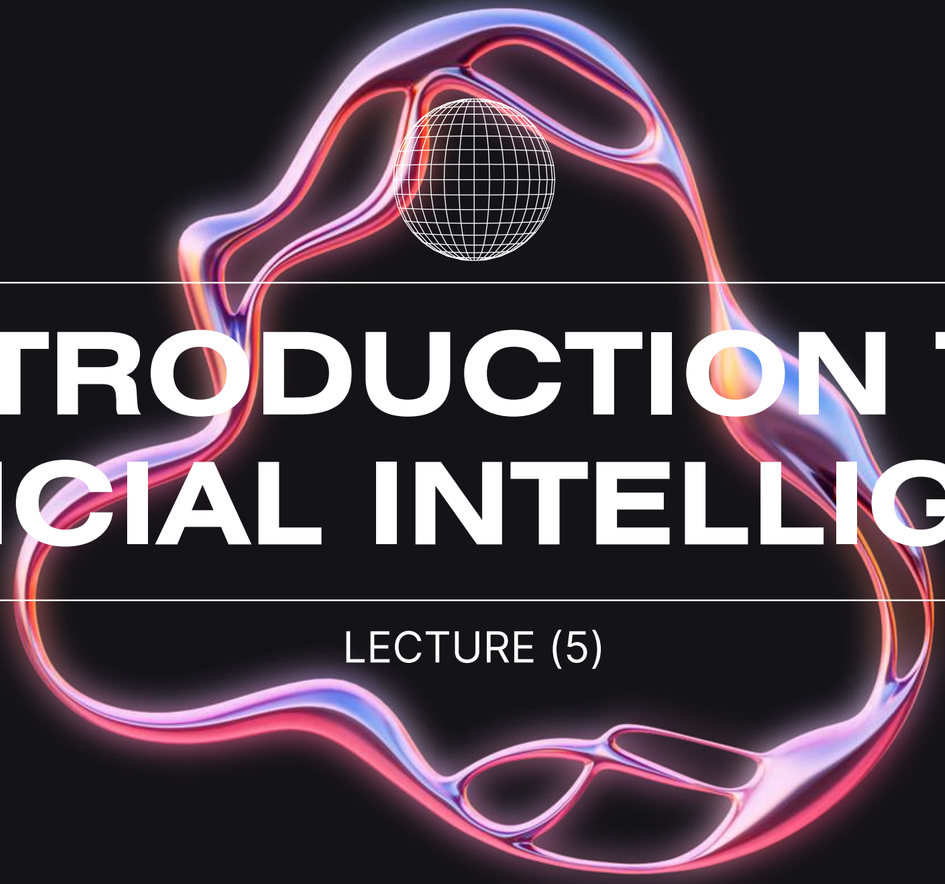
MARK KOSTANTINE



INTRODUCTION TO ARTIFICIAL INTELLIGENCE

LECTURE (5)

AGENDA

- Quiz
- Recap
- Array Shape and Reshape
- Array Iterating
- Searching Arrays
- Sorting Arrays
- Filter Array
- NumPy Project Idea
- Questions

QUIZ

Create NumPy array then slice elements from index 1 to index 4

RECAP ON LAST LECTURE

RECAP

CREATING ARRAY

- NumPy is used to work with arrays. The array object in NumPy is called **ndarray**.
- We can create a NumPy **ndarray** object by using the **array( )** function.

```
import numpy as np

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

print(arr)

print(type(arr))
```

RECAP

DIMENSIONS

```
import numpy as np

arr = np.array(42)

print(arr)
```

0D ARRAY

```
import numpy as np

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

print(arr)
```

