Numpy Scipy Matplotlib

Numpy

Short for Numerical Python

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

NumPy contains, among other things:

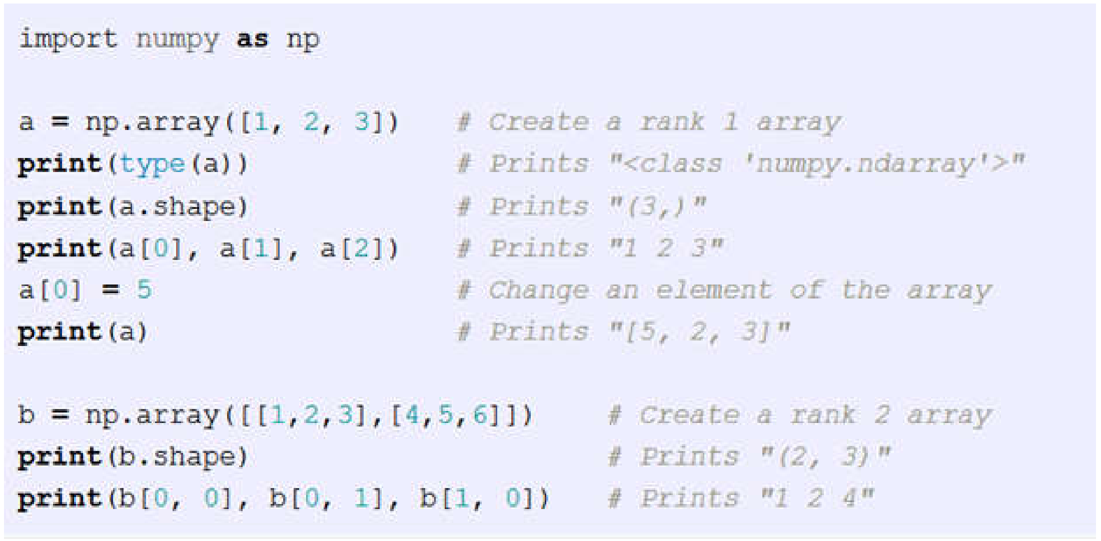
- A fast and efficient multidimensional array object ndarray
- Functions for performing element-wise computations with arrays or mathematical operations between arrays
- Tools for reading and writing array-based datasets to disk
- Linear algebra operations, Fourier transform, and random number generation
- A mature C API to enable Python extensions and native C or C++ code to access NumPy’s data structures and computational facilities

Install: pip install numpy

Arrays

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.

We can initialize numpy arrays from nested Python lists, and access elements using square brackets

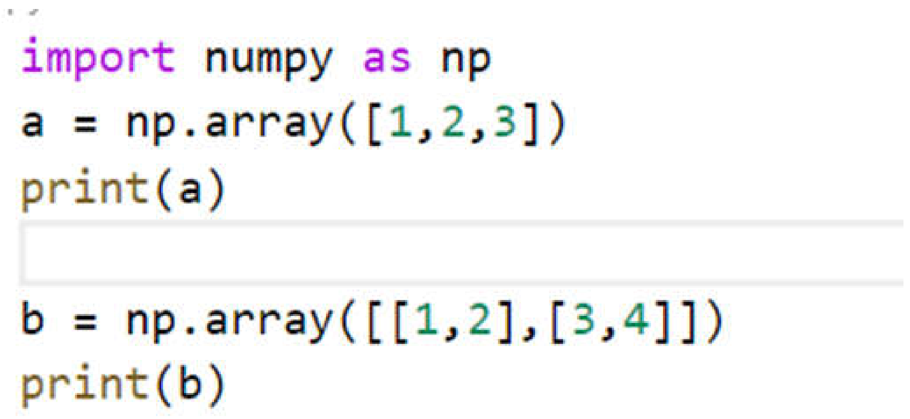


```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

The most important object defined in NumPy is an N-dimensional array type called **ndarray**

It describes the collection of items of the same type.

Items in the collection can be accessed using a zero-based index.



```
import numpy as np
a = np.array([1,2,3])
print(a)
```

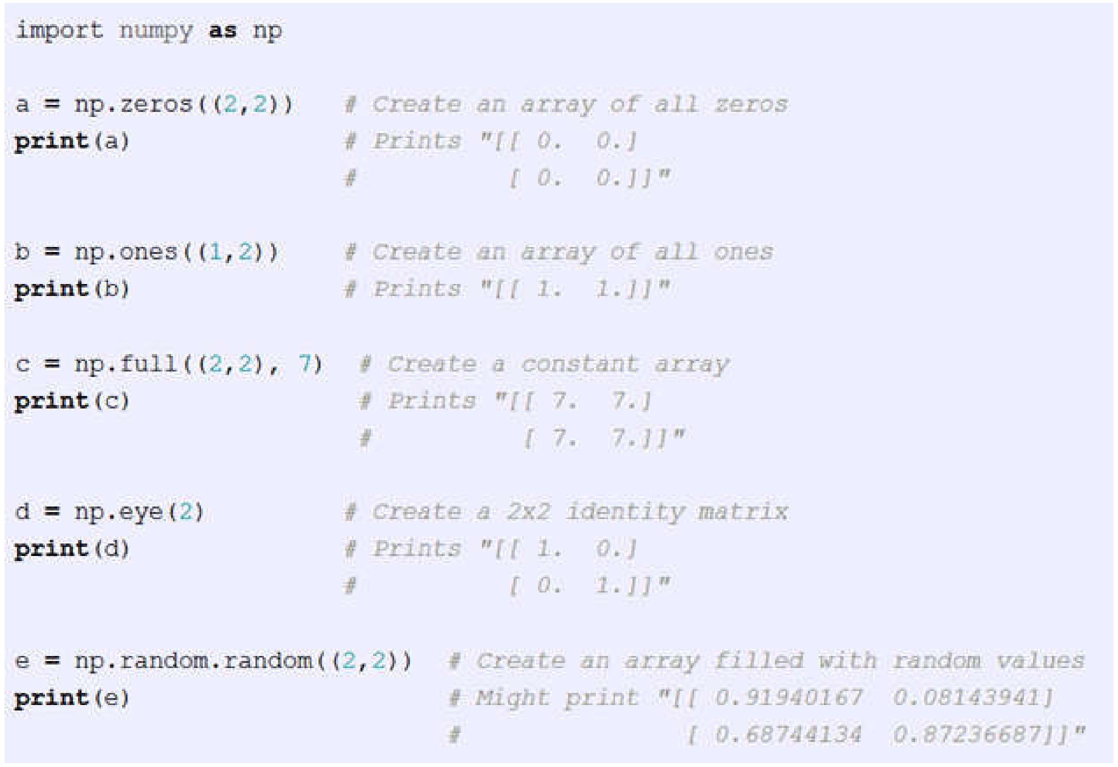
```
b = np.array([[1,2],[3,4]])
print(b)
```

Array Attributes

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print(a.shape)

a.shape = (3,2)
print(a)

b = a.reshape(2,3)
print(b)
```

```
import numpy as np
a = np.arange(12)
print(a)
print(a.ndim)
print(a.shape)

b = a.reshape(2,3,2)
print(b)
print(b.ndim)
print(b.shape)
```

Numpy also provides many functions to create arrays

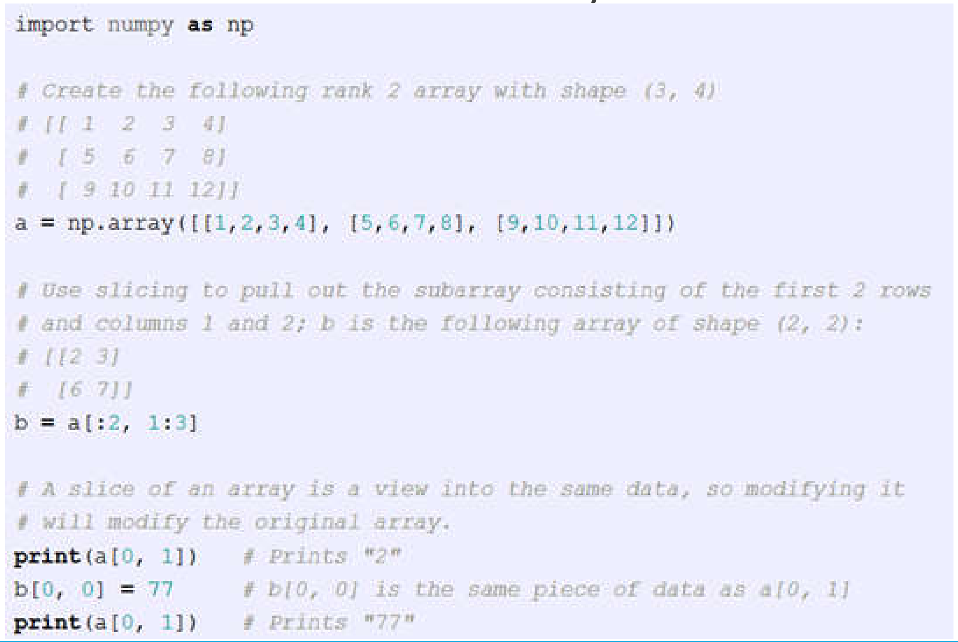


```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"

d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #               [ 0.68744134  0.87236687]]"
```

Array indexing

Numpy offers several ways to index into arrays.

Slicing: Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:

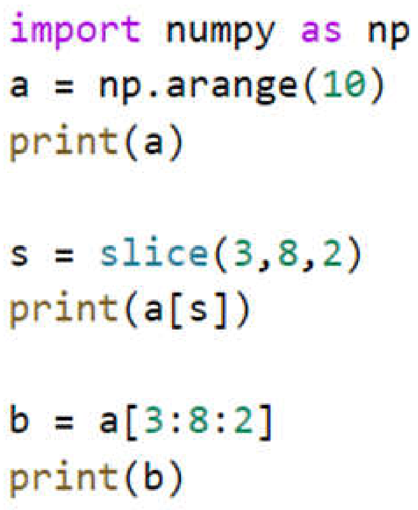


```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

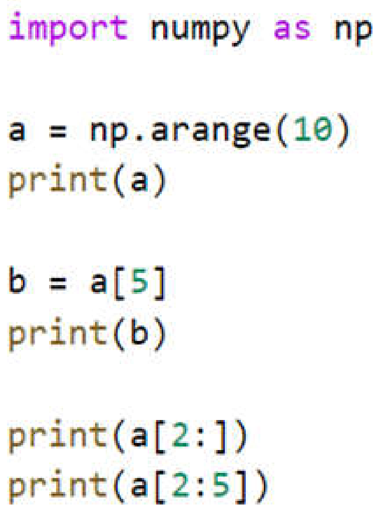
Array Slicing

Basic slicing is an extension of Python's basic concept of slicing to n dimensions.

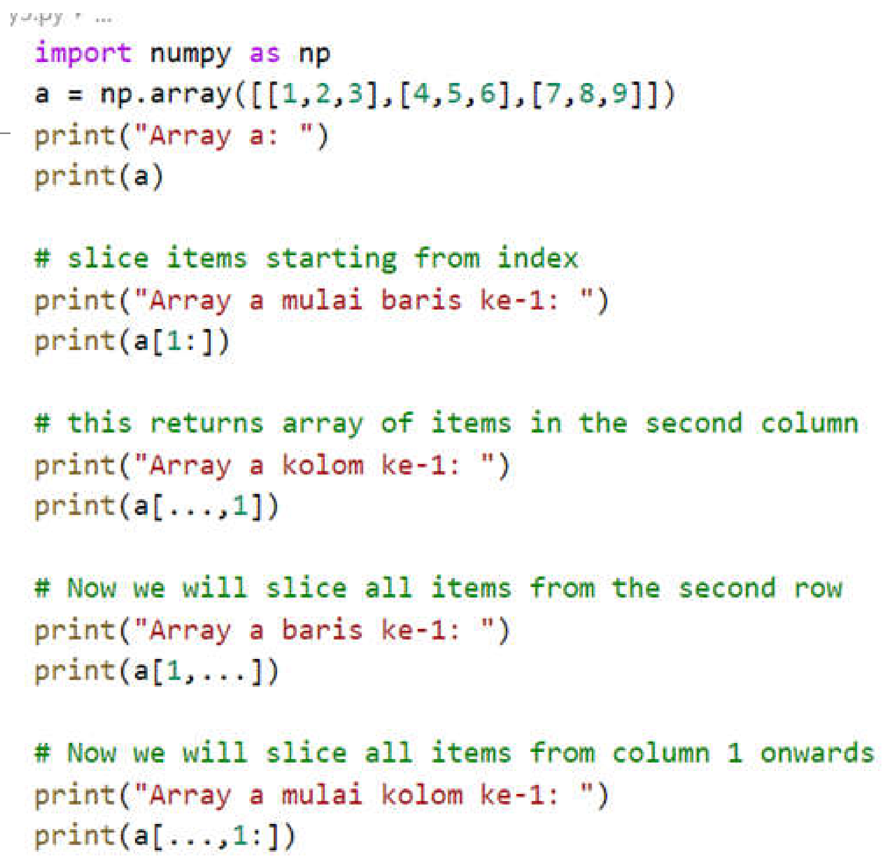
A Python slice object is constructed by giving **start, stop**, and **step** parameters to the built-in **slice** function.



```
import numpy as np
a = np.arange(10)
print(a)

s = slice(3,8,2)
print(a[s])

b = a[3:8:2]
print(b)
```



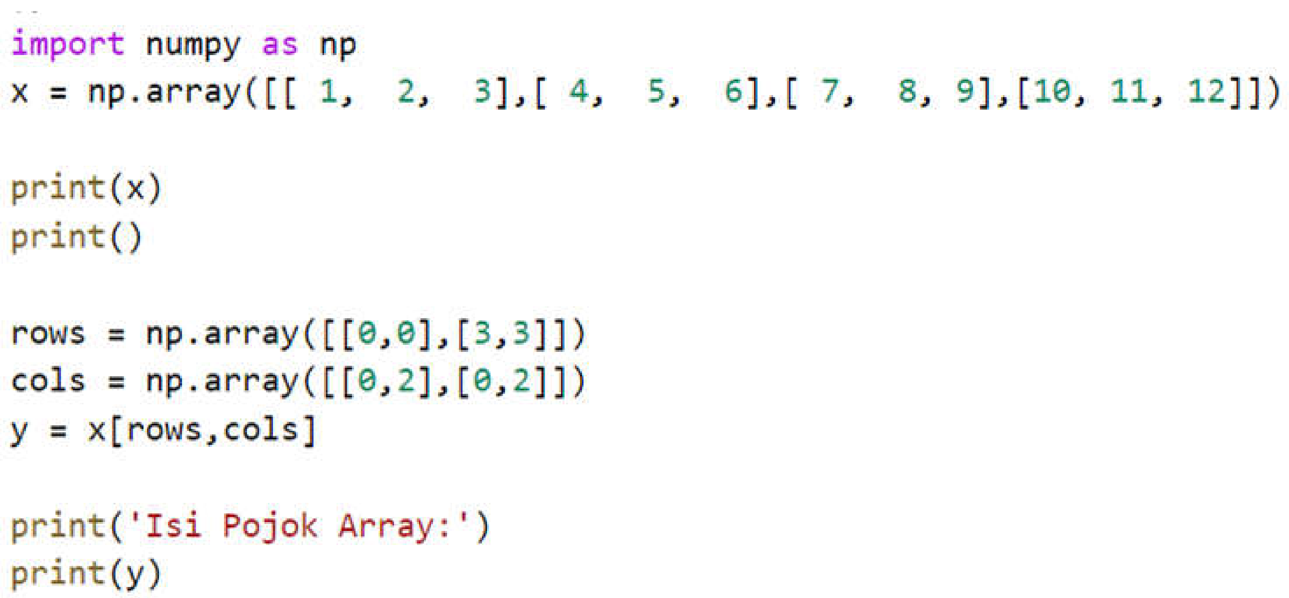
```
import numpy as np

a = np.arange(10)
print(a)

b = a[5]
print(b)

print(a[2:])
print(a[2:5])
```



y5.py › ...

```python
import numpy as np
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
print("Array a: ")
print(a)

# slice items starting from index
print("Array a mulai baris ke-1: ")
print(a[1:])

# this returns array of items in the second column
print("Array a kolom ke-1: ")
print(a[...,1])

# Now we will slice all items from the second row
print("Array a baris ke-1: ")
print(a[1,...])

# Now we will slice all items from column 1 onwards
print("Array a mulai kolom ke-1: ")
print(a[...,1:])
```

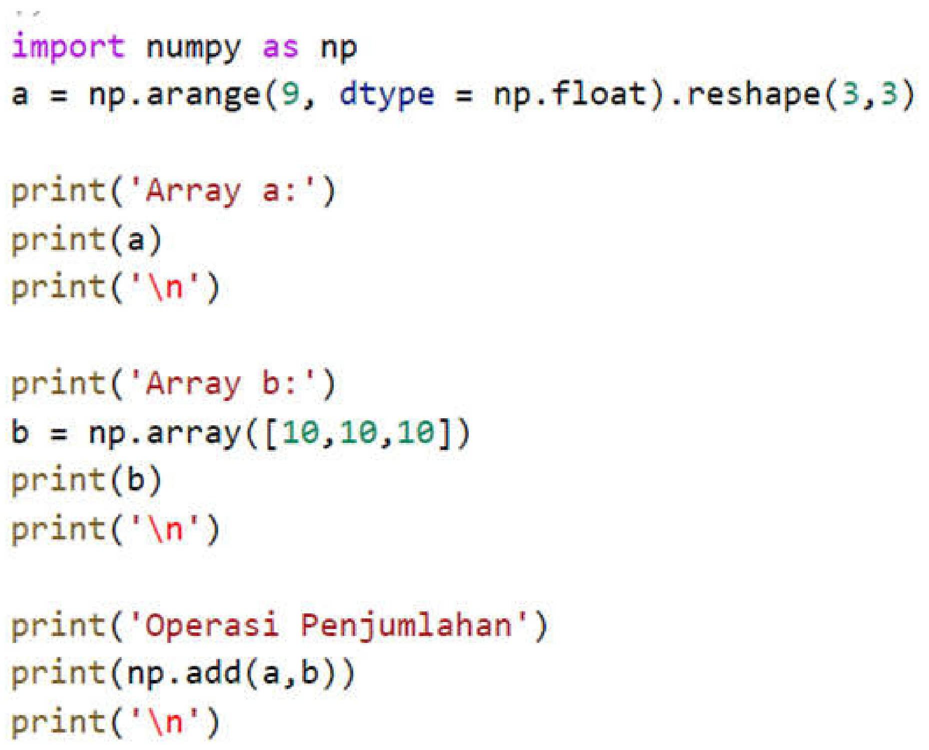
Advanced Indexing



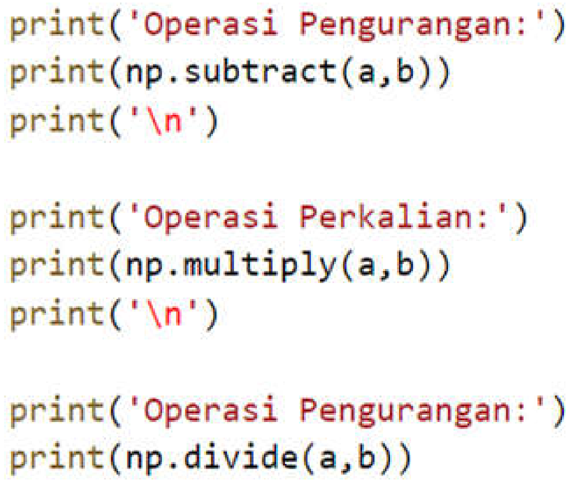
```
import numpy as np
x = np.array([[ 1,  2,  3],[ 4,  5,  6],[ 7,  8, 9],[10, 11, 12]])

print(x)
print()

rows = np.array([[0,0],[3,3]])
cols = np.array([[0,2],[0,2]])
y = x[rows,cols]

print('Isi Pojok Array:')
print(y)
```

```python
import numpy as np
x = np.array([[ 1,  2,  3],[ 4,  5,  6],[ 7,  8, 9],[10, 11, 12]])

z = x[1:4,1:3]
print(z)
print('\n')

