Class #13: Data Analysis with NumPy

1. Brief introduction to NumPy.
2. Installing NumPy.
3. Working with NumPy arrays.
4. Using built-in NumPy methods.
5. Array methods and attributes.
6. Indexing and slicing arrays.
7. Broadcasting in NumPy.
8. Array layouts.
9. Boolean masking.
10. Performing arithmetic operations with NumPy.
11. Using universal functions (ufuncs).
12. Overview of exercises for practice.
13. Solutions to exercises. **Assignment #13: Class #**

1. Brief introduction to NumPy.

NumPy (Numerical Python) হলো একটি জনপ্রিয় পাইথন লাইব্রেরি যা গাণিতিক এবং বৈজ্ঞানিক গণনার জন্য ব্যবহৃত হয়। এটি প্রধানত বড় আকারের ডেটা এবং গাণিতিক গণনার কাজ সহজ এবং দ্রুত করতে সাহায্য করে।

NumPy-র কিছু প্রধান সুবিধা:

1. **এ্যারেগুলির কাজ:** NumPy মূলত "array" নামে একটি ডেটা স্ট্রাকচার ব্যবহার করে। এই এ্যারে গুলোতে একসাথে অনেক ধরনের সংখ্যা রাখা যায় এবং এগুলোর উপর সহজে গাণিতিক হিসাব করা যায়।
2. **দ্রুত গাণিতিক হিসাব:** NumPy পাইথনের সাধারণ তালিকা (list) এর চেয়ে অনেক দ্রুত গণনা করতে পারে।
3. **বহুমাত্রিক সমর্থন:** NumPy দিয়ে একক মাত্রা থেকে শুরু করে অনেকগুলো মাত্রার (যেমন 2D, 3D ম্যাট্রিক্স) অ্যারে তৈরি করা সম্ভব।

এই কারণে, বিজ্ঞানী, ইঞ্জিনিয়ার, ডেটা বিশ্লেষকরা NumPy ব্যবহার করেন তাদের গণনা সহজ এবং দ্রুত করতে।

বৈজ্ঞানিক গণনার জন্য একটি মৌলিক প্যাকেজ হিসেবে, **NumPy** গণিত, বিজ্ঞান, ইঞ্জিনিয়ারিং এবং ডেটা সায়েন্স প্রোগ্রামিংয়ের জন্য Python ইকো-সিস্টেমে ভিত্তি প্রদান করে। **NumPy**-এর প্রধান অবজেক্ট হলো **সমজাতীয় মাল্টিডাইমেনশনাল অ্যারে**।

এই লেকচারে, আমরা NumPy-এর গুরুত্বপূর্ণ কিছু ধারণা এবং বিল্ট-ইন ফাংশনগুলো নিয়ে আলোচনা করব, যেগুলো আমরা পরবর্তী সেকশনে বারবার ব্যবহার করব।

2. Installing NumPy.

NumPy ইনস্টল করার জন্য, আপনি Python এর প্যাকেজ ম্যানেজার pip ব্যবহার করতে পারেন। নিচে ধাপে ধাপে নির্দেশনা দেয়া হলো:

1. পাইথন ইনস্টল করা

প্রথমে নিশ্চিত হয়ে নিন যে আপনার সিস্টেমে Python ইনস্টল করা রয়েছে।

2. NumPy ইনস্টল করা

এখন NumPy ইনস্টল করতে, টার্মিনাল বা কমান্ড প্রম্পট এ এই কমান্ডটি লিখুন:

```
pip3 install numpy
```

Anaconda তে আগে থেকেই numpy থাকে সেজন্য, numpy install করতে হয় না ।

3. Working with NumPy arrays.

NumPy অ্যারে (arrays) হলো একই ধরনের (homogeneous) ডেটার একটি সজ্জিত সংগ্রহ। একটি NumPy অ্যারে তৈরি করতে হলে, সাধারণত numpy.array() মেথড ব্যবহার করা হয়। এই অ্যারে গুলি সাধারণ Python লিস্টের মতোই, তবে NumPy অ্যারেগুলি আরো দ্রুত এবং দক্ষভাবে গণনা পরিচালনা করতে সক্ষম।

Numpy Arrays

NumPy arrays will be the main concept that we will be using in this course. These arrays essentially come in two flavors:

- **Vectors:** Vectors are strictly 1-dimensional array
- **Matrices:** Matrices are 2-dimensional (matrix can still have only one row or one column).

NumPy ইনস্টল করার পর, আমরা এটি ব্যবহার করতে চাইলে প্রথমে Python স্ক্রিপ্টে import numpy কমান্ডটি ব্যবহার করে NumPy মডিউলটি আমদানি (import) করি। NumPy একটি **মডিউল** (module) বা **ফাইল** যা পাইথনের জন্য বিভিন্ন গণনা এবং অ্যারে সম্পর্কিত কাজ সহজ করে। তবে, NumPy নামটি অনেক বড়, তাই সাধারণত আমরা এটি সংক্ষিপ্ত আকারে ব্যবহার করার জন্য **এটি np নামে এলাইস** (alias) করে থাকি।

```
import numpy as np
```

NumPy has many built-in functions and capabilities. We will focus on some of the most important and key concepts of this powerful library.

Python list data type:

```
my_list = [-1,0,1]
print(my_list)
print(my_list[0])
print(type(my_list))
```

```
[-1, 0, 1]
-1
<class 'list'>
```

To create a NumPy array, from a Python data structure (লিস্ট, টাপল, সেট, ডিশনারি), we use NumPy's array function.
The NumPy's array function can be accessed by typing "np.array".
We need to cast our Python data structure, my_list, as a parameter to the array function.

```
import numpy as np
my_list = [-1,0,1]
my_array = np.array(my_list)
print(my_array)
print(my_array[0])
print(type(my_array))
```

```
[-1  0  1]
-1
<class 'numpy.ndarray'>
```

n dimensional array = ndarray

```
import numpy as np
# Lets create and cast a list of list to generate 2-D array
nested_list = [[1,2,3],[4,5,6],[7,8,9],[4, 5, 9]]
print(f"nested_list= {nested_list}")
twoD_array = np.array(nested_list)
print(f"twoD_array= {twoD_array}")
```

```
nested_list= [[1, 2, 3], [4, 5, 6], [7, 8, 9], [4, 5, 9]]
twoD_array= [[1 2 3]
 [4 5 6]
 [7 8 9]
 [4 5 9]]
```

Nested list = 2d NumPy array. Comma missing in the NumPy Array

We can use Tuple instead of list as well.

```
import numpy as np
my_tuple = (-1,0,1)
```

```
my_array = np.array(my_tuple)
print(f"tuple to array= {my_array} and data type=
{type(my_array)}")
```

```
tuple to array= [-1  0  1] and data type= <class 'numpy.ndarray'>
```

4. Using built-in NumPy methods.

Numpy Arrays: arange(), linspace(), zeros(), ones(), eye(), rand(), randn(), randint()

NumPy এর বিল্ট-ইন মেথড ব্যবহার করে অ্যারে তৈরি করা

প্রায়ই আমরা NumPy এর বিল্ট-ইন মেথডগুলো ব্যবহার করে অ্যারে তৈরি করি, কারণ এগুলি অনেক সহজ এবং দ্রুত।

arange(): **arange()** ফাংশনটি Python এর range() ফাংশনের মতো কাজ করে।

সিনট্যাক্স: arange([start,] stop[, step,], dtype=None)

- **start:** (ঐচ্ছিক) অ্যারের প্রথম মান (ডিফল্ট হল 0)।
- **stop:** অ্যারের শেষ মান (অন্তর্ভুক্ত হবে না)।
- **step:** (ঐচ্ছিক) মানগুলোর মধ্যে ব্যবধান (ডিফল্ট হল 1)।
- **dtype:** (ঐচ্ছিক) অ্যারের ডেটা টাইপ।

ফলাফল: range() ফাংশনের মতন কাজ করে, একটি অ্যারে প্রদান করে।

উদাহরণ:

```
import numpy as np

# 0 থেকে 9 পর্যন্ত, step 1 সহ, 10 বাদে মানের অ্যারে তৈরি করা (range() এর মতো)
n = np.arange(0, 10)
print(n)  # আউটপুট হবে: [0 1 2 3 4 5 6 7 8 9]

# 0 থেকে 10 পর্যন্ত, step 2 সহ মানের অ্যারে তৈরি করা
n = np.arange(0, 11, 2)
print(n)  # আউটপুট হবে: [0 2 4 6 8 10]

# 0 থেকে 10 পর্যন্ত, step 2 সহ, ডেটা টাইপ float সহ মানের অ্যারে তৈরি করা
n = np.arange(0, 10, 2, dtype=float)
print(n)  # আউটপুট হবে: [0. 2. 4. 6. 8.]
```

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

ব্যাখ্যা:

1. **np.arange(0, 10):**

- এটি 0 থেকে শুরু করে 10 এর আগে পর্যন্ত মানের একটি অ্যারে তৈরি করে। যেমন, range(0, 10) এর মতো কাজ করে, তবে range() একটি লিস্ট দেয়, আর arange() একটি NumPy অ্যারে তৈরি করে।
- আউটপুট: [0 1 2 3 4 5 6 7 8 9]

2. **np.arange(0, 11, 2):**
   - এখানে, 0 থেকে শুরু করে 11 এর আগে পর্যন্ত, 2 ব্যবধানে মানের অ্যারে তৈরি করা হয়েছে।
   - আউটপুট: [0 2 4 6 8 10]
3. **np.arange(0, 10, 2, dtype=float):**
   - এখানে, 0 থেকে শুরু করে 10 এর আগে পর্যন্ত, 2 ব্যবধানে মানের অ্যারে তৈরি করা হয়েছে, এবং dtype=float ব্যবহৃত হয়েছে, যার ফলে অ্যারের উপাদানগুলি ফ্লোট (float) ধরনের হবে।
   - আউটপুট: [0. 2. 4. 6. 8.]

এই কোডটি NumPy এর arange() ফাংশনের ব্যবহার প্রদর্শন করে, যা Python এর range() এর মতো কাজ করে, তবে এটি আরো দ্রুত এবং দক্ষভাবে গণনা পরিচালনা করতে সক্ষম।

linspace() ফাংশনটি একটি নির্দিষ্ট পরিসরে (range) সমানভাবে বিরত সংখ্যা তৈরি করে। অর্থাৎ, যখন আপনি কোনো নির্দিষ্ট শুরুর এবং শেষের মান দেবেন, তখন এটি সেই পরিসরে সমান ব্যবধানে কিছু মান তৈরি করে দেয়। উদাহরণস্বরূপ, যদি আপনি 0 থেকে 10 এর মধ্যে 5টি মান চান, তবে linspace() ফাংশনটি 0 থেকে 10 পর্যন্ত সমান ব্যবধানে 5টি মান দিবে। এর মধ্যে ব্যবধান (step size) স্বয়ংক্রিয়ভাবে হিসাব করা হয়।

সিনট্যাক্স:

```
numpy.linspace(start, stop, num=50, retstep=False)
```

- **start:** শুরুর মান।
- **stop:** শেষ মান।
- **num:** (ঐচ্ছিক) মোট টুকরোগুলির সংখ্যা যা আপনি চান (ডিফল্ট হল 50)।
- **retstep:** (ঐচ্ছিক) যদি True থাকে, তাহলে ফাংশনটি ব্যবধান (step size) সহ একটি টাপল রিটার্ন করবে (ডিফল্ট হল False)।

উদাহরণ:

```
import numpy as np

# 0 থেকে 10 পর্যন্ত 5টি সমানভাবে বিরত মানের অ্যারে তৈরি করা
arr1 = np.linspace(0, 10, 5)
print(arr1)

# 0 থেকে 10 পর্যন্ত 5টি সমানভাবে বিরত মানের অ্যারে তৈরি করা, step সহ
arr3 = np.linspace(0, 10, 5, retstep=True)
print(arr3)
```

```
# 0 থেকে 10 পর্যন্ত 4টি সমানভাবে বিরত মানের অ্যারে তৈরি করা, stop মানটি অন্তর্ভুক্ত
arr2 = np.linspace(0, 10, 5, endpoint=False)
print(arr2)
```

```
[ 0.   2.5  5.   7.5 10. ]
(array([ 0. ,  2.5,  5. ,  7.5, 10. ]), 2.5)
[0. 2. 4. 6. 8.]
```

1. arr1 = np.linspace(0, 10, 5):

- **ব্যাখ্যা**: এখানে, linspace() ফাংশনটি 0 থেকে 10 পর্যন্ত 5টি সমানভাবে বিরত মান তৈরি করছে। endpoint=True (ডিফল্ট) থাকার কারণে, 10 অন্তর্ভুক্ত হবে।
- **আউটপুট**:
  - এটি 0 থেকে 10 পর্যন্ত সমানভাবে বিরত 5টি মান তৈরি করবে। 0 থেকে 10 পর্যন্ত 5টি মানের মধ্যে ব্যবধান হবে 2.5 (10-0 / 4 = 2.5)। ফলে মানগুলো হবে: 0, 2.5, 5, 7.5, 10।

2. arr3 = np.linspace(0, 10, 5, retstep=True):

- **ব্যাখ্যা**: এই ক্ষেত্রে retstep=True দেয়া হয়েছে, যার মানে হলো শুধু অ্যারের মানগুলিই নয়, বরং মানগুলোর মধ্যে ব্যবধান (step size) কেমন হবে, সেটিও রিটার্ন করা হবে।
- **আউটপুট**:
  - প্রথমে np.linspace() ফাংশনটি 0 থেকে 10 পর্যন্ত 5টি সমান মান তৈরি করবে, যেগুলোর মধ্যে ব্যবধান 2.5 হবে।
  - দ্বিতীয় অংশে, ব্যবধান (step) 2.5 রিটার্ন হবে, যা দুটি পরবর্তী মানের মধ্যে ব্যবধান।

3. arr2 = np.linspace(0, 10, 5, endpoint=False):

- **ব্যাখ্যা**: এখানে endpoint=False ব্যবহার করা হয়েছে, যার মানে হলো 10 অন্তর্ভুক্ত হবে না। 0 থেকে 10 পর্যন্ত 5টি মান তৈরি হবে, কিন্তু 10 বাদ দেয়া হবে।
- **আউটপুট**:
  - এখানে 5টি মান হবে: 0, 2, 4, 6, 8। কারণ endpoint=False থাকার ফলে 10 বাদ দেয়া হয়েছে।

সারাংশ:

- **endpoint=True** (ডিফল্ট): শেষ মান (এক্ষেত্রে 10) অন্তর্ভুক্ত থাকে।
- **retstep=True**: এটি ব্যবধান (step size) জানিয়ে দেয়, অর্থাৎ পরবর্তী দুইটি মানের মধ্যে কত ব্যবধান থাকবে।
- **endpoint=False**: শেষ মান (এক্ষেত্রে 10) অন্তর্ভুক্ত হয় না।

এভাবে, linspace() ফাংশনটি নির্দিষ্ট পরিসরে সমান বিরত মান তৈরি করতে এবং প্রয়োজনে ব্যবধান সম্পর্কে তথ্য জানাতে সাহায্য করে।

Don't Confuse!