1D ARRAY

RECAP

DIMENSIONS

```python
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

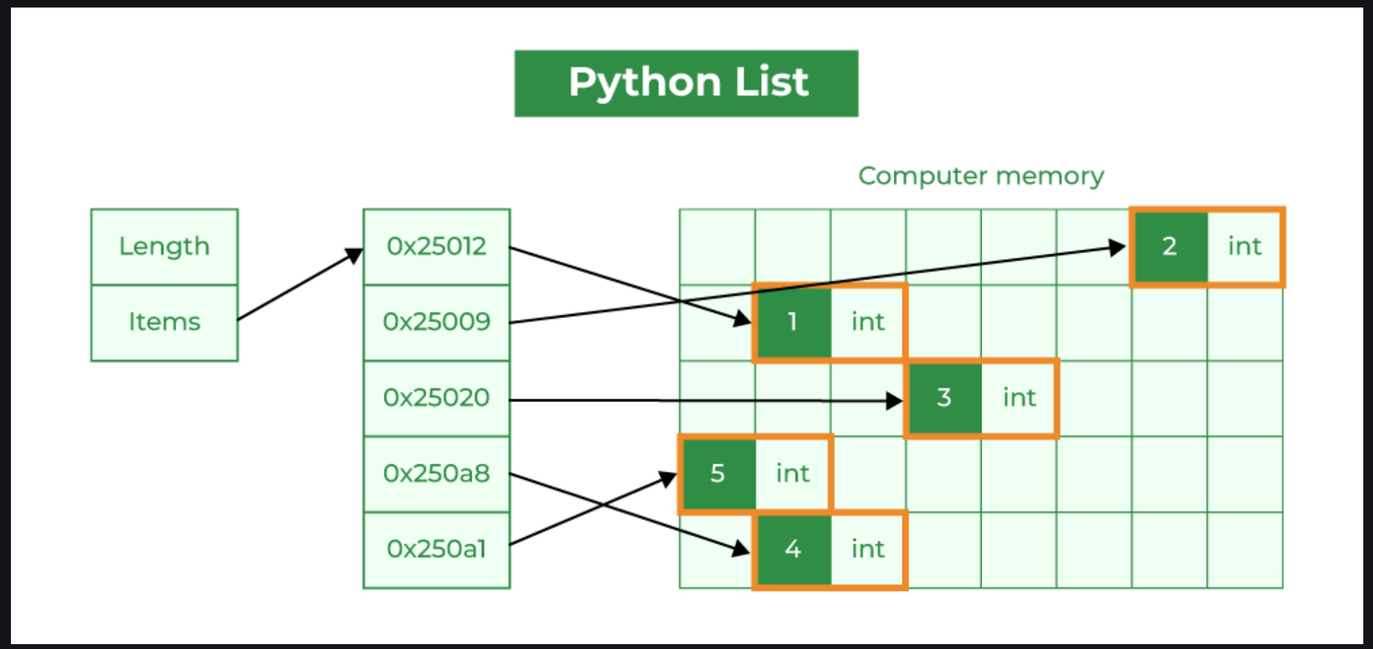
2D ARRAY

```python
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

3D ARRAY

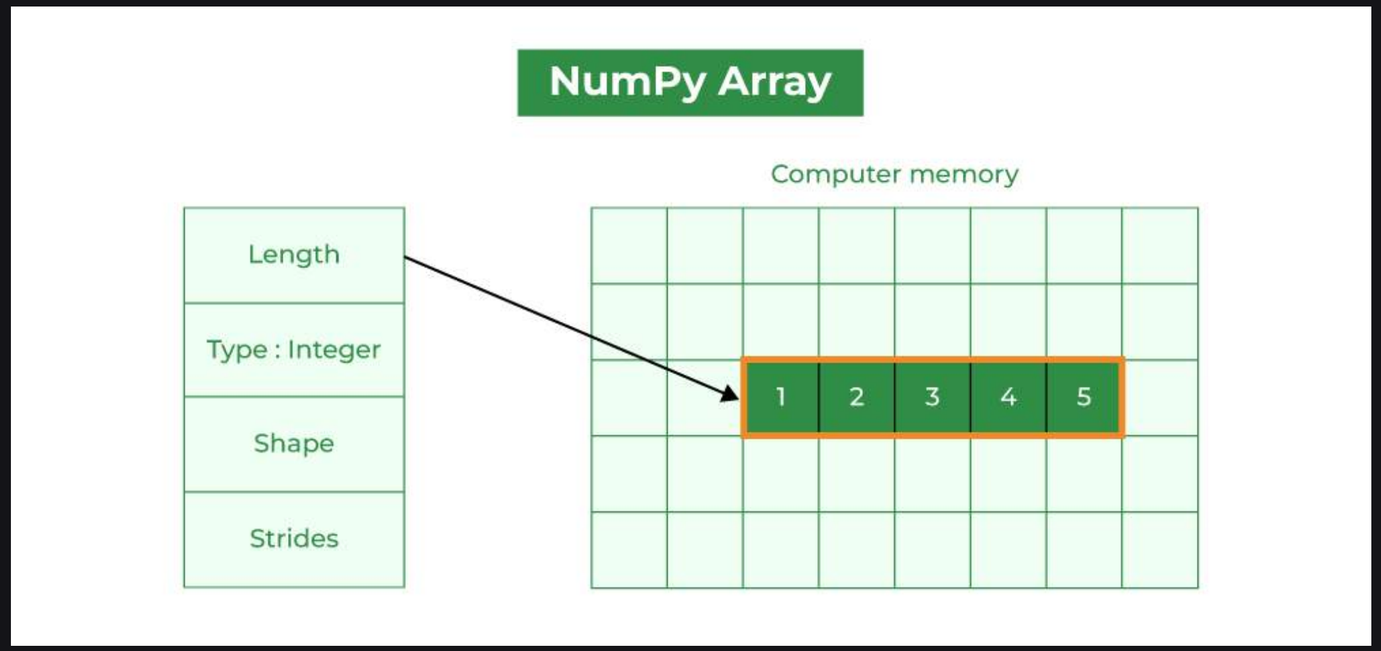
RECAP

LISTS VS ARRAYS



RECAP

LISTS VS ARRAYS



ARRAY SHAPE AND RESHAPE

ARRAY SHAPE

- The shape of an array is the number of elements in each dimension.
- NumPy arrays have an attribute called **shape** that returns a tuple with each index having the number of corresponding elements.

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```

```
import numpy as np

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

print(arr)
print('shape of array :', arr.shape)
```

ARRAY RESHAPE

- Reshaping means changing the shape of an array.
- The shape of an array is the number of elements in each dimension.
- By reshaping we can add or remove dimensions or change number of elements in each dimension.

RESHAPE FROM 1-D TO 2-D

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(4, 3)

print(newarr)
```

RESHAPE FROM 1-D TO 3-D

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(2, 3, 2)

print(newarr)
```

CAN WE RESHAPE INTO ANY SHAPE ?

- Yes, as long as the elements required for reshaping are equal in both shapes.
- We can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3 elements 3 rows 2D array as that would require 3x3 = 9 elements.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(3, 3)

print(newarr)
```

UNKNOWN DIMENSION

- You are allowed to have one **"unknown"** dimension.
- Meaning that you do not have to specify an exact number for one of the dimensions in the reshape method.
- Pass **-1** as the value, and NumPy will calculate this number for you

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

newarr = arr.reshape(2, 2, -1)

print(newarr)
```

Note : We can not pass **-1** to more than one dimension.

ARRAY RESHAPE EXERCISE

Use the correct NumPy method to change the shape of an array from 1-D to 2-D

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

newarr = arr.          (4, 3)
```


ARRAY ITERATING

ARRAY ITERATING

Iterating Arrays

- Iterating means going through elements one by one.
- As we deal with multi-dimensional arrays in numpy, we can do this using basic **for** loop of python.
- If we iterate on a 1-D array it will go through each element one by one.

```
import numpy as np

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

for x in arr:
  print(x)
```

ARRAY ITERATING

Iterating 2-D Arrays

Note : If we iterate on a n-D array it will go through n-1th dimension one by one.

- In a 2-D array it will go through all the rows

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  print(x)
```

- To return the actual values we have to iterate in each dimension

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  for y in x:
    print(y)
```

ARRAY ITERATING

Iterating 3-D Arrays

- In a 3-D array it will go through all the 2-D arrays

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

for x in arr:
  print(x)
```

- To return the actual values we have to iterate in each dimension

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

for x in arr:
  for y in x:
    for z in y:
      print(z)
```

ARRAY ITERATING

Iterating Arrays Using nditer( )

- The function **nditer( )** is a helping function that can be used from very basic to very advanced iterations. It solves some basic issues which we face in iteration
- In basic **for** loops, iterating through each scalar of an array we need to use n **for** loops which can be difficult to write for arrays with very high dimensionality.

```
import numpy as np

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):
  print(x)
```

ARRAY ITERATING

Iterating With Different Step Size

We can use filtering and followed by iteration

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for x in np.nditer(arr[:, ::2]):
  print(x)
```

ARRAY ITERATING

Enumerated Iteration Using ndenumerate()

- Enumeration means mentioning sequence number of somethings one by one
- Sometimes we require corresponding index of the element while iterating, the **ndenumerate( )** method can be used for those usecases

1D ARRAYS

```
import numpy as np

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

for idx, x in np.ndenumerate(arr):
  print(idx, x)
```

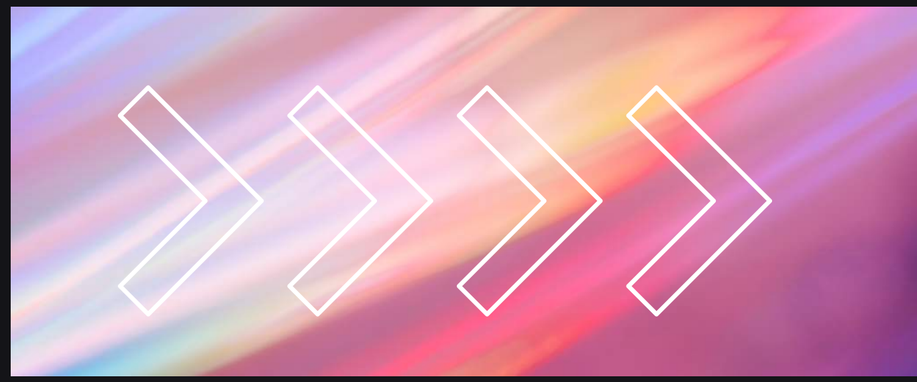
2D ARRAYS

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for idx, x in np.ndenumerate(arr):
  print(idx, x)
```


ARRAY SEARCHING

ARRAY SEARCHING

SEARCHING ARRAYS

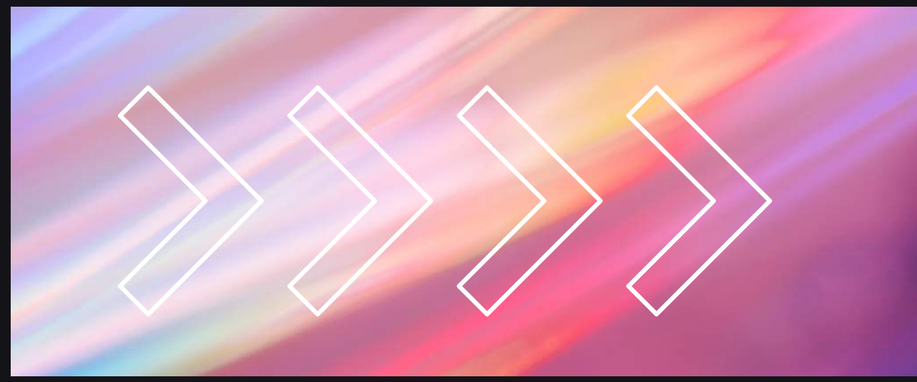


- You can search an array for a certain value, and return the indexes that get a match.
- To search an array, use the **where( )** method.

