Worksheet 0

Name : Bibek Timalsina

UniID: 2331017

Exercise 4.1

Task 1

```
[ ] def convert_length(value, unit):
        """
        Converts length between meters and feet.
        :param value: float, the numeric value to convert.
        :param unit: str, 'm_to_ft' for meters to feet, 'ft_to_m' for feet to meters.
        :return: float, converted value.
        """
        if unit == 'm_to_ft':
            return value * 3.28084
        elif unit == 'ft_to_m':
            return value / 3.28084
        else:
            raise ValueError("Invalid unit for length conversion.")

    def convert_weight(value, unit):
        """
        Converts weight between kilograms and pounds.
        :param value: float, the numeric value to convert.
        :param unit: str, 'kg_to_lbs' for kg to lbs, 'lbs_to_kg' for lbs to kg.
        :return: float, converted value.
        """
        if unit == 'kg_to_lbs':
            return value * 2.20462
        elif unit == 'lbs_to_kg':
            return value / 2.20462
        else:
            raise ValueError("Invalid unit for weight conversion.")

    def convert_volume(value, unit):
        """
        Converts volume between liters and gallons.
        :param value: float, the numeric value to convert.
        :param unit: str, 'L_to_gal' for liters to gallons, 'gal_to_L' for gallons to liters.
        :return: float, converted value.
        """
        if unit == 'L_to_gal':
            return value * 0.264172
        elif unit == 'gal_to_L':
            return value / 0.264172
        else:
            raise ValueError("Invalid unit for volume conversion.")
```

```python
def main():
    print("Unit Converter")
    print("1. Length (Meters to Feet / Feet to Meters)")
    print("2. Weight (Kilograms to Pounds / Pounds to Kilograms)")
    print("3. Volume (Liters to Gallons / Gallons to Liters)")

    try:
        choice = int(input("Select conversion type (1-3): "))
        value = float(input("Enter the value to convert: "))

        if choice == 1:
            unit = input("Enter 'm_to_ft' for meters to feet or 'ft_to_m' for feet to meters: ")
            result = convert_length(value, unit)
        elif choice == 2:
            unit = input("Enter 'kg_to_lbs' for kg to lbs or 'lbs_to_kg' for lbs to kg: ")
            result = convert_weight(value, unit)
        elif choice == 3:
            unit = input("Enter 'L_to_gal' for liters to gallons or 'gal_to_L' for gallons to liters: ")
            result = convert_volume(value, unit)
        else:
            print("Invalid choice.")
            return

        print(f"Converted value: {result:.4f}")
    except ValueError as e:
        print(f"Error: {e}")
    except Exception:
        print("An unexpected error occurred.")

if __name__ == "__main__":
    main()
```

```
Unit Converter
1. Length (Meters to Feet / Feet to Meters)
2. Weight (Kilograms to Pounds / Pounds to Kilograms)
3. Volume (Liters to Gallons / Gallons to Liters)
Select conversion type (1-3): 3
Enter the value to convert: 3
Enter 'L_to_gal' for liters to gallons or 'gal_to_L' for gallons to liters: gal_to_L
Converted value: 11.3562
```

Task 2

Task 2

```python
def calculate_sum(numbers):
    """
    Calculates the sum of a list of numbers.
    :param numbers: list of floats.
    :return: float, sum of numbers.
    """
    return sum(numbers)

def calculate_average(numbers):
    """
    Calculates the average of a list of numbers.
    :param numbers: list of floats.
    :return: float, average of numbers.
    """
    return sum(numbers) / len(numbers) if numbers else 0

def find_maximum(numbers):
    """
    Finds the maximum number in a list.
    :param numbers: list of floats.
    :return: float, maximum number.
    """
    return max(numbers)
```

```python
def find_minimum(numbers):
    """
    Finds the minimum number in a list.
    :param numbers: list of floats.
    :return: float, minimum number.
    """
    return min(numbers)

def main_math_operations():
    print("Mathematical Operations")
    try:
        numbers = input("Enter a list of numbers separated by spaces: ").split()
        numbers = [float(num) for num in numbers]
        print("Choose an operation: sum, average, max, min")
        operation = input("Enter operation: ").strip().lower()

        if operation == "sum":
            print(f"Sum: {calculate_sum(numbers)}")
        elif operation == "average":
            print(f"Average: {calculate_average(numbers)}")
        elif operation == "max":
            print(f"Maximum: {find_maximum(numbers)}")
        elif operation == "min":
            print(f"Minimum: {find_minimum(numbers)}")
        else:
            print("Invalid operation.")
    except ValueError:
        print("Invalid input. Please enter numbers only.")
    except Exception:
        print("An unexpected error occurred.")
```

```python
        print("An unexpected error occurred.")

def main():
    main_math_operations()

if __name__ == "__main__":
    main()
```

```
Mathematical Operations
Enter a list of numbers separated by spaces: 1 2 34 5 6 76 32 9 81 22 45
Choose an operation: sum, average, max, min
Enter operation: sum
Sum: 313.0
```

4.2

4.2

```python
# 1
def extract_every_other(lst):
    result = []
    for i in range(0, len(lst), 2):
        result.append(lst[i])
    return result

input_list = [1, 2, 3, 4, 5, 6]
output_list = extract_every_other(input_list)
print(output_list)
```

```
[1, 3, 5]
```

```python
# 2
def get_sublist(lst, start, end):
    result = []
    for i in range(start, end):
        result.append(lst[i])
    return result

input_list = [1, 2, 3, 4, 5, 6]
sublist = get_sublist(input_list, 2, 5)
print(sublist)
```

```
[3, 4, 5]
```

```python
# 3
def reverse_list(lst):
    return lst[::-1]

input_list = [1, 2, 3, 4, 5]
reversed_list = reverse_list(input_list)
# print(reversed_list)
```

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

```python
# 4
def remove_first_last(lst):
    return lst[1:-1]

input_list = [1, 2, 3, 4, 5, 6]
result_list = remove_first_last(input_list)
print(result_list)
```

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

```python
# 5
def get_first_n(lst, n):
    return lst[:n]

input_list = [1, 2, 3, 4, 5, 6, 7]
n = 4
result_list = get_first_n(input_list, n)
print(result_list)
```

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

```python
# 6
def get_last_n(lst, n):
    return lst[-n:]

input_list = [1, 2, 3, 4, 5]
n = 2
result_list = get_last_n(input_list, n)
print(result_list)
```

```
[4, 5]
```

```python
# 7
def reverse_skip(lst):
    result = []
    for i in range(len(lst) - 2, -1, -2):
        result.append(lst[i])
    return result

input_list = [1, 2, 3, 4, 5, 6]
result_list = reverse_skip(input_list)
print(result_list)
```

```
[5, 3, 1]
```

4.3

```python
# 1
def flatten(lst):
    result = []
    for sublist in lst:
        for item in sublist:
            result.append(item)
    return result

input_list = [[1, 2], [3, 4], [5]]
flattened_list = flatten(input_list)
print(flattened_list)
```

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

```python
# 2
def access_nested_element(lst, indices):
    element = lst
    for index in indices:
        element = element[index]
    return element

lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
indices = [1, 2]
result = access_nested_element(lst, indices)
print(result)
```

```
6
```

```python
# 3
def sum_nested(lst):
    total = 0
    for element in lst:
        if isinstance(element, list):
            total += sum_nested(element)
        else:
            total += element
    return total

input_list = [[1, 2], [3, [4, 5]], 6]
result = sum_nested(input_list)
print(result)
```

```
21
```

```python
# 4
def remove_element(lst, elem):
    for i in range(len(lst)):
        if isinstance(lst[i], list):
            lst[i] = remove_element(lst[i], elem)
        elif elem == lst[i]:
            lst.pop(i)
            return remove_element(lst, elem)
    return lst

input_list = [[1, 2], [3, 2], [4, 5]]
elem_to_remove = 2
result = remove_element(input_list, elem_to_remove)
print(result)
```

