CS112
Introduction to Python Programming

Session 10: NumPy and SciPy

Shengwei Hou
Ph.D., Assistant Professor
Department of Ocean Science and Engineering
Fall 2022

Contents

Southern University of Science and Technology

Introduction

- NumPy is a Python library.
- NumPy is used for working with arrays.
- NumPy is short for "Numerical Python".
- SciPy is a scientific computation library that uses NumPy underneath.
- SciPy stands for Scientific Python.

Introduction

- SciPy is a Python library of mathematical routines
- Many of the SciPy routines are Python “wrappers” that is, Python routines that provide a Python interface, for numerical libraries and routines originally written in Fortran, C, or C++
- Because the Fortran, C, or C++ code that Python accesses is compiled, these routines typically run very fast
- SciPy makes extensive use of NumPy arrays, so NumPy should be imported with SciPy

Check package version

SUSTech Southern University of Science and Technology

In:
```
import numpy
import scipy

print(numpy.__version__)
print(scipy.__version__)
```

Out:
```
1.20.3
1.7.1
```

Note: two underscore ("_") characters are used in `__version__`.

Array Slicing

- In 2D, the first dimension corresponds to **rows**, the second to **columns**.
- for multidimensional a, a[0] is interpreted by taking all elements in the unspecified dimensions.
- for each dimension, slicing follows the [start:end:step] format. By default, start is 0, end is the last and step is 1.

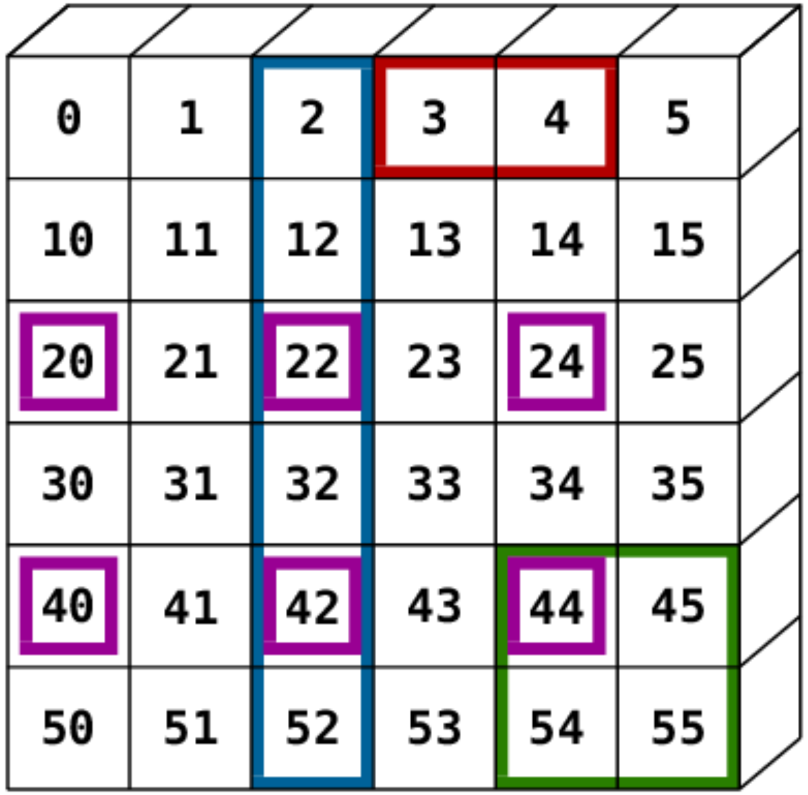
```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5,
6, 7, 8, 9])
>>> a[2:9:3]
array([2, 5, 8])
>>> a[:4]
array([0, 1, 2, 3])
```

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



NumPy data types

By default, Python has these data types:

