```
# Function with no parameters and no return value
def greet():
    print("Hello, welcome to Python!")

# Function with multiple parameters
def add(a, b):
    return a + b

# Function with a variable number of parameters
def multiply(*args):
    result = 1
    for num in args:
        result *= num
    return result

# Function with both positional and keyword arguments
def describe_person(name, age, **additional_info):
    info = f"Name: {name}, Age: {age}"
    for key, value in additional_info.items():
        info += f", {key}: {value}"
    return info

# Example usages
greet()
print(add(5, 3))        # 8
print(multiply(2, 3, 4))  # 24
print(describe_person("Eva", 29, location="Vancouver", profession="Trainer"))
```

```
Hello, welcome to Python!
8
24
Name: Eva, Age: 29, location: Vancouver, profession: Trainer
```

```
# Global variable
x = 10

def outer_function():
    # Local variable
    y = 5

    def inner_function():
        # Accessing both global and local variables
        print(f"Global x: {x}, Local y: {y}")

    inner_function()

outer_function()

# The following would raise an error because y is not accessible outside the function
# print(y)
```

```
Global x: 10, Local y: 5
```

```
# Function with default parameter values
def greet_user(name="Guest", message="Welcome!"):
    print(f"Hello {name}, {message}")

# Example usages
greet_user()               # Uses default values
greet_user("Eva")          # Overrides only 'name'
greet_user("Eva", "Good morning!")  # Overrides both parameters
```

```
Hello Guest, Welcome!
Hello Eva, Welcome!
Hello Eva, Good morning!
```

```
# Recursive function to calculate factorial
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Example usage
print(factorial(5))  # 120
```

```
120
```

```
def add_numbers(a, b):
    """
    This function takes two numbers as input and returns their sum.

    Parameters:
    a (int or float): The first number
    b (int or float): The second number

    Returns:
    int or float: The sum of a and b
    """
    return a + b

# Example usage
print(add_numbers(3, 4))  # 7

# Accessing the function's docstring
print(add_numbers.__doc__)
```

```
7

    This function takes two numbers as input and returns their sum.

    Parameters:
    a (int or float): The first number
    b (int or float): The second number

    Returns:
    int or float: The sum of a and b
```

```
# Lambda function for addition
add = lambda x, y: x + y
print(add(3, 5))  # Output: 8

# Lambda function for multiplication
multiply = lambda x, y: x * y
print(multiply(4, 7))  # Output: 28

# Lambda function for finding the square of a number
square = lambda x: x ** 2
print(square(6))  # Output: 36
```

```
8
28
36
```

```
from functools import reduce

# Using lambda with map
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x ** 2, numbers))
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]

# Using lambda with filter
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4]

# Using lambda with reduce
sum_of_numbers = reduce(lambda x, y: x + y, numbers)
print(sum_of_numbers)  # Output: 15
```

```
[1, 4, 9, 16, 25]
[2, 4]
15
```

```
import numpy as np

# 1. Create a 1D array
array_1d = np.array([1, 2, 3, 4, 5])
print("1D Array:")
print(array_1d)

# 2. Create a 2D array
array_2d = np.array([[1, 2, 3], [4, 5, 6]])
print("\n2D Array:")
print(array_2d)

# 3. Create a 3D array
array_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("\n3D Array:")
print(array_3d)

# 4. Create a zero-initialized array
zeros_array = np.zeros((3, 4))
print("\nZeros Array:")
print(zeros_array)

# 5. Create an array with a range of numbers
range_array = np.arange(0, 20, 5)
print("\nArray with Range:")
print(range_array)

# Arrays for arithmetic operations
array_a = np.array([1, 2, 3])
array_b = np.array([4, 5, 6])

# 1. Addition
addition = array_a + array_b
print("Addition:")
print(addition)

# 2. Subtraction
subtraction = array_b - array_a
print("\nSubtraction:")
print(subtraction)

# 3. Multiplication
multiplication = array_a * array_b
print("\nMultiplication:")
print(multiplication)

# 4. Division
division = array_b / array_a
print("\nDivision:")
print(division)

# 5. Square of elements
square = array_a**2
print("\nSquare of elements:")
print(square)

# Create a 2D array
array = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])

# 1. Access a single element
print("Single Element (row 1, col 2):", array[1, 2])

# 2. Access a row
print("\nRow 0:", array[0])

# 3. Access a column
print("\nColumn 1:", array[:, 1])

# 4. Slice a subarray
print("\nSubarray (rows 0-1, cols 1-2):")
print(array[0:2, 1:3])

# 5. Reverse rows and columns
print("\nReversed Array:")
print(array[::-1, ::-1])

# Create arrays
array = np.array([[1, 2], [3, 4]])
array_2 = np.array([[5, 6], [7, 8]])

# 1. Reshape array
reshaped = array.reshape(4, 1)
```

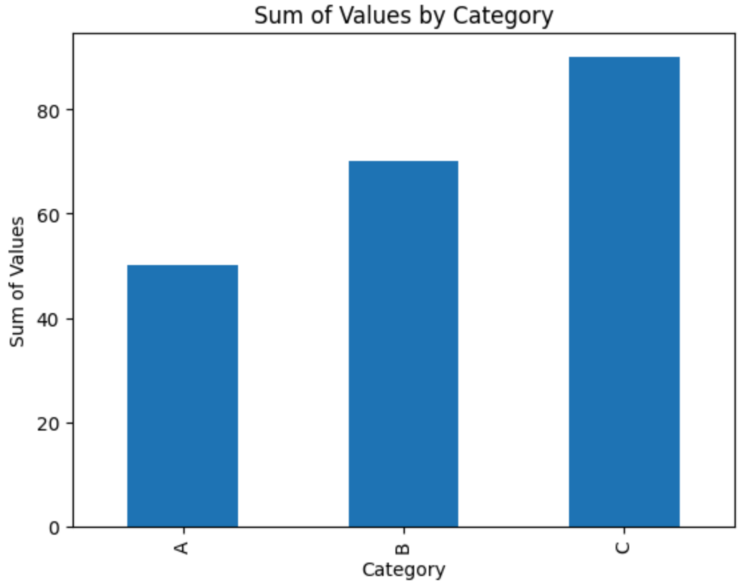
```
print("Reshaped Array:")
print(reshaped)

# 2. Transpose array
transposed = array.T
print("\nTransposed Array:")
print(transposed)

# 3. Concatenate arrays along rows
concatenated_row = np.concatenate((array, array_2), axis=0)
print("\nConcatenated along Rows:")
print(concatenated_row)

# 4. Concatenate arrays along columns
concatenated_col = np.concatenate((array, array_2), axis=1)
print("\nConcatenated along Columns:")
print(concatenated_col)

# 5. Flatten an array
flattened = array.flatten()
print("\nFlattened Array:")
print(flattened)

# 1. Random numbers between 0 and 1
random_floats = np.random.rand(3, 3)
print("Random Floats:")
print(random_floats)

# 2. Random integers
random_integers = np.random.randint(0, 10, size=(2, 3))
print("\nRandom Integers:")
print(random_integers)

# 3. Random numbers from a normal distribution
normal_dist = np.random.normal(0, 1, size=(2, 3))
print("\nNormal Distribution:")
print(normal_dist)

# 4. Random choice from a list
random_choice = np.random.choice([10, 20, 30, 40], size=5)
print("\nRandom Choice:")
print(random_choice)

# 5. Random seed for reproducibility
np.random.seed(42)
seeded_random = np.random.rand(2, 2)
print("\nSeeded Random Numbers:")
print(seeded_random)
```

```
1D Array:
[1 2 3 4 5]

2D Array:
[[1 2 3]
 [4 5 6]]

3D Array:
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]

Zeros Array:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

Array with Range:
[ 0  5 10 15]
Addition:
[5 7 9]

Subtraction:
[3 3 3]

Multiplication:
[ 4 10 18]

Division:
[4.  2.5 2. ]

Square of elements:
```

```
[1 4 9]
Single Element (row 1, col 2): 60

Row 0: [10 20 30]

Column 1: [20 50 80]

Subarray (rows 0-1, cols 1-2):
[[20 30]
 [50 60]]

Reversed Array:
[[90 80 70]
 [60 50 40]
 [30 20 10]]
Reshaped Array:
[[1]
 [2]
 [3]
 [4]]

Transposed Array:
[[1 3]
 [2 4]]
```

```
import pandas as pd

# 1. Create a Pandas Series
series = pd.Series([10, 20, 30, 40], name="Numbers")
print("Pandas Series:")
print(series)

# 2. Create a Pandas DataFrame
data = {
    "Name": ["Alice", "Bob", "Charlie"],
    "Age": [25, 30, 35],
    "Score": [85, 90, 95]
}
df = pd.DataFrame(data)
print("\nPandas DataFrame:")
print(df)

# 1. Load CSV file
csv_df = pd.read_csv('/content/sample_data/california_housing_test.csv')
print("Loaded CSV File:")
print(csv_df.head())

# 2. Create DataFrame directly from a dictionary (for demonstration purposes)
data = {'A': [1, 2], 'B': [3, 4]}
df = pd.DataFrame(data)
print("\nDirectly Created DataFrame:")
print(df)

# Data Cleaning
df = pd.DataFrame({
    "Name": ["Alice", "Bob", None],
    "Age": [25, None, 35],
    "Score": [85, 90, None]
})

# 1. Handle missing values
df_cleaned = df.fillna({"Name": "Unknown", "Age": df["Age"].mean(), "Score": 0})
print("Data After Cleaning:")
print(df_cleaned)

# 2. Rename columns
df_renamed = df_cleaned.rename(columns={"Name": "Full Name", "Score": "Exam Score"})
print("\nRenamed Columns:")
print(df_renamed)

# 3. Filter rows
filtered_df = df_cleaned[df_cleaned["Age"] > 30]
print("\nFiltered Rows (Age > 30):")
print(filtered_df)

import matplotlib.pyplot as plt

# Sample data
df = pd.DataFrame({
    "Category": ["A", "B", "C", "A", "B", "C"],
    "Values": [10, 20, 30, 40, 50, 60]
})

# 1. Group and aggregate data
grouped = df.groupby("Category").sum()
print("Grouped Data:")
print(grouped)

# 2. Visualization
grouped.plot(kind='bar', legend=False)
plt.title("Sum of Values by Category")
plt.xlabel("Category")
plt.ylabel("Sum of Values")
plt.show()

# Sample data
data = {
    "Department": ["HR", "HR", "IT", "IT", "Sales", "Sales"],
    "Employee": ["Alice", "Bob", "Charlie", "David", "Eve", "Frank"],
    "Salary": [50000, 55000, 60000, 65000, 70000, 75000],
    "Bonus": [5000, 4000, 6000, 7000, 8000, 9000]
}
df = pd.DataFrame(data)

# 1. Create a Pivot Table
pivot_table = df.pivot_table(values="Salary", index="Department", aggfunc="mean")
print("Pivot Table (Average Salary by Department):")
print(pivot_table)

# 2. Group data
grouped = df.groupby("Department").agg({"Salary": "mean", "Bonus": "sum"})
print("\nGrouped Data (Mean Salary and Total Bonus):")
print(grouped)
```

```
Pandas Series:
0    10
1    20
2    30
3    40
Name: Numbers, dtype: int64

Pandas DataFrame:
      Name  Age  Score
0    Alice   25     85
1      Bob   30     90
2  Charlie   35     95
Loaded CSV File:
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.05     37.37                27.0       3885.0           661.0   
1    -118.30     34.26                43.0       1510.0           310.0   
2    -117.81     33.78                27.0       3589.0           507.0   
3    -118.36     33.82                28.0         67.0            15.0   
4    -119.67     36.33                19.0       1241.0           244.0   

   population  households  median_income  median_house_value  
0      1537.0       606.0         6.6085            344700.0  
1       809.0       277.0         3.5990            176500.0  
2      1484.0       495.0         5.7934            270500.0  
3        49.0        11.0         6.1359            330000.0  
4       850.0       237.0         2.9375             81700.0  

Directly Created DataFrame:
   A  B
0  1  3
1  2  4
Data After Cleaning:
      Name   Age  Score
0    Alice  25.0   85.0
1      Bob  30.0   90.0
2  Unknown  35.0    0.0

Renamed Columns:
  Full Name   Age  Exam Score
0     Alice  25.0        85.0
1       Bob  30.0        90.0
2   Unknown  35.0         0.0

Filtered Rows (Age > 30):
      Name   Age  Score
2  Unknown  35.0    0.0
Grouped Data:
          Values
Category        
A             50
B             70
C             90
```



```
Pivot Table (Average Salary by Department):
             Salary
Department         
HR          52500.0
IT          62500.0
Sales       72500.0

Grouped Data (Mean Salary and Total Bonus):
             Salary  Bonus
Department                
HR          52500.0   9000
IT          62500.0  13000
```

```
    Sales       72500.0  17000
```

```
age = 20
income = 40000

credit_score = 750


if age < 18:
    print("You are a minor.")
elif 18 <= age < 60:
    print("You are an adult.")
else:
    print("You are a senior citizen.")


if income > 50000 and credit_score > 700:
    print("Eligible for premium credit card.")
elif income > 30000 or credit_score > 650:
    print("Eligible for basic credit card.")
else:
    print("Not eligible for credit card.")

age = 25
is_student = True

if age < 30:
    if is_student:
        print("Discount applied for students under 30.")
    else:
        print("No discount for non-students under 30.")
else:
    print("No discount available.")

# Iterate over a list
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    print(number)

# Count down from 5
count = 5
while count > 0:
    print(count)
    count -= 1

    # Nested loop for a multiplication table
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} x {j} = {i * j}")


# Break example
for i in range(1, 10):
    if i == 5:
        break
    print(i)

# Continue example
for i in range(1, 10):
    if i % 2 == 0:
        continue
    print(i)
```

```
You are an adult.
Eligible for basic credit card.
Discount applied for students under 30.
1
2
3
4
5
5
4
3
2
1
1 x 1 = 1
```

```
1 x 2 = 2
1 x 3 = 3
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
1
2
3
4
1
3
5
7
9
```

```
# Create and manipulate a list
my_list = [1, 2, 3, 4]
my_list.append(5)
my_list.remove(2)
print(my_list)
print(my_list[1:3])  # Slicing

# Tuples are immutable
my_tuple = (10, 20, 30)
print(my_tuple[1])

# Sets are unordered and have unique elements
my_set = {1, 2, 3}
my_set.add(4)
my_set.remove(2)
print(my_set)

# Create and manipulate a dictionary
my_dict = {"name": "Alice", "age": 25}
my_dict["age"] = 26
my_dict["city"] = "Vancouver"
print(my_dict)
```