In [ ]:

```
#My Jupiter Notebook
```

In [ ]:

```
#Python Functions
```

In [2]:

```
#1.Function to add two numbers
def add_numbers(a, b):
    return a + b
result = add_numbers(5, 7)
print(result)
```

```
12
```

In [3]:

```
#2.Function with default argument
def greet(name="Guest"):
    return f"Hello, {name}!"
print(greet())
print(greet("Vineela"))
```

```
Hello, Guest!
Hello, Vineela!
```

In [4]:

```
#3.Recursive function to calculate factorial
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n - 1)
print(factorial(5))
```

```
120
```

In [5]:

```
#4.Function scope example
def outer_function():
    x = "outer variable"

    def inner_function():
        nonlocal x  # Access the outer variable
        x = "inner variable"
        print("Inside inner function:", x)

    inner_function()
    print("Outside inner function:", x)

# Example usage
outer_function()
```

```
Inside inner function: inner variable
Outside inner function: inner variable
```

In [6]:

```
#5.Function with docstring
def multiply(a, b):
    return a * b
help(multiply)
```

```
Help on function multiply in module __main__:

multiply(a, b)
    #5.Function with docstring
```

In [ ]:

```
#Lambda Functions
```

In [7]:

```
#1.Basic Lambda function to add two numbers
add = lambda a, b: a + b
result = add(5, 3)
print(result)
```

```
8
```

In [10]:

```
#2.Using Lambda with map function to square each element in the list
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x ** 2, numbers))
print(squared_numbers)
```

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

In [11]:

```
#3.Using Lambda with filter function to filter out even numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)
```

```
[2, 4, 6, 8]
```

In [13]:

```
#4.Lambda Function with reduce (from functools)
from functools import reduce
numbers = [1, 2, 3, 4]
product = reduce(lambda x, y: x * y, numbers)
print(product)
```

```
24
```

In [14]:

```
#5.Regular function to multiply two numbers
def multiply(a, b):
    return a * b
multiply_lambda = lambda a, b: a * b
print(multiply(5, 4))
print(multiply_lambda(5, 4))
```

```
20
20
```

In [ ]:

```
#NumPy
```

In [16]:

```
#1.Creating NumPy Arrays (1D, 2D, 3D)
import numpy as np

arr_1d = np.array([1, 2, 3, 4])
print("1D Array:", arr_1d)

arr_2d = np.array([[1, 2], [3, 4]])
print("2D Array:\n", arr_2d)
```

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

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

 [[5 6]
  [7 8]]]
```

In [17]:

```
#2.Basic Arithmetic Operations on Arrays
# Create two arrays
arr1 = np.array([10, 20, 30, 40])
arr2 = np.array([1, 2, 3, 4])

# Arithmetic operations
print("Addition:", arr1 + arr2)
print("Subtraction:", arr1 - arr2)
print("Multiplication:", arr1 * arr2)
print("Division:", arr1 / arr2)
```

```
Addition: [11 22 33 44]
Subtraction: [ 9 18 27 36]
Multiplication: [ 10  40  90 160]
Division: [10. 10. 10. 10.]
```

In [18]:

```
#3.Indexing and Slicing Arrays
# Create a 2D array
arr = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])

# Accessing an element
print("Element at (1, 2):", arr[1, 2])

# Slicing a sub-array
print("First two rows:\n", arr[:2, :])
```

```
Element at (1, 2): 60
First two rows:
 [[10 20 30]
 [40 50 60]]
```

In [19]:

```
#4.Array Manipulation (reshape, transpose, concatenate)
# Reshape a 1D array into a 2D array
arr = np.array([1, 2, 3, 4, 5, 6])
reshaped_arr = arr.reshape((2, 3))
print("Reshaped Array:\n", reshaped_arr)

# Transpose of a matrix
transposed_arr = reshaped_arr.T
print("Transposed Array:\n", transposed_arr)

# Concatenate two arrays
arr1 = np.array([1, 2, 3])
```

```python
arr2 = np.array([4, 5, 6])
concatenated_arr = np.concatenate((arr1, arr2))
print("Concatenated Array:", concatenated_arr)
```

```
Reshaped Array:
 [[1 2 3]
 [4 5 6]]
Transposed Array:
 [[1 4]
 [2 5]
 [3 6]]
Concatenated Array: [1 2 3 4 5 6]
```

In [20]:

```python
#5.NumPy Random Number Generators
# Generate a random 1D array of 5 numbers
random_arr = np.random.rand(5)
print("Random Array:", random_arr)

# Generate a random integer between a range
random_int = np.random.randint(1, 10)
print("Random Integer:", random_int)

# Generate a random 2D array of integers
random_2d = np.random.randint(0, 10, size=(2, 3))
print("Random 2D Array:\n", random_2d)
```

```
Random Array: [0.84302159 0.87247151 0.43905443 0.11074141 0.57979368]
Random Integer: 8
Random 2D Array:
 [[8 9 8]
 [5 8 0]]
```

In [ ]:

```python
#Pandas
```

In [21]:

```python
#1.Creating Pandas Series and DataFrames
import pandas as pd

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

# Create a Pandas DataFrame from a dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 30, 35, 40],
        'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']}
df = pd.DataFrame(data)
print("\nPandas DataFrame:\n", df)
```

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

Pandas DataFrame:
       Name  Age         City
0    Alice   25     New York
1      Bob   30  Los Angeles
2  Charlie   35      Chicago
3    David   40      Houston
```

In [22]:

```python
#2.Load data from a CSV file
import pandas as pd
df_csv = pd.read_csv('accounts.csv')
print(df_csv.head())  # Display the first 5 rows
```

```
   account_id  customer_id account_type  balance
0           1           45      Savings  1000.50
1           2           12     Checking  2500.75
2           3           78      Savings  1500.00
3           4           34     Checking  3000.25
```

In [23]:

```python
#3.Performing Data Cleaning and Manipulation
# Creating a DataFrame with missing values
data_with_nan = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
                 'Age': [25, None, 35, 40],
                 'City': ['New York', None, 'Chicago', 'Houston']}
df_nan = pd.DataFrame(data_with_nan)

# Handling missing values
df_cleaned = df_nan.fillna("Unknown")  # Fill NaN values with "Unknown"
print("\nDataFrame after cleaning missing values:\n", df_cleaned)
```

```
DataFrame after cleaning missing values:
       Name      Age      City
0    Alice     25.0  New York
1      Bob  Unknown   Unknown
2  Charlie     35.0   Chicago
3    David     40.0   Houston
```

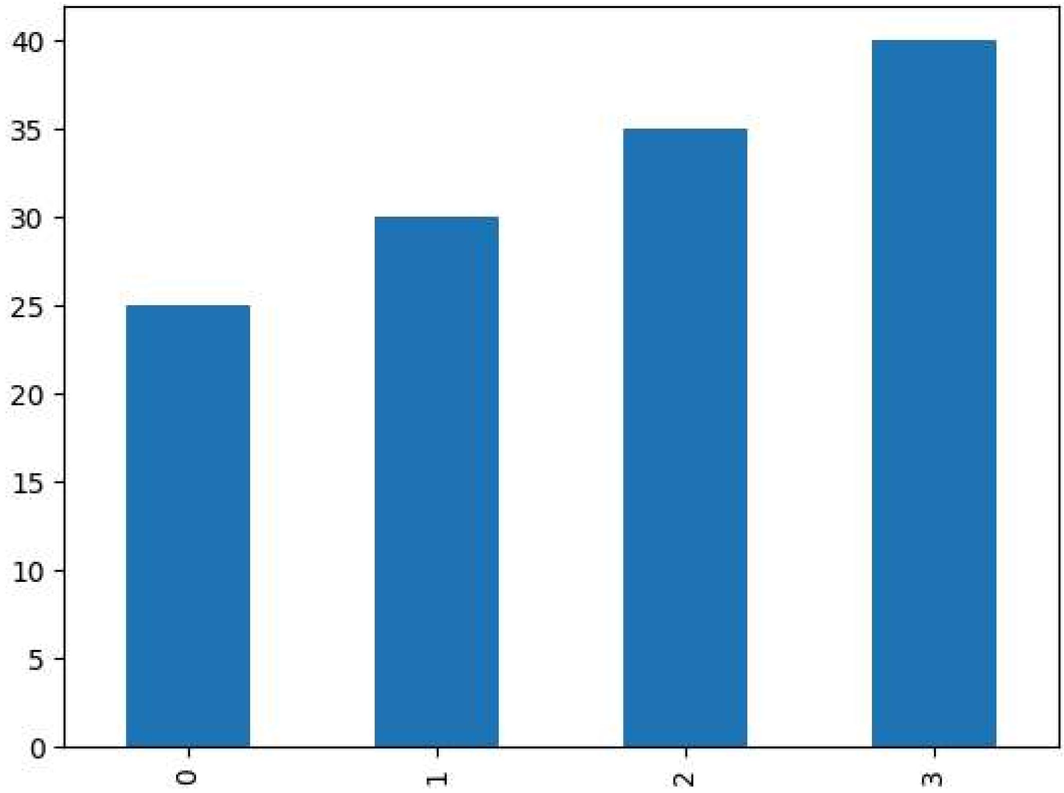
In [24]:

```python
#4.Exploring Data Analysis and Visualization
# Quick summary statistics of numerical columns
import pandas as pd;
data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 30, 35, 40],
        'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']}
df = pd.DataFrame(data)
print("\nSummary statistics:\n", df.describe())

# Plotting data (optional, requires matplotlib)
import matplotlib.pyplot as plt

df['Age'].plot(kind='bar')
plt.show()
```

```
Summary statistics:
              Age
count   4.000000
mean   32.500000
std     6.454972
min    25.000000
25%    28.750000
50%    32.500000
75%    36.250000
max    40.000000
```



```
In [25]: #5.Pivot Tables and Grouping Data
         # Group data by a column (e.g., City) and calculate the average Age
         grouped_df = df.groupby('City')['Age'].mean()
         print("\nAverage age by City:\n", grouped_df)

         # Creating a pivot table (useful for multi-dimensional data analysis)
         pivot_table = pd.pivot_table(df, values='Age', index='City', aggfunc='mean')
         print("\nPivot Table of average age by City:\n", pivot_table)
```

```
Average age by City:
 City
Chicago        35.0
Houston        40.0
Los Angeles    30.0
New York       25.0
Name: Age, dtype: float64

Pivot Table of average age by City:
               Age
City
Chicago      35.0
Houston      40.0
Los Angeles  30.0
New York     25.0
```