- `string` - used to represent text data, the text is given under quotation marks. e.g. `"ABCD"`
- `integer` - used to represent integer numbers. e.g. `-1, -2, -3`
- `float` - used to represent real numbers. e.g. `1.2, 42.42`
- `boolean` - used to represent `True` or `False`.
- `complex` - used to represent complex numbers. e.g. `1.0 + 2.0j, 1.5 + 2.5j`

https://www.w3schools.com/python/numpy

NumPy data types

NumPy has some extra data types, and refer to data types with one character

- **`i`** - integer
- **`b`** - boolean
- **`u`** - unsigned integer
- **`f`** - float
- **`c`** - complex float
- **`m`** - timedelta
- **`M`** - datetime
- **`O`** - object
- **`S`** - string
- **`U`** - unicode string
- **`V`** - fixed chunk of memory for other types (void)

Check data type of an existing array

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

```
int64
```

https://www.w3schools.com/python/numpy

NumPy data types

Convert data type of an existing array

* for boolean index.

```
import numpy as np

arr = np.array([1.1, 2.1, 3.1])

new_arr = arr.astype('i')
                      integer.

print(new_arr)
print(new_arr.dtype)
```

```
[1 2 3]
int32
```

```
import numpy as np

arr = np.array([1, 0, 3])

new_arr = arr.astype(bool)
                          ≠0 ⇒ True
print(new_arr)             =0 ⇒ False
print(new_arr.dtype)
```

```
[True False True]
bool
```

https://www.w3schools.com/python/numpy

NumPy data types

Complex:	`>>> d = np.array([1+2j, 3+4j, 5+6*1j])`
	`>>> d.dtype`
	`dtype('complex128')`

Bool:	`>>> e = np.array([True, False, False, True])`
	`>>> e.dtype`
	`dtype('bool')`

Strings:	`>>> f = np.array(['Bonjour', 'Hello', 'Hallo'])`
	`>>> f.dtype # <--- strings containing max. 7 letters`
	`dtype('S7')`

Much more:	• int32 • int64 • uint32 • uint64 unsigned → save memory

NumPy Array Copy vs View

SUSTech Southern University of Science and Technology

- The main difference between a copy and a view of an array is that: **the copy is a new array**, and **the view is just a view of the original array.**

- The copy owns the data, and any changes made to the copy will not affect the original array, and any changes made to the original array will not affect the copy.

point to the same address

- The view does not own the data, and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

https://www.w3schools.com/python/numpy

NumPy Array Copy vs View

COPY – change the new array

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
x[0] = 42

print(arr)
print(x)
```

```
[1  2  3  4  5]
[42  2  3  4  5]
```

VIEW – change the new array

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 42

print(arr)
print(x)
```

```
[42  2  3  4  5]
[42  2  3  4  5]
```

https://www.w3schools.com/python/numpy

NumPy Array Copy vs View

SUSTech Southern University of Science and Technology

Check if the array is a copy or a view

```
import numpy as np

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

x = arr.copy()
y = arr.view()

print(x.base)
print(y.base)
print(np.may_share_memory(x, y))
```

independent → None

depend on arr → [1, 2, 3, 4, 5]

arr

x, y ⇒ False

```
None
[1 2 3 4 5]
True
```

- `numpy.ndarray.base`: base object if memory is from some other object.
- The base of an array that owns its memory is `None`
- Slicing creates a view, whose memory is shared with the original array object
- You can use `np.may_share_memory()` to check if two arrays share the same memory block.

https://www.w3schools.com/python/numpy

NumPy Array Copy vs View

SUSTech Southern University of Science and Technology

Views

```
A = np.array([[0,1,2],[3,4,5],[6,7,8]])
B = A # A and B reference the _same_ object
A is B
```

```
True
```

```
B[0,0] = 1000
A
```

```
array([[1000,  1,  2],
       [  3,  4,  5],
       [  6,  7,  8]])
```

http://mscbio2025.csb.pitt.edu/notes/numpy.slides.html

NumPy Array Copy vs View

SUSTech Southern University of Science and Technology

Sliced Views

```
row = A[1,:]
row
```

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

```
row[2] = 5000
A
```

```
array([[1000, 1, 2],
       [ 3,    4, 5000],
       [ 6,    7, 8]])
```

