Create a NumPy array from a Python list:

```
import numpy as np

my_list = [1, 2, 3, 4, 5]

numpy_array = np.array(my_list)
```

Find the shape of a NumPy array:

python

Copy code

```
print(numpy_array.shape)
```

Perform element-wise addition of two NumPy arrays:

```
array1 = np.array([1, 2, 3])

array2 = np.array([4, 5, 6])

result = array1 + array2
```

Calculate the mean of a NumPy array:

```
mean_value = np.mean(numpy_array)
```

Calculate the median of a NumPy array:

python

Copy code

```
median_value = np.median(numpy_array)
```

Find the maximum and minimum values in a NumPy array:

```
max_value = np.max(numpy_array)

min_value = np.min(numpy_array)
```

Concatenate two NumPy arrays horizontally and vertically:

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

array2 = np.array([[5, 6], [7, 8]])


horizontal_concat = np.hstack((array1, array2))

vertical_concat = np.vstack((array1, array2))
```

Calculate the dot product of two NumPy arrays:

```
dot_product = np.dot(array1, array2)
```

Find the unique elements and their counts in a NumPy array:

```
unique_elements, counts = np.unique(numpy_array, return_counts=True)
```

Create a NumPy array with elements 1 to 10:

```
array_1_to_10 = np.arange(1, 11)
```

Create a 3x3 identity matrix using NumPy:

```
identity_matrix = np.eye(3)
```

Create a NumPy array with a specified upper and lower limit:

```
array_limit = np.linspace(1, 10, 10)
```

Calculate the sum of all elements in a NumPy array:

```
sum_value = np.sum(numpy_array)
```

Replace all even numbers in a NumPy array with 0:

```
numpy_array[numpy_array % 2 == 0] = 0
```

Convert a NumPy array to a Python list:

```
python_list = numpy_array.tolist()
```

Calculate the inverse of a square NumPy matrix:

```
square_matrix = np.array([[1, 2], [3, 4]])

inverse_matrix = np.linalg.inv(square_matrix)
```

Remove all NaN values from a NumPy array:

```
numpy_array = numpy_array[~np.isnan(numpy_array)]
```

Perform element-wise subtraction of two NumPy arrays:

```
result_subtraction = array1 - array2
```

Perform element-wise division of two NumPy arrays:

```
result_division = array1 / array2
```

Find the indices of the minimum and maximum values in a NumPy array:

```
min_index = np.argmin(numpy_array)

max_index = np.argmax(numpy_array)
```

Check if two NumPy arrays are equal:

```
are_equal = np.array_equal(array1, array2)
```

Extract specific rows and columns from a NumPy array:

```
subset = numpy_array[1:3, 1:3]  # Extracts rows 1 and 2, columns 1 and 2
```

Sort a NumPy array in ascending order:

```
sorted_array_asc = np.sort(numpy_array)
```

Sort a NumPy array in descending order:

```
sorted_array_desc = np.sort(numpy_array)[::-1]
```

Round the elements of a NumPy array to the nearest integer:

```
rounded_array = np.round(numpy_array)
```

Check if any element in a NumPy array is NaN:

```
has_nan = np.isnan(numpy_array).any()
```

Create a NumPy array and print its size, and data type:

```
array = np.array([1, 2, 3])

print("Size:", array.size)

print("Data type:", array.dtype)
```

Write a Python program to print a pyramid pattern:

```
rows = 5

for i in range(1, rows+1):

    print(" "*(rows-i) + "*"*(2*i-1))
```

Create a Python program to print a diamond pattern with a given number of rows:

```
rows = 5

for i in range(1, rows+1):

    print(" "*(rows-i) + "*"*(2*i-1))

for i in range(rows-1, 0, -1):

    print(" "*(rows-i) + "*"*(2*i-1))
```

Write a Python program to print a pyramid pattern with numbers:

```
rows = 5

num = 1

for i in range(1, rows+1):

    for j in range(1, i+1):

        print(num, end=" ")

        num += 1

    print()
```

Create a Python program to check if a given number is an Adam number:

```
num = 121

rev = int(str(num)[::-1])

square_num = num ** 2

square_rev = rev ** 2

if square_rev == int(str(square_num)[::-1]):

    print(f"{num} is an Adam number.")

else:
    print(f"{num} is not an Adam number.")
```

Write a Python program that checks if a given number is an automorphic number:

```
num = 76
square_num = num ** 2
if str(num) == str(square_num)[-len(str(num)):]:
    print(f"{num} is an automorphic number.")
else:
    print(f"{num} is not an automorphic number.")
```

Develop a Python program to find and display all perfect numbers within a given range:

```
for num in range(1, 101):
    sum_factors = sum([i for i in range(1, num) if num % i == 0])
    if sum_factors == num:
        print(num)
```

Create a Python program to determine if a given number is a happy number:

```
def is_happy_number(n):
    seen = set()
    while n != 1 and n not in seen:
        seen.add(n)
        n = sum(int(digit)**2 for digit in str(n))
    return n == 1


for num in range(1, 101):
    if is_happy_number(num):
        print(f"{num} is a happy number.")
```

Write a Python program that checks if a given number is an Armstrong number:

```
num = 153
sum_cubes = sum(int(digit)**3 for digit in str(num))
if sum_cubes == num:
    print(f"{num} is an Armstrong number.")
else:
    print(f"{num} is not an Armstrong number.")
```

Employee Management System:

```
from abc import ABC, abstractmethod


class Employee(ABC):
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @abstractmethod
    def calculate_salary(self):
        pass

    @abstractmethod
    def display_info(self):
        pass

class Manager(Employee):
    def calculate_salary(self):
        return 5000 + (self.age * 100)

    def display_info(self):
        print(f"Manager: {self.name}, Age: {self.age}")

class Developer(Employee):
    def calculate_salary(self):
        return 4000 + (self.age * 80)

    def display_info(self):
        print(f"Developer: {self.name}, Age: {self.age}")

class Designer(Employee):
    def calculate_salary(self):
        return 4500 + (self.age * 90)
    def display_info(self):
        print(f"Designer: {self.name}, Age: {self.age}")
```

Banking Operations System:

```
from abc import ABC, abstractmethod


class BankAccount(ABC):
    def __init__(self, balance=0):
        self.balance = balance

    @abstractmethod
    def deposit(self, amount):
        pass

    @abstractmethod
    def withdraw(self, amount):
        pass

class SavingsAccount(BankAccount):
    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient funds.")

class CheckingAccount(BankAccount):
    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
        else:
            print("Insufficient funds.")
```

Text Processing Tool:

```
from abc import ABC, abstractmethod

class TextProcessor(ABC):
    @abstractmethod
    def format_text(self, text):
        pass

    @abstractmethod
    def analyze_text(self, text):
        pass

class UpperCaseFormatter(TextProcessor):
    def format_text(self, text):
        return text.upper()

    def analyze_text(self, text):
        return f"Number of characters: {len(text)}"

class LowerCaseFormatter(TextProcessor):
    def format_text(self, text):
        return text.lower()

    def analyze_text(self, text):
        return f"Number of words: {len(text.split())}"
```

Geometric Shapes:

```
from abc import ABC, abstractmethod
import math

class Shape(ABC):
    @abstractmethod
    def calculate_area(self):
        pass

    @abstractmethod
    def calculate_perimeter(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return math.pi * self.radius**2

    def calculate_perimeter(self):
        return 2 * math.pi * self.radius

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def calculate_area(self):
        return self.width * self.height

    def calculate_perimeter(self):
        return 2 * (self.width + self.height)
```

Online Shopping System:

```
class Product(ABC):
    def __init__(self, name, price):
        self.name = name
        self.price = price

    @abstractmethod
    def calculate_shipping_cost(self):
        pass

    def get_details(self):
        return f"Product: {self.name}, Price: {self.price}"

class Electronics(Product):
    def calculate_shipping_cost(self):
        return 50

class Clothing(Product):
    def calculate_shipping_cost(self):
        return 20

class Books(Product):
    def calculate_shipping_cost(self):
        return 10
```

Factorial Using Recursion:

```
def factorial_recursive(n):
    if n == 0:
        return 1
    return n * factorial_recursive(n-1)
```

Factorial Without Recursion:

python

Copy code

```
def factorial_iterative(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result
```

Check Prime Number:

python

Copy code

```
def is_prime(num):
    if num <= 1:
        return False
    for i in range(2, int(math.sqrt(num))+1):
        if num % i == 0:
            return False
    return True
```

Number Guessing Game:

```
import random


random_number = random.randint(1, 100)
attempts = 0


print("Guess the number between 1 and 100!")


while True:
    guess = int(input("Enter your guess: "))
    attempts += 1


    if guess < random_number:
        print("Too low!")
    elif guess > random_number:
        print("Too high!")
    else:
        print(f"Congratulations! You guessed it in {attempts} attempts.")
        break
```

Find Prime Numbers in Range:

```
def find_primes_in_range(start, end):
    primes = []
    for num in range(start, end+1):
        if is_prime(num):
            primes.append(num)
    return primes
```

Encrypt String:

```
def encrypt_string(text, shift):
    encrypted = ""
    for char in text:
        if char.isalpha():
            shifted = ord(char) + shift
            if char.islower():
                if shifted > ord('z'):
                    shifted -= 26
            else:
                if shifted > ord('Z'):
                    shifted -= 26
            encrypted += chr(shifted)
        else:
            encrypted += char
    return encrypted
```

List Operations:

python

Copy code

```
def list_operations(lst):
    lst.append(10)
    lst = list(set(lst))
    lst.sort()
    return lst
```

Count Words in Sentence:

```
def count_words(sentence):
    return len(sentence.split())
```

Filter Even and Odd Numbers:

```
def filter_even_odd(numbers):
    evens = [num for num in numbers if num % 2 == 0]
    odds = [num for num in numbers if num % 2 != 0]
    return evens, odds
```

Tuple Operations:

```
def tuple_operations(t1, t2):
    concatenated = t1 + t2
    indexed = concatenated[3]
    sliced = concatenated[2:5]
    return concatenated, indexed, sliced
```

Read CSV and Display:

```
import pandas as pd


df = pd.read_csv('file.csv')
print(df.iloc[:, 1])
df.iloc[:, 2] += 10
```

Filter and Multiply CSV Columns:

```
df = pd.read_csv('file.csv')
filtered_df = df[df['column1'] > 50]
filtered_df['column2'] *= 1.5
```

Reorder CSV Columns and Display:

```
df = pd.read_csv('file.csv')
new_order = ['column3', 'column1', 'column2']
df = df[new_order]
print(df.head())
```

Calculate New Column from CSV Columns:

```
df = pd.read_csv('file.csv')
df['new_column'] = df['column1'] + df['column2']
```

Handle Missing Values in CSV:

```
df = pd.read_csv('file.csv')
df = df.dropna(subset=['specific_column'])
df['another_column'].fillna(df['another_column'].mean(), inplace=True)
```

Scatter Plot Using Pandas and Matplotlib:

```
import pandas as pd
import matplotlib.pyplot as plt


df = pd.read_csv('file.csv')
```

```
plt.scatter(df['column1'], df['column2'])
plt.xlabel('Column 1')
plt.ylabel('Column 2')
plt.show()
```

Bar Chart for Categorical Data Using Pandas:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('file.csv')
df['category_column'].value_counts().plot(kind='bar')
plt.xlabel('Categories')
plt.ylabel('Counts')
plt.show()
```

Make sure to replace 'file.csv', 'column1', 'column2', 'column3', 'specific_column', 'another_column', and 'category_column' with actual file names and column names as per your dataset.

CSV Data Preprocessing with Pandas and Visualization with Matplotlib:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Read CSV file
df = pd.read_csv('data.csv')

# Data preprocessing with numpy
df['new_column'] = np.log(df['old_column'])

# Scatter plot
plt.scatter(df['column1'], df['column2'], c=df['new_column'], cmap='viridis')
plt.xlabel('Column 1')
plt.ylabel('Column 2')
plt.title('Scatter Plot with Data Transformation')
plt.colorbar(label='New Column')
plt.show()
```

Calculate Greatest Common Divisor (GCD):

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a
```

Calculate Least Common Multiple (LCM):

```
def lcm(a, b):
    return abs(a*b) // gcd(a, b)
```

Add Two Matrices:

```
import numpy as np


def add_matrices(matrix1, matrix2):
    return np.add(matrix1, matrix2)
```

Find Hypotenuse of Right Triangle:

```
def hypotenuse(a, b):
    return (a**2 + b**2)**0.5
```

Calculate Volume of a Cube:

```
def volume_cube(side):
    return side**3
```

Convert Decimal to Binary:

```
def decimal_to_binary(n):
    return bin(n)[2:]
```

Convert Binary to Decimal:

```
def binary_to_decimal(binary):
    return int(binary, 2)
```

Calculate Average of List:

```
def average(numbers):
    return sum(numbers) / len(numbers)
```

Sum of Alternate Even Numbers:

```
def sum_alternate_even(n):
    return sum(range(2, n+1, 4))
```

Sum of Alternate Odd Numbers:

```
def sum_alternate_odd(n):
    return sum(range(1, n+1, 4))
```

Convert Celsius to Fahrenheit:

```
def celsius_to_fahrenheit(c):
    return (c * 9/5) + 32
```

Convert Fahrenheit to Celsius:

```
def fahrenheit_to_celsius(f):
    return (f - 32) * 5/9
```

Calculate Area of Parallelogram:

```
def area_parallelogram(base, height):
    return base * height
```

Read Text File Without Newline:

```
with open('file.txt', 'r') as file:
    content = file.read().replace('\n', '')
print(content)
```

Calculate Volume of Rectangular Prism:

```
def volume_rectangular_prism(length, width, height):
    return length * width * height
```

Find Roots of Quadratic Equation:

```
def quadratic_roots(a, b, c):
    d = (b**2) - (4*a*c)
    root1 = (-b + d**0.5) / (2*a)
    root2 = (-b - d**0.5) / (2*a)
    return root1, root2
```

Reverse a List Without Built-in Functions:

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

Check Anagrams:

```
def is_anagram(str1, str2):
    return sorted(str1) == sorted(str2)
```

Find Longest Word in Sentence:

```
def longest_word(sentence):
    words = sentence.split()
    return max(words, key=len)
```

Sort a List Using Bubble Sort:

```
def bubble_sort(lst):
    n = len(lst)
    for i in range(n):
        for j in range(0, n-i-1):
            if lst[j] > lst[j+1]:
                lst[j], lst[j+1] = lst[j+1], lst[j]
    return lst
```

Remember to replace 'data.csv', 'file.txt', and 'old_column' with actual file names and column names as per your dataset or file.

Copy Contents of One Text File to Another:

```
with open('source.txt', 'r') as source_file:
    with open('destination.txt', 'w') as destination_file:
        destination_file.write(source_file.read())
```

Count Occurrences of Specific Character in Text File:

```
with open('file.txt', 'r') as file:
    content = file.read()
    count = content.count('a')  # Count occurrences of 'a'
print(count)
```

Count Upper and Lower Case Letters:

```
def count_case(sentence):
    upper_count = sum(1 for char in sentence if char.isupper())
    lower_count = sum(1 for char in sentence if char.islower())
    return upper_count, lower_count
```

Concatenate Strings to .txt File:

```
def concatenate_to_txt(str1, str2):
    with open('output.txt', 'w') as file:
        file.write(str1 + str2)
```

Check Leap Year:

```
def is_leap_year(year):
    if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):
        return True
    else:
        return False
```

Date Difference Calculator:

```
from datetime import datetime


def date_difference(date1, date2):
    date_format = "%Y-%m-%d"
    diff = datetime.strptime(date2, date_format) - datetime.strptime(date1, date_format)
    return diff.days
```

Convert Kilometers to Miles:

```
def km_to_miles(km):
    return km / 1.60934
```

Convert Miles to Kilometers:

```
def miles_to_km(miles):
    return miles * 1.60934
```

Calculate Area of Triangle:

```
def area_triangle(base, height):
    return 0.5 * base * height
```

Rock, Paper, Scissors:

```
import random


def rock_paper_scissors(player_choice):
    choices = ['rock', 'paper', 'scissors']
    computer_choice = random.choice(choices)

    if player_choice == computer_choice:
        return 'Tie!'

    if (player_choice == 'rock' and computer_choice == 'scissors') or \
       (player_choice == 'scissors' and computer_choice == 'paper') or \
       (player_choice == 'paper' and computer_choice == 'rock'):
        return 'You win!'
    else:
        return 'You lose!'
```

Error Handling:

```
try:
    # Some code that might raise an error
except ExceptionType:
    # Handle the error
```

Generate All Possible Strings:

```
from itertools import permutations


perms = [''.join(p) for p in permutations('aeioI')]
```

Add Two Positive Integers Without '+' Operator:

```
def add_without_plus(a, b):
    while b != 0:
        carry = a & b
        a = a ^ b
        b = carry << 1
    return a
```

Mean, Median, Mode:

```
import statistics


def calculate_stats(numbers):
    mean = statistics.mean(numbers)
    median = statistics.median(numbers)
    mode = statistics.mode(numbers)
    return mean, median, mode
```

Calculate Simple Interest:

```
def simple_interest(principal, rate, time):
    return (principal * rate * time) / 100
```

Calculate Compound Interest:

```
def compound_interest(principal, rate, time):
    return principal * (pow((1 + rate / 100), time))
```

Replace "Python" with "Java" and Vice Versa:

```
def replace_words(sentence):

    sentence = sentence.replace('Python', 'temp').replace('Java', 'Python').replace('temp', 'Java')

    return sentence
```

Multiplication Table:

```
def multiplication_table(num):

    for i in range(1, 11):

        print(f"{num} x {i} = {num*i}")
```

Generate Fibonacci Series:

```
def fibonacci(n):

    fib_series = [0, 1]

    while len(fib_series) < n:

        fib_series.append(fib_series[-1] + fib_series[-2])

    return fib_series[:n]
```

Array Rotation:

```
def array_rotation(arr, k):

    n = len(arr)

    k = k % n

    return arr[-k:] + arr[:-k]
```

Replace 'source.txt', 'destination.txt', 'file.txt', and 'output.txt' with your file names as needed.

Swap Two Elements in a List:

```
def swap_elements(lst, idx1, idx2):

    lst[idx1], lst[idx2] = lst[idx2], lst[idx1]
```

Check if Element Exists in List:

```
def element_exists(lst, element):

    return element in lst
```

Calculate Square Root:

```
import math


def square_root(num):

    return math.sqrt(num)
```

Find Maximum Value Between Two Numbers:

```
def max_value(num1, num2):
    return max(num1, num2)
```

Calculate Sine of an Angle in Degrees:

```
def sine_degrees(degrees):
    radians = math.radians(degrees)
    return math.sin(radians)
```

Find Floor Value of Floating-Point Number:

```
def floor_value(num):
    return math.floor(num)
```

Calculate Natural Logarithm:

```
def natural_log(num):
    return math.log(num)
```

Generate Random Number in Given Range:

```
import random


def random_number(start, end):
    return random.randint(start, end)
```

Calculate Absolute Value:

```
def absolute_value(num):
    return abs(num)
```

Calculate Cosine of an Angle in Degrees:

```
def cosine_degrees(degrees):
    radians = math.radians(degrees)
    return math.cos(radians)
```

Round Floating-Point Number to Nearest Integer:

```
def round_to_nearest_integer(num):
    return round(num)
```

Calculate Power of a Number:

```
def power(base, exponent):
    return base ** exponent
```

Calculate Tangent of an Angle in Degrees:

```
def tangent_degrees(degrees):
    radians = math.radians(degrees)
    return math.tan(radians)
```

These functions can be used by passing the required parameters to get the desired results.

Find Ceiling Value of Floating-Point Number:

```
def ceiling_value(num):
    return math.ceil(num)
```

Calculate Exponential Value:

```
def exponential_value(num):
    return math.exp(num)
```

Calculate Hyperbolic Sine:

```
def hyperbolic_sine(num):
    return math.sinh(num)
```

Calculate Logarithm with Given Base:

```
def logarithm(base, num):
    return math.log(num, base)
```

Calculate Hyperbolic Cosine:

```
def hyperbolic_cosine(num):
    return math.cosh(num)
```

Calculate Hyperbolic Tangent:

```
def hyperbolic_tangent(num):
    return math.tanh(num)
```

