Python Functions

```python
# 1. Functions with Different Numbers of Parameters and Return Types

# Example 1: No parameters, returns a string
def greet_world():
    return "Hello, World!"

print(greet_world())
```

```
Hello, World!
```

```python
# Example 2: One parameter, returns an integer
def double_value(x):
    return x * 2
print(double_value(10))
```

```
20
```

```python
# Example 3: Two parameters, returns a concatenated string
def full_name(first_name, last_name):
    return f"{first_name} {last_name}"

print(full_name("John", "Doe"))
```

```
John Doe
```

```python
# Example 4: Multiple parameters, returns a list
def create_list(a, b, c):
    return [a, b, c]

print(create_list(1, 2, 3))
```

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

```python
# Example 5: Three parameters, returns a dictionary
def person_info(name, age, city):
    return {"Name": name, "Age": age, "City": city}

print(person_info("Alice", 25, "New York"))
```

```
{'Name': 'Alice', 'Age': 25, 'City': 'New York'}
```

1.2 Explore function scope and variable accessibility.

```python
# Example 1: Local scope
def local_variable_example():
    local_var = 10
    return local_var

print(local_variable_example())
```

```
10
```

```python
# Example 2: Global variable usage
global_var = 20

def use_global():
    return global_var

print(use_global())
```

```
20
```

```python
# Example 3: Modify global variable inside a function
def modify_global():
    global global_var
    global_var += 10
    return global_var

print(modify_global())
```

```
30
```

```python
# Example 4: Nonlocal variable (used in nested functions)
def outer_function():
    nonlocal_var = "I am outer"

    def inner_function():
        nonlocal nonlocal_var
        nonlocal_var = "I am modified by inner"
        return nonlocal_var

    inner_function()
    return nonlocal_var

print(outer_function())
```

```
I am modified by inner
```

```python
# Example 5: Parameter shadowing
x = 50

def shadow_example(x):
    return x + 5  # Uses the local x

print(shadow_example(10))
```

```
15
```

1.3 Implement functions with default argument values.

```python
# Example 1: Single default argument
def greet(name="World"):
    return f"Hello, {name}!"

print(greet())
```

```
Hello, World!
```

```python
# Example 2: Two parameters, one with a default
def calculate_area(length, width=5):
    return length * width

print(calculate_area(10))
```

```
50
```

```python
# Example 3: Multiple default arguments
def introduce(name="John", age=30, city="New York"):
    return f"My name is {name}, I am {age} years old, and I live in 
{city}."

print(introduce())
```

```
My name is John, I am 30 years old, and I live in New York.
```

```python
# Example 4: Default value based on another argument
def add_with_offset(a, b=10):
    return a + b

print(add_with_offset(5))
```

```
15
```

```python
# Example 5: Default argument that changes dynamically
def append_item_to_list(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items

print(append_item_to_list("apple"))
```

```
['apple']
```

1.4 recursive functions.

```python
# Example 1: Factorial function
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
```

```
120
```

```
# Example 2: Fibonacci sequence
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(6))
```

```
8
```

```
# Example 3: Sum of list elements
def sum_list(lst):
    if len(lst) == 0:
        return 0
    else:
        return  lst[0] +sum_list(lst[1:])

print(sum_list([1, 2, 3, 4]))
```

```
10
```

```
# Example 4: Count down to zero
def countdown(n):
    if n == 0:
        print("Blast off!")
    else:
        print(n)
        countdown(n - 1)

countdown(5)
```

```
5
4
3
2
1
Blast off!
```

```
# Example 5: Reverse a string
def reverse_string(s):
    if len(s) == 0:
        return s
    else:
        return reverse_string(s[1:]) + s[0]

print(reverse_string("hello"))
```

```
olleh
```

        1.5 Demonstrate how to use docstrings to document functions.

```
# Example 1: Simple docstring
def greet(name):
    """
    Greets a person by their name.

    Parameters:
    name (str): The name of the person to greet.

    Returns:
    str: A greeting message.
    """
    return f"Hello, {name}!"

print(greet.__doc__)
```

```

    Greets a person by their name.

    Parameters:
    name (str): The name of the person to greet.

    Returns:
    str: A greeting message.
    
```

```
# Example 2: Docstring with multiple parameters
def add_numbers(a, b):
    """
    Adds two numbers together.

    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
```

```
print(add_numbers.__doc__)


    Adds two numbers together.

    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.



# Example 3: Docstring for a function with default arguments
def describe_person(name, age=30):
    """
    Provides a description of a person.

    Parameters:
    name (str): The person's name.
    age (int, optional): The person's age. Defaults to 30.

    Returns:
    str: A description of the person.
    """
    return f"{name} is {age} years old."

print(describe_person.__doc__)


    Provides a description of a person.

    Parameters:
    name (str): The person's name.
    age (int, optional): The person's age. Defaults to 30.

    Returns:
    str: A description of the person.



# Example 4: Docstring with a recursive function
def factorial(n):
    """
    Calculates the factorial of a number using recursion.

    Parameters:
    n (int): The number to calculate the factorial of.
    Returns:
    int: The factorial of n.
    """
    if n == 1:
        return 1
    return n * factorial(n - 1)

print(factorial.__doc__)
```

```

    Calculates the factorial of a number using recursion.

    Parameters:
    n (int): The number to calculate the factorial of.

    Returns:
    int: The factorial of n.
```

```
# Example 5: Docstring with a return type of None
def print_message():
    """
    Prints a simple message.

    Returns:
    None
    """
    print("Hello, world!")

print(print_message.__doc__)
```

```

    Prints a simple message.

    Returns:
    None
```

Lambda Functions

```
# Example 1: Lambda function for adding two numbers
add = lambda a, b: a + b
print(add(5, 3))

# Example 2: Lambda function for squaring a number
square = lambda x: x ** 2
print(square(4))
```

```
# Example 3: Lambda function for finding the maximum of two numbers
maximum = lambda a, b: a if a > b else b
print(maximum(10, 15))

# Example 4: Lambda function for checking if a number is even
is_even = lambda x: x % 2 == 0
print(is_even(7))

# Example 5: Lambda function for concatenating two strings
concat = lambda s1, s2: s1 + s2
print(concat("Hello, ", "World!"))
```

```
8
16
15
False
Hello, World!
```

```
from functools import reduce

# Example 1: Using lambda with map (to square all numbers in a list)
numbers = [1, 2, 3, 4]
squares = list(map(lambda x: x ** 2, numbers))
print(squares)

# Example 2: Using lambda with filter (to filter even numbers from a
list)
numbers = [1, 2, 3, 4, 5, 6]
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)

# Example 3: Using lambda with reduce (to find the product of all
numbers in a list)
numbers = [1, 2, 3, 4]
product = reduce(lambda x, y: x * y, numbers)
print(product)

# Example 4: Using lambda with map (to convert temperatures from
Celsius to Fahrenheit)
celsius = [0, 10, 20, 30]
fahrenheit = list(map(lambda c: (c * 9/5) + 32, celsius))
print(fahrenheit)

# Example 5: Using lambda with filter (to find words longer than 3
characters)
words = ["hi", "hello", "sun", "cat", "elephant"]
long_words = list(filter(lambda word: len(word) > 3, words))
print(long_words)
```

```
[1, 4, 9, 16]
[2, 4, 6]
```

```
24
[32.0, 50.0, 68.0, 86.0]
['hello', 'elephant']
```

```python
# Example 1: Regular function for squaring a number
def square_function(x):
    return x ** 2

# Lambda equivalent
square_lambda = lambda x: x ** 2

# Use
print(square_function(4))
print(square_lambda(4))
```

```
16
16
```

```python
# Example 2: Regular function for checking if a number is positive
def is_positive(n):
    return n > 0

# Lambda equivalent
is_positive_lambda = lambda n: n > 0

# Use
print(is_positive(5))
print(is_positive_lambda(5))
```

```
True
True
```

```python
# Example 3: Regular function for adding two numbers
def add_function(a, b):
    return a + b

# Lambda equivalent
add_lambda = lambda a, b: a + b

# Use
print(add_function(10, 20))
print(add_lambda(10, 20))
```

```
30
30
```

```python
# Example 4: Regular function for filtering even numbers from a list
def filter_even(numbers):
    return [n for n in numbers if n % 2 == 0]

# Lambda with filter equivalent
```

```python
numbers = [1, 2, 3, 4, 5, 6]
filter_even_lambda = list(filter(lambda x: x % 2 == 0, numbers))

# Use
print(filter_even(numbers))
print(filter_even_lambda)
```

```
[2, 4, 6]
[2, 4, 6]
```

```python
# Example 5: Regular function for sorting a list of tuples by the
second element
def sort_by_second_element(tuples):
    return sorted(tuples, key=lambda x: x[1])

# Equivalent lambda directly in sorted
tuples = [(1, 2), (3, 1), (5, 4)]
sorted_tuples = sorted(tuples, key=lambda x: x[1])

# Use
print(sort_by_second_element(tuples))
print(sorted_tuples)
```

