original version of this file was used for the course "Astroinformatics" (course ID: AS6095) offered at Institute of Astronomy, National Central University from September 2022 to January 2023.
  - The file is provided in the hope that it will be useful, but there is no guarantee for the correctness. Use this file at your own risk.
  - If you are willing to use this file for your study, please feel free to use. I'll be very happy to receive feedback from you.
  - If you are willing to use this file for your teaching, please contact to Kinoshita Daisuke. When you use this file partly or entirely, please mention clearly that the author of the original version is Kinoshita Daisuke. Please help me to improve the contents of this file by sending your feedback.
  - Contact address: `https://www.instagram.com/daisuke23888/`

For this session, we try useful Python modules.

1 Sample Python scripts for this session

Sample Python scripts for this session can be downloaded from GitHub repository. Visit following GitHub repository.

- `https://github.com/kinoshitadaisuke/ncu_astroinformatics_202209`

1.1 Executing sample Python scripts on a terminal emulator

If you prefer to execute sample Python scripts for this session on a terminal emulator, download `.py` files from GitHub repository.

1.2 Executing sample Python scripts on JupyterLab

If you prefer to execute sample Python scripts for this session on JupyterLab (or Jupyter Notebook), download `.ipynb` file from GitHub repository.

1.3 Executing sample Python scripts using Binder

If you prefer to execute sample Python scripts for this session on Binder, visit following web page.

- `https://mybinder.org/v2/gh/kinoshitadaisuke/ncu_astroinformatics_202209/HEAD`

Start your favourite web browser and go to above web page. (Fig. 1) In a minute or two, you see JupyterLab working on your web browser. (Fig. 2) Go to the directory (folder) "`s03`". (Fig. 3) Choose the file "`ai202209_s03.ipynb`" (Fig. 4 and 5) and open it (Fig. 6).

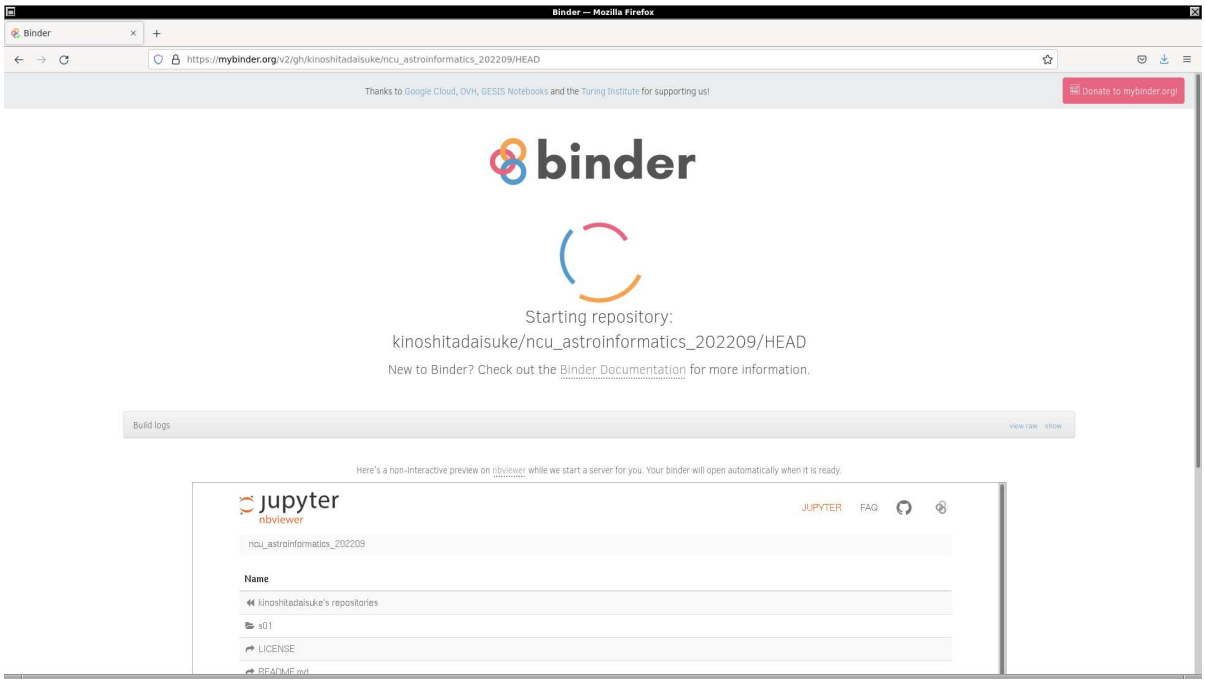

Figure 1: Using Binder to execute sample Python scripts for this session.

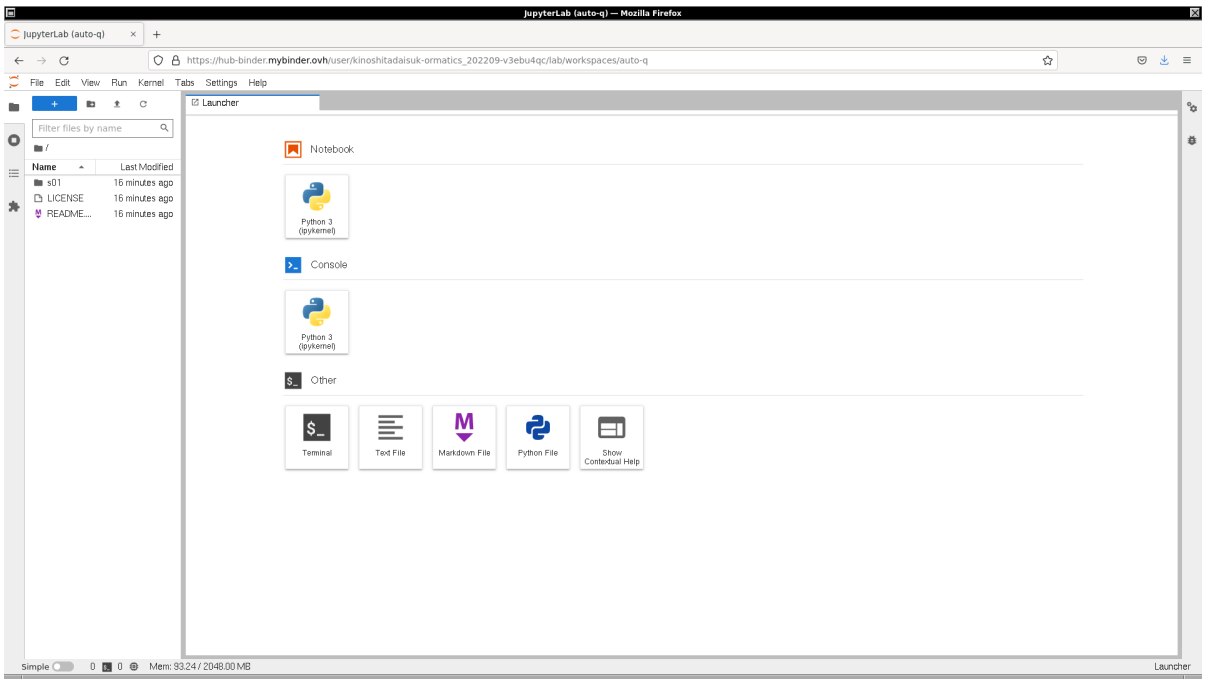

Figure 2: Using Binder to execute sample Python scripts for this session.

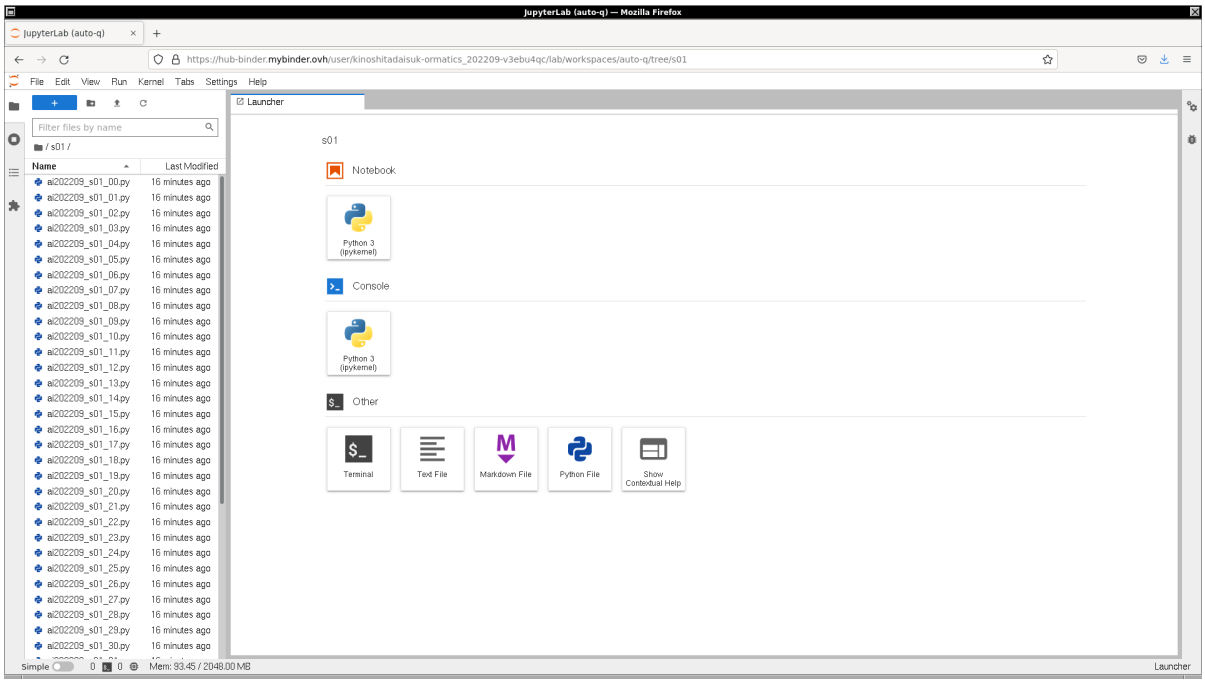

Figure 3: Using Binder to execute sample Python scripts for this session.

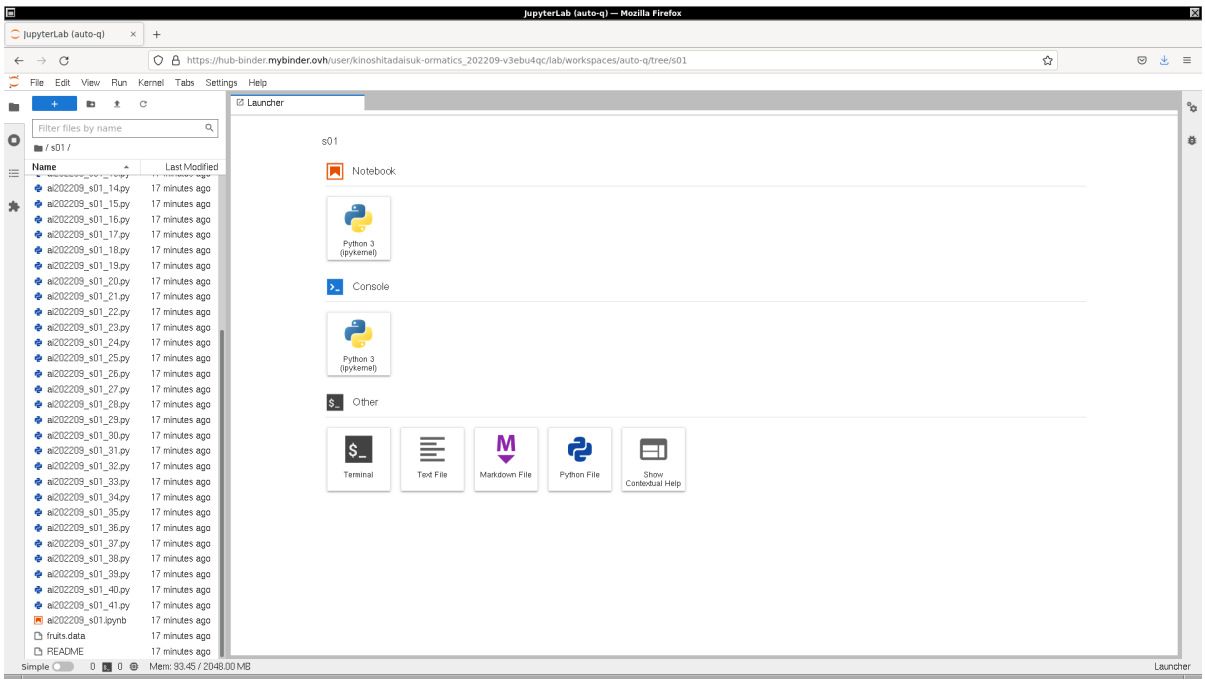

Figure 4: Using Binder to execute sample Python scripts for this session.