```
import numpy as np

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

x = np.where(arr == 4)

print(x)
```

ARRAY SEARCHING

SEARCHING ARRAYS



Find Even Values

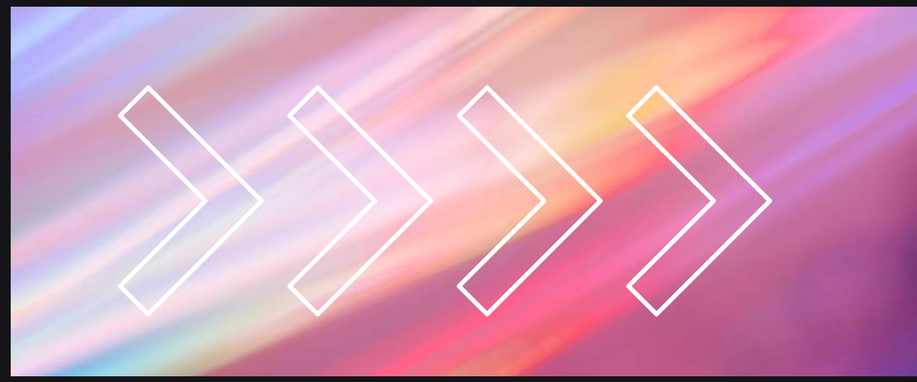
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.where(arr%2 == 0)

print(x)
```

Find Odd Values

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.where(arr%2 == 1)

print(x)
```

ARRAY SEARCHING

SEARCH SORTED



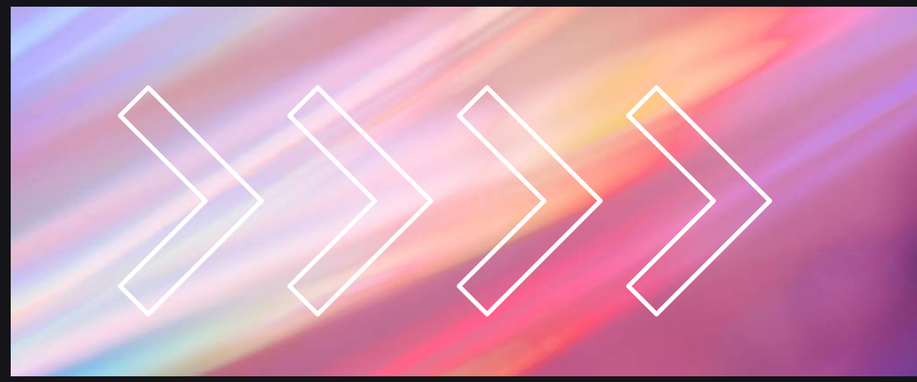
- There is a method called **searchsorted( )** which performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order.

Note : The **searchsorted( )** method is assumed to be used on sorted arrays.

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7)

print(x)
```

ARRAY SEARCHING

SEARCH FROM THE RIGHT SIDE

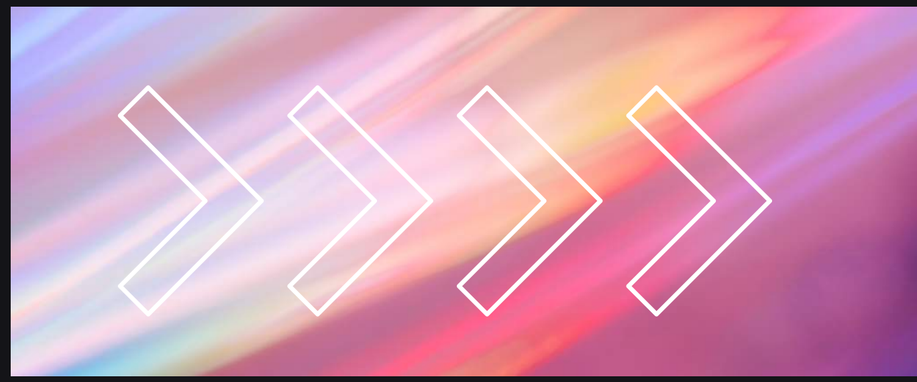


By default the left most index is returned, but we can give **side='right'** to return the right most index instead

```
import numpy as np

arr = np.array([6, 7, 8, 9])

x = np.searchsorted(arr, 7, side='right')

print(x)
```

ARRAY SEARCHING

MULTIPLE VALUES



To search for more than one value, use an array with the specified values

```
import numpy as np

arr = np.array([1, 3, 5, 7])

x = np.searchsorted(arr, [2, 4, 6])

print(x)
```

