2. Numpy_Practice

September 22, 2024

1 PRACTICE - NUMPY

1.0.1 Complete the following questions. Add additional code cells as needed (recommended).

Name: Mrudu Lahari Malayanur

[1]:
```
import numpy as np
```

1.0.2 Creating Arrays

[2]:
```
# QUESTION 1

# Create an 1d array from a list
# Print the array and its type
# Add 2 to each element of an array
# Explain the key difference between a list and an array
# What happens when you put an integer, a Boolean, and a string in the same␣
↪array. Illustrate.
```

[3]:
```
# QUESTION 1 : Create an 1d array from a list

list_1 = [1, 2, 3, 4,5]
array1 = np.array(list_1)

# Print the array and its type
print(array1, "\ntype", array1.dtype)
```

```
[1 2 3 4 5]
type int64
```

[4]:
```
# Add 2 to each element of an array
array2 = array1 +2
print(array2)
```

```
[3 4 5 6 7]
```

#Explain the key difference between a list and an array

The main difference is that Arrays are homogeneous which means all elements in an array are of same data type, list can contain differnt type of elements

[8]:
```
#What happens when you put an integer, a Boolean, and a string in the same␣
↪array. Illustrate.
array3=[False, 2, 'any string']
print(type(array3[0]))
print(type(array3[1]))
print(type(array3[2]))
```

```
<class 'bool'>
<class 'int'>
<class 'str'>
```

[10]:
```
#the elements co exists while maintaining their data type
#because arrays are hetrogeneous in nature
```

[11]:
```
# QUESTION 2

# Create a 2d array from a list of lists
# Create a float 2d array
# Convert to 'int' datatype
# Convert to int then to str datatype
# # Create an object to hold numbers as well as strings
# If you want your array will hold characters and numbers in the same array,␣
↪what will you set you datatype as...
# Convert an array back to a list
```

[12]:
```
# Create a 2d array from a list of lists

list_of_lists = [[1,2,3], [4,5,6], [7,8,9]]
array2d = np.array(list_of_lists)
print(array2d)
```

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

[13]:
```
array_float = np.array([[1.5, 2.2, 9.6], [8.3,3.3, 1.1]])
print(array_float)
```

```
[[1.5 2.2 9.6]
 [8.3 3.3 1.1]]
```

[14]:
```
array_int = array_float.astype(int)
print(array_int)
```

```
[[1 2 9]
```

```
 [8 3 1]]
```

[15]:
```
array_str = array_float.astype(int).astype(str)
print(array_str)
```

```
[['1' '2' '9']
 ['8' '3' '1']]
```

[16]:
```
array_obj = np.array([1, 2, 3, 'four', 'five', 'six'], dtype = object)
print(array_obj)
```

```
[1 2 3 'four' 'five' 'six']
```

[17]:
```
# If you want your array will hold characters and numbers in the same array,␣
 ↪what will you set you datatype as...
#since an array can hold any data type as its elements, Specific type of␣
 ↪assignment is not required
```

[22]:
```
#using tolist to chanage the data type of array2d back to list.
#using type function to check the new datatype
array2d=array2d.tolist()
print(type(array2d))
```

```
<class 'list'>
```

1.0.3 Inspecting size and shape

[ ]:
```
# QUESTION 3

# Create a 2d array with 3 rows and 4 columns
# Print the shape of the array. Explain the output
# Print the size of the array. Explain the output
# Print the dimensions of the array. Explain the output.
```

[23]:
```
array3 = np.arange(1,13).reshape(3,4)
print(array3)
```

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

[35]:
```
#using shape function to get the shape array.
#The output 3, 4 shows that array3 is 3 rows and 4 columns
array3.shape
```

[35]:
```
(3, 4)
```

[36]:
```
#size is used to define number of items present in the array.
#Output 12 defines the total number of items present in the array
array3.size
```

[36]:
```
12
```

[38]:
```
#dimension states if the array is 1D array or 2D array.
#Using ndim function to get the dimensions which is 2 here.
array3.ndim
```

[38]:
```
2
```

[39]:
```
arr_2d=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]], dtype=float)
subset = arr_2d[:2, :2]
print(subset)
#  we are creating a new array subset by slicing arr_2d.
#The [:2, :2] notation means we're selecting the first two rows and first two␣
↪columns of arr_2d.
```

```
[[1. 2.]
 [5. 6.]]
```

[40]:
```
condition = arr_2d > 5
print(condition)
#We're creating a new array condition that contains Boolean values.
#Each element of condition will be True if the corresponding element in arr_2d␣
↪is greater than 5, and False otherwise.
```

