python libraries1.NumPy

*NumPy is a Python Library.

*NumPy used for working with arrays.

*NumPy is short for Numerical Python".

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

Why Use NumPy?

1.In Python we have lists that serve the purpose of arrays, but they are slow to process.

2.NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

3.The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

4.Arrays are very frequently used in data science, where speed and resources are very important.

5.NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

In [ ]:

```

```

#How we can create numPy and how we can read

In [ ]:

```

```

1.We have created 43 tutorial pages for you to learn more about Numpy.

2.Starting with a basic introduction and ends up with creating and plotting random data sets and working with NumPy functions.

In [ ]:

```
Data Science: Is a branch of computer science where we study how to store ,use and anaylse data for deriving in
```

Why is NumPy Faster then Lists?

1.NumPy arrays are stored at one continuous place in memory unlike lists,so processes can access and manipulate them very effieciently. 2.This behaviour is called locality of reference in computer science. 3.This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Which Language is NumPy written in?

*NumPy is Python Library and written partially in Python but most of the parts that require fast computation are written in C or C++

Installation of Numpy

If you have Python and PIP already installed on a system,then Installation of NumPy is very easy.

Install it using this command:

C:\Users\Your Name>pip install numpy

If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

Import NumPy

Once NumPy is installed,import it in your applications by adding the import keyword:

In [5]:

```
import numpy
```

Now NumPy is imported and ready to use.

In [13]:

```
import numpy
arr =numpy.array([1,2,3,4,5])
print(arr)
```

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

NumPy as np

Numpy is useally imported under the np alias

Alias: In Python are an alternate name for reffering to the same thing.

Create an alias with the as keyword while importing:

In [19]:

```
import numpy as np
```

In [21]:

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

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

```
In [23]: arr=np.array([1,"string",2,3,4,4,5])
         print(arr)
```

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

Checking NumPy Version

The version string stored under _version_attribute.

```
In [26]: print(np.__version__)
```

```
1.26.4
```

NumPy Creating Arrays

Create a NumPy ndarray Object

NumPy is used for work with arrays.The array Object in NumPy is called ndarray

We can create NumPy ndarray object by using the array() function.

```
In [29]: import numpy as np
         arr=np.array([1,2,3,4,5,6,2])
         print(arr)
         print(type(arr))
```

```
[1 2 3 4 5 6 2]
<class 'numpy.ndarray'>
```

to create an ndarray ,we can pass alist,tuple or any array_like objec into the array() method,and it will be converted into an ndarray()

```
In [32]: # Use a tuple to create a NumPy array:
         import numpy as np
         arr=np.array((1,2,3,4,4,5))
         print(arr)
```

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

Dimensions in Arrays

A dimentions in arrays is one level of array depth(nested arrays).

nested array: are arrays that have arrays as their elements.

O-D Arrays

0-D arrays or scalars are the elements in an array.Each value in an array is a 0-D array.

```
In [38]: arr=np.array(42)
         print(arr)
```

```
42
```

1-D arrays

An array that has 0-D arrsys as its elements is called uni-dimentional arrays or 1-D array.

```
In [40]: arr=np.array([1,2,3,4,5])
         print(arr)
```

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

2-D Arrays

An array that has 1-D arrays as its elements is called a 2_D array. These are often represent used to represent matrix or 2nd order tensors.

```
In [ ]: NumPy has a whole sub module dedicated towards matrix operations called numpy.mat
```

```
In [53]: arr=np.array([[1,2,3],[2,3,4]])
         print(arr)
```

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

3-D arrays

An arrays that has 2-D arrays(matrices) as its elements is called 3-D array. These often used to represent 3rd order tensor.

```
In [60]: arr=np.array([[[1,2,3,4],[6,7,8,9]],[[1,2,3,4],[2,4,5,6]]])
         print(arr)
```

```
[[[1 2 3 4]
  [6 7 8 9]]

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

Check Number of Dimentions?

NumPy Arrays provides the "ndim" attribute that returns an integer that tells us how many dimentions the array have.

In [63]:
```
import numpy as np
```

In [67]:
```
a=np.array(23)
b=np.array([1,2,34,3])
c=np.array([[1,2,3,4],[3,4,5,5]])
d=np.array([[[1,2,3,4],[233,4,4,5]],[[2,3,4,4],[4,4,55,3]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

Higher Dimentional Arrays

In [ ]:
```
An Array can have any number of dimentions.
When the array is created,you can  define the number of dimentions by using "ndmin" argument
```

In [44]:
```
arr=np.array([1,2,4,5],ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
```

```
[[[[[1 2 4 5]]]]]
number of dimensions : 5
```

In this array the innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dim has 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

NumPy Array IndexingAccess Array Elements

1.Array indexing is the same as accessing an array element. 2.You can access an array element by reffering to its index number. 3. The indexes inNumPy arrays start with 0,meaning that the first element has index 0,and the second has index 1 etc.

In [77]:
```
arr=np.array([1,2,4,5])
print(arr[0])
```

```
1
```

In [79]:
```
import numpy as np

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

print(arr[1])
```

```
2
```

In [84]:
```
# Get third and fourth elements from the following array and add them.
arr = np.array([1, 2, 3, 4])

print(arr[1]+arr[3])
```

```
6
```

In [ ]:
```

```

Access 2-D arrays

To access the elements from 2-d arrays we can use the comma seperated integers representing the dimetions and the index of the element.

think of 2-d arrays like a table with rows and columns ,where the dimentions represent the rows and the index represents the column .

