01_NUMPY

Monday, 14 June, 2021 09:14 PM

Deepankar Sharma

NumPy

NumPy (**Num**erical **Py**thon) is a Python library used to work with numerical data.
NumPy includes functions and data structures that can perform a wide variety of mathematical operations.

```python
# importing the numpy module
import numpy as np
print(np.__version__)
```

```python
# creating the numpy array
x = np.array([1, 2, 3, 4])
print(x)
print(type(x))
```

Output:

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

NumPy Arrays

In Python,

lists ----> store data.

NumPy provides an **array** structure for performing operations with data.
NumPy arrays are faster and more compact than lists.

```python
# value at index 0
print(x[0])
```

A NumPy array can be created using the **np.array()** function.
NumPy arrays are **homogeneous**, meaning they can contain only a single data type, while lists can contain multiple different types of data.

NumPy arrays are indexed , starting from the index 0.
Numpy arrays are also called ndarrays -----> multiple dimensions.

```python
# create a NumPy array of 3*3
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(x[1][2]) # print value at row 2 and column 3 ---> 6
```

Properties of NumPY arrays

```python
# defining the array
x = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(x.ndim) # prints the number of the dimensions in the array ===> 2
print(x.size) # prints number of elements in the array ===> 9
print(x.shape) # prints the shape of the array ===> (3, 3)
```

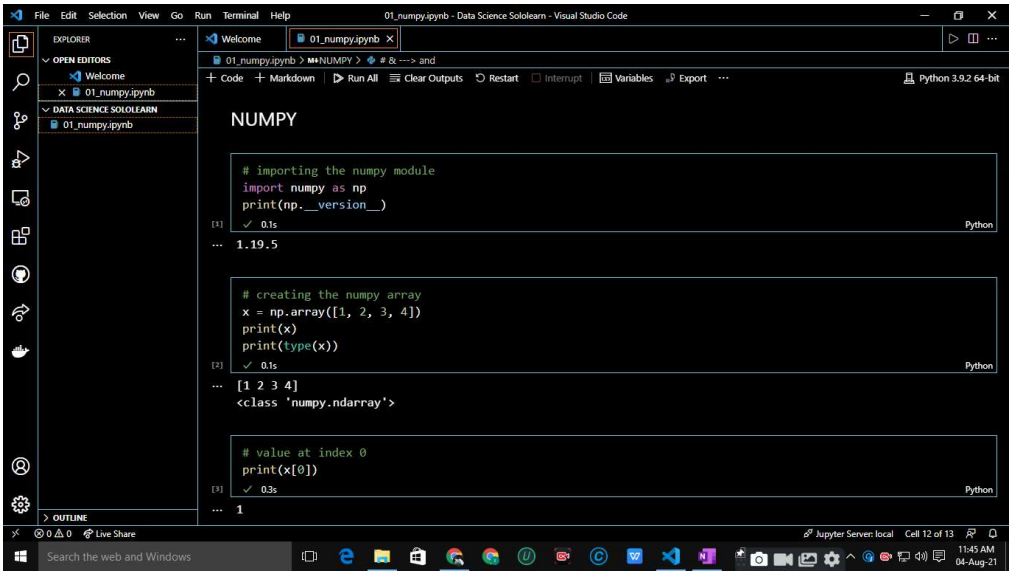
```python
# defining the array
x = np.array([2, 9, 3])
x = np.append(x, 2) # appends 4 to the array x
x = np.delete(x, 0) # delete the element at index 0 of the array x ----> 2
x = np.sort(x) # sorts the array X -----> [2 3 9]
print(x) # prints the array ----> [2 3 9]
```