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

NumPy

```python
 import numpy as np
# Example 1: 1D Array
arr_1d = np.array([1, 2, 3, 4, 5])
print("1D Array:", arr_1d)

# Example 2: 2D Array (Matrix)
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
print("2D Array:\n", arr_2d)

# Example 3: 3D Array
arr_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("3D Array:\n", arr_3d)

# Example 4: Array with zeros
arr_zeros = np.zeros((3, 3))
print("Array with Zeros:\n", arr_zeros)

# Example 5: Array with a range of numbers
arr_range = np.arange(1, 10)
print("Array with Range:\n", arr_range)
```

```
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]]]
Array with Zeros:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
Array with Range:
 [1 2 3 4 5 6 7 8 9]
```

```python
# Example 1: Adding a scalar to an array
arr = np.array([1, 2, 3, 4])
arr_add = arr + 10
print("Add 10 to each element:", arr_add)

# Example 2: Element-wise addition between two arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr_sum = arr1 + arr2
print("Element-wise addition:", arr_sum)

# Example 3: Element-wise multiplication
arr_mul = arr1 * arr2
print("Element-wise multiplication:", arr_mul)

# Example 4: Array division by a scalar
arr_div = arr1 / 2
print("Array divided by 2:", arr_div)

# Example 5: Matrix multiplication
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])
matrix_mul = np.dot(matrix1, matrix2)
print("Matrix multiplication:\n", matrix_mul)
```

```
Add 10 to each element: [11 12 13 14]
Element-wise addition: [5 7 9]
Element-wise multiplication: [ 4 10 18]
Array divided by 2: [0.5 1.  1.5]
Matrix multiplication:
 [[19 22]
 [43 50]]
```

```python
     # Example 1: Access a specific element (2D array)
arr_2d = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])
element = arr_2d[1, 2]  # Row 2, Column 3
print("Access element:", element)

# Example 2: Slice a portion of a 1D array
arr_1d = np.array([10, 20, 30, 40, 50])
slice_1d = arr_1d[1:4]
print("Sliced 1D array:", slice_1d)

# Example 3: Slice a portion of a 2D array
slice_2d = arr_2d[0:2, 1:3]
print("Sliced 2D array:\n", slice_2d)

# Example 4: Reverse a 1D array
reversed_arr = arr_1d[::-1]
print("Reversed 1D array:", reversed_arr)

# Example 5: Use Boolean indexing
bool_index = arr_1d > 30
filtered_arr = arr_1d[bool_index]
print("Filtered array (elements > 30):", filtered_arr)
```

```
Access element: 60
Sliced 1D array: [20 30 40]
Sliced 2D array:
 [[20 30]
 [50 60]]
Reversed 1D array: [50 40 30 20 10]
Filtered array (elements > 30): [40 50]
```

```python
# Example 1: Reshape a 1D array to a 2D array
arr = np.array([1, 2, 3, 4, 5, 6])
reshaped_arr = arr.reshape(2, 3)
print("Reshaped array:\n", reshaped_arr)

# Example 2: Transpose of a 2D array
arr_2d = np.array([[1, 2], [3, 4], [5, 6]])
transposed_arr = arr_2d.T
print("Transposed array:\n", transposed_arr)

# Example 3: Concatenate two 1D arrays
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
concatenated_arr = np.concatenate((arr1, arr2))
print("Concatenated array:", concatenated_arr)

# Example 4: Concatenate along a new axis (stacking)
stacked_arr = np.stack((arr1, arr2))
print("Stacked array:\n", stacked_arr)
```

```
# Example 5: Flatten a 2D array to 1D
flattened_arr = arr_2d.flatten()
print("Flattened array:", flattened_arr)
```

```
Reshaped array:
 [[1 2 3]
 [4 5 6]]
Transposed array:
 [[1 3 5]
 [2 4 6]]
Concatenated array: [1 2 3 4 5 6]
Stacked array:
 [[1 2 3]
 [4 5 6]]
Flattened array: [1 2 3 4 5 6]
```

```
# Example 1: Generate an array of random numbers (uniform
distribution)
random_arr = np.random.rand(3, 3)
print("Random array (uniform distribution):\n", random_arr)

# Example 2: Generate random integers within a specific range
random_ints = np.random.randint(0, 10, size=(2, 3))
print("Random integers:\n", random_ints)

# Example 3: Generate random numbers from a normal distribution
random_normal = np.random.randn(3, 3)
print("Random normal distribution array:\n", random_normal)

# Example 4: Set a random seed for reproducibility
np.random.seed(42)
random_seeded = np.random.rand(3)
print("Random array with seed:\n", random_seeded)

# Example 5: Random choice from an array
arr = np.array([10, 20, 30, 40, 50])
random_choice = np.random.choice(arr, size=3)
print("Random choice from array:", random_choice)
```

```
Random array (uniform distribution):
 [[0.70448115 0.46881541 0.15907308]
 [0.0832982  0.23796389 0.5865192 ]
 [0.23510121 0.11874905 0.93898082]]
Random integers:
 [[0 3 1]
 [7 6 2]]
```

```
Random normal distribution array:
 [[ 0.82949006  1.56632586 -0.48290652]
 [-0.86296755  0.21939892  0.79896061]
 [-1.09288237  0.34883215 -0.72899549]]
Random array with seed:
 [0.37454012 0.95071431 0.73199394]
Random choice from array: [50 50 20]
```

Pandas

```
import pandas as pd

# Example 1: Create a Pandas Series from a list
data = [10, 20, 30, 40]
series = pd.Series(data)
print("Pandas Series:\n", series)
```