In [111…
```
import numpy as np
arr=np.array([[1,2,3,45,4],[7,5,3,4,4]])
print(arr)
print("2nd element of 1st row:",arr[0,1])
print(arr.ndim)# 0 indecates row and 1 indicates column
```

```
[[ 1  2  3 45  4]
 [ 7  5  3  4  4]]
2nd element of 1st row: 2
2
```

In [16]:
```
#Access the element on the 2nd row, 5th column:
import numpy as np
arr=np.array([[1,2,4,55,54],[3,43,3,32,23]])
print("5th element in 2cond column",arr[1,4])
```

```
5th element in 2cond column 23
```

Access 3-D Arrays

to access elements from 3-d arrays we can use comma seperaed integers representing the dimentions and the index of the element.

In [24]:
```
#acces the third element of the second array of the first array:
arr=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
print(arr)
print("acces the third element of the second array of the first array:",
arr[0,1,2])
```

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

 [[ 7  8  9]
  [10 11 12]]]
acces the third element of the second array of the first array: 6
```

In [ ]:
```
Example Explained
arr[0, 1, 2] prints the value 6.

And this is why:

The first number represents the first dimension, which contains two arrays:
[[1, 2, 3], [4, 5, 6]]
and:
[[7, 8, 9], [10, 11, 12]]
Since we selected 0, we are left with the first array:
[[1, 2, 3], [4, 5, 6]]

The second number represents the second dimension, which also contains two arrays:
[1, 2, 3]
and:
[4, 5, 6]
Since we selected 1, we are left with the second array:
[4, 5, 6]

The third number represents the third dimension, which contains three values:
4
5
6
Since we selected 2, we end up with the third value:
6
```

In [ ]:
```
**Negative indexing
```

In [ ]:
```
use negative indexing to acces an array from the end
```

In [26]:
```
import numpy as np

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

print('Last element from 2nd dim: ', arr[1, -1])

# 1 indicates the second list
#and -1 indecates the direction from last
```

```
Last element from 2nd dim:  10
```

In [ ]:
```

```

NumPy Array Slicing

In [ ]:
```
Slicing arrays
1.slicing in python means taking elements from one index to another index.
2.we pass the slice instead of index like this:[start:end]
3.we can also define the step ,like this:[start:end:step]
4.If don't the start its considered 0
5.if we don't pass the end its considered length of the arrat in that dimention
6.if we don't pass step its considered 1
```

In [29]:
```
# slice  elements from index 1 to index 5 from the following array:
import numpy as np
arr=np.array([1,2,34,3,5,56,6,5])
print(arr[1:5])
```

```
[ 2 34  3  5]
```

Note: The result includes the start index, but excludes the end index.

```
In [ ]: #slice the elements from index 4 to end of the array
```

```
In [33]: arr=np.array([1,2,34,3,5,56,6,5])
         print(arr[4:])
```

```
[ 5 56  6  5]
```

```
In [36]: # Slice elements from the beginning to index 4 (not included)
         arr=np.array([1,2,34,3,5,56,6,5])
         print(arr[:4])
```

```
[ 1  2 34  3]
```

```
In [ ]: 
```

```
In [ ]: Negative slicing
```

use the minus operator to refer to an index from the end:

```
In [39]: arr=np.array([1,2,34,3,5,56,6,5])
         print(arr[-3:-1])
```

```
[56  6]
```

```
In [ ]: *STEP
```

Use the step value to determine the step of the slicing:

```
In [44]: import numpy as np
         #Return every other element from index 1 to index 5:
         arr = np.array([1, 2, 3, 4, 5, 6, 7])

         print(arr[1:5:2])
```

```
[2 4]
```

```
In [46]: # Return every other element from the entire array:

         import numpy as np

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

         print(arr[::2])
```

```
[1 3 5 7]
```

```
In [ ]: Slicing 2-D Arrays
```

```
In [48]: # From the second element, slice elements from index 1 to index 4 (not included):

         import numpy as np

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

         print(arr[1, 1:4]) # second array to take values from 1 ro 4
```

```
[7 8 9]
```

Note: Remember that second element has index 1.

```
In [57]: # From both elements, return index 2:

         import numpy as np

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

         print(arr[0:2, 2]) # from both arrays we took 2nd index element
```

```
[3 8]
```

```
In [1]: # From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:

        import numpy as np

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

        print(arr[0:2, 1:4])
```

```
[[2 3 4]
 [7 8 9]]
```

```
In [ ]: 
```

*NumPy Data Types

1.data types in Python

Strings use-to represent text data,the text is given under qoute marks e.g "abcd" integer-used to represent integer numbers e.g -1,-2,-3 *float -used to represent real numbers eg:1.2,3.2* boolean -used to represent true or false *complex -used to represent complex numbers eg:1.0+2.j.1+j

**data types in NumPy

In [ ]:
```
NumPy has some extra data types and refer to data types with one character ,like 'i' for integers ,u for unsighn
Below is a list of all data types in NumPy and the characters used to represent them.
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 type ( void )
```

Checking the Data Type of an Array

The NumPy array object has a property called dtype that returns the data type of the array:

In [8]:
```
#get the type of an array object:
import numpy as np
arr=np.array([1,2,3,4])
print(arr.dtype)
```

```
int32
```

In [10]:
```
# Get the data type of an array containing strings:
arr=np.array(["apple","banana","cherry"])
print(arr.dtype)
```

```
<U6
```

Creating Arrays With a Defined Data Type

we use the array() function to create arrays ,this function can take an optional argument : "dtype" that allows us to define the expected data type of the array elements:

In [14]:
```
import numpy as np
arr=np.array([1,2,3,4],dtype='S')
print(arr)
print(arr.dtype)
```