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

[41]:
```
reversed_rows = arr_2d[::-1, :]
print(reversed_rows)
# The ::-1 notation means to reverse the order of rows. The : in the second␣
↪position means to select all columns.
```

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

[42]:
```
arr_2d[1, 1] = np.nan
arr_2d[2, 2] = np.inf
print(arr_2d)
# In these two lines, we're setting specific elements of arr_2d to special␣
↪values,
# arr_2d[1, 1] is setting the element at the second row and second column to np.
↪nan, which represents "Not a Number"
#(a way to represent missing or undefined values in numerical computations).
# arr_2d[2, 2] is setting the element at the third row and third column to np.
↪inf, which represents positive infinity.
```

```
[[ 1.  2.  3.  4.]
 [ 5. nan  7.  8.]
 [ 9. 10. inf 12.]]
```

[43]:
```
arr_2d[np.isnan(arr_2d)] = -1
arr_2d[np.isinf(arr_2d)] = -1
print(arr_2d)
# In these two lines, we're using boolean indexing to find and replace specific␣
↪values in arr_2d. np.isnan(arr_2d) creates a boolean mask where True␣
↪corresponds to elements that are np.nan.
#Then, we're using this mask to replace those elements with -1.
#Similarly, np.isinf(arr_2d) creates a boolean mask for elements that are np.
↪inf,and we're replacing them with -1.

#After running these lines, arr_2d will have -1 in place of any np.nan or np.
↪inf values.
```

```
[[ 1.  2.  3.  4.]
 [ 5. -1.  7.  8.]
 [ 9. 10. -1. 12.]]
```

1.0.4 Extracting Items, Reversing Rows and adding Missing Values

[ ]:
```
# QUESTION 4

# Using the array in Q3 extract the first 2 rows and columns
# Get a boolean output by applying the condition to each element.
# Reverse only the row positions
# Reverse the row and column positions
# # Replace nan and inf with -1.
```

[48]:
```
#starting from 0 to 2 as indexing for rows, to get 0th and 1st row, and all␣
↪columns as required
arr_2d[:2,]
```

[48]:
```
array([[ 1.,  2.,  3.,  4.],
       [ 5., -1.,  7.,  8.]])
```

[51]:
```
#to get boolean array on applying conditions, using > logical operator
arr_2d>2.0
```

```
#False denotes that, value present in that index is not > 2.0. True denotes␣
↪that it is greated than 2.0
```

```
[51]: array([[False, False,  True,  True],
             [ True, False,  True,  True],
             [ True,  True, False,  True]])
```

```
[67]: #Reverse only the row positions
      arr_2d[::-1,::]
      #The ::-1 notation means to reverse the order of rows.
      #The : in the second position means to select all columns.
```

```
[67]: array([[ 9., 10., -1., 12.],
             [ 5., -1.,  7.,  8.],
             [ 1.,  2.,  3.,  4.]])
```

```
[68]: # Reverse the row and column positions. appying ::-1, ::-1 will reverse both␣
      ↪rows and columns
      arr_2d[::-1, ::-1]
```

```
[68]: array([[12., -1., 10.,  9.],
             [ 8.,  7., -1.,  5.],
             [ 4.,  3.,  2.,  1.]])
```

```
[73]: #inserting nan and inf in the 2x2 index and 2x3 index respectively
      arr_2d[2, 2] = np.nan
      arr_2d[2, 3] = np.inf
      arr_2d
```

```
[73]: array([[ 1.,  2.,  3.,  4.],
             [ 5., -1.,  7.,  8.],
             [ 9., 10., nan, inf]])
```

```
[75]: #replacing nan and inf using isnan and isinf
      arr_2d[np.isnan(arr_2d)] = -1
      arr_2d[np.isinf(arr_2d)] = -1
      arr_2d
```

```
[75]: array([[ 1.,  2.,  3.,  4.],
             [ 5., -1.,  7.,  8.],
             [ 9., 10., -1., -1.]])
```

1.0.5 Creating a new array from existing array

[76]:
```
# QUESTION 5

# Explain the precautions you need to take when assigning a portion of an array␣
↪to a new array. Illustrate with an example.
# What are some approaches you can take to keep the parent array intact?␣
↪Illustrate with examples.
```

[78]:
```
arrayexisting = np.arange(0, 10).reshape(2, 5)
print(arrayexisting)
arraynew = arrayexisting[:1, :3]
print(arraynew)
arraynew[:, 1] = 9
print(arrayexisting)
```

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

[79]:
```
arrayold = np.arange(0, 10).reshape(2, 5)
print(arrayold)
arraynew = arrayold.copy()
print(arraynew)
arraynew[:, 1] = 9
print(arrayold)
print(arraynew)
```

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

In the above case, shallow copy has been made. Where the objects are being referenced to the same memory and so any change to new array will affect the old array as well. Hence we should be careful in copying the array and making changes to it.