Calculate Arc Sine:

```
def arc_sine(num):
    return math.asin(num)
```

Calculate Arc Cosine:

```
def arc_cosine(num):
    return math.acos(num)
```

Check for Palindrome:

```
def is_palindrome(s):
    s = s.lower()
    return s == s[::-1]
```

Calculate Power Using Recursion:

```
def power_recursive(base, exponent):
    if exponent == 0:
        return 1
    elif exponent == 1:
        return base
    else:
        return base * power_recursive(base, exponent-1)
```

Check for Perfect Number:

```
def is_perfect_number(num):
    divisors = [i for i in range(1, num) if num % i == 0]
    return sum(divisors) == num
```

Create 2D Matrix:

```
def create_matrix(rows, cols):
    return [[0 for _ in range(cols)] for _ in range(rows)]
```

Matrix Addition:

```
def matrix_addition(matrix1, matrix2):
    return [[matrix1[i][j] + matrix2[i][j] for j in range(len(matrix1[0]))] for i in range(len(matrix1))]
```

Create an Array Without Using Numpy or Standard Data Types:

```
def create_array(size):
    return [0] * size
```

Reverse a String Without Using Built-in Functions:

```
def reverse_string(s):
    return s[::-1]
```

Concatenate Strings Without Using + Operator:

```
def concat_strings(s1, s2):
    return "".join([s1, s2])
```

Capitalise First Letter of Each Word:

```
def capitalize_words(sentence):
    return " ".join([word.capitalize() for word in sentence.split()])
```

Reverse Words in a Sentence:

```
def reverse_words(sentence):
    return " ".join(sentence.split()[::-1])
```

Concatenate Two Strings:

```
def concatenate_strings(s1, s2):
    return s1 + s2
```

Simple Calculator:

```
def simple_calculator(num1, num2, operator):
    if operator == '+':
        return num1 + num2
    elif operator == '-':
        return num1 - num2
    elif operator == '*':
        return num1 * num2
    elif operator == '/':
        if num2 != 0:
            return num1 / num2
        else:
            return "Error: Division by zero!"
```

Generate Numpy Array and Plot Histogram:

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data = np.random.rand(100)
sns.histplot(data, kde=True)
plt.show()
```

Calculate Letter Grade:

```
def calculate_grade(score):
    if score >= 90:
        return 'A'
    elif score >= 80:
        return 'B'
    elif score >= 70:
        return 'C'
    elif score >= 60:
        return 'D'
    else:
        return 'F'
```

Create 2D Numpy Array and Calculate Transpose:

```
import numpy as np

arr = np.random.rand(5, 3)
transpose_arr = arr.T
```

Calculate Exponential of Each Element and Plot Scatter Plot:

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

arr = np.random.rand(100)
exp_values = np.exp(arr)

sns.scatterplot(x=range(100), y=exp_values)
plt.show()
```

Generate Random Numbers and Plot Boxplot:

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data = np.random.uniform(-10, 10, 1000)
sns.boxplot(data)
```

```
plt.show()
```

Create a NumPy Array and Print Elements from Index 5 to 15:

```
import numpy as np

arr = np.arange(1, 26)
print(arr[5:16])
```

Concatenate Two NumPy Arrays Along Axis 0 and Axis 1:

```
import numpy as np

arr1 = np.array([[1, 2, 3], [4, 5, 6]])
arr2 = np.array([[7, 8, 9], [10, 11, 12]])

concat_axis0 = np.concatenate((arr1, arr2), axis=0)
concat_axis1 = np.concatenate((arr1, arr2), axis=1)

print("Concatenated along axis 0:")
print(concat_axis0)

print("\nConcatenated along axis 1:")
print(concat_axis1)
```

Perform Element-wise Comparison Between Two NumPy Arrays:

```
import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([5, 4, 3, 2, 1])

greater_than = arr1 > arr2
less_than = arr1 < arr2

print("Element-wise Greater Than Comparison:")
print(greater_than)

print("\nElement-wise Less Than Comparison:")
print(less_than)
```

Create a 3D NumPy Array, Flatten it, and Print Flattened Array:

```
import numpy as np

arr = np.arange(1, 25).reshape(2, 3, 4)
flattened_arr = arr.flatten()

print("Original 3D Array:")
print(arr)

print("\nFlattened Array:")
print(flattened_arr)
```