LEARNING OUTCOMES

1) Python basics: Data Types, Arrays, Lists, tuples, dictionaries, if else, for loop, numpy, pandas, matplotlib, opencv, openpyxl

a) String palindrome: A palindrome is a word or phrase that reads the same forward and backward.
Examples: madam, radar, level. Non-palindromic examples: hello, python.

Code:

```
def is_palindrome(s):
    return s == s[::-1]
string = input("Enter a string: ")
if is_palindrome(string):
  print("The given string is Palindrome")
else:
  print("The given string is not a Palindrome")
```

Output:

```
string = input("Enter a string: ")
if is_palindrome(string):
  print("The given string is Palindrome")
else:
  print("The given string is not a Palindrome")
```

```
Enter a string:  radar
The given string is Palindrome
```

INTERPRETATION:

The function is_palindrome(s) checks whether a given string is the same when reversed (s[::-1]). The user is prompted to enter a string. If the string remains unchanged when reversed, it is a palindrome; otherwise, it is not.

b) Number palindrome: A numerical palindrome remains the same when read backward. Examples: 121, 1331, 454. Non-palindromic examples: 123, 789, 4567.

Code:

```
def is_palindrome(num):
    return str(num) == str(num)[::-1]  # Ensure str is not redefined elsewhere

num = int(input("Enter a number: "))

if is_palindrome(num):

    print(f"{num} is a palindrome.")

else:

    print(f"{num} is not a palindrome.")
```

Output:

```
num=input("Enter a string:")
n=num[::-1]
if num==n:
  print("yes,The given string is palindrome")
else:
  print("No,The given string is not a palindrome")
```

```
Enter a string: 1256
No,The given string is not a palindrome
```

INTERPRETATION:

The function is_palindrome(num) converts the number into a string and checks if it remains unchanged when reversed (str(num)[::-1]). The user inputs a number, and the program verifies if it is a palindrome.

c)Check Given number is Arm strong Number or not.

Arm Strong Number: An Armstrong number (or narcissistic number) satisfies: sum of (each digit raised to the power of number of digits) = original number\text {sum of (each digit raised to the power of number of digits)} = \text {original number} sum of (each digit raised to the power of number of digits) = original number.

Example:

- 153 → 13+53+33=1+125+27=1531^3 + 5^3 + 3^3 = 1 + 125 + 27 = 15313+53+33=1+125+27=153
- 9474 → 94+44+74+44=94749^4 + 4^4 + 7^4 + 4^4 = 947494+44+74+44=9474

Non-Armstrong examples: 123, 100, 200.

Code:

```
def is_armstrong(n):
    digits = [int(d) for d in str(n)]
    power = len(digits)
    return sum(d**power for d in digits) == n
num = int(input("Enter a number: "))
if is_armstrong(num):
    print("The given number is Armstrong Number")
else:
    print("The given number is not an Armstrong Number")
```

Output:

```
if is_armstrong(num):
    print("The given number is Armstrong Number")
else:
    print("The given number is not an Armstrong Number")
```

```
Enter a number:  153
The given number is Armstrong Number
```

INTERPRETATION:

The function is_armstrong(n) computes the sum of each digit raised to the power of the total number of digits. If the sum equals the original number, it is an Armstrong number. The user inputs a number, and the program verifies if it meets the Armstrong condition.

2) OOPS - Class, object, Inheritance, Polymorphism, Encapsulation etc.…

a) Method overloading: It allows multiple methods with the same name but different numbers of parameters.

Code:

```
#Method overloading with a finite number of parameters
class OverloadingExample:
    def add(self, a, b, c=0):
        return a + b + c
obj = OverloadingExample()
print(obj.add(2, 3))
print(obj.add(2, 3, 4,))
```

Output:

```
print(obj.add(2, 3, 4,))

5
9
```

(or)

Code:

```
#Method overloading with a infinite number of parameters
class  Test:
    def  sum(self,*n):
        sum=0
        for i in n:
            sum+=i
        print("Sum is : ",sum)
t = Test( )
t.sum(5,10)
t.sum(10,20,30)
t.sum(5.2,3.4,5,7)
```

Output:

```
Sum is :  15
Sum is :  60
Sum is :  20.6
```

INTERPRETATION:

Python does not support traditional method overloading like Java or C++, but it allows handling multiple arguments using default parameters or *args. This makes Python flexible for defining functions that work with different numbers of inputs.

b) Method overriding: It occurs when a child class provides a different implementation for a method already defined in the parent class. It allows a subclass to **redefine** a method inherited from a parent class