```
[b'1' b'2' b'3' b'4']
|S1
```

In [18]:
```
#Create an array with data type 4 bytes integer:
import numpy as np
arr=np.array([1,2,3,4],dtype='i')
print(arr)
print(arr.dtype)
```

```
[1 2 3 4]
int32
```

In [22]:
```
import numpy as np
arr=np.array([1,2,3,4],dtype='f')
print(arr)
print(arr.dtype)
```

```
[1. 2. 3. 4.]
float32
```

In [26]:
```
import numpy as np
arr=np.array([1,2,3,4],dtype='U')
print(arr)
print(arr.dtype)
```

```
['1' '2' '3' '4']
<U1
```

In [28]:
```
# Create an array with data type 4 bytes integer:

import numpy as np

arr = np.array([1, 2, 3, 4], dtype='i4')

print(arr)
print(arr.dtype)
```

```
[1 2 3 4]
int32
```

In [ ]:

What if a Value Can Not Be Converted?

If a type is given in which elements can't be casted then NumPy will raise a ValueError.

ValueError: In Python ValueError is raised when the type of passed argument to a function is unexpected/incorrect.

In [32]:
```
# A non integer string like 'a' can not be converted to integer (will raise an error):

import numpy as np

arr = np.array(['a', '2', '3'], dtype='i')
print(arr)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[32], line 5
      1 # A non integer string like 'a' can not be converted to integer (will raise an error):
      3 import numpy as np
----> 5 arr = np.array(['a', '2', '3'], dtype='i')
      6 print(arr)

ValueError: invalid literal for int() with base 10: 'a'
```

In [ ]:

Converting Data Type on Existing Arrays

The best way to change data type of an existing array,is to make a copy of the array with the astype() method.

The astype() function creates a copy of the array,and allows you to specify the data type as a parameter. the data type can be specified using a string ,like 'f' for float 'i' for integer etc. or you can use the data type directly like 'float' for float and 'int' for integer.

In [46]:
```
import numpy as np
arr=np.array([1.1,2.1,3.1])
newarr=arr.astype('i')
print(newarr.dtype)
print(newarr)
```

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

In [40]:
```
# Change data type from float to integer by using int as parameter value:

import numpy as np

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

newarr = arr.astype(int)

print(newarr)
print(newarr.dtype)
```

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

In [42]:
```
# Change data type from integer to boolean:

import numpy as np

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

newarr = arr.astype(bool)

print(newarr)
print(newarr.dtype)
```

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

In [ ]:

NumPy Array Copy vs View

The Difference Between Copy and View

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 original array,and any changes made to the original array will not affect the copy

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.

In [45]:
```
COPY:

# Make a copy and change the original array and display both arrays:
arr=np.array([1,2,34,566])
x=arr.copy()
arr[0]=23
print(arr)
print(x)
```

```
[ 23   2  34 566]
[  1   2  34 566]
```

view:

In [48]:
```
# make a view ,change the original array and display both arrays:
arr=np.array([1,2,3,4,5])
x=arr.view()
arr[0]=22
print(arr)
print(x)
```

```
[22  2  3  4  5]
[22  2  3  4  5]
```

The view SHOULD be affected by the changes made to the original array

In [ ]:
```
Make changes in the view:
Make a view, change the view, and display both arrays:
```

In [51]:
```
arr=np.array([1,2,3,4,5])
x=arr.view()
x[0]=22
print(arr)
print(x)
```

```
[22  2  3  4  5]
[22  2  3  4  5]
```

The original array SHOULD be affected by the changes made to the view.

Check if Array Owns its Data:

As mentioned above,copies owns the data,and views does not owns the data but how can we check this?

Every NumPy array has the attribute 'base' that returns 'None' if the array owns the data.

Otherwise,the 'base ' attribute refers to the original object

In [57]:
```
arr=np.array([1,2,3,4,5])
x=arr.copy()
y=arr.view()
print(x.base)
print(y.base)
#The copy returns None.
# The view returns the original array.
```

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

NumPy Array Shape

Shape of an Array: The shape of an array is the number of elements of each dimention.

In [ ]:
```
Get the Shape of an Array:

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of correspon
```

In [82]:
```
# print the shape of a 2-D array:
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

```
(2, 4)
```

The example above returns (2, 4), which means that the array has 2 dimensions, where the first dimension has 2 elements and the second dimension has 4.like 2 rows with 4 colomns

In [72]:
```
# Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimensi
```

```
arr=np.array([2,1,2,34,45] ,ndmin=5)
print(arr.ndim)
print(arr)
print('shape of an array',arr.shape)
```

```
5
[[[[[ 2  1  2 34 45]]]]]
shape of an array (1, 1, 1, 1, 5)
```

What does the shape tuple represent?

Integers at every index tells about the number of elements the corresponding dimension has.

In the example above at index-4 we have value 4, so we can say that 5th (4 + 1 th) dimension has 4 elements.

In [ ]:
```
NumPy Arrays Reshaping
```

In [ ]:
```
Reshaping arrays:

1.Reshape means changing the shape of an array.
2.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

In [88]:
```
# Convert the following 1-D array with 12 elements into a 2-D array.

# The outermost dimension will have 4 arrays, each with 3 elements:
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr=arr.reshape(4,3)
print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

In [ ]:
```
Reshape From 1-D to 3-D
```

In [90]:
```
# Convert the following 1-D arraywith 12 elements into a 3-D array.

# The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:

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

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

print(newarr)
```

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

 [[ 7  8]
  [ 9 10]
  [11 12]]]
```

Can we reshape into any shape ?

yes,as long as the elements required for reshaping are wqual 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.

In [107…
```
import numpy as np

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

newarr = arr.reshape(3,3 )

print(newarr)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[107], line 5
      1 import numpy as np
      3 arr = np.array([1, 2, 3,  5, 6, 8,2,2])
----> 5 newarr = arr.reshape(3,3 )
      7 print(newarr)

ValueError: cannot reshape array of size 8 into shape (3,3)
```

In [ ]:
```
Returns Copy or View?
Example
```

In [379…
```
# check if the returned array is copy or view
import numpy as np
arr=np.array([1,2,3,4,3,5,3,5,55,4])
print(arr.reshape(2,5).base)
```

```
[ 1  2  3  4  3  5  3  5 55  4]
```

The example above returns the original array, so it is a view.

In [ ]:
```
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

In [117…
```
# Convert 1D array with 8 elements to 3D array with 2x2 elements:
arr=np.array([1,2,3,4,5,6,7,8])
newarr=arr.reshape(2,2,-1)
print(newarr)
```

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

 [[5 6]
  [7 8]]]
```

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

In [ ]:
```
Flattening the arrays
```

In [ ]:
```
flattening array means converting a multidimensional array into 1D array.
we can reshape(-1) to do this.
```

In [122…
```
import numpy as np
arr= np.array([[1,2,3],[3,4,5]])
newarr=arr.reshape(-1)
print(newarr)
```

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

In [130…
```
import numpy as np
arr= np.array([[1,2,3],[3,4,5]])
newarr=arr.reshape(-1)
print(newarr)
```

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

In [ ]:

*NumPy Array Iterating:

Iterating Arrays:

Iterating means going through elements one by one.

As we deal with multi-dimentional arrays in NumPy,we can do this using basic for loop of python

If we go through 1-D array it will go through each element one by one.

In [7]:
```
"iterate an elements of the following 1-D array:"

import numpy as np
arr=np.array([1,2,3,4])
for x in arr:
    print(x)
```

```
1
2
3
4
```

Iteratig 2-D Arrays

In [9]:
```
#In  2-D it will go through  all the rows
arr = np.array([[1, 2, 3], [4, 5, 6]])
for i in arr:
    print(i)
```

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

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

In [383…
```
#iterate on each scalar element of the 2-D array

arr=np.array([[3,2,1],[2,3,4]])
for i in arr:
    for j in i:
        print(j)
```

```
3
2
1
2
3
4
```

In [ ]:
```
Iterating 3-D Arrays:
```

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

In [ ]:
```
# Iterate on the elements of the following 3-D array:
```

In [17]:
```
#iterate down to scalars
import numpy as np
arr=np.array([[[1,2,3,44],[2,3,4,4]],[[2,4,4,4],[4,42,4,24]]])
for i in arr:
    for j in i:
        for k in j:
            print(k)
```

```
1
2
3
44
2
3
4
4
2
4
4
4
4
42
4
24
```

In [19]:
```
arr=np.array([[[1,2,3,44],[2,3,4,4]],[[2,4,4,4],[4,42,4,24]]])
for i in arr:
    print(i)
```

```
[[ 1  2  3 44]
 [ 2  3  4  4]]
[[ 2  4  4  4]
 [ 4 42  4 24]]
```

In [ ]:

In [ ]:
```
**Itarating Arrays Using nditer():
```

The function nditer() function is helping function that can be used from very basic to very advanced iterations .

It solves some basic isses which we face in iteration ,lets go through it with exapmles

In [ ]:
```
**Iterating on Each Scalar Element:
```

In [ ]:
```
In basic for loops ,ietarting through each scalar of an array we need to use n for lops which can be difficult
```

In [26]:
```
#Iterate through the following 3-D array:

arr=np.array([[[1,2],[2,3]],[[2,4],[4,5]]])
for i in np.nditer(arr):
    print(i)
```

```
1
2
2
3
2
4
4
5
```

In [ ]:
```
*Itarating array with different Data Types*
```

we can use op_dtype arguement and pass it the expected if the expected datatype to change the datatype of elements while itarating.

NumPy does not chnage the datatype of the element in place(where the element is in array) so it needs some other space to perform this action ,extra space is called "buffer" ,and in order to enable it in "nditer()" we pass flags=['buffered']

In [58]:
```
# Iterate through the as a String

import numpy as np

arr=np.array([1,2,3])
for i in np.nditer(arr,flags=['buffered'],op_dtypes=['m']):
    print(i)
```

```
1 generic time units
2 generic time units
3 generic time units
```

In [60]:
```
import numpy as np

arr=np.array([1,2,3])
for i in np.nditer(arr,flags=['buffered'],op_dtypes=['S']):
    print(i)
```

```
b'1'
b'2'
b'3'
```

Iterating with Different step size

we can use filtering and followed by iteration.

In [35]:
```
# Iterate through every scalar element of the 2D array skipping 1 element:

arr=np.array([[1,2,3,4],[5,6,6,6]])
for i in np.nditer(arr[:,::2]):
    print(i)
```

```
1
3
5
6
```

In [79]:
```
arr=np.array([[2,3,44,5,55,5],[2,4,55,5,5,4]])
for i in np.nditer(arr[0:2,1:6:2]):# 0 difines starting array and 2 is ending in 2D  and 1:6:2 1 deffines stari
    print(i)
```

```
3
5
5
4
5
4
```

In [ ]:
```
**Enumerated Iteration Using ndenumerate()**
```

Enumaration means mentioning sequence number of somethings one by one .

sometimes we require corresponding index of elements while iterating,the 'ndenumerate()' method can be used for those usecases.

In [ ]:
```
Enumerate on following 1D arrays elements:
```

In [88]:
```
arr=np.array([1,3,34,5,5,6,6])
for idx,i in np.ndenumerate(arr):
    print(idx,i)
```

```
(0,) 1
(1,) 3
(2,) 34
(3,) 5
(4,) 5
(5,) 6
(6,) 6
```

In [ ]:
```
Enumerate on following 2D array's elements:
```

In [90]:
```
arr=np.array([[1,2,3,4],[2,4,45,55]])
for idx,i in np.ndenumerate(arr):
    print(idx,i)
```

```
(0, 0) 1
(0, 1) 2
(0, 2) 3
(0, 3) 4
(1, 0) 2
(1, 1) 4
(1, 2) 45
(1, 3) 55
```

In [ ]:

```

```

In [ ]:

```
            "Joing NumPy Arrays"
```

Joining means putting contents of one or more arrays in single array.

In Sql,we join tables based on a key,in numpy we join arrays by axis.

We pass a sequence of arrays that we want to join to the 'concatenete()' function,along with the sxis ,If axis is not exlpicitly passed ,it taken as 0.

In [ ]:

```
#join two arrays
```

In [99]:

```
arr1=np.array([1,2,4,45,3,6,6])
arr2=np.array([3,4,5,6,6,6,77])
arr=np.concatenate((arr1,arr2))
print(arr)
```

```
[ 1  2  4 45  3  6  6  3  4  5  6  6  6 77]
```

In [ ]:

```
# Join two 2-D arrays along rows (axis=1):
```

In [65]:

```
arr1=np.array([[1,2,3,3],[2,3,4,4]])
arr2=np.array([[2,2,3,4],[32,3,32,32]])

arr=np.concatenate((arr1,arr2),axis=1) #axis=1 means 2-D
print(arr)
```

```
[[ 1  2  3  3  2  2  3  4]
 [ 2  3  4  4 32  3 32 32]]
```

In [125…

```
arr1=np.array([[[1,2,3,3],[2,3,4,4]]])
arr2=np.array([[[2,2,3,4],[32,3,32,32]]])
arr3=np.array([[[1,2,3,3],[2,3,4,4]]])
arr=np.concatenate((arr1,arr2,arr3),axis=2) axis =2 means 3-D
print(arr)
```

```
[[[ 1  2  3  3  2  2  3  4  1  2  3  3]
  [ 2  3  4  4 32  3 32 32  2  3  4  4]]]
```

In [ ]:

```
"Joining Arrays Using Stack Functions"
```

Stacking is same as concatination,the only difference is that stacking is done along a new axis.

we can concatenate two 1-D arrays along the second axis wolud resut in putting them one over the other ie. stacking

we pass the sequence of arrays thet we want to join to the stack() method along with the axis .If the axis is not explicitly passed it is taken as 0.

In [391…

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

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

In [ ]:

```
Stacking Along Rows(horizontally)
```

NumPy provides a helper function:hstack() to stack along rows.

In [133…

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

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

In [157…

```
arr1 = np.array([1, 2, 3])

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

```
arr = np.hstack((arr1, arr2))

print(arr)
```

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

In [ ]:
```
Stacking Along Columns(vertically)
```

NumPy provides a helper function :' vstack() ' to stack columns

In [144…
```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.vstack((arr1, arr2))

print(arr)
```

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

In [155…
```
arr1 = np.array([1, 2, 3,3])

arr2 = np.array([4, 5, 6])
arr = np.vstack((arr1, arr2))

print(arr)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[155], line 4
      1 arr1 = np.array([1, 2, 3,3])
      3 arr2 = np.array([4, 5, 6])
----> 4 arr = np.vstack((arr1, arr2))
      6 print(arr)

File ~\anaconda3\Lib\site-packages\numpy\core\shape_base.py:289, in vstack(tup, dtype, casting)
    287 if not isinstance(arrs, list):
    288     arrs = [arrs]
--> 289 return _nx.concatenate(arrs, 0, dtype=dtype, casting=casting)

ValueError: all the input array dimensions except for the concatenation axis must match exactly, but along dimen
sion 1, the array at index 0 has size 4 and the array at index 1 has size 3
```

In [ ]:
```
Stacking Along Height (depth)
```

NumPy provides a helper function: dstack() to stack along height, which is the same as depth.

In [147…
```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.dstack((arr1, arr2))

print(arr)
```

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

In [ ]:
```
    "NumPy Sliptting Array"
```

splitting is reverse operation of joins

joining merges multiple arrays into one array and splitting is breaks the single array into multiple arrays.

we use ' array_split()' for splitting array ,we pass it the array we want to split and the number of splits.

In [160…
```
#split the array into three parts

import numpy as np
arr=np.array([1,2,36,6,4,5,5,6,65,4,66,4])
newarr=np.array_split(arr,3)
print(newarr)
```

```
[array([ 1,  2, 36,  6]), array([4, 5, 5, 6]), array([65,  4, 66,  4])]
```

Note: The return value is a list containing three arrays.

If the array has less elements than required, it will adjust from the end accordingly.

In [163…
```
import numpy as np
arr=np.array([1,2,36,6,4,5,5,6,65,4,66,4,424,24])
```

```
newarr=np.array_split(arr,3)
print(newarr) # here last array has less elements
```

```
[array([ 1,  2, 36,  6,  4]), array([ 5,  5,  6, 65,  4]), array([ 66,   4, 424,  24])]
```

In [165…

```
#split the array into 4 parts:
import numpy as np
arr=np.array([1,2,36,6,4,5,5,6,65,4,66,4])
newarr=np.array_split(arr,4)
print(newarr)
```

```
[array([ 1,  2, 36]), array([6, 4, 5]), array([ 5,  6, 65]), array([ 4, 66,  4])]
```

Note: We also have the method split() available but it will not adjust the elements when elements are less in source array for splitting like in example above, array_split() worked properly but split() would fail.

In [168…

```
import numpy as np
arr=np.array([1,2,36,6,4,5,5,6,65,4,66,4])
newarr=np.split(arr,4)
print(newarr)
```

```
[array([ 1,  2, 36]), array([6, 4, 5]), array([ 5,  6, 65]), array([ 4, 66,  4])]
```

In [ ]:

```
"Split into arrays"
```

the return values of the array_split() method is an array containing each of the split as an array.

If you split an array into 3 arrays, you can access them from the result just like any array element:

In [ ]:

```
Access the splitted arrays:
```

In [183…

```
arr=np.array([1,2,3,4,5,66,6,7,7])
newarr=np.array_split(arr,3)
print("oroginal:",newarr)

print(newarr[0])
print(newarr[1])
print(newarr[2])
```

```
oroginal: [array([1, 2, 3]), array([ 4,  5, 66]), array([6, 7, 7])]
[1 2 3]
[ 4  5 66]
[6 7 7]
```

In [ ]:

```
"Splitting 2-D Arrays"
```

use the same syntax when splting 2-D arrays;

Use the array_split() method, pass in the array you want to split and the number of splits you want to do.

Example

In [188…

```
arr=np.array([[1,2],[3,4],[5,66],[6,7],[7,6]])
newarr=np.array_split(arr,3)
print("oroginal:",newarr)

print(newarr[0])
print(newarr[1])
print(newarr[2])
```

```
oroginal: [array([[1, 2],
       [3, 4]]), array([[ 5, 66],
       [ 6,  7]]), array([[7, 6]])]
[[1 2]
 [3 4]]
[[ 5 66]
 [ 6  7]]
[[7 6]]
```

The example above returns three 2-D arrays.

Let's look at another example, this time each element in the 2-D arrays contains 3 elements.

In [ ]:

```
Split the 2-D array into three 2-D arrays.
```

In [191…

```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

newarr=np.array_split(arr,3)
print(newarr)
```

```
[array([[1, 2, 3],
       [4, 5, 6]]), array([[ 7,  8,  9],
       [10, 11, 12]]), array([[13, 14, 15],
       [16, 17, 18]])]
```

The example above returns three 2-D arrays.

In addition,we can specify which axis youo wnat to do the splits around.

the example below also returns three 2-D arrays ,but they are split along the row (axis=1)

In [200…
```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

newarr = np.array_split(arr, 3, axis=1)
print(newarr)
```

```
[array([[ 1],
       [ 4],
       [ 7],
       [10],
       [13],
       [16]]), array([[ 2],
       [ 5],
       [ 8],
       [11],
       [14],
       [17]]), array([[ 3],
       [ 6],
       [ 9],
       [12],
       [15],
       [18]])]
```

An alternate solution is using hsplit() opposite of hstack()

In [203…
```
# Use the hsplit() method to split the 2-D array into three 2-D arrays along rows.

import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])

newarr = np.hsplit(arr, 3)

print(newarr)
```

```
[array([[ 1],
       [ 4],
       [ 7],
       [10],
       [13],
       [16]]), array([[ 2],
       [ 5],
       [ 8],
       [11],
       [14],
       [17]]), array([[ 3],
       [ 6],
       [ 9],
       [12],
       [15],
       [18]])]
```

Note: Similar alternates to vstack() and dstack() are available as vsplit() and dsplit().

In [ ]:
```

```

In [ ]:
```
        "NumPy Searching Arrays"
```

you can search an array for a certain value ,and return the indexs that get a match

to search an array,use the where() method.

In [213…
```
import numpy as np

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

x = np.where(arr == 4)

print(x)
```

```
#it will give the indexs of getting matched values
```

```
(array([3, 5, 6], dtype=int64),)
```

In [4]:
```
import numpy as np

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

x = np.where(arr == 3)

print(x)
```

```
(array([2, 3, 4], dtype=int64),)
```

The example above will return a tuple: (array([2, 3, 4],)

Which means that the value 4 is present at index 3, 5, and 6.

In [ ]:
```
Find the indexes where the values are even:
```

In [220…
```
arr=np.array([1,2,3,45,6,77,32,43,8,99,9,0,0,23,54,3,3,3,2])
x=np.where(arr%2==0) #it will checks only indexes having even values
print(x)
```

```
(array([ 1,  4,  6,  8, 11, 12, 14, 18], dtype=int64),)
```

In [ ]:
```
Find the indexes where the values are odd:
```

In [229…
```
arr=np.array([1,2,3,45,6,77,32,43,8,99,9,0,0,23,54,3,3,3,2])
x=np.where(arr%2!=0) #it will checks only indexes having odd values
print(x)
```

```
(array([ 0,  2,  3,  5,  7,  9, 10, 13, 15, 16, 17], dtype=int64),)
```

In [ ]:
```
"Search Sorted"
```

there is a method called 'searchsorted()' which performs a binary search in the array, and returns the index where the specified value wolud be inserted to maintain the search order

example, whenever we inserted any value it will checks the entire array and assume it is sorted and occupy the least index value left or right of the inside values.

like [2,3,7,4,6] in this we have 5 elements but not sorted if give that method it will assume it issorted like [2,3,4,6,7] and if insert some value bre like 5 it will take position after 4 .

and if we have same values like [1,2,3,3,4,5,5,] like this we have to mention which side you want to place it means side='right' or side='left'

In [18]:
```
arr=np.array([3,4,5,6,7,8,9])
x=np.searchsorted(arr,7)  #here we have 7 so it will occupy the before 7 index ,it takes side as side='left ' wi
print(x)
```

```
4
```

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

Example explained: The number 7 should be inserted on index 4 to remain the sort order.