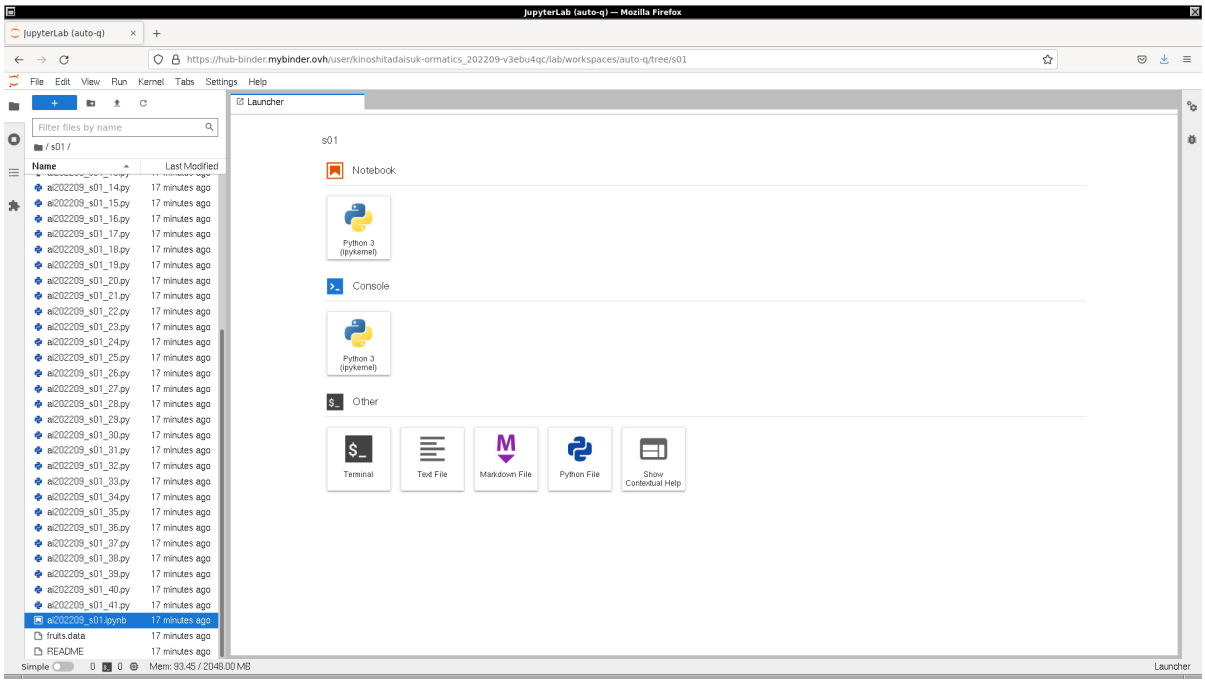

Figure 5: Using Binder to execute sample Python scripts for this session.

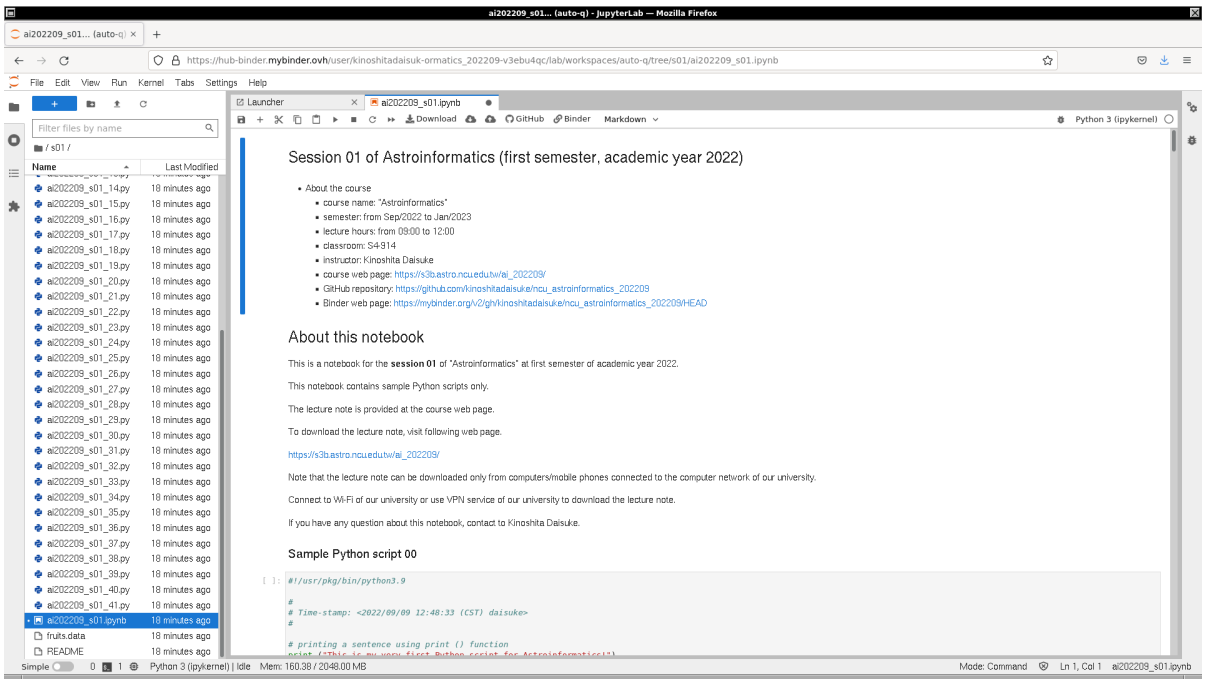

Figure 6: Using Binder to execute sample Python scripts for this session.

2 About Numpy

Numpy is a Python package to provide Numpy arrays (ndarrays) for fast numerical calculations. Numpy also provide number of functions useful for calculations. Numpy is also used by many other external Python packages, such as SciPy and Astropy. Numpy is a powerful tool for your astronomical calculations and data analysis. Visit the official website of Numpy to learn about it. (Fig. 7, 8, and 9)

- Numpy: `https://numpy.org/`
  - Numpy Documentation: `https://numpy.org/doc/stable/`

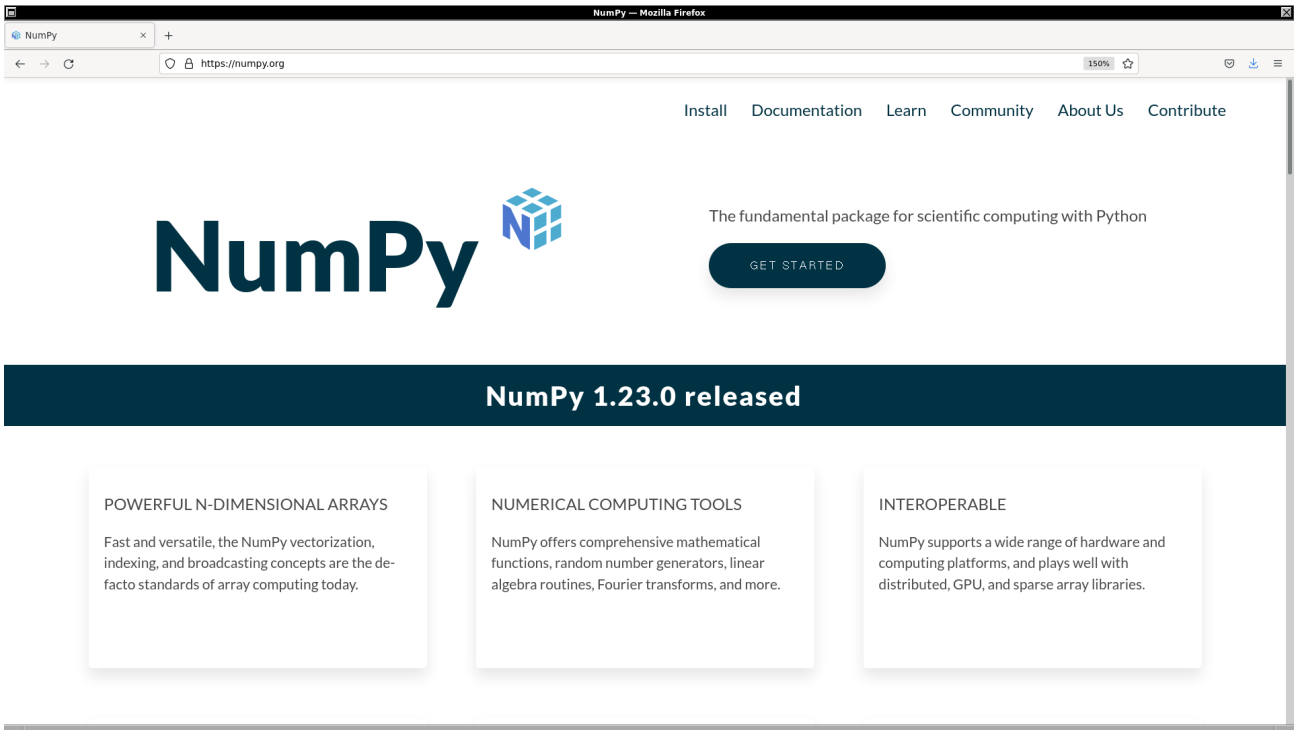

Figure 7: The official website of Numpy.

3 Numpy arrays

Numpy provides Numpy arrays (ndarrays) for efficient calculations.

3.1 Making a Numpy array using `numpy.array ()`

Make a Numpy array. Here is an example.

Python Code 1: ai202209_s03_00.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 18:02:00 (CST) daisuke>
#