Code:

```
class Shape:
  def draw(self):
    print(" Draw Shape ")
class Square(Shape):
  def draw(self):
    print(" Draw Square ")
class Circle(Shape):
  def draw(self):
    print(" Draw Circle ")
class Hexagon(Shape):
  def draw(self):
    print(" Draw Hexagon ")
s = Square( )
s.draw()
c = Circle( )
c.draw()
h = Hexagon( )
h.draw()
```

Output:

```
s = Square( )
s.draw()
c = Circle( )
c.draw()
h = Hexagon( )
h.draw()
```

```
Draw Square
Draw Circle
Draw Hexagon
```

(or)

Code:

```
class Parent:
    def show(self):
        print("This is Parent class")
class Child(Parent):
    def show(self):
        print("This is Child class")
obj = Child()
obj.show()
```

Output:

```
obj = Child()
obj.show()
```

```
This is Child class
```

INTERPRETATION:

Method overriding allows subclasses to **customize** inherited methods while maintaining a consistent interface. This is useful in cases like UI components (where different shapes can be drawn) or hierarchical structures (where child behaviour differs from the parent).

3) Data Frame & Data Visualization:

Create A python data frame from any excel/CSV data and visualize the data with graph representation.

- The code efficiently loads, explores, and visualizes real-world visa application data.
- Data Frames help in organizing and analysing structured data.
- Visualization using bar charts helps in identifying trends and making data-driven decisions.
- These techniques are crucial for data analytics and business intelligence in immigration and employment sectors.

Code:

#Installing necessary packages

pip install pandas matplotlib seaborn ipywidgets

Output:

```
Requirement already satisfied: pandas in c:\users\lahar\appdata\local\programs\python\pytho
n312\lib\site-packages (2.2.3)
Requirement already satisfied: matplotlib in c:\users\lahar\appdata\local\programs\python\p
ython312\lib\site-packages (3.9.2)
```

Code: The dataset is loaded using pd.read_csv(), which reads the CSV file into a pandas Data Frame.

#Uploading the file(visadataset.CSV)

import pandas as pd

df=pd.read_csv("C:\\Users\\lahar\\OneDrive\\Desktop\\Sem-3\\Ad.Python\\visadataset.csv")

print(df)

Output:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employment	prevailing_wage	unit_of_wage	full_time_position	case_status
0	EZYV01	Asia	High School	N	N	14513	2007	West	592.2029	Hour	Y	Denied
1	EZYV02	Asia	Master's	Y	N	2412	2002	Northeast	83425.6500	Year	Y	Certified
2	EZYV03	Asia	Bachelor's	N	Y	44444	2008	West	122996.8600	Year	Y	Denied
3	EZYV04	Asia	Bachelor's	N	N	98	1897	West	83434.0300	Year	Y	Denied
4	EZYV05	Africa	Master's	Y	N	1082	2005	South	149907.3900	Year	Y	Certified

[25480 rows x 12 columns]

Code: Provides basic details like column names, data types, and missing values.

#info() Info is used to display the basic information of the data.

print(df.info())

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25480 entries, 0 to 25479
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   case_id                25480 non-null  object
 1   continent              25480 non-null  object
 2   education_of_employee  25480 non-null  object
 3   has_job_experience     25480 non-null  object
 4   requires_job_training  25480 non-null  object
 5   no_of_employees        25480 non-null  int64
 6   yr_of_estab            25480 non-null  int64
 7   region_of_employment   25480 non-null  object
 8   prevailing_wage        25480 non-null  float64
 9   unit_of_wage           25480 non-null  object
 10  full_time_position     25480 non-null  object
 11  case_status            25480 non-null  object
dtypes: float64(1), int64(2), object(9)
memory usage: 2.3+ MB
None
```

Code: gives statistical insights like mean, count, min, and max values.

#describe()

print(df.describe)

Output:

