numpyassignment

June 30, 2024

[37]:
```
import numpy as np
```

[38]:
```
# 1. Create a NumPy array 'arr' of integers from 0 to 5 and print its data type.
arr=np.array([0,1,2,3,4,5])
arr.dtype
```

[38]:
```
dtype('int64')
```

[39]:
```
# 2. Given a NumPy array 'arr', check if its data type is float64.
arr = np.array([1.5, 2.6, 3.7])
arr.dtype
```

[39]:
```
dtype('float64')
```

[40]:
```
# 3. Create a NumPy array 'arr' with a data type of complex128 containing three␣
↪complex numbers.
arr=np.array([1+2j,3+8j,8+23j], dtype=np.complex128)
arr.dtype
```

[40]:
```
dtype('complex128')
```

[41]:
```
# 4. Convert an existing NumPy array 'arr' of integers to float32 data type.
arr=np.array([1,2,3,4], dtype=np.int32)
arr.dtype
```

[41]:
```
dtype('int32')
```

[42]:
```
# 5. Given a NumPy array 'arr' with float64 data type, convert it to float32 to␣
↪reduce decimal precision.
arr=np.array([1.2,2.2,3.3], dtype=np.float64)
arr.dtype
arr_float32=arr.astype(np.float32)
arr_float32.dtype
```

[42]:
```
dtype('float32')
```

[43]:
```
# 6. Write a function array_attributes that takes a NumPy array as input and␣
 ↪returns its shape, size, and data
# type.
def array_attributes():
  print(arr.shape)
  print(arr.size)
  print(arr.dtype)
arr=np.array([[1,2,3],[4,5,6]])
array_attributes()
```

```
(2, 3)
6
int64
```

[44]:
```
# 7. Create a function array_dimension that takes a NumPy array as input and␣
 ↪returns its dimensionality
def array_dimension():
  return arr.ndim
arr=np.array([1,2,3,4])
array_dimension()
```

[44]: 1

[45]:
```
# 8. Design a function item_size_info that takes a NumPy array as input and␣
 ↪returns the item size and the total
# size in bytes.
def item_size_info():
  print(arr.size)
  print(arr.dtype.itemsize)
a=np.random.rand(12)
arr=np.array(a)
item_size_info()
```

```
12
8
```

[46]:
```
# 9. Create a function array_strides that takes a NumPy array as input and␣
 ↪returns the strides of the array.
def array_strides(arr):
  return arr.strides
arr=np.array([[1,2],[3,4],[4,5]])
array_strides(arr)
```

[46]: (16, 8)

[47]:
```
# 10. Design a function shape_stride_relationship that takes a NumPy array as␣
 ↪input and returns the shape
```

```
#and strides of the array.
def shape_stride_relationship(arr):
  print(arr.shape)
  print(arr.strides)
arr=np.array([[1,2],[3,4],[4,5]])
shape_stride_relationship(arr)
```

```
(3, 2)
(16, 8)
```

[48]:
```
# 11. Create a function `create_zeros_array` that takes an integer `n` as input␣
 ↪and returns a NumPy array of
# zeros with `n` elements.
def create_zeros_array(n):
  return np.zeros(n)
n=55
create_zeros_array(n)
```

[48]:
```
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., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0.])
```

[49]:
```
# 12. Write a function `create_ones_matrix` that takes integers `rows` and␣
 ↪`cols` as inputs and generates a 2D
# NumPy array filled with ones of size `rows x cols
def create_ones_matrix(rows,cols):
  return np.matrix((rows,cols))
rows=3
cols=4
create_ones_matrix(rows,cols)
```

[49]:
```
matrix([[3, 4]])
```

[50]:
```
# 13. Write a function `generate_range_array` that takes three integers start,␣
 ↪stop, and step as arguments and
# creates a NumPy array with a range starting from `start`, ending at stop␣
 ↪(exclusive), and with the specified
# `step`.):
import numpy as np

def generate_range_array(start, stop, step):
  return np.arange(start, stop, step)
start_val = 5
end_val = 20
step_val = 3
arr = generate_range_array(start_val, end_val, step_val)
```

```
print(arr)
```

```
[ 5  8 11 14 17]
```