In [ ]:

```python
#If Statements
```

In [26]:

```python
#1.Example of a simple if statement
x = 10

if x > 5:
    print(f"{x} is greater than 5")
```

```
10 is greater than 5
```

In [27]:

```python
#2.Example of if-else statement
x = 3

if x > 5:
    print(f"{x} is greater than 5")
else:
    print(f"{x} is not greater than 5")
```

```
3 is not greater than 5
```

In [28]:

```python
#3.Example of if-elif-else statement
x = 7

if x > 10:
    print(f"{x} is greater than 10")
elif x > 5:
    print(f"{x} is greater than 5 but less than or equal to 10")
else:
    print(f"{x} is 5 or less")
```

```
7 is greater than 5 but less than or equal to 10
```

In [29]:

```python
#4.Example with complex conditions
x = 8
y = 15

if x > 5 and y < 20:
    print(f"Both conditions are True: x = {x}, y = {y}")
else:
    print("One or both conditions are False")
```

```
Both conditions are True: x = 8, y = 15
```

In [30]:

```
#5.Example of nested if statements
x = 12

if x > 10:
    print(f"{x} is greater than 10")
    if x % 2 == 0:
        print(f"{x} is also an even number")
else:
    print(f"{x} is not greater than 10")
```

```
12 is greater than 10
12 is also an even number
```

In [ ]:

```
#Loops
```

In [31]:

```
# 1.Example of a for loop iterating over a list
numbers = [1, 2, 3, 4, 5]

for num in numbers:
    print(num)
```

```
1
2
3
4
5
```

In [32]:

```
#2.Example of a while loop for Indefinite Iteration
x = 0

while x < 5:
    print(x)
    x += 1  # Increment x by 1
```

```
0
1
2
3
4
```

In [33]:

```
#3.Example of nested 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
```

In [34]:

```
#4.Example using break to exit a loop early
for num in range(10):
    if num == 5:
        break
    print(num)
```

```
0
1
2
3
4
```

In [35]:

```
#5.Example using continue to skip an iteration
for num in range(6):
    if num == 3:
        continue  # Skip the rest of the code for this iteration
    print(num)
```

```
0
1
2
4
5
```