```
print(df.describe)
```

```
<bound method NDFrame.describe of          case_id continent education_of_employee
0         EZYV01      Asia           High School                  N
1         EZYV02      Asia             Master's                   Y
```

1) Below graph shows the overall Visa application distribution status

- The first visualization is a bar chart showing the count of visa applications for each case_status.
- The value_counts() function is used to count occurrences of each status.
- seaborn.barplot() creates a bar chart with a color palette (viridis).

Code:

#Import package

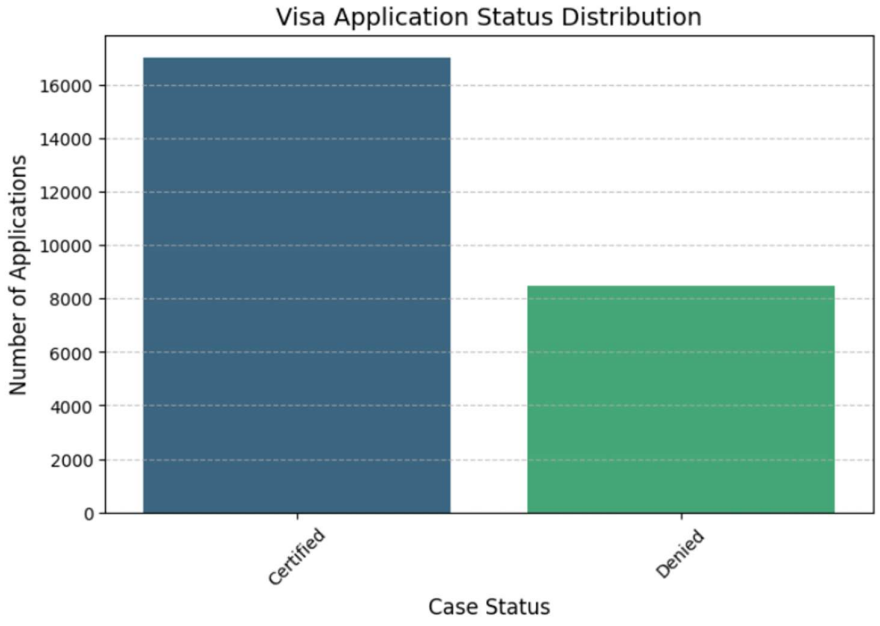
import matplotlib.pyplot as plt

import seaborn as sns

```
# Count the number of applications per case status
case_status_counts = df["case_status"].value_counts()
print(case_status_counts)
# Plot the distribution of visa application status
plt.figure(figsize=(8, 5))
sns.barplot(x=case_status_counts.index,
            y=case_status_counts.values,
            hue=case_status_counts.index,
            legend=False,
            palette="viridis")
# Add labels and title
plt.title("Visa Application Status Distribution", fontsize=14)
plt.xlabel("Case Status", fontsize=12)
plt.ylabel("Number of Applications", fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

Output:

```
case_status
Certified    17018
Denied        8462
Name: count, dtype: int64
```



INTERPRETATION:

This graph helps in identifying trends, such as how many applications were approved or denied. It provides an overall picture of the visa application outcomes.

2) Below graph shows the visa application distribution by continents

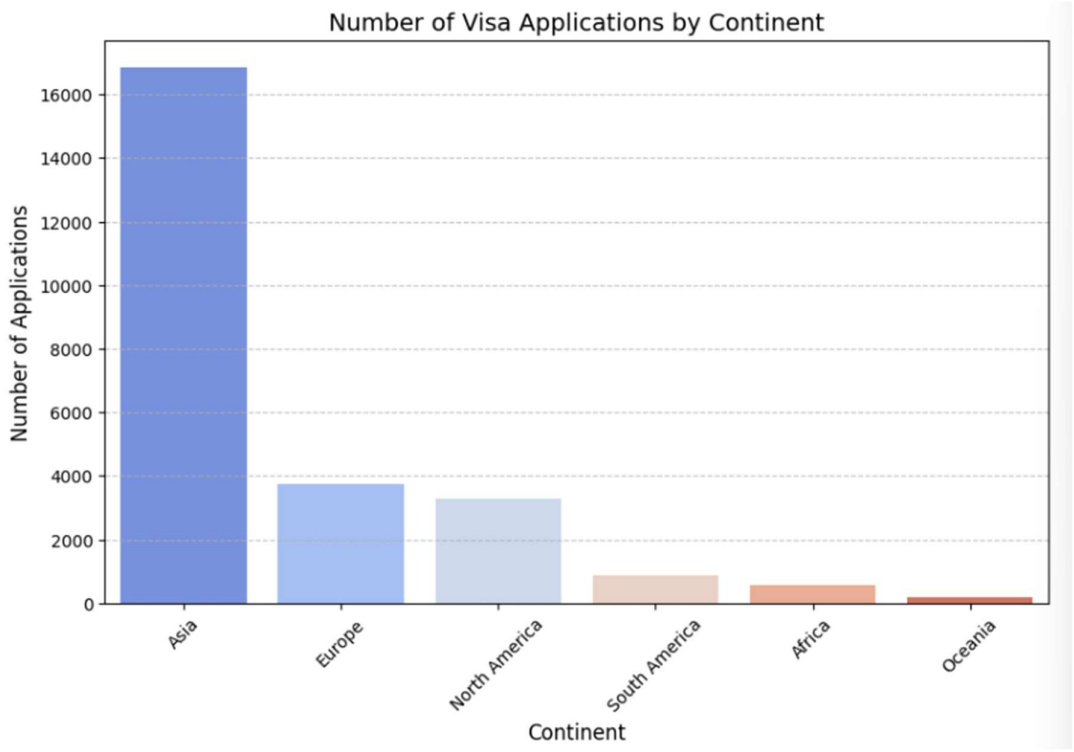
- This visualization shows the number of applications grouped by continent.
- df["continent"].value_counts() counts the applications from different regions.
- A bar chart is plotted using seaborn.barplot().

Code:

```
# Count the number of applications per continent
continent_counts = df["continent"].value_counts()
print(continent_counts)
# Plot the distribution of visa applications by continent
plt.figure(figsize=(10, 6))
sns.barplot(x=continent_counts.index,
            y=continent_counts.values,
            hue=continent_counts.index,
            legend=False,
            palette="coolwarm")
# Add labels and title
plt.title("Number of Visa Applications by Continent", fontsize=14)
plt.xlabel("Continent", fontsize=12)
plt.ylabel("Number of Applications", fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

Output:

```
continent
Asia             16861
Europe            3732
North America     3292
South America      852
Africa             551
Oceania            192
Name: count, dtype: int64
```



INTERPRETATION:

This graph highlights which continents have the highest or lowest number of visa applicants. It can help in identifying regional trends in visa applications.

3) Below graph shows the Visa Application Status by Education Level

- This visualization is a stacked bar chart comparing visa statuses based on applicants' education levels.
- groupby(["education_of_employee", "case_status"]).size().unstack() is used to count applications per education level and visa status.
- The plot(kind="bar", stacked=True) function generates a stacked bar chart.

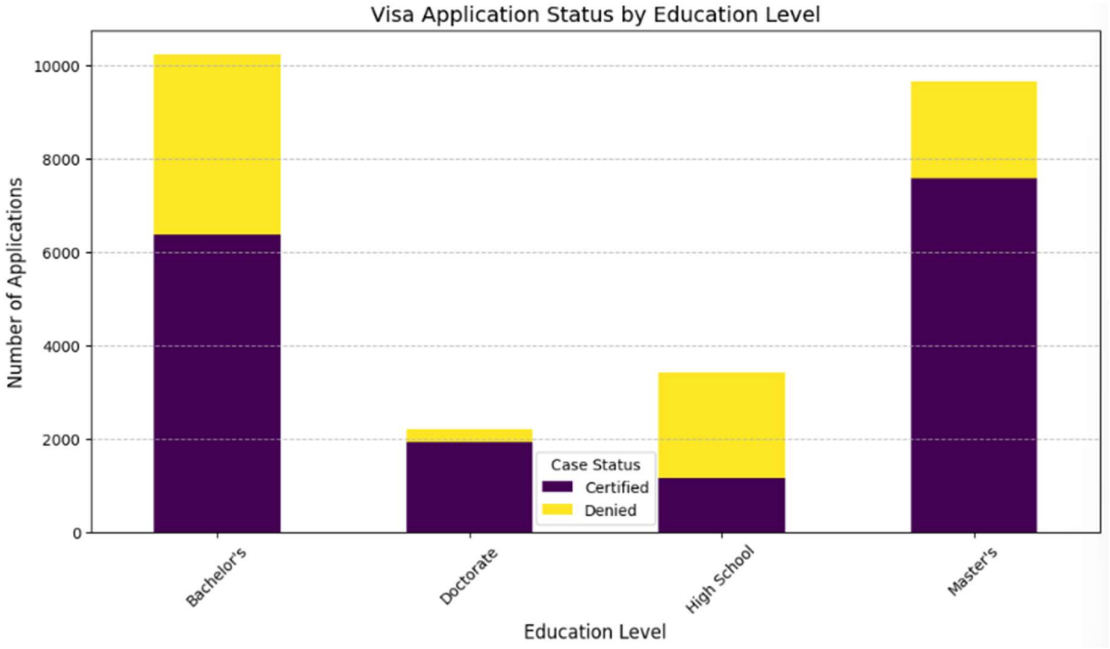
Code:

```
# Count visa status by education level

education_status_counts = df.groupby(["education_of_employee",
"case_status"]).size().unstack()

print(education_status_counts)

# Plot a stacked bar chart

education_status_counts.plot(kind="bar", stacked=True, figsize=(12, 6), colormap="viridis")

# Add labels and title

plt.title("Visa Application Status by Education Level", fontsize=14)
```

```
plt.xlabel("Education Level", fontsize=12)
plt.ylabel("Number of Applications", fontsize=12)
plt.xticks(rