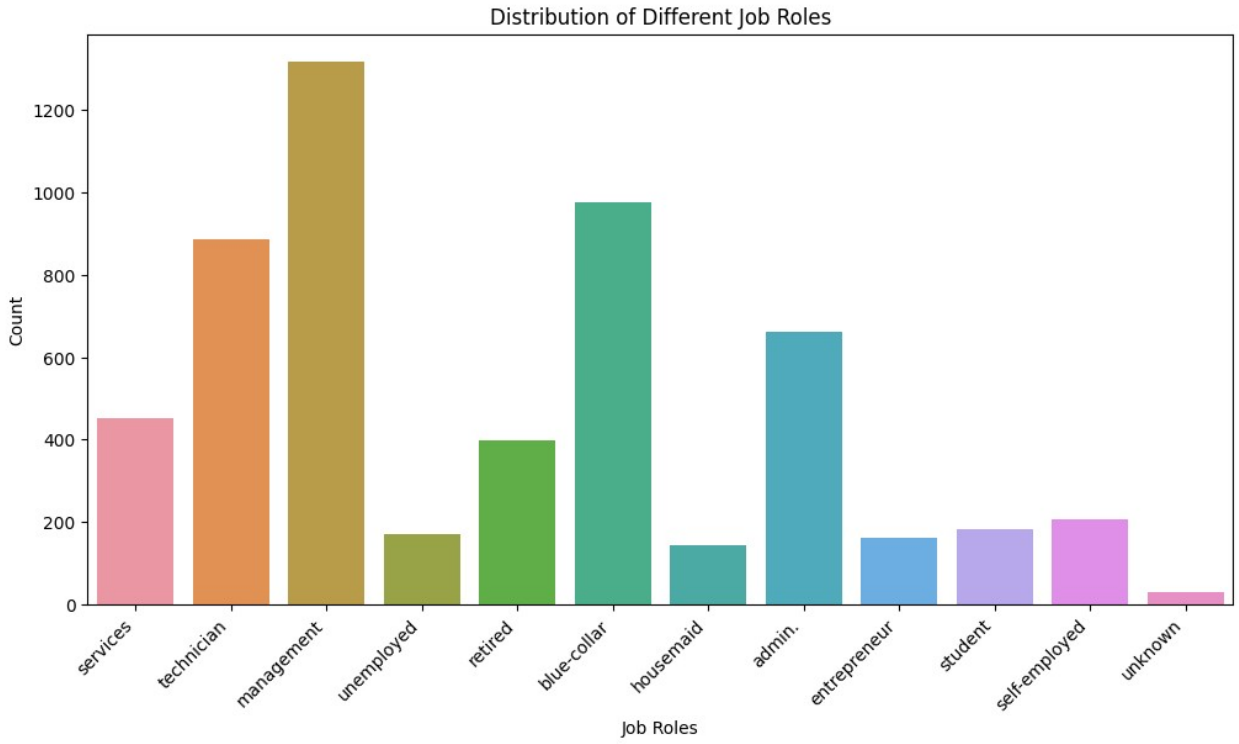
```python
import pandas as pd

df = pd.read_csv('bank_marketing.csv')  # Replace with the actual file path
print("Shape of the DataFrame:", df.shape)
print("Preview of the DataFrame:")
print(df.head())
print("Column Names:", df.columns)
print("Data Types of Columns:")
print(df.dtypes)
print("Statistical Summary:")
print(df.describe())
print("Missing Values:")
print(df.isnull().sum())
```

```
Shape of the DataFrame: (5581, 18)
Preview of the DataFrame:
   Unnamed: 0  age         job  marital  education default  balance
housing  \
0           0   41    services  married    unknown      no       88
yes
1           1   56  technician  married  secondary      no     1938
no
2           2   30    services   single  secondary      no      245
no
3           3   34  management   single   tertiary      no     1396
yes
4           4   29  technician   single  secondary      no      -13
yes

  loan   contact  day month  duration  campaign  pdays  previous
poutcome  \
0   no  cellular   11   may       105         1    336         2
failure
1  yes  cellular   26   feb       229         1    192         4
success
2  yes  cellular    8   jul       187         2     -1         0
unknown
3   no  cellular   17   jul       630         1     -1         0
unknown
4   no  cellular   14   may       512         3     -1         0
unknown

  deposit
0      no
1     yes
2      no
3      no
4      no
Column Names: Index(['Unnamed: 0', 'age', 'job', 'marital',
```

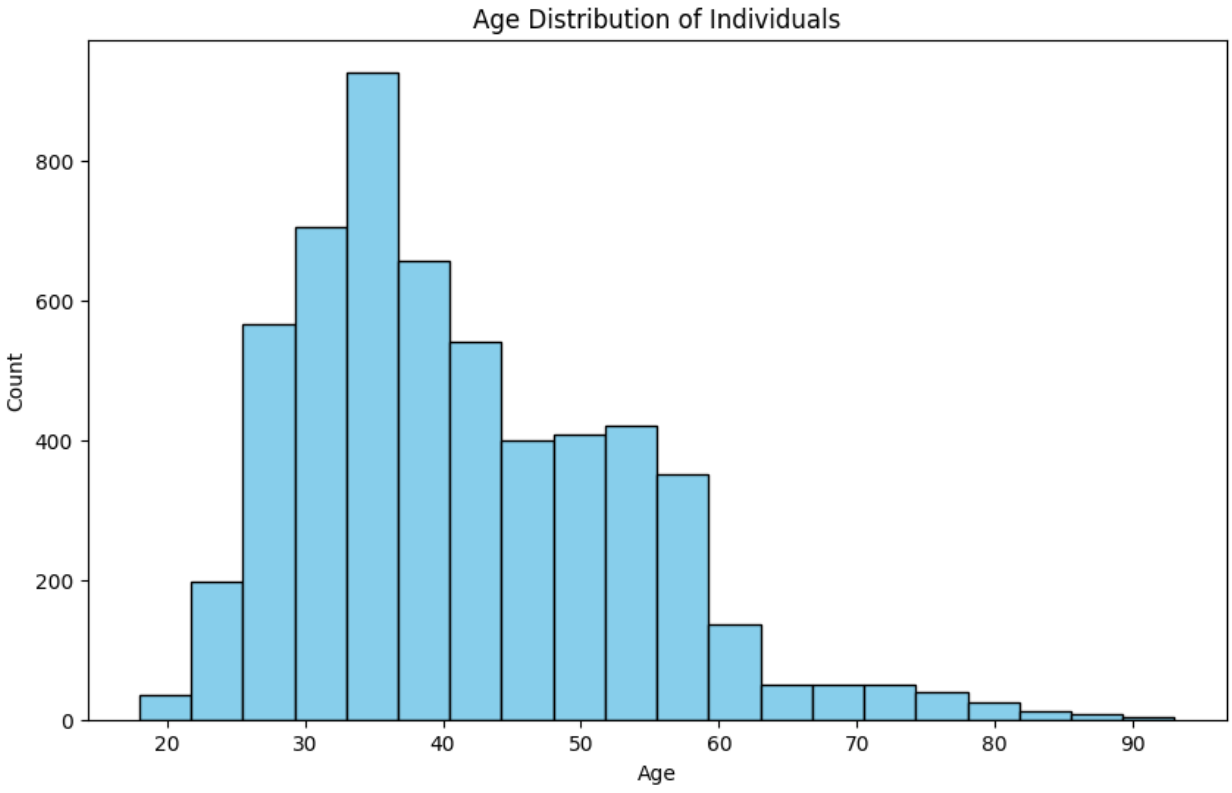
```
'education', 'default',
       'balance', 'housing', 'loan', 'contact', 'day', 'month',
'duration',
       'campaign', 'pdays', 'previous', 'poutcome', 'deposit'],
      dtype='object')
Data Types of Columns:
Unnamed: 0     int64
age            int64
job           object
marital       object
education     object
default       object
balance        int64
housing       object
loan          object
contact       object
day            int64
month         object
duration       int64
campaign       int64
pdays          int64
previous       int64
poutcome      object
deposit       object
dtype: object
Statistical Summary:
        Unnamed: 0          age       balance          day
duration  \
count  5581.000000  5581.000000   5581.000000  5581.000000
5581.000000
mean   2790.000000    41.169683   1514.736786    15.693603
368.175954
std    1611.240257    11.926044   3266.534626     8.461086
344.131053
min       0.000000    18.000000  -3058.000000     1.000000
3.000000
25%    1395.000000    32.000000    110.000000     8.000000
137.000000
50%    2790.000000    39.000000    542.000000    15.000000
254.000000
75%    4185.000000    49.000000   1747.000000    22.000000
485.000000
max    5580.000000    93.000000  81204.000000    31.000000
3284.000000

          campaign        pdays     previous
count  5581.000000  5581.000000  5581.000000
mean      2.507436    52.534313     0.849669
std       2.770717   110.754995     2.311684
```

```
min       1.000000    -1.000000     0.000000
25%       1.000000    -1.000000     0.000000
50%       2.000000    -1.000000     0.000000
75%       3.000000    57.000000     1.000000
max      63.000000   842.000000    41.000000
Missing Values:
Unnamed: 0    0
age           0
job           0
marital       0
education     0
default       0
balance       0
housing       0
loan          0
contact       0
day           0
month         0
duration      0
campaign      0
pdays         0
previous      0
poutcome      0
deposit       0
dtype: int64
```

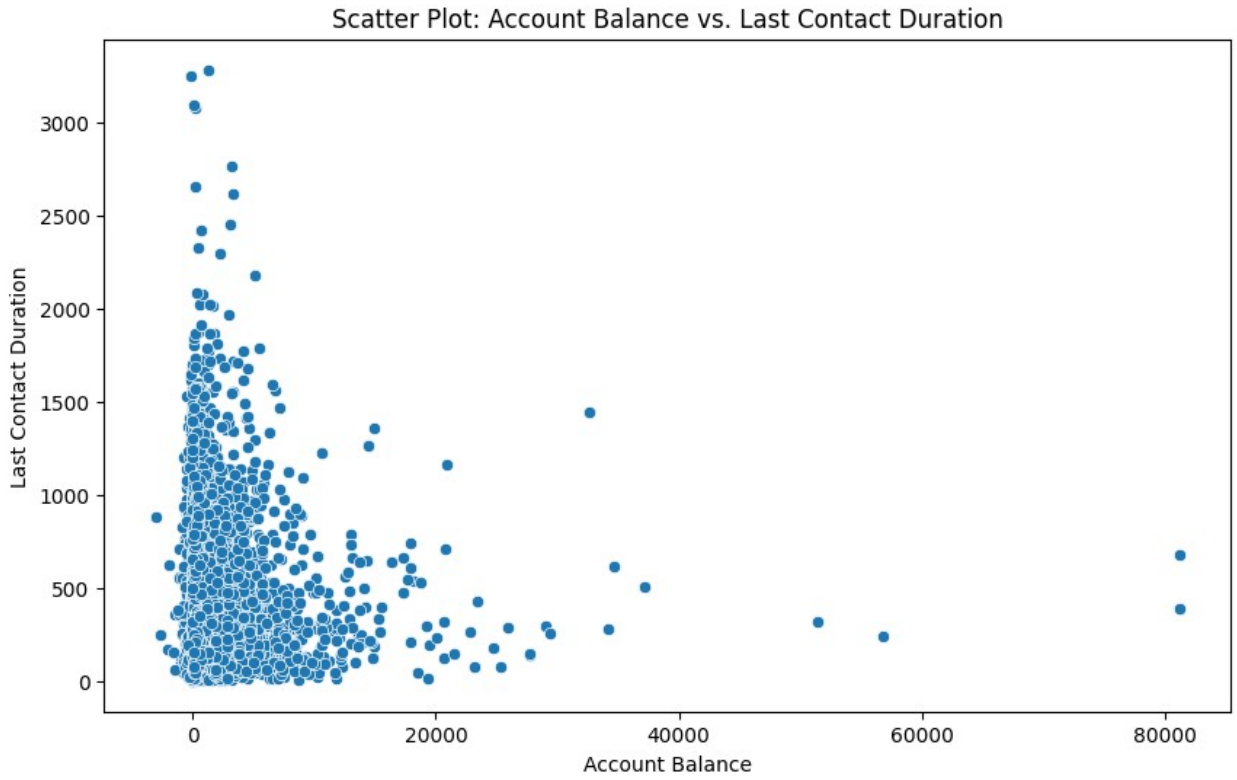
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your DataFrame
# Example DataFrame:
df = pd.read_csv('bank_marketing.csv')  # Replace with the actual file path

plt.figure(figsize=(12, 6))
sns.countplot(x='job', data=df)
plt.title('Distribution of Different Job Roles')
plt.xlabel('Job Roles')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.show()
```




```
plt.figure(figsize=(10, 6))
plt.hist(df['age'], bins=20, color='skyblue', edgecolor='black')
plt.title('Age Distribution of Individuals')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='balance', y='duration', data=df)
plt.title('Scatter Plot: Account Balance vs. Last Contact Duration')
plt.xlabel('Account Balance')
plt.ylabel('Last Contact Duration')
plt.show()
```




```
# Drop Unnamed: 0, missing values, and duplicates
df_cleaned = df.drop(columns=['Unnamed:
0']).dropna().drop_duplicates()

print("Shape after operations:", df_cleaned.shape)
```

```
Shape after operations: (5581, 17)
```

```
average_age_subscribed = df[df['deposit'] == 'yes']['age'].mean()
print("Average age of subscribed clients:", average_age_subscribed)
```

```
Average age of subscribed clients: 41.42677345537757
```

2 Create a list of 10 random players in a football team, display the list, sort it, and remove the first item with the specified value.

```
import random

players = ["Player" + str(i) for i in range(1, 11)]
print("Original List:", players)
```

```
random.shuffle(players)
print("Shuffled List:", players)

players.sort()
print("Sorted List:", players)

value_to_remove = "Player5"  # Replace with the specified value
players.remove(value_to_remove)
print("List after removing", value_to_remove + ":", players)
```

```
Original List: ['Player1', 'Player2', 'Player3', 'Player4', 'Player5',
'Player6', 'Player7', 'Player8', 'Player9', 'Player10']
Shuffled List: ['Player7', 'Player10', 'Player3', 'Player9',
'Player1', 'Player6', 'Player2', 'Player5', 'Player4', 'Player8']
Sorted List: ['Player1', 'Player10', 'Player2', 'Player3', 'Player4',
'Player5', 'Player6', 'Player7', 'Player8', 'Player9']
List after removing Player5: ['Player1', 'Player10', 'Player2',
'Player3', 'Player4', 'Player6', 'Player7', 'Player8', 'Player9']
```

#3 Write a program to randomly select 10 integer elements from the range 100 to 200 and find the smallest among all.

```
import random

random_integers = random.sample(range(100, 201), 10)
print("Random Integers:", random_integers)

smallest_element = min(random_integers)
print("Smallest Element:", smallest_element)
```

```
Random Integers: [117, 149, 181, 138, 108, 102, 176, 144, 130, 143]
Smallest Element: 102
```

4 Create a dictionary of 5 countries with their currency details and display them.