ARRAY SEARCHING EXERCISE

Use the correct NumPy method to find all items with the value 4

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

x = np.      (arr == 4)
```


ARRAY SORTING

ARRAY SORTING

- Sorting means putting elements in an ordered sequence.
- Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.
- The NumPy ndarray object has a function called **sort( )**, that will sort a specified array

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

```
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])

print(np.sort(arr))
```

Note : This method returns a copy of the array, leaving the original array unchanged.

SORTING A 2D ARRAY

- If you use the **sort( )** method on a 2-D array, both arrays will be sorted

```
import numpy as np

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

print(np.sort(arr))
```

ARRAY SORTING EXERCISE

Make a NumPy array and sort it

ARRAY FILTERING

NUMPY FILTER ARRAY

FILTERING ARRAYS

- Getting some elements out of an existing array and creating a new array out of them is called **filtering.**
- In NumPy, you filter an array using a *boolean index list*

A boolean index list : a list of booleans corresponding to indexes in the array.

If the value is **True** that element is contained in the filtered array, if the value is **False** that element is excluded from the filtered array.

```
import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```

NUMPY FILTER ARRAY

CREATING THE FILTER ARRAY

The common use is to create a filter array based on conditions

```
import numpy as np

arr = np.array([41, 42, 43, 44])

# Create an empty list
filter_arr = []

# go through each element in arr
for element in arr:
  # if the element is higher than 42, set the value to True, otherwise False:
  if element > 42:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

The common use is to create a filter array based on conditions

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

# Create an empty list
filter_arr = []

# go through each element in arr
for element in arr:
  # if the element is completely divisble by 2, set the value to True, otherwise False
  if element % 2 == 0:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

NUMPY FILTER ARRAY

CREATING THE FILTER ARRAY

NUMPY FILTER ARRAY

CREATING FILTER DIRECTLY FROM ARRAY

We can directly substitute the array instead of the iterable variable in our condition and it will work just as we expect it to

```
import numpy as np

arr = np.array([41, 42, 43, 44])

filter_arr = arr > 42

newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

01 PROJECT 1

Create a filter array that will return only even elements from the original array

02 PROJECT 2

Write a NumPy program to create a 3x3 matrix with values ranging from 2 to 10

03 PROJECT 3

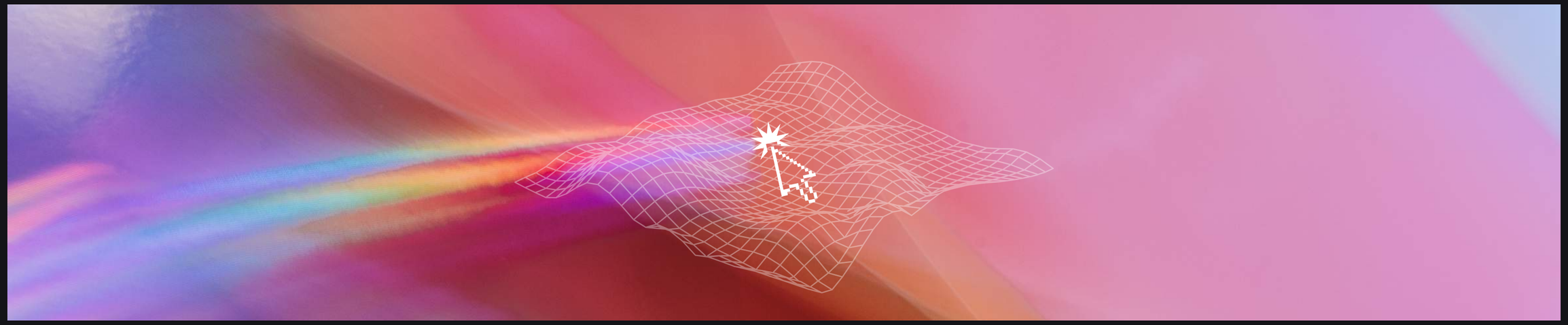
Write a NumPy program to get the values and indices of the elements that are bigger than 10 in a given array.

04 PROJECT 4

Write a NumPy program to change an array's dimension

ANY QUESTIONS ?

Feel free to ask




MARK KOSTANTINE
THANK YOU
UPCOMING NEXT : INTRO TO AI LECTURE (6)