# importing numpy
import numpy

# making a Numpy array (ndarray)
array1 = numpy.array ([0, 1, 2, 3, 4])
```

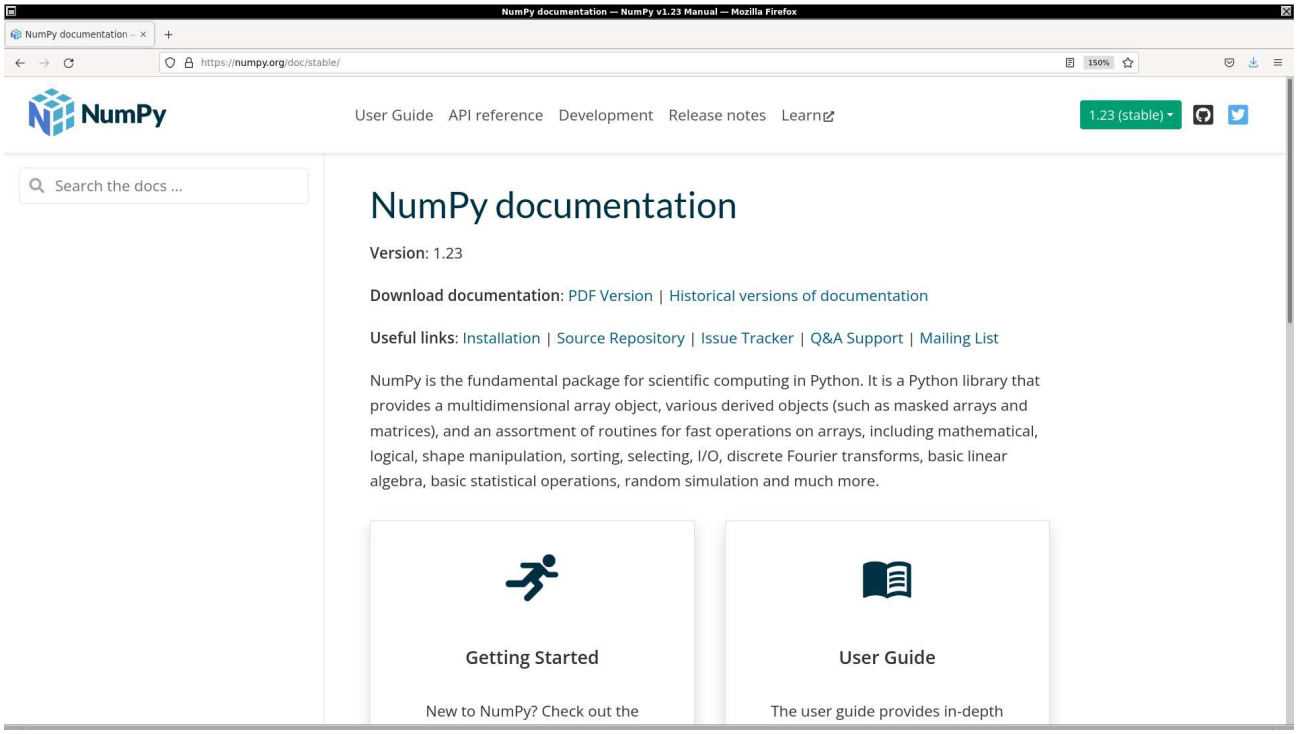

Figure 8: The official web page of Numpy Documentation.

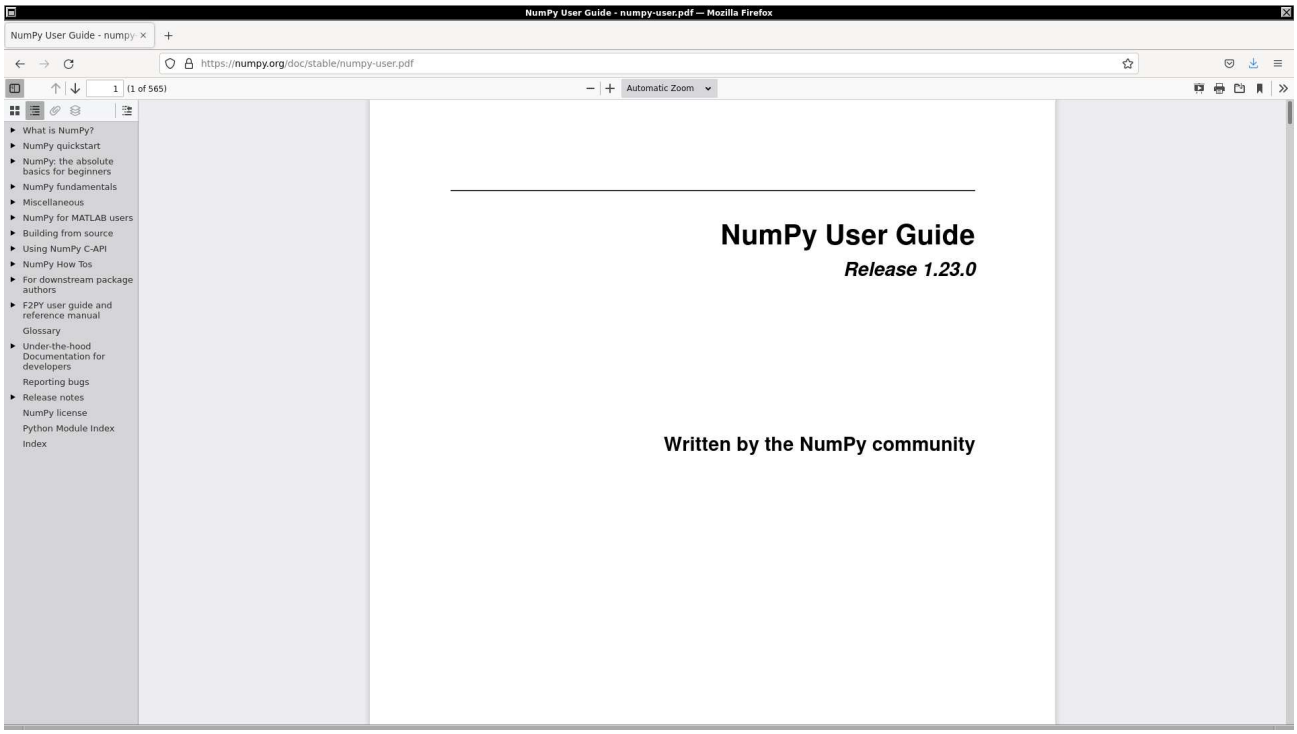

Figure 9: PDF version of Numpy's users guide.

```
# printing Numpy array
print (f'array1 = {array1}')

# type of "array1"
type_array1 = type (array1)

# printing type of "array1"
print (f'type of "array1" = {type_array1}')
```

Execute above script.

```
% ./ai202209_s03_00.py
array1 = [0 1 2 3 4]
type of "array1" = <class 'numpy.ndarray'>
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array.

Make a Python's list first, and then make a Numpy array. Here is an example.

Python Code 2: ai202209_s03_01.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 18:05:27 (CST) daisuke>
#

# importing numpy
import numpy

# making a list
list2 = [0, 1, 2, 3, 4]

# making a Numpy array (ndarray)
array2 = numpy.array (list2)

# printing list
print (f'list2  = {list2}')

# printing Numpy array
print (f'array2 = {array2}')

# type of "list2"
type_list2 = type (list2)

# type of "array2"
type_array2 = type (array2)

# printing type of "list2"
print (f'type of "list2"  = {type_list2}')

# printing type of "array2"
print (f'type of "array2" = {type_array2}')
```

Execute above script.

```
% ./ai202209_s03_01.py
list2  = [0, 1, 2, 3, 4]
array2 = [0 1 2 3 4]
type of "list2"  = <class 'list'>
type of "array2" = <class 'numpy.ndarray'>
```

Try following practice.

Practice 03-01

Make your own Python script to make a Python's list first and then make a Numpy array.

3.2 Printing information about Numpy array

Make a Python script to obtain information of a Numpy array, and then print it. Here is an example.

Python Code 3: ai202209_s03_02.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 18:06:05 (CST) daisuke>
#

# importing numpy
import numpy

# making a Numpy array (ndarray)
array3 = numpy.array ([0, 1, 2, 3, 4])

# printing Numpy array
print (f'array3 = {array3}')

# type of "array3"
type_array3 = type (array3)

# printing type of "array3"
print (f'type of "array3" = {type_array3}')

# dimension of "array3"
ndim_array3 = array3.ndim

# size of "array3"
size_array3 = array3.size

# shape of "array3"
shape_array3 = array3.shape

# data type of elements in "array3"
dtype_array3 = array3.dtype

# size of one element in "array3"
itemsize_array3 = array3.itemsize

# printing information
print (f'information of "array3":')
print (f'  ndim     = {ndim_array3}')
print (f'  size     = {size_array3}')
print (f'  shape    = {shape_array3}')
```

```
print (f'  dtype    = {dtype_array3}')
print (f'  itemsize = {itemsize_array3} byte')
```

Execute above script.

```
% ./ai202209_s03_02.py
array3 = [0 1 2 3 4]
type of "array3" = <class 'numpy.ndarray'>
information of "array3":
  ndim     = 1
  size     = 5
  shape    = (5,)
  dtype    = int64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array and print information about it.

Make a 2-dimensional Numpy array, and print information about it. Here is an example.

Python Code 4: ai202209_s03_03.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 18:09:35 (CST) daisuke>
#

# importing numpy
import numpy

# making a Numpy array (ndarray)
array4 = numpy.array ([ [0.0, 1.0, 2.0, 3.0, 4.0], \
                        [5.0, 6.0, 7.0, 8.0, 9.0], \
                        [10.0, 11.0, 12.0, 13.0, 14.0] ])

# printing Numpy array
print (f'array4 = {array4}')

# type of "array4"
type_array4 = type (array4)

# printing type of "array4"
print (f'type of "array4" = {type_array4}')

# dimension of "array4"
ndim_array4 = array4.ndim

# size of "array4"
size_array4 = array4.size

# shape of "array4"
shape_array4 = array4.shape

# data type of elements in "array4"
dtype_array4 = array4.dtype
```

```
# size of one element in "array4"
itemsize_array4 = array4.itemsize

# printing information
print (f'information of "array4":')
print (f'  ndim     = {ndim_array4}')
print (f'  size     = {size_array4}')
print (f'  shape    = {shape_array4}')
print (f'  dtype    = {dtype_array4}')
print (f'  itemsize = {itemsize_array4} byte')
```

Execute above script.

```
% ./ai202209_s03_03.py
array4 = [[ 0.  1.  2.  3.  4.]
 [ 5.  6.  7.  8.  9.]
 [10. 11. 12. 13. 14.]]
type of "array4" = <class 'numpy.ndarray'>
information of "array4":
  ndim     = 2
  size     = 15
  shape    = (3, 5)
  dtype    = float64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a 2-dim. Numpy array and print information about it.

Make a 3-dimensional Numpy array, and print information about it. Here is an example.

Python Code 5: ai202209_s03_04.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 18:23:21 (CST) daisuke>
#

# importing numpy
import numpy

# making a Numpy array (ndarray)
array5 = numpy.array ([
    [
        [0.0, 1.0, 2.0], \
        [3.0, 4.0, 5.0], \
        [6.0, 7.0, 8.0]
    ],
    [
        [10.0, 11.0, 12.0], \
        [13.0, 14.0, 15.0], \
        [16.0, 17.0, 18.0]
    ]
])

# printing Numpy array
print (f'array5 = {array5}')
```

```
# type of "array5"
type_array5 = type (array5)

# printing type of "array5"
print (f'type of "array5" = {type_array5}')

# dimension of "array5"
ndim_array5 = array5.ndim

# size of "array5"
size_array5 = array5.size

# shape of "array5"
shape_array5 = array5.shape

# data type of elements in "array5"
dtype_array5 = array5.dtype

# size of one element in "array5"
itemsize_array5 = array5.itemsize

# printing information
print (f'information of "array5":')
print (f'  ndim     = {ndim_array5}')
print (f'  size     = {size_array5}')
print (f'  shape    = {shape_array5}')
print (f'  dtype    = {dtype_array5}')
print (f'  itemsize = {itemsize_array5} byte')
```