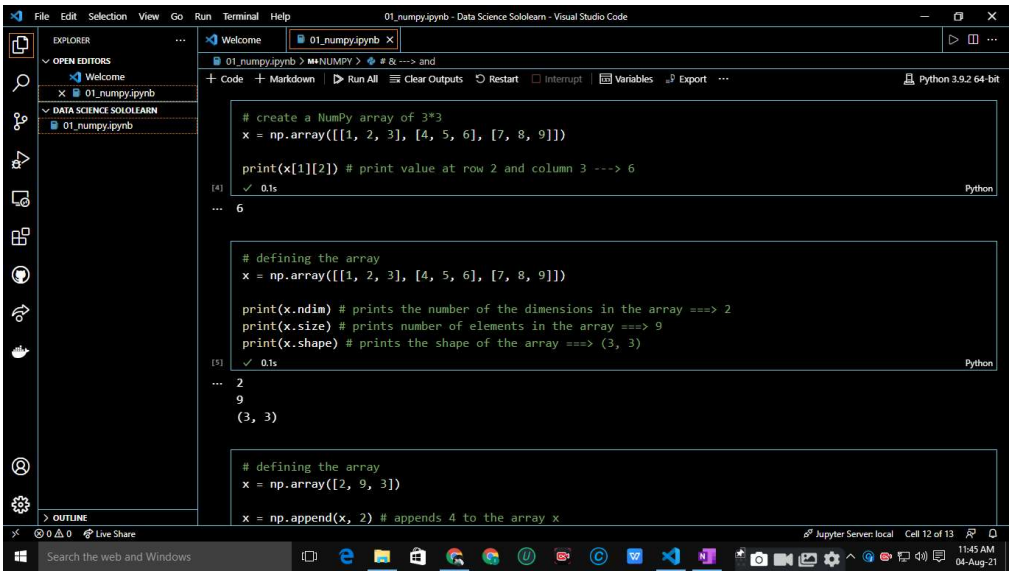
```python
# creates an array that contains elements in range(2, 10, 3)
x = np.arange(2, 20, 3)
print(x) # prints the array ---> [ 2  5  8 11 14 17]
```

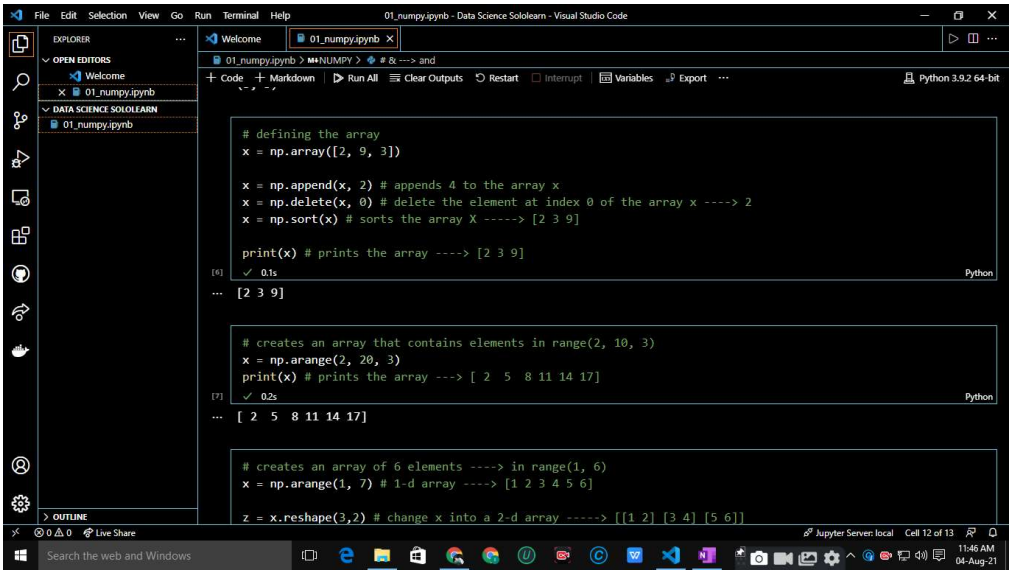
```python
# creates an array of 6 elements ----> in range(1, 6)
x = np.arange(1, 7) # 1-d array ----> [1 2 3 4 5 6]
z = x.reshape(3,2) # change x into a 2-d array -----> [[1 2] [3 4] [5 6]]
```

```python
# Indexing and Slicing
# creates an array of 9 elements in -----> range(1, 10)
x = np.arange(1, 10)
print(x[0:2]) # prints ----> [1 2]
print(x[5:]) # prints ----> [6 7 8 9]
print(x[:2]) # prints ----> [1 2]
print(x[-3:]) # prints ----> [7 8 9]
```

NumPy understands that the given operation should be performed with each element. This is called **broadcasting**.







```python
# creates an array of 6 elements ----> in range(1, 6)
x = np.arange(1, 7) # 1-d array ----> [1 2 3 4 5 6]

z = x.reshape(3,2) # change x into a 2-d array -----> [[1 2] [3 4] [5 6]]
```

```python
# Indexing and Slicing
# creates an array of 9 elements in -----> range(1, 10)
x = np.arange(1, 10)

print(x[0:2]) # prints ----> [1 2]
print(x[5:]) # prints ----> [6 7 8 9]
print(x[:2]) # prints ----> [1 2]
print(x[-3:]) # prints ----> [7 8 9]
```

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

```python
x = np.arange(1, 10) # creates an array of elements 1-9
print(x)
```

```python
x = np.arange(1, 10) # creates an array of elements 1-9
print(x)
print(x[x%2==0]) # prints the even elements in the array
```

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

```python
# & ---> and
# | ---> or
y = np.arange(31) # creates an array of elements 0-30
print(y)
print(y[(y%3==0)&(y%4==0)]) # prints the elements in the array which are multiple of both 4 and 3
print(y[(y%3==0)|(y%4==0)]) # prints the elements in the array which are multiple of both 4 or 3
```

```
[ 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 30]
[ 0 12 24]
[ 0  3  4  6  8  9 12 15 16 18 20 21 24 27 28 30]
```

```python
x = np.arange(1, 10) # creates an array in ----> range(1, 10)
print(x)
print(x.sum()) # prints sum of all the elements of the array
print(x.min()) # prints the minimum element in the array
print(x.max()) # prints the maximum element in the array
print(x*2) # prints the 2*value of each element of the array
```

```
[1 2 3 4 5 6 7 8 9]
45
1
9
[ 2  4  6  8 10 12 14 16 18]
```

```python
print(np.mean(x)) # prints the mean of all the elements in the array
print(np.median(x)) # prints the median of all the elements in the array
print(np.var(x)) # prints the variance of all the elements in the array
print(np.std(x)) # prints the standard deviation of all the elements in the array
```

```
5.0
5.0
6.666666666666667
2.581988897471611
```

```python
import numpy as np
print(np.__version__)
```

```
1.19.5
```

```python
print(np.array([1,2,3,4,5])) # convert the list into numpy array
print(np.array((1,2,3,4,5))) # convert the tuple into numpy array
```

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

```python
# np.arange(start, stop, step)
# create range of values in row wise order

val1 = np.arange(50, 100, 3)
print(val1)
```

```
[50 53 56 59 62 65 68 71 74 77 80 83 86 89 92 95 98]
```

```python
tup1 = ('a', 'e', 'i', 'o', 'u') # tuple of alphabets
```

```python
tup1 = ('a', 'e', 'i', 'o', 'u') # tuple of alphabets
tup2 = (1, 2, 3, 4, 5) # tuple of numbers
np12 = np.array((tup1, tup2)) # numpy array of two tuples
print(np12)
```

```
[['a' 'e' 'i' 'o' 'u']
 ['1' '2' '3' '4' '5']]
```

```python
# np.zeroes(shape)
# to create an array having all values equal to 0
val1 = np.zeros(8) # an one dimensional array with 8 elements as zeroes
print('1D array: ', val1)
val2 = np.zeros((8,8)) # an two dimensional array with 8*8 elements as zeroes
print('\n2D array: \n', val2)
val3 = np.zeros((8, 8, 8)) # an three dimensional array with 8*8*8 elements as zeroes
print('\n3D array: \n', val3)
```

```
1D array:  [0. 0. 0. 0. 0. 0. 0. 0.]

2D array:
 [[0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
```

```python
# np.ones(shape)
# to create an array having all values equal to 1
val1 = np.ones(8)  # an one dimensional array with 8 elements as ones
print('1D array: ', val1)
val2 = np.ones((8, 8))  # an two dimensional array with 8*8 elements as ones
print('\n2D array: \n', val2)
# an three dimensional array with 8*8*8 elements as ones
val3 = np.ones((8, 8, 8))
print('\n3D array: \n', val3)
```

```python
val = np.ones((4, 5)) # a 4*5 array will all values equal to 1
val*=1 # a 4*5 array with all values equal to 11
print(val)
```