In [ ]:

```
#Lists, Tuples, Sets, and Dictionaries
```

In [36]:

```
#1.Creating and Manipulating Lists
# Creating a list
fruits = ["apple", "banana", "cherry"]

# Accessing elements
print(fruits[1])  # Output: banana

# Modifying an element
fruits[1] = "blueberry"
print(fruits)  # Output: ['apple', 'blueberry', 'cherry']

# Adding an element
fruits.append("date")
print(fruits)  # Output: ['apple', 'blueberry', 'cherry', 'date']

# Removing an element
fruits.remove("cherry")
print(fruits)  # Output: ['apple', 'blueberry', 'date']
```

```
banana
['apple', 'blueberry', 'cherry']
['apple', 'blueberry', 'cherry', 'date']
['apple', 'blueberry', 'date']
```

In [37]:

```
#2.Creating and Manipulating Tuples
# Creating a tuple
coordinates = (10, 20, 30)

# Accessing elements
print(coordinates[0])  # Output: 10

# Tuples are immutable, so the following line would raise an error:
# coordinates[0] = 40  # This will raise a TypeError
```

```
10
```

In [38]:

```
#3.Creating and Manipulating Sets
# Creating a set
numbers = {1, 2, 3, 4}

# Adding an element
numbers.add(5)
print(numbers)  # Output: {1, 2, 3, 4, 5}

# Removing an element
numbers.remove(3)
print(numbers)  # Output: {1, 2, 4, 5}

# Set operations (union, intersection)
set1 = {1, 2, 3}
set2 = {3, 4, 5}

union = set1.union(set2)
intersection = set1.intersection(set2)

print("Union:", union)  # Output: {1, 2, 3, 4, 5}
print("Intersection:", intersection)  # Output: {3}
```

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

In [39]:

```
#4.Creating and Manipulating Dictionaries
# Creating a dictionary
person = {"name": "John", "age": 30, "city": "New York"}

# Accessing values by key
print(person["name"])  # Output: John

# Modifying a value
person["age"] = 31
print(person)  # Output: {'name': 'John', 'age': 31, 'city': 'New York'}

# Adding a new key-value pair
person["job"] = "Engineer"
print(person)  # Output: {'name': 'John', 'age': 31, 'city': 'New York', 'job': 'En

# Removing a key-value pair
del person["city"]
print(person)  # Output: {'name': 'John', 'age': 31, 'job': 'Engineer'}
```

```
John
{'name': 'John', 'age': 31, 'city': 'New York'}
{'name': 'John', 'age': 31, 'city': 'New York', 'job': 'Engineer'}
{'name': 'John', 'age': 31, 'job': 'Engineer'}
```

In [ ]:

```
#Operators
```

In [40]:

```
#1.Arithmetic operations
a = 10
```

```
b = 3

print("Addition:", a + b)       # Output: 13
print("Subtraction:", a - b)    # Output: 7
print("Multiplication:", a * b) # Output: 30
print("Division:", a / b)       # Output: 3.33
print("Floor Division:", a // b) # Output: 3
print("Modulus:", a % b)        # Output: 1
print("Exponentiation:", a ** b) # Output: 1000
```

```
Addition: 13
Subtraction: 7
Multiplication: 30
Division: 3.3333333333333335
Floor Division: 3
Modulus: 1
Exponentiation: 1000
```

In [41]:

```
#2.Comparison operations
x = 5
y = 10

print("Equal:", x == y)          # Output: False
print("Not Equal:", x != y)      # Output: True
print("Greater than:", x > y)    # Output: False
print("Less than:", x < y)       # Output: True
print("Greater than or equal:", x >= y) # Output: False
print("Less than or equal:", x <= y)    # Output: True
```

```
Equal: False
Not Equal: True
Greater than: False
Less than: True
Greater than or equal: False
Less than or equal: True
```

In [42]:

```
#3.Logical operations
x = True
y = False

print("Logical AND:", x and y)  # Output: False
print("Logical OR:", x or y)    # Output: True
print("Logical NOT:", not x)    # Output: False
```

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

In [43]:

```
#4.Assignment operations
x = 10
x += 5  # Equivalent to x = x + 5
print("x after += 5:", x)  # Output: 15

x *= 2  # Equivalent to x = x * 2
print("x after *= 2:", x)  # Output: 30
```

```
x after += 5: 15
x after *= 2: 30
```

In [44]:

```
#5.Operator precedence
result = 10 + 5 * 2  # Multiplication happens first
print("Result of 10 + 5 * 2:", result)  # Output: 20

result = (10 + 5) * 2  # Parentheses change the order
print("Result of (10 + 5) * 2:", result)  # Output: 30
```

```
Result of 10 + 5 * 2: 20
Result of (10 + 5) * 2: 30
```