Execute above script.

```
% ./ai202209_s03_04.py
array5 = [[[ 0.  1.  2.]
  [ 3.  4.  5.]
  [ 6.  7.  8.]]

 [[10. 11. 12.]
  [13. 14. 15.]
  [16. 17. 18.]]]
type of "array5" = <class 'numpy.ndarray'>
information of "array5":
  ndim     = 3
  size     = 18
  shape    = (2, 3, 3)
  dtype    = float64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a 3-dim. Numpy array and print information about it.

3.3 Making a Numpy array with specified data type

Make a Numpy array with elements of 8-bit unsigned integer data. Here is an example.

Python Code 6: ai202209_s03_05.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 18:35:00 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with a specified data type
array6 = numpy.array ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], \
                      dtype=numpy.dtype ('u1') )

# printing Numpy array
print (f'array6:\n{array6}')

# printing information
print (f'information:')
print (f'  ndim     = {array6.ndim}')
print (f'  size     = {array6.size}')
print (f'  shape    = {array6.shape}')
print (f'  dtype    = {array6.dtype}')
print (f'  itemsize = {array6.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_05.py
array6:
[ 0  1  2  3  4  5  6  7  8  9 10]
information:
  ndim     = 1
  size     = 11
  shape    = (11,)
  dtype    = uint8
  itemsize = 1 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of 16-bit unsigned integer.

Make a Numpy array with elements of 16-bit signed integer data. Here is an example.

Python Code 7: ai202209_s03_06.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 18:55:14 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with a specified data type
array7 = numpy.array ([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5], \
                      dtype=numpy.dtype ('i2') )
```

```
# printing Numpy array
print (f'array7:\n{array7}')

# printing information
print (f'information:')
print (f'  ndim     = {array7.ndim}')
print (f'  size     = {array7.size}')
print (f'  shape    = {array7.shape}')
print (f'  dtype    = {array7.dtype}')
print (f'  itemsize = {array7.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_06.py
array7:
[-5 -4 -3 -2 -1  0  1  2  3  4  5]
information:
  ndim     = 1
  size     = 11
  shape    = (11,)
  dtype    = int16
  itemsize = 2 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of 32-bit signed integer.

Here is the other way to make a Numpy array with elements of 16-bit signed integer data.

Python Code 8: ai202209_s03_07.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 19:08:47 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with a specified data type
array8 = numpy.array ([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5], \
                      dtype=numpy.int16)

# printing Numpy array
print (f'array8:\n{array8}')

# printing information
print (f'information:')
print (f'  ndim     = {array8.ndim}')
print (f'  size     = {array8.size}')
print (f'  shape    = {array8.shape}')
print (f'  dtype    = {array8.dtype}')
print (f'  itemsize = {array8.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_07.py
array8:
[-5 -4 -3 -2 -1  0  1  2  3  4  5]
information:
  ndim     = 1
  size     = 11
  shape    = (11,)
  dtype    = int16
  itemsize = 2 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of 64-bit signed integer.

Make a Numpy array with elements of 32-bit floating point number data. Here is an example.

Python Code 9: ai202209_s03_08.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 19:09:55 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with a specified data type
array9 = numpy.array ([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5], \
                      dtype=numpy.dtype ('f4') )

# printing Numpy array
print (f'array9:\n{array9}')

# printing information
print (f'information:')
print (f'  ndim     = {array9.ndim}')
print (f'  size     = {array9.size}')
print (f'  shape    = {array9.shape}')
print (f'  dtype    = {array9.dtype}')
print (f'  itemsize = {array9.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_08.py
array9:
[-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.]
information:
  ndim     = 1
  size     = 11
  shape    = (11,)
  dtype    = float32
  itemsize = 4 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of 64-bit floating point number.

Here is the other way to make a Numpy array with elements of 32-bit floating point number data.

Python Code 10: ai202209_s03_09.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 19:14:39 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with a specified data type
array10 = numpy.array ([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5], \
                       dtype=numpy.float32)

# printing Numpy array
print (f'array10:\n{array10}')

# printing information
print (f'information:')
print (f'  ndim     = {array10.ndim}')
print (f'  size     = {array10.size}')
print (f'  shape    = {array10.shape}')
print (f'  dtype    = {array10.dtype}')
print (f'  itemsize = {array10.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_09.py
array10:
[-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.]
information:
  ndim     = 1
  size     = 11
  shape    = (11,)
  dtype    = float32
  itemsize = 4 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of 128-bit floating point number.

Here is one more way to make a Numpy array with elements of 32-bit floating point number data.

Python Code 11: ai202209_s03_10.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 19:15:05 (CST) daisuke>
#

# importing numpy module
```

```
import numpy

# making a Numpy array (ndarray) with a specified data type
array11 = numpy.array ([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5], \
                       dtype='float32')

# printing Numpy array
print (f'array11:\n{array11}')

# printing information
print (f'information:')
print (f'  ndim     = {array11.ndim}')
print (f'  size     = {array11.size}')
print (f'  shape    = {array11.shape}')
print (f'  dtype    = {array11.dtype}')
print (f'  itemsize = {array11.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_10.py
array11:
[-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.]
information:
  ndim     = 1
  size     = 11
  shape    = (11,)
  dtype    = float32
  itemsize = 4 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of 16-bit floating point number.

Make a Numpy array with elements of complex numbers consist of two 32-bit floating point numbers. Here is an example.

Python Code 12: ai202209_s03_11.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 19:27:58 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with a specified data type
array12 = numpy.array ([1.0 + 2.0j, 3.0j, 4.0, 5.0 - 6.0j, -7.0 + 8.0j, \
                        -9.0 - 10.0j, -11.0j, -12.0, 13.0 + 14.0j, 15.0j], \
                       dtype=numpy.dtype ('c8'))

# printing Numpy array
print (f'array12:\n{array12}')

# printing information
print (f'information:')
print (f'  ndim     = {array12.ndim}')
```

```
print (f'  size     = {array12.size}')
print (f'  shape    = {array12.shape}')
print (f'  dtype    = {array12.dtype}')
print (f'  itemsize = {array12.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_11.py
array12:
[  1. +2.j   0. +3.j   4. +0.j   5. -6.j  -7. +8.j  -9.-10.j  -0.-11.j
 -12. +0.j  13.+14.j   0.+15.j]
information:
  ndim     = 1
  size     = 10
  shape    = (10,)
  dtype    = complex64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of complex numbers consist of two 64-bit floating point numbers.

Here is the other way to make a Numpy array with elements of complex numbers consist of two 32-bit floating point numbers.

Python Code 13: ai202209_s03_12.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 19:33:17 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with a specified data type
array13 = numpy.array ([1.0 + 2.0j, 3.0j, 4.0, 5.0 - 6.0j, -7.0 + 8.0j, \
                        -9.0 - 10.0j, -11.0j, -12.0, 13.0 + 14.0j, 15.0j], \
                       dtype=numpy.complex64)

# printing Numpy array
print (f'array13:\n{array13}')

# printing information
print (f'information:')
print (f'  ndim     = {array13.ndim}')
print (f'  size     = {array13.size}')
print (f'  shape    = {array13.shape}')
print (f'  dtype    = {array13.dtype}')
print (f'  itemsize = {array13.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_12.py
array13:
[  1. +2.j   0. +3.j   4. +0.j   5. -6.j  -7. +8.j  -9.-10.j  -0.-11.j
```

```
 -12. +0.j   13.+14.j    0.+15.j]
information:
  ndim     = 1
  size     = 10
  shape    = (10,)
  dtype    = complex64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of complex numbers consist of two 64-bit floating point numbers.

Here is one more way to make a Numpy array with elements of complex numbers consist of two 32-bit floating point numbers.

Python Code 14: ai202209_s03_13.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 19:35:16 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with a specified data type
array14 = numpy.array ([1.0 + 2.0j, 3.0j, 4.0, 5.0 - 6.0j, -7.0 + 8.0j, \
                        -9.0 - 10.0j, -11.0j, -12.0, 13.0 + 14.0j, 15.0j], \
                       dtype='complex64')

# printing Numpy array
print (f'array14:\n{array14}')

# printing information
print (f'information:')
print (f'  ndim     = {array14.ndim}')
print (f'  size     = {array14.size}')
print (f'  shape    = {array14.shape}')
print (f'  dtype    = {array14.dtype}')
print (f'  itemsize = {array14.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_13.py
array14:
[  1. +2.j   0. +3.j   4. +0.j   5. -6.j  -7. +8.j  -9.-10.j  -0.-11.j
 -12. +0.j  13.+14.j   0.+15.j]
information:
  ndim     = 1
  size     = 10
  shape    = (10,)
  dtype    = complex64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of complex numbers consist of two 64-bit floating point numbers.

Make a Numpy array with elements of strings. Here is an example.

Python Code 15: ai202209_s03_14.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 19:40:30 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with a specified data type
array15 = numpy.array (['Ceres', 'Pallas', 'Juno', 'Vesta', 'Astraea', \
                        'Hebe', 'Iris', 'Flora', 'Metis', 'Hygiea'], \
                       dtype=numpy.dtype ('U10') )

# printing Numpy array
print (f'array15:\n{array15}')

# printing information
print (f'information:')
print (f'  ndim     = {array15.ndim}')
print (f'  size     = {array15.size}')
print (f'  shape    = {array15.shape}')
print (f'  dtype    = {array15.dtype}')
print (f'  itemsize = {array15.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_14.py
array15:
['Ceres' 'Pallas' 'Juno' 'Vesta' 'Astraea' 'Hebe' 'Iris' 'Flora' 'Metis'
 'Hygiea']
information:
  ndim     = 1
  size     = 10
  shape    = (10,)
  dtype    = <U10
  itemsize = 40 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array of strings.

3.4 Making a Numpy array using `numpy.zeros ()`

A Numpy array initialised by zeros can be created using the function `numpy.zeros ()`. Here is an example.

Python Code 16: ai202209_s03_15.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 20:03:44 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) with 10 elements all equal to zeros
array16 = numpy.zeros ( (10,) )

# printing Numpy array
print (f'array16:\n{array16}')

# printing information
print (f'information:')
print (f'  ndim     = {array16.ndim}')
print (f'  size     = {array16.size}')
print (f'  shape    = {array16.shape}')
print (f'  dtype    = {array16.dtype}')
print (f'  itemsize = {array16.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_15.py
array16:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
information:
  ndim     = 1
  size     = 10
  shape    = (10,)
  dtype    = float64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array initialised by zeros.

A 2-dimensional Numpy array initialised by zeros can be created using the function `numpy.zeros ()`. Here is an example.

Python Code 17: ai202209_s03_16.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 20:04:49 (CST) daisuke>
#

# importing numpy module
import numpy

# making a 2-dim Numpy array (ndarray) with elements all equal to zeros
array17 = numpy.zeros ( (3, 3) )
```

```
# printing Numpy array
print (f'array17:\n{array17}')

# printing information
print (f'information:')
print (f'  ndim     = {array17.ndim}')
print (f'  size     = {array17.size}')
print (f'  shape    = {array17.shape}')
print (f'  dtype    = {array17.dtype}')
print (f'  itemsize = {array17.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_16.py
array17:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
information:
  ndim     = 2
  size     = 9
  shape    = (3, 3)
  dtype    = float64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a 2-dim. Numpy array initialised by zeros.

A 3-dimensional Numpy array initialised by zeros can be created using the function `numpy.zeros ()`. Here is an example.

Python Code 18: ai202209_s03_17.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 20:07:07 (CST) daisuke>
#

# importing numpy module
import numpy

# making a 3-dim Numpy array (ndarray) with elements all equal to zeros
array18 = numpy.zeros ( (3, 3, 3) )

# printing Numpy array
print (f'array18:\n{array18}')

# printing information
print (f'information:')
print (f'  ndim     = {array18.ndim}')
print (f'  size     = {array18.size}')
print (f'  shape    = {array18.shape}')
print (f'  dtype    = {array18.dtype}')
print (f'  itemsize = {array18.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_17.py
array18:
[[[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]]
information:
  ndim     = 3
  size     = 27
  shape    = (3, 3, 3)
  dtype    = float64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a 3-dim. Numpy array initialised by zeros.

3.5 Making a Numpy array using `numpy.arange ()`

A Numpy array can be created also by the function `numpy.arange ()`. Here is an example.

Python Code 19: ai202209_s03_18.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:20:47 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) using numpy.arange ()
array19 = numpy.arange (0, 10, 2)

# printing Numpy array
print (f'array19:\n{array19}')

# printing information
print (f'information:')
print (f'  ndim     = {array19.ndim}')
print (f'  size     = {array19.size}')
print (f'  shape    = {array19.shape}')
print (f'  dtype    = {array19.dtype}')
print (f'  itemsize = {array19.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_18.py
array19:
```

```
[0 2 4 6 8]
information:
  ndim     = 1
  size     = 5
  shape    = (5,)
  dtype    = int64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array using `numpy.arange ()`.

3.6 Making a Numpy array using `numpy.linspace ()`

A Numpy array can be created also by the function `numpy.linspace ()`. Here is an example.

Python Code 20: ai202209_s03_19.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:22:47 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) using numpy.linspace ()
array20 = numpy.linspace (100, 200, 11)

# printing Numpy array
print (f'array20:\n{array20}')

# printing information
print (f'information:')
print (f'  ndim     = {array20.ndim}')
print (f'  size     = {array20.size}')
print (f'  shape    = {array20.shape}')
print (f'  dtype    = {array20.dtype}')
print (f'  itemsize = {array20.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_19.py
array20:
[100. 110. 120. 130. 140. 150. 160. 170. 180. 190. 200.]
information:
  ndim     = 1
  size     = 11
  shape    = (11,)
  dtype    = float64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array using `numpy.linspace ()`.

3.7 Making a Numpy array using `numpy.logspace ()`

A Numpy array can be created also by the function `numpy.logspace ()`. Here is an example.

Python Code 21: ai202209_s03_20.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:24:22 (CST) daisuke>
#

# importing numpy module
import numpy

# making a Numpy array (ndarray) using numpy.logspace ()
array21 = numpy.logspace (0, 3, 10)

# printing Numpy array
print (f'array21:\n{array21}')

# printing information
print (f'information:')
print (f'  ndim     = {array21.ndim}')
print (f'  size     = {array21.size}')
print (f'  shape    = {array21.shape}')
print (f'  dtype    = {array21.dtype}')
print (f'  itemsize = {array21.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_20.py
array21:
[   1.            2.15443469    4.64158883   10.           21.5443469
   46.41588834  100.          215.443469    464.15888336 1000.         ]
information:
  ndim     = 1
  size     = 10
  shape    = (10,)
  dtype    = float64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to make a Numpy array using `numpy.logspace ()`.

4 Appending an element to existing Numpy array

The function `numpy.append ()` can be used to append an element to an existing Numpy array. Here is an example.

Python Code 22: ai202209_s03_21.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:32:45 (CST) daisuke>
#
```

```
# importing numpy module
import numpy

# making a Numpy array (ndarray) using numpy.linspace ()
array22 = numpy.linspace (0, 10, 11)

# printing Numpy array
print (f'array22:\n{array22}')

# printing information
print (f'information:')
print (f'  ndim     = {array22.ndim}')
print (f'  size     = {array22.size}')
print (f'  shape    = {array22.shape}')
print (f'  dtype    = {array22.dtype}')
print (f'  itemsize = {array22.itemsize} byte')

# appending one more data to "array22"
array22 = numpy.append (array22, 11.0)

# printing Numpy array
print (f'array22:\n{array22}')

# printing information
print (f'information:')
print (f'  ndim     = {array22.ndim}')
print (f'  size     = {array22.size}')
print (f'  shape    = {array22.shape}')
print (f'  dtype    = {array22.dtype}')
print (f'  itemsize = {array22.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_21.py
array22:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
information:
  ndim     = 1
  size     = 11
  shape    = (11,)
  dtype    = float64
  itemsize = 8 byte
array22:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
information:
  ndim     = 1
  size     = 12
  shape    = (12,)
  dtype    = float64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to append an element to an existing Numpy array.

5 Concatenating multiple numpy arrays

The function `numpy.concatenate ()` can be used to join two or more Numpy arrays into a single Numpy array. Here is an example.

Python Code 23: ai202209_s03_22.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:46:08 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays using numpy.linspace ()
array23 = numpy.linspace (0, 10, 11)
array24 = numpy.linspace (11, 20, 10)

# printing Numpy array
print (f'array23:\n{array23}')

# printing information
print (f'information:')
print (f'  ndim     = {array23.ndim}')
print (f'  size     = {array23.size}')
print (f'  shape    = {array23.shape}')
print (f'  dtype    = {array23.dtype}')
print (f'  itemsize = {array23.itemsize} byte')

# printing Numpy array
print (f'array24:\n{array24}')

# printing information
print (f'information:')
print (f'  ndim     = {array24.ndim}')
print (f'  size     = {array24.size}')
print (f'  shape    = {array24.shape}')
print (f'  dtype    = {array24.dtype}')
print (f'  itemsize = {array24.itemsize} byte')

# concatenating array23 and array24
array25 = numpy.concatenate ([array23, array24])

# printing Numpy array
print (f'array25:\n{array25}')

# printing information
print (f'information:')
print (f'  ndim     = {array25.ndim}')
print (f'  size     = {array25.size}')
print (f'  shape    = {array25.shape}')
print (f'  dtype    = {array25.dtype}')
print (f'  itemsize = {array25.itemsize} byte')
```

Execute above script.

```
% ./ai202209_s03_22.py
array23:
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
information:
  ndim     = 1
  size     = 11
  shape    = (11,)
  dtype    = float64
  itemsize = 8 byte
array24:
[11. 12. 13. 14. 15. 16. 17. 18. 19. 20.]
information:
  ndim     = 1
  size     = 10
  shape    = (10,)
  dtype    = float64
  itemsize = 8 byte
array25:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20.]
information:
  ndim     = 1
  size     = 21
  shape    = (21,)
  dtype    = float64
  itemsize = 8 byte
```

Try following practice.

Practice 03-01

Make your own Python script to concatenate two Numpy arrays.

6 Operations of Numpy arrays

Try some operations of Numpy arrays.

6.1 Adding two Numpy arrays

Try addition of Numpy arrays. Here is an example.

Python Code 24: ai202209_s03_23.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:48:47 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays using numpy.linspace ()
a = numpy.linspace (0.0, 9.0, 10)
b = numpy.linspace (10.0, 19.0, 10)

# printing a and b
print (f'a = {a}')
print (f'b = {b}')

# calculation
```

```
# no need of using "for"
c = a + b

# printing c
print (f'c = a + b = {c}')
```

Execute above script.

```
% ./ai202209_s03_23.py
a = [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
b = [10. 11. 12. 13. 14. 15. 16. 17. 18. 19.]
c = a + b = [10. 12. 14. 16. 18. 20. 22. 24. 26. 28.]
```

Try following practice.

Practice 03-01

Make your own Python script to carry out addition of two Numpy arrays.

6.2 Subtracting one Numpy array from the other

Try subtraction of Numpy arrays. Here is an example.

Python Code 25: ai202209_s03_24.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:53:47 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays using numpy.linspace ()
a = numpy.linspace (100.0, 109.0, 10)
b = numpy.linspace (0.0, 9.0, 10)

# printing a and b
print (f'a = {a}')
print (f'b = {b}')

# calculation
# no need of using "for"
c = a - b

# printing c
print (f'c = a - b = {c}')
```

Execute above script.

```
% ./ai202209_s03_24.py
a = [100. 101. 102. 103. 104. 105. 106. 107. 108. 109.]
b = [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
c = a - b = [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
```

Try following practice.

Practice 03-01

Make your own Python script to carry out subtraction of one Numpy array from the other.

6.3 Multiplication of two Numpy arrays

Try multiplication of two Numpy arrays. Here is an example.

Python Code 26: ai202209_s03_25.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:53:34 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays using numpy.linspace ()
a = numpy.linspace (0.0, 9.0, 10)
b = numpy.linspace (1.0, 10.0, 10)

# printing a and b
print (f'a = {a}')
print (f'b = {b}')

# calculation
# no need of using "for"
c = a * b

# printing c
print (f'c = a * b = {c}')
```

Execute above script.

```
% ./ai202209_s03_25.py
a = [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
b = [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
c = a * b = [ 0.  2.  6. 12. 20. 30. 42. 56. 72. 90.]
```

Try following practice.

Practice 03-01

Make your own Python script to carry out multiplication of two Numpy arrays.

6.4 Division of one Numpy array by the other

Try division of one Numpy array by the other. Here is an example.

Python Code 27: ai202209_s03_26.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:57:34 (CST) daisuke>
#

# importing numpy module
```

```
import numpy

# making Numpy arrays using numpy.linspace ()
a = numpy.linspace (4.0, 40.0, 10)
b = numpy.linspace (2.0, 20.0, 10)

# printing a and b
print (f'a = {a}')
print (f'b = {b}')

# calculation
# no need of using "for"
c = a / b

# printing c
print (f'c = a / b = {c}')
```

Execute above script.

```
% ./ai202209_s03_26.py
a = [ 4.  8. 12. 16. 20. 24. 28. 32. 36. 40.]
b = [ 2.  4.  6.  8. 10. 12. 14. 16. 18. 20.]
c = a / b = [2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
```

Try following practice.

Practice 03-01

Make your own Python script to carry out division of one Numpy array by the other.

6.5 Square of a Numpy array

Try to calculate a square of a Numpy array. Here is an example.

Python Code 28: ai202209_s03_27.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 21:59:48 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays using numpy.linspace ()
a = numpy.linspace (0.0, 10.0, 11)

# printing a
print (f'a = {a}')

# calculation
# no need of using "for"
b = a**2

# printing b
print (f'b = a**2 = {b}')
```

Execute above script.

```
% ./ai202209_s03_27.py
a = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
b = a**2 = [  0.   1.   4.   9.  16.  25.  36.  49.  64.  81. 100.]
```

Try following practice.

Practice 03-01

Make your own Python script to carry out a calculation of taking a square of a Numpy array.

6.6 Square-root of a Numpy array

Try to calculate a square-root of a Numpy array. Here is an example.

Python Code 29: ai202209_s03_28.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 22:01:59 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays using numpy.linspace ()
a = numpy.linspace (0.0, 10.0, 11)

# printing a
print (f'a = {a}')

# calculation
# no need of using "for"
b = numpy.sqrt (a)

# printing b
print (f'b = sqrt (a) = {b}')
```

Execute above script.

```
% ./ai202209_s03_28.py
a = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
b = sqrt (a) = [0.         1.         1.41421356 1.73205081 2.         2.23606798
 2.44948974 2.64575131 2.82842712 3.         3.16227766]
```

Try following practice.

Practice 03-01

Make your own Python script to carry out a calculation of taking a square-root of a Numpy array.

6.7 Taking a log of a Numpy array

Try to take a log of a Numpy array. Here is an example.

Python Code 30: ai202209_s03_29.py

```
#!/usr/pkg/bin/python3.9
```

```
#
# Time-stamp: <2022/09/25 22:04:43 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays using numpy.logspace ()
a = numpy.logspace (-5.0, 5.0, 11)

# printing a
print (f'a = {a}')

# calculation
# no need of using "for"
b = numpy.log10 (a)

# printing b
print (f'b = log10 (a) = {b}')
```

Execute above script.

```
% ./ai202209_s03_29.py
a = [1.e-05 1.e-04 1.e-03 1.e-02 1.e-01 1.e+00 1.e+01 1.e+02 1.e+03 1.e+04
 1.e+05]
b = log10 (a) = [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.]
```

Try following practice.

Practice 03-01

Make your own Python script to carry out a calculation of taking a log of a Numpy array.

6.8 Trigonometric functions

Try trigonometric functions. Here is an example.

Python Code 31: ai202209_s03_30.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 22:12:53 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays using numpy.linspace ()
a_deg = numpy.linspace (0.0, 360.0, 13)

# printing a_deg
print (f'a_deg = {a_deg}')

# angle in radian
a_rad = a_deg / 180.0 * numpy.pi

# printing a_rad
print (f'a_rad = {a_rad}')
```

```
# calculation
# no need of using "for"
sin_a = numpy.sin (a_rad)
cos_a = numpy.cos (a_rad)

# printing b
print (f'sin (a) = {sin_a}')
print (f'cos (a) = {cos_a}')
```

Execute above script.

```
% ./ai202209_s03_30.py
a_deg = [  0.  30.  60.  90. 120. 150. 180. 210. 240. 270. 300. 330. 360.]
a_rad = [0.          0.52359878 1.04719755 1.57079633 2.0943951  2.61799388
 3.14159265 3.66519143 4.1887902  4.71238898 5.23598776 5.75958653
 6.28318531]
sin (a) = [ 0.00000000e+00  5.00000000e-01  8.66025404e-01  1.00000000e+00
  8.66025404e-01  5.00000000e-01  1.22464680e-16 -5.00000000e-01
 -8.66025404e-01 -1.00000000e+00 -8.66025404e-01 -5.00000000e-01
 -2.44929360e-16]
cos (a) = [ 1.00000000e+00  8.66025404e-01  5.00000000e-01  6.12323400e-17
 -5.00000000e-01 -8.66025404e-01 -1.00000000e+00 -8.66025404e-01
 -5.00000000e-01 -1.83697020e-16  5.00000000e-01  8.66025404e-01
  1.00000000e+00]
```

Try following practice.

Practice 03-01

Make your own Python script to use the function `numpy.tan ()`.

6.9 Operations of 2-dimensional Numpy arrays

Try addition of two 2-dimensional Numpy arrays. Here is an example.

Python Code 32: ai202209_s03_31.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 22:26:42 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays
a = numpy.array ([ [1, 2], [3, 4] ])
b = numpy.array ([ [5, 6], [7, 8] ])

# printing a and b
print (f'a:\n{a}')
print (f'b:\n{b}')

# calculation
# no need of using "for"
c = a + b
```

```
# printing c
print (f'c = a + b:\n{c}')
```

Execute above script.

```
% ./ai202209_s03_31.py
a:
[[1 2]
 [3 4]]
b:
[[5 6]
 [7 8]]
c = a + b:
[[ 6  8]
 [10 12]]
```

Try following practice.

Practice 03-01

Make your own Python script to carry out addition, subtraction, multiplication, division of two 2-dim. Numpy arrays.

6.10 Vector calculations

Try dot product and inner product of two 2-dim. vectors. Here is an example.

Python Code 33: ai202209_s03_32.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 22:32:33 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays
a = numpy.array ([1.0, 2.0])
b = numpy.array ([3.0, 4.0])

# printing a and b
print (f'a = {a}')
print (f'b = {b}')

# dot product of two vectors
dot = numpy.dot (a, b)

# printing dot product
print (f'dot = {dot}')

# inner product of two vectors
inner = numpy.inner (a, b)

# printing inner product
print (f'inner = {inner}')
```

Execute above script.

```
% ./ai202209_s03_32.py
a = [1. 2.]
b = [3. 4.]
dot = 11.0
inner = 11.0
```

Try following practice.

Practice 03-01

Make your own Python script to calculate dot and inner products of two 2-dim. vectors.

Try dot product, inner product, and cross product of two 3-dim. vectors. Here is an example.

Python Code 34: ai202209_s03_33.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 22:36:59 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays
a = numpy.array ([1.0, 1.0, 0.0])
b = numpy.array ([-1.0, 3.0, 0.0])

# printing a and b
print (f'a = {a}')
print (f'b = {b}')

# dot product of two vectors
dot = numpy.dot (a, b)

# printing dot product
print (f'dot = {dot}')

# inner product of two vectors
inner = numpy.inner (a, b)

# printing inner product
print (f'inner = {inner}')

# cross product of two vectors
cross = numpy.cross (a, b)

# printing cross product
print (f'cross = {cross}')
```

Execute above script.

```
% ./ai202209_s03_33.py
a = [1. 1. 0.]
b = [-1.  3.  0.]
dot = 2.0
inner = 2.0
cross = [ 0. -0.  4.]
```

Try following practice.

Practice 03-01

Make your own Python script to calculate dot, inner, and cross products of two 3-dim. vectors.

6.11 Matrix calculations

Try multiplication of two matrices. Here is an example.

Python Code 35: ai202209_s03_34.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 22:42:33 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays (2x2 matrix)
A = numpy.array ([ [1.0, 2.0], [3.0, 4.0] ])
B = numpy.array ([ [4.0, 2.0], [1.0, 3.0] ])

# printing A and B
print (f'A:\n{A}')
print (f'B:\n{B}')

# matrix product
C = A @ B

# printing C
print (f'C = A @ B:\n{C}')
```

Execute above script.

```
% ./ai202209_s03_34.py
A:
[[1. 2.]
 [3. 4.]]
B:
[[4. 2.]
 [1. 3.]]
C = A @ B:
[[ 6.  8.]
 [16. 18.]]
```

Try following practice.

Practice 03-01

Make your own Python script to calculate matrix product of two matrices.

Here is the other way to take a matrix product.

Python Code 36: ai202209_s03_35.py

```
#!/usr/pkg/bin/python3.9

#
```

```
# Time-stamp: <2022/09/25 22:43:29 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays (2x2 matrix)
A = numpy.array ([ [1.0, 2.0], [3.0, 4.0] ])
B = numpy.array ([ [4.0, 2.0], [1.0, 3.0] ])

# printing A and B
print (f'A:\n{A}')
print (f'B:\n{B}')

# the other way to take a matrix product
C = numpy.matmul (A, B)

# printing C
print (f'C = matmul (A, B):\n{C}')
```

Execute above script.

```
% ./ai202209_s03_35.py
A:
[[1. 2.]
 [3. 4.]]
B:
[[4. 2.]
 [1. 3.]]
C = matmul (A, B):
[[ 6.  8.]
 [16. 18.]]
```

Try following practice.

Practice 03-01

Make your own Python script to calculate matrix product of two matrices.

Here is one more way to take a matrix product.

Python Code 37: ai202209_s03_36.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 22:43:54 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy arrays (2x2 matrix)
A = numpy.array ([ [1.0, 2.0], [3.0, 4.0] ])
B = numpy.array ([ [4.0, 2.0], [1.0, 3.0] ])

# printing A and B
print (f'A:\n{A}')
print (f'B:\n{B}')
```

```
# matrix product
C = A.dot (B)

# printing C
print (f'C = A.dot (B):\n{C}')
```

Execute above script.

```
% ./ai202209_s03_36.py
A:
[[1. 2.]
 [3. 4.]]
B:
[[4. 2.]
 [1. 3.]]
C = A.dot (B):
[[ 6.  8.]
 [16. 18.]]
```

Try following practice.

Practice 03-01

Make your own Python script to calculate matrix product of two matrices.

Try to make a unit matrix. Here is an example.

Python Code 38: ai202209_s03_37.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 22:54:03 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array (2x2 matrix)
A = numpy.array ([ [1.0, 2.0], [3.0, 4.0] ])

# printing A
print (f'A:\n{A}')

# making Numpy array (2x2 unit matrix)
E2 = numpy.identity (2)

# printing E2
print (f'E2:\n{E2}')

# calculation
B = A @ E2

# printing B
print (f'B = A @ E2:\n{B}')

# making Numpy array (3x3 matrix)
C = numpy.array ([ [1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0] ])

# printing C
```

```
print (f'C:\n{C}')

# making Numpy array (3x3 unit matrix)
E3 = numpy.identity (3)

# printing E3
print (f'E3:\n{E3}')

# calculation
D = E3 @ C

# printing D
print (f'D = E3 @ C:\n{D}')
```

Execute above script.

```
% ./ai202209_s03_37.py
A:
[[1. 2.]
 [3. 4.]]
E2:
[[1. 0.]
 [0. 1.]]
B = A @ E2:
[[1. 2.]
 [3. 4.]]
C:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
E3:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
D = E3 @ C:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

Try following practice.

Practice 03-01

Make your own Python script to make a unit matrix.

Try to calculate inverse matrix. Here is an example.

Python Code 39: ai202209_s03_38.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 22:57:53 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array (2x2 matrix)
A = numpy.array ([ [5.0, 3.0], [6.0, 4.0] ])
```

```
# printing A
print (f'A:\n{A}')

# calculating inverse matrix of A
B = numpy.linalg.inv (A)

# priting B
print (f'B = A^-1:\n{B}')

# calculation of A @ B
C = A @ B

# printing C
print (f'C = A @ B:\n{C}')
```

Execute above script.

```
% ./ai202209_s03_38.py
A:
[[5. 3.]
 [6. 4.]]
B = A^-1:
[[ 2.  -1.5]
 [-3.   2.5]]
C = A @ B:
[[ 1.0000000e+00 -8.8817842e-16]
 [ 0.0000000e+00  1.0000000e+00]]
```

Try following practice.

Practice 03-01

Make your own Python script to calculate inverse matrix of a given 3×3 matrix.

7 Accessing to elements by indexing and slicing

Here is an example of accessing to an element by indexing for 1-dim. Numpy array.

Python Code 40: ai202209_s03_39.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 23:08:41 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array
a = numpy.linspace (0.0, 10.0, 11)

# printing A
print (f'a:\n{a}')

# accessing to an element by indexing
print (f'a[0] = {a[0]}')
print (f'a[1] = {a[1]}')
```

```
print (f'a[5] = {a[5]}')
print (f'a[-1] = {a[-1]}')
print (f'a[-3] = {a[-3]}')
```

Execute above script.

```
% ./ai202209_s03_39.py
a:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
a[0] = 0.0
a[1] = 1.0
a[5] = 5.0
a[-1] = 10.0
a[-3] = 8.0
```

Try following practice.

Practice 03-01

Make your own Python script to access to elements of 1-dim. Numpy array by indexing.

Here is an example of accessing to an element by indexing for 2-dim. Numpy array.

Python Code 41: ai202209_s03_40.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 23:17:43 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array
a = numpy.array ([ [0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0] ])

# printing A
print (f'a:\n{a}')

# accessing to an element by indexing
print (f'a[0][0] = {a[0][0]}')
print (f'a[1][2] = {a[1][2]}')
print (f'a[1,2] = {a[1,2]}')
print (f'a[0,-1] = {a[0,-1]}')
print (f'a[0] = {a[0]}')
print (f'a[-1] = {a[-1]}')
```

Execute above script.

```
% ./ai202209_s03_40.py
a:
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
a[0][0] = 0.0
a[1][2] = 5.0
a[1,2] = 5.0
a[0,-1] = 2.0
a[0] = [0. 1. 2.]
```

```
a[-1] = [6. 7. 8.]
```

Try following practice.

Practice 03-01

Make your own Python script to access to elements of 2-dim. Numpy array by indexing.

Here is an example of accessing to elements by slicing for 1-dim. Numpy array.

Python Code 42: ai202209_s03_41.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 23:19:50 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array
a = numpy.linspace (0.0, 10.0, 11)

# printing A
print (f'a:\n{a}')

# accessing to an element by indexing
print (f'a[2:5] = {a[2:5]}')
print (f'a[6:] = {a[6:]}')
print (f'a[:3] = {a[:3]}')
print (f'a[:] = {a[:]}')
print (f'a[-3:] = {a[-3:]}')
```

Execute above script.

```
% ./ai202209_s03_41.py
a:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
a[2:5] = [2. 3. 4.]
a[6:] = [ 6.  7.  8.  9. 10.]
a[:3] = [0. 1. 2.]
a[:] = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
a[-3:] = [ 8.  9. 10.]
```

Try following practice.

Practice 03-01

Make your own Python script to access to elements of 1-dim. Numpy array by slicing.

Here is an example of accessing to elements by slicing for 2-dim. Numpy array.

Python Code 43: ai202209_s03_42.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 23:30:26 (CST) daisuke>
#
```

```
# importing numpy module
import numpy

# making Numpy array
a = numpy.array ([ [0.0, 1.0, 2.0, 3.0], [4.0, 5.0, 6.0, 7.0], \
                   [8.0, 9.0, 10.0, 11.0], [12.0, 13.0, 14.0, 15.0] ])

# printing A
print (f'a:\n{a}')

# accessing to an element by indexing
print (f'a[0:2,1:3]:\n{a[0:2,1:3]}')
print (f'a[1:3,:]:\n{a[1:3,:]}')
print (f'a[:,1:3]:\n{a[:,1:3]}')
```

Execute above script.

```
% ./ai202209_s03_42.py
a:
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
 [12. 13. 14. 15.]]
a[0:2,1:3]:
[[1. 2.]
 [5. 6.]]
a[1:3,:]:
[[ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]]
a[:,1:3]:
[[ 1.  2.]
 [ 5.  6.]
 [ 9. 10.]
 [13. 14.]]
```

Try following practice.

Practice 03-01

Make your own Python script to access to elements of 2-dim. Numpy array by slicing.

8 Copying a Numpy array

First, try following script. The statement `b = a` does not copy a Numpy array, but it only gives a different name, and the two names share the same object after executing `b = a`. So, both `a` and `b` share the same object ID. If `a` is modified, then, `b` is also modified.

Python Code 44: ai202209_s03_43.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 23:42:51 (CST) daisuke>
#

# importing numpy module
import numpy
```

```
# making Numpy array
a = numpy.linspace (0.0, 10.0, 11)

# printing "a"
print (f'a:\n{a}')

# trying b = a
b = a

# printing "b"
print (f'b:\n{b}')

# IDs of "a" and "b"
print (f'id (a) = {id (a)}')
print (f'id (b) = {id (b)}')

# changing "a[5]"
a[5] += 10

# printing "a"
print (f'a:\n{a}')

# printing "b"
print (f'b:\n{b}')

# IDs of "a" and "b"
print (f'id (a) = {id (a)}')
print (f'id (b) = {id (b)}')
```

Execute above script.

```
% ./ai202209_s03_43.py
a:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
b:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
id (a) = 126069716304160
id (b) = 126069716304160
a:
[ 0.  1.  2.  3.  4. 15.  6.  7.  8.  9. 10.]
b:
[ 0.  1.  2.  3.  4. 15.  6.  7.  8.  9. 10.]
id (a) = 126069716304160
id (b) = 126069716304160
```

If above is not what you want, try `.copy ()` instead.