[51]:
```
# 14. Design a function `generate_linear_space` that takes two floats `start`, `stop`, and an integer `num` as
# arguments and generates a NumPy array with num equally spaced values between `start` and `stop`
# (inclusive).
def generate_linear_space(start,stop, num):
  return np.linspace(start,stop,num, endpoint=True)
start=1.2
stop=88.2
num= 5
generate_linear_space(start,stop, num)
```

[51]:
```
array([ 1.2 , 22.95, 44.7 , 66.45, 88.2 ])
```

[52]:
```
# 15. Create a function `create_identity_matrix` that takes an integer `n` as input and generates a square
# identity matrix of size `n x n` using `numpy.eye`.
def create_identity_matrix(n):
  return np.eye(n)
n=5
create_identity_matrix(n)
```

[52]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

[53]:
```
# 16. Write a function that takes a Python list and converts it into a NumPy array.
l=[1,2,3,4,5,5]
np.array(l)
```

[53]:
```
array([1, 2, 3, 4, 5, 5])
```

[54]:
```
# 17. Create a NumPy array and demonstrate the use of `numpy.view` to create a new array object with the
# same data.
arr=np.array([1,2,3,4,5,6,7])
arr.view()
arr[4]=190
arr.view()
```

[54]:
```
array([  1,   2,   3,   4, 190,   6,   7])
```

[55]:
```
# 18. Write a function that takes two NumPy arrays and concatenates them along
 ↪a specified axis.
def  concatenates(a,b, axis=0):
  return np.concatenate((a,b), axis=axis)
a=np.array([1,2,3,4])
b=np.array([5,6,7,8])
concatenates(a,b)
```

[55]:
```
array([1, 2, 3, 4, 5, 6, 7, 8])
```

[56]:
```
# 19. Create two NumPy arrays with different shapes and concatenate them
 ↪horizontally using `numpy.

array1 = np.array([[1, 2, 3],
                   [4, 5, 6]])

array2 = np.array([[7, 8],
                   [9, 10]])

np.hstack((array1, array2))
```

[56]:
```
array([[ 1,  2,  3,  7,  8],
       [ 4,  5,  6,  9, 10]])
```

[57]:
```
# 20. Write a function that vertically stacks multiple NumPy arrays given as a
 ↪list.
def stack_arrays_vertically(arrays):
  return np.vstack(arrays)
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr3 = np.array([7, 8, 9])
stacked_arr = stack_arrays_vertically([arr1, arr2, arr3])
print(stacked_arr)
```

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

[58]:
```
# 21. Write a Python function using NumPy to create an array of integers within
 ↪a specified range (inclusive)
# with a given step size
def create_integer_array(start, stop, step):
  return np.arange(start, stop + 1, step)
start_val = 5
end_val = 20
```

```
step_val = 3
arr = create_integer_array(start_val, end_val, step_val)
print(arr)
```

```
[ 5  8 11 14 17 20]
```

[59]:
```
# 22. Write a Python function using NumPy to generate an array of 10 equally␣
 ↪spaced values between 0 and 1
# (inclusive).
def Spaced(start,stop,num):
  return np.linspace(start,stop,num, endpoint=True)
a=0.0
b=1.0
c=10
Spaced(a,b,c)
```

[59]:
```
array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])
```

[60]:
```
# 23. Write a Python function using NumPy to create an array of 5␣
 ↪logarithmically spaced values between 1 and
# 1000 (inclusive).
def generate_logarithmic_array(start, stop, num):

  return np.logspace(np.log10(start), np.log10(stop), num, endpoint=True)
start_val = 1.0
end_val = 1000.0
num_elements = 5
arr = generate_logarithmic_array(start_val, end_val, num_elements)
print(arr)
```

```
[   1.            5.62341325   31.6227766   177.827941   1000.        ]
```

[61]:
```
# 24. Create a Pandas DataFrame using a NumPy array that contains 5 rows and 3␣
 ↪columns, where the values
# are random integers between 1 and 100.
import pandas as pd
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15]])
df=pd.DataFrame(arr)
df
```

[61]:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12
4	13	14	15

[62]:
```
# Important 25. Write a function that takes a Pandas DataFrame and replaces all␣
↪negative values in a specific column
# with zeros. Use NumPy operations within the Pandas DataFrame.
def replace_negative_with_zero(df, column_name):
  df[column_name] = np.where(df[column_name] < 0, 0, df[column_name])
  return df
data = {'col1': [1, -2, 3], 'col2': [4, 5, 6]}
df = pd.DataFrame(data)
column_to_modify = 'col1'

df = replace_negative_with_zero(df.copy(), column_to_modify)
print(df)
```

```
   col1  col2
0     1     4
1     0     5
2     3     6
```

[63]:
```
# 26. Access the 3rd element from the given NumPy array
arr = np.array([10, 20, 30, 40, 50])
print(arr[2])
```

```
30
```

[64]:
```
# 27. Retrieve the element at index (1, 2) from the 2D NumPy array.
arr= np.array([[1,2,3],[4,5,6],[7,8,9]])
print(arr[1,2])
```

```
6
```

[65]:
```
# 28. Using boolean indexing, extract elements greater than 5 from the given␣
↪NumPy array.
arr = np.array([3, 8, 2, 10, 5, 7])
G=arr[arr>5]
G
```

[65]:
```
array([ 8, 10,  7])
```

[66]:
```
# Perform basic slicing to extract elements from index 2 to 5 (inclusive) from␣
↪the given NumPy array.
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
arr[2:5]
```

[66]:
```
array([3, 4, 5])
```

[67]:
```
#Slice the 2D NumPy array to extract the sub-array `[[2, 3], [5, 6]]` from the␣
↪given array.
```

```
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
sub_array = arr[2:3, 5:6]  # Rows 1 (inclusive) to 3 (exclusive), Columns 0␣
 ↪(inclusive) to 2 (exclusive)

# Print the sub-array
print(sub_array)
```

```
[]
```

[68]:
```
'''31.Write a NumPy function to extract elements in specific order from a given␣
 ↪2D array based on indices
provided in another array.'''
def extract_by_indices(arr, indices):
  flat_arr = arr.flatten() # return collapsed one d array
  print(flat_arr)
  extracted_elements = flat_arr[indices]
  return extracted_elements
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
indices = np.array([1, 3, 5])

extracted_elements = extract_by_indices(arr, indices)
print(extracted_elements)
```

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

[69]:
```
# 32. Create a NumPy function that filters elements greater than a threshold␣
 ↪from a given 1D array using
# boolean indexing.
def Filter(arr,threshold):
  f=arr[arr>threshold]
  print(f)


arr = np.array([1, 5, 3, 8, 2])
threshold = 4
Filter(arr, threshold)
```

```
[5 8]
```

[70]:
```
# 33. Develop a NumPy function that extracts specific elements from a 3D array␣
 ↪using indices provided in three
# separate arrays for each dimension.
def Tharr(arr,r,c,d):
  return arr[r,c,d]

arr=np.arange(24).reshape(2,3,4)
```

```
r=np.array([0,1])
c=np.array([1,2])
d=np.array([0,2])
Tharr(arr,r,c,d)
```

[70]:
```
array([ 4, 22])
```

[71]:
```
# 34. Write a NumPy function that returns elements from an array where both two␣
 ↪conditions are satisfied
# using boolean indexing.
def elements_satisfying_conditions(arr, condition1, condition2):
    # Ensure conditions are boolean arrays of the same shape as arr
    assert arr.shape == condition1.shape == condition2.shape, "Conditions must␣
 ↪have the same shape as the input array"

    # Boolean indexing to select elements from arr
    selected_elements = arr[(condition1) & (condition2)]

    return selected_elements

# Example usage:
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
condition1 = arr > 3
condition2 = arr % 2 == 0

selected_elements = elements_satisfying_conditions(arr, condition1, condition2)
print("Selected elements:", selected_elements)
```

```
Selected elements: [ 4  6  8 10]
```

[72]:
```
# 35. Create a NumPy function that extracts elements from a 2D array using row␣
 ↪and column indices provided
# in separate arrays.
def extract_by_indices(arr, row_indices, col_indices):

  extracted_elements = arr[row_indices, col_indices]

  return extracted_elements
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
row_indices = np.array([0, 2])
col_indices = np.array([1, 2])

extracted_elements = extract_by_indices(arr, row_indices, col_indices)
print(extracted_elements)
```

```
[2 9]
```

[73]:
```
# 36. Given an array arr of shape (3, 3), add a scalar value of 5 to each␣
 ↪element using NumPy broadcasting.
arr=np.arange(9).reshape((3,3))
arr+5
```

[73]:
```
array([[ 5,  6,  7],
       [ 8,  9, 10],
       [11, 12, 13]])
```

[74]:
```
# 37. Consider two arrays arr1 of shape (1, 3) and arr2 of shape (3, 4).␣
 ↪Multiply each row of arr2 by the
# corresponding element in arr1 using NumPy broadcasting.
arr1 = np.array([1, 2, 3])
arr2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
result = arr1[:, None]*arr2
result
```

[74]:
```
array([[ 1,  2,  3,  4],
       [10, 12, 14, 16],
       [27, 30, 33, 36]])
```

[75]:
```
# 38. Given a 1D array arr1 of shape (1, 4) and a 2D array arr2 of shape (4,␣
 ↪3), add arr1 to each row of arr2 using
#NumPy broadcasting.
arr1 = np.array([1, 2, 3, 4])
arr2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
arr1+arr2
```

[75]:
```
array([[ 2,  4,  6,  8],
       [ 6,  8, 10, 12],
       [10, 12, 14, 16]])
```

[76]:
```
# 39. Consider two arrays arr1 of shape (3, 1) and arr2 of shape (1, 3). Add␣
 ↪these arrays using NumPy
# broadcasting.
arr1 = np.array([[1], [2], [3]])
arr2= np.array([1, 2, 3])
arr1+arr2
```

[76]:
```
array([[2, 3, 4],
       [3, 4, 5],
       [4, 5, 6]])
```

[77]:
```
# 40.Given arrays arr1 of shape (2, 3) and arr2 of shape (2, 2), perform␣
 ↪multiplication using NumPy
arr1 = np.array([[1, 2, 3], [4, 5, 6]])  # shape (2, 3)
arr2 = np.array([[2, 3], [1, 2]])      # shape (2, 2)
```

```
# Reshape arr2 to (2, 1, 2) to align shapes for broadcasting
arr2_reshaped = arr2[:, :, np.newaxis]  # shape (2, 2, 1)

# Perform element-wise multiplication using broadcasting
result = arr1 * arr2_reshaped

print("arr1:")
print(arr1)
print("arr2:")
print(arr2)
print("Result of multiplication:")
print(result)
arr = np.array([[1, 2, 3], [4, 5, 6]])
```

```
arr1:
[[1 2 3]
 [4 5 6]]
arr2:
[[2 3]
 [1 2]]
Result of multiplication:
[[[ 2  4  6]
  [12 15 18]]

 [[ 1  2  3]
  [ 8 10 12]]]
```

[78]:
```
# 41. Calculate column wise mean for the given array:
arr = np.array([[1, 2, 3], [4, 5, 6]])
np.mean(arr, axis=0)
```

[78]:
```
array([2.5, 3.5, 4.5])
```

[79]:
```
# 42. Find maximum value in each row of the given array:
arr = np.array([[1, 2, 3], [4, 5, 6]])
np.max(arr)
```

[79]:
```
6
```

[80]:
```
# 43. For the given array, find indices of maximum value in each column.
arr = np.array([[1, 2, 3], [4, 5, 6]])
np.argmax(arr, axis=0)
```

[80]:
```
array([1, 1, 1])
```

[81]:
```
# 45. In the given array, check if all elements in each column are even.
arr = np.array([[2, 4, 6], [3, 5, 7]])
are_all_even = np.all(arr % 2 == 0, axis=0)

print("Check if all elements in each column are even:")
print(are_all_even)
```

```
Check if all elements in each column are even:
[False False False]
```

[82]:
```
# 46. Given a NumPy array arr, reshape it into a matrix of dimensions `m` rows and `n` columns. Return the
# reshaped matrix.
original_array = np.array([1, 2, 3, 4, 5, 6]).reshape((2,3))
print(original_array)
```

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

[83]:
```
# 47. Create a function that takes a matrix as input and returns the flattened array.
def Flattened(arr):
  return arr.flatten()

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

[83]:
```
array([2, 4, 6, 3, 5, 7])
```

[84]:
```
# 48. Write a function that concatenates two given arrays along a specified axis
import numpy as np

def concatenate_arrays(array1, array2, axis=0):
    """
    Concatenates two given arrays along a specified axis.

    Parameters:
    - array1: NumPy array
    - array2: NumPy array
    - axis: int, optional (default=0)
        Axis along which arrays are concatenated.
        If axis=0, arrays are stacked vertically (row-wise).
        If axis=1, arrays are stacked horizontally (column-wise).

    Returns:
    - concatenated_array: NumPy array
        Concatenated array.
```

```
    """
    concatenated_array = np.concatenate((array1, array2), axis=axis)
    return concatenated_array

# Example usage:
array1 = np.array([[1, 2], [3, 4]])
array2 = np.array([[5, 6], [7, 8]])

# Concatenate arrays along axis 0 (vertically)
result_axis0 = concatenate_arrays(array1, array2, axis=0)
print("Concatenated along axis 0:")
print(result_axis0)

# Concatenate arrays along axis 1 (horizontally)
result_axis1 = concatenate_arrays(array1, array2, axis=1)
print("\nConcatenated along axis 1:")
print(result_axis1)
```

```
Concatenated along axis 0:
[[1 2]
 [3 4]
 [5 6]
 [7 8]]

Concatenated along axis 1:
[[1 2 5 6]
 [3 4 7 8]]
```

[85]:
```
#49. Create a function that splits an array into multiple sub-arrays along a␣
 ↪specified axis.

def split_array(original_array, axis=0, num_splits=2):
    sub_arrays = np.split(original_array, num_splits, axis=axis)
    return sub_arrays

# Example usage:
original_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Split along axis 0 into 3 sub-arrays (row-wise)
sub_arrays_axis0 = split_array(original_array, axis=0, num_splits=3)
print("Split along axis 0:")
for arr in sub_arrays_axis0:
    print(arr)

# Split along axis 1 into 3 sub-arrays (column-wise)
sub_arrays_axis1 = split_array(original_array, axis=1, num_splits=3)
print("\nSplit along axis 1:")
```

```
for arr in sub_arrays_axis1:
    print(arr)
```

```
Split along axis 0:
[[1 2 3]]
[[4 5 6]]
[[7 8 9]]

Split along axis 1:
[[1]
 [4]
 [7]]
[[2]
 [5]
 [8]]
[[3]
 [6]
 [9]]
```

[86]:
```
#50. Write a function that inserts and then deletes elements from a given array␣
 ↪at specified indices.
def insert_and_delete_elements(original_array, indices_to_insert,␣
 ↪values_to_insert, indices_to_delete):
    for idx, val in zip(indices_to_insert, values_to_insert):
        original_array = np.insert(original_array, idx, val)
    modified_array = np.delete(original_array, indices_to_delete)
    return modified_array

# Example usage:
original_array = np.array([1, 2, 3, 4, 5])
indices_to_insert = [2, 4]
values_to_insert = [10, 11]
indices_to_delete = [1, 3]

modified_array = insert_and_delete_elements(original_array, indices_to_insert,␣
 ↪values_to_insert, indices_to_delete)
print(modified_array)
```

```
[ 1 10 11  4  5]
```

[87]:
```
# 51. Create a NumPy array `arr1` with random integers and another array `arr2`␣
 ↪with integers from 1 to 10.
# Perform element-wise addition between `arr1` and `arr2`.
arr1=np.random.randint(10)
arr2=np.arange(1,11)
arr1+arr2
```

[87]:
```
array([ 4,  5,  6,  7,  8,  9, 10, 11, 12, 13])
```

[88]:
```
# 52. Generate a NumPy array `arr1` with sequential integers from 10 to 1 and another array `arr2` with integers
# from 1 to 10. Subtract `arr2` from `arr1` element-wise.
arr1=np.arange(10,0,-1)
arr2=np.arange(1,11)
arr1-arr2
```

[88]:
```
array([ 9,  7,  5,  3,  1, -1, -3, -5, -7, -9])
```

[89]:
```
# 53. Create a NumPy array `arr1` with random integers and another array `arr2` with integers from 1 to 5.
# Perform element-wise multiplication between `arr1` and `arr2`.
arr1 = np.random.randint(1, 10)
arr2=np.arange(1,6)
arr1*arr2
```

[89]:
```
array([ 8, 16, 24, 32, 40])
```

[90]:
```
# 54. Generate a NumPy array `arr1` with even integers from 2 to 10 and another array `arr2` with integers from 1
# to 5. Perform element-wise division of `arr1` by `arr2`.
arr1 = np.arange(2, 11, 2)


arr2 = np.arange(1, 6)

arr1 / arr2
```

[90]:
```
array([2., 2., 2., 2., 2.])
```

[91]:
```
# 55. Create a NumPy array `arr1` with integers from 1 to 5 and another array `arr2` with the same numbers
# reversed. Calculate the exponentiation of `arr1` raised to the power of `arr2` element-wise.
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = arr1[::-1]
np.power(arr1, arr2)
```

[91]:
```
array([ 1, 16, 27, 16,  5])
```

[92]:
```
# 56. Write a function that counts the occurrences of a specific substring within a NumPy array of strings.
arr = np.array(['hello', 'world', 'hello', 'numpy', 'hello'])
substring = 'hello'
sum([string.count(substring) for string in arr])
```

[92]: 3

[93]:
```
#57. Write a function that extracts uppercase characters from a NumPy array of␣
 ↪strings.

arr = np.array(['Hello', 'World', 'OpenAI', 'GPT'])
[char for string in arr for char in string if char.isupper()]
```

[93]: ['H', 'W', 'O', 'A', 'I', 'G', 'P', 'T']

[94]:
```
# 58. Write a function that replaces occurrences of a substring in a NumPy␣
 ↪array of strings with a new string.
arr = np.array(['apple', 'banana', 'grape', 'pineapple'])

def replace_substring(arr, old_substring, new_substring):
    modified_arr = np.copy(arr)  # Create a copy to avoid modifying the␣
 ↪original array

    # Iterate over each element in the array
    for i, string in enumerate(modified_arr):
        # Replace occurrences of old_substring with new_substring
        modified_arr[i] = string.replace(old_substring, new_substring)

    return modified_arr

# Example usage:
arr = np.array(['apple', 'banana', 'grape', 'pineapple'])
old_substring = 'apple'
new_substring = 'orange'

modified_arr = replace_substring(arr, old_substring, new_substring)
print("Original array:", arr)
print("Modified array:", modified_arr)
```

```
Original array: ['apple' 'banana' 'grape' 'pineapple']
Modified array: ['orange' 'banana' 'grape' 'pineorang']
```

[95]:
```
# 59. Write a function that concatenates strings in a NumPy array element-wise.
def concatenate_strings(arr1, arr2):
  return np.char.add(arr1, arr2)
arr1 = np.array(['Hello', 'World'])
arr2 = np.array(['Open', 'AI'])
concatenate_strings(arr1, arr2)
```

[95]: array(['HelloOpen', 'WorldAI'], dtype='<U9')

[96]:
```
#60. Write a function that finds the length of the longest string in a NumPy␣
 ↪array.
def LongStr(arr):
  return max(len(string) for string in arr)

arr = np.array(['apple', 'banana', 'grape', 'pineapple'])
LongStr(arr)
```

[96]: 9

[97]:
```
# 61. Create a dataset of 100 random integers between 1 and 1000. Compute the␣
 ↪mean, median, variance, and
# standard deviation of the dataset using NumPy's functions.
arr=np.arange(1,1000)
print(np.mean(arr))
print(np.median(arr))
print(np.var(arr))
print(np.std(arr))
```

```
500.0
500.0
83166.66666666667
288.38631497813253
```

[98]:
```
# 62. Generate an array of 50 random numbers between 1 and 100. Find the 25th␣
 ↪and 75th percentiles of the
# dataset
arr=np.random.randint(1,100, size=50)
percentile_25th = np.percentile(arr, 25)
percentile_75th = np.percentile(arr, 75)
print(percentile_75th)
print(percentile_25th)
```

```
80.0
20.75
```

[99]:
```
# 63. Create two arrays representing two sets of variables. Compute the␣
 ↪correlation coefficient between these
# arrays using NumPy's `corrcoef` function
set1 = np.array([1, 2, 3, 4, 5])
set2 = np.array([5, 4, 3, 2, 1])
np.corrcoef(set1, set2)[0, 1]
```

[99]: -0.9999999999999999

[100]:
```
# 64. Create two matrices and perform matrix multiplication using NumPy's `dot`␣
 ↪function.
```

```
matrix1 = np.array([[1, 2, 3],
                    [4, 5, 6]])

matrix2 = np.array([[7, 8],
                    [9, 10],
                    [11, 12]])
np.dot(matrix1, matrix2)
```

[100]:
```
array([[ 58,  64],
       [139, 154]])
```

[101]:
```
# 65. Create an array of 50 integers between 10 and 1000. Calculate the 10th,␣
 ↪50th (median), and 90th
# percentiles along with the first and third quartiles
data = np.random.randint(10, 1001, size=50)
percentile_10th = np.percentile(data, 10)
percentile_50th = np.percentile(data, 50)  # Median
percentile_90th = np.percentile(data, 90)
first_quartile = np.percentile(data, 25)
third_quartile = np.percentile(data, 75)

# Print the results
print(f"Dataset: {data}")
print(f"10th Percentile: {percentile_10th:.2f}")
print(f"50th Percentile (Median): {percentile_50th}")
print(f"90th Percentile: {percentile_90th:.2f}")
print(f"First Quartile (25th Percentile): {first_quartile}")
print(f"Third Quartile (75th Percentile): {third_quartile}")
```

```
Dataset: [407 561  11 971 666 470 526 249 537 876 804  57 956 262 123 172 115
124
 960 776  48  45  77 806  82 304 254 181 933 806 890 649 547  81 944 781
 211 802  95 896 186 489 290 886 415 441 639 845 619 691]
10th Percentile: 80.60
50th Percentile (Median): 507.5
90th Percentile: 899.70
First Quartile (25th Percentile): 182.25
Third Quartile (75th Percentile): 803.5
```

[102]:
```
# 66. Create a NumPy array of integers and find the index of a specific element.
arr = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
element_to_find = 50
np.where(arr == element_to_find)[0]
```

[102]:
```
array([4])
```

[103]:
```
# 67. Generate a random NumPy array and sort it in ascending order.
arr = np.random.rand(10)
np.sort(arr)
```

[103]:
```
array([0.02831554, 0.11849111, 0.13931801, 0.33441932, 0.38883372,
       0.58447469, 0.61020078, 0.63004275, 0.81218638, 0.89833043])
```

[104]:
```
# Filter elements >20  in the given NumPy array
arr = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
arr[arr > 20]
```

[104]:
```
array([ 30,  40,  50,  60,  70,  80,  90, 100])
```

[105]:
```
# 69. Filter elements which are divisible by 3 from a given NumPy array.
arr = np.array([1, 5, 8, 12, 15])
arr[arr % 3 == 0]
```

[105]:
```
array([12, 15])
```

[106]:
```
# 70. Filter elements which are  20 and  40 from a given NumPy array.
arr = np.array([10, 20, 30, 40, 50])
arr[(arr >= 20) & (arr <= 40)]
```

[106]:
```
array([20, 30, 40])
```

[107]:
```
# 71. For the given NumPy array, check its byte order using the `dtype`␣
 ↪attribute byteorder.
arr = np.array([1, 2, 3])
arr.dtype.byteorder
```

[107]:
```
'='
```

[108]:
```
# 72. For the given NumPy array, perform byte swapping in place using␣
 ↪`byteswap()`.
arr = np.array([1, 2, 3], dtype=np.int32)
arr.byteswap()
```

[108]:
```
array([16777216, 33554432, 50331648], dtype=int32)
```

[109]:
```
# 73. For the given NumPy array, swap its byte order without modifying the␣
 ↪original array using
# `newbyteorder()`.
arr = np.array([1, 2, 3], dtype=np.int32)
arr.newbyteorder()
```

[109]:
```
array([16777216, 33554432, 50331648], dtype='>i4')
```

[110]:
```
# 74. For the given NumPy array and swap its byte order conditionally based on␣
 ↪system endianness using
# `newbyteorder()`.
arr = np.array([1, 2, 3], dtype=np.int32)
if arr.dtype.byteorder not in ('=', '|'):
    # Swap byte order if not native
    arr = arr.newbyteorder()

print("Original array:")
print(arr)
```

```
Original array:
[1 2 3]
```

[111]:
```
# 75. For the given NumPy array, check if byte swapping is necessary for the␣
 ↪current system using `dtype`
# attribute `byteorder`.
arr = np.array([1, 2, 3], dtype=np.int32)
if arr.dtype.byteorder == '<':
    print("Byte swapping is necessary.")
    arr = arr.byteswap()
    print("Swapped array:")
    print(arr)
else:
    print("Byte swapping is not necessary.")
```