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

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

```python
# np.linspace(start, stop, num)
# it equally divides the start and stop 'num' times and stores them in the form of an array
val = np.linspace(1, 50, 100) # divides 1 and 50 into 100 equal parts
print(val)
```

```
[ 1.          1.49494949  1.98989899  2.48484848  2.97979798  3.47474747
  3.96969697  4.46464646  4.95959596  5.45454545  5.94949495  6.44444444
  6.93939394  7.43434343  7.92929293  8.42424242  8.91919192  9.41414141
  9.90909091 10.4040404  10.8989899  11.39393939 11.88888889 12.38383838
 12.87878788 13.37373737 13.86868687 14.36363636 14.85858586 15.35353535
 15.84848485 16.34343434 16.83838384 17.33333333 17.82828283 18.32323232
 18.81818182 19.31313131 19.80808081 20.3030303  20.7979798  21.29292929
 21.78787879 22.28282828 22.77777778 23.27272727 23.76767677 24.26262626
 24.75757576 25.25252525 25.74747475 26.24242424 26.73737374 27.23232323
 27.72727273 28.22222222 28.71717172 29.21212121 29.70707071 30.2020202
 30.6969697  31.19191919 31.68686869 32.18181818 32.67676768 33.17171717
 33.66666667 34.16161616 34.65656566 35.15151515 35.64646465 36.14141414
 36.63636364 37.13131313 37.62626263 38.12121212 38.61616162 39.11111111
 39.60606061 40.1010101  40.5959596  41.09090909 41.58585859 42.08080808
 42.57575758 43.07070707 43.56565657 44.06060606 44.55555556 45.05050505
 45.54545455 46.04040404 46.53535354 47.03030303 47.52525253 48.02020202
 48.51515152 49.01010101 49.50505051 50.        ]
```

3.Numpy.3

```python
# print random values by different "functions"
# np.random.function(V1, V2, V3,......, Vn)

temp1 = np.random.rand(4) # sets 4 random values between 0 and 1
print(temp1)
```

```
[0.18635716 0.75853421 0.67368672 0.04996642]
```

```python
temp2 = np.random.randint(10, 100, size=12) # sets 12 random values between 10 and 100
print(temp2)
```

```
[93 97 52 91 21 47 76 78 11 54 10 44]
```

```python
arr1 = np.array([[1, 4, 8], [2, 3, 4], [8, 3, 8], [1, 3, 87]])
print(arr1)
```

