arrays-practise

November 24, 2023

[3]:
```
import numpy as np

twod_array= np.array([[1,2,3,4],[4,5,7]])
```

[5]:
```
print(twod_array)
```

```
[list([1, 2, 3, 4]) list([4, 5, 7])]
```

[27]:
```
def pair_numbers(arr,num):
    start = time.time()
    pair_nums = []
    for i in range(len(arr)):
        for j in range(len(arr)) :
            if j !=i:
                if arr[i]+arr[j]==num:
                    pair_nums.append((arr[i],arr[j]))
    print(time.time()-start)
    return pair_nums
```

[63]:
```
from timeit import default_timer as timer
from datetime import timedelta
import time
num=35
arr =[1,2,3,5,6,30]
```

[46]:
```
start=timer()
pair_numbers(arr,num)
end=timer()
print(timedelta(end,start))
```

```
0.0
189 days, 11:16:51.346685
```

[66]:
```
#O(n)
def sum_numbers(arr,num):
    pair_nums = []
    seen={}
    for i in range(len(arr)):
```

```python
        comp = num-arr[i]

        if comp in seen:
            pair_nums.append((comp,arr[i]))
        seen[arr[i]]=i
    return pair_nums
```

[67]:
```python
sum_numbers(arr,num)
```

[67]: [(5, 30)]

[ ]:
```python
def max_product(arr):
    arr.sort(reverse=True)
    return arr[0]*arr[1]
```

[68]:
```python
def max_product(arr):
    # Initialize two variables to store the two largest numbers
    max1, max2 = 0, 0  # O(1), constant time initialization

    # Iterate through the array
    for num in arr:  # O(n), where n is the length of the array
        # If the current number is greater than max1, update max1 and max2
        if num > max1:  # O(1), constant time comparison
            max2 = max1  # O(1), constant time assignment
            max1 = num  # O(1), constant time assignment
        # If the current number is greater than max2 but not max1, update max2
        elif num > max2:  # O(1), constant time comparison
            max2 = num  # O(1), constant time assignment

    # Return the product of the two largest numbers
    return max1 * max2  # O(1), constant time multiplication

arr = [1, 7, 3, 4, 9, 5]
print(max_product(arr))  # Output: 63 (9*7)
```

63

[69]:
```python
def middle(lst):
    # Return a new list containing all elements from the original list,␣
↪excluding the first and last elements
    return lst[1:-1]

my_list = [1, 2, 3, 4]

print(middle(my_list))  # Output: [2, 3]
```

[2, 3]

[70]:
```python
def diagonal_sum(matrix):
    # Initialize the sum to 0
    total = 0

    # Iterate through the rows of the matrix
    for i in range(len(matrix)):
        # Add the diagonal element to the total sum
        total += matrix[i][i]

    return total
```

[ ]:
```python
def first_second(my_list):
    max1, max2 = float('-inf'), float('-inf')

    for num in my_list:
        if num > max1:
            max2 = max1
            max1 = num
        elif num > max2 and num != max1:
            max2 = num

    return max1, max2

my_list = [84, 85, 86, 87, 85, 90, 85, 83, 23, 45, 84, 1, 2, 0]
print(first_second(my_list))  # Output: (90, 87)
```

[ ]:
```python
def remove_duplicates(lst):
    unique_lst = []
    seen = set()
    for item in lst:
        if item not in seen:
            unique_lst.append(item)
            seen.add(item)
    return unique_lst

my_list = [1, 1, 2, 2, 3, 4, 5]
print(remove_duplicates(my_list))  # Output: [1, 2, 3, 4, 5]
```

[71]:
```python
def remove_duplicates(lst):
    return list(set(lst))
```

[76]:
```python
def pair_sum(myList, sum):
    # TODO
    outList=[]
    seen=[]
    for i in range(len(myList)):
        com = sum - myList[i]
```

```
        if com in seen:
            outList.append('+'.join([str(com),str(myList[i])]))
        seen.append(myList[i])
    return outList
```

[75]:
```
pair_sum(arr, num)
```

[75]: []

[77]:
```
def pair_sum(arr, target_sum):
    result = []
    for i in range(len(arr)):
        for j in range(i+1, len(arr)):
            if arr[i] + arr[j] == target_sum:
                result.append(f"{arr[i]}+{arr[j]}")
    return result
```

[78]:
```
def contains_duplicate(nums):
    # TODO
    if len(nums) > len(list(set(nums))) :
        return True
    else :
        return False
```

[80]:
```
def contains_duplicate(nums):
    seen = set()
    for num in nums:
        if num in seen:
            return True
        seen.add(num)
    return False
```

[82]:
```
# Example usage
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 1]
print(contains_duplicate(nums))  # Output: True
```

True

[83]:
```
def rotate(matrix):
    n = len(matrix)

    # Transpose the matrix
    for i in range(n):  # Iterate over the rows
        for j in range(i, n):  # Iterate over the columns starting from the␣
↪current row 'i'
            # Swap the elements at positions (i, j) and (j, i)
```

```
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

    # Reverse each row
    for row in matrix:  # Iterate over each row in the matrix
        row.reverse()  # Reverse the elements in the current row
'''Explanation:



n = len(matrix) - Get the number of rows/columns in the square matrix and store␣
 ↪it in the variable n.

Transpose the matrix:
a. for i in range(n): - Start a loop that iterates over the rows.
b. for j in range(i, n): - Start a nested loop that iterates over the columns␣
 ↪starting from the current row i. This ensures we only swap elements in the␣
 ↪upper triangle of the matrix, avoiding double swaps.
c. matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j] - Swap the elements␣
 ↪at positions (i, j) and (j, i).

Reverse each row:
a. for row in matrix: - Start a loop that iterates over each row in the matrix.
b. row.reverse() - Reverse the elements in the current row.

The time complexity of this code is O(n^2), as both the transpose and reverse␣
 ↪steps involve nested loops that iterate over all the elements in the matrix.␣
 ↪The space complexity is O(1), as the rotation is performed in-place without␣
 ↪allocating any additional data structures.

'''
```

[83]:

```
'Explanation:\n\n\n\nn = len(matrix) - Get the number of rows/columns in the
square matrix and store it in the variable n.\n\nTranspose the matrix: a. for i
in range(n): - Start a loop that iterates over the rows. b. for j in range(i,
n): - Start a nested loop that iterates over the columns starting from the
current row i. This ensures we only swap elements in the upper triangle of the
matrix, avoiding double swaps. c. matrix[i][j], matrix[j][i] = matrix[j][i],
matrix[i][j] - Swap the elements at positions (i, j) and (j, i).\n\nReverse each
row: a. for row in matrix: - Start a loop that iterates over each row in the
matrix. b. row.reverse() - Reverse the elements in the current row.\n\nThe time
complexity of this code is O(n^2), as both the transpose and reverse steps
involve nested loops that iterate over all the elements in the matrix. The space
complexity is O(1), as the rotation is performed in-place without allocating any
additional data structures.\n\n'
```

[ ]: