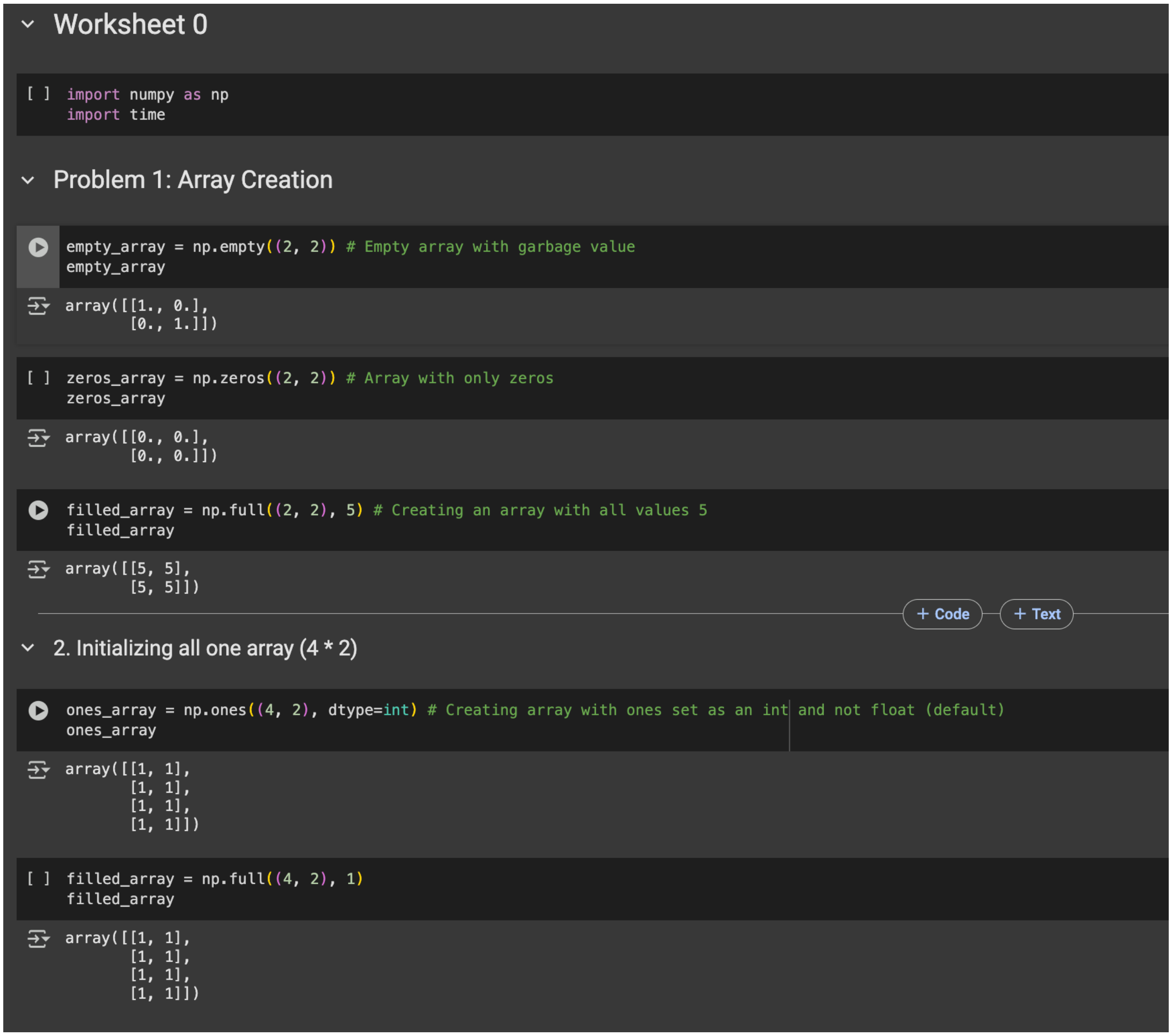
1. Array Creation



Worksheet 0

```
import numpy as np
import time
```

Problem 1: Array Creation

```
empty_array = np.empty((2, 2)) # Empty array with garbage value
empty_array
```

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

```
zeros_array = np.zeros((2, 2)) # Array with only zeros
zeros_array
```

```
array([[0., 0.],
       [0., 0.]])
```

```
filled_array = np.full((2, 2), 5) # Creating an array with all values 5
filled_array
```

```
array([[5, 5],
       [5, 5]])
```

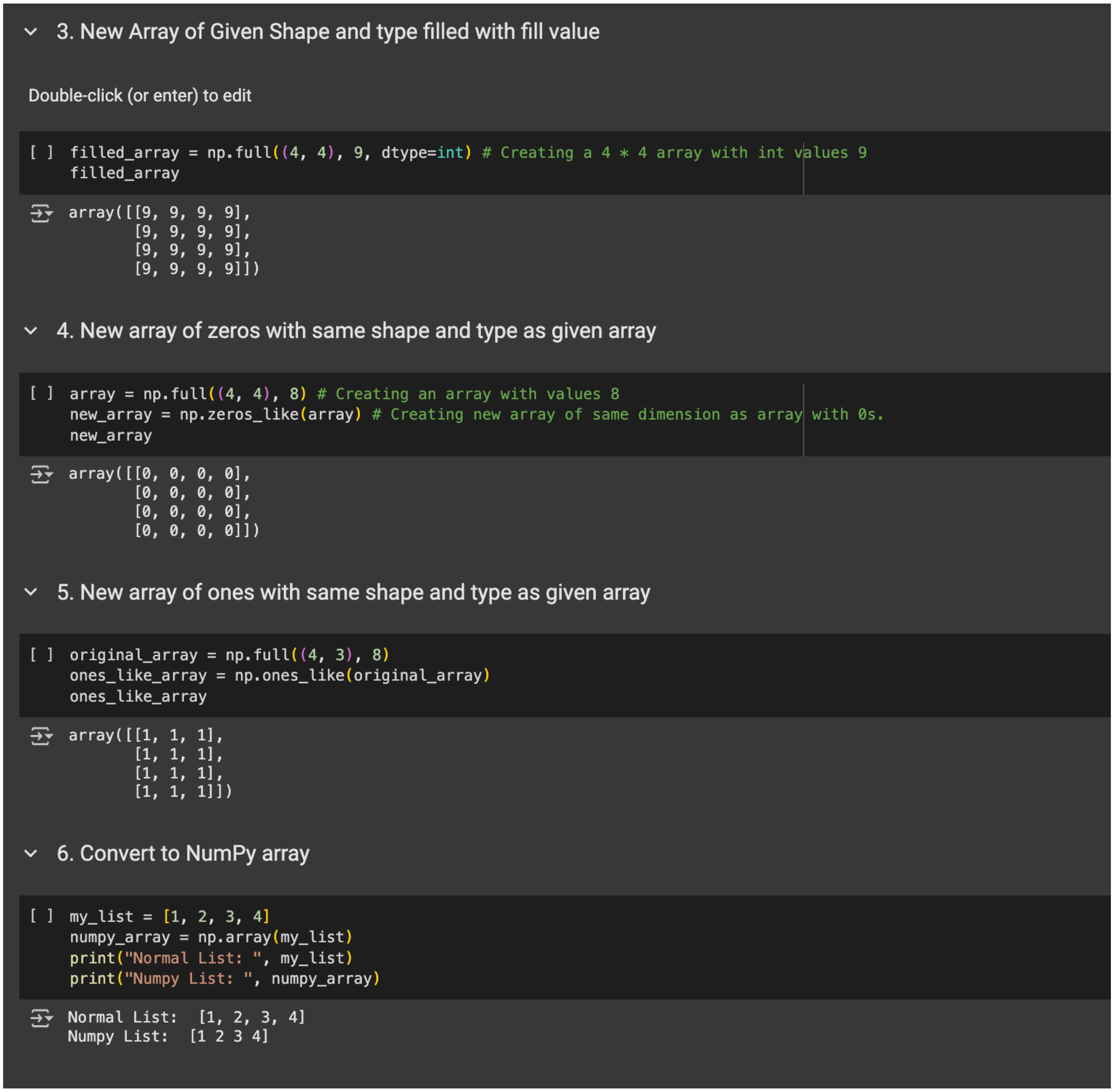
2. Initializing all one array (4 * 2)

```
ones_array = np.ones((4, 2), dtype=int) # Creating array with ones set as an int and not float (default)
ones_array
```

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

```
filled_array = np.full((4, 2), 1)
filled_array
```

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



3. New Array of Given Shape and type filled with fill value

Double-click (or enter) to edit

```
filled_array = np.full((4, 4), 9, dtype=int) # Creating a 4 * 4 array with int values 9
filled_array
```

```
array([[9, 9, 9, 9],
       [9, 9, 9, 9],
       [9, 9, 9, 9],
       [9, 9, 9, 9]])
```

4. New array of zeros with same shape and type as given array

```
array = np.full((4, 4), 8) # Creating an array with values 8
new_array = np.zeros_like(array) # Creating new array of same dimension as array with 0s.
new_array
```

```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

5. New array of ones with same shape and type as given array

```
original_array = np.full((4, 3), 8)
ones_like_array = np.ones_like(original_array)
ones_like_array
```

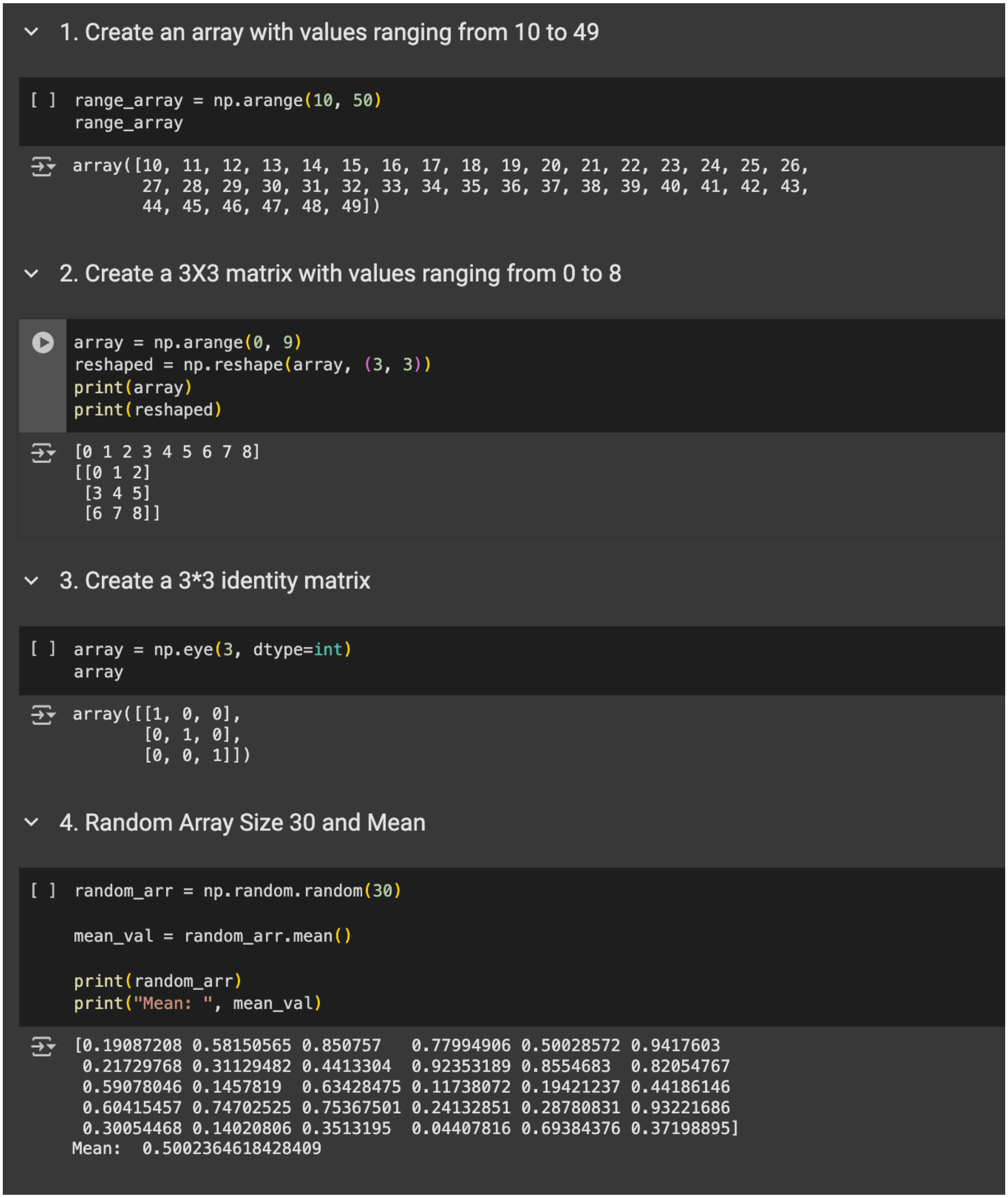
```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

6. Convert to NumPy array

```
my_list = [1, 2, 3, 4]
numpy_array = np.array(my_list)
print("Normal List: ", my_list)
print("Numpy List: ", numpy_array)
```

```
Normal List:  [1, 2, 3, 4]
Numpy List:  [1 2 3 4]
```

2. Array Manipulation: Numerical Ranges and Array Indexing



1. Create an array with values ranging from 10 to 49

```
range_array = np.arange(10, 50)
range_array
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
       27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
       44, 45, 46, 47, 48, 49])
```

2. Create a 3X3 matrix with values ranging from 0 to 8

```
array = np.arange(0, 9)
reshaped = np.reshape(array, (3, 3))
print(array)
print(reshaped)
```

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

3. Create a 3*3 identity matrix

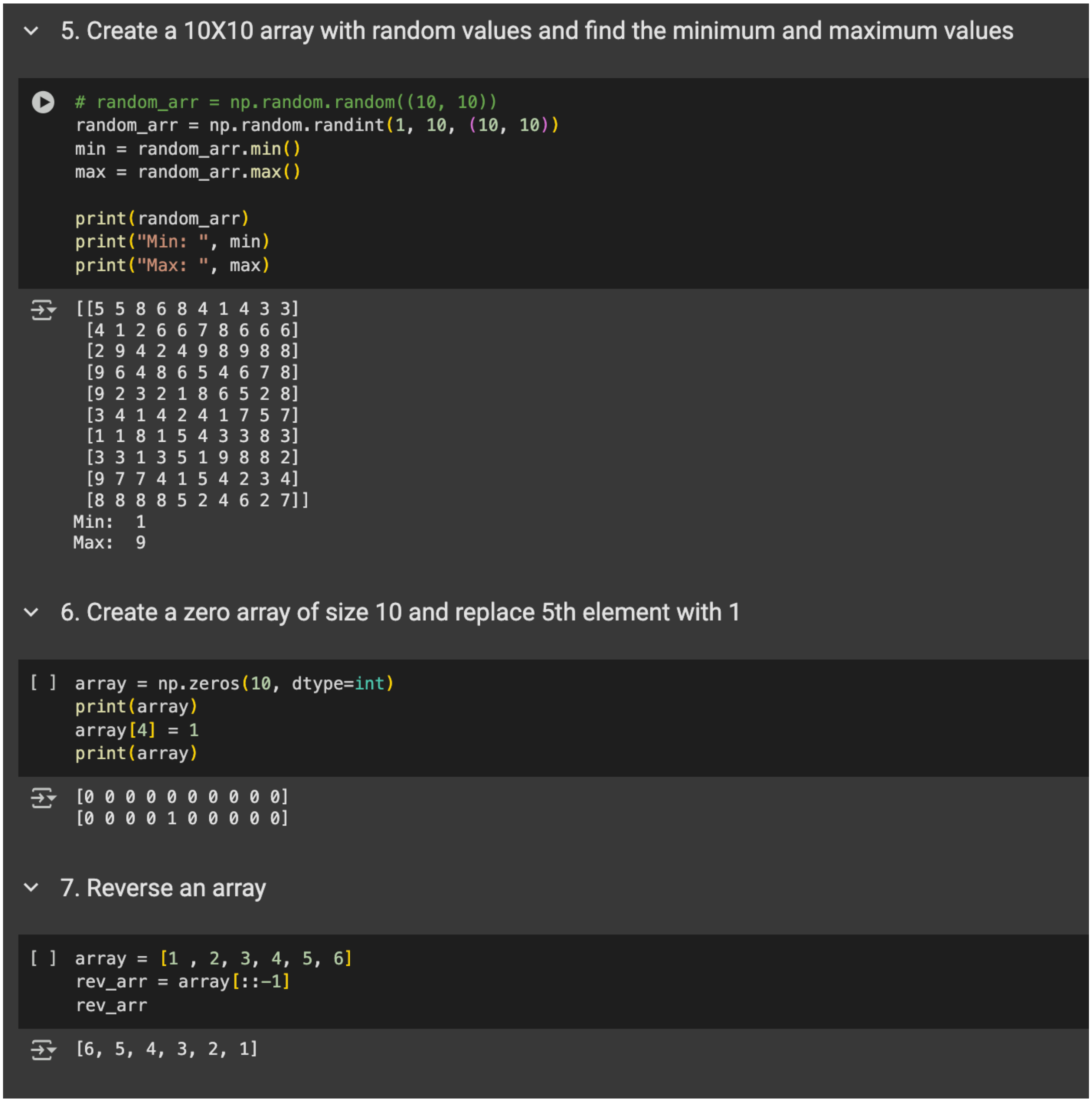
```
array = np.eye(3, dtype=int)
array
```

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

4. Random Array Size 30 and Mean

```
random_arr = np.random.random(30)

mean_val = random_arr.mean()

print(random_arr)
print("Mean: ", mean_val)
```

```
[0.19087208 0.58150565 0.850757   0.77994906 0.50028572 0.9417603
 0.21729768 0.31129482 0.4413304  0.92353189 0.8554683  0.82054767
 0.59078046 0.1457819  0.63428475 0.11738072 0.19421237 0.44186146
 0.60415457 0.74702525 0.75367501 0.24132851 0.28780831 0.93221686
 0.30054468 0.14020806 0.3513195  0.04407816 0.69384376 0.37198895]
Mean:  0.5002364618428409
```



5. Create a 10X10 array with random values and find the minimum and maximum values

```python
# random_arr = np.random.random((10, 10))
random_arr = np.random.randint(1, 10, (10, 10))
min = random_arr.min()
max = random_arr.max()

print(random_arr)
print("Min: ", min)
print("Max: ", max)
```

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

6. Create a zero array of size 10 and replace 5th element with 1

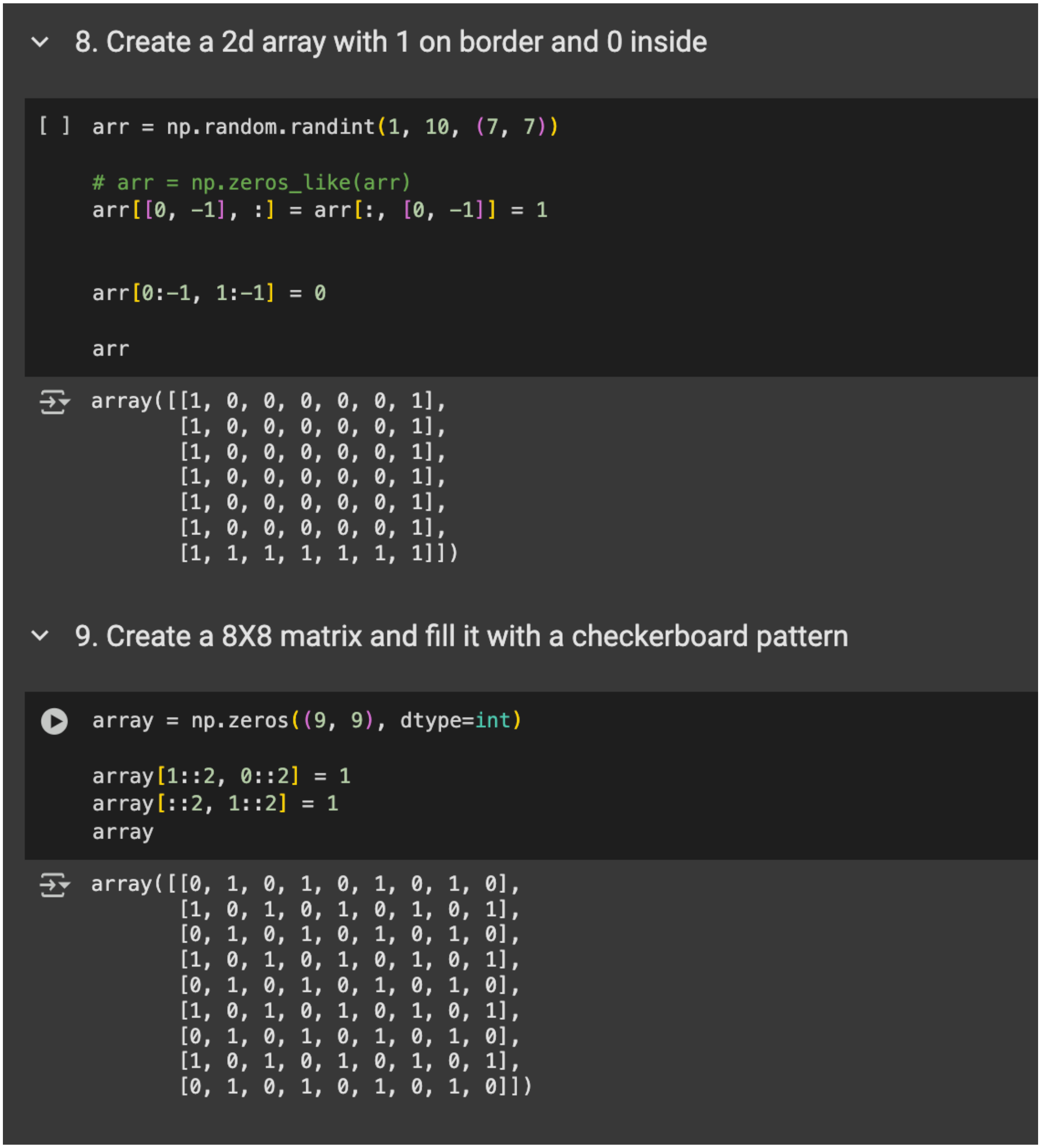
```python
array = np.zeros(10, dtype=int)
print(array)
array[4] = 1
print(array)
```

```
[0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
```

7. Reverse an array

```python
array = [1 , 2, 3, 4, 5, 6]
rev_arr = array[::-1]
rev_arr
```

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



8. Create a 2d array with 1 on border and 0 inside

```
arr = np.random.randint(1, 10, (7, 7))

# arr = np.zeros_like(arr)
arr[[0, -1], :] = arr[:, [0, -1]] = 1


arr[0:-1, 1:-1] = 0

arr
```

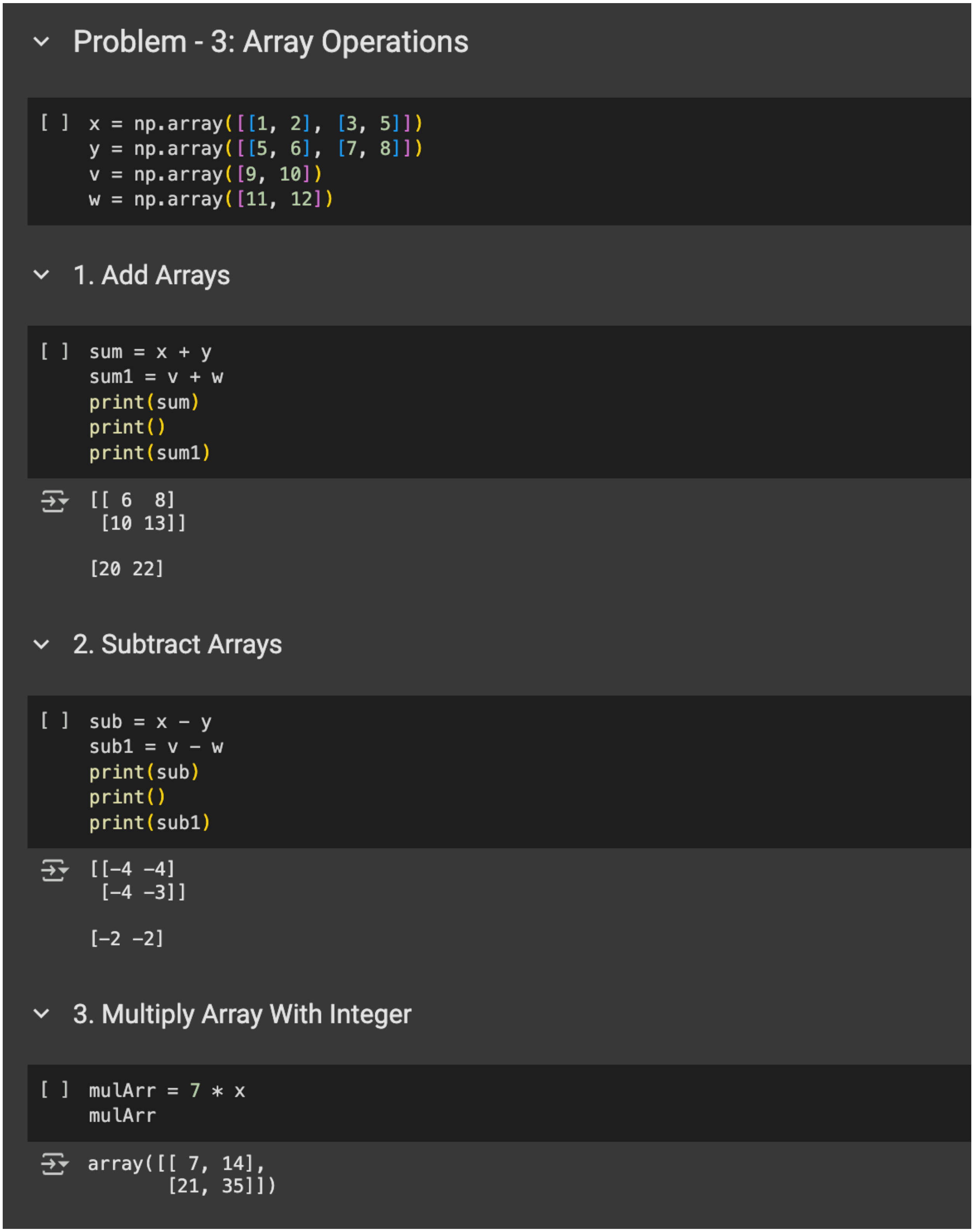
```
array([[1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 0, 0, 0, 1],
       [1, 1, 1, 1, 1, 1, 1]])
```

9. Create a 8X8 matrix and fill it with a checkerboard pattern

```
array = np.zeros((9, 9), dtype=int)

array[1::2, 0::2] = 1
array[::2, 1::2] = 1
array
```

```
array([[0, 1, 0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0, 1, 0],
       [1, 0, 1, 0, 1, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 1, 0, 1, 0]])
```

3. Array Operation



Problem - 3: Array Operations

```
x = np.array([[1, 2], [3, 5]])
y = np.array([[5, 6], [7, 8]])
v = np.array([9, 10])
w = np.array([11, 12])
```

1. Add Arrays

```
sum = x + y
sum1 = v + w
print(sum)
print()
print(sum1)
```

```
[[ 6  8]
 [10 13]]

[20 22]
```

2. Subtract Arrays

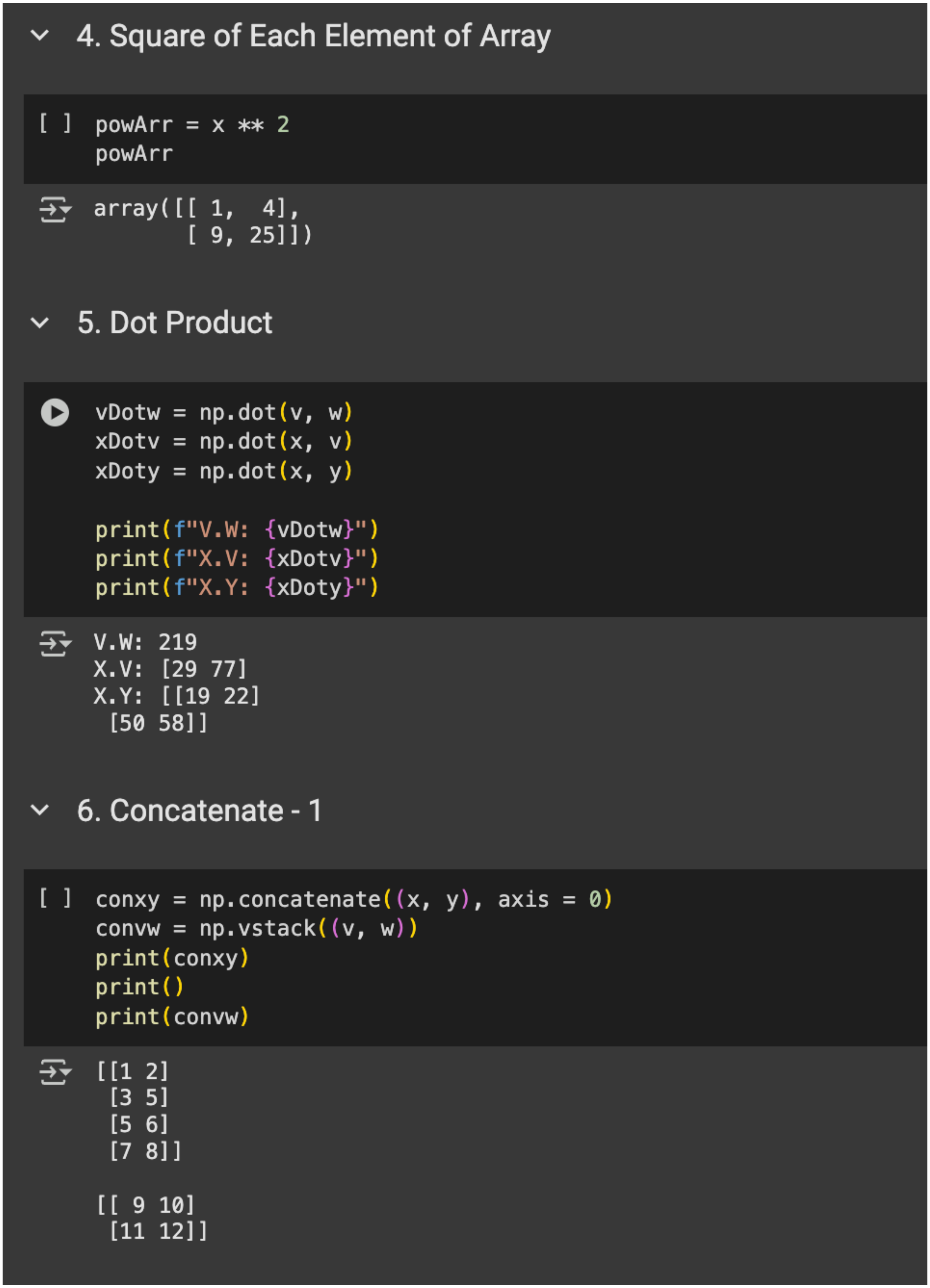
```
sub = x - y
sub1 = v - w
print(sub)
print()
print(sub1)
```

```
[[-4 -4]
 [-4 -3]]

[-2 -2]
```

3. Multiply Array With Integer

```
mulArr = 7 * x
mulArr
```

```
array([[ 7, 14],
       [21, 35]])
```



4. Square of Each Element of Array

```
powArr = x ** 2
powArr
```

```
array([[ 1,  4],
       [ 9, 25]])
```

5. Dot Product

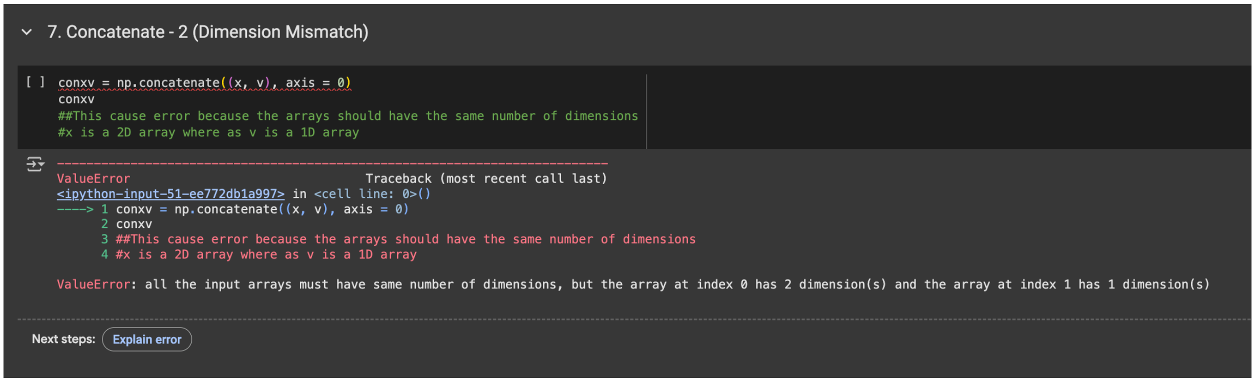
```
vDotw = np.dot(v, w)
xDotv = np.dot(x, v)
xDoty = np.dot(x, y)

print(f"V.W: {vDotw}")
print(f"X.V: {xDotv}")
print(f"X.Y: {xDoty}")
```

```
V.W: 219
X.V: [29 77]
X.Y: [[19 22]
 [50 58]]
```

6. Concatenate - 1

```
conxy = np.concatenate((x, y), axis = 0)
convw = np.vstack((v, w))
print(conxy)
print()
print(convw)
```

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

[[ 9 10]
 [11 12]]
```

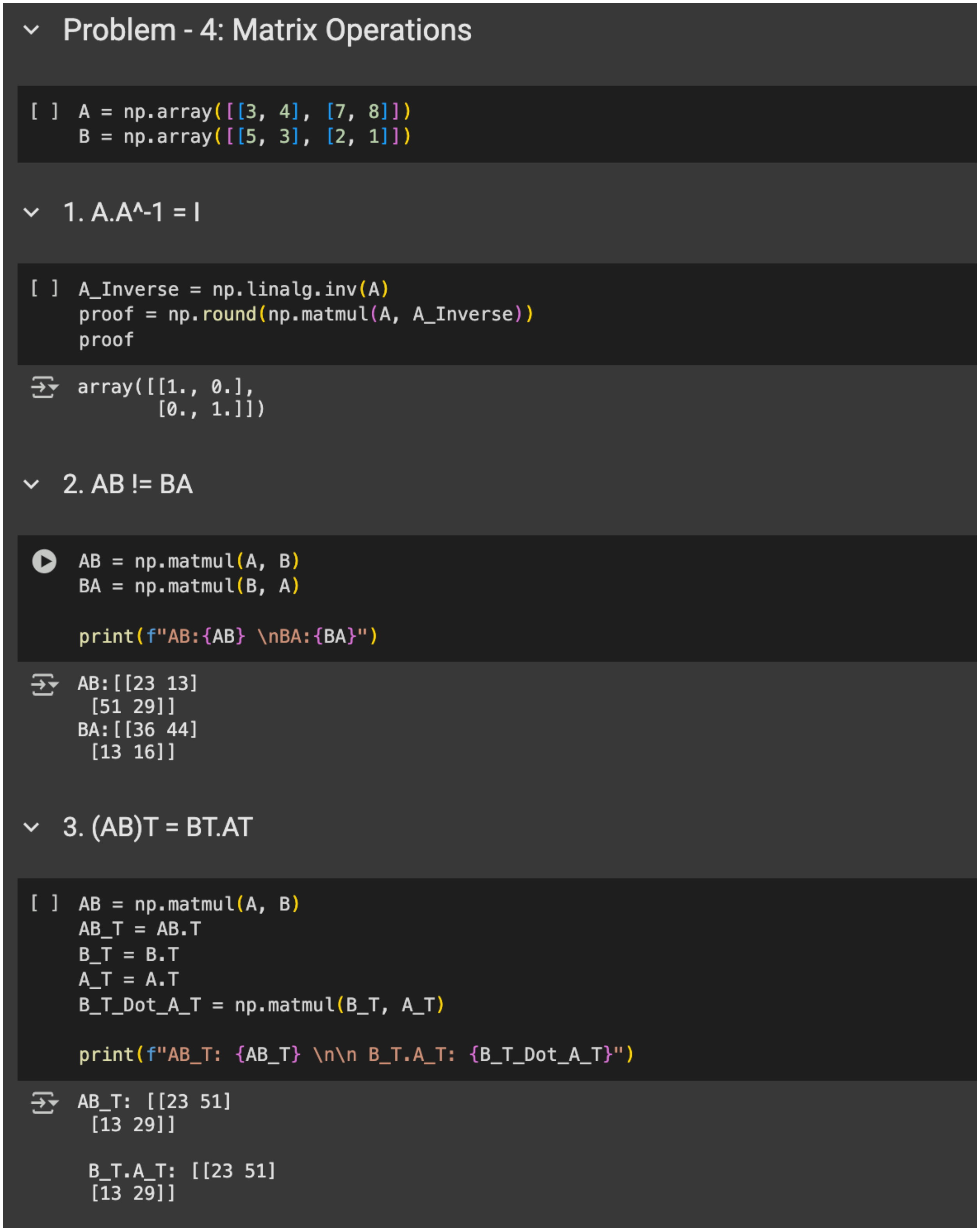


7. Concatenate - 2 (Dimension Mismatch)

```
conxv = np.concatenate((x, v), axis = 0)
conxv
##This cause error because the arrays should have the same number of dimensions
#x is a 2D array where as v is a 1D array
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-51-ee772db1a997> in <cell line: 0>()
----> 1 conxv = np.concatenate((x, v), axis = 0)
      2 conxv
      3 ##This cause error because the arrays should have the same number of dimensions
      4 #x is a 2D array where as v is a 1D array

ValueError: all the input arrays must have same number of dimensions, but the array at index 0 has 2 dimension(s) and the array at index 1 has 1 dimension(s)
```

4. Matrix Operation



Problem - 4: Matrix Operations

```
A = np.array([[3, 4], [7, 8]])
B = np.array([[5, 3], [2, 1]])
```

1. A.A^-1 = I

```
A_Inverse = np.linalg.inv(A)
proof = np.round(np.matmul(A, A_Inverse))
proof
```

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

2. AB != BA

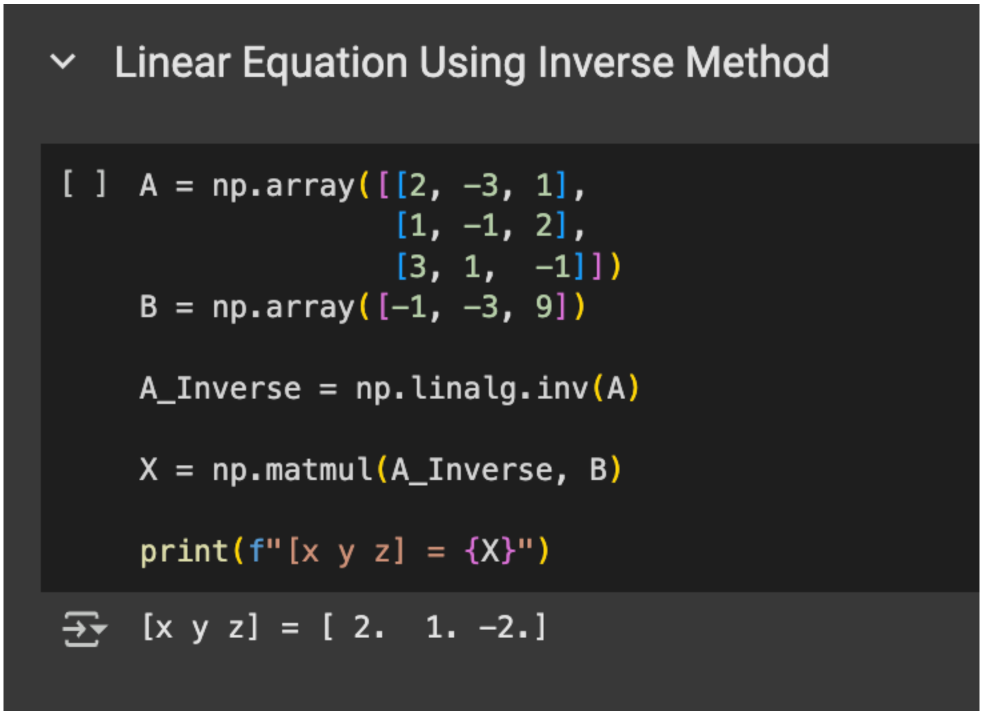
```
AB = np.matmul(A, B)
BA = np.matmul(B, A)

print(f"AB:{AB} \nBA:{BA}")
```

```
AB:[[23 13]
 [51 29]]
BA:[[36 44]
 [13 16]]
```

3. (AB)T = BT.AT

```
AB = np.matmul(A, B)
AB_T = AB.T
B_T = B.T
A_T = A.T
B_T_Dot_A_T = np.matmul(B_T, A_T)

print(f"AB_T: {AB_T} \n\n B_T.A_T: {B_T_Dot_A_T}")
```

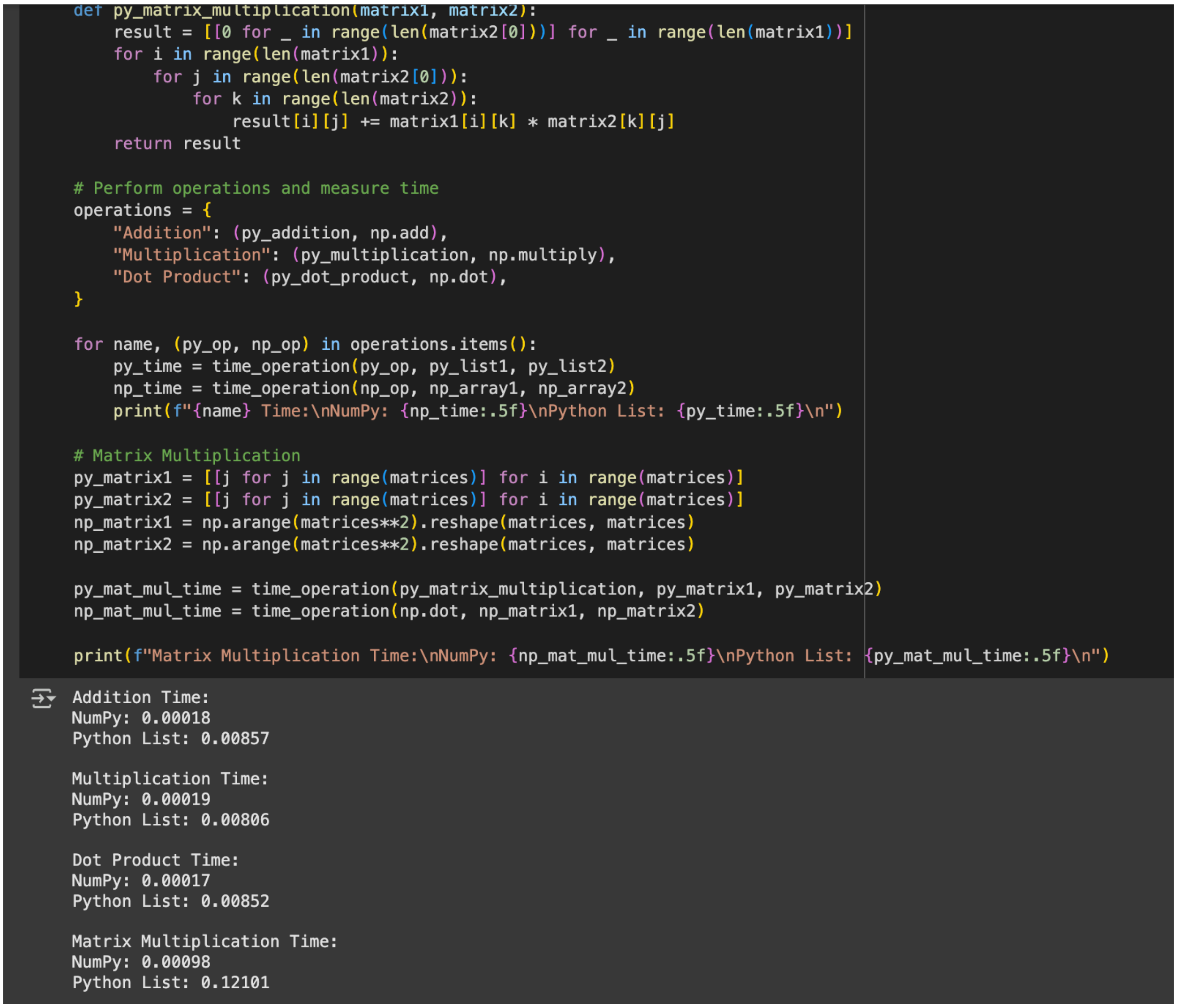
```
AB_T: [[23 51]
 [13 29]]

 B_T.A_T: [[23 51]
 [13 29]]
```



Linear Equation Using Inverse Method

```
A = np.array([[2, -3, 1],
              [1, -1, 2],
              [3, 1,  -1]])
B = np.array([-1, -3, 9])

A_Inverse = np.linalg.inv(A)

X = np.matmul(A_Inverse, B)

print(f"[x y z] = {X}")
```

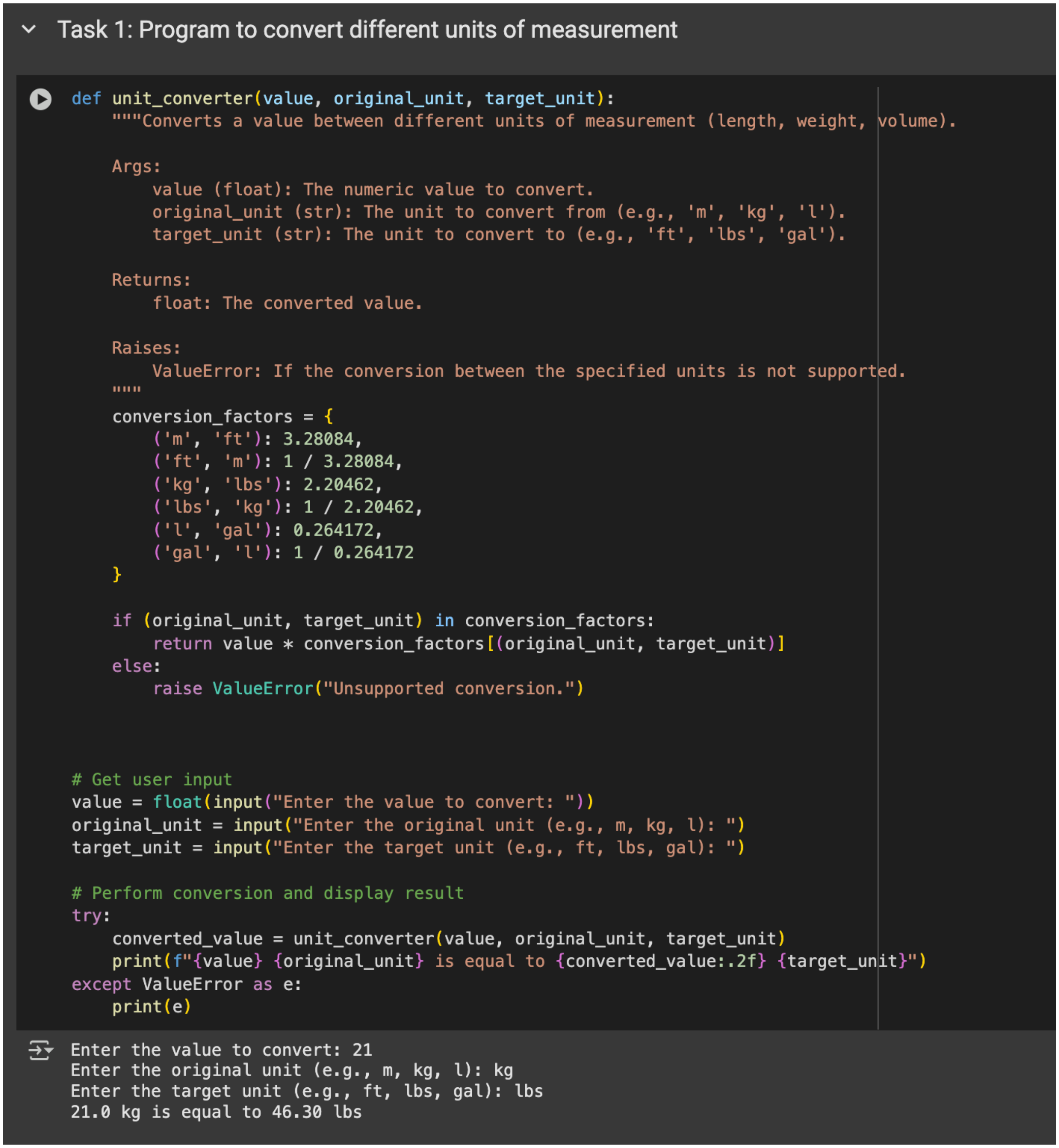
```
[x y z] = [ 2.  1. -2.]
```



```python
def py_matrix_multiplication(matrix1, matrix2):
    result = [[0 for _ in range(len(matrix2[0]))] for _ in range(len(matrix1))]
    for i in range(len(matrix1)):
        for j in range(len(matrix2[0])):
            for k in range(len(matrix2)):
                result[i][j] += matrix1[i][k] * matrix2[k][j]
    return result

# Perform operations and measure time
operations = {
    "Addition": (py_addition, np.add),
    "Multiplication": (py_multiplication, np.multiply),
    "Dot Product": (py_dot_product, np.dot),
}

for name, (py_op, np_op) in operations.items():
    py_time = time_operation(py_op, py_list1, py_list2)
    np_time = time_operation(np_op, np_array1, np_array2)
    print(f"{name} Time:\nNumPy: {np_time:.5f}\nPython List: {py_time:.5f}\n")

# Matrix Multiplication
py_matrix1 = [[j for j in range(matrices)] for i in range(matrices)]
py_matrix2 = [[j for j in range(matrices)] for i in range(matrices)]
np_matrix1 = np.arange(matrices**2).reshape(matrices, matrices)
np_matrix2 = np.arange(matrices**2).reshape(matrices, matrices)

py_mat_mul_time = time_operation(py_matrix_multiplication, py_matrix1, py_matrix2)
np_mat_mul_time = time_operation(np.dot, np_matrix1, np_matrix2)

print(f"Matrix Multiplication Time:\nNumPy: {np_mat_mul_time:.5f}\nPython List: {py_mat_mul_time:.5f}\n")
```

```
Addition Time:
NumPy: 0.00018
Python List: 0.00857

Multiplication Time:
NumPy: 0.00019
Python List: 0.00806

Dot Product Time:
NumPy: 0.00017
Python List: 0.00852

Matrix Multiplication Time:
NumPy: 0.00098
Python List: 0.12101
```
]]>

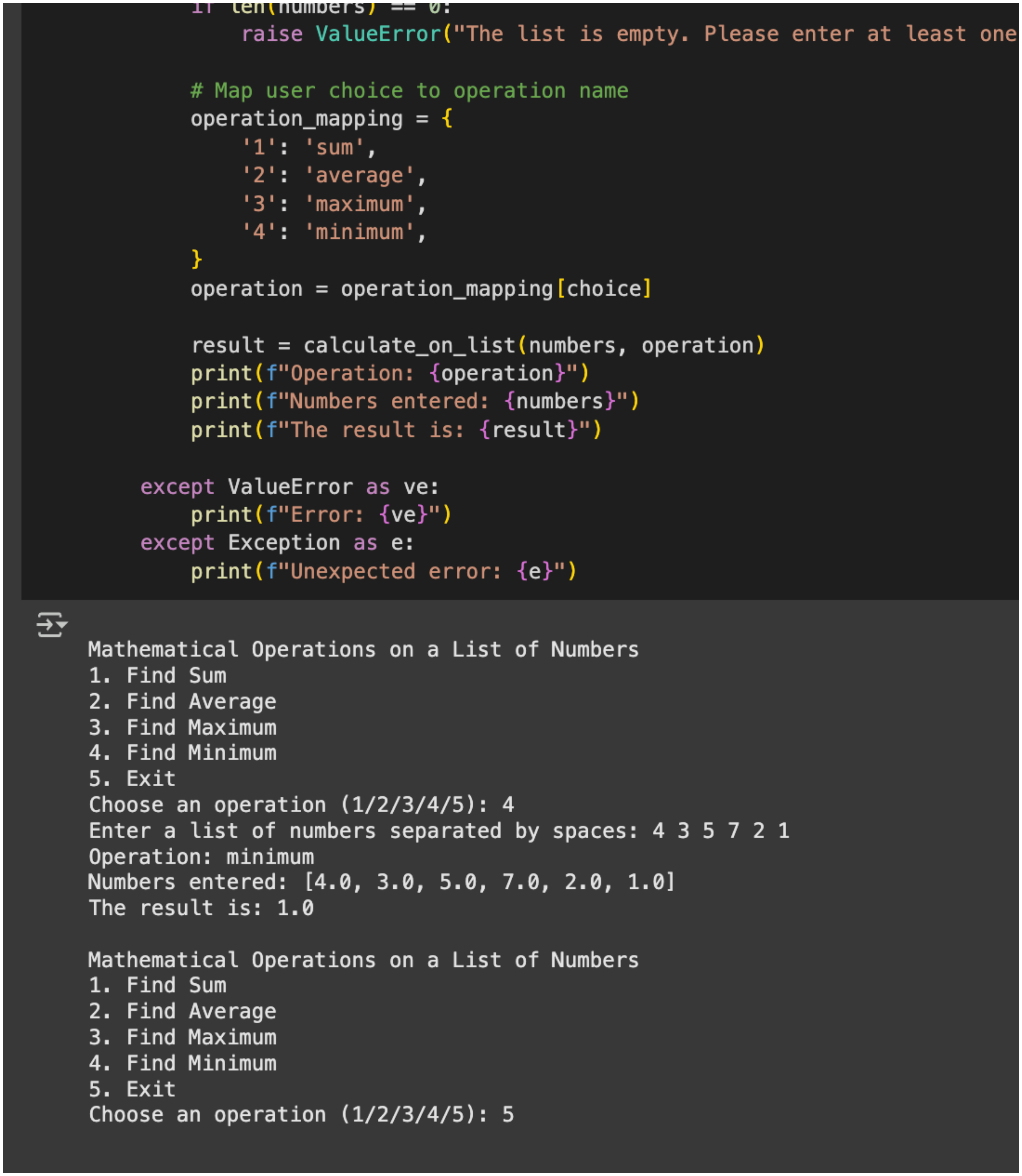
5. To Do



Task 1: Program to convert different units of measurement

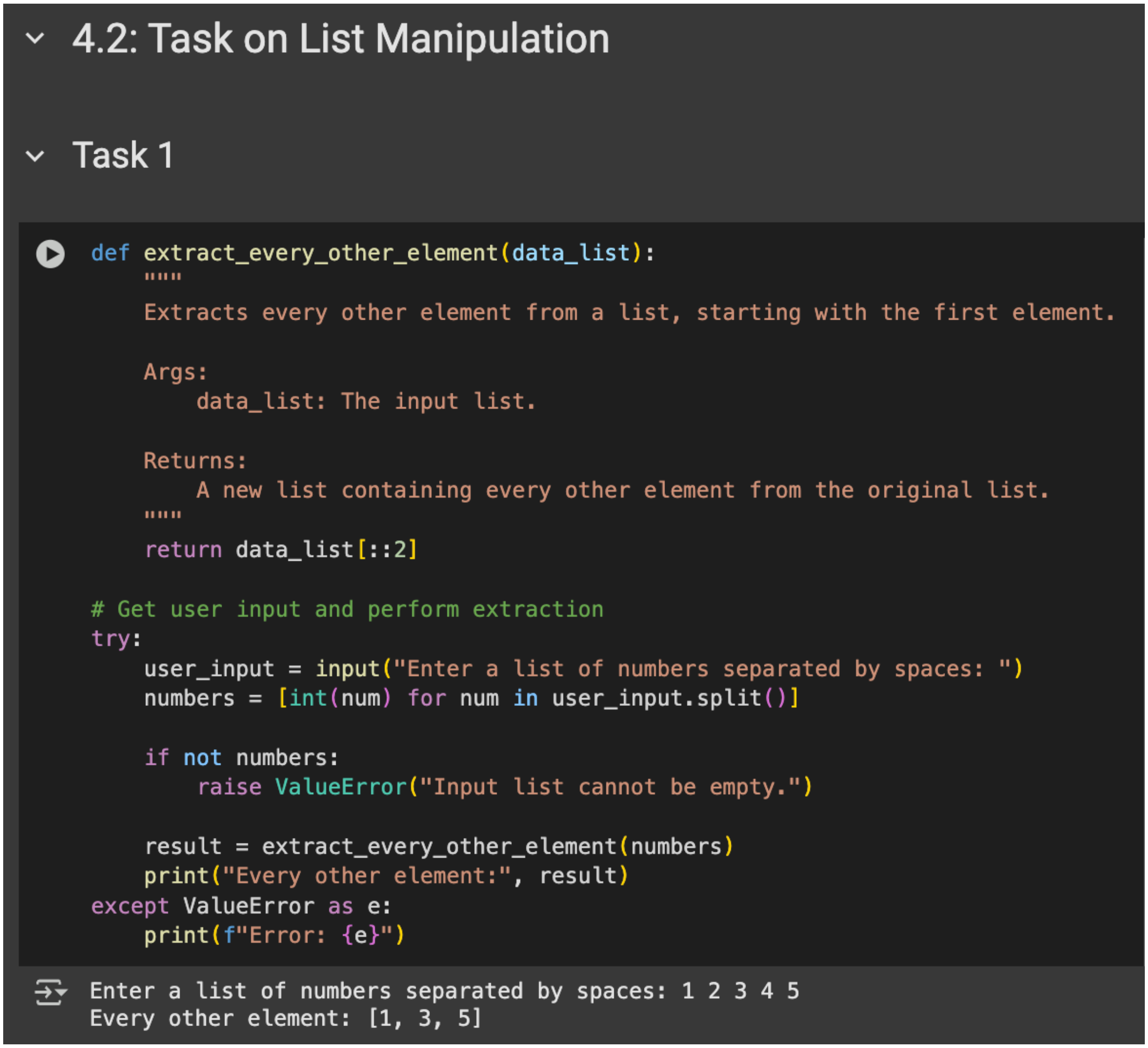
```
def unit_converter(value, original_unit, target_unit):
    """Converts a value between different units of measurement (length, weight, volume).

    Args:
        value (float): The numeric value to convert.
        original_unit (str): The unit to convert from (e.g., 'm', 'kg', 'l').
        target_unit (str): The unit to convert to (e.g., 'ft', 'lbs', 'gal').

    Returns:
        float: The converted value.

    Raises:
        ValueError: If the conversion between the specified units is not supported.
    """
    conversion_factors = {
        ('m', 'ft'): 3.28084,
        ('ft', 'm'): 1 / 3.28084,
        ('kg', 'lbs'): 2.20462,
        ('lbs', 'kg'): 1 / 2.20462,
        ('l', 'gal'): 0.264172,
        ('gal', 'l'): 1 / 0.264172
    }

    if (original_unit, target_unit) in conversion_factors:
        return value * conversion_factors[(original_unit, target_unit)]
    else:
        raise ValueError("Unsupported conversion.")



# Get user input
value = float(input("Enter the value to convert: "))
original_unit = input("Enter the original unit (e.g., m, kg, l): ")
target_unit = input("Enter the target unit (e.g., ft, lbs, gal): ")

# Perform conversion and display result
try:
    converted_value = unit_converter(value, original_unit, target_unit)
    print(f"{value} {original_unit} is equal to {converted_value:.2f} {target_unit}")
except ValueError as e:
    print(e)
```

```
Enter the value to convert: 21
Enter the original unit (e.g., m, kg, l): kg
Enter the target unit (e.g., ft, lbs, gal): lbs
21.0 kg is equal to 46.30 lbs
```



```python
        if len(numbers) == 0:
            raise ValueError("The list is empty. Please enter at least one

        # Map user choice to operation name
        operation_mapping = {
            '1': 'sum',
            '2': 'average',
            '3': 'maximum',
            '4': 'minimum',
        }
        operation = operation_mapping[choice]

        result = calculate_on_list(numbers, operation)
        print(f"Operation: {operation}")
        print(f"Numbers entered: {numbers}")
        print(f"The result is: {result}")

    except ValueError as ve:
        print(f"Error: {ve}")
    except Exception as e:
        print(f"Unexpected error: {e}")
```

```
Mathematical Operations on a List of Numbers
1. Find Sum
2. Find Average
3. Find Maximum
4. Find Minimum
5. Exit
Choose an operation (1/2/3/4/5): 4
Enter a list of numbers separated by spaces: 4 3 5 7 2 1
Operation: minimum
Numbers entered: [4.0, 3.0, 5.0, 7.0, 2.0, 1.0]
The result is: 1.0

Mathematical Operations on a List of Numbers
1. Find Sum
2. Find Average
3. Find Maximum
4. Find Minimum
5. Exit
Choose an operation (1/2/3/4/5): 5
```

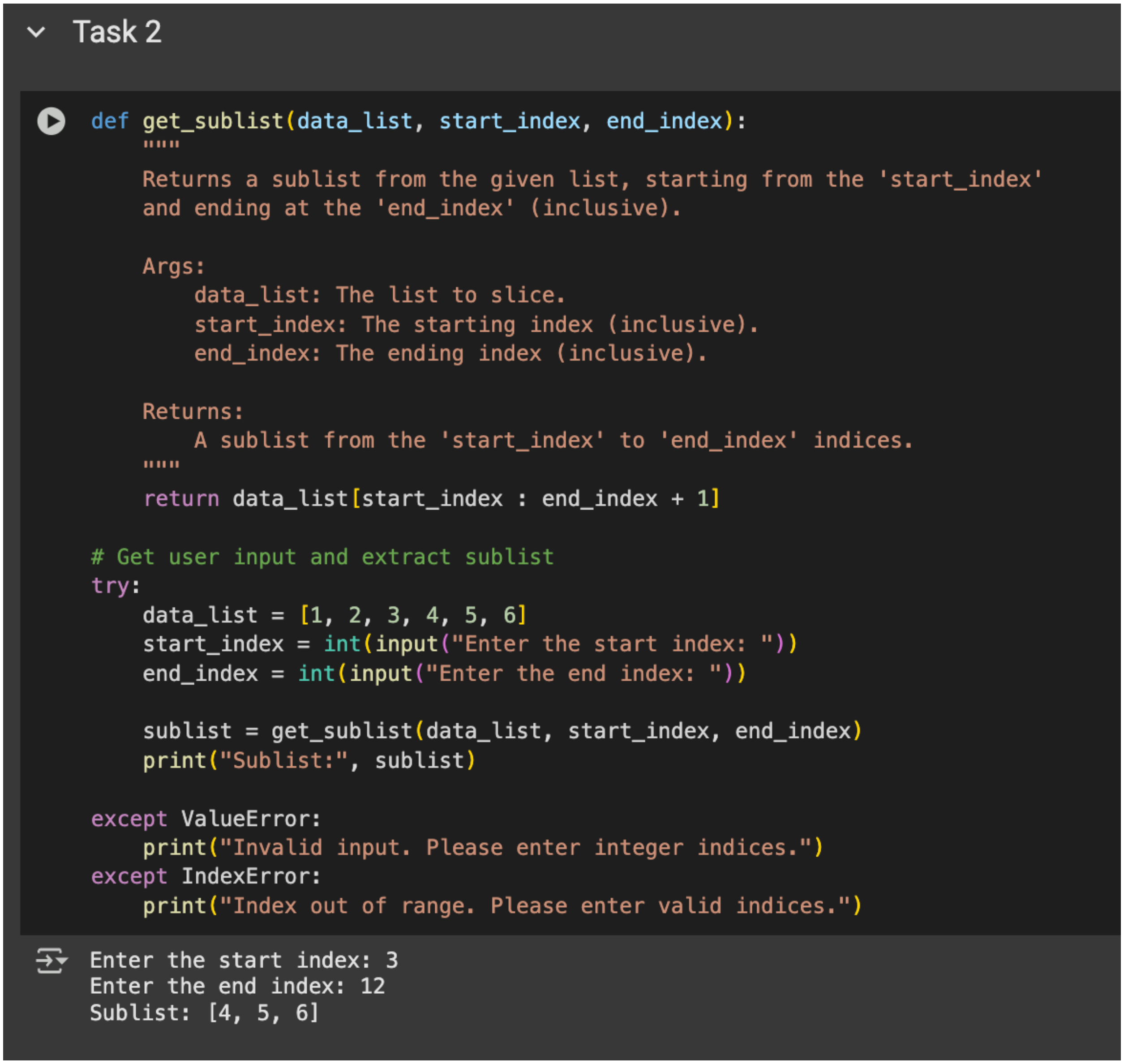


4.2: Task on List Manipulation

Task 1

```python
def extract_every_other_element(data_list):
    """
    Extracts every other element from a list, starting with the first element.

    Args:
        data_list: The input list.

    Returns:
        A new list containing every other element from the original list.
    """
    return data_list[::2]

# Get user input and perform extraction
try:
    user_input = input("Enter a list of numbers separated by spaces: ")
    numbers = [int(num) for num in user_input.split()]

    if not numbers:
        raise ValueError("Input list cannot be empty.")

    result = extract_every_other_element(numbers)
    print("Every other element:", result)
except ValueError as e:
    print(f"Error: {e}")
```

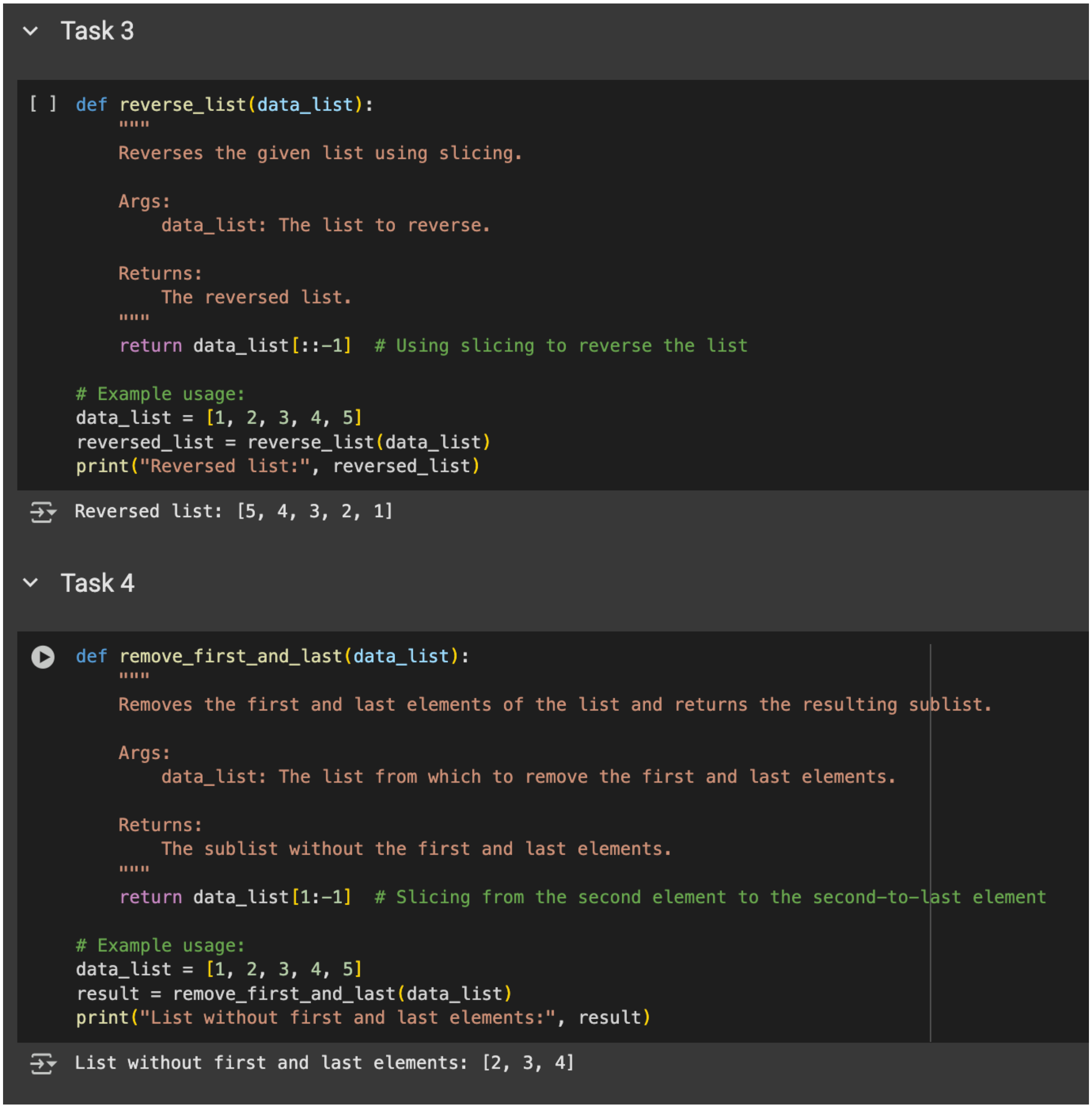
```
Enter a list of numbers separated by spaces: 1 2 3 4 5
Every other element: [1, 3, 5]
```



Task 2

```python
def get_sublist(data_list, start_index, end_index):
    """
    Returns a sublist from the given list, starting from the 'start_index'
    and ending at the 'end_index' (inclusive).

    Args:
        data_list: The list to slice.
        start_index: The starting index (inclusive).
        end_index: The ending index (inclusive).

    Returns:
        A sublist from the 'start_index' to 'end_index' indices.
    """
    return data_list[start_index : end_index + 1]

# Get user input and extract sublist
try:
    data_list = [1, 2, 3, 4, 5, 6]
    start_index = int(input("Enter the start index: "))
    end_index = int(input("Enter the end index: "))

    sublist = get_sublist(data_list, start_index, end_index)
    print("Sublist:", sublist)

except ValueError:
    print("Invalid input. Please enter integer indices.")
except IndexError:
    print("Index out of range. Please enter valid indices.")
```

```
Enter the start index: 3
Enter the end index: 12
Sublist: [4, 5, 6]
```



Task 3

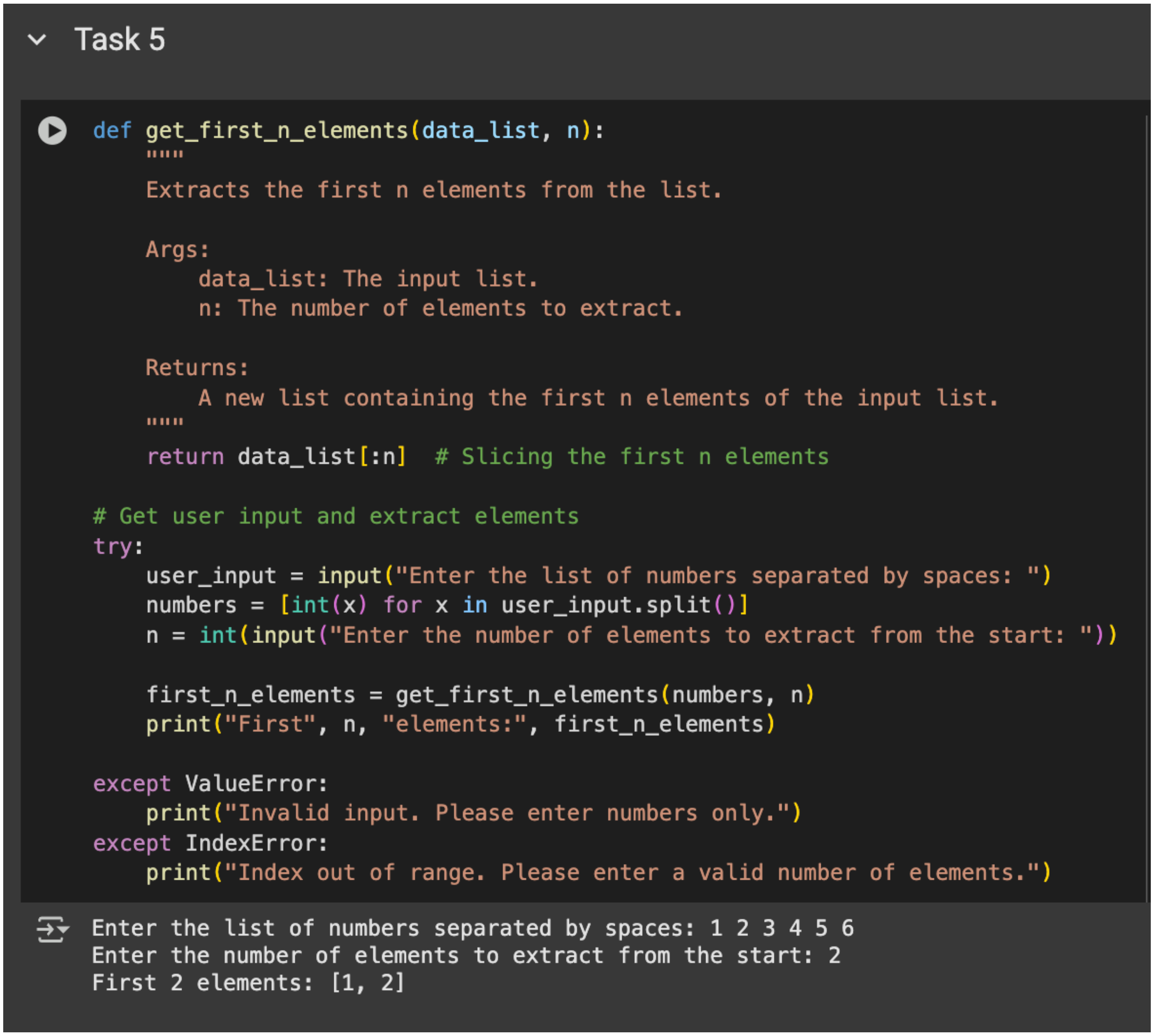
```python
def reverse_list(data_list):
    """
    Reverses the given list using slicing.

    Args:
        data_list: The list to reverse.

    Returns:
        The reversed list.
    """
    return data_list[::-1]  # Using slicing to reverse the list

# Example usage:
data_list = [1, 2, 3, 4, 5]
reversed_list = reverse_list(data_list)
print("Reversed list:", reversed_list)
```

```
Reversed list: [5, 4, 3, 2, 1]
```

Task 4

```python
def remove_first_and_last(data_list):
    """
    Removes the first and last elements of the list and returns the resulting sublist.

    Args:
        data_list: The list from which to remove the first and last elements.

    Returns:
        The sublist without the first and last elements.
    """
    return data_list[1:-1]  # Slicing from the second element to the second-to-last element

# Example usage:
data_list = [1, 2, 3, 4, 5]
result = remove_first_and_last(data_list)
print("List without first and last elements:", result)
```

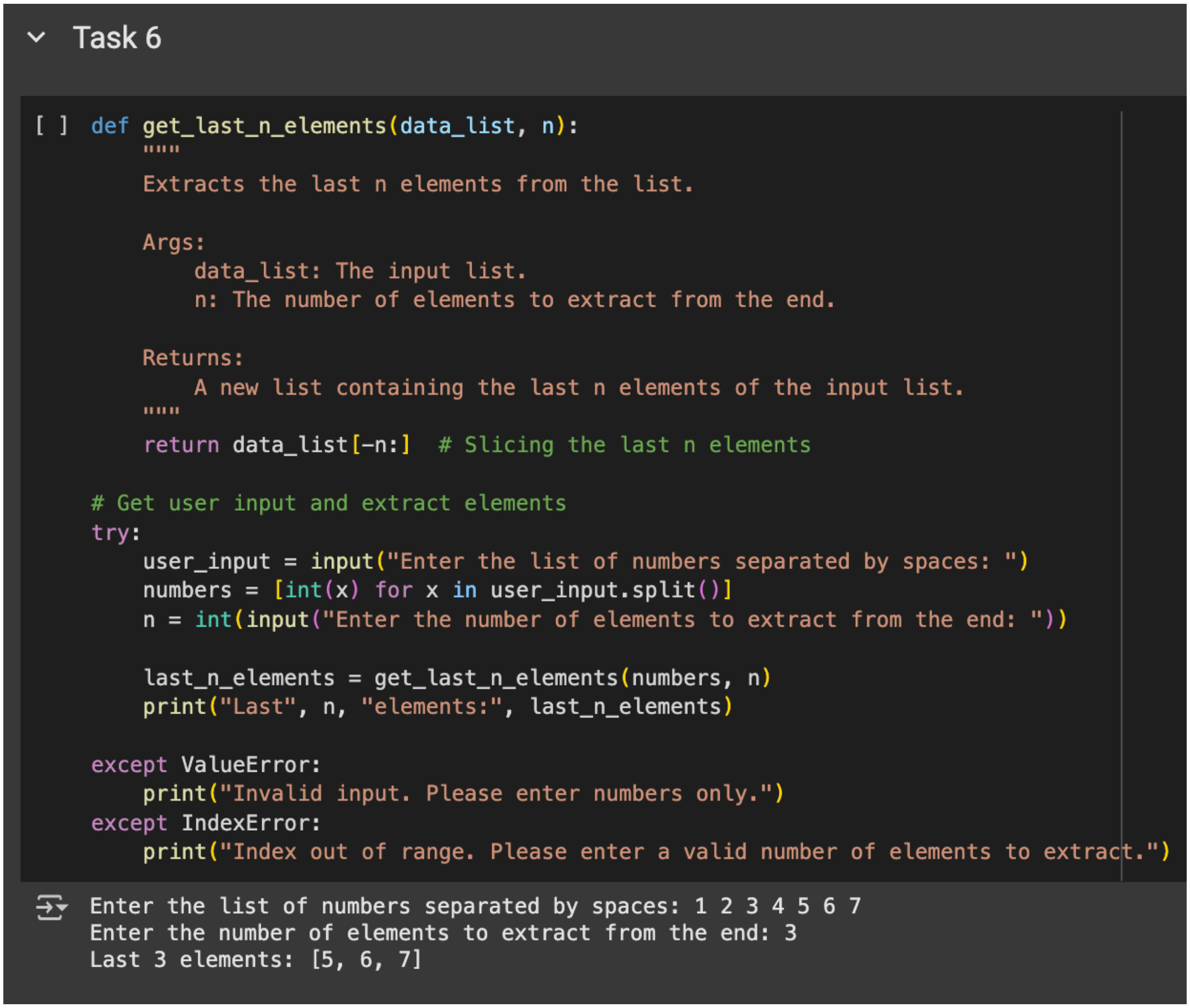
```
List without first and last elements: [2, 3, 4]
```



Task 5

```python
def get_first_n_elements(data_list, n):
    """
    Extracts the first n elements from the list.

    Args:
        data_list: The input list.
        n: The number of elements to extract.

    Returns:
        A new list containing the first n elements of the input list.
    """
    return data_list[:n]  # Slicing the first n elements

# Get user input and extract elements
try:
    user_input = input("Enter the list of numbers separated by spaces: ")
    numbers = [int(x) for x in user_input.split()]
    n = int(input("Enter the number of elements to extract from the start: "))

    first_n_elements = get_first_n_elements(numbers, n)
    print("First", n, "elements:", first_n_elements)

except ValueError:
    print("Invalid input. Please enter numbers only.")
except IndexError:
    print("Index out of range. Please enter a valid number of elements.")
```

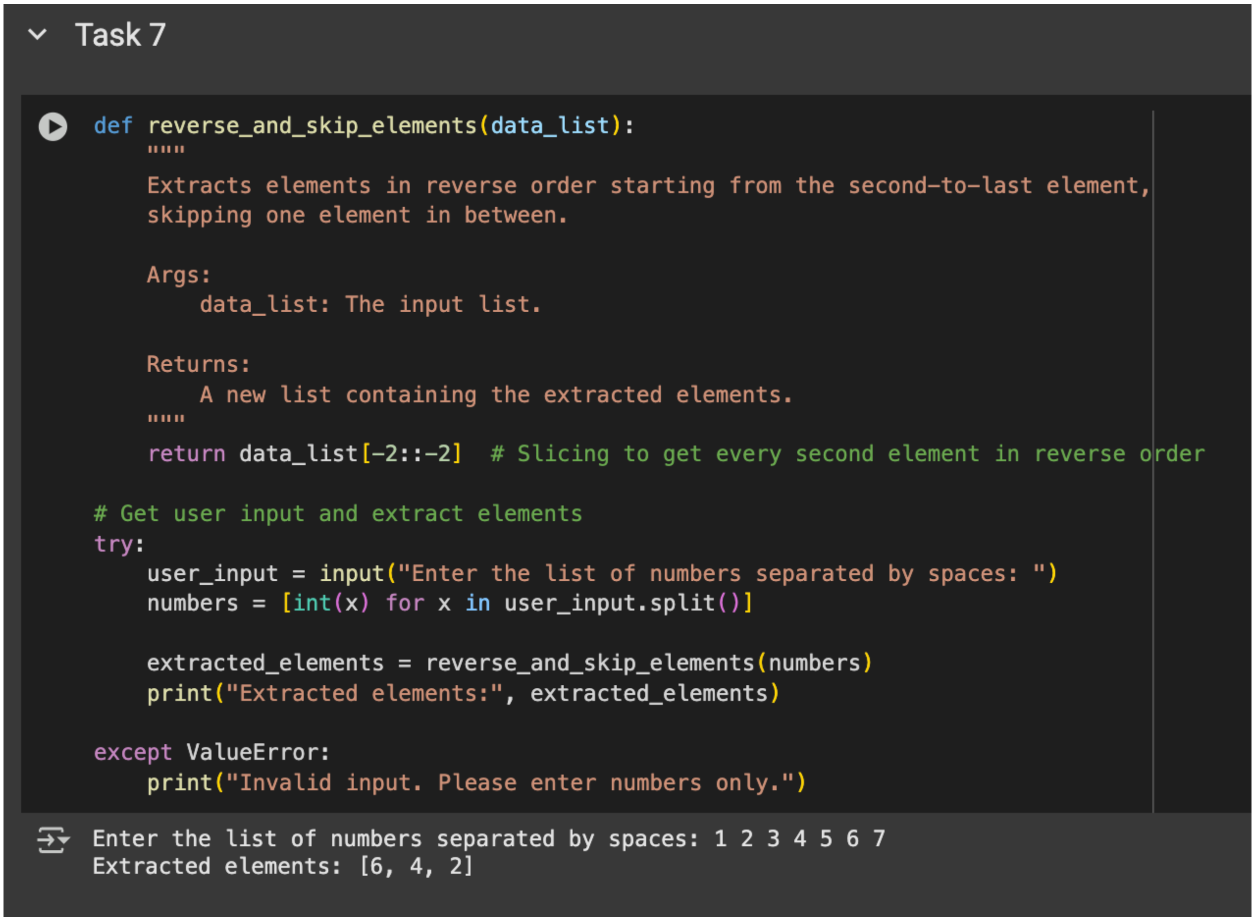
```
Enter the list of numbers separated by spaces: 1 2 3 4 5 6
Enter the number of elements to extract from the start: 2
First 2 elements: [1, 2]
```



Task 6

```python
def get_last_n_elements(data_list, n):
    """
    Extracts the last n elements from the list.

    Args:
        data_list: The input list.
        n: The number of elements to extract from the end.

    Returns:
        A new list containing the last n elements of the input list.
    """
    return data_list[-n:]  # Slicing the last n elements

# Get user input and extract elements
try:
    user_input = input("Enter the list of numbers separated by spaces: ")
    numbers = [int(x) for x in user_input.split()]
    n = int(input("Enter the number of elements to extract from the end: "))

    last_n_elements = get_last_n_elements(numbers, n)
    print("Last", n, "elements:", last_n_elements)

except ValueError:
    print("Invalid input. Please enter numbers only.")
except IndexError:
    print("Index out of range. Please enter a valid number of elements to extract.")
```

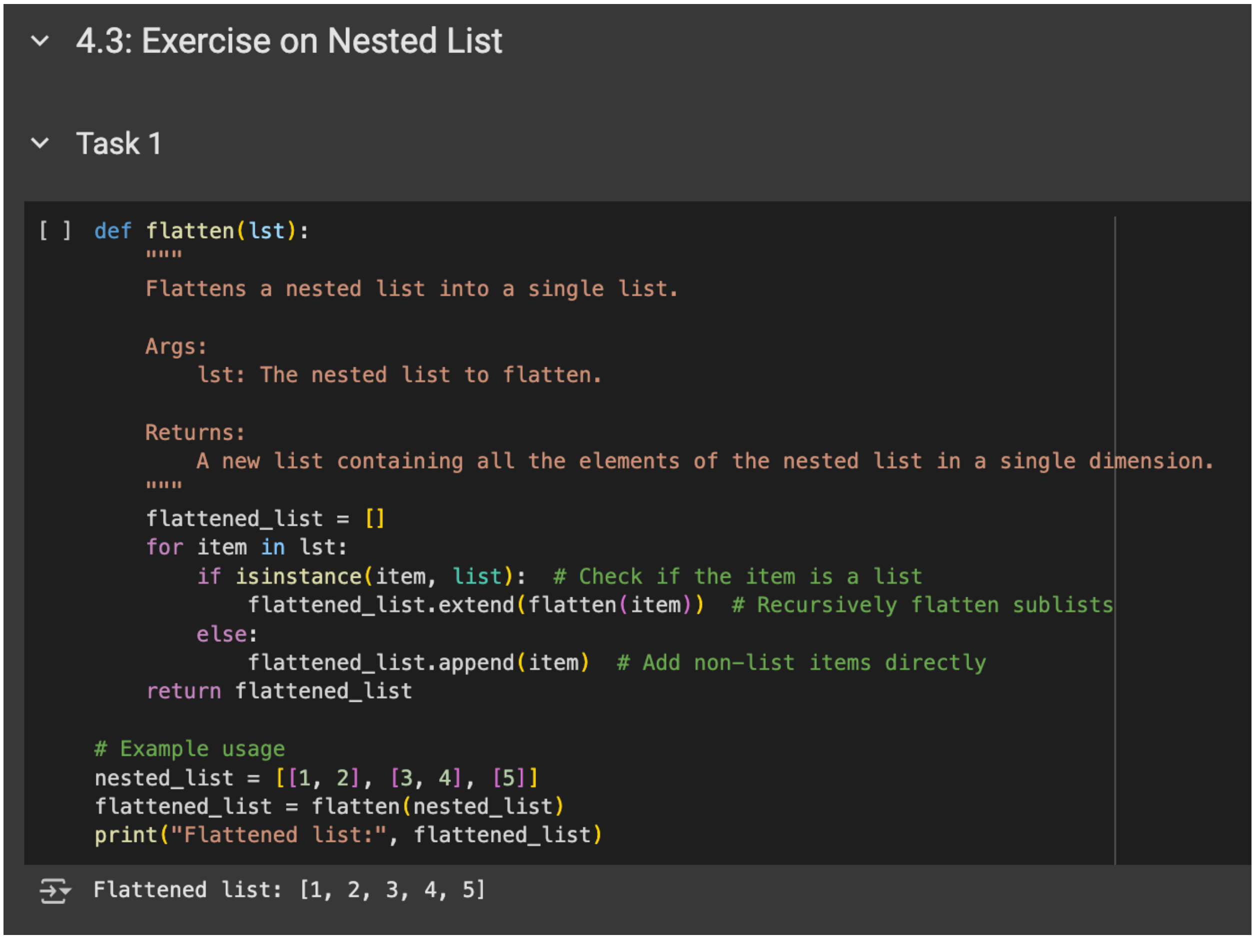
```
Enter the list of numbers separated by spaces: 1 2 3 4 5 6 7
Enter the number of elements to extract from the end: 3
Last 3 elements: [5, 6, 7]
```



Task 7

```
def reverse_and_skip_elements(data_list):
    """
    Extracts elements in reverse order starting from the second-to-last element,
    skipping one element in between.

    Args:
        data_list: The input list.

    Returns:
        A new list containing the extracted elements.
    """
    return data_list[-2::-2]  # Slicing to get every second element in reverse order

# Get user input and extract elements
try:
    user_input = input("Enter the list of numbers separated by spaces: ")
    numbers = [int(x) for x in user_input.split()]

    extracted_elements = reverse_and_skip_elements(numbers)
    print("Extracted elements:", extracted_elements)

except ValueError:
    print("Invalid input. Please enter numbers only.")
```

```
Enter the list of numbers separated by spaces: 1 2 3 4 5 6 7
Extracted elements: [6, 4, 2]
```

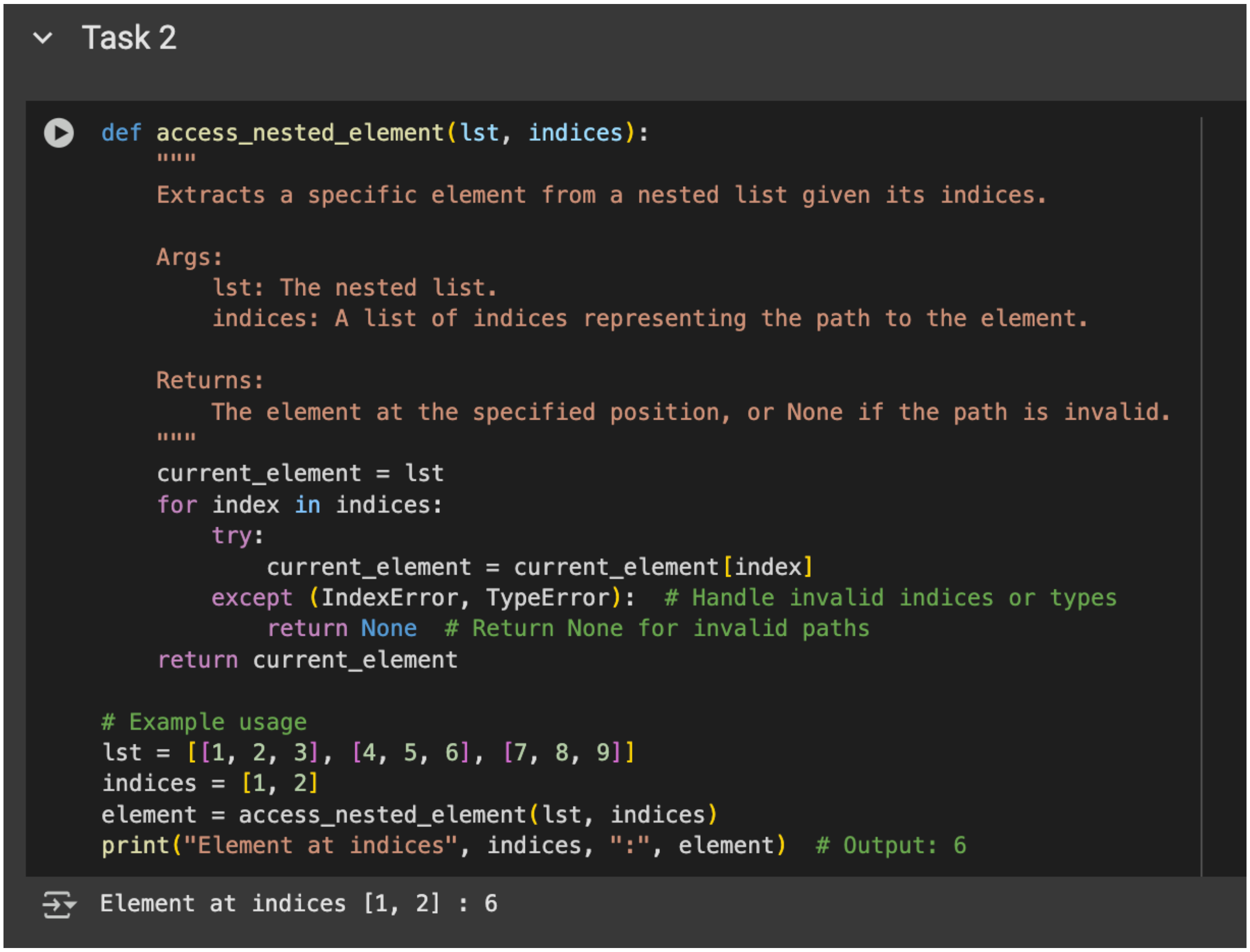


4.3: Exercise on Nested List

Task 1

```
[ ] def flatten(lst):
        """
        Flattens a nested list into a single list.

        Args:
            lst: The nested list to flatten.

        Returns:
            A new list containing all the elements of the nested list in a single dimension.
        """
        flattened_list = []
        for item in lst:
            if isinstance(item, list):  # Check if the item is a list
                flattened_list.extend(flatten(item))  # Recursively flatten sublists
            else:
                flattened_list.append(item)  # Add non-list items directly
        return flattened_list

    # Example usage
    nested_list = [[1, 2], [3, 4], [5]]
    flattened_list = flatten(nested_list)
    print("Flattened list:", flattened_list)
```

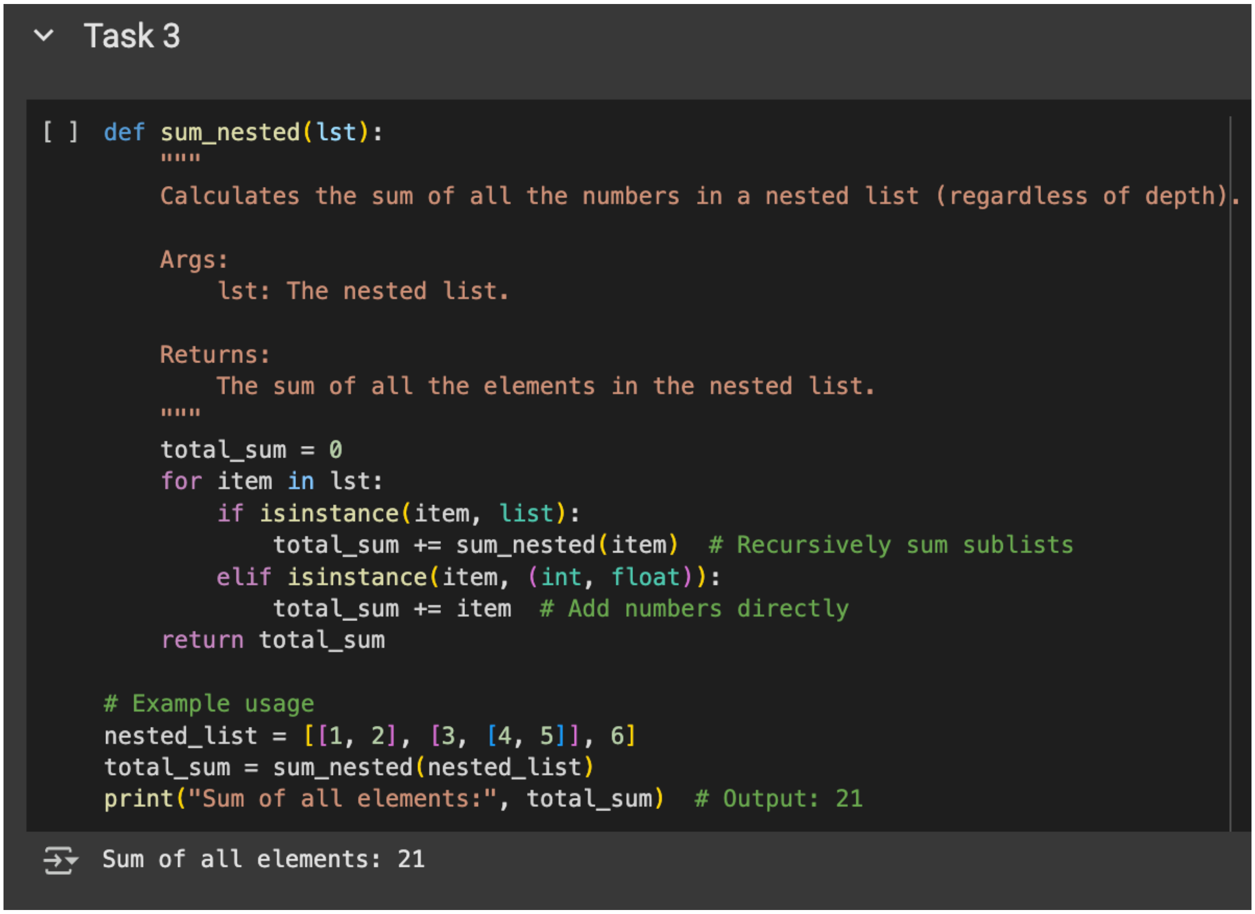
```
Flattened list: [1, 2, 3, 4, 5]
```



Task 2

```python
def access_nested_element(lst, indices):
    """
    Extracts a specific element from a nested list given its indices.

    Args:
        lst: The nested list.
        indices: A list of indices representing the path to the element.

    Returns:
        The element at the specified position, or None if the path is invalid.
    """
    current_element = lst
    for index in indices:
        try:
            current_element = current_element[index]
        except (IndexError, TypeError):  # Handle invalid indices or types
            return None  # Return None for invalid paths
    return current_element

# Example usage
lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
indices = [1, 2]
element = access_nested_element(lst, indices)
print("Element at indices", indices, ":", element)  # Output: 6
```

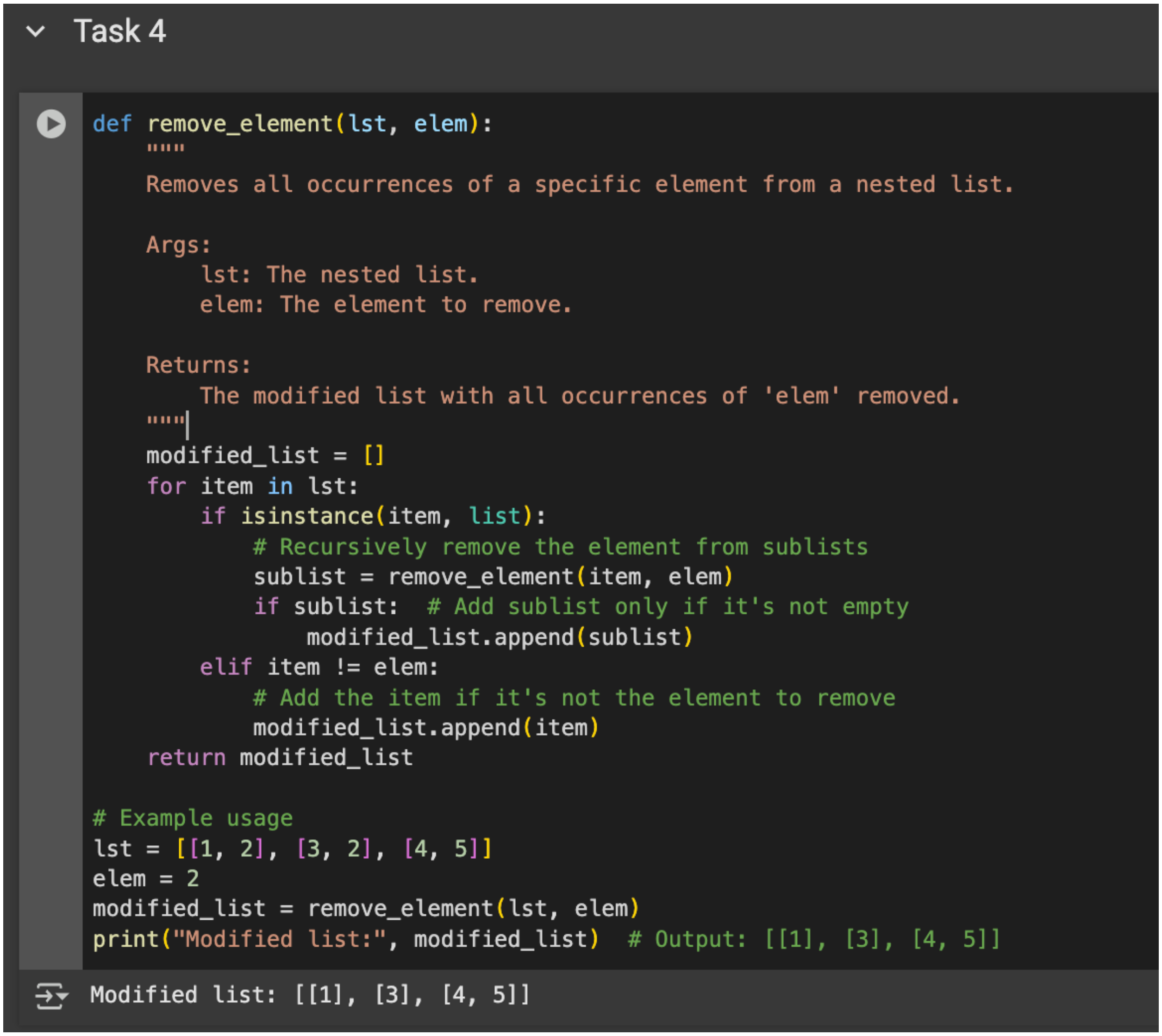
```
Element at indices [1, 2] : 6
```



Task 3

```python
def sum_nested(lst):
    """
    Calculates the sum of all the numbers in a nested list (regardless of depth).

    Args:
        lst: The nested list.

    Returns:
        The sum of all the elements in the nested list.
    """
    total_sum = 0
    for item in lst:
        if isinstance(item, list):
            total_sum += sum_nested(item)  # Recursively sum sublists
        elif isinstance(item, (int, float)):
            total_sum += item  # Add numbers directly
    return total_sum

# Example usage
nested_list = [[1, 2], [3, [4, 5]], 6]
total_sum = sum_nested(nested_list)
print("Sum of all elements:", total_sum)  # Output: 21
```

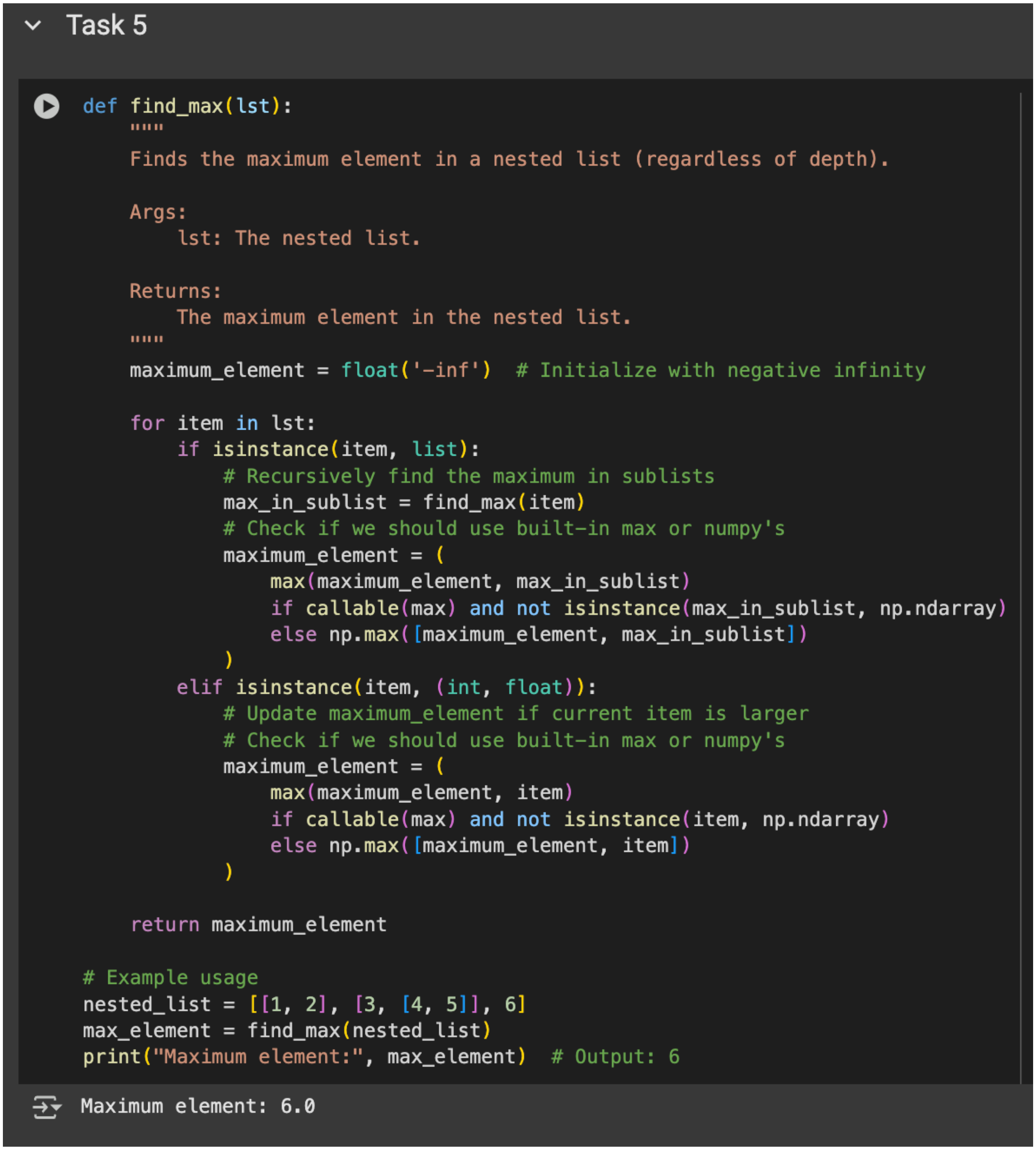
```
Sum of all elements: 21
```



Task 4

```python
def remove_element(lst, elem):
    """
    Removes all occurrences of a specific element from a nested list.

    Args:
        lst: The nested list.
        elem: The element to remove.

    Returns:
        The modified list with all occurrences of 'elem' removed.
    """
    modified_list = []
    for item in lst:
        if isinstance(item, list):
            # Recursively remove the element from sublists
            sublist = remove_element(item, elem)
            if sublist:  # Add sublist only if it's not empty
                modified_list.append(sublist)
        elif item != elem:
            # Add the item if it's not the element to remove
            modified_list.append(item)
    return modified_list

# Example usage
lst = [[1, 2], [3, 2], [4, 5]]
elem = 2
modified_list = remove_element(lst, elem)
print("Modified list:", modified_list)  # Output: [[1], [3], [4, 5]]
```

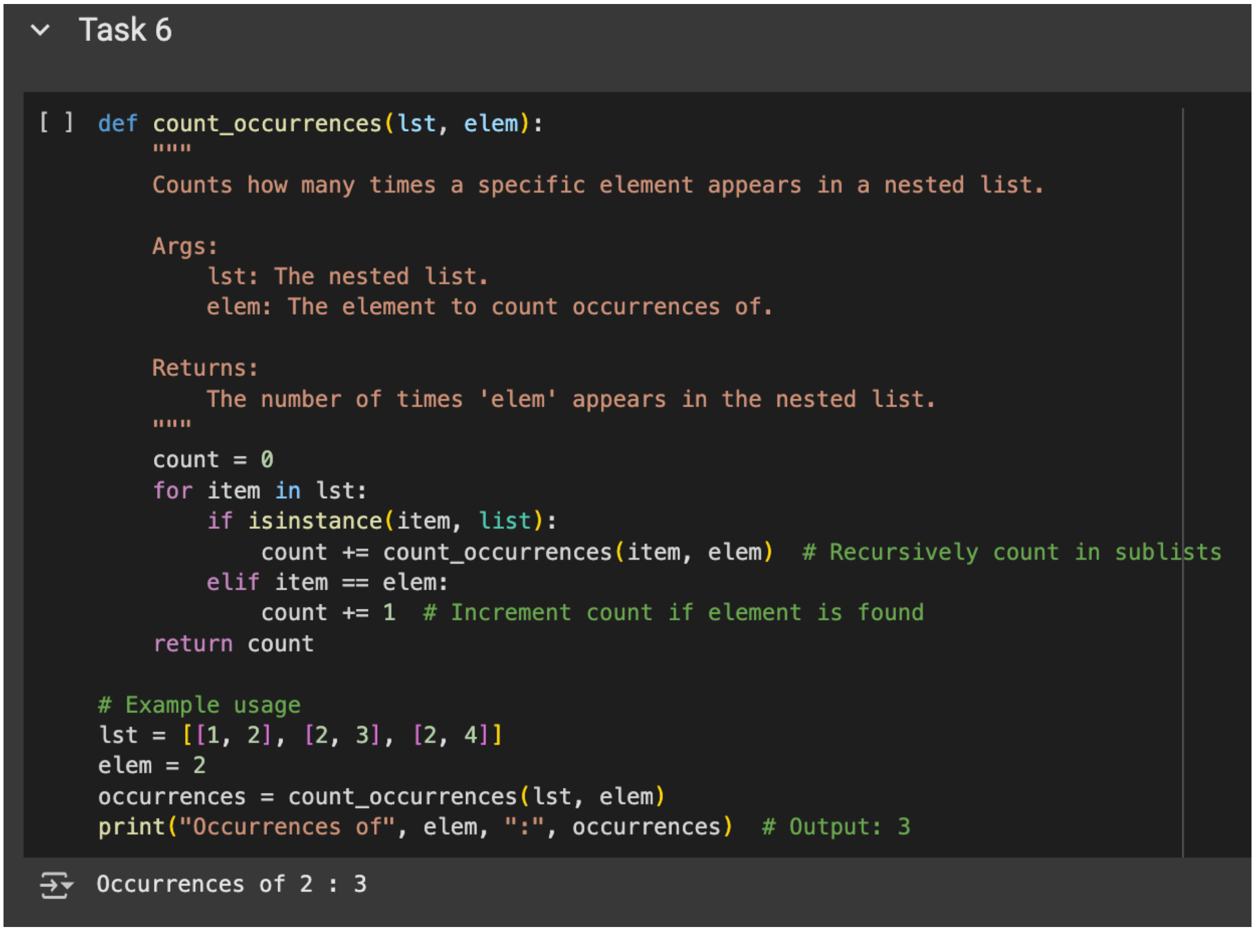
```
Modified list: [[1], [3], [4, 5]]
```



Task 5

```python
def find_max(lst):
    """
    Finds the maximum element in a nested list (regardless of depth).

    Args:
        lst: The nested list.

    Returns:
        The maximum element in the nested list.
    """
    maximum_element = float('-inf')  # Initialize with negative infinity

    for item in lst:
        if isinstance(item, list):
            # Recursively find the maximum in sublists
            max_in_sublist = find_max(item)
            # Check if we should use built-in max or numpy's
            maximum_element = (
                max(maximum_element, max_in_sublist)
                if callable(max) and not isinstance(max_in_sublist, np.ndarray)
                else np.max([maximum_element, max_in_sublist])
            )
        elif isinstance(item, (int, float)):
            # Update maximum_element if current item is larger
            # Check if we should use built-in max or numpy's
            maximum_element = (
                max(maximum_element, item)
                if callable(max) and not isinstance(item, np.ndarray)
                else np.max([maximum_element, item])
            )

    return maximum_element

# Example usage
nested_list = [[1, 2], [3, [4, 5]], 6]
max_element = find_max(nested_list)
print("Maximum element:", max_element)  # Output: 6
```

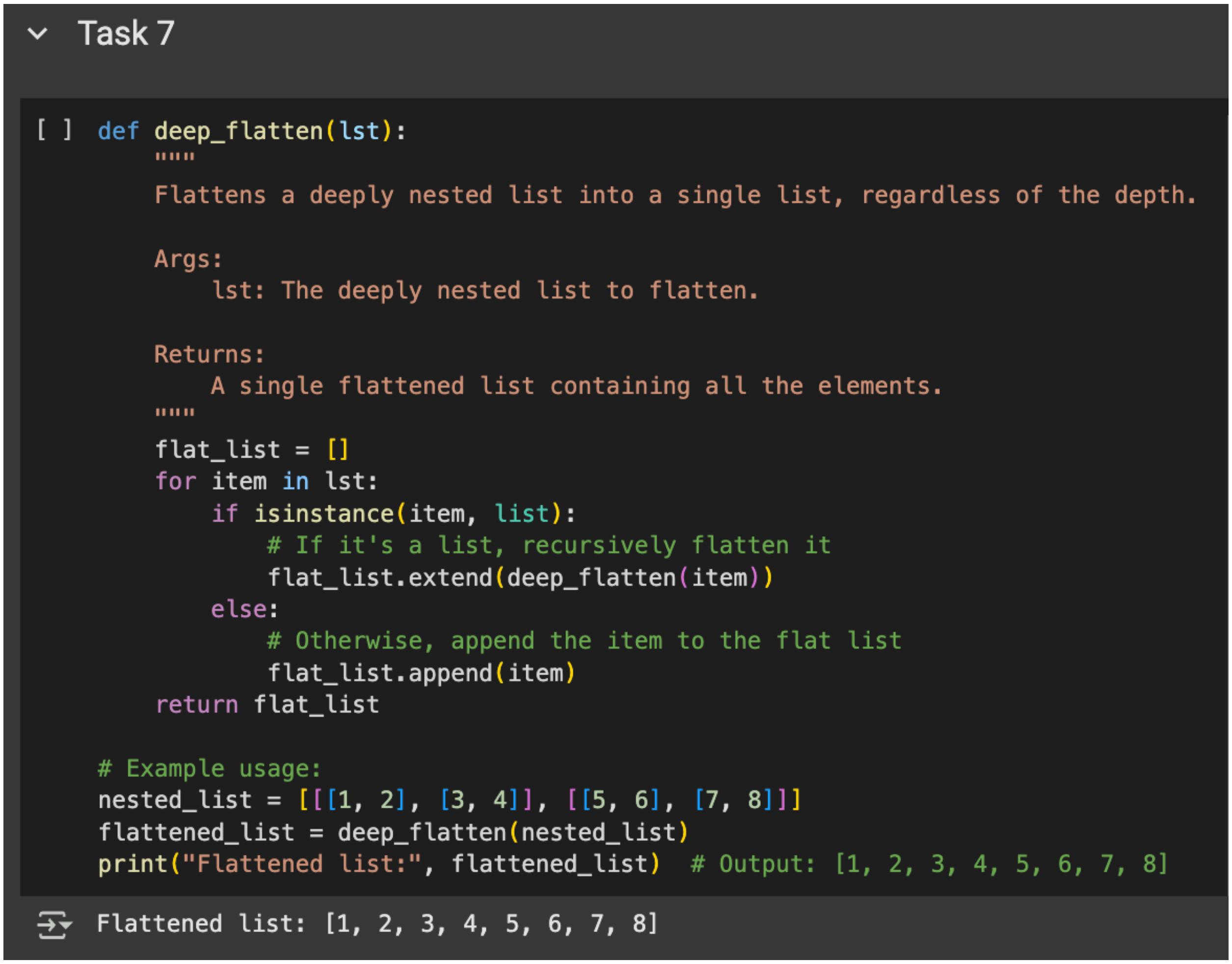
```
Maximum element: 6.0
```



Task 6

```
def count_occurrences(lst, elem):
    """
    Counts how many times a specific element appears in a nested list.

    Args:
        lst: The nested list.
        elem: The element to count occurrences of.

    Returns:
        The number of times 'elem' appears in the nested list.
    """
    count = 0
    for item in lst:
        if isinstance(item, list):
            count += count_occurrences(item, elem)  # Recursively count in sublists
        elif item == elem:
            count += 1  # Increment count if element is found
    return count

# Example usage
lst = [[1, 2], [2, 3], [2, 4]]
elem = 2
occurrences = count_occurrences(lst, elem)
print("Occurrences of", elem, ":", occurrences)  # Output: 3
```

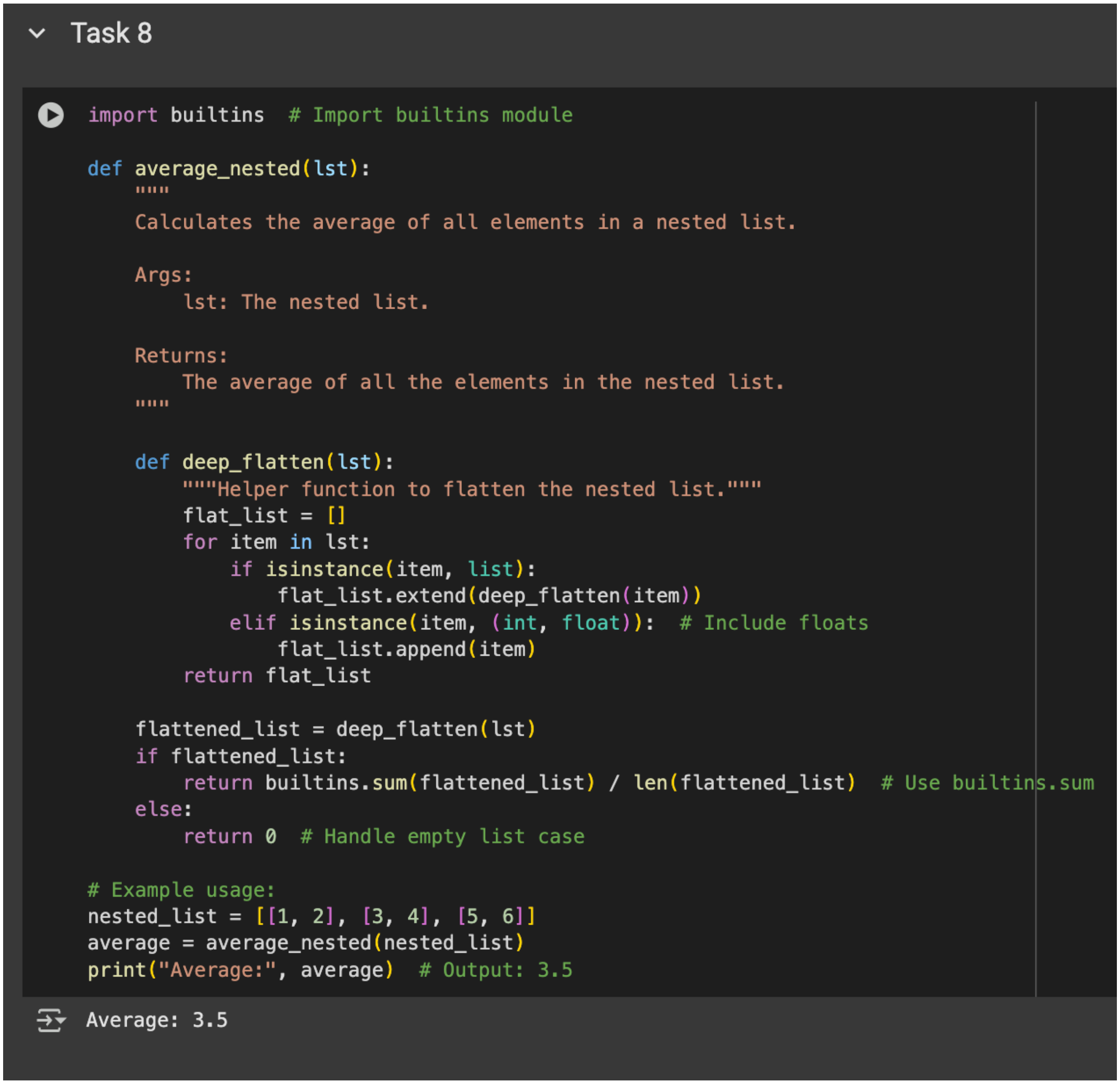
```
Occurrences of 2 : 3
```



Task 7

```python
def deep_flatten(lst):
    """
    Flattens a deeply nested list into a single list, regardless of the depth.

    Args:
        lst: The deeply nested list to flatten.

    Returns:
        A single flattened list containing all the elements.
    """
    flat_list = []
    for item in lst:
        if isinstance(item, list):
            # If it's a list, recursively flatten it
            flat_list.extend(deep_flatten(item))
        else:
            # Otherwise, append the item to the flat list
            flat_list.append(item)
    return flat_list

# Example usage:
nested_list = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
flattened_list = deep_flatten(nested_list)
print("Flattened list:", flattened_list)  # Output: [1, 2, 3, 4, 5, 6, 7, 8]
```

```
Flattened list: [1, 2, 3, 4, 5, 6, 7, 8]
```



Task 8

```python
import builtins  # Import builtins module

def average_nested(lst):
    """
    Calculates the average of all elements in a nested list.

    Args:
        lst: The nested list.

    Returns:
        The average of all the elements in the nested list.
    """

    def deep_flatten(lst):
        """Helper function to flatten the nested list."""
        flat_list = []
        for item in lst:
            if isinstance(item, list):
                flat_list.extend(deep_flatten(item))
            elif isinstance(item, (int, float)):  # Include floats
                flat_list.append(item)
        return flat_list

    flattened_list = deep_flatten(lst)
    if flattened_list:
        return builtins.sum(flattened_list) / len(flattened_list)  # Use builtins.sum
    else:
        return 0  # Handle empty list case

# Example usage:
nested_list = [[1, 2], [3, 4], [5, 6]]
average = average_nested(nested_list)
print("Average:", average)  # Output: 3.5
```

```
Average: 3.5
```