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

```python
# 5
def find_max(lst):
    max_elem = 0

    for element in lst:
        if isinstance(element, list):
            max_elem = max(max_elem, find_max(element))
        else:
            max_elem = max(max_elem, element)

    return max_elem

input_list = [[1, 2], [3, [4, 5]], 6]
result = find_max(input_list)
print(result)
```

```
6
```

```python
# 6
def count_occurrences(lst, elem):
    count = 0

    for item in lst:
        if isinstance(item, list):
            count += count_occurrences(item, elem)
        elif item == elem:
            count += 1

    return count

input_list = [[1, 2], [2, 3], [2, 4]]
elem_to_count = 2
result = count_occurrences(input_list, elem_to_count)
print(result)
```

```
3
```

```python
# 7
def deep_flatten(lst):
    flat_list = []

    for item in lst:
        if isinstance(item, list):
            flat_list.extend(deep_flatten(item))
        else:
            flat_list.append(item)

    return flat_list

input_list = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
result = deep_flatten(input_list)
print(result)
```

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

10.1

Problem 1

```python
import numpy as np

# Task 1: Initialize an empty array with size 2x2
empty_array = np.empty((2, 2))
print("Empty array (2x2):")
print(empty_array)

# Task 2: Initialize an all one array with size 4x2
ones_array = np.ones((4, 2))
print("\nArray of ones (4x2):")
print(ones_array)

# Task 3: Return a new array of given shape and type, filled with fill value
shape = (3, 3)
fill_value = 7
filled_array = np.full(shape, fill_value)
print("\nArray filled with value 7 (3x3):")
print(filled_array)

# Task 4: Return a new array of zeros with same shape and type as a given array
existing_array = np.array([1, 2, 3, 4])  # Existing array
zeros_array = np.zeros_like(existing_array)
print("\nArray of zeros with the same shape and type as existing array:")
print(zeros_array)

# Task 5: Return a new array of ones with same shape and type as a given array
ones_like_array = np.ones_like(existing_array)
print("\nArray of ones with the same shape and type as existing array:")
print(ones_like_array)

# Task 6: Convert an existing list to a NumPy array
new_list = [1, 2, 3, 4]
array_from_list = np.array(new_list)
print("\nNumPy array from list [1, 2, 3, 4]:")
print(array_from_list)
```

```
Empty array (2x2):
[[1.1852152e-316 0.0000000e+000]
 [0.0000000e+000 0.0000000e+000]]

Array of ones (4x2):
[[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]

Array filled with value 7 (3x3):
[[7 7 7]
 [7 7 7]
 [7 7 7]]

Array of zeros with the same shape and type as existing array:
[0 0 0 0]

Array of ones with the same shape and type as existing array:
[1 1 1 1]

NumPy array from list [1, 2, 3, 4]:
[1 2 3 4]
```

Problem 2

Task 1

Problem - 2

```
[ ] import numpy as np

    # Task 1: Create an array with values ranging from 10 to 49
    array_10_to_49 = np.arange(10, 50)
    print("Array from 10 to 49:")
    print(array_10_to_49)
```

```
Array from 10 to 49:
[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]
```

Task 2

```
# Task 2: Create a 3x3 matrix with values ranging from 0 to 8
matrix_3x3 = np.arange(9).reshape(3, 3)
print("\n3x3 matrix with values from 0 to 8:")
print(matrix_3x3)
```

```

3x3 matrix with values from 0 to 8:
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Task 3:

```python
# Task 3: Create a 3x3 identity matrix
identity_matrix = np.eye(3)
print("\n3x3 identity matrix:")
print(identity_matrix)
```

```
3x3 identity matrix:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Task 4

```python
# Task 4: Create a random array of size 30 and find the mean of the array
random_array = np.random.random(30)
mean_value = random_array.mean()
print("\nRandom array of size 30:")
print(random_array)
print("Mean of the random array:", mean_value)
```

```
Random array of size 30:
[0.83724189 0.65379884 0.34089264 0.79832932 0.78620985 0.66649242
 0.94624002 0.21671648 0.77026657 0.76055251 0.33432819 0.07094991
 0.83209743 0.20562835 0.9515489  0.04185155 0.19788873 0.06550579
 0.04921626 0.28712883 0.8212101  0.50902998 0.12035884 0.11620666
 0.0415786  0.08900972 0.29210989 0.02733537 0.51819799 0.0826557 ]
Mean of the random array: 0.4143525785663581
```

task 5

Task 5:

```python
# Task 5: Create a 10x10 array with random values and find the minimum and maximum values
random_10x10 = np.random.random((10, 10))
min_value = random_10x10.min()
max_value = random_10x10.max()
print("\n10x10 array with random values:")
print(random_10x10)
print("Minimum value:", min_value)
print("Maximum value:", max_value)
```

```
10x10 array with random values:
[[0.57128628 0.64228644 0.29992287 0.26059832 0.22946564 0.29647417
  0.76106263 0.56664688 0.20507711 0.07406452]
 [0.60542192 0.48284518 0.79960232 0.13541297 0.51506407 0.03516513
  0.78644821 0.66092392 0.11513596 0.50695793]
 [0.6647865  0.0383398  0.76094013 0.682416   0.39908772 0.86929446
  0.19299116 0.621036   0.32502597 0.26593117]
 [0.55990848 0.67392481 0.58398366 0.54988176 0.15736292 0.64400683
  0.46532021 0.24723126 0.64320698 0.45084883]
 [0.54589803 0.59652161 0.81972048 0.93265606 0.40190245 0.30182208
  0.99464356 0.86800134 0.20517459 0.15385105]
 [0.16748441 0.32473912 0.92590183 0.80552748 0.12834501 0.14789843
  0.88053895 0.6524658  0.00498678 0.76262253]
 [0.85613018 0.25050798 0.52185202 0.50863213 0.91803788 0.6661611
  0.34145724 0.52727705 0.15387918 0.00105594]
 [0.99229812 0.86610469 0.02529823 0.19835367 0.3987422  0.55326664
  0.87457389 0.69123088 0.8445907  0.60001992]
 [0.0353522  0.4774439  0.36379623 0.60333963 0.75285594 0.73284567
  0.81201222 0.72134571 0.44388602 0.462701  ]
 [0.42854889 0.95142339 0.92689905 0.44257375 0.82446602 0.88783324
  0.74257883 0.86235696 0.29085699 0.49580722]]
Minimum value: 0.0010559357695225646
Maximum value: 0.9946435628877366
```

Task 6:

```python
# Task 6: Create a zero array of size 10 and replace 5th element with 1
zero_array = np.zeros(10)
zero_array[4] = 1
print("\nZero array with 5th element replaced with 1:")
print(zero_array)
```

```
Zero array with 5th element replaced with 1:
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

Task 7:

```
# Task 7: Reverse an array arr = [1, 2, 0, 0, 4, 0]
arr = [1, 2, 0, 0, 4, 0]
reversed_arr = arr[::-1]
print("\nReversed array:")
print(reversed_arr)
```

```
Reversed array:
[0, 4, 0, 0, 2, 1]
```

Task 8

```
# Task 8: Create a 2D array with 1 on the border and 0 inside
border_array = np.ones((5, 5))
border_array[1:-1, 1:-1] = 0
print("\n2D array with 1 on the border and 0 inside:")
print(border_array)
```

```
2D array with 1 on the border and 0 inside:
[[1. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 1. 1. 1. 1.]]
```

Task 9

```
# Task 9: Create an 8x8 matrix and fill it with a checkerboard pattern
checkerboard = np.zeros((8, 8), dtype=int)
checkerboard[1::2, ::2] = 1
checkerboard[::2, 1::2] = 1
print("\n8x8 checkerboard pattern:")
print(checkerboard)
```

```
8x8 checkerboard pattern:
[[0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0]]
```

Problem 3:

Task 1:

```
[ ] import numpy as np

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

    # Task 1: Add the two arrays
    add_arrays = x + y
    print("Task 1: Sum of x and y:")
    print(add_arrays)
```

```
Task 1: Sum of x and y:
[[ 6  8]
 [10 13]]
```

Task 2

```
# Task 2: Subtract the two arrays
subtract_arrays = x - y
print("\nTask 2: Difference of x and y:")
print(subtract_arrays)
```

```
Task 2: Difference of x and y:
[[-4 -4]
 [-4 -3]]
```

Task 3

```
[ ] # Task 3: Multiply the array with any integers of your choice (let's multiply by 2)
    multiply_array = x * 2
    print("\nTask 3: x multiplied by 2:")
    print(multiply_array)
```

```
Task 3: x multiplied by 2:
[[ 2  4]
 [ 6 10]]
```

Task 4

```
# Task 4: Find the square of each element of the array
square_elements = np.square(x)
print("\nTask 4: Square of each element in x:")
print(square_elements)
```

```
Task 4: Square of each element in x:
[[ 1  4]
 [ 9 25]]
```

Task 5:

```
# Task 5: Find the dot product between v and w, x and v, x and y
dot_v_w = np.dot(v, w)
dot_x_v = np.dot(x, v)
dot_x_y = np.dot(x, y)

print("\nTask 5: Dot Products")
print("Dot product of v and w:", dot_v_w)
print("Dot product of x and v:", dot_x_v)
print("Dot product of x and y:", dot_x_y)
```

```
Task 5: Dot Products
Dot product of v and w: 219
Dot product of x and v: [29 77]
Dot product of x and y: [[19 22]
 [50 58]]
```

Task 6

```python
# Task 6: Concatenate x and y along rows and concatenate v and w along columns
concat_x_y_row = np.concatenate((x, y), axis=0)
concat_v_w_col = np.concatenate((v.reshape(-1, 1), w.reshape(-1, 1)), axis=1)

print("\nTask 6: Concatenation")
print("x and y concatenated along rows:")
print(concat_x_y_row)
print("v and w concatenated along columns:")
print(concat_v_w_col)
```

```
Task 6: Concatenation
x and y concatenated along rows:
[[1 2]
 [3 5]
 [5 6]
 [7 8]]
v and w concatenated along columns:
[[ 9 11]
 [10 12]]
```

Task 7

```python
# Task 7: Concatenate x and v; if you get an error, observe and explain why
try:
    concat_x_v = np.concatenate((x, v), axis=0)
    print("\nTask 7: Concatenate x and v:")
    print(concat_x_v)
except ValueError as e:
    print("\nTask 7 Error:", e)
```

```
Task 7 Error: 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)
```

Problem 4

Task 1

```python
# Given Matrices A and B
A = np.array([[3, 4], [7, 8]])
B = np.array([[5, 3], [2, 1]])

# Task 1: A * A^(-1) = I
A_inv = np.linalg.inv(A)
identity_matrix = np.dot(A, A_inv)
print("Task 1: A * A^(-1) = I:")
print(identity_matrix)
```

```
Task 1: A * A^(-1) = I:
[[1.00000000e+00 0.00000000e+00]
 [1.77635684e-15 1.00000000e+00]]
```

Task 2

```python
# Task 2: AB != BA
AB = np.dot(A, B)
BA = np.dot(B, A)
print("\nTask 2: AB and BA:")
print("AB:")
print(AB)
print("BA:")
print(BA)
are_equal = np.array_equal(AB, BA)
print("Are AB and BA equal?", are_equal)
```

```
Task 2: AB and BA:
AB:
[[23 13]
 [51 29]]
BA:
[[36 44]
 [13 16]]
Are AB and BA equal? False
```

Task 3

```python
# Task 3: (AB)^T = B^T A^T
AB_transpose = np.transpose(AB)
BT_AT = np.dot(np.transpose(B), np.transpose(A))
print("\nTask 3: (AB)^T and B^T A^T:")
print("Transpose of AB:")
print(AB_transpose)
print("B^T * A^T:")
print(BT_AT)
are_equal_transposes = np.array_equal(AB_transpose, BT_AT)
print("Are (AB)^T and B^T * A^T equal?", are_equal_transposes)
```

```
Task 3: (AB)^T and B^T A^T:
Transpose of AB:
[[23 51]
 [13 29]]
B^T * A^T:
[[23 51]
 [13 29]]
Are (AB)^T and B^T * A^T equal? True
```

Task 4:

```
# Task 4: Solve the system of Linear equations using Inverse Method
A_matrix = np.array([[2, -3, 1], [1, -1, 2], [3, 1, -1]])
B_vector = np.array([-1, -3, 9])

A_inv = np.linalg.inv(A_matrix)
X = np.dot(A_inv, B_vector)
print("\nTask 4: Solving Linear Equation System using Inverse Method:")
print("Solution (x, y, z):")
print(X)
```

```
Task 4: Solving Linear Equation System using Inverse Method:
Solution (x, y, z):
[ 2.  1. -2.]
```

Task 5:

```
# Task 5: Solve the system of Linear equations using np.linalg.inv()
X_solution = np.linalg.inv(A_matrix).dot(B_vector)
print("\nTask 5: Solve using np.linalg.inv:")
print("Solution (x, y, z):")
print(X_solution)
```

```
Task 5: Solve using np.linalg.inv:
Solution (x, y, z):
[ 2.  1. -2.]
```

10.2

Task 1:

```python
import time
import numpy as np

# Define the size
size = 1000000
matrix_size = 1000

# 1. Element-wise Addition

# Using Python lists
list1 = [i for i in range(size)]
list2 = [i for i in range(size)]

start_time = time.time()
list_add = [list1[i] + list2[i] for i in range(size)]
python_list_add_time = time.time() - start_time
print(f"Element-wise addition using Python lists took: {python_list_add_time:.6f} seconds.")

# Using Numpy arrays
np_array1 = np.arange(size)
np_array2 = np.arange(size)

start_time = time.time()
np_array_add = np_array1 + np_array2
numpy_array_add_time = time.time() - start_time
print(f"Element-wise addition using NumPy arrays took: {numpy_array_add_time:.6f} seconds.")
```

```
Element-wise addition using Python lists took: 0.117682 seconds.
Element-wise addition using NumPy arrays took: 0.001897 seconds.
```

Task 2:

```python
# 2. Element-wise Multiplication

# Using Python lists
start_time = time.time()
list_mul = [list1[i] * list2[i] for i in range(size)]
python_list_mul_time = time.time() - start_time
print(f"Element-wise multiplication using Python lists took: {python_list_mul_time:.6f} seconds.")

# Using Numpy arrays
start_time = time.time()
np_array_mul = np_array1 * np_array2
numpy_array_mul_time = time.time() - start_time
print(f"Element-wise multiplication using NumPy arrays took: {numpy_array_mul_time:.6f} seconds.")
```

```
Element-wise multiplication using Python lists took: 0.174980 seconds.
Element-wise multiplication using NumPy arrays took: 0.007491 seconds.
```

Task 3:

```python
# 3. Dot Product

# Using Python lists
start_time = time.time()
dot_product_python = sum(list1[i] * list2[i] for i in range(size))
python_dot_product_time = time.time() - start_time
print(f"Dot product using Python lists took: {python_dot_product_time:.6f} seconds.")

# Using Numpy arrays
start_time = time.time()
np_dot_product = np.dot(np_array1, np_array2)
numpy_dot_product_time = time.time() - start_time
print(f"Dot product using NumPy arrays took: {numpy_dot_product_time:.6f} seconds.")
```

```
Dot product using Python lists took: 0.059879 seconds.
Dot product using NumPy arrays took: 0.000973 seconds.
```

Task 4:

```python
# 4. Matrix Multiplication

# Using Python lists
matrix1 = [[i+j for j in range(matrix_size)] for i in range(matrix_size)]
matrix2 = [[i-j for j in range(matrix_size)] for i in range(matrix_size)]

start_time = time.time()
matrix_mul_python = [[sum(matrix1[i][k] * matrix2[k][j] for k in range(matrix_size)) for j in range(matrix_size)] for i in range(matrix_size)]
python_matrix_mul_time = time.time() - start_time
print(f"Matrix multiplication using Python lists took: {python_matrix_mul_time:.6f} seconds.")

# Using Numpy arrays
np_matrix1 = np.random.rand(matrix_size, matrix_size)
np_matrix2 = np.random.rand(matrix_size, matrix_size)

start_time = time.time()
matrix_mul_numpy = np.dot(np_matrix1, np_matrix2)
numpy_matrix_mul_time = time.time() - start_time
print(f"Matrix multiplication using NumPy arrays took: {numpy_matrix_mul_time:.6f} seconds.")
```

```
Matrix multiplication using Python lists took: 131.329796 seconds.
Matrix multiplication using NumPy arrays took: 0.047067 seconds.
```