1.0.6 Reshaping and Flattening

[80]:
```
# QUESTION 6

# Describe what is reshaping an array. Illustrate with an example.
# Describe what is flattening an array. Illustrate with an example.
# What are the two methods for flattening an array? Illustrate both methods␣
↪with an example.
# What are the differences between these two methods.
```

[85]:
```
#Create a (2,4) array
arr_shape=np.array([[1,2,4,5],[4,5,6,7]])
print(arr_shape)
#Reshaping an array is changing the structure or dimensions of an array without␣
↪changing its data

#Reshape to a (4,2) array (keeps all data in order when reshaping)
print(arr_shape.reshape(4,2))

#Or use T (transpose)- each row is reshaped into a single column
print(arr_shape.T)
```

```
[[1 2 4 5]
 [4 5 6 7]]
[[1 2]
 [4 5]
 [4 5]
 [6 7]]
[[1 4]
 [2 5]
 [4 6]
 [5 7]]
```

[86]:
```
#converting multi dimensional array to 1D array is called flattening.

arr_shape=np.array([[1,2,4,5],[4,5,6,7]])
print(arr_shape)

print(arr_shape.flatten())
```

```
[[1 2 4 5]
 [4 5 6 7]]
[1 2 4 5 4 5 6 7]
```

[82]:
```
#flatten() and #ravel()
print(arr_shape.ravel())
print(arr_shape.flatten())
```

```
[1 2 4 5 4 5 6 7]
[1 2 4 5 4 5 6 7]
```

QN: What are the two methods for flattening an array?

#flatten() and #ravel() are the 2 methods.

QN: What are the differences between these two methods.

The main difference is that, in flatten, The deep copy of original array is made so any change made to new array does not change original array. In ravel method, a view of original array is made hence any change made to new array changes old array as well

1.0.7 Creating Sequences, repetitions and random numbers

[89]:
```
# QUESTION 7



# Create an array with numbers 0 to 9
g = np.arange(0,10)

# Create an array 0 to 9 with step of 2
h = np.arange(0,10,2)

# Create an array 10 to 1, decreasing order
i = np.arange(0,10,-1)

# Create an array with zeros or ones
j = np.zeros((2,3))

'''
SIDENOTE: If you want to create an array of exactly 10 numbers between 1 and 50␣
  ↪you can use np.linspace.

np.linspace(start=1, stop=50, num=10, dtype=int)
array([ 1,  6, 11, 17, 22, 28, 33, 39, 44, 50])

Note the dtype is forced as type int



To repeat an entire array n times use np.tile()
If you have an array assign to a variable a. To repeat the array twice ...np.
  ↪tile(a, 2)

To repeat an element n times use np.repeat()
To repeat each element in the array a twice  ... np.repeat(a, 2))

'''
```

```python
# Create an array using np.linspace()
k = np.linspace(start=1, stop=20, num=8, dtype=int)
print(k)

# Repeat your array 4 times
k = np.tile(k,4)
print(k)

# Repeat each element in the array 3 times
k = np.repeat(k,3)
print(k)

# Create an array with random numbers between [0,1) of shape 2,2
l = np.random.random((2,2))
print(l)

# Create an array with random integers between [0, 10) of shape 2,2
m = np.random.random((2,2))*10
print(m)
```

```
[ 1  3  6  9 11 14 17 20]
[ 1  3  6  9 11 14 17 20  1  3  6  9 11 14 17 20  1  3  6  9 11 14 17 20
  1  3  6  9 11 14 17 20]
[ 1  1  1  3  3  3  6  6  6  9  9  9 11 11 11 14 14 14 17 17 17 20 20 20
  1  1  1  3  3  3  6  6  6  9  9  9 11 11 11 14 14 14 17 17 17 20 20 20
  1  1  1  3  3  3  6  6  6  9  9  9 11 11 11 14 14 14 17 17 17 20 20 20
  1  1  1  3  3  3  6  6  6  9  9  9 11 11 11 14 14 14 17 17 17 20 20 20]
[[0.04672101 0.50825577]
 [0.76042308 0.93955876]]
[[9.55355034 9.47407892]
 [5.99964602 5.93240133]]
```