Numpy_Practice

October 7, 2024

1 Name: Ramya Chowdary Patchala

2 PRACTICE - NUMPY

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

[1]:

```python
import numpy as np
```

2.0.2 Creating Arrays

[ ]:

```python
# 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.
```

[2]:

```python
# 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
```

[3]:

```python
# Add 2 to each element of an array
array2 = array1 +2
print(array2)
```

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

[13]:
```
l3 = [1, True, "Ramya"]
a3 = np.array(l3)
print(a3)
print(l3)
```

```
['1' 'True' 'Ramya']
[1, True, 'Ramya']
```

2.0.3 Arrays can store only one kind of element. They change the type of data.

2.0.4 Lists can have anytype of data.

[ ]:
```

```

[ ]:
```
# 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
```

[15]:
```
# 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]]
```

[16]:
```
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]]
```

[17]:
```
array_int = array_float.astype(int)
print(array_int)
```

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

[18]:

```
array_str = array_float.astype(int).astype(str)
print(array_str)
```

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

[22]:

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

## IF we want array to hold all the types of data then you have to set datatype of array as object
```

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

[25]:

```
l4 = array_obj.tolist()
print(l4)
```

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

2.0.5 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.
```

[26]:

```
array3 = np.arange(1,13).reshape(3,4)
print(array3)
```

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

[33]:

```
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.]]
```

[34]:

```
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]]
```

[35]:

```
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.]]
```

[36]:

```
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.]]
```

[37]:

```
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.]]
```

2.0.6 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
# Insert a nan and an inf in positions of your choosing
# # Replace nan and inf with -1.
```

[40]:

```
subset = arr_2d[:2, :2]
print(subset)
```

```
[[ 1.  2.]
 [ 5. -1.]]
```

[41]:

```
condition = arr_2d > 1
print(condition)
```

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

[42]:

```
reversed_rows = arr_2d[::-1, :]
print(reversed_rows)
```

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

[44]:

```
reversed_rows = arr_2d[::-1, ::-1]
print(reversed_rows)
```

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

[45]:

```
reversed_rows[0, 1] = np.nan
reversed_rows[1, 2] = np.inf
print(reversed_rows)
```

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

[46]:

```
reversed_rows[np.isnan(reversed_rows)] = 11
reversed_rows[np.isinf(reversed_rows)] = 11
```

```
print(reversed_rows)
```

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

2.0.7 Creating a new array from existing array

[ ]:

```
# 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.
```

[47]:

```
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]]
```

[48]:

```
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]]
```

2.1 We use copy() so that we can keep parent array intact

[ ]:

[ ]:

[ ]:

2.1.1 Reshaping and Flattening

[ ]:

```
# 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.
```

2.1.2 Reshaping an array is basically changing how the data is arranged without changing the actual data itself.

[13]:

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

#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]]
```

2.1.3 Flattening an array is basically taking a multi-dimensional array and turning it into a single, long list of elements.

[14]:

```
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]
```

2.1.4 two methods for flattening an array - the flatten() method and the ravel() method.

2.1.5 flatten() always returns a copy of the array. ravel() returns a view of the original array when possible, and a copy only if necessary.

[15]:

```
#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]
```

2.1.6 Creating Sequences, repetitions and random numbers

[21]:

```
# 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))

'''

# 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]
[[0.12063039 0.79604236]
 [0.28599207 0.970994  ]]
[[3.029639   3.26387435]
 [9.53073421 2.43576676]]
```