# TAKSHASHILA UNIVERSITY

## Tamil Nadu State Private University

(Established under Tamil Nadu Private Universities Act 2019
& Recognized by UGC u/s 2(f) of the UGC Act, 1956)
Ongur, Tindivanam Taluk, Villupuram District, Tamil Nadu - 604305

TAKSHASHILA
UNIVERSITY

## FACULTY OF SCIENCES

## SCHOOL OF COMPUTER SCIENCE

**STUDENT RECORD**

**REGISTER NUMBER** :

**NAME OF THE STUDENT** :

**PROGRAM NAME** :

**YEAR / SEMESTER** :

**COURSE CODE** :

**COURSE NAME** :

# TAKSHASHILA UNIVERSITY

## Tamil Nadu State Private University

(Established under Tamil Nadu Private Universities Act 2019
& Recognized by UGC u/s 2(f) of the UGC Act, 1956)
Ongur, Tindivanam Taluk, Villupuram District, Tamil Nadu – 604305

**FACULTY OF SCIENCES**

## School of Computer Science

# Bonafide Certificate

| | |
|---|---|
| **Register Number** | |

*This is to certify that this is a Bonafide Record of practical work done by Mr./Ms. ……………………………….…... a Student of ……………………………….…... in School of Computer Science, has successfully completed the……………………………….…...……………………………….…...…………….…... Laboratory during the academic year ……………………………..……*

*Signature of Subject Staff* *Signature of School In-charge*

Submitted the University Practical Examination held on ………………………….

Signature Internal Examiner Signature External Examiner

# INDEX

| PRACTICAL- 1 | Program to Perform Arithmetic Operations on Two Numbers |
|---|---|
| | |

**AIM:**

To write a Python program to input two numbers and perform arithmetic operations.

**PROCEDURE:**

Step 1: Input two numbers from the user.

Step 2: Perform arithmetic operations ( +, –, *, /, //, % ).

Step 3: Store the results.

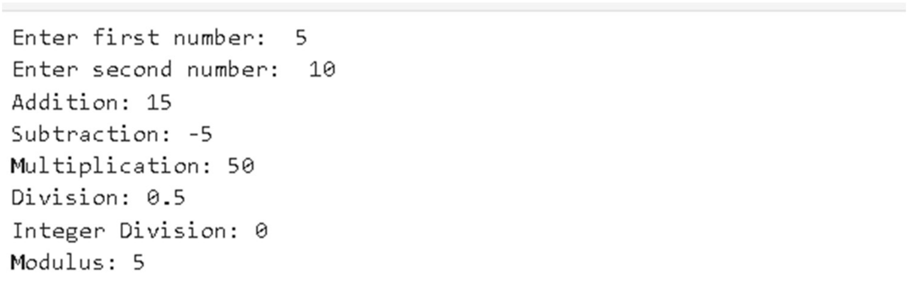
Step 4: Display all the operation results.

Step 5: End the program.

**CODE:**

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))

print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)
print("Integer Division:", a // b)
print("Modulus:", a % b)
```

## OUTPUT



```
Enter first number:  5
Enter second number:  10
Addition: 15
Subtraction: -5
Multiplication: 50
Division: 0.5
Integer Division: 0
Modulus: 5
```

## RESULT

The program is successfully executed and verified .

| | |
|---|---|
| **PRACTICAL- 2** | **Program to Concatenate Multiple Dictionaries in Python** |
| | |

**AIM**

To write a Python script to concatenate multiple dictionaries.

**PROCEDURE**

Step 1: Create multiple dictionaries.

Step 2: Use dictionary unpacking or update() to merge them.

Step 3: Store the merged dictionary.

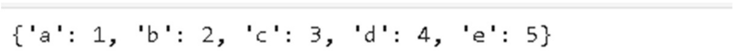
Step 4: Display the final dictionary.

Step 5: End the script.

**CODE**

```
dict1 = {"a": 1, "b": 2}
dict2 = {"c": 3, "d": 4}
dict3 = {"e": 5}

new_dict = {**dict1, **dict2, **dict3}
print(new_dict)
```

**OUTPUT**



```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

**RESULT**

The program is successfully executed and verified

| | |
|---|---|
| **PRACTICAL- 3** | **Program to Concatenate Two Lists in Python** |
| | |

**AIM**

To concatenate two lists using Python programming.

**PROCEDURE**

Step 1: Create the first list.

Step 2: Create the second list.