```
countries = {
    "Country1": {"Currency": "Currency1"},
    "Country2": {"Currency": "Currency2"},
    "Country3": {"Currency": "Currency3"},
    "Country4": {"Currency": "Currency4"},
    "Country5": {"Currency": "Currency5"},
}

for country, details in countries.items():
    print(country, ":", details)
```

```
Country1 : {'Currency': 'Currency1'}
Country2 : {'Currency': 'Currency2'}
Country3 : {'Currency': 'Currency3'}
Country4 : {'Currency': 'Currency4'}
Country5 : {'Currency': 'Currency5'}
```

5 Write a program that takes a sentence as input, replaces each blank with a hyphen, and prints the modified sentence.

```
sentence = input("Enter a sentence: ")
modified_sentence = sentence.replace(" ", "-")
print("Modified Sentence:", modified_sentence)
```

```
Enter a sentence: Hello Owrld
Modified Sentence: Hello-Owrld
```

6 Python Program to Solve Quadratic Equations.

```
import cmath

def solve_quadratic(a, b, c):
    d = cmath.sqrt(b**2 - 4*a*c)
    root1 = (-b + d) / (2*a)
    root2 = (-b - d) / (2*a)
    return root1, root2

# Example usage:
coeff_a = 1
coeff_b = -3
coeff_c = 2
print("Roots of the quadratic equation:", solve_quadratic(coeff_a,
coeff_b, coeff_c))
```

```
Roots of the quadratic equation: ((2+0j), (1+0j))
```

7 Write a program to print a specified number of Fibonacci numbers.

```
def fibonacci(n):
    fib_sequence = [0, 1]
    while len(fib_sequence) < n:
        fib_sequence.append(fib_sequence[-1] + fib_sequence[-2])
    return fib_sequence[:n]

# Example usage:
num_fibonacci = 8  # Replace with the specified number
print("Fibonacci Numbers:", fibonacci(num_fibonacci))
```

```
Fibonacci Numbers: [0, 1, 1, 2, 3, 5, 8, 13]
```

8 Create a NumPy array filled with all ones.

```
import numpy as np

ones_array = np.ones((3, 4))  # Replace with desired shape
print("NumPy Array with Ones:\n", ones_array)
```

```
NumPy Array with Ones:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

9 Check whether a NumPy array contains a specified row.

```
import numpy as np

array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  # Replace with
your array
specified_row = np.array([4, 5, 6])  # Replace with the specified row
contains_row = any(np.array_equal(row, specified_row) for row in
array)

print("Does the array contain the specified row?", contains_row)
```

```
Does the array contain the specified row? True
```

10 Compute mathematical operations on Array, Add & Multiply two matrices.

```
import numpy as np

# Example matrices
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])

# Addition
result_addition = np.add(matrix1, matrix2)

# Multiplication
result_multiplication = np.multiply(matrix1, matrix2)

print("Matrix Addition:\n", result_addition)
print("Matrix Multiplication:\n", result_multiplication)
```

```
Matrix Addition:
 [[ 6  8]
 [10 12]]
Matrix Multiplication:
 [[ 5 12]
 [21 32]]
```

11 Find the most frequent value in a NumPy array.

```
import numpy as np

array = np.array([1, 2, 2, 3, 4, 4, 4, 5])  # Replace with your array
most_frequent_value = np.bincount(array).argmax()

print("Most frequent value:", most_frequent_value)
```

```
Most frequent value: 4
```

12 Flatten a 2D numpy array into a 1D array.

```
import numpy as np

array_2d = np.array([[1, 2, 3], [4, 5, 6]])  # Replace with your array
array_1d = array_2d.flatten()

print("Flattened 1D Array:", array_1d)
```

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

13 Calculate the sum of all columns in a 2D NumPy array.

```
import numpy as np

array_2d = np.array([[1, 2, 3], [4, 5, 6]])  # Replace with your array
sum_columns = np.sum(array_2d, axis=0)

print("Sum of Columns:", sum_columns)
```

```
Sum of Columns: [5 7 9]
```