- **arange() takes 3rd argument as step size.**
- **linspace() take 3rd argument as no of slice we want.**

zeros()

- We want to create an array with **all zeros**

zeros() ফাংশনটি NumPy লাইব্রেরির একটি বিল্ট-ইন ফাংশন যা একটি অ্যারে তৈরি করে, যার প্রতিটি উপাদান 0। এটি সাধারণত ব্যবহৃত হয় যখন আপনি এমন একটি অ্যারে তৈরি করতে চান যেখানে সকল উপাদান প্রাথমিকভাবে শূন্য (0) থাকবে।

সিনট্যাক্স:

```
numpy.zeros(shape, dtype=float)
```

- **shape**: এটি অ্যারের আকার বা আয়তন। এটি একটি একক সংখ্যা (যেমন 5) বা একটি টিউপল হতে পারে (যেমন (3, 4) বা (2, 3, 4))।
- **dtype**: (ঐচ্ছিক) আউটপুট অ্যারের ডেটা টাইপ। ডিফল্টভাবে এটি float হয়, তবে আপনি এটিকে int বা অন্য কোনো ধরনের করতে পারেন।

উদাহরণ:

একটি 1D অ্যারে তৈরি করা:

```
import numpy as np
arr = np.zeros(5)
print(arr)
```

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

এটি একটি 1D অ্যারে তৈরি করেছে, যার 5টি উপাদান আছে এবং প্রতিটি উপাদান 0। ডিফল্টভাবে, সব উপাদান ফ্লোট (float) হিসেবে থাকবে।

একটি 2D অ্যারে তৈরি করা:

```
import numpy as np
arr = np.zeros((3, 4))
print(arr)
```

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

এটি একটি 3x4 আকারের 2D অ্যারে তৈরি করেছে, যেখানে প্রতিটি উপাদান 0।

int ডেটা টাইপের সাথে অ্যারে তৈরি করা:

```
import numpy as np
arr = np.zeros(5, dtype=int)
print(arr)
```

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

এখানে, dtype=int ব্যবহার করা হয়েছে, যাতে অ্যারের উপাদানগুলি পূর্ণসংখ্যা (integer) হয়।

সারাংশ:

- **zeros()** ফাংশনটি ব্যবহৃত হয় একটি অ্যারে তৈরি করতে, যেখানে সব উপাদান 0 হয়।
- এটি বিভিন্ন আকারের (shape) অ্যারে তৈরি করতে সহায়তা করে, এবং আপনি চাইলে ডেটা টাইপও নির্ধারণ করতে পারেন।

ones()

- We want to create an array with **all ones**

```
import numpy as np
n=np.ones(3)
print(f"one dimentional 0 array: {n}")
n = np.ones((4,6)) #(no_row, no_col) passing a tuple
print(f"two dimentional 0 array:\n {n}")
```

```
one dimentional 0 array: [1. 1. 1.]
two dimentional 0 array:
 [[1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]]
```

eye()

Creates an identity matrix must be a square matrix, which is useful in several linear algebra problems.

- Return a 2-D array with **ones on the diagonal and zeros elsewhere.**

একটি **identity matrix** এমন একটি স্কয়ার মেট্রিক্স (square matrix) যা ডায়াগোনাল (diagonal) বা কোণাকুণি গুলোতে 1 এবং বাকি সব জায়গায় 0 থাকে।

সিনট্যাক্স:

```
numpy.eye(N, dtype=float)
```

N: এটি মেট্রিক্সের সারির সংখ্যা (rows)। এটি একটি পূর্ণসংখ্যা হতে হবে।

dtype: (ঐচ্ছিক) আউটপুট অ্যারের ডেটা টাইপ। ডিফল্টভাবে এটি float হয়, তবে আপনি এটি int বা অন্য যেকোনো ধরনের করতে পারেন।

উদাহরণ:

একটি 5x5 স্কয়ার আইডেন্টিটি ম্যাট্রিক্স তৈরি করা:

```
import numpy as np
np.eye(5)
```

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

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

এটি একটি 5x5 স্কয়ার ম্যাট্রিক্স তৈরি করেছে, যেখানে ডায়াগোনাল গুলোতে 1 এবং বাকি সব জায়গায় 0 রয়েছে।

ডেটা টাইপ int দিয়ে একটি ম্যাট্রিক্স তৈরি করা:

```
import numpy as np
np.eye(5,dtype=int)
```

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

এখানে, dtype=int ব্যবহার করা হয়েছে, যাতে আউটপুটটি পূর্ণসংখ্যা (integer) ধরনের হয়।

সারাংশ: eye() ফাংশনটি একটি স্কয়ার আইডেন্টিটি ম্যাট্রিক্স তৈরি করে, যেখানে ডায়াগোনাল (কোণাকুণি) গুলোতে 1 এবং বাকি সব জায়গায় 0 থাকে।

Random

We can also create arrays with random numbers using Numpy's built-in functions in Random module.

rand()

Create an array of the given shape and populate it with random samples from a uniform distribution over [0, 1). মান ০ থেকে ১ এর চেয়ে কম।

rand() ফাংশনের সিনট্যাক্স:

```
numpy.random.rand(d0, d1, ..., dn)
```

d0, d1, ..., dn: এগুলি হচ্ছে অ্যারের বিভিন্ন মাত্রার আকার (shape)

```
import numpy as np
p = np.random.rand(3) # 1-D array with three elements
print(f"1-D array with three elements: {p}")
print()
p = np.random.rand(3,2) # row, col, note we are not passing a
tuple here, each dimension as a separate argument
print(f"2-D array with 3 row & 2 column: \n{p}")
```

```
1-D array with three elements: [0.4989456  0.9949083  0.41243656]

2-D array with 3 row & 2 column:
[[0.91818264 0.90109184]
 [0.72196254 0.7739256 ]
```

```
 [0.33404053 0.38518517]]
```

randn()

Return a sample (or samples) from the "standard normal" or a "Gaussian" distribution. Unlike rand which is uniform.

np.random.randn() ফাংশনটি NumPy এর **random** মডিউলের একটি ফাংশন যা **"standard normal"** বা **"Gaussian"** বণ্টন থেকে এলোমেলো নমুনা তৈরি করে।

```
import numpy as np

# Create a 1D array with 5 random numbers
arr1 = np.random.randn(5)
print("1D array (5 random numbers):")
print(arr1)

# Create a 2D array with shape 3x4
arr2 = np.random.randn(4, 5)
print("\n2D array (4x5 shape):")
print(arr2)

# Customize data type (convert random numbers to integers)
arr3 = np.random.randn(5).astype(int)
print("\nCustomize data type (convert to integers):")
print(arr3)
```

```
1D array (5 random numbers):
[ 0.03360147 -1.23310179 -0.51406974  0.30272439 -0.53984268]

2D array (4x5 shape):
[[ 0.23069266  0.24559833  1.8232182  -0.31464002  0.17463437]
 [ 0.73733389 -0.99339877  1.05118027  0.4737124   0.68070318]
 [ 0.96821596  0.74153555 -1.01012738  0.60667696 -0.42275535]
 [ 0.77005213 -0.31497435 -1.79784353 -0.39772501 -1.20904729]]

Customize data type (convert to integers):
[ 0 -1  0  0  0]
```

ব্যাখ্যা:

- **১ম উদাহরণ**: np.random.randn(5) 5টি এলোমেলো সংখ্যা তৈরি করেছে, যা গসিয়ান বণ্টন থেকে এসেছে।
- **২য় উদাহরণ**: np.random.randn(4, 5) 4x5 আকারের একটি 2D অ্যারে তৈরি করেছে, যেখানে প্রতিটি উপাদান গসিয়ান বণ্টন থেকে এলোমেলোভাবে এসেছে।
- **৩য় উদাহরণ**: astype(int) দিয়ে এলোমেলো সংখ্যাগুলিকে পূর্ণসংখ্যায় রূপান্তর করা হয়েছে।

এভাবেই **np.random.randn()** ফাংশনটি ব্যবহার করা হয় বিভিন্ন আকারের এলোমেলো নমুনা তৈরি করতে।

randint()

Return random integers from low (inclusive) to high (exclusive).

সাধারণ সিনট্যাক্স:

```
np.random.randint(low, high, size)
```

low: যে সংখ্যা থেকে শুরু হবে, সেটি (অন্তর্ভুক্ত)।

high: সর্বোচ্চ যে সংখ্যা পর্যন্ত যাবে, সেটি (অন্তর্ভুক্ত নয়)।

size: কতটি এলোমেলো সংখ্যা তৈরি করতে হবে।

```
import numpy as np

# Generate a single random integer between 1 (inclusive) and 100
(exclusive)
p = np.random.randint(1, 100)
print("Random integer between 1 and 100:", p)

# Generate an array of 10 random integers between 1 (inclusive)
and 500 (exclusive)
p = np.random.randint(1, 500, 10)
print("Array of 10 random integers between 1 and 500:", p)
```

```
Random integer between 1 and 100: 31
Array of 10 random integers between 1 and 500: [273 201 219 424
368 141 123 388 146 147]
```

5. Array methods and attributes.

Array Methods & Attributes

Some important Methods and Attributes are important to know:

Methods:

- reshape(), max(), min(), argmax(), argmin()

reshape(): The reshape() function returns a new array with the same data but a different shape.

```
import numpy as np

# arange() ফাংশন ব্যবহার করে 0 থেকে 20 পর্যন্ত একটি একমাত্রিক অ্যারে তৈরি করা হচ্ছে
array_arange1 = np.arange(21)
print(f"NumPy arange function: {array_arange1}\n")

# reshape() ফাংশন ব্যবহার করে একমাত্রিক অ্যারেটিকে 3x7 আকারের বহু-মাত্রিক অ্যারেতে রূপান্তর করা হচ্ছে
reshape_test1 = array_arange1.reshape(3, 7)
print(f"NumPy reshape function to convert one-dimensional array to multi-dimensional
array: \n {reshape_test1}")
```

```
NumPy arange function: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]

NumPy reshape function to convert one-dimensional array to multi-dimensional array:
[[ 0  1  2  3  4  5  6]
 [ 7  8  9 10 11 12 13]
 [14 15 16 17 18 19 20]]
```

এখানে:

1. **arange()** ফাংশন ব্যবহার করে 0 থেকে 20 পর্যন্ত একটি একমাত্রিক অ্যারে তৈরি করা হয়েছে।
2. **reshape()** ফাংশন দ্বারা ওই অ্যারেটিকে 3টি সারি এবং 7টি কলাম বিশিষ্ট একটি 2D অ্যারেতে রূপান্তর করা হয়েছে।

max() & min(): Useful methods for finding max or min values.

```
import numpy as np

# Generate 10 random integers between 0 and 100
array_ranint1 = np.random.randint(0, 100, 10)
print(f"Random values: {array_ranint1}")

# Find the maximum value
print(f"Maximum value: {array_ranint1.max()}")
# Find the minimum value
print(f"Minimum value: {array_ranint1.min()}")
```

```
Random values: [66 69 46 38 50 72 39 34 24 90]
Maximum value: 90
Minimum value: 24
```

ব্যাখ্যা দেওয়া হলো:

1. **np.random.randint(0, 100, 10)**: এই লাইনটি 0 থেকে 100 এর মধ্যে 10টি এলোমেলো পূর্ণসংখ্যা (random integers) তৈরি করে এবং array_ranint1 ভেরিয়েবলে সংরক্ষণ করে।
2. **array_ranint1.max()**: এটি array_ranint1 অ্যারের মধ্যে সর্বোচ্চ মান (maximum value) খুঁজে বের করে।
3. **array_ranint1.min()**: এটি array_ranint1 অ্যারের মধ্যে সর্বনিম্ন মান (minimum value) খুঁজে বের করে।

কোডটি এলোমেলো পূর্ণসংখ্যা তৈরি করে এবং এর মধ্যে সর্বোচ্চ এবং সর্বনিম্ন মান বের করে দেখায়।

argmax() & argmin(): To find the position number of max and min values in array.

```
import numpy as np

# Generate 10 random integers between 0 and 100
array_ranint1 = np.random.randint(0, 100, 10)
print(f"Random values: {array_ranint1}")

# Find the maximum value
print(f"Maximum value: {array_ranint1.max()}")
```

```
# index starts from 0
print(f"Index of maximum value: {array_ranint1.argmax()}")

# Find the minimum value
print(f"Minimum value: {array_ranint1.min()}")
# index starts from 0
print(f"Index of minimum value: {array_ranint1.argmin()}")
```

```
Random values: [24 62 87 84 95 34 31 26 55 88]
Maximum value: 95
Index of maximum value: 4
Minimum value: 24
Index of minimum value: 0
```

Attributes

- size, shape, dtype

```
import numpy as np

# Using arange() function to create a one-dimensional array from 0
to 20
array_arange1 = np.arange(21)
print(f"NumPy arange function: {array_arange1}\n")

# Size of the array
print(f"Size of the array: {array_arange1.size}")

# Type of the data
print(f"Data type of the array: {array_arange1.dtype}")
```

```
NumPy arange function: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13
14 15 16 17 18 19 20]

Size of the array: 21
Data type of the array: int32
```

shape & reshape: NumPy ব্যবহার করে একটি 1d অ্যারে তৈরি এবং তাকে বিভিন্ন shape রূপান্তর করার প্রক্রিয়া

```
import numpy as np
array_arange1 = np.arange(21)
print(f"NumPy arange function: {array_arange1}\n")

# Checking the shape of the array
print(f"Shape of the array: {array_arange1.shape}")

# Reshaping into valid shapes and checking the resulting shapes
# Reshaping into 3x7 (valid)
print(f"Reshaped to 3x7: {array_arange1.reshape(3, 7).shape}")
# Reshaping into 7x3 (valid)
print(f"Reshaped to 7x3: {array_arange1.reshape(7, 3).shape}")
```

```
# Attempting to reshape into 1x16 (this will raise an error since
16 elements are needed)
try:
    print(f"Reshaped to 1x16: {array_arange1.reshape(1,
16).shape}")
except Exception as e:
    print(f"Error reshaping to 1x16 = {e}")
```

```
NumPy arange function: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13
14 15 16 17 18 19 20]

Shape of the array: (21,)
Reshaped to 3x7: (3, 7)
Reshaped to 7x3: (7, 3)
Error reshaping to 1x16 = cannot reshape array of size 21 into
shape (1,16)
```

ব্যাখ্যা:

1. **np.arange(21):** এটি 0 থেকে 20 পর্যন্ত 21টি সংখ্যার একটি একমাত্রিক অ্যারে তৈরি করে।
2. **array_arange1.shape:** এটি অ্যারের আকার (shape) দেখাবে, যা (21,) হবে, অর্থাৎ এটি একটি একমাত্রিক অ্যারে এবং এর মধ্যে 21টি উপাদান রয়েছে।
3. **Reshape:**
   - **array_arange1.reshape(3, 7):** এটি অ্যারেটিকে 3 সারি ও 7 কলামে রূপান্তর করবে (এটি বৈধ, কারণ 3 * 7 = 21)।
   - **array_arange1.reshape(7, 3):** এটি অ্যারেটিকে 7 সারি ও 3 কলামে রূপান্তর করবে (এটি বৈধ, কারণ 7 * 3 = 21)।
4. **অবৈধ reshape:**
   - **array_arange1.reshape(1, 16):** এটি একটি অবৈধ রূপান্তর কারণ এখানে 16টি উপাদান প্রয়োজন, কিন্তু অ্যারে মোট 21টি উপাদান ধারণ করে।
   - এখানে **try-except** ব্যবহৃত হয়েছে, যা এর ফলে উত্থিত ত্রুটি (error) ক্যাচ করবে এবং এর বার্তা প্রদর্শন করবে।

সংক্ষেপে:

কোডটি 0 থেকে 20 পর্যন্ত 21টি সংখ্যার অ্যারে তৈরি করে, তারপর বিভিন্ন বৈধ ও অবৈধ আকারে রূপান্তর করে এবং আকারের ফলাফল দেখায়।

6. Indexing and slicing arrays.

Indexing & slicing of 1-D arrays (vectors): আমরা string এবং লিস্ট যেভাবে বিভিন্ন অংশ দেখেছি, সেভাবে দেখবো। main array একই থাকবে, কিন্তু output এ অংশ অংশ করে দেখবো।

একটি একমাত্রিক NumPy অ্যারে তৈরি করে এবং বিভিন্ন উপায়ে অ্যারের উপাদানগুলি অ্যাক্সেস, স্লাইস এবং পরিবর্তন করার প্রক্রিয়া

```
import numpy as np

# Creating a one-dimensional array with specified values
array_1d = np.array([-10, -2, 0, 2, 17, 106, 200])
print(f"Full Array: {array_1d}")

# Getting value at a certain index
print(f"Value at index 0: {array_1d[0]}")

# Getting a range of values
print(f"Values from index 0 to 2: {array_1d[0:3]}, Full array:
{array_1d}")

# Using negative index
print(f"Second last element (using negative index): {array_1d[-
2]}, Full array: {array_1d}")

# Using negative index for a range
print(f"Values from index 1 to second last element (exclusive):
{array_1d[1:-2]}, Full array: {array_1d}")

# Getting values up to and from a certain index (remember,
indexing starts from 0)
print(f"Values up to index 2: {array_1d[:2]}, Values from index 2
onward: {array_1d[2:]}")

# Assigning a new value to a certain index in the array
array_1d[0] = -102
print(f"Array after changing the first element to -102:
{array_1d}")
```

```
Full Array: [-10  -2   0   2  17 106 200]
Value at index 0: -10
Values from index 0 to 2: [-10  -2   0], Full array: [-10  -2   0   2  17 106 200]
Second last element (using negative index): 106, Full array: [-10  -2   0   2  17
106 200]
Values from index 1 to second last element (exclusive): [-2  0  2 17], Full array: [-
10  -2   0   2  17 106 200]
Values up to index 2: [-10  -2], Values from index 2 onward: [  0   2  17 106 200]
Array after changing the first element to -102: [-102   -2    0    2   17  106  200]
```

ব্যাখ্যা:

1. **অ্যারে তৈরি**: array_1d = np.array([-10, -2, 0, 2, 17, 106, 200]) — এই লাইনটি array_1d নামে একটি একমাত্রিক অ্যারে তৈরি করে যা -10 থেকে শুরু করে 200 পর্যন্ত কিছু সংখ্যার মান ধারণ করে।
2. **একটি নির্দিষ্ট সূচকের মান পাওয়া**: array_1d[0] — এটি প্রথম উপাদান (ইন্ডেক্স 0) প্রদান করে, যা -10।
3. **একটি রেঞ্জের মান পাওয়া**: array_1d[0:3] — এটি সূচক 0 থেকে 2 পর্যন্ত উপাদানগুলি প্রদান করে (3 অন্তর্ভুক্ত নয়), অর্থাৎ [-10, -2, 0]।
4. **নেগেটিভ সূচক ব্যবহার**: array_1d[-2] — এটি দ্বিতীয় শেষ উপাদান (106) প্রদান করে।
5. **নেগেটিভ সূচক সহ রেঞ্জ**: array_1d[1:-2] — এটি সূচক 1 থেকে -2 (দ্বিতীয় শেষ) পর্যন্ত উপাদানগুলি প্রদান করে, অর্থাৎ [-2, 0, 2, 17]।

6. **নির্দিষ্ট সূচক থেকে উপাদান নেওয়া**: array_1d[:2] — এটি সূচক 0 এবং 1-এর উপাদানগুলি দেয়, অর্থাৎ [-10, -2]। array_1d[2:] — এটি সূচক 2 থেকে শুরু করে বাকী সমস্ত উপাদান প্রদান করে, অর্থাৎ [0, 2, 17, 106, 200]।
7. **অ্যারের একটি নির্দিষ্ট সূচকে নতুন মান নির্ধারণ**: array_1d[0] = -102 — এটি প্রথম উপাদানটিকে -102-এ পরিবর্তন করে।

এই কোডটি দেখাচ্ছে কীভাবে একটি NumPy অ্যারে থেকে নির্দিষ্ট সূচক বা রেঞ্জের উপাদান সংগ্রহ করা যায় এবং একই সঙ্গে সূচকগুলি পরিবর্তন করা যায়।

Indexing & slicing 2-D arrays (matrices): Lets create an array with 24 elements using arange() and convert it to 2D matrix using "shape". *note, 6 x 4 = 24*

```
import numpy as np

# Create a one-dimensional array with values from 0 to 23
array_2d = np.arange(24)
print(f"Original 1D array:\n{array_2d}\n")

# Reshape the array into a 2D array with 6 rows and 4 columns
array_2d.shape = (6, 4)  # Alternatively, you could use
array_2d.reshape(6, 4)

# Display the reshaped array
print(f"Reshaped 2D array with 6 rows and 4 columns:\n{array_2d}")
```

```
Original 1D array:
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
22 23]

Reshaped 2D array with 6 rows and 4 columns:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

To access any element, the general format is:

array_2d[row][col] or **array_2d[row,col]**.

We will use [row,col], easier to use comma ',' for clarity.

```
import numpy as np

# Create a one-dimensional array with values from 0 to 23
array_2d = np.arange(24)
array_2d.shape = (6, 4)  # Alternatively, you could use
array_2d.reshape(6, 4)
```

```
print(f"2D Array:\n{array_2d}\n")

# To get a complete row (row with index 2)
print(f"Row at index 2: {array_2d[2]}")

# Using negative index to get the row (row with index -4, which is
the same as index 2)
print(f"Row at index -4 (same as index 2): {array_2d[-4]}")

# To get an individual element value at row = 5 and column = 2
print(f"Element at row 5, column 2: {array_2d[5, 2]}")

# Another way to access the same element
row, column = 5, 2
print(f"Element using ROW {row} and COLUMN {column}:
{array_2d[row, column]}")

# Just to make sure, using [row][col] syntax
print(f"Element using [row][col] syntax: {array_2d[5][2]}")

# Display the full 2D array again
print(f"Full 2D array again:\n{array_2d}")

# 2D array slicing
# Extracting the top-left 2x2 corner of the array
print(f"Top-left 2x2 corner of the array:\n{array_2d[:2, :2]}")
```

```
2D Array:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]

Row at index 2: [ 8  9 10 11]
Row at index -4 (same as index 2): [ 8  9 10 11]
Element at row 5, column 2: 22
Element using ROW 5 and COLUMN 2: 22
Element using [row][col] syntax: 22
Full 2D array again:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
Top-left 2x2 corner of the array:
[[0 1]
 [4 5]]
```

ব্যাখ্যা:

1. **একমাত্রিক অ্যারে তৈরি করা**: np.arange(24) ব্যবহার করে একটি একমাত্রিক (1D) অ্যারে তৈরি করা হয়েছে, যা 0 থেকে 23 পর্যন্ত মান ধারণ করে।
2. **অ্যারের আকার পরিবর্তন (reshape)**: array_2d.shape = (6, 4) এই লাইনটি অ্যারেটিকে 6টি সারি এবং 4টি কলামে রূপান্তরিত করেছে, ফলে এটি একটি 2D অ্যারে হয়ে উঠেছে।
3. **একটি পূর্ণ সারি পেতে**:
   - array_2d[2] ব্যবহার করে তৃতীয় সারিটি (index 2) নেওয়া হয়েছে।
   - array_2d[-4] ব্যবহার করে নেগেটিভ ইনডেক্সের মাধ্যমে একই সারি (index -4) নেওয়া হয়েছে, যেটি আসলে index 2 এর সমান।
4. **একটি নির্দিষ্ট উপাদান পাওয়া**: array_2d[5, 2] ব্যবহার করে ৫ম সারির (index 5) ২য় কলামের (index 2) উপাদান পাওয়া গেছে। এটি array_2d[5, 2] ও array_2d[5][2] দুইভাবেই একই ফলাফল দেয়।
5. **অ্যারের পূর্ণ আউটপুট দেখানো**: print(f"Full 2D array again:\n{array_2d}") দিয়ে পুরো 2D অ্যারের মান দেখানো হয়েছে।
6. **২D অ্যারে স্লাইসিং**: array_2d[:2, :2] ব্যবহার করে অ্যারের টপ-লেফ ২x২ অংশ নেওয়া হয়েছে। এটি প্রথম দুটি সারি এবং প্রথম দুটি কলামকে একত্রে প্রদর্শন করে।

7. Broadcasting in NumPy.

Broadcasting: Numpy arrays are different from normal Python lists because of their ability to broadcast. We will only cover the basics, for further details on broadcasting rules,

```
import numpy as np

# Create a 1D array with values from 0 to 9
array_1d = np.arange(0, 10)
print(f"Original 1D array:\n{array_1d}\n")

# Take a slice of the array and set it equal to 500 (broadcasting)
array_1d[0:5] = 500
print(f"1D array after broadcasting 500 to the first 5
elements:\n{array_1d}\n")

# Create a 2D matrix with ones (4x4 matrix)
array_2d = np.ones((4, 4))
print(f"Original 2D array (4x4 matrix of ones):\n{array_2d}\n")

# Broadcast 300 to the first row of the 2D array
array_2d[0] = 300
print(f"2D array after broadcasting 300 to the first
row:\n{array_2d}")
```

```
Original 1D array:
[0 1 2 3 4 5 6 7 8 9]

1D array after broadcasting 500 to the first 5 elements:
[500 500 500 500 500   5   6   7   8   9]
```

```
Original 2D array (4x4 matrix of ones):
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