Step 3: Use + operator to concatenate both lists.

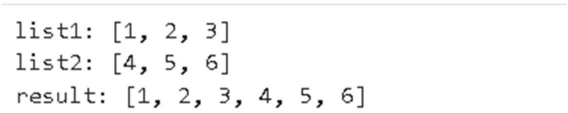
Step 4: Print the combined list.

Step 5: End the program.

**CODE**

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
print("list1:",list1)
print("list2:",list2)
result = list1 + list2
print("result:",result)
```

## OUTPUT



```
list1: [1, 2, 3]
list2: [4, 5, 6]
result: [1, 2, 3, 4, 5, 6]
```

## RESULT

The program is successfully executed and verified

| | |
|---|---|
| **PRACTICAL- 4** | **Program to Create a Pandas DataFrame from a NumPy Array** |
| | |

## AIM

To create a Pandas DataFrame from a NumPy array with custom index and column names.

## PROCEDURE

Step 1: Import NumPy and Pandas.

Step 2: Create a NumPy array.

Step 3: Define row index and column names.

Step 4: Convert the array to a DataFrame.

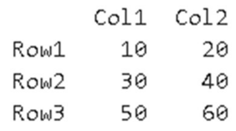
Step 5: Print the DataFrame.

Step 6: End the program.

## CODE

```
import numpy as np
import pandas as pd

arr = np.array([[10, 20], [30, 40], [50, 60]])
df = pd.DataFrame(arr, index=["Row1", "Row2", "Row3"], columns=["Col1", "Col2"])
print(df)
```

## OUTPUT



```
      Col1  Col2
Row1    10    20
Row2    30    40
Row3    50    60
```

## RESULT

The program is successfully executed and verified

| **PRACTICAL- 5** | **Program to Sort a Dictionary by Its Values (Ascending & Descending)** |
|---|---|
| | |

## AIM

To sort a dictionary based on its values in ascending and descending order.

## PROCEDURE

Step 1: Create a dictionary with values.

Step 2: Sort it in ascending order using sorted().

Step 3: Sort it in descending order.

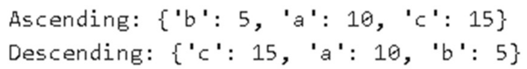
Step 4: Display both results.

Step 5: End the script.

## CODE

```
data = {"a": 10, "b": 5, "c": 15}

asc = dict(sorted(data.items(), key=lambda x: x[1]))
desc = dict(sorted(data.items(), key=lambda x: x[1], reverse=True))

print("Ascending:", asc)
print("Descending:", desc)
```

**OUTPUT**



```
Ascending: {'b': 5, 'a': 10, 'c': 15}
Descending: {'c': 15, 'a': 10, 'b': 5}
```

**RESULT**

The program is successfully executed and verified

| | |
|---|---|
| **PRACTICAL- 6** | **Program to Perform Indexing, Manipulation, and Concatenation in Pandas DataFrames** |
| | |

### AIM

To perform indexing, manipulation, and concatenation operations on Pandas DataFrames.

### PROCEDURE

Step 1: Import Pandas library.

Step 2: Create two DataFrames.

Step 3: Perform indexing (rows & columns).

Step 4: Concatenate DataFrames using pd.concat().

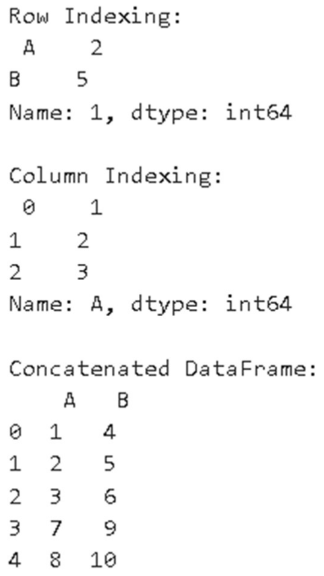
Step 5: Display results.

Step 6: End the program.

### CODE

```
import pandas as pd

df1 = pd.DataFrame({"A":[1,2,3], "B":[4,5,6]})
df2 = pd.DataFrame({"A":[7,8], "B":[9,10]})

print("Row Indexing:\n", df1.iloc[1])
print("\nColumn Indexing:\n", df1["A"])

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

**OUTPUT:**



```
Row Indexing:
 A    2
B    5
Name: 1, dtype: int64

Column Indexing:
 0    1
1    2
2    3
Name: A, dtype: int64

Concatenated DataFrame:
    A   B
0  1   4
1  2   5
2  3   6
3  7   9
4  8  10
```

**RESULT**

The program is successfully executed and verified

| PRACTICAL- 7 | **Program to Perform Arithmetic and Slicing Operations Using NumPy Arrays** |
|---|---|
| | |

### AIM

To apply arithmetic operations and slicing techniques on NumPy arrays.

### PROCEDURE

Step 1: Import NumPy.

Step 2: Create a NumPy array.

Step 3: Perform arithmetic operations.

Step 4: Apply slicing using index ranges.

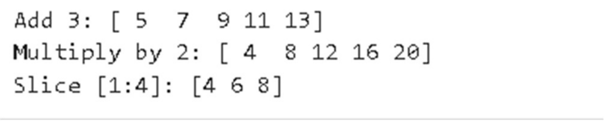
Step 5: Display the results.

Step 6: End the program.

### CODE

```
import numpy as np

arr = np.array([2, 4, 6, 8, 10])

print("Add 3:", arr + 3)
print("Multiply by 2:", arr * 2)
print("Slice [1:4]:", arr[1:4])
```

## OUTPUT



```
Add 3: [ 5  7  9 11 13]
Multiply by 2: [ 4  8 12 16 20]
Slice [1:4]: [4 6 8]
```

## RESULT

The program is successfully executed and verified

| PRACTICAL- 8 | **Program to Implement Merge Operations (Inner, Outer, Left, Right Join) in Pandas** |
|---|---|
|  |  |

**AIM**

To implement merge operations such as inner, outer, left, and right join using Pandas.

**PROCEDURE**

Step 1: Import Pandas.

Step 2: Create two DataFrames with a common column.

Step 3: Perform inner join.

Step 4: Perform outer, left, and right joins.

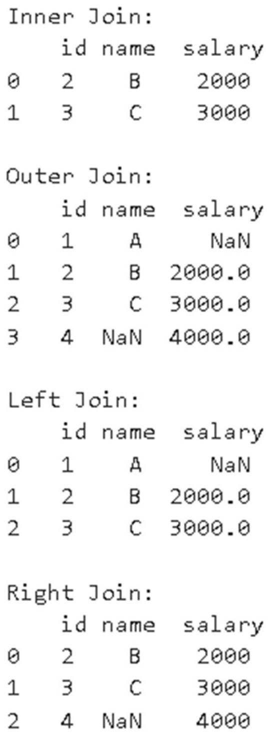
Step 5: Display all merged outputs.

Step 6: End the program.

**CODE**

```
import pandas as pd


df1 = pd.DataFrame({"id":[1,2,3], "name":["A","B","C"]})
df2 = pd.DataFrame({"id":[2,3,4], "salary":[2000,3000,4000]})


print("Inner Join:\n", pd.merge(df1, df2, on="id", how="inner"))
print("\nOuter Join:\n", pd.merge(df1, df2, on="id", how="outer"))
print("\nLeft Join:\n", pd.merge(df1, df2, on="id", how="left"))
print("\nRight Join:\n", pd.merge(df1, df2, on="id", how="right"))
```

## OUTPUT



```
Inner Join:
   id name  salary
0   2    B    2000
1   3    C    3000

Outer Join:
   id name  salary
0   1    A     NaN
1   2    B  2000.0
2   3    C  3000.0
3   4  NaN  4000.0

Left Join:
   id name  salary
0   1    A     NaN
1   2    B  2000.0
2   3    C  3000.0

Right Join:
   id name  salary
0   2    B    2000
1   3    C    3000
2   4  NaN    4000
```

## RESULT

The program is successfully executed and verified

| | |
|---|---|
| **PRACTICAL- 9** | **Program for Data Visualization Using Matplotlib and Seaborn (Histogram, KDE, Boxplot, Countplot, Heatmap, etc.)** |
| | |

**AIM**

To visualize data using Matplotlib and Seaborn through various plots without using external datasets.

**PROCEDURE**

Step 1: Import Matplotlib, Seaborn, and Pandas.

Step 2: Create a small sample dataset manually.

Step 3: Convert it to a DataFrame.

Step 4: Plot all required graphs (Histogram, Boxplot, KDE, Strip, Pair, Count, Heatmap).

Step 5: Show the plots.

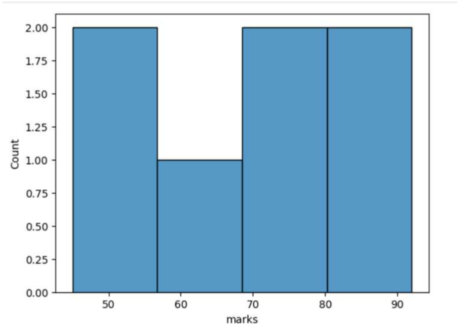
Step 6: End the program.

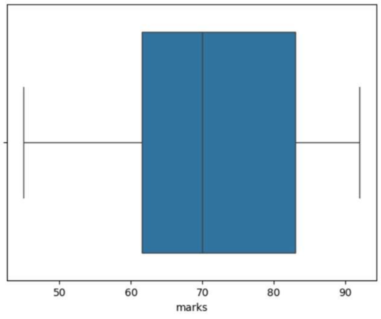
**CODE**

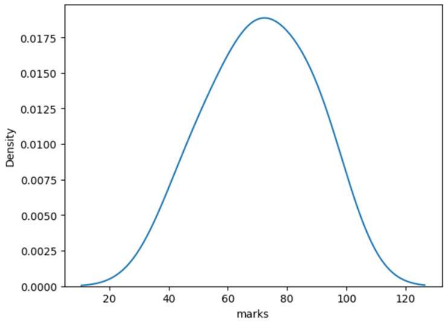
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Manual sample dataset
data = pd.DataFrame({
    "marks": [45, 67, 89, 70, 56, 92, 77],
    "age":   [18, 19, 18, 20, 19, 21, 18],
    "gender": ["M","F","M","F","M","F","M"]
})
```

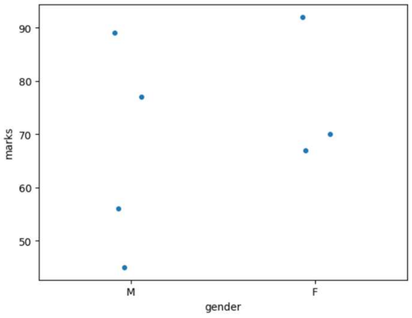
```
plt.figure(); sns.histplot(data["marks"]); plt.show()
plt.figure(); sns.boxplot(x=data["marks"]); plt.show()
plt.figure(); sns.kdeplot(data["marks"]); plt.show()
plt.figure(); sns.stripplot(x="gender", y="marks", data=data); plt.show()
plt.figure(); sns.pairplot(data); plt.show()
plt.figure(); sns.countplot(x="gender", data=data); plt.show()

# FIXED Heatmap
plt.figure()
numeric_data = data.select_dtypes(include=['number'])
sns.heatmap(numeric_data.corr(), annot=True)
plt.show()
```

**OUTPUT**

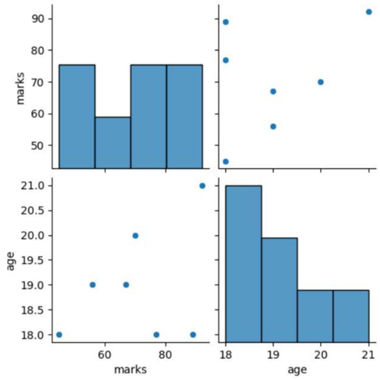


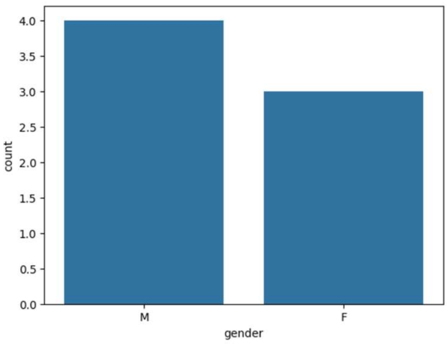


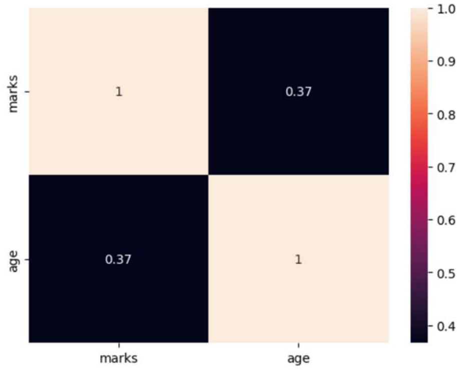




```
<Figure size 640x480 with 0 Axes>
```







## RESULT

The program is successfully executed and verified