Python Code 45: ai202209_s03_44.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 23:43:24 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array
```

```
a = numpy.linspace (0.0, 10.0, 11)

# printing "a"
print (f'a:\n{a}')

# trying a.copy ()
b = a.copy ()

# printing "b"
print (f'b:\n{b}')

# IDs of "a" and "b"
print (f'id (a) = {id (a)}')
print (f'id (b) = {id (b)}')

# changing "a[5]"
a[5] += 10

# printing "a"
print (f'a:\n{a}')

# printing "b"
print (f'b:\n{b}')

# IDs of "a" and "b"
print (f'id (a) = {id (a)}')
print (f'id (b) = {id (b)}')
```

Execute above script.

```
% ./ai202209_s03_44.py
a:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
b:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
id (a) = 135787519201568
id (b) = 135787503351328
a:
[ 0.  1.  2.  3.  4. 15.  6.  7.  8.  9. 10.]
b:
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
id (a) = 135787519201568
id (b) = 135787503351328
```

Try following practice.

Practice 03-01

Make your own Python script to copy a Numpy array using `.copy ()`.

9 Sorting a Numpy array

Try sorting of a Numpy array. Here is an example of sorting by quicksort.

Python Code 46: ai202209_s03_45.py

```
#!/usr/pkg/bin/python3.9

#
```

```
# Time-stamp: <2022/09/25 23:55:05 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array
a = numpy.array ([5.0, 3.0, 7.0, 4.0, 9.0, 8.0, 1.0, 6.0, 2.0, 0.0])

# printing "a"
print (f'a:\n{a}')

# sorting
b = numpy.sort (a)

# printing "b"
print (f'b = sort (a):\n{b}')
```

Execute above script.

```
% ./ai202209_s03_45.py
a:
[5. 3. 7. 4. 9. 8. 1. 6. 2. 0.]
b = sort (a):
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

Try following practice.

Practice 03-01

Make your own Python script to sort a Numpy array using `numpy.sort ()`.

Here is an example of sorting by timsort.

Python Code 47: ai202209_s03_46.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/25 23:59:48 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array
a = numpy.array ([5.0, 3.0, 7.0, 4.0, 9.0, 8.0, 1.0, 6.0, 2.0, 0.0])

# printing "a"
print (f'a:\n{a}')

# sorting
b = numpy.sort (a, kind='stable')

# printing "b"
print (f'b = sort (a, kind="stable"):\n{b}')
```

Execute above script.

```
% ./ai202209_s03_46.py
a:
[5. 3. 7. 4. 9. 8. 1. 6. 2. 0.]
b = sort (a, kind="stable"):
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

Try following practice.

Practice 03-01

Make your own Python script to sort a Numpy array by timsort using `numpy.sort ()`.

Here is an example of sorting by descending order.

Python Code 48: ai202209_s03_47.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 00:05:49 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array
a = numpy.array ([5.0, 3.0, 7.0, 4.0, 9.0, 8.0, 1.0, 6.0, 2.0, 0.0])

# printing "a"
print (f'a:\n{a}')

# sorting by descending order
b = numpy.sort (a) [::-1]

# printing "b"
print (f'b = sort (a) [::-1]:\n{b}')
```

Execute above script.

```
% ./ai202209_s03_47.py
a:
[5. 3. 7. 4. 9. 8. 1. 6. 2. 0.]
b = sort (a) [::-1]:
[9. 8. 7. 6. 5. 4. 3. 2. 1. 0.]
```

Here is the other way to sort a Numpy array by descending order.

Python Code 49: ai202209_s03_48.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 00:13:11 (CST) daisuke>
#

# importing numpy module
import numpy

# making Numpy array
```

```
a = numpy.array ([5.0, 3.0, 7.0, 4.0, 9.0, 8.0, 1.0, 6.0, 2.0, 0.0])

# printing "a"
print (f'a:\n{a}')

# sorting by descending order
b = numpy.flip (numpy.sort (a))

# printing "b"
print (f'b = flip (sort (a)):\n{b}')
```

Execute above script.

```
% ./ai202209_s03_48.py
a:
[5. 3. 7. 4. 9. 8. 1. 6. 2. 0.]
b = flip (sort (a)):
[9. 8. 7. 6. 5. 4. 3. 2. 1. 0.]
```

Try following practice.

Practice 03-01

Make your own Python script to sort a Numpy array by descending order using `numpy.sort ()`.

10 Random numbers

Numpy offers some functions and methods to generate random numbers.

10.1 Generating one random number of uniform distribution between 0 and 1

Try to generate one random number of uniform distribution between 0 and 1. Here is an example.

Python Code 50: ai202209_s03_49.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 00:27:26 (CST) daisuke>
#

# importing numpy module
import numpy

# random number generator
rng = numpy.random.default_rng ()

# generating a random number of uniform distribution between 0 and 1
r = rng.random ()

# printing generated random numbers
print (f'{r}')
```

Execute above script.

```
% ./ai202209_s03_49.py
0.3904039447313903
```

Try following practice.

Practice 03-01

Make your own Python script to generate one random number of uniform distribution between 0 and 1.

10.2 Generating 10 random numbers of uniform distribution between 0 and 1

Try to generate 10 random numbers of uniform distribution between 0 and 1. Here is an example.

Python Code 51: ai202209_s03_50.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 00:29:08 (CST) daisuke>
#

# importing numpy module
import numpy

# random number generator
rng = numpy.random.default_rng ()

# generating 10 random numbers of uniform distribution between 0 and 1
r = rng.random (10)

# printing generated random numbers
print (f'{r}')
```

Execute above script.

```
% ./ai202209_s03_50.py
[0.53244026 0.55734892 0.23242763 0.18164257 0.19263895 0.59408213
 0.22247783 0.18343465 0.61206641 0.72081444]
```

Try following practice.

Practice 03-01

Make your own Python script to generate 30 random number of uniform distribution between 0 and 1.

10.3 Generating 10 random numbers of uniform distribution between 100 and 200

Try to generate 10 random numbers of uniform distribution between 100 and 200. Here is an example.

Python Code 52: ai202209_s03_51.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 00:32:00 (CST) daisuke>
#

# importing numpy module
import numpy

# random number generator
rng = numpy.random.default_rng ()
```

```
# generating 10 random numbers of uniform distribution between 100 and 200
r = rng.uniform (100.0, 200.0, 10)

# printing generated random numbers
print (f'{r}')
```

Execute above script.

```
% ./ai202209_s03_51.py
[110.98978943 173.23280441 120.5302074  101.91212261 113.19576223
 187.13194498 158.81612029 123.24907021 158.20918945 191.77226679]
```

Try following practice.

Practice 03-01

Make your own Python script to generate 100 random numbers of uniform distribution between 0 and 1000.

10.4 Generating 10 random numbers of Gaussian distribution of mean 100 and standard deviation 10

Try to generate 10 random numbers of Gaussian distribution of mean 100.0 and standard deviation 10.0.

Python Code 53: ai202209_s03_52.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 00:34:59 (CST) daisuke>
#

# importing numpy module
import numpy

# random number generator
rng = numpy.random.default_rng ()

# generating 10 random numbers of Gaussian distribution
# of mean of 100.0 and standard deviation of 10.0
r = rng.normal (100.0, 10.0, 10)

# printing generated random numbers
print (f'{r}')
```

Execute above script.

```
% ./ai202209_s03_52.py
[105.44951045 106.77532509 102.25700255  98.21870363 115.16619569
  99.50833816 105.17178203  99.98809302  98.71643927  92.20143368]
```

Try following practice.

Practice 03-01

Make your own Python script to generate 50 random numbers of Gaussian distribution of mean 1000 and standard deviation 300.

11 Calculating statistical values

11.1 Mean, median, variance, and standard deviation

Generate 10^6 random numbers of uniform distribution, and then calculate mean, median, variance, and standard deviation. Here is an example.

Python Code 54: ai202209_s03_53.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 14:04:49 (CST) daisuke>
#

# importing numpy module
import numpy

# importing statistics module
import statistics

# random number generator
rng = numpy.random.default_rng ()

# generating 10^6 random numbers of uniform distribution
# between 10000 and 20000
r = rng.uniform (10000.0, 20000.0, 10**6)

# printing generated random numbers
print (f'{r}')

# printing number of data
print (f'number of data = {len (r):g}')

# statistical values calculated by numpy module
mean_n     = numpy.mean (r)
median_n   = numpy.median (r)
variance_n = numpy.var (r)
stddev_n   = numpy.std (r)

# printing statistical values
print (f'statistical values by Numpy:')
print (f'  mean     = {mean_n}')
print (f'  median   = {median_n}')
print (f'  variance = {variance_n}')
print (f'  stddev   = {stddev_n}')

# statistical values calculated by statistics module
mean_s     = statistics.fmean (r)
median_s   = statistics.median (r)
variance_s = statistics.pvariance (r)
stddev_s   = statistics.pstdev (r)

# printing statistical values
print (f'statistical values by statistics module:')
print (f'  mean     = {mean_s}')
print (f'  median   = {median_s}')
print (f'  variance = {variance_s}')
print (f'  stddev   = {stddev_s}')
```

Execute above script.

```
% ./ai202209_s03_53.py
[13970.01044173 17012.40385614 14270.32662826 ... 19763.8143464
 15349.01302689 13172.01437432]
number of data = 1e+06
statistical values by Numpy:
  mean     = 14995.53167406384
  median   = 14992.620784759158
  variance = 8335248.896802402
  stddev   = 2887.0831122089994
statistical values by statistics module:
  mean     = 14995.53167406384
  median   = 14992.620784759158
  variance = 8335248.896802404
  stddev   = 2887.083112209
```

Try following practice.

Practice 03-01

Make your own Python script to generate 10^8 random numbers of uniform distribution between 5000.0 and 10000.0 and calculate mean, median, variance, and standard deviation.

Generate 10^6 random numbers of Gaussian distribution, and then calculate mean, median, variance, and standard deviation. Here is an example.

Python Code 55: ai202209_s03_54.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 14:07:12 (CST) daisuke>
#

# importing numpy module
import numpy

# importing statistics module
import statistics

# random number generator
rng = numpy.random.default_rng ()

# generating 10^6 random numbers of Gaussian distribution
# of mean = 10000.0 and stddev = 1000.0
r = rng.normal (10000.0, 1000.0, 10**6)

# printing generated random numbers
print (f'{r}')

# printing number of data
print (f'number of data = {len (r):g}')

# statistical values calculated by numpy module
mean_n     = numpy.mean (r)
median_n   = numpy.median (r)
variance_n = numpy.var (r)
stddev_n   = numpy.std (r)

# printing statistical values
```

```
print (f'statistical values by Numpy:')
print (f'  mean     = {mean_n}')
print (f'  median   = {median_n}')
print (f'  variance = {variance_n}')
print (f'  stddev   = {stddev_n}')

# statistical values calculated by statistics module
mean_s     = statistics.fmean (r)
median_s   = statistics.median (r)
variance_s = statistics.pvariance (r)
stddev_s   = statistics.pstdev (r)

# printing statistical values
print (f'statistical values by statistics module:')
print (f'  mean     = {mean_s}')
print (f'  median   = {median_s}')
print (f'  variance = {variance_s}')
print (f'  stddev   = {stddev_s}')
```

Execute above script.

```
% ./ai202209_s03_54.py
[10035.34579689 10099.37484884  9738.85505424 ... 11729.65885492
  9538.66288799 10584.50971903]
number of data = 1e+06
statistical values by Numpy:
  mean     = 10001.72344673506
  median   = 10002.641261568453
  variance = 1000209.0307698984
  stddev   = 1000.1045099237872
statistical values by statistics module:
  mean     = 10001.723446735066
  median   = 10002.641261568453
  variance = 1000209.030769898
  stddev   = 1000.1045099237868
```

Try following practice.