14 Write a function to calculate the arithmetic mean of a variable number of values.

```
def arithmetic_mean(*values):
    return sum(values) / len(values)

# Example usage:
result_mean = arithmetic_mean(2, 4, 6, 8, 10)
print("Arithmetic Mean:", result_mean)
```

```
Arithmetic Mean: 6.0
```

15 Write a lambda function to find the largest of 2 numbers.

```
largest_number = lambda x, y: x if x > y else y

# Example usage:
result_largest = largest_number(7, 12)
print("Largest of Two Numbers:", result_largest)
```

```
Largest of Two Numbers: 12
```

16 Write a program to find the factorial of a given number using recursion.

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

# Example usage:
result_factorial = factorial_recursive(5)
print("Factorial:", result_factorial)
```

```
Factorial: 120
```

17 Make an "Invalid Marks" exception class that is thrown when marks obtained by a student exceed 100.

```
class InvalidMarksException(Exception):
    pass

# Example usage:
marks = 105  # Replace with student's marks
try:
    if marks > 100:
        raise InvalidMarksException("Invalid Marks: Marks cannot
exceed 100.")
except InvalidMarksException as e:
    print(e)
```

```
Invalid Marks: Marks cannot exceed 100.
```

18 Create a user-defined exception to display a proper message when the value of (b*d) is zero.

```
class BDMultiplicationZero(Exception):
    pass

# Example usage:
b_value = 0  # Replace with the actual value of b
d_value = 5  # Replace with the actual value of d

try:
    if b_value * d_value == 0:
```

```
        raise BDMultiplicationZero("Invalid: (b*d) should not be
zero.")
except BDMultiplicationZero as e:
    print(e)
```

```
Invalid: (b*d) should not be zero.
```

19 Make use of the assert statement to catch AssertionError.

```
try:
    assert 2 + 2 == 5, "Assertion Error: 2 + 2 is not equal to 5"
except AssertionError as e:
    print(e)
```

```
Assertion Error: 2 + 2 is not equal to 5
```

21 Write a program to calculate the factorial of a number using loops.

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

# Example usage:
result_factorial_loop = factorial_loop(5)
print("Factorial (using loops):", result_factorial_loop)
```

```
Factorial (using loops): 120
```

22 Create a script to reverse a string entered by the user.

```
user_input = input("Enter a string: ")
reversed_string = user_input[::-1]
print("Reversed String:", reversed_string)
```

```
Enter a string: Helloworld
Reversed String: dlrowolleH
```

23 Develop a program to find the largest and smallest elements in an array.

```
import numpy as np

array = np.array([4, 2, 7, 1, 9, 5])  # Replace with your array
largest_element = np.max(array)
smallest_element = np.min(array)

print("Largest Element:", largest_element)
print("Smallest Element:", smallest_element)
```

```
Largest Element: 9
Smallest Element: 1
```

24 Write a function that checks whether a passed string is a palindrome or not.

```
def is_palindrome(s):
    s = s.lower()  # Convert to lowercase for case-insensitivity
    return s == s[::-1]

# Example usage:
input_string = "level"  # Replace with your string
if is_palindrome(input_string):
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

```
The string is a palindrome.
```

25 Develop a Python function that merges two dictionaries into one.

```
dict1 = {"a": 1, "b": 2}
dict2 = {"b": 3, "c": 4}

merged_dict = {**dict1, **dict2}
print("Merged Dictionary:", merged_dict)
```

```
Merged Dictionary: {'a': 1, 'b': 3, 'c': 4}
```

26 Implement a Python class representing a Rectangle with methods to calculate area and perimeter.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

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

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

# Example usage:
rectangle = Rectangle(5, 3)
print("Area:", rectangle.calculate_area())
print("Perimeter:", rectangle.calculate_perimeter())
```

```
Area: 15
Perimeter: 16
```

27 Create a Numpy program to compute the inverse of a matrix.

```
import numpy as np

matrix = np.array([[4, 7], [2, 6]])  # Replace with your matrix
inverse_matrix = np.linalg.inv(matrix)

print("Original Matrix:\n", matrix)
print("Inverse Matrix:\n", inverse_matrix)
```

```
Original Matrix:
 [[4 7]
 [2 6]]
Inverse Matrix:
 [[ 0.6 -0.7]
 [-0.2  0.4]]
```

28 Develop a program to merge two Pandas DataFrames.

```
import pandas as pd

# Assuming df1 and df2 are your DataFrames
# Example DataFrames:
df1 = pd.DataFrame({"A": [1, 2], "B": [3, 4]})
df2 = pd.DataFrame({"A": [5, 6], "B": [7, 8]})

merged_df = pd.concat([df1, df2], ignore_index=True)
print("Merged DataFrame:\n", merged_df)
```

```
Merged DataFrame:
    A  B
0  1  3
1  2  4
2  5  7
3  6  8
```

29 Develop a script to find the GCD (Greatest Common Divisor) of two numbers.

```
import math

def find_gcd(x, y):
    return math.gcd(x, y)

# Example usage:
number1 = 24
number2 = 36
result_gcd = find_gcd(number1, number2)
print("GCD of", number1, "and", number2, "is", result_gcd)
```

```
GCD of 24 and 36 is 12
```

30 Write a Python function to find the sum of digits of a number using recursion.

```
def sum_of_digits_recursive(number):
    if number == 0:
        return 0
    else:
```

```
        return number % 10 + sum_of_digits_recursive(number // 10)

# Example usage:
number_to_sum = 12345
result_sum = sum_of_digits_recursive(number_to_sum)
print("Sum of Digits:", result_sum)
```

```
Sum of Digits: 15
```

31 Create a Python program to catch multiple exceptions in a single block.

```
try:
    # Code that may raise exceptions
    x = 10 / 0
    y = int("abc")
except (ZeroDivisionError, ValueError) as e:
    print("Exception caught:", e)
```

```
Exception caught: division by zero
```

32 Create a Python program to sort an array of integers using Numpy.

```
import numpy as np

array_to_sort = np.array([3, 1, 4, 1, 5, 9, 2, 6, 5, 3])  # Replace
with your array
sorted_array = np.sort(array_to_sort)

print("Original Array:", array_to_sort)
print("Sorted Array:", sorted_array)
```

```
Original Array: [3 1 4 1 5 9 2 6 5 3]
Sorted Array: [1 1 2 3 3 4 5 5 6 9]
```

Implement a simple banking application with methods:

create Account(), deposit() , withdraw(), computeInterest(), displayBalance()

```python
class BankingApplication:
    def __init__(self):
        self.balance = 0

    def create_account(self, account_holder):
        self.account_holder = account_holder
        print(f"Account created successfully for {account_holder}.")

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited ${amount}. New balance: $
{self.balance}")
        else:
            print("Invalid deposit amount.")

    def withdraw(self, amount):
        if 0 < amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawn ${amount}. New balance: $
{self.balance}")
        else:
            print("Invalid withdrawal amount or insufficient funds.")

    def compute_interest(self, rate):
        interest = (self.balance * rate) / 100
        self.balance += interest
        print(f"Interest added. New balance: ${self.balance}")

    def display_balance(self):
        print(f"Current balance for {self.account_holder}: $
{self.balance}")

# Example usage:
bank = BankingApplication()
bank.create_account("John Doe")
bank.deposit(1000)
bank.withdraw(500)
bank.compute_interest(5)
bank.display_balance()
```

```
Account created successfully for John Doe.
Deposited $1000. New balance: $1000
Withdrawn $500. New balance: $500
Interest added. New balance: $525.0
Current balance for John Doe: $525.0
```