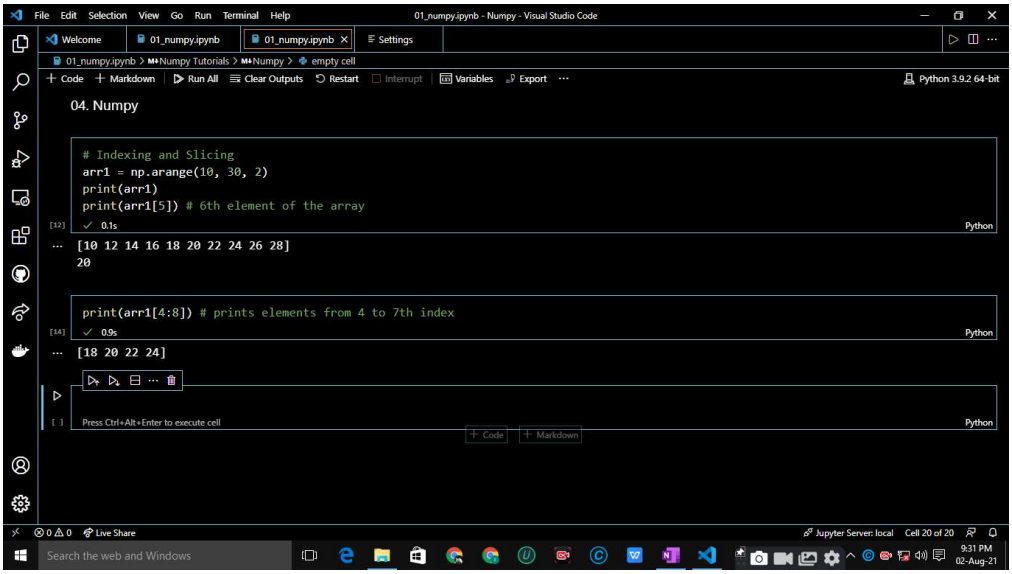
```
[[ 1  4  8]
 [ 2  3  4]
 [ 8  3  8]
 [ 1  3 87]]
```

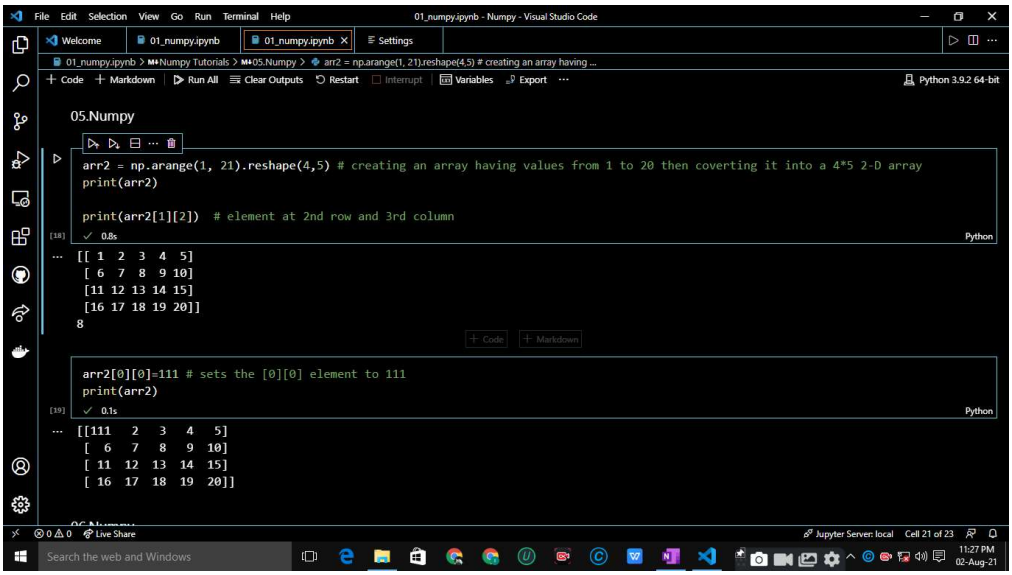
```python
print(arr1.reshape(2, 6)) # changes the shape of array from 4*2 to 2*6
```

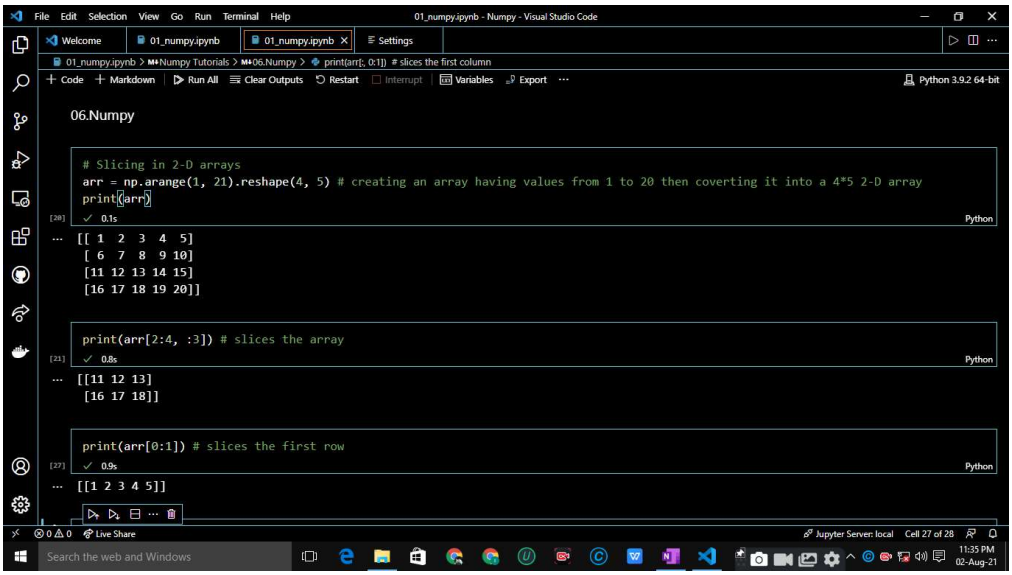
```
[[ 1  4  8  2  3  4]
 [ 8  3  8  1  3 87]]
```

```python
print(arr1.max()) # maximum value in the array 'arr1'
print(arr1.min()) # minimum value in the array 'arr1'
```

```
87
1
```


01_numpy.ipynb - Numpy - Visual Studio Code
04. Numpy
# Indexing and Slicing
arr1 = np.arange(10, 30, 2)
print(arr1)
print(arr1[5]) # 6th element of the array
[10 12 14 16 18 20 22 24 26 28]
20
print(arr1[4:8]) # prints elements from 4 to 7th index
[18 20 22 24]
Press Ctrl+Alt+Enter to execute cell


01_numpy.ipynb - Numpy - Visual Studio Code
05.Numpy
arr2 = np.arange(1, 21).reshape(4,5) # creating an array having values from 1 to 20 then coverting it into a 4*5 2-D array
print(arr2)
print(arr2[1][2]) # element at 2nd row and 3rd column
[[1 2 3 4 5]
[6 7 8 9 10]
[11 12 13 14 15]
[16 17 18 19 20]]
8
arr2[0][0]=111 # sets the [0][0] element to 111
print(arr2)
[[111 2 3 4 5]
[6 7 8 9 10]
[11 12 13 14 15]
[16 17 18 19 20]]


01_numpy.ipynb - Numpy - Visual Studio Code
06.Numpy
# Slicing in 2-D arrays
arr = np.arange(1, 21).reshape(4, 5) # creating an array having values from 1 to 20 then coverting it into a 4*5 2-D array
print(arr)
[[1 2 3 4 5]
[6 7 8 9 10]
[11 12 13 14 15]
[16 17 18 19 20]]
print(arr[2:4, :3]) # slices the array
[[11 12 13]
[16 17 18]]
print(arr[0:1]) # slices the first row
[[1 2 3 4 5]]

```
[[11 12 13]
 [16 17 18]]
```

```python
print(arr[0:1]) # slices the first row
```

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

```python
print(arr[:, 0:1])  # slices the first column
```

```
[[ 1]
 [ 6]
 [11]
 [16]]
```

07.Numpy

```python
arr=np.arange(1, 20, 3)
print(arr)
print(arr+5)  # increments each element of arr by 5
```

```
[ 1  4  7 10 13 16 19]
[ 6  9 12 15 18 21 24]
```

```python
arr1 = np.arange(1, 20, 3)
arr2 = np.arange(20, 40, 3)

print(arr1)
print(arr2)
print(arr1+arr2) # adds corresponding elements of arr1 and arr2  ( both arrays should have same number of elements !!! )
```

```
[ 1  4  7 10 13 16 19]
[20 23 26 29 32 35 38]
[21 27 33 39 45 51 57]
```

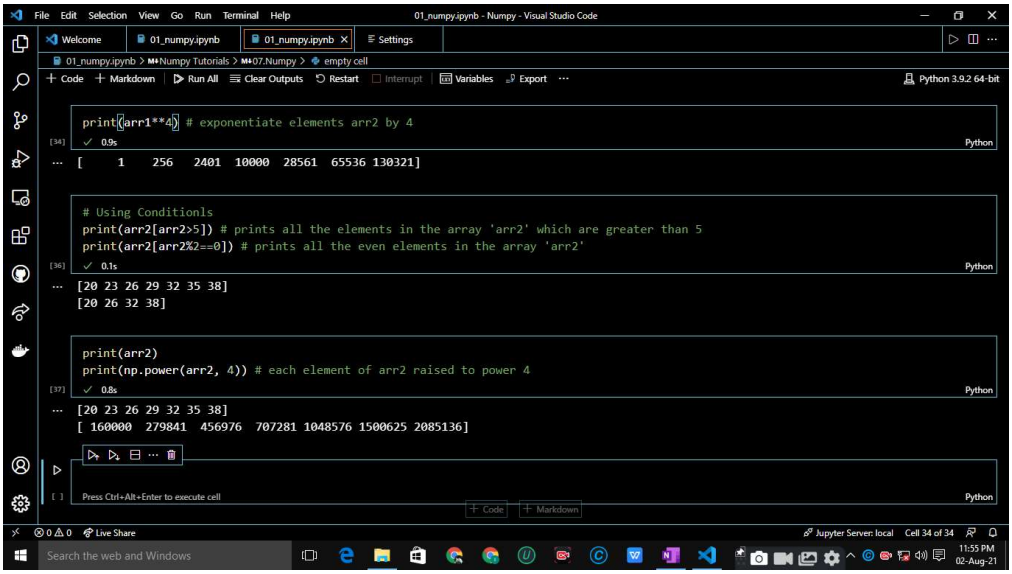
```python
arr1 = np.arange(1, 20, 3)
arr2 = np.arange(20, 40, 3)

print(arr1)
print(arr2)
print(arr1+arr2) # adds corresponding elements of arr1 and arr2  ( both arrays should have same number of elements !!! )
print(arr1-arr2) # subtracts corresponding elements of arr1 and arr2  ( both arrays should have same number of elements !!! )
print(arr1*arr2) # multiplies corresponding elements of arr1 and arr2  ( both arrays should have same number of elements !!! )
```

```
[ 1  4  7 10 13 16 19]
[20 23 26 29 32 35 38]
[21 27 33 39 45 51 57]
[-19 -19 -19 -19 -19 -19 -19]
[ 20  92 182 290 416 560 722]
```

```python
print(arr1**4) # exponentiate elements arr2 by 4
```

```
[     1    256   2401  10000  28561  65536 130321]
```


File Edit Selection View Go Run Terminal Help
01_numpy.ipynb - Numpy - Visual Studio Code
Welcome
01_numpy.ipynb
01_numpy.ipynb
Settings
01_numpy.ipynb > Numpy Tutorials > 07.Numpy > empty cell
Code Markdown Run All Clear Outputs Restart Interrupt Variables Export
Python 3.9.2 64-bit
print(arr1**4) # exponentiate elements arr2 by 4
[34] 0.9s
Python
[1 256 2401 10000 28561 65536 130321]
Using Conditionls
print(arr2[arr2>5]) # prints all the elements in the array 'arr2' which are greater than 5
print(arr2[arr2%2==0]) # prints all the even elements in the array 'arr2'
[36] 0.1s
Python
[20 23 26 29 32 35 38]
[20 26 32 38]
print(arr2)
print(np.power(arr2, 4)) # each element of arr2 raised to power 4
[37] 0.8s
Python
[20 23 26 29 32 35 38]
[160000 279841 456976 707281 1048576 1500625 2085136]
[ ]
Press Ctrl+Alt+Enter to execute cell
Python
Code Markdown
0 0 Live Share
Jupyter Server: local
Cell 34 of 34
Search the web and Windows
11:55 PM
02-Aug-21