Program No.: 2.4

Aim:
Demonstrate error handling in Python using the try-except block to handle division by zero.

Software used:
Colab (or) Jupyter Notebook

Description:
This program takes two numbers (numerator and denominator) from the user and attempts to perform division. If the denominator is zero, a ZeroDivisionError is raised. Using the try-except block, the program gracefully handles this error by printing an error message instead of crashing.

SOURCE CODE:

```
def safe division():
  """
  Demonstrates error handling for division by zero.
  """
  try:
    numerator = float(input("Enter numerator: "))
    denominator = float(input("Enter denominator: "))
    result = numerator / denominator
    print(f"Result: {result:.2f}")
  except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
  except ValueError:
    print("Invalid input. Please enter numerical values only.")

# Run the function
safe_division()
```


PROCEDURE:

1. Prompt the user to enter a numerator and denominator.
2. Try to divide the numerator by the denominator.

3. If denominator is zero, catch the ZeroDivisionError and display an error message.
4. If the input is not numeric, catch the ValueError and display a message.
5. Otherwise, display the result rounded to 2 decimal places.

INPUT (Case 1):

Enter numerator: 10

Enter denominator: 2

EXPECTED / ACTUAL OUTPUT (Case 1):

Result: 5.00

INPUT (Case 2):

Enter numerator: 10

Enter denominator: 0

EXPECTED / ACTUAL OUTPUT (Case 2):

Error: Division by zero is not allowed.

Experiment 3

Program No.: 3.2

Aim:

Demonstrate the use of NumPy methods for array creation, element-wise operations, comparisons, and conversions.

Software used: Colab (or) Jupyter Notebook

Description:

This program demonstrates element-wise operations, comparisons, array creation, and conversion between NumPy arrays and Python lists.

SOURCE CODE:

```
import numpy as np

# Get help on add
help(np.add)

# Check if none are zero
arr = np.array([1, 2, 3, 4, 5])
print("Array:", arr)
print("None of the elements is zero:", np.all(arr))

# Comparisons
a = np.array([1, 2, 3, 4])
b = np.array([2, 2, 1, 4])
print("Greater:", np.greater(a, b))
print("Greater or Equal:", np.greater_equal(a, b))
print("Less:", np.less(a, b))
print("Less or Equal:", np.less_equal(a, b))
print("Equal:", np.equal(a, b))
print("Allclose:", np.allclose(a, b))

# Create arrays
zeros_array = np.zeros(5)
ones_array = np.ones(5)
linspace_array = np.linspace(0, 10, 5)
print("Zeros array:", zeros_array)
print("Ones array:", ones_array)
print("Linspace array:", linspace_array)
```

```
# Convert to list
array_list = a.tolist()
print("Array converted to list:", array_list)
```

PROCEDURE:

1. Import NumPy.
2. Use help() for np.add.
3. Check all non-zero elements.
4. Perform comparisons.
5. Create zeros, ones, linspace arrays.
6. Convert array to list.

INPUT:

Predefined arrays in code.

EXPECTED / ACTUAL OUTPUT:

```
Array: [1 2 3 4 5]
None of the elements is zero: True

Greater: [False False  True False]
Greater or Equal: [False  True  True  True]
Less: [ True False False False]
Less or Equal: [ True  True False  True]
Equal: [False  True False  True]
Allclose: False

Zeros array: [0. 0. 0. 0. 0.]
Ones array: [1. 1. 1. 1. 1.]
Linspace array: [ 0.   2.5  5.   7.5 10. ]
Array converted to list: [1, 2, 3, 4]
```

Program No.: 3.2

Aim:

Demonstrate NumPy methods for extracting, analyzing, and summarizing array data.

Software used: Colab (or) Jupyter Notebook

Description:

This program demonstrates extracting elements based on a condition, finding max/min values, counting, and unique elements.

SOURCE CODE:

```
import numpy as np

arr = np.array([1, 3, 5, 7, 3, 5, 9, 2, 5])
num = 4

less_than_num = arr[arr < num]
greater_than_num = arr[arr > num]
print("Original array:", arr)
print("Numbers less than 4:", less_than_num)
print("Numbers greater than 4:", greater_than_num)

print("Maximum value:", np.max(arr))
print("Minimum value:", np.min(arr))
print("Index of maximum value:", np.argmax(arr))
print("Index of minimum value:", np.argmin(arr))

print("Unique elements:", np.unique(arr))
print("Count of each element:", np.bincount(arr))
print("Representation of array:", np.array_repr(arr))
```


PROCEDURE:

1. Import NumPy.
2. Extract elements using condition.
3. Find max/min values and indices.
4. Use unique and bincount.
5. Display array representation.

INPUT:

Predefined array and threshold.

EXPECTED / ACTUAL OUTPUT:

Original array: [1 3 5 7 3 5 9 2 5]
Numbers less than 4: [1 3 3 2]
Numbers greater than 4: [5 7 5 9 5]
Maximum value: 9
Minimum value: 1
Index of maximum value: 6
Index of minimum value: 0
Unique elements: [1 2 3 5 7 9]
Count of each element: [0 1 1 2 0 3 0 1 0 1]
Representation of array: array([1, 3, 5, 7, 3, 5, 9, 2, 5])

Experiment 4

Program No.: 4.1

Aim:

To create and display a one-dimensional Pandas Series and convert it to a Python list.

Software used: Colab (or) Jupyter Notebook

Description:

This program demonstrates how to create a Pandas Series, display it, and convert it into a Python list.

SOURCE CODE:

```
import pandas as pd

# Create and display a Pandas Series
data_list = pd.Series([10, 20, 30, 40, 50])
print("Pandas Series created from a list:")
print(data_list)

# Convert the Series to a Python list and display type
data_list_converted = data_list.tolist()
print("\nConverted to Python list:", data_list_converted)
print("Type of converted object:", type(data_list_converted))
```


PROCEDURE:

1. Import Pandas.
2. Create a Pandas Series from a list.
3. Display the Series.
4. Convert to list and display type.

INPUT:

Predefined list: [10, 20, 30, 40, 50]

EXPECTED / ACTUAL OUTPUT:

Pandas Series created from a list:
0 10
1 20
2 30
3 40
4 50
dtype: int64

Converted to Python list: [10, 20, 30, 40, 50]
Type of converted object: <class 'list'>

Program No.: 4.2

Aim:

Create and manipulate a Pandas DataFrame from a dictionary, update values, add a new column, and display column headers.

Software used: Colab (or) Jupyter Notebook

Description: This program demonstrates creation and manipulation of a Pandas DataFrame including updating values and adding new columns.

SOURCE CODE:

```
import pandas as pd
import numpy as np

exam_data = {
   'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
   'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
   'attempts': [1, 3, 2, 3, 1, 1, 2, 1, 1, 2],
   'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'yes', 'yes']
}
labels = ['a','b','c','d','e','f','g','h','i','j']

df = pd.DataFrame(exam_data, index=labels)
print("DataFrame created from dictionary:")
print(df)

df.loc['d','name'] = 'Suresh'
print("\nDataFrame after updating name 'James' to 'Suresh':")
print(df)

salaries = [50000, 60000, 75000, 45000, 55000, 80000, 70000, 48000, 62000, 78000]
df['salary'] = salaries
print("\nDataFrame after adding 'salary' column:")
print(df)

column_headers = df.columns.tolist()
print("\nList of column headers:", column_headers)
```

PROCEDURE:

1. Import Pandas and NumPy.
2. Create dictionary and index labels.
3. Create DataFrame.
4. Update a value.
5. Add new column.
6. Display DataFrame and headers.

INPUT:

Predefined dictionary with student data.

EXPECTED / ACTUAL OUTPUT:

DataFrame created from dictionary:

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	James	NaN	3	no
e	Emily	9.0	1	no
f	Michael	20.0	1	yes
g	Matthew	14.5	2	yes
h	Laura	NaN	1	no
i	Kevin	8.0	1	yes
j	Jonas	19.0	2	yes

DataFrame after updating name 'James' to 'Suresh':

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	Suresh	NaN	3	no
e	Emily	9.0	1	no
f	Michael	20.0	1	yes
g	Matthew	14.5	2	yes
h	Laura	NaN	1	no
i	Kevin	8.0	1	yes
j	Jonas	19.0	2	yes

DataFrame after adding 'salary' column:

	name	score	attempts	qualify	salary
a	Anastasia	12.5	1	yes	50000

b Dima 9.0 3 no 60000
c Katherine 16.5 2 yes 75000
d Suresh NaN 3 no 45000
e Emily 9.0 1 no 55000
f Michael 20.0 1 yes 80000
g Matthew 14.5 2 yes 70000
h Laura NaN 1 no 48000
i Kevin 8.0 1 yes 62000
j Jonas 19.0 2 yes 78000

List of column headers: ['name', 'score', 'attempts', 'qualify', 'salary']