NumPy Practice Set for Beginners

August 17, 2025

1 Introduction

This document provides a comprehensive guide to NumPy, a fundamental Python library for numerical computing. It is designed for beginners and covers array creation, indexing, operations, and more, based on a set of Jupyter notebooks. Each section includes code examples and brief descriptions to illustrate key concepts.

2 NumPy Array Creation

Creating arrays is the foundation of working with NumPy. This section demonstrates how to create 1D and 2D arrays, generate sequences, and initialize arrays with zeros, ones, or random values.

2.1 Creating Arrays from Lists

Converting Python lists to NumPy arrays is straightforward using `np.array()`.

```
import numpy as np
my_list = [1, 3, 4]
np.array(my_list)
# Output: array([1, 3, 4])
my_newlist = [[1, 3, 4], [5, 6, 7], [12, 15, 16]]
np.array(my_newlist)
# Output: array([[ 1,  3,  4],
#                [ 5,  6,  7],
#                [12, 15, 16]])
```

Description: The `np.array()` function converts lists into NumPy arrays, enabling efficient numerical operations. It supports both 1D and 2D arrays.

2.2 Generating Sequences with arange

The `np.arange()` function creates arrays with evenly spaced values.

```
np.arange(0, 10)
# Output: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
np.arange(0, 12)
# Output: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

Description: `np.arange(start, stop)` generates values from `start` to `stop-1`, ideal for creating sequential data.

2.3 Arrays of Zeros and Ones

Initialize arrays with zeros or ones using `np.zeros()` and `np.ones()`.

```
np.zeros(3)
# Output: array([0., 0., 0.])
np.zeros((4, 4))
```

```
# Output: array([[0., 0., 0., 0.],
#                [0., 0., 0., 0.],
#                [0., 0., 0., 0.],
#                [0., 0., 0., 0.]])
np.ones((3, 4))
# Output: array([[1., 1., 1., 1.],
#                [1., 1., 1., 1.],
#                [1., 1., 1., 1.]])
```

Description: These functions create arrays filled with zeros or ones, useful for initializing matrices.

2.4 Linearly Spaced Arrays

The `np.linspace()` function generates evenly spaced numbers over a specified interval.

```
np.linspace(0, 3, 10)
# Output: array([0.        , 0.33333333, 0.66666667, 1.        , 1.33333333,
#                1.66666667, 2.        , 2.33333333, 2.66666667, 3.        ])
```

Description: `np.linspace(start, stop, num)` creates `num` evenly spaced points, including both endpoints.

2.5 Identity Matrix

Create an identity matrix using `np.eye()`.

```
np.eye(3)
# Output: array([[1., 0., 0.],
#                [0., 1., 0.],
#                [0., 0., 1.]])
```

Description: `np.eye(n)` generates an $n \times n$ identity matrix with ones on the diagonal.

2.6 Random Arrays

Generate random arrays with `np.random.rand()` (uniform distribution) and `np.random.randn()` (normal distribution).

```
np.random.rand(5)
# Output: array([0.62889776, 0.45770933, 0.47638155, 0.96017348, 0.28636821])
np.random.rand(5, 5)
# Output: array([[0.5517618 , 0.69878984, 0.25766145, 0.45690737, 0.57660424],
#                [0.3898689 , 0.96778987, 0.47648464, 0.66548908, 0.32394625],
#                ...])
np.random.randn(3, 3)
# Output: array([[-0.62523838, -0.26794496,  0.6786926 ],
#                [ 0.43584346, -1.37228438, -0.76440182],
#                [-0.78496211,  0.67717777, -1.84900016]])
```

Description: These functions are useful for simulations and testing with random data.

2.7 Random Integers

Generate random integers with `np.random.randint()`.

```
np.random.randint(1, 88)
# Output: 82
np.random.randint(1, 56, 8)
# Output: array([42, 18, 35, 29, 26, 41, 11, 38], dtype=int32)
```

Description: `np.random.randint(low, high, size)` generates random integers, useful for discrete data.

3 1D Array Indexing

This section covers indexing and slicing 1D NumPy arrays, including the behavior of array slices.

3.1 Basic Indexing and Slicing

Access elements or slices of a 1D array using indices.

```
arr = np.arange(0, 12)
arr
# Output: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
arr[8]
# Output: 8
arr[1:5]
# Output: array([1, 2, 3, 4])
arr[:6]
# Output: array([0, 1, 2, 3, 4, 5])
arr[5:]
# Output: array([ 5,  6,  7,  8,  9, 10, 11])
```

Description: Use `arr[i]` for single elements and `arr[start:stop]` for slices. Omitting `start` or `stop` implies the beginning or end of the array.

3.2 Modifying Slices

Slices are views of the original array, so modifications affect the original data.

```
slice_of_arr = arr[0:6]
slice_of_arr
# Output: array([0, 1, 2, 3, 4, 5])
slice_of_arr[:] = 77
slice_of_arr
# Output: array([77, 77, 77, 77, 77, 77])
arr
# Output: array([77, 77, 77, 77, 77, 77,  6,  7,  8,  9, 10, 11])
```

Description: Modifying a slice changes the original array since slices are views, not copies.

4 2D Array Indexing

This section explores indexing and slicing in 2D NumPy arrays, including boolean indexing.

4.1 Accessing Elements and Rows

Access specific elements or entire rows in a 2D array.

```
arr_2d = np.array([[5, 10, 15], [20, 25, 30], [35, 40, 45]])
arr_2d
# Output: array([[ 5, 10, 15],
#                [20, 25, 30],
#                [35, 40, 45]])
arr_2d[0]
# Output: array([ 5, 10, 15])
arr_2d[0][0]
# Output: 5
arr_2d[1, 1]
# Output: 25
```

Description: Use $\texttt{arr}_2d[row][col]or$