2D array after broadcasting 300 to the first row:
[[300. 300. 300. 300.]
 [  1.   1.   1.   1.]
 [  1.   1.   1.   1.]
 [  1.   1.   1.   1.]]
```

ব্যাখ্যা:

1. **1D অ্যারে তৈরি করা:**
   - np.arange(0, 10) দ্বারা একটি একমাত্রিক (1D) অ্যারে তৈরি করা হয়েছে, যেটি 0 থেকে 9 পর্যন্ত মান ধারণ করে।
   - প্রথমে অ্যারেটি প্রিন্ট করা হয়েছে, যেমন: [0 1 2 3 4 5 6 7 8 9]
2. **ব্রডকাস্টিং (Broadcasting):**
   - array_1d[0:5] = 500 এই লাইনটি অ্যারের প্রথম ৫টি উপাদান (index 0 থেকে 4) পরিবর্তন করে ৫০০ দিয়ে। অর্থাৎ, প্রথম পাঁচটি উপাদান সবই ৫০০ হয়ে যাবে। নতুন অ্যারেটি হয়ে যাবে: [500 500 500 500 500 5 6 7 8 9]
3. **2D অ্যারে তৈরি করা:**
   - np.ones((4, 4)) দিয়ে একটি 8x8 ম্যাট্রিক্স তৈরি করা হয়েছে, যার সব মান ১।
   - প্রথমে অ্যারের প্রিন্ট আউট দেখানো হয়েছে, যেমন:

   [[1. 1. 1. 1.]
   [1. 1. 1. 1.]
   [1. 1. 1. 1.]
   [1. 1. 1. 1.]]
4. **২ডি অ্যারে ব্রডকাস্টিং:**
   - array_2d[0] = 300 এই লাইনটি ২ডি অ্যারের প্রথম সারি (row 0) এর সব উপাদানকে ৩০০ দিয়ে ব্রডকাস্ট করে পরিবর্তন করেছে। নতুন অ্যারেটি হবে:

   [[300. 300. 300. 300.]
   [1. 1. 1. 1.]
   [1. 1. 1. 1.]
   [1. 1. 1. 1.]]

সারাংশ:

- প্রথমে 1D এবং 2D অ্যারে তৈরি করা হয়, তারপর সেগুলিতে ব্রডকাস্টিং এর মাধ্যমে নির্দিষ্ট অংশের মান পরিবর্তন করা হয়।

বিভিন্ন ধরনের অপারেশন দেখানো হয়েছে যা NumPy এর 2D অ্যারে ব্যবহার করে করা হয়েছে।

```
import numpy as np

# Create a 2D matrix with arange (4x4 matrix)
array_2d = np.arange(16).reshape(4, 4)
print(f"Original 2D array (4x4 matrix of numbers from 0 to
15):\n{array_2d}\n")

# Adding a 1D array [0, 1, 2, 3] to the 2D array
print(f"Result after adding [0, 1, 2, 3] to each row of the 2D
array:\n{array_2d + np.arange(0, 4)}\n")
# Adding 300 to each element of the 2D array
print(f"Result after adding 300 to each element of the 2D
array:\n{array_2d + 300}\n")

# Accessing rows in a specific order using array indexing
print(f"Selecting rows [1, 2, 3] from the 2D array:\n{array_2d[[1,
2, 3]]}\n")
# Selecting rows in a different order
print(f"Selecting rows [3, 0, 1] from the 2D array:\n{array_2d[[3,
0, 1]]}\n")

# Display the original 2D array again
print(f"Original 2D array (unchanged):\n{array_2d}\n")

# Attempting to add an array with mismatched dimensions
try:
    print(f"After adding [300, 2] to the 2D array:\n{array_2d +
[300, 2]}")
except Exception as e:
    print(f"Error when trying to add [300, 2] to the 2D array:
{e}")
```

```
Original 2D array (4x4 matrix of numbers from 0 to 15):
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]


Result after adding [0, 1, 2, 3] to each row of the 2D array:
[[ 0  2  4  6]
 [ 4  6  8 10]
 [ 8 10 12 14]
 [12 14 16 18]]
```

```
Result after adding 300 to each element of the 2D array:
[[300 301 302 303]
 [304 305 306 307]
 [308 309 310 311]
 [312 313 314 315]]

Selecting rows [1, 2, 3] from the 2D array:
[[ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

Selecting rows [3, 0, 1] from the 2D array:
[[12 13 14 15]
 [ 0  1  2  3]
 [ 4  5  6  7]]

Original 2D array (unchanged):
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

Error when trying to add [300, 2] to the 2D array: operands could
not be broadcast together with shapes (4,4) (2,)
```

ব্যাখ্যাঃ

1. **মৌলিক 2D অ্যারে তৈরি:** প্রথমে একটি 4x4 ম্যাট্রিক্স তৈরি করা হয়েছে, যার মান 0 থেকে 15 পর্যন্ত।
2. **1D অ্যারে যোগ করা:** একটি 1D অ্যারে [0, 1, 2, 3] তৈরি করা হয়েছে এবং সেটি 2D অ্যারের প্রতিটি রোতে যোগ করা হয়েছে। এটি "ব্রডকাস্টিং" নামে পরিচিত, যেখানে 1D অ্যারে স্বয়ংক্রিয়ভাবে 2D অ্যারের প্রতিটি রোতে প্রযোজ্য হয়ে যায়।
3. **একটি স্কেলার মান যোগ করা:** 2D অ্যারের প্রতিটি উপাদানে 300 যোগ করা হয়েছে। এটি প্রতিটি উপাদানকে 300 বাড়িয়ে দেয়।
4. **বিশেষ রো নির্বাচন:** একাধিক রো নির্বাচিত হয়েছে। প্রথমে [1, 2, 3] রো এবং পরে [3, 0, 1] রো নির্বাচন করা হয়েছে। এটি ইনডেক্সিংয়ের মাধ্যমে করা হয়েছে, যেখানে নির্দিষ্ট ইনডেক্স দিয়ে রো গুলি নির্বাচন করা হয়।

5. **মৌলিক 2D অ্যারের পুনরায় প্রদর্শন**: 2D অ্যারে যেহেতু অপরিবর্তিত থাকে, তাই সেটি আবার প্রিন্ট করা হয়েছে।
6. **ভিন্ন আকারের অ্যারে যোগ করার চেষ্টা**: একটি ছোট আকারের অ্যারে [300, 2] যোগ করার চেষ্টা করা হয়েছে 2D অ্যারে সঙ্গে। এটি সম্ভব হয়নি, কারণ দুটি অ্যারের আকার মেলেনি এবং এর ফলে একটি ত্রুটি ঘটেছে। NumPy এমন অপারেশন অনুমোদন করে না যেখানে আকারের অমিল থাকে।

8. Array layouts.

শুরুতে এই টপিক নিয়ে আলোচনা করা হয়েছে।

9. Boolean masking.

We can apply condition such as >, <, == etc

```
import numpy as np

# Create a simple array using arange()
array_1d = np.arange(1, 11)
print(f"Original 1D array:\n{array_1d}\n")

# Print the result of the comparison (array_1d > 3)
print(f"Result of comparison (array_1d > 3):\n{array_1d > 3}\n")

# Create a boolean array where elements greater than 3 are True
bool_array = array_1d > 3
print(f"Boolean array where elements greater than 3 are True:\n{bool_array}")
```

```
Original 1D array:
[ 1  2  3  4  5  6  7  8  9 10]

Result of comparison (array_1d > 3):
[False False False  True  True  True  True  True  True  True]

Boolean array where elements greater than 3 are True:
[False False False  True  True  True  True  True  True  True]
```

1. **মৌলিক 1D অ্যারে তৈরি**: np.arange(1, 11) ব্যবহার করে একটি 1D অ্যারে তৈরি করা হয়েছে যা ১ থেকে ১০ পর্যন্ত সংখ্যাগুলি ধারণ করে।
2. **তুলনা (Comparison)**: array_1d > 3 মাধ্যমে অ্যারের প্রতিটি উপাদান যদি ৩ এর বেশি হয়, তা boolean মান (True অথবা False) ফেরত দেয়। এই ফলাফলটি print() দিয়ে দেখানো হয়।

3. **Boolean অ্যারে তৈরি:** array_1d > 3 এর ফলস্বরূপ boolean অ্যারে bool_array তৈরি করা হয়, যেখানে ৩ এর বেশি যেসব উপাদান আছে, সেগুলোর জন্য True এবং অন্যদের জন্য False থাকবে।

Lets create a mask to **filter out the even numbers in "array_1d"**