http://mscbio2025.csb.pitt.edu/notes/numpy.slides.html

NumPy Array Copy vs View

SUSTech Southern University of Science and Technology

Explicit Copy

```
new_mat = A.copy()
new_mat[0,0] = 0
A
```

independent memory of array

```
array([[1000,    1,    2],
       [   3,    4, 5000],
       [   6,    7,    8]])
```

```
new_mat
```

```
array([[   0,    1,    2],
       [   3,    4, 5000],
       [   6,    7,    8]])
```

http://mscbio2025.csb.pitt.edu/notes/numpy.slides.html

NumPy Array Copy vs View

Southern University of Science and Technology

Advanced Slices Copy

```
A = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
B = A[A>4]   # Boolean indexing creates copies not views
B
```

```
array([5, 6, 7, 8])
```

```
A
```

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

NumPy arrays can be indexed with slices, but also with boolean or integer arrays (**masks**). This method is called *fancy indexing*. It creates **copies not views**.

http://mscbio2025.csb.pitt.edu/notes/numpy.slides.html

NumPy Searching Arrays

SUSTech Southern University of Science and Technology

To search an array, use the `np.where()` method.

```
import numpy as np

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

```
(array([3, 5, 6]),)
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
```

```
(array([1, 3, 5, 7]),)
```

```
np.where(condition, output if True, output if False)
```

https://www.w3schools.com/python/numpy

```
where(...)
    where(condition, [x, y])

    Return elements chosen from `x` or `y` depending on
`condition`.

    Returns
    -------
    out : ndarray
        An array with elements from `x` where `condition`
is True, and elements from `y` elsewhere.
```

```
def sinc(x):
    if x == 0.0:
        y = 1.0
    else:
        y = np.sin(x)/x
    return y
```

→ avoid ZeroDivisionError

```
-----------------------------------------
np.where(x == 0.0, 1.0, np.sin(x)/x)
```

All and Any

axis=0 ⇒ column; axis=1 ⇒ row

SUSTech Southern University of Science and Technology

The `np.any()` and `np.all()` methods.

```
>>> np.any([True, True, False])
True
>>> np.any([[True, False],
     [False, False]], axis=0)
array([ True, False])
>>> np.any([-1, 0, 5])
True
>>> np.any(np.nan)
True
```

- `numpy.any()` tests whether any array element along a given axis evaluates to `True`, returns a single boolean value (`axis=None`), or an array of boolean values (axis is given).
- Not a Number (NaN), positive infinity and negative infinity will be evaluated to be `True` because these are not equal to zero.

```
>>> np.all([True, True, False])
False
>>> np.all([[True, False],
     [True, True]], axis=0)
array([ True, False])
>>> np.all([-1, 4, 5])
True
```

- `numpy.all()` tests whether all array elements along a given axis evaluate to `True`, returns a single boolean value (`axis=None`), or an array of boolean values (axis is given).
- Not a Number (NaN), positive infinity and negative infinity evaluate to `True` because these are not equal to zero.

NumPy Saving/Loading Data

SUSTech Southern University of Science and Technology

`genfromtxt` (and the simpler `loadtxt`) will read in deliminated files.

```
import numpy as np
np.genfromtxt('./csv_file.csv')
```

```
array([nan, nan, nan])
```

```
np.genfromtxt('./csv_file.csv', delimiter=',')
```

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

csv_file.csv

```
1,2,3
4,5,6
7,8,9
```

http://mscbio2025.csb.pitt.edu/notes/numpy.slides.html

NumPy Saving/Loading Data

- Binary files have two notable advantages over text-based files: file size and read/write speeds
- In NumPy, files can be accessed in binary format using `numpy.save` and `numpy.load`:

```
>>> import numpy as np
>>> data = np.empty((1000, 1000))
>>> np.save('test.npy', data)
>>> np.savetxt('test.txt', data)
>>> newdata = np.load('test.npy')
```

File size:

```
-rw-r--r--  1 shengwei  staff   7.6M Nov 12 16:09 test.npy
-rw-r--r--  1 shengwei  staff    24M Nov 12 16:09 test.txt
```

NumPy Random

- Random numbers are widely used in science and engineering computations
- The basic idea of a random number generator is that it should be able to produce a sequence of numbers that are distributed according to some predetermined distribution functions
- NumPy provides a number of such random number generators in its library `numpy.random`

NumPy Random

SUSTech Southern University of Science and Technology

Generate a random integer from 0 to 100:

```
from numpy import random

x = random.randint(100)
print(x)
```

```
98
```

Random numbers are expected to be different from time to time

Generate a random float from 0 to 1:

```
x = random.rand()
print(x)
```

```
0.7895916740359221
```

https://www.w3schools.com/python/numpy

NumPy Random

Generate random arrays

```
from numpy import random

x = random.randint(100, size=(3, 5))
print(x)
```

```
[[90 99 11 30 34]
 [66 40 63 36 37]
 [63 35 89 51 58]]
```

```
x = random.rand(2, 2)
print(x)
```

```
[[0.13273287 0.92268245]
 [0.05374344 0.98251619]]
```

Random numbers are expected to be different from time to time

https://www.w3schools.com/python/numpy

NumPy Random

Pick a random number from an array

```
from numpy import random

x = random.choice([3, 5, 7, 9])
print(x)
```

```
3
```

```
x = random.choice([3, 5, 7, 9],
size=(2, 2))
print(x)
```

```
[[5 5]
 [3 3]]
```

The `random.choice()` method takes an array as a parameter and randomly returns one of its values.

Add a `size` parameter to specify the shape of the array.

Random numbers are expected to be different from time to time

https://www.w3schools.com/python/numpy

NumPy Random

SUSTech Southern University of Science and Technology

Generate shuffling or permutation of arrays

```
from numpy import random
import numpy as np

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

```
[4 2 3 5 1]
```

```
print(random.permutation(arr))
```

```
[1 5 4 2 3]
```

The `random.shuffle()` method modify a sequence **in-place** by shuffling its contents. It makes changes to the original array.

The `random.permutation()` method returns a randomly re-arranged array (and leaves the original array unchanged).

Random numbers are expected to be different from time to time

https://www.w3schools.com/python/numpy

NumPy Random

SUSTech Southern University of Science and Technology

Normal distribution

正态分布

```
from numpy import random

x = random.normal(size=(2, 3))
print(x)
```

```
[[ 2.2799698 -1.51394603 2.07008094]
[ 0.03266912 1.27996989 1.1790585 ]]
```

```
x = random.normal(loc=10, scale=2,
size=(2, 3))
print(x)
```

```
[[13.19029955 10.19232563 11.56913201]
 [10.96922729 10.10848335  7.53244539]]
```

Arguments of `random.normal`:

`loc` - (Mean) where the peak of the bell exists, default=0.

`scale` - (Standard Deviation) how flat the graph distribution should be, default=1.

`size` - The shape of the returned array.

Random numbers are expected to be different from time to time

https://www.w3schools.com/python/numpy

NumPy Random

SUSTech Southern University of Science and Technology

Uniform distribution

```
from numpy import random

x = random.uniform(size=(2, 3))
print(x)
```

```
[[0.44566407 0.38663387 0.00408744]
 [0.6377317  0.19637125 0.51735068]]
```

```
x = random.uniform(low=1, high=2,
size=(2, 3))
print(x)
```

```
[[1.29679823 1.3713605  1.91737559]
 [1.40184556 1.9386972  1.62989992]]
```

Arguments of `random.uniform`:

`low` - lower bound – default=0.

`high` - upper bound – default=1.

`size` - The shape of the returned array.