Practice 03-01

Make your own Python script to generate 10^8 random numbers of Gaussian distribution of mean 30000.0 and standard deviation 500.0 and calculate mean, median, variance, and standard deviation.

11.2 Making histogram

Generate a number of random numbers and construct a histogram of the distribution. Here is an example.

Python Code 56: ai202209_s03_55.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 14:27:59 (CST) daisuke>
#

# importing numpy module
import numpy

# importing statistics module
```

```
import statistics

# random number generator
rng = numpy.random.default_rng ()

# generating 10^6 random numbers of Gaussian distribution
# of mean = 10000.0 and stddev = 300.0
r = rng.normal (10000.0, 300.0, 10**6)

# printing generated random numbers
print (f'{r}')

# printing number of data
print (f'number of data = {len (r):g}')

# statistical values calculated by numpy module
mean_n     = numpy.mean (r)
median_n   = numpy.median (r)
variance_n = numpy.var (r)
stddev_n   = numpy.std (r)

# printing statistical values
print (f'statistical values by Numpy:')
print (f'  mean     = {mean_n}')
print (f'  median   = {median_n}')
print (f'  variance = {variance_n}')
print (f'  stddev   = {stddev_n}')

# constructing a histogram
hist = numpy.histogram (r, bins=20, range=(9000.0, 11000.0))

# printing histogram
for i in range (20):
    print (f'{hist[1][i]:7.1f}-{hist[1][i]+100:7.1f} : {hist[0][i]:6d}')
```

Execute above script.

```
% ./ai202209_s03_55.py
[ 9953.36008001  9862.29538792 10256.98501728 ... 10298.40288103
 10469.31702101  9976.73077219]
number of data = 1e+06
statistical values by Numpy:
  mean     = 10000.0408424035
  median   = 10000.042201762888
  variance = 89799.37624445165
  stddev   = 299.66544052401446
 9000.0- 9100.0 :    938
 9100.0- 9200.0 :   2412
 9200.0- 9300.0 :   6042
 9300.0- 9400.0 :  12825
 9400.0- 9500.0 :  24795
 9500.0- 9600.0 :  43384
 9600.0- 9700.0 :  67574
 9700.0- 9800.0 :  94297
 9800.0- 9900.0 : 116791
 9900.0-10000.0 : 130473
10000.0-10100.0 : 130509
10100.0-10200.0 : 116990
10200.0-10300.0 :  94249
```

```
10300.0-10400.0 :  67300
10400.0-10500.0 :  43317
10500.0-10600.0 :  24984
10600.0-10700.0 :  13093
10700.0-10800.0 :   5882
10800.0-10900.0 :   2413
10900.0-11000.0 :    918
```

Try following practice.

Practice 03-01

Make your own Python script to generate 10^8 random numbers of Gaussian distribution of mean 20000.0 and standard deviation 200.0 and construct a histogram of the distribution.

11.3 Finding maximum and minimum

Make a Python script to find maximum and minimum values. Here is an example.

Python Code 57: ai202209_s03_56.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 14:53:29 (CST) daisuke>
#

# importing numpy module
import numpy

# importing statistics module
import statistics

# random number generator
rng = numpy.random.default_rng ()

# generating 10^6 random numbers of Gaussian distribution
# of mean = 10000.0 and stddev = 300.0
r = rng.normal (10000.0, 300.0, 10**6)

# printing generated random numbers
print (f'{r}')

# printing number of data
print (f'number of data = {len (r):g}')

# statistical values calculated by numpy module
mean_n     = numpy.mean (r)
median_n   = numpy.median (r)
variance_n = numpy.var (r)
stddev_n   = numpy.std (r)

# printing statistical values
print (f'statistical values by Numpy:')
print (f'  mean     = {mean_n}')
print (f'  median   = {median_n}')
print (f'  variance = {variance_n}')
print (f'  stddev   = {stddev_n}')
```

```
# finding maximum value
maximum = numpy.amax (r)

# finding minimum value
minimum = numpy.amin (r)

# printing maximum and minimum
print (f'maximum and minimum:')
print (f'  maximum = {maximum}')
print (f'  minimum = {minimum}')
```

Execute above script.

```
% ./ai202209_s03_56.py
[ 9292.1803652  10259.68146418  9793.5168449  ...  9707.65436637
  9775.40701533 10254.74527278]
number of data = 1e+06
statistical values by Numpy:
  mean     = 9999.82493552549
  median   = 9999.704204011447
  variance = 89904.62559000099
  stddev   = 299.84100051527474
maximum and minimum:
  maximum = 11380.911873142602
  minimum = 8592.80890312918
```

Try following practice.

Practice 03-01

Make your own Python script to generate 10^8 random numbers of Gaussian distribution of mean 50000.0 and standard deviation 1000.0 and find maximum and minimum values.

12 Masking outliers by masked array

Real data may contain outliers. Those outliers can be masked by masked array of Numpy. Try to mask data outside of $(mean - 3 \times stddev, mean + 3 \times stddev)$. Then, calculate statistical values.

Python Code 58: ai202209_s03_57.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/09/26 15:16:02 (CST) daisuke>
#

# importing numpy module
import numpy

# importing statistics module
import statistics

# random number generator
rng = numpy.random.default_rng ()

# generating 30 random numbers of Gaussian distribution
# of mean = 1000.0 and stddev = 100.0
r = rng.normal (1000.0, 100.0, 30)
```

```
# adding outliers
r[10] += 1500.0
r[20] += 2000.0

# printing generated random numbers
print (f'{r}')

# printing number of data
print (f'number of data = {len (r):g}')

# statistical values calculated by numpy module
mean_n     = numpy.mean (r)
median_n   = numpy.median (r)
variance_n = numpy.var (r)
stddev_n   = numpy.std (r)

# printing statistical values
print (f'statistical values by Numpy:')
print (f'  mean     = {mean_n}')
print (f'  median   = {median_n}')
print (f'  variance = {variance_n}')
print (f'  stddev   = {stddev_n}')

# finding data outside of [ mean - 3.0 * stddev, mean + 3.0 * stddev ]
llimit = median_n - 3.0 * stddev_n
ulimit = median_n + 3.0 * stddev_n

# making an empty array for a mask
mask = numpy.array ([])

# examining data
for i in range (len (r)):
    if ( (r[i] < llimit) or (r[i] > ulimit) ):
        mask = numpy.append (mask, True)
    else:
        mask = numpy.append (mask, False)

# printing mask
print (f'mask:\n{mask}')

# making masked array
r_masked = numpy.ma.array (r, mask=mask)

# printing masked array
print (f'r_masked:\n{r_masked}')

# calculation of statistical values
mean_m     = numpy.ma.mean (r_masked)
median_m   = numpy.ma.median (r_masked)
variance_m = numpy.ma.var (r_masked)
stddev_m   = numpy.ma.std (r_masked)

# printing statistical values
print (f'statistical values by Numpy.ma:')
print (f'  mean     = {mean_m}')
print (f'  median   = {median_m}')
print (f'  variance = {variance_m}')
print (f'  stddev   = {stddev_m}')
```

Execute above script.

```
% ./ai202209_s03_57.py
[ 927.6227543   939.01318322  959.69841656 1078.81639932  987.73178932
 1074.5113368  1023.68045802  917.16956672  860.00944848  932.49966094
 2528.14266414  993.69646148  995.21820233  998.93230305 1096.20077609
 1057.20680243 1133.79323157 1090.73071584  897.52875846  895.80883737
 2981.06220418  961.2498228  1037.11675381  911.00619529 1084.28876337
  942.26340331 1095.65694554 1204.71146035  900.33797785 1133.72153451]
number of data = 30
statistical values by Numpy:
  mean     = 1121.3142275816633
  median   = 997.0752526898933
  variance = 201085.7692860511
  stddev   = 448.42587936698203
mask:
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.
 0. 0. 0. 0. 0. 0.]
r_masked:
[927.6227542989321 939.0131832182485 959.6984165610623 1078.8163993173905
 987.731789324035 1074.511336804074 1023.6804580168505 917.1695667168722
 860.009448480189 932.4996609445693 -- 993.6964614829418 995.2182023324949
 998.9323030472917 1096.200776088654 1057.2068024251598 1133.793231566604
 1090.7307158389237 897.5287584624365 895.8088373721225 --
 961.2498227984952 1037.1167538082225 911.0061952876137 1084.2887633737805
 942.263403307733 1095.6569455418505 1204.7114603467885 900.3379778526038
 1133.7215345119123]
statistical values by Numpy.ma:
  mean     = 1004.6507842545661
  median   = 994.4573319077183
  variance = 7630.508569671254
  stddev   = 87.35278226634372
```

Try following practice.

Practice 03-01

Make your own Python script to generate 10^4 random numbers of Gaussian distribution of mean 3000.0 and standard deviation 200.0. Add some outliers. Then, mask the data outside of $(mean-3\times stddev, mean+3\times stddev)$, and then calculate statistical values.

13 For your further reading

Visit the official website of Numpy, and read the official Numpy documentation.

- Numpy: `https://numpy.org/`
  - Numpy documentation: `https://numpy.org/doc/stable/`

14 Assignment

- Visit the official website of Numpy (`https://numpy.org/`). Read the official document and learn about Numpy. Summarise what you have learnt.
- Consider vectors $\vec{a} = (1, 3)$ and $\vec{b} = (4, 2)$. Make a Python script to calculate $\vec{a} \cdot \vec{b}$. Show the source code of your Python script. Execute the script and show the result.
- Consider vectors $\vec{c} = (1, 0, 0)$ and $\vec{d} = (0, 1, 0)$. Make a Python script to calculate $\vec{a} \times \vec{b}$. Show the source code of your Python script. Execute the script and show the result.

- Consider a 2×2 matrix A.

$$A = \begin{pmatrix} 5 & 3 \\ 6 & 4 \end{pmatrix}$$

  - Make a Python script to calculate the determinant of the matrix A. Show the source code of your Python script. Execute the script and show the result.
  - Make a Python script to calculate the eigenvalues of the matrix A. Show the source code of your Python script. Execute the script and show the result.
  - Make a Python script to calculate the eigenvectors of the matrix A. Show the source code of your Python script. Execute the script and show the result.
  - Make a Python script to calculate the matrix A^{-1}. Show the source code of your Python script. Execute the script and show the result.

- Use `numpy.linspace` to calculate values of $\tan\left(\frac{1}{12}\pi\right)$, $\tan\left(\frac{1}{6}\pi\right)$, $\tan\left(\frac{1}{4}\pi\right)$, $\tan\left(\frac{1}{3}\pi\right)$, $\tan\left(\frac{5}{12}\pi\right)$, $\tan\left(\frac{1}{2}\pi\right)$, $\tan\left(\frac{7}{12}\pi\right)$, $\tan\left(\frac{2}{3}\pi\right)$, $\tan\left(\frac{3}{4}\pi\right)$, $\tan\left(\frac{5}{6}\pi\right)$, $\tan\left(\frac{11}{12}\pi\right)$, and $\tan(\pi)$. Show the source code of your Python script. Execute the script and show the result.

- Use `numpy.arange` and `numpy.sum` to calculate $\sum_{i=0}^{10000} i$. Show the source code of

- Generate 10^8 random numbers of gamma distribution of $k = 2$ and $\theta = 2$. Find mean, median, variance, and standard deviation of this dataset. Show the source code of your Python script. Execute the script and show the result.

- Generate 10^8 random numbers of Gaussian distribution of mean value of 1000 and standard deviation of 100. Make a histogram of the distribution. Show the source code of your Python script. Execute the script and show the result.

- Generate 10^8 random numbers of Gaussian distribution of mean value of 1000 and standard deviation of 100. Add 1000 outliers to the distribution. Examine the data, and find and mask those outliers. Calculate mean, median, standard deviation of the masked data. Show the source code of your Python script. Execute the script and show the result.