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

```
# Example 2: Create a Pandas DataFrame from a dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print("Pandas DataFrame:\n", df)
```

```
Pandas DataFrame:
       Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
```

```
# Example 3: Create a DataFrame with a custom index
data = {'Product': ['A', 'B', 'C'], 'Price': [100, 150, 200]}
df_custom_index = pd.DataFrame(data, index=['x1', 'x2', 'x3'])
print("DataFrame with custom index:\n", df_custom_index)
```

```
DataFrame with custom index:
    Product  Price
x1       A    100
x2       B    150
x3       C    200
```

```
# Example 4: Create a DataFrame from a NumPy array
import numpy as np
data = np.random.rand(3, 3)
```

```
df_numpy = pd.DataFrame(data, columns=['A', 'B', 'C'])
print("DataFrame from NumPy array:\n", df_numpy)

DataFrame from NumPy array:
           A         B         C
0  0.817075  0.508633  0.992858
1  0.831436  0.035796  0.841585
2  0.170212  0.015384  0.479893


# Example 5: Create a Series with a custom index
series_custom_index = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
print("Series with custom index:\n", series_custom_index)

Series with custom index:
 a    1
b    2
c    3
dtype: int64



        Load data from various file formats (CSV, Excel, etc.).

import pandas as pd
# Example 1: Load data from a CSV file
df_csv = pd.read_csv('data.csv')
print("Data loaded from CSV:\n", df_csv.head())

Data loaded from CSV:
 Empty DataFrame
Columns: [this is data.csv file ]
Index: []

# Example 2: Load data from an Excel file
df_excel = pd.read_excel('data.xlsx', sheet_name='Sheet1')
print("Data loaded from Excel:\n", df_excel.head())

Data loaded from Excel:
    Unnamed: 0  Unnamed: 1  Unnamed: 2  Unnamed: 3 Unnamed: 4
0          NaN         NaN         NaN         NaN        NaN
1          NaN         NaN         NaN         NaN        NaN
2          NaN         NaN         NaN         NaN        NaN
3          NaN         NaN         NaN         NaN        NaN
4          NaN         NaN         NaN         NaN        NaN

# Example 3: Load data from a JSON file
df_json = pd.read_json('data.json')
print("Data loaded from JSON:\n", df_json.head())

Data loaded from JSON:
        name  age      city  isStudent  skills
```

```
0  John Doe   30  New York      False  Python
1  John Doe   30  New York      False    Java
2  John Doe   30  New York      False     SQL


# Example 4: Load data from a URL (CSV format)
url = 'https://people.sc.fsu.edu/~jburkardt/data/csv/airtravel.csv'
df_url = pd.read_csv(url)
print("Data loaded from URL:\n", df_url.head())

Data loaded from URL:
   Month   "1958"   "1959"   "1960"
0   JAN      340      360      417
1   FEB      318      342      391
2   MAR      362      406      419
3   APR      348      396      461
4   MAY      363      420      472


# Example 5: Load data from a text file with custom delimiters
df_txt = pd.read_csv('data.txt', delimiter='\t')
print("Data loaded from text file:\n", df_txt.head())

Data loaded from text file:
        hfguhdjvnjhfdvn
0  this is a text file



# just to check the file location of the jupiter notebook
import os
print(os.getcwd())

C:\Users\bharz



    Perform data cleaning and manipulation tasks.

# Example 1: Handling missing values
df = pd.DataFrame({'A': [1, 2, None], 'B': [4, None, 6]})
df_cleaned = df.fillna(0)  # Replace missing values with 0
print("DataFrame with missing values handled:\n", df_cleaned)

# Example 2: Drop missing values
df_dropped = df.dropna()  # Drop rows with missing values
print("Dropped missing values:\n", df_dropped)

# Example 3: Renaming columns
df_renamed = df.rename(columns={'A': 'Column_A', 'B': 'Column_B'})
print("Renamed columns:\n", df_renamed)
```

```
# Example 4: Filtering rows based on a condition
df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [25,
30, 35]})
df_filtered = df[df['Age'] > 30]
print("Filtered DataFrame:\n", df_filtered)

# Example 5: Adding a new column
df['Salary'] = [50000, 60000, 70000]
print("DataFrame with new column:\n", df)
```

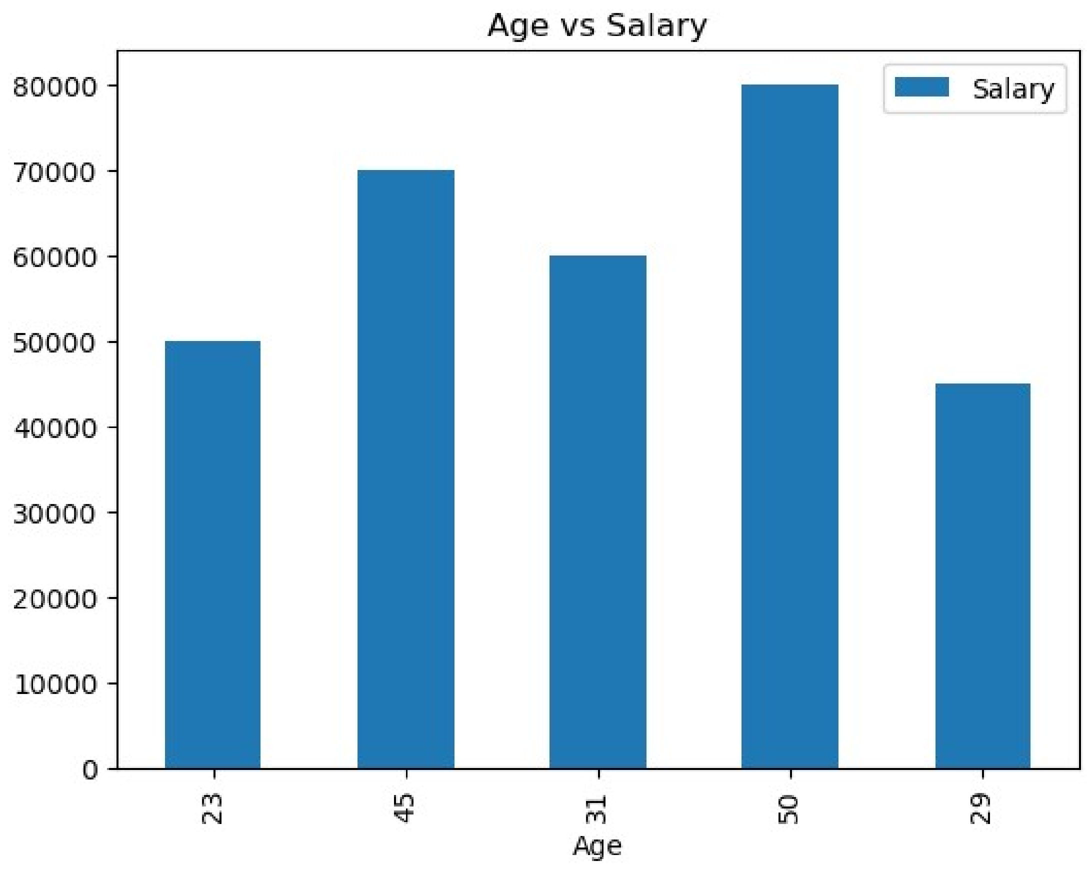
```
DataFrame with missing values handled:
      A    B
0  1.0  4.0
1  2.0  0.0
2  0.0  6.0
Dropped missing values:
      A    B
0  1.0  4.0
Renamed columns:
    Column_A  Column_B
0       1.0       4.0
1       2.0       NaN
2       NaN       6.0
Filtered DataFrame:
       Name  Age
2  Charlie   35
DataFrame with new column:
       Name  Age  Salary
0    Alice   25   50000
1      Bob   30   60000
2  Charlie   35   70000
```

Explore data analysis and visualization using Pandas.