# using advanced index for column
y = x[1:4,[1,2]]
print(y)

# boolean index
print(x[x > 3])
```

Arithmetic Operations

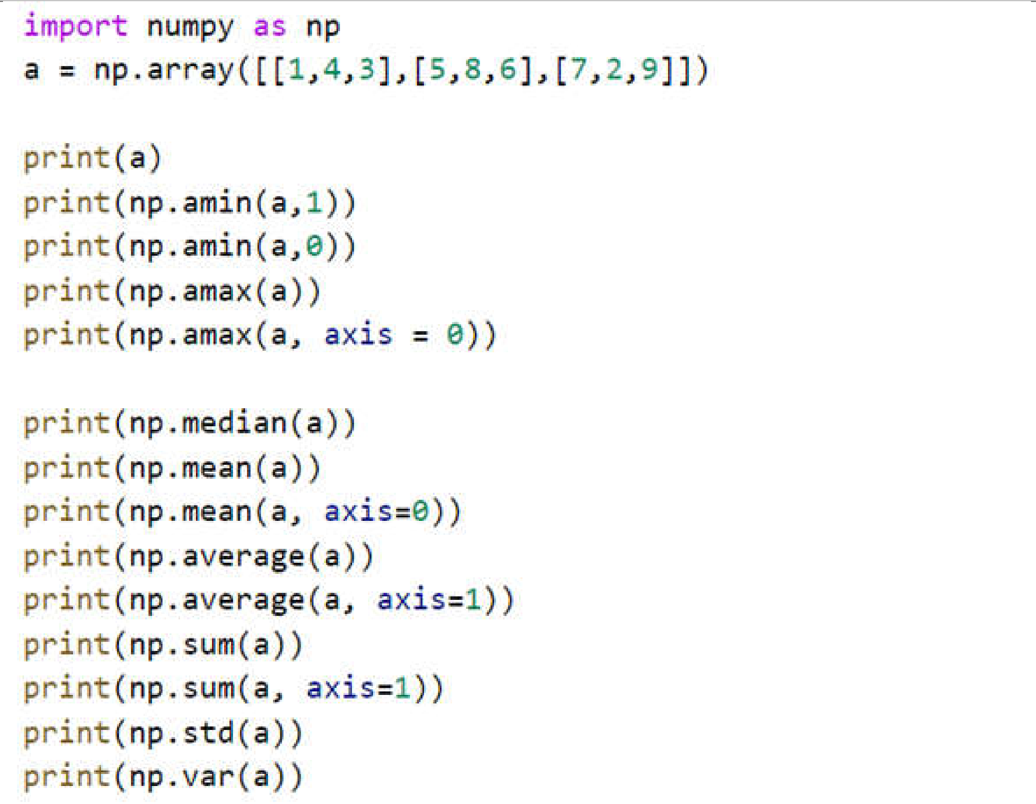


```
import numpy as np
a = np.arange(9, dtype = np.float).reshape(3,3)

print('Array a:')
print(a)
print('\n')

print('Array b:')
b = np.array([10,10,10])
print(b)
print('\n')

print('Operasi Penjumlahan')
print(np.add(a,b))
print('\n')
```



```
print('Operasi Pengurangan:')
print(np.subtract(a,b))
print('\n')

print('Operasi Perkalian:')
print(np.multiply(a,b))
print('\n')

print('Operasi Pengurangan:')
print(np.divide(a,b))
```

Statistics Functions



```python
import numpy as np
a = np.array([[1,4,3],[5,8,6],[7,2,9]])

print(a)
print(np.amin(a,1))
print(np.amin(a,0))
print(np.amax(a))
print(np.amax(a, axis = 0))

print(np.median(a))
print(np.mean(a))
print(np.mean(a, axis=0))
print(np.average(a))
print(np.average(a, axis=1))
print(np.sum(a))
print(np.sum(a, axis=1))
print(np.std(a))
print(np.var(a))
```

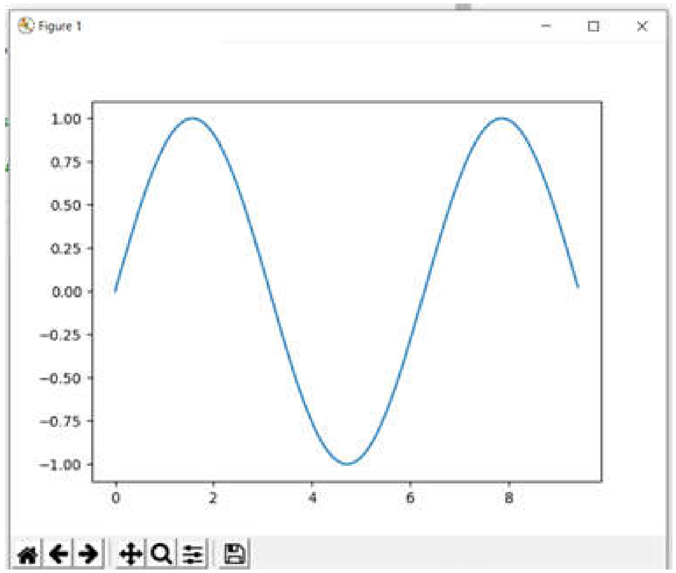
Matplotlib

Matplotlib is the most popular Python library for producing plots and other twodimensional data visualizations.

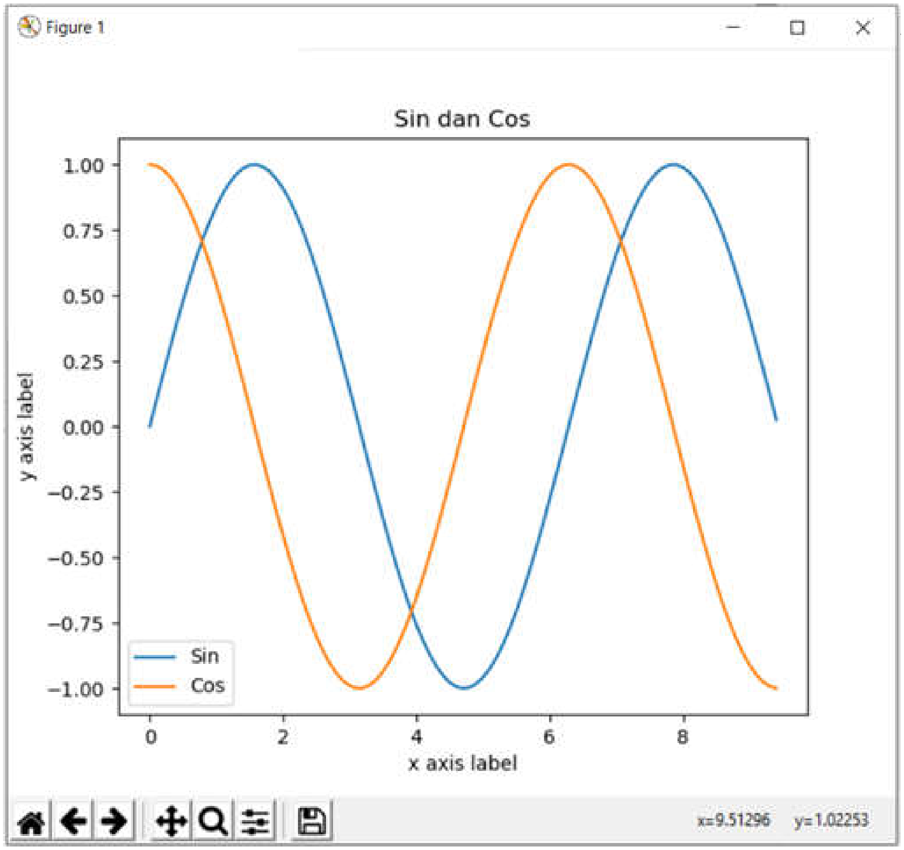
It was originally created by John D. Hunter and is now maintained by a large team of developers.

It is designed for creating plots suitable for publication.

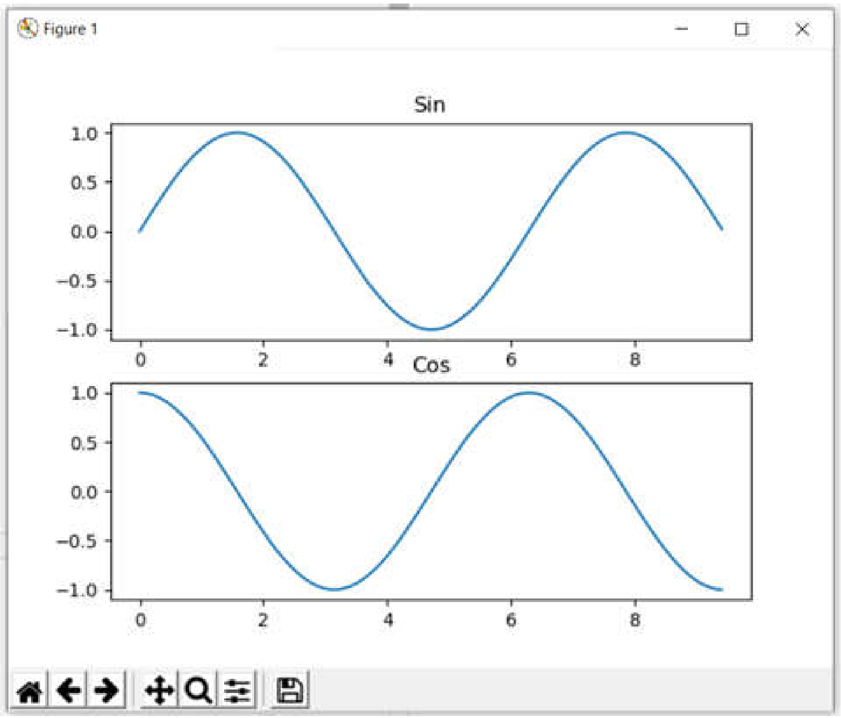
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to make graphics appear.
```



```python
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sin dan Cos')
plt.legend(['Sin', 'Cos'])
plt.show()
```



```python
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sin')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cos')

# Show the figure.
plt.show()
```



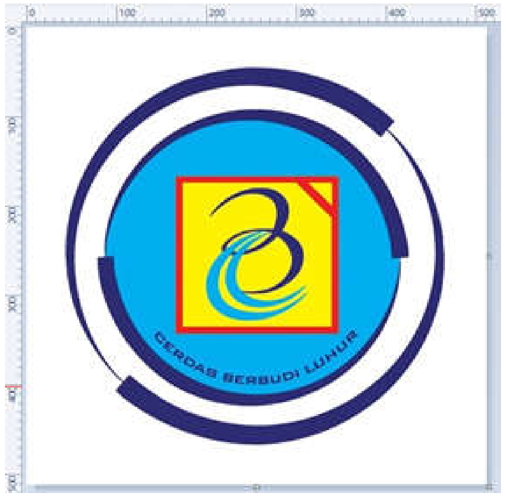
Scipy

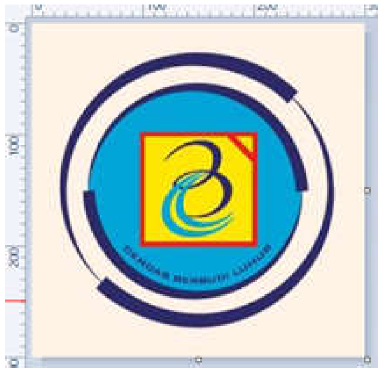
Scipy is a collection of packages addressing a number of different standard problem domains in scientific computing

Numpy provides a high-performance multidimensional array and basic tools to compute with and manipulate these arrays.

Scipy builds on this, and provides a large number of functions that operate on numpy arrays and are useful for different types of scientific and engineering applications.

```
from scipy.misc import imread, imsave, imresize

# Read an JPEG image into a numpy array
img = imread('ubl.jpg')
print(img.dtype, img.shape)  # Prints "uint8 (512, 512, 3)"

# We can tint the image by scaling each of the color channels
# by a different scalar constant. The image has shape (512, 512, 3);
# we multiply it by the array [1, 0.95, 0.9] of shape (3,);
# numpy broadcasting means that this leaves the red channel unchanged,
# and multiplies the green and blue channels by 0.95 and 0.9
# respectively.
img_tinted = img * [1, 0.95, 0.9]

# Resize the tinted image to be 300 by 300 pixels.
img_tinted = imresize(img_tinted, (300, 300))

# Write the tinted image back to disk
imsave('ubl_tinted.jpg', img_tinted)
```





```
import numpy as np
from scipy.misc import imread, imresize
import matplotlib.pyplot as plt

img = imread('ubl.jpg')
img_tinted = img * [1, 0.95, 0.9]

# Show the original image
plt.subplot(1, 2, 1)
plt.imshow(img)

# Show the tinted image
plt.subplot(1, 2, 2)

# A slight gotcha with imshow is that it might give strange results
# if presented with data that is not uint8. To work around this, we
# explicitly cast the image to uint8 before displaying it.
plt.imshow(np.uint8(img_tinted))
plt.show()
```

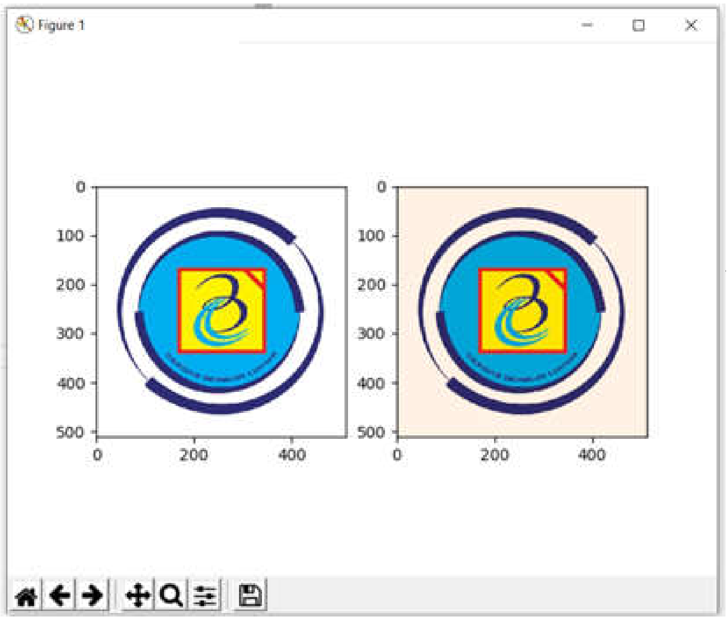


Figure 1
0
100
200
300
400
500
0
200
400
0
100
200
300
400
500
0
200
400