```
import numpy as np

# Create a simple array using arange()
array_1d = np.arange(1, 11)
print(f"Original 1D array:\n{array_1d}\n")

# Print the result of array_1d % 2
print(f"Result of array_1d % 2 (checking even/odd):\n{array_1d %
2}\n")

# Print the result of checking if elements are even (0 means even)
print(f"Boolean array indicating if elements are even (True for
even, False for odd):\n{0 == array_1d % 2}")
```

```
Original 1D array:
[ 1  2  3  4  5  6  7  8  9 10]

Result of array_1d % 2 (checking even/odd):
[1 0 1 0 1 0 1 0 1 0]

Boolean array indicating if elements are even (True for even,
False for odd):
[False  True False  True False  True False  True False  True]
```

এই কোডটি একটি এক-মাত্রিক অ্যারে তৈরি করে যার মান ১ থেকে ১০ পর্যন্ত। এরপর কোডটি নিম্নলিখিত কাজগুলো করে:

1. **array_1d % 2:** এটি প্রত্যেকটি উপাদানের ২ দ্বারা ভাগফলটির ভাগশেষ বের করে (যা কোনো সংখ্যা কি অপরিবর্তিত বা সোজা হবে কিনা তা চিহ্নিত করে)। যদি ভাগশেষ ০ হয়, তাহলে তা একটি *even* (জোড়া) সংখ্যা এবং ১ হলে তা একটি *odd* (বিজোড়া) সংখ্যা।
2. **0 == array_1d % 2:** এটি চেক করে কোন উপাদানটি even (০ দিয়ে ভাগফল) এবং কোনটি odd (১ দিয়ে ভাগফল)। True দেখাবে *even* সংখ্যার জন্য এবং False *odd* সংখ্যার জন্য।

এভাবেই, কোডটি **even** এবং **odd** সংখ্যা চিহ্নিত করে একটি বুলিয়ান অ্যারে তৈরি করে, যেখানে True মানে *even* এবং False মানে *odd*।

```
import numpy as np

# Create a simple array using arange()
array_1d = np.arange(1, 11)
print(f"Original 1D array:\n{array_1d}\n")

# Create a boolean mask for elements that are even (mod 2 == 0)
mod_2_mask_1d = 0 == array_1d % 2
print(f"Boolean mask for even numbers (True for even, False for
odd):\n{mod_2_mask_1d}\n")
```

```
# Use the mask to filter and print only the even elements
even_elements = array_1d[mod_2_mask_1d]
print(f"Even elements in the array:\n{even_elements}")
```

```
Original 1D array:
[ 1  2  3  4  5  6  7  8  9 10]

Boolean mask for even numbers (True for even, False for odd):
[False  True False  True False  True False  True False  True]

Even elements in the array:
[ 2  4  6  8 10]
```

10. Performing arithmetic operations with NumPy.

Arithmetic operations: We can perform arithmetic operations with NumPy arrays.

দুইটি অ্যারে তৈরি করে এবং তাদের মধ্যে বিভিন্ন গাণিতিক অপারেশন প্রয়োগ:

```
import numpy as np

# Create two arrays
arr_one = np.arange(1, 7)
arr_two = np.arange(0, 6)

# Print the arrays
print(f"Array 1: {arr_one}")
print(f"Array 2: {arr_two}\n")

# Perform element-wise arithmetic operations between the two
arrays
print(f"Element-wise addition (arr_one + arr_two):\n{arr_one +
arr_two}")
print(f"Element-wise subtraction (arr_one - arr_two):\n{arr_one -
arr_two}")
print(f"Element-wise multiplication (arr_one * arr_two):\n{arr_one
* arr_two}") #Error
print(f"Element-wise division (arr_one / arr_two):\n{arr_one /
arr_two}")

# Perform operations on arr_one
print(f"Element-wise division (1 / arr_one):\n{1 / arr_one}")
print(f"Element-wise square of arr_one (arr_one ** 2):\n{arr_one
** 2}")
print(f"Doubling arr_one (2 * arr_one):\n{2 * arr_one}")
```

```
Array 1: [1 2 3 4 5 6]
Array 2: [0 1 2 3 4 5]

Element-wise addition (arr_one + arr_two):
[ 1  3  5  7  9 11]
Element-wise subtraction (arr_one - arr_two):
```

```
[1 1 1 1 1 1]
Element-wise multiplication (arr_one * arr_two):
[ 0  2  6 12 20 30]
Element-wise division (arr_one / arr_two):
[       inf 2.         1.5        1.33333333 1.25       1.2
]
Element-wise division (1 / arr_one):
[1.         0.5        0.33333333 0.25       0.2
0.16666667]
Element-wise square of arr_one (arr_one ** 2):
[ 1  4  9 16 25 36]
Doubling arr_one (2 * arr_one):
[ 2  4  6  8 10 12]
C:\Users\MinhazulKabir\AppData\Local\Temp\ipykernel_5824\2123782928.py:15: RuntimeWarning: divide by zero encountered in divide
  print(f"Element-wise division (arr_one / arr_two):\n{arr_one / arr_two}")
```

ব্যাখ্যা দেওয়া হলো:

1. **arr_one এবং arr_two অ্যারে তৈরি:**
   - arr_one: 1 থেকে 6 পর্যন্ত একটি অ্যারে (অর্থাৎ [1, 2, 3, 4, 5, 6])।
   - arr_two: 0 থেকে 5 পর্যন্ত একটি অ্যারে (অর্থাৎ [0, 1, 2, 3, 4])।
2. **অ্যারেগুলির মধ্যে গাণিতিক অপারেশন:**
   - **যোগফল (Addition):** arr_one এবং arr_two এর প্রতিটি উপাদান যোগ করা হয়েছে।
   - **বিয়োগফল (Subtraction):** arr_one থেকে arr_two এর প্রতিটি উপাদান বিয়োগ করা হয়েছে।
   - **গুণফল (Multiplication):** arr_one এবং arr_two এর প্রতিটি উপাদান গুণ করা হয়েছে। এই ক্ষেত্রে, যেহেতু arr_two এর মধ্যে 0 আছে, ফলে গুণফলে কিছু উপাদান 0 হবে।
   - **ভাগফল (Division):** arr_one এর প্রতিটি উপাদান arr_two এর সাথে ভাগ করা হয়েছে। তবে, arr_two এর মধ্যে 0 থাকায় কিছু বিভাজন অপারেশন ভুল হবে এবং "division by zero" এর ভুল (Error) হতে পারে।
3. **অতিরিক্ত অপারেশন arr_one এর উপর:**
   - **1 / arr_one:** প্রতিটি উপাদানকে 1 দিয়ে ভাগ করা হয়েছে।
   - **arr_one ** 2:** arr_one এর প্রতিটি উপাদানকে বর্গ করা হয়েছে।
   - **2 * arr_one:** arr_one এর প্রতিটি উপাদানকে 2 দিয়ে গুণ করা হয়েছে।

নোট: গুণফল অপারেশনটি "Error" দেখাবে যদি arr_two তে 0 থাকে, কারণ শূন্য দিয়ে ভাগ করা যাবে না।

11. Using universal functions (ufuncs).

Universal functions:

NumPy have a range of built-in universal functions (ufunc). These are essentially just mathematical operations and we can use them to perform specific task, associate with the

function, across the NumPy array.

একটি অ্যারে তৈরি করে এবং তার উপর বিভিন্ন গাণিতিক অপারেশন সম্পাদন

```
import numpy as np

# Create an array
arr_minhaz = np.arange(1, 11)
print(f"Original array (arr_minhaz):\n{arr_minhaz}\n")

# Square root of each element in the array
print(f"Square root of each element in
arr_minhaz:\n{np.sqrt(arr_minhaz)}\n")

# Maximum and minimum values of the array
print(f"Maximum value in arr_minhaz: {np.max(arr_minhaz)}")
print(f"Minimum value in arr_minhaz: {np.min(arr_minhaz)}\n")

# Trigonometric function (sine of each element in the array)
print(f"Sine of each element in arr_minhaz (in
radians):\n{np.sin(arr_minhaz)}")
```

```
Original array (arr_minhaz):
[ 1  2  3  4  5  6  7  8  9 10]

Square root of each element in arr_minhaz:
[1.         1.41421356 1.73205081 2.         2.23606798 2.44948974
 2.64575131 2.82842712 3.         3.16227766]

Maximum value in arr_minhaz: 10
Minimum value in arr_minhaz: 1

Sine of each element in arr_minhaz (in radians):
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -
0.2794155
  0.6569866   0.98935825  0.41211849 -0.54402111]
```

1. **অ্যারেটি তৈরি**: np.arange(1, 11) ব্যবহার করে 1 থেকে 10 পর্যন্ত সংখ্যার একটি 1D অ্যারে তৈরি করা হয় (যার নাম arr_minhaz)।
2. **স্কয়ার রুট**: np.sqrt(arr_minhaz) প্রতিটি উপাদানের স্কয়ার রুট বের করে এবং আউটপুট হিসেবে তা প্রদর্শন করা হয়। উদাহরণস্বরূপ, 1 এর স্কয়ার রুট 1, 4 এর স্কয়ার রুট 2, ইত্যাদি।
3. **সর্বোচ্চ এবং সর্বনিম্ন মান**: np.max(arr_minhaz) সর্বোচ্চ মান এবং np.min(arr_minhaz) সর্বনিম্ন মান বের করে প্রদর্শন করে। এখানে, সর্বোচ্চ মান হবে 10 এবং সর্বনিম্ন মান হবে 1।
4. **ট্রিগনোমেট্রিক ফাংশন**: np.sin(arr_minhaz) প্রতিটি উপাদানের সাইন (sin) বের করে এবং রেডিয়ানে প্রদর্শন করে। এটি কেবল রেডিয়ান আঙ্গিকে কাজ করে, এবং এখানে 1 থেকে 10 পর্যন্ত সংখ্যার সাইন বের করা হচ্ছে।

এই কোডটি একটি অ্যারের উপর গাণিতিক অপারেশন যেমন স্কয়ার রুট, সর্বোচ্চ এবং সর্বনিম্ন মান বের করা এবং ট্রিগনোমেট্রিক ফাংশন (সাইন) ব্যবহার করা দেখিয়েছে।

একটি 2D অ্যারে (6x5 মেট্রিক্স) তৈরি করে এবং তার উপর কিছু গাণিতিক অপারেশন সম্পাদন:

```
import numpy as np

# Create a 2D array of shape (6, 5)
array_2d = np.arange(30).reshape(6, 5)
print(f"Original 2D array (6x5 matrix):\n{array_2d}\n")

# Calculate the sum of all elements in the array
total_sum = array_2d.sum()
print(f"Sum of all elements in the array: {total_sum}\n")

# Calculate the sum of each row
row_sum = array_2d.sum(axis=1)
print(f"Sum of each row in the array:\n{row_sum}\n")

# Calculate the sum of each column
column_sum = array_2d.sum(axis=0)
print(f"Sum of each column in the array:\n{column_sum}\n")

# Calculate the standard deviation of the values in the array
std_deviation = array_2d.std()
print(f"Standard deviation of the array values: {std_deviation}")
```

```
Original 2D array (6x5 matrix):
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]]

Sum of all elements in the array: 435

Sum of each row in the array:
[ 10  35  60  85 110 135]

Sum of each column in the array:
[75 81 87 93 99]

Standard deviation of the array values: 8.65544144839919
```

1. **অ্যারেটি তৈরি**: np.arange(30).reshape(6, 5) দ্বারা 0 থেকে 29 পর্যন্ত সংখ্যা নিয়ে একটি 6x5 আকারের 2D অ্যারে তৈরি করা হয়েছে।
2. **সমস্ত উপাদানের যোগফল**: array_2d.sum() দ্বারা অ্যারের সমস্ত উপাদানের যোগফল বের করা হয়েছে। এখানে অ্যারের সমস্ত উপাদান একত্রে যোগ করা হবে।
3. **প্রতিটি সারির যোগফল**: array_2d.sum(axis=1) দ্বারা প্রতিটি সারির (row) উপাদানগুলির যোগফল বের করা হয়েছে। axis=1 দ্বারা নির্দেশিত হয় যে সারি অনুযায়ী যোগফল হবে।
4. **প্রতিটি কলামের যোগফল**: array_2d.sum(axis=0) দ্বারা প্রতিটি কলামের (column) উপাদানগুলির যোগফল বের করা হয়েছে। axis=0 দ্বারা নির্দেশিত হয় যে কলাম অনুযায়ী যোগফল হবে।

5. **স্ট্যান্ডার্ড ডেভিয়েশন**: array_2d.std() দ্বারা অ্যারের মানগুলির স্ট্যান্ডার্ড ডেভিয়েশন (বিচ্যুতি) বের করা হয়েছে। এটি একটি পরিসংখ্যানমূলক পরিমাপ যা সংখ্যাগুলির গড় থেকে বিচ্যুতি পরিমাপ করে।

এই কোডটি একটি 2D অ্যারে তৈরি করে এবং তার উপর মোট যোগফল, সারি ও কলামের যোগফল, এবং স্ট্যান্ডার্ড ডেভিয়েশন বের করার কাজটি সম্পাদন করে।

একটি 2D অ্যারে তৈরি করে এবং বিভিন্ন অপারেশন সম্পাদন:

```
import numpy as np

# Create a 2D array of shape (6, 5)
array_2d = np.arange(30).reshape(6, 5)
print(f"Original 2D array (6x5 matrix):\n{array_2d}\n")

# Check if elements are not divisible by 3
mask_mod_3 = array_2d % 3 != 0
print(f"Boolean mask where elements are not divisible by
3:\n{mask_mod_3}\n")

# Use the mask to filter the elements not divisible by 3
print(f"Elements of array_2d that are not divisible by
3:\n{array_2d[mask_mod_3]}\n")

# Slice a specific region (rows 2 to 3, columns 2 to 3)
print(f"Sliced section (rows 2 to 3, columns 2 to 3) of
array_2d:\n{array_2d[2:4, 2:4]}")
```

```
Original 2D array (6x5 matrix):
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]]

Boolean mask where elements are not divisible by 3:
[[False  True  True False  True]
 [ True False  True  True False]
 [ True  True False  True  True]
 [False  True  True False  True]
 [ True False  True  True False]
 [ True  True False  True  True]]

Elements of array_2d that are not divisible by 3:
[ 1  2  4  5  7  8 10 11 13 14 16 17 19 20 22 23 25 26 28 29]

Sliced section (rows 2 to 3, columns 2 to 3) of array_2d:
[[12 13]
 [17 18]]
```

1. **অ্যারেকে তৈরি করা:** array_2d নামক একটি 6x5 আকারের 2D অ্যারে তৈরি করা হয়েছে, যেটি np.arange(30) দিয়ে 0 থেকে 29 পর্যন্ত সংখ্যাগুলি রাখে এবং reshape(6, 5) দিয়ে সেগুলিকে 6টি সারি এবং 5টি কলামে রূপান্তরিত করা হয়েছে।
2. **মডুলাস অপারেশন:** array_2d % 3 != 0 এর মাধ্যমে একটি বুলিয়ান মাস্ক তৈরি করা হয়েছে, যেটি চেক করে প্রতিটি উপাদান 3 দিয়ে বিভাজ্য কিনা। যেগুলি 3 দিয়ে বিভাজ্য নয়, সেগুলি True এবং বাকি গুলি False থাকবে।
3. **মাস্ক ব্যবহার:** এই মাস্ক ব্যবহার করে, 3 দিয়ে বিভাজ্য না হওয়া উপাদানগুলি আলাদা করে বের করা হয়েছে এবং প্রিন্ট করা হয়েছে।
4. **স্লাইসিং:** একটি নির্দিষ্ট অংশ (২য় থেকে ৩য় সারি এবং ২য় থেকে ৩য় কলাম) স্লাইস করে দেখানো হয়েছে, যাতে একটি ছোট অংশ বের করা হয়।

এই কোডে মূলত 2D অ্যারে তৈরির পর, একটি শর্ত (3 দিয়ে বিভাজ্য না হওয়া) প্রয়োগ করা হয়েছে এবং তারপর স্লাইসিং করা হয়েছে।