Random numbers are expected to be different from time to time

https://www.w3schools.com/python/numpy

NumPy Random

SUSTech Southern University of Science and Technology

```
random.binomial(n=10, p=0.5, size=10)         # Binomial distribution
random.poisson(lam=2, size=10)                # Poisson distribution
random.logistic(loc=1, scale=2, size=(2, 3)) # Logistic distribution
random.multinomial(n=3, pvals=[1/3, 1/3, 1/3]) # Multinomial dist.
random.exponential(scale=2, size=(2, 3)). # Exponential distribution
random.chisquare(df=2, size=(2, 3))           # Chi-square distribution
random.rayleigh(scale=2, size=(2, 3))         # Rayleigh distribution
random.pareto(a=2, size=(2, 3))               # Pareto distribution
random.zipf(a=2, size=(2, 3))                 # Zipf distribution
```

You will know more after you learn statistics!

https://www.w3schools.com/python/numpy

NumPy Set Operations

```
import numpy as np

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

```
[1 2 3 4 5]
```

Unique:
Returns the sorted unique elements of an array.

```
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])
new_arr = np.union1d(arr1, arr2)
print(new_arr)
```

union1d ⇒ sorted input array

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

Union:
Return the unique, sorted array of values that are in either of the two input arrays.

https://www.w3schools.com/python/numpy

NumPy Set Operations

```
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])

new_arr = np.intersect1d(arr1, arr2)
print(new_arr)
```

```
[3 4]
```

Intersection:
Find the intersection of two arrays. Return the sorted, unique values that are in both of the input arrays.

```
new_arr = np.setdiff1d(arr1, arr2)
print(new_arr)
```

```
[1 2]
```

Difference:
Find the set difference of two arrays. Return the unique values in `arr1` that are not in `arr2`.

https://www.w3schools.com/python/numpy

NumPy Set Operations

SUSTech Southern University of Science and Technology

```
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])

new_arr = np.setxor1d(arr1, arr2)
print(new_arr)
```

```
[1 2 5 6]
```

Symmetric difference:
Find the set exclusive-or of two arrays. Return the sorted, unique values that are in only one (not both) of the input arrays.

```
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([3, 4, 5, 6])

print(np.in1d(arr1, arr2))
```

```
[False False  True  True]
```

Membership test:
Test whether each element of a 1-D array is also present in a second array. Returns a boolean array the same length as \`arr1\` that is True where an element of \`arr1\` is in \`arr2\` and False otherwise.

https://www.w3schools.com/python/numpy

SciPy Constants

SUSTech Southern University of Science and Technology

- As SciPy is more focused on scientific implementations, it provides many built-in scientific constants.
- These constants can be helpful when you are working with Data Science.

```
from scipy import constants

print(constants.pi)
```

```
3.141592653589793
```

```
# List all constants
print(dir(constants))
```

https://www.w3schools.com/python/scipy

SciPy sub-modules

Southern University of Science and Technology

- SciPy is composed of task-specific sub-modules.
- They all depend on NumPy, which should be imported first.

scipy.cluster	Vector quantization / Kmeans
scipy.constants	Physical and mathematical constants
scipy.fftpack	Fourier transform
scipy.integrate	Integration routines
scipy.interpolate	Interpolation
scipy.io	Data input and output
scipy.linalg	Linear algebra routines
scipy.ndimage	n-dimensional image package
scipy.odr	Orthogonal distance regression
scipy.optimize	Optimization
scipy.signal	Signal processing
scipy.sparse	Sparse matrices
scipy.spatial	Spatial data structures and algorithms
scipy.special	Any special mathematical functions
scipy.stats	Statistics

SciPy & NumPy for Linear Algebra

- NumPy and SciPy have extensive tools for numerically solving problems in linear algebra
- The SciPy package for linear algebra is `scipy.linalg`

SciPy & NumPy for Linear Algebra

SUSTech Southern University of Science and Technology

```
>>> import scipy.linalg
>>> import numpy as np

>>> a = np.array([[-2, 3], [4, 5]])
>>> scipy.linalg.det(a)
-22.0

>>> b = scipy.linalg.inv(a)
>>> b
array([[-0.22727273,  0.13636364],
       [ 0.18181818,  0.09090909]])

>>> np.dot(a, b)
array([[1., 0.],
       [0., 1.]])
```

Matrix determinant

The determinant of a square matrix is a value derived arithmetically from the coefficients of the matrix. The determinant for a 3x3 matrix, for example, is computed as follows::

a b c
d e f = A
g h i

det(A) = a*e*i + b*f*g + c*d*h - c*e*g - b*d*i - a*f*h

Matrix inversion:

Compute the inverse of a matrix.

Matrix production:

Compute dot product of two arrays.

SciPy & NumPy for Linear Algebra

SUSTech Southern University of Science and Technology

```
>>> a.T
 [[-2  4]
 [ 3  5]]

>>> np.diag(a)
[-2  5]

>>> np.diag(np.diag(a))
 [[-2  0]
 [ 0  5]]

>>> np.trace(a)
3
```

Matrix transpose

Matrix diagonal:
Extract a diagonal or construct a diagonal array.

1d array to square matrix

Matrix trace:
Return the sum along diagonals of the array.

Solving systems of linear equations

- Solving systems of equations is nearly as simple as constructing a coefficient matrix and a column vector
- Suppose having the following system of linear equations to solve:

$$\begin{cases} 2x_1 + 4x_2 + 6x_3 = 4 \\ x_1 - 3x_2 - 9x_3 = -11 \\ 8x_1 + 5x_2 - 7x_3 = 1 \end{cases} \rightarrow \mathbf{A} = \begin{bmatrix} 2 & 4 & 6 \\ 1 & -3 & -9 \\ 8 & 5 & -7 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 4 \\ -11 \\ 1 \end{bmatrix}$$

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

Solving systems of linear equations

$$\begin{cases} 2x_1 + 4x_2 + 6x_3 = 4 \\ x_1 - 3x_2 - 9x_3 = -11 \\ 8x_1 + 5x_2 - 7x_3 = 1 \end{cases} \longrightarrow \mathbf{A} = \begin{bmatrix} 2 & 4 & 6 \\ 1 & -3 & -9 \\ 8 & 5 & -7 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 4 \\ -11 \\ 1 \end{bmatrix}$$

```
>>> A = np.array([[2,4,6], [1,-3,-9], [8,5,-7]])
>>> b = np.array([4, -11, 2])
>>> scipy.linalg.solve(A, b)
array([-8.91304348, 10.2173913 , -3.17391304])
>>> np.dot(scipy.linalg.inv(A), b)
array([-8.91304348, 10.2173913 , -3.17391304])
```

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

Using `scipy.linalg.solve` is faster and numerically more stable than $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$

Exercises

Exercise 1

SUSTech Southern University of Science and Technology

From 2 numpy arrays, extract the indexes in which the elements in the 2 arrays match

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

Exercise 2

Find indices of non-zero elements from [1,2,0,0,4,0]

Exercise 3

SUSTech Southern University of Science and Technology

Create an 8x8 matrix and fill it with a checkerboard pattern.

Expected output:

```
[[0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]]
```

Exercise 4

Write a program that can place 3 ones randomly into a 4x4 array of zeros (each position has the same probability).

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

```
[[0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]]
```

```
[[0. 1. 1. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]]
```

```
[[0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]]
```

Exercise 5

Southern University of Science and Technology

Write a function to simulate the random walk of a drunk:

(1) starting at position 0

(2) with steps of 1 and -1 occurring with equal probability

(3) 100 steps in total

Repeat the random walk 100 times and report the average position of these random walks.

Exercise 6

Solve the following equation with SciPy/Numpy

$$\begin{aligned} 2a + b + c &= 4 \\ a + 3b + 2c &= 5 \\ a &= 6 \end{aligned}$$