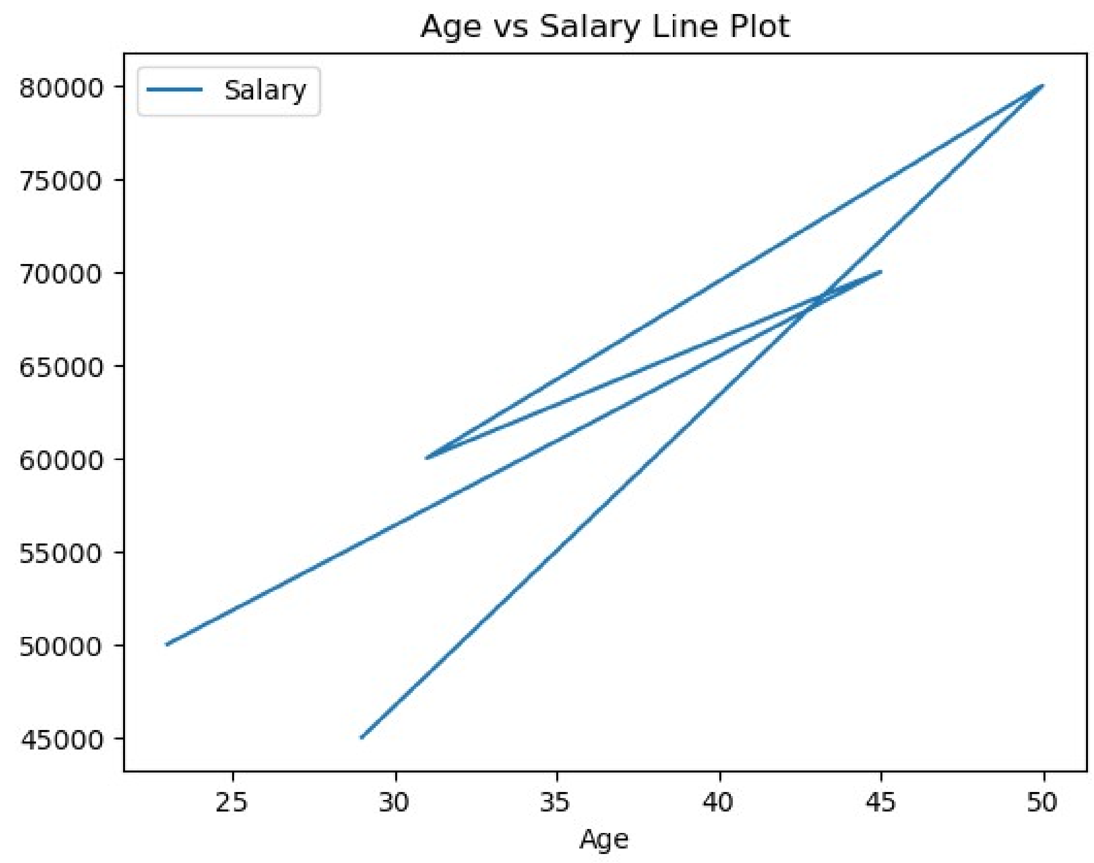
```
import matplotlib.pyplot as plt

# Example 1: Descriptive statistics
df = pd.DataFrame({'Age': [23, 45, 31, 50, 29], 'Salary': [50000,
70000, 60000, 80000, 45000]})
print("Descriptive statistics:\n", df.describe())
```

```
Descriptive statistics:
             Age        Salary
count   5.00000      5.000000
mean   35.60000  61000.000000
std    11.39298  14317.821063
min    23.00000  45000.000000
25%    29.00000  50000.000000
```

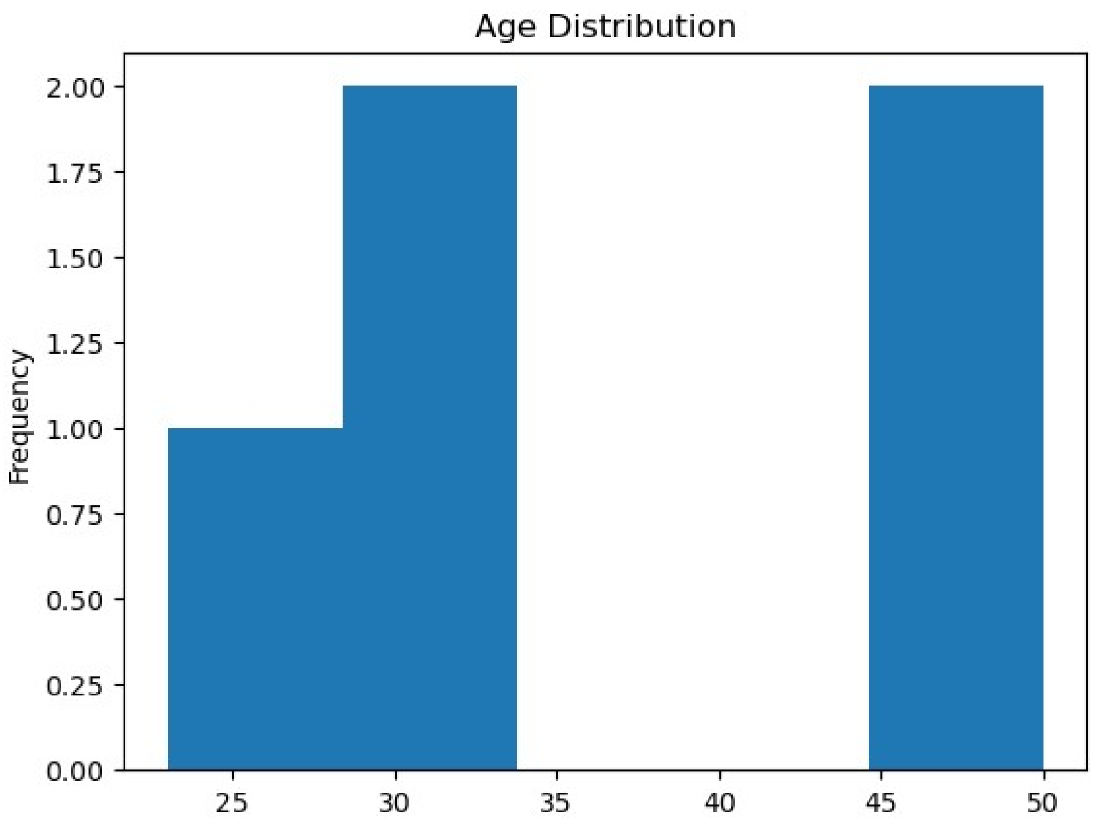
```
50%    31.00000  60000.000000
75%    45.00000  70000.000000
max    50.00000  80000.000000



# Example 3: Visualizing data with a bar plot
df.plot(kind='bar', x='Age', y='Salary')
plt.title('Age vs Salary')
plt.show()
```



```
# Example 4: Visualizing data with a line plot
df.plot(kind='line', x='Age', y='Salary')
plt.title('Age vs Salary Line Plot')
plt.show()
```



```python
# Example 5: Plotting a histogram
df['Age'].plot(kind='hist', bins=5)
plt.title('Age Distribution')
plt.show()
```



```
 #Creating a pivot table
df = pd.DataFrame({'Product': ['A', 'B', 'A', 'B'], 'Sales': [100,
200, 150, 250], 'Region': ['North', 'South', 'North', 'South']})
pivot_table = pd.pivot_table(df, values='Sales', index='Product',
columns='Region', aggfunc='sum')
print("Pivot Table:\n", pivot_table)

Pivot Table:
 Region   North  South
Product
A        250.0    NaN
B          NaN  450.0


 #Grouping data by multiple columns and calculating sum
df_grouped = df.groupby(['Product', 'Region']).sum()
print("Grouped DataFrame:\n", df_grouped)


Grouped DataFrame:
                 Sales
Product Region
A       North     250
B       South     450
```

```
# Grouping data by one column and counting occurrences
df_count = df.groupby('Product').size()
print("Count of occurrences by Product:\n", df_count)

Count of occurrences by Product:
 Product
A    2
B    2
dtype: int64
```

If Statements

```
     # Example 1: Basic if statement
x = 10
if x > 5:
    print("x is greater than 5")

# Example 2: if-else statement
x = 3
if x > 5:
    print("x is greater than 5")
else:
    print("x is less than or equal to 5")

# Example 3: if-elif-else statement
x = 7
if x > 10:
    print("x is greater than 10")
elif x > 5:
    print("x is greater than 5 but less than or equal to 10")
else:
    print("x is less than or equal to 5")

# Example 4: if statement with a string condition
name = "Alice"
if name == "Alice":
    print("Hello, Alice!")

#Example 5 :  Login Status Check
user_logged_in = False
admin_logged_in = True

if user_logged_in:
    print("Welcome, user!")
elif admin_logged_in:
    print("Welcome, admin!")
else:
    print("Please log in.")
```

```
x is greater than 5
x is less than or equal to 5
x is greater than 5 but less than or equal to 10
Hello, Alice!
Welcome, admin!
```

```python
# Example 1: Multiple conditions with logical AND
age = 25
income = 40000
if age > 18 and income > 30000:
    print("Eligible for loan")

# Example 2: Multiple conditions with logical OR
x = 5
if x < 0 or x > 10:
    print("x is outside the range 0-10")
else:
    print("x is within the range 0-10")

# Example 3: Using not operator in condition
is_sunny = False
if not is_sunny:
    print("It is not sunny today")

# Example 4: Combining multiple logical operators
x = 7
if (x > 5 and x < 10) or x == 15:
    print("x is between 5 and 10 or equal to 15")

# Example 5: Complex condition using comparison chaining
y = 15
if 10 < y < 20:
    print("y is between 10 and 20")
```

```
Eligible for loan
x is within the range 0-10
It is not sunny today
x is between 5 and 10 or equal to 15
y is between 10 and 20
```

```python
     # Example 1: Nested if statement (checking multiple conditions)
x = 20
if x > 10:
    print("x is greater than 10")
    if x > 15:
        print("x is also greater than 15")
    else:
        print("x is less than or equal to 15")
```

```python
# Example 2: Nested if-else statement (evaluating within another
condition)
age = 25
if age > 18:
    if age >= 21:
        print("You can legally drink alcohol")
    else:
        print("You are an adult but can't drink yet")
else:
    print("You are not an adult")

# Example 3: Nested conditions with multiple logical operators
x = 30
if x > 10:
    print("x is greater than 10")
    if x % 2 == 0:
        print("x is also even")

# Example 4: Nested if within an elif block
num = 50
if num < 30:
    print("num is less than 30")
elif num >= 30:
    print("num is greater than or equal to 30")
    if num == 50:
        print("num is exactly 50")

# Example 5: Deeply nested if conditions
marks = 85
if marks > 40:
    if marks >= 60:
        if marks >= 75:
            print("You passed with distinction")
        else:
            print("You passed with first class")
    else:
        print("You passed")
else:
    print("You failed")
```

```
x is greater than 10
x is also greater than 15
You can legally drink alcohol
x is greater than 10
x is also even
num is greater than or equal to 30
num is exactly 50
You passed with distinction
```

Loops

```
# Example 1: Iterating over a list
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)

# Example 2: Iterating over a range of numbers
for i in range(5):
    print(i)

# Example 3: Iterating over a string
word = "hello"
for letter in word:
    print(letter)

# Example 4: Iterating over a dictionary
person = {'name': 'Alice', 'age': 25}
for key, value in person.items():
    print(f"{key}: {value}")

# Example 5: Iterating over a list with index
numbers = [10, 20, 30]
for index, number in enumerate(numbers):
    print(f"Index: {index}, Number: {number}")
```

```
apple
banana
cherry
0
1
2
3
4
h
e
l
l
o
name: Alice
age: 25
Index: 0, Number: 10
Index: 1, Number: 20
Index: 2, Number: 30
```

```
# Example 1: Basic while loop
count = 0
```

```python
while count < 5:
    print(count)
    count += 1

# Example 2: While loop with a break condition
x = 0
while True:
    print(x)
    x += 1
    if x == 3:
        break

# Example 3: Using a while loop to prompt user input
user_input = ''
while user_input != 'exit':
    user_input = input("Type 'exit' to stop: ")

# Example 4: Counting down with a while loop
n = 5
while n > 0:
    print(n)
    n -= 1

# Example 5: While loop with a conditional check
balance = 100
while balance > 0:
    print(f"Balance: {balance}")
    balance -= 20
```

```
0
1
2
3
4
0
1
2

Type 'exit' to stop:  exit

5
4
3
2
1
Balance: 100
Balance: 80
Balance: 60
```

```
Balance: 40
Balance: 20
```

```
Implement nested loops.
```

```python
# Example 1: Nested for loops
for i in range(3):
    for j in range(2):
        print(f"i = {i}, j = {j}")
```

```
i = 0, j = 0
i = 0, j = 1
i = 1, j = 0
i = 1, j = 1
i = 2, j = 0
i = 2, j = 1
```

```python
# Example 2: Nested loops for multiplication table
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} x {j} = {i * j}")
```

```
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
```

```python
# Example 3: Nested loop with a list of lists
matrix = [[1, 2], [3, 4], [5, 6]]
for row in matrix:
    for element in row:
        print(element)
```

```
1
2
3
4
5
6
```

```python
# Example 4: Nested loop to print a triangle pattern
n = 5
for i in range(1, n + 1):
    for j in range(i):
        print('*', end="")
    print()
```

```
*
**
***
****
*****
```

```python
# Example 5: Nested loop with if condition inside
for i in range(1, 4):
    for j in range(1, 4):
        if i == j:
            print(f"{i} is equal to {j}")
```

```
1 is equal to 1
2 is equal to 2
3 is equal to 3
```

```
    Utilize break and continue statements.
```

```python
     # Example 1: Break statement in a loop
for i in range(5):
    if i == 3:
        break
    print(i)
```

```
0
1
2
```

```python
# Example 2: Continue statement in a loop
for i in range(5):
    if i == 3:
        continue
    print(i)
```

```
0
1
2
4
```

```python
# Example 3: Break statement in a while loop
n = 0
while n < 5:
    print(n)
    if n == 2:
        break
    n += 1

0
1
2

# Example 4: Continue statement in a while loop
n = 0
while n < 5:
    n += 1
    if n == 3:
        continue
    print(n)

1
2
4
5

# Example 5: Nested loop with break statement
for i in range(5):
    for j in range(5):
        if j == 2:
            break
        print(f"i = {i}, j = {j}")

i = 0, j = 0
i = 0, j = 1
i = 1, j = 0
i = 1, j = 1
i = 2, j = 0
i = 2, j = 1
i = 3, j = 0
i = 3, j = 1
i = 4, j = 0
i = 4, j = 1
```

Lists, Tuples, Sets, Dictionaries

```
                                        Create and Manipulate Lists


# Example 1: Create a list
fruits = ['apple', 'banana', 'cherry']
print(fruits)


['apple', 'banana', 'cherry']


# Example 2: Add an element to the list
fruits.append('orange')
print(fruits)

['apple', 'banana', 'cherry', 'orange']

# Example 3: Remove an element from the list
fruits.remove('banana')
print(fruits)


['apple', 'cherry', 'orange']


# Example 4: Indexing and slicing in lists
print(fruits[1])  # Access the second element
print(fruits[0:2])  # Slice first two elements

cherry
['apple', 'cherry']


# Example 5: Insert an element at a specific index
fruits.insert(1, 'mango')
print(fruits)

['apple', 'mango', 'cherry', 'orange']




                                      Built-in Methods for Lists




# Example 1: Sort the list
fruits.sort()
print(fruits)

# Example 2: Reverse the list
```

```
fruits.reverse()
print(fruits)

# Example 3: Pop an element (removes the last item by default)
popped_item = fruits.pop()
print(popped_item)
print(fruits)

# Example 4: Count occurrences of an element
count = fruits.count('apple')
print(f"Number of 'apple' in the list:", count)

# Example 5: Extend a list with another list
more_fruits = ['pineapple', 'grapes']
fruits.extend(more_fruits)
print(fruits)
```

```
['apple', 'cherry', 'mango', 'orange']
['orange', 'mango', 'cherry', 'apple']
apple
['orange', 'mango', 'cherry']
Number of 'apple' in the list: 0
['orange', 'mango', 'cherry', 'pineapple', 'grapes']
```

Create and Manipulate Tuples

```
# Example 1: Create a tuple
numbers = (10, 20, 30)
print(numbers)

# Example 2: Access elements in a tuple (indexing)
print(numbers[1])

# Example 3: Slicing a tuple
print(numbers[:2])

# Example 4: Concatenating tuples
new_tuple = numbers + (40, 50)
print(new_tuple)

# Example 5: Unpacking tuples
a, b, c = numbers
print(a, b, c)
```

```
(10, 20, 30)
20
(10, 20)
(10, 20, 30, 40, 50)
10 20 30


                                    2. Built-in Methods for Tuples

# Example 1: Get the length of a tuple
print(len(numbers))

# Example 2: Count occurrences of an element
print(numbers.count(20))

# Example 3: Find the index of an element
print(numbers.index(30))

# Example 4: Nested tuple access
nested_tuple = (1, (2, 3), 4)
print(nested_tuple[1][0])

# Example 5: Immutable nature of tuples (can't change values)
# numbers[0] = 100  # This would throw an error, since tuples are
immutable


3
1
2
2



                            Create and Manipulate Sets



# Example 1: Create a set
my_set = {1, 2, 3, 4}
print(my_set)

# Example 2: Add an element to the set
my_set.add(5)
print(my_set)

# Example 3: Remove an element from the set
my_set.remove(3)
print(my_set)

# Example 4: Check if an element is in the set
print(2 in my_set)
```

```
# Example 5: Set union and intersection
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print("Union:", set1.union(set2))
print("Intersection:", set1.intersection(set2))
```

```
{1, 2, 3, 4}
{1, 2, 3, 4, 5}
{1, 2, 4, 5}
True
Union: {1, 2, 3, 4, 5}
Intersection: {3}
```

Built-in Methods for Sets

```
# Example 1: Difference between sets
print(set1.difference(set2))  # Elements in set1 but not in set2

# Example 2: Symmetric difference (elements in either set1 or set2,
but not both)
print(set1.symmetric_difference(set2))

# Example 3: Check if one set is a subset of another
print(set1.issubset({1, 2, 3, 4}))

# Example 4: Discard an element (won't raise an error if the element
is not found)
my_set.discard(10)  # No error if 10 is not in the set
print(my_set)

# Example 5: Clear all elements in a set
my_set.clear()
print(my_set)  # Output: set()
```

```
{1, 2}
{1, 2, 4, 5}
True
{1, 2, 4, 5}
set()
```

Create and Manipulate Dictionaries

```
# Example 1: Create a dictionary
person = {'name': 'Alice', 'age': 25, 'city': 'New York'}
print(person)

# Example 2: Access values using keys
print(person['name'])

# Example 3: Add or update a key-value pair
person['job'] = 'Engineer'
print(person)

# Example 4: Remove a key-value pair
del person['age']
print(person)

# Example 5: Check if a key exists in a dictionary
print('name' in person)
```

```
{'name': 'Alice', 'age': 25, 'city': 'New York'}
Alice
{'name': 'Alice', 'age': 25, 'city': 'New York', 'job': 'Engineer'}
{'name': 'Alice', 'city': 'New York', 'job': 'Engineer'}
True
```

```
                                        2. Built-in Methods for
Dictionaries
```

```
# Example 1: Get all keys in a dictionary
print(person.keys())

# Example 2: Get all values in a dictionary
print(person.values())

# Example 3: Get all key-value pairs as tuples
print(person.items())

# Example 4: Use get method to retrieve a value (with a default if key
doesn't exist)
age = person.get('age', 'Not available')
print(age)

# Example 5: Iterate over dictionary key-value pairs
for key, value in person.items():
    print(f"{key}: {value}")
```

```
dict_keys(['name', 'city', 'job'])
dict_values(['Alice', 'New York', 'Engineer'])
dict_items([('name', 'Alice'), ('city', 'New York'), ('job',
'Engineer')])
Not available
name: Alice
city: New York
job: Engineer

#8.3 Perform operations like indexing, slicing, adding, removing
Elements.

#Lists
# Indexing and Slicing
my_list = ['a', 'b', 'c', 'd']
print(my_list[1])
print(my_list[1:3])

# Adding elements
my_list.append('e')
my_list.insert(2, 'z')
print(my_list)

# Removing elements
my_list.remove('b')
my_list.pop(2)
print(my_list)

#Tuples
# Indexing and Slicing
my_tuple = ('a', 'b', 'c', 'd')
print(my_tuple[1])
print(my_tuple[1:3])

# Tuples are immutable, so you cannot add or remove elements directly
# If you need to modify a tuple, you can convert it to a list first:
temp_list = list(my_tuple)
temp_list.append('e')
my_tuple = tuple(temp_list)
print(my_tuple)

#Sets
# Adding elements
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)

# Removing elements
my_set.remove(2)
print(my_set)
```

```
# No indexing or slicing since sets are unordered

#Dictionaries
# Adding and Accessing elements
my_dict = {'name': 'Alice', 'age': 25}
my_dict['city'] = 'New York'
print(my_dict['name'])

# Removing elements
del my_dict['age']
print(my_dict)

# No indexing or slicing since dictionaries use keys for access
```

```
b
['b', 'c']
['a', 'b', 'z', 'c', 'd', 'e']
['a', 'z', 'd', 'e']
b
('b', 'c')
('a', 'b', 'c', 'd', 'e')
{1, 2, 3, 4}
{1, 3, 4}
Alice
{'name': 'Alice', 'city': 'New York'}
```

```
    #8.4 Explore built-in methods for each data structure.
```

```
#Lists
my_list = [1, 2, 3, 4]

# append(): Adds an element to the end of the list
my_list.append(5)
print(my_list)  # Output: [1, 2, 3, 4, 5]

# extend(): Extend the list by appending elements from another list
my_list.extend([6, 7])
print(my_list)  # Output: [1, 2, 3, 4, 5, 6, 7]

# pop(): Removes and returns the last element (or the element at the
specified index)
removed_element = my_list.pop()
print(removed_element)  # Output: 7
print(my_list)  # Output: [1, 2, 3, 4, 5, 6]

# sort(): Sorts the list in ascending order
```

```
my_list.sort()
print(my_list)  # Output: [1, 2, 3, 4, 5, 6]

# reverse(): Reverses the order of the list
my_list.reverse()
print(my_list)  # Output: [6, 5, 4, 3, 2, 1]
```

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

```
#Tuples
my_tuple = (1, 2, 3, 2, 4)

# count(): Counts occurrences of an element
print(my_tuple.count(2))  # Output: 2

# index(): Returns the index of the first occurrence of an element
print(my_tuple.index(3))  # Output: 2

# Tuples have fewer methods since they are immutable, unlike lists
```

```
2
2
```

```
#Sets
my_set = {1, 2, 3, 4}

# add(): Adds an element to the set
my_set.add(5)
print(my_set)  # Output: {1, 2, 3, 4, 5}

# remove(): Removes an element from the set (raises an error if not
found)
my_set.remove(2)
print(my_set)  # Output: {1, 3, 4, 5}

# union(): Returns the union of two sets
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1.union(set2))  # Output: {1, 2, 3, 4, 5}

# intersection(): Returns the intersection of two sets
print(set1.intersection(set2))  # Output: {3}
```

```
# difference(): Returns the difference between two sets
print(set1.difference(set2))  # Output: {1, 2}
```

```
{1, 2, 3, 4, 5}
{1, 3, 4, 5}
{1, 2, 3, 4, 5}
{3}
{1, 2}
```

```
#Dictionaries
my_dict = {'name': 'Alice', 'age': 25}

# keys(): Returns all keys in the dictionary
print(my_dict.keys())  # Output: dict_keys(['name', 'age'])

# values(): Returns all values in the dictionary
print(my_dict.values())  # Output: dict_values(['Alice', 25])

# items(): Returns all key-value pairs in the dictionary
print(my_dict.items())  # Output: dict_items([('name', 'Alice'),
('age', 25)])

# get(): Returns the value for a key (returns None if key is not
found)
print(my_dict.get('name'))  # Output: 'Alice'
print(my_dict.get('city', 'Not Found'))  # Output: 'Not Found'

# update(): Updates the dictionary with another dictionary or key-
value pairs
my_dict.update({'city': 'New York'})
print(my_dict)  # Output: {'name': 'Alice', 'age': 25, 'city': 'New
York'}
```

```
dict_keys(['name', 'age'])
dict_values(['Alice', 25])
dict_items([('name', 'Alice'), ('age', 25)])
Alice
Not Found
{'name': 'Alice', 'age': 25, 'city': 'New York'}
```

Operators

```
# Example of arithmetic operators
a = 10
b = 5

# Addition
```

```python
print("Addition:", a + b)  # Output: 15

# Subtraction
print("Subtraction:", a - b)  # Output: 5

# Multiplication
print("Multiplication:", a * b)  # Output: 50

# Division
print("Division:", a / b)  # Output: 2.0

# Modulus (remainder of division)
print("Modulus:", a % b)  # Output: 0
```

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Modulus: 0
```

```python
# Example of comparison operators
x = 7
y = 10

# Greater than
print("x > y:", x > y)  # Output: False

# Less than
print("x < y:", x < y)  # Output: True

# Equal to
print("x == y:", x == y)  # Output: False

# Not equal to
print("x != y:", x != y)  # Output: True

# Greater than or equal to
print("x >= y:", x >= y)  # Output: False
```

```
x > y: False
x < y: True
x == y: False
x != y: True
x >= y: False
```

```python
# Example of logical operators
p = True
q = False

# Logical AND
print("p and q:", p and q)  # Output: False
```

```
# Logical OR
print("p or q:", p or q)  # Output: True

# Logical NOT
print("not p:", not p)  # Output: False
```

```
p and q: False
p or q: True
not p: False
```

```
# Example of assignment operators
z = 5

# Addition assignment
z += 3  # Equivalent to z = z + 3
print("z after += 3:", z)  # Output: 8

# Subtraction assignment
z -= 2  # Equivalent to z = z - 2
print("z after -= 2:", z)  # Output: 6

# Multiplication assignment
z *= 2  # Equivalent to z = z * 2
print("z after *= 2:", z)  # Output: 12

# Division assignment
z /= 4  # Equivalent to z = z / 4
print("z after /= 4:", z)  # Output: 3.0

# Modulus assignment
z %= 2  # Equivalent to z = z % 2
print("z after %= 2:", z)  # Output: 1.0
```

```
z after += 3: 8
z after -= 2: 6
z after *= 2: 12
z after /= 4: 3.0
z after %= 2: 1.0
```

```
#Examples of Operator Precedence:
# Example 1: Exponentiation has higher precedence than multiplication
result = 2 ** 3 * 4
print(result)

# Example 2: Parentheses have the highest precedence
result = (2 + 3) * 4
print(result)

# Example 3: Logical operators and comparison
result = (5 > 3) and (2 < 4) or not (3 == 3)
```

```
print(result)
#Using Parentheses to Override Precedence:
#You can use parentheses to explicitly specify the order of
operations, overriding the default precedence.
result = 2 + 3 * 4
print(result)

result = (2 + 3) * 4
print(result)
```

```
32
20
True
14
20
```

```
# Apply Operators in Expressions and Calculations
# Example 1: Combining arithmetic and assignment operators
x = 10
x += 5  # equivalent to x = x + 5
x *= 2  # equivalent to x = x * 2
print(x)
```

```
30
```

```
# Example 2: Applying logical and comparison operators
age = 20
income = 50000

is_eligible = (age > 18) and (income > 30000)
print(is_eligible)
```

```
True
```

```
# Example 3:Modulo and Exponentiation Operators
x = 7
y = 2

# Modulo (Remainder)
result_mod = x % y
# Exponentiation
result_exp = x ** y
print("Modulo:", result_mod)
print("Exponentiation:", result_exp)
```

```
Modulo: 1
Exponentiation: 49
```

```
# Example 4:Comparison Operators
```

```
a = 15
b = 10

# Greater than
print(a > b)
# Less than
print(a < b)
# Equal to
print(a == b)
# Not equal to
print(a != b)
```

```
True
False
False
True
```

```
# Example 5: Logical Operators
x = True
y = False

# AND operator
result_and = x and y
# OR operator
result_or = x or y
# NOT operator
result_not = not x

print("AND:", result_and)
print("OR:", result_or)
print("NOT:", result_not)
```

```
AND: False
OR: True
NOT: False
```

Reading CSV files

```
import pandas as pd

# Example 1: Basic CSV reading
df1 = pd.read_csv('data1.csv')
print(df1.head())
```

```
Empty DataFrame
Columns: [data1.csv  file]
Index: []
```

```
# Example 2: Reading a CSV file with specific column names
df2 = pd.read_csv('data2.csv', names=['A', 'B', 'C'])
print(df2.head())
```

```
       A      B     C
0   col1   col2  col3
1  43535  45345    54
2  43543     56    67
3   6746   5456   564
```

```
# Example 3: Reading a CSV from a URL
url = 'https://people.sc.fsu.edu/~jburkardt/data/csv/hw_200.csv'
df3 = pd.read_csv(url)
print(df3.head())
```

```
   Index   Height(Inches)"   "Weight(Pounds)"
0      1             65.78             112.99
1      2             71.52             136.49
2      3             69.40             153.03
3      4             68.22             142.34
4      5             67.79             144.30
```

```
# Example 4: Reading a CSV with index column
df4 = pd.read_csv('data2.csv', index_col=0)  # Setting the first
column as the index
print(df4.head())
```

```
        col2  col3
col1
43535  45345    54
43543     56    67
6746    5456   564
```

```
# Example 5: Reading a CSV with specific data types
df5 = pd.read_csv('data4.csv', dtype={'A': int, 'B': float})
print(df5.head())
```

```
   id month  storeid     passkey
0   1   may    33334  5678976543
1   2   jan    76879  4567899876
2   3   feb    38768  8976576333
3   4   dec    33445  2345565432
4   5   oct    66543  2345667765
```

```
    Explore different CSV reading options and parameters.
```

```
#Explore Different CSV Reading Options and Parameters
# Example 1: Reading a CSV with a different delimiter (semicolon-
separated)
df1 = pd.read_csv('data1.csv', delimiter=';')
print(df1.head())
```

```
Empty DataFrame
Columns: [data1.csv  file]
Index: []
```

```
# Example 2: Skipping a specific number of rows
df2 = pd.read_csv('data2.csv', skiprows=2)  # Skip first 2 rows
print(df2.head())
```

```
   43543    56   67
0   6746  5456  564
```

```
# Example 3: Reading only specific columns
df3 = pd.read_csv('data2.csv', usecols=['col1', 'col3'])
print(df3.head())
```

```
    col1  col3
0  43535    54
1  43543    67
2   6746   564
```

```
# Example 4: Reading CSV with custom NA values
df4 = pd.read_csv('data4.csv', na_values=['N/A', 'missing', '-'])
print(df4.head())
```

```
   id month  storeid     passkey
0   1   may    33334  5678976543
1   2   jan    76879  4567899876
2   3   feb    38768  8976576333
3   4   dec    33445  2345565432
4   5   oct    66543  2345667765
```

```
# Example 5: Reading a large CSV file in chunks
chunksize = 100
for chunk in pd.read_csv('data4.csv', chunksize=chunksize):
    print(chunk.head())
```

```
   id month  storeid     passkey
0   1   may    33334  5678976543
1   2   jan    76879  4567899876
2   3   feb    38768  8976576333
3   4   dec    33445  2345565432
4   5   oct    66543  2345667765
```

```
#Handle Missing Values and Data Cleaning

# Example 1: Checking for missing values
df1 = pd.read_csv('data4.csv')
print(df1.isnull().sum())  # Checking how many missing values each
column has

# Example 2: Filling missing values with a specific value
df2 = df1.fillna(0)  # Fill missing values with 0
print('missing values')
print(df2.head())


# Example 3: Dropping rows with missing values
df3 = df1.dropna()  # Drop rows with any missing values
print('dropping')
print(df3.head())

# Example 4: Replacing missing values with the mean of a column
df4 = df1.copy()
df4['storeid'] = df4['storeid'].fillna(df4['storeid'].mean())
print(df4.head())

# Example 5: Removing duplicate rows
df5 = pd.read_csv('data2.csv')
df5_cleaned = df5.drop_duplicates()
print(df5_cleaned.head())
```

```
id         0
month      0
storeid    0
passkey    0
dtype: int64
missing values
   id month  storeid     passkey
0   1   may    33334  5678976543
1   2   jan    76879  4567899876
2   3   feb    38768  8976576333
3   4   dec    33445  2345565432
4   5   oct    66543  2345667765
dropping
   id month  storeid     passkey
0   1   may    33334  5678976543
1   2   jan    76879  4567899876
2   3   feb    38768  8976576333
3   4   dec    33445  2345565432
4   5   oct    66543  2345667765
   id month  storeid     passkey
0   1   may    33334  5678976543
1   2   jan    76879  4567899876
```

```
2   3   feb    38768  8976576333
3   4   dec    33445  2345565432
4   5   oct    66543  2345667765
    col1   col2  col3
0  43535  45345    54
1  43543     56    67
2   6746   5456   564

                                 Python String Methods

#Manipulate Strings Using Various Built-in Methods

# Example 1: Replace a substring in a string
text = "Hello, World!"
new_text = text.replace("World", "Python")
print(new_text)  # Output: Hello, Python!

# Example 2: Join a list of strings into a single string
words = ['Python', 'is', 'awesome']
sentence = 'hey '.join(words)
print(sentence)  # Output: Python is awesome

# Example 3: Counting occurrences of a substring
text = "banana"
count = text.count('a')
print(count)  # Output: 3

# Example 4: Checking if a string starts with a specific substring
print(text.startswith('ban'))  # Output: True

# Example 5: Finding the position of a substring
position = text.find('ana')
print(position)  # Output: 1

Hello, Python!
Pythonhey ishey awesome
3
True
1

#Perform Operations Like Concatenation, Slicing, and Finding
Substrings

# Example 1: Concatenate two strings
str1 = "Hello"
str2 = "World"
result = str1 + " " + str2
print(result)  # Output: Hello World

# Example 2: Slice a string
text = "Python programming"
```

```python
sliced_text = text[0:6]  # Extract 'Python'
print(sliced_text)  # Output: Python

# Example 3: Find if a substring exists
print("programming" in text)  # Output: True

# Example 4: Get a substring from the end
last_word = text[-11:]
print(last_word)  # Output: programming

# Example 5: Extract every second character from a string
every_second_char = text[::2]
print(every_second_char)  # Output: Pto rgamn
```

```
Hello World
Python
True
programming
Pto rgamn
```

```python
#Convert Strings to Uppercase, Lowercase, and Title Case

# Example 1: Convert to uppercase
text = "hello world"
uppercase_text = text.upper()
print(uppercase_text)  # Output: HELLO WORLD

# Example 2: Convert to lowercase
lowercase_text = text.lower()
print(lowercase_text)  # Output: hello world

# Example 3: Convert to title case
title_text = text.title()
print(title_text)  # Output: Hello World

# Example 4: Swap case of a string (convert uppercase to lowercase and
vice versa)
swapped_case = text.swapcase()
print(swapped_case)  # Output: HELLO WORLD

# Example 5: Capitalize only the first letter of the string
capitalized_text = text.capitalize()
print(capitalized_text)  # Output: Hello world
```

```
HELLO WORLD
hello world
Hello World
HELLO WORLD
Hello world
```

```
#Remove Whitespace and Split Strings

# Example 1: Remove leading and trailing whitespace
text = "   Hello, World!   "
trimmed_text = text.strip()
print(trimmed_text)  # Output: Hello, World!

# Example 2: Remove only leading whitespace
leading_trimmed_text = text.lstrip()
print(leading_trimmed_text)  # Output: "Hello, World!   "

# Example 3: Remove only trailing whitespace
trailing_trimmed_text = text.rstrip()
print(trailing_trimmed_text)  # Output: "   Hello, World!"

# Example 4: Split a string into a list by spaces
text = "Python is awesome"
split_text = text.split()
print(split_text)  # Output: ['Python', 'is', 'awesome']

# Example 5: Split a string using a specific delimiter
csv_text = "apple,banana,cherry"
split_csv = csv_text.split(',')
print(split_csv)  # Output: ['apple', 'banana', 'cherry']
```

```
Hello, World!
Hello, World!
   Hello, World!
['Python', 'is', 'awesome']
['apple', 'banana', 'cherry']
```