In [ ]:

```
#Reading CSV Files
```

In [45]:

```
#1.Reading a CSV File into a Pandas DataFrame
import pandas as pd

# Reading a CSV file into a DataFrame (assuming 'data.csv' exists in the same direc
df = pd.read_csv('accounts.csv')

# Display the first 5 rows of the DataFrame
print(df.head())
```

```
   account_id  customer_id account_type  balance
0           1           45      Savings  1000.50
1           2           12     Checking  2500.75
2           3           78      Savings  1500.00
3           4           34     Checking  3000.25
```

In [46]:

```
#2.Reading a CSV File with Specific Parameters
import pandas as pd
# Reading a CSV file with a custom delimiter and skipping the first row
df = pd.read_csv('accounts.csv', delimiter=';', skiprows=1)

# Display the first 5 rows
print(df.head())
```

```
     1,45,Savings,1000.5
0  2,12,Checking,2500.75
1      3,78,Savings,1500
2  4,34,Checking,3000.25
```

In [47]:

```
#3.Handling Missing Values in a CSV File
import pandas as pd

# Reading a CSV file and handling missing values
df = pd.read_csv('accounts.csv', na_values=["?", "N/A", "null"])

# Filling missing values with a default value
df_filled = df.fillna(0)
print(df_filled.head())

# Dropping rows with missing values
df_dropped = df.dropna()
print(df_dropped.head())
```

```
   account_id  customer_id account_type  balance
0           1           45      Savings  1000.50
1           2           12     Checking  2500.75
2           3           78      Savings  1500.00
3           4           34     Checking  3000.25
   account_id  customer_id account_type  balance
0           1           45      Savings  1000.50
1           2           12     Checking  2500.75
2           3           78      Savings  1500.00
3           4           34     Checking  3000.25
```

In [48]:

```
#4.Specifying Column Types
import pandas as pd
# Specifying data types for columns when reading a CSV file
df = pd.read_csv('accounts.csv', dtype={'column_name': 'int32', 'another_column': '

# Display the data types of the columns
print(df.dtypes)
```

```
account_id        int64
customer_id       int64
account_type     object
balance         float64
dtype: object
```

In [51]:

```
#5.Writing a DataFrame to a new CSV file
df.to_csv('accounts.csv', index=False)

# 'index=False' prevents the index column from being written to the CSV file
```

In [ ]:

```
#Python String Methods
```

In [52]:

```
#1.Concatenating strings
str1 = "Hello"
str2 = "World"
result = str1 + " " + str2
print(result)  # Output: Hello World

# Joining a list of strings
words = ["Python", "is", "fun"]
sentence = " ".join(words)
print(sentence)  # Output: Python is fun
```

```
Hello World
Python is fun
```

In [53]:

```
#2.Slicing a string
text = "Hello, World!"
print(text[0:5])   # Output: Hello
print(text[7:])    # Output: World!
print(text[-6:])   # Output: World!
```

```
Hello
World!
World!
```

In [54]:

```
#3.Changing Case (Upper, Lower, Title)
text = "python programming"

# Convert to uppercase
print(text.upper())  # Output: PYTHON PROGRAMMING

# Convert to lowercase
print(text.lower())  # Output: python programming

# Convert to title case
print(text.title())  # Output: Python Programming
```

```
PYTHON PROGRAMMING
python programming
Python Programming
```

In [55]:

```
#4.Removing Whitespace and Stripping
text = "   Hello, World!   "

# Remove leading and trailing whitespace
print(text.strip())  # Output: Hello, World!

# Remove only leading whitespace
print(text.lstrip())  # Output: Hello, World!

# Remove only trailing whitespace
print(text.rstrip())  # Output:    Hello, World!
```

```
Hello, World!
Hello, World!
   Hello, World!
```

In [56]:

```
#5.Finding Substrings
text = "Hello, World! Hello again!"

# Find the position of the first occurrence of 'World'
print(text.find("World"))  # Output: 7

# Count occurrences of 'Hello'
print(text.count("Hello"))  # Output: 2
```

```
7
2
```

In [57]:

```
#5.Splitting and Replacing Strings
text = "apple,banana,cherry"

# Split the string into a list
fruits = text.split(",")
print(fruits)  # Output: ['apple', 'banana', 'cherry']

# Replace part of a string
new_text = text.replace("banana", "grape")
print(new_text)  # Output: apple,grape,cherry
```

```
['apple', 'banana', 'cherry']
apple,grape,cherry
```