```
Byte swapping is not necessary.
```

[112]:
```
# # 76. Create a NumPy array `arr1` with values from 1 to 10. Create a copy of␣
 ↪`arr1` named `copy_arr` and modify
# an element in `copy_arr`. Check if modifying `copy_arr` affects `arr1`.
arr1 = np.arange(1, 11)
copy_arr = arr1.copy()
copy_arr[0] = 100
print(arr1)
print(copy_arr)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
[100   2   3   4   5   6   7   8   9  10]
```

[113]:
```
# 77. Create a 2D NumPy array `matrix` of shape (3, 3) with random integers.␣
 ↪Extract a slice `view_slice` from
# the matrix. Modify an element in `view_slice` and observe if it changes the␣
 ↪original `matrix`.
matrix = np.random.randint(1, 10, size=(3, 3))
view_slice = matrix[:2, :2]
view_slice[0, 0] = 100
```

```
print(matrix)
print(view_slice)
```

```
[[100   1   6]
 [  9   6   4]
 [  9   6   7]]
[[100   1]
 [  9   6]]
```

[114]:
```
# 78. Create a NumPy array `array_a` of shape (4, 3) with sequential integers␣
 ↪from 1 to 12. Extract a slice
# `view_b` from `array_a` and broadcast the addition of 5 to view_b. Check if␣
 ↪it alters the original `array_a`.
array_a = np.arange(1, 13).reshape(4, 3)
view_b = array_a[:2, :2]
view_b += 5
print(array_a)
print(view_b)
```

```
[[ 6  7  3]
 [ 9 10  6]
 [ 7  8  9]
 [10 11 12]]
[[ 6  7]
 [ 9 10]]
```

[115]:
```
# 79. Create a NumPy array `orig_array` of shape (2, 4) with values from 1 to 8.
 ↪ Create a reshaped view
# `reshaped_view` of shape (4, 2) from orig_array. Modify an element in␣
 ↪`reshaped_view` and check if it
# reflects changes in the original `orig_array`.
orig_array = np.arange(1, 9).reshape(2, 4)
reshaped_view = orig_array.reshape(4, 2)
reshaped_view[0, 0] = 100
print(orig_array)
```

```
[[100   2   3   4]
 [  5   6   7   8]]
```

[116]:
```
# 80. Create a NumPy array `data` of shape (3, 4) with random integers. Extract␣
 ↪a copy `data_copy` of
# elements greater than 5. Modify an element in `data_copy` and verify if it␣
 ↪affects the original `data`.
data = np.random.randint(0, 10, size=(3, 4))
print("Original data:")
print(data)
data_copy = data[data > 5].copy()
```

```
print("\nCopied data with elements greater than 5:")
print(data_copy)
if data_copy.size > 0:
    data_copy[0] = 10
    print("\nModified data_copy:")
print(data_copy)
print("\nOriginal data after modification check:")
print(data)
```

```
Original data:
[[2 6 1 5]
 [5 7 5 7]
 [0 3 1 5]]

Copied data with elements greater than 5:
[6 7 7]

Modified data_copy:
[10  7  7]

Original data after modification check:
[[2 6 1 5]
 [5 7 5 7]
 [0 3 1 5]]
```

[117]:
```
# 81. Create two matrices A and B of identical shape containing integers and␣
 ↪perform addition and subtraction
# operations between them.
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
addition = A + B
subtraction = A - B
print(addition)
print(subtraction)
```

```
[[ 6  8]
 [10 12]]
[[-4 -4]
 [-4 -4]]
```

[118]:
```
# 82. Generate two matrices `C` (3x2) and `D` (2x4) and perform matrix␣
 ↪multiplication.
C = np.array([[1, 2],
              [3, 4],
              [5, 6]])
D = np.array([[7, 8, 9, 10],
              [11, 12, 13, 14]])
```

```
np.dot(C, D)
```

[118]:
```
array([[ 29,  32,  35,  38],
       [ 65,  72,  79,  86],
       [101, 112, 123, 134]])
```

[119]:
```
E=np.array([[1, 2],
            [3, 4],
            [5, 6]])
E.T
```

[119]:
```
array([[1, 3, 5],
       [2, 4, 6]])
```

[120]:
```
#  84.Generate a square matrix `F` and compute its determinant.a
F = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
np.linalg.det(F)
```

[120]:
```
0.0
```

[121]:
```
# 85. Create a square matrix `G` and find its inverse.
G = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 10]])
np.linalg.inv(G)
```

[121]:
```
array([[-0.66666667, -1.33333333,  1.        ],
       [-0.66666667,  3.66666667, -2.        ],
       [ 1.        , -2.        ,  1.        ]])
```

[121]:
```

```