The method starts the search from the left and returns the first index where the number 7 is no longer larger than the next value.

In [ ]:
```
"search from the right side"
```

by default the left most index is returned ,but we can give side="right" to return right most index insted

In [23]:
```
import numpy as np

arr = np.array([6, 7,9, 8, 9]) #takes after previous 7 index

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

print(x)
```

```
2
```

the number 7 should be insterted on index 2 to remain the sort order. the methos strats the search from the right and returns the first index where number 7 is no longer less then next value

In [ ]:
```
"Multple values"
```

to search for more then one value ,use an array with the specified values

In [265…
```
#find the indexs for 2,4,5 where values should be inserted

import numpy as np
arr=np.array([1,2,3,4,5,6,7])
x=np.searchsorted(arr,[2,3,5])
print(x)
```
```
[1 2 4]
```

The return value is an array: [1 2 3] containing the three indexes where 2, 4, 6 would be inserted in the original array to maintain the order.

In [ ]:
```

```

In [ ]:
```
"NumPy Sorting Arrays"
```

sorting means putting elements in ordered sequence.

ordered sequence is any sequence that has an order corresponding to elements like numeric or alphanumeric ,ascending or descending.

The numpy ndarray object has a function called sort(),that will sort a specified array.

In [30]:
```
import numpy as np
arr=np.array([1,2,33,32,32,3,2,4,5])
x=np.sort(arr)
print(x)
```
```
[ 1  2  2  3  4  5 32 32 33]
```

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

In [ ]:
```
we can also sort arrays of strings or any other data type:
```

In [33]:
```
arr=np.array(["srinu","ravi","sai"])
print(np.sort(arr))
```
```
['ravi' 'sai' 'srinu']
```

In [ ]:
```
sort the boolean array
```

In [35]:
```
arr=np.array([True,False,True])
print(np.sort(arr))
```
```
[False  True  True]
```

In [ ]:
```
Sorting 2-D arrays:
```

if we you sort() method on 2-D arrays it will sort both arrays

In [40]:
```
arr=np.array([[1,5,3,4],[4,5,8,2]])
print(np.sort(arr))
```
```
[[1 3 4 5]
 [2 4 5 8]]
```

In [ ]:
```
"NumPy filtering Array"
```

filtering Arrays: getting some elements out of an existing array and creating a new array out of them is called filtering

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

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

In [69]:
```
import numpy as np
arr=np.array([41,42,43,44])
x=[True,False,True,False]
newarr=arr[x]
print(newarr)
```
```
[41 43]
```

The example above will return [41, 43], why?

Because the new array contains only the values where the filter array had the value True, in this case, index 0 and 2.

In [ ]:
```
Creating the Filter Array
```

In [ ]:
```
in the above example we hard-coded the True and False values ,but the common use it to create a filter based on
```

In [74]:
```
import numpy as np
arr=np.array([41,42,43,43,45,56])

#create an empty list
filter_arr=[]

#go through each element in arr
for i in arr:
    #if the element is higher then 43 set the value to True otherwise false
    if i>43:
        filter_arr.append(True)
    else:
        filter_arr.append(False)

newarr=arr[filter_arr]
print(filter_arr)
print(newarr)
```

```
[False, False, False, False, True, True]
[45 56]
```

In [77]:
```
# ex2:create a filter that will return only even values from the original values

arr=np.array([1,3,4,5,6,3,7,44,88,7,4,5,33,53,5,535,54])

filter_arr=[]
for i in arr:
    if i%2==0:
        filter_arr.append(True)
    else:
        filter_arr.append(False)

newarr=arr[filter_arr]
print(filter_arr)
print(newarr)
```

```
[False, False, True, False, True, False, False, True, True, False, True, False, False, False, False, False, True
]
[ 4  6 44 88  4 54]
```

In [79]:
```
# ex2:create a filter that will return only odd values from the original values

arr=np.array([1,3,4,5,6,3,7,44,88,7,4,5,33,53,5,535,54])

filter_arr=[]
for i in arr:
    if i%2!=0:
        filter_arr.append(True)
    else:
        filter_arr.append(False)

newarr=arr[filter_arr]
print(filter_arr)
print(newarr)
```

```
[True, True, False, True, False, True, True, False, False, True, False, True, True, True, True, True, False]
[  1   3   5   3   7   7   5  33  53   5 535]
```

In [ ]:
```
Creating Filter Directly From Array
```

the above example is quite a common task in numpy and numpy provides a nice to tackle it.

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

In [86]:
```
import numpy as np

arr=np.array([32,32,53,53,43,53,43,32])
filter_arr=arr>40
newarr=arr[filter_arr]
print(filter_arr)
print(newarr)
```

```
[False False  True  True  True  True  True False]
[53 53 43 53 43]
```

In [88]:
```
#even check
arr=np.array([1,2,3,4,5,6,7,8])
filter_arr=arr%2==0
newarr=arr[filter_arr]
print(filter_arr)
print(newarr)
```

```
[False  True False  True False  True False  True]
[2 4 6 8]
```

```
In [90]: #odd check
         arr=np.array([1,2,3,4,5,6,7,8])
         filter_arr=arr%2!=0
         newarr=arr[filter_arr]
         print(filter_arr)
         print(newarr)

[ True False  True False  True False  True False]
[1 3 5 7]
```

```
In [1]: from nbconvert import PDFExporter
```

```
In [3]: pip install nbconvert

Requirement already satisfied: nbconvert in c:\users\gadam\anaconda3\lib\site-packages (7.10.0)
Requirement already satisfied: beautifulsoup4 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (4.
12.3)
Requirement already satisfied: bleach!=5.0.0 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (4.1
.0)
Requirement already satisfied: defusedxml in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (3.1.4
)
Requirement already satisfied: jupyter-core>=4.7 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert)
(5.7.2)
Requirement already satisfied: jupyterlab-pygments in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert
) (0.1.2)
Requirement already satisfied: markupsafe>=2.0 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (2
.1.3)
Requirement already satisfied: mistune<4,>=2.0.3 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert)
(2.0.4)
Requirement already satisfied: nbclient>=0.5.0 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (0
.8.0)
Requirement already satisfied: nbformat>=5.7 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (5.9
.2)
Requirement already satisfied: packaging in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (23.2)
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\gadam\anaconda3\lib\site-packages (from nbconver
t) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (2
.15.1)
Requirement already satisfied: tinycss2 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (1.2.1)
Requirement already satisfied: traitlets>=5.1 in c:\users\gadam\anaconda3\lib\site-packages (from nbconvert) (5.
14.3)
Requirement already satisfied: six>=1.9.0 in c:\users\gadam\anaconda3\lib\site-packages (from bleach!=5.0.0->nbc
onvert) (1.16.0)
Requirement already satisfied: webencodings in c:\users\gadam\anaconda3\lib\site-packages (from bleach!=5.0.0->n
bconvert) (0.5.1)
Requirement already satisfied: platformdirs>=2.5 in c:\users\gadam\anaconda3\lib\site-packages (from jupyter-cor
e>=4.7->nbconvert) (3.10.0)
Requirement already satisfied: pywin32>=300 in c:\users\gadam\anaconda3\lib\site-packages (from jupyter-core>=4.
7->nbconvert) (305.1)
Requirement already satisfied: jupyter-client>=6.1.12 in c:\users\gadam\anaconda3\lib\site-packages (from nbclie
nt>=0.5.0->nbconvert) (8.6.0)
Requirement already satisfied: fastjsonschema in c:\users\gadam\anaconda3\lib\site-packages (from nbformat>=5.7-
>nbconvert) (2.16.2)
Requirement already satisfied: jsonschema>=2.6 in c:\users\gadam\anaconda3\lib\site-packages (from nbformat>=5.7
->nbconvert) (4.19.2)
Requirement already satisfied: soupsieve>1.2 in c:\users\gadam\anaconda3\lib\site-packages (from beautifulsoup4-
>nbconvert) (2.5)
Requirement already satisfied: attrs>=22.2.0 in c:\users\gadam\anaconda3\lib\site-packages (from jsonschema>=2.6
->nbformat>=5.7->nbconvert) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in c:\users\gadam\anaconda3\lib\site-package
s (from jsonschema>=2.6->nbformat>=5.7->nbconvert) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in c:\users\gadam\anaconda3\lib\site-packages (from jsonschem
a>=2.6->nbformat>=5.7->nbconvert) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in c:\users\gadam\anaconda3\lib\site-packages (from jsonschema>=2.
6->nbformat>=5.7->nbconvert) (0.10.6)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\gadam\anaconda3\lib\site-packages (from jupyte
r-client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.9.0.post0)
Requirement already satisfied: pyzmq>=23.0 in c:\users\gadam\anaconda3\lib\site-packages (from jupyter-client>=6
.1.12->nbclient>=0.5.0->nbconvert) (25.1.2)
Requirement already satisfied: tornado>=6.2 in c:\users\gadam\anaconda3\lib\site-packages (from jupyter-client>=
6.1.12->nbclient>=0.5.0->nbconvert) (6.4.1)
Note: you may need to restart the kernel to use updated packages.
```

```
In [5]: from nbconvert import PDFExporter
```

```
In [7]: pdf_exporter_latex = PDFExporter()
        pdf_exporter_latex.template_name = 'classic'
```

```
In [19]: pdf_data_latex, _ = pdf_exporter_latex.from_notebook_node(')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[19], line 1
----> 1 pdf_data_latex, _ = pdf_exporter_latex.from_notebook_node('numpys.ipynb')

File ~\anaconda3\Lib\site-packages\nbconvert\exporters\pdf.py:187, in PDFExporter.from_notebook_node(self, nb, r
esources, **kw)
    185 def from_notebook_node(self, nb, resources=None, **kw):
    186     """Convert from notebook node."""
--> 187     latex, resources = super().from_notebook_node(nb, resources=resources, **kw)
    188     # set texinputs directory, so that local files will be found
    189     if resources and resources.get("metadata", {}).get("path"):

File ~\anaconda3\Lib\site-packages\nbconvert\exporters\latex.py:75, in LatexExporter.from_notebook_node(self, nb
, resources, **kw)
     73 def from_notebook_node(self, nb, resources=None, **kw):
     74     """Convert from notebook node."""
---> 75     langinfo = nb.metadata.get("language_info", {})
     76     lexer = langinfo.get("pygments_lexer", langinfo.get("name", None))
     77     highlight_code = self.filters.get(
     78         "highlight_code", Highlight2Latex(pygments_lexer=lexer, parent=self)
     79     )

AttributeError: 'str' object has no attribute 'metadata'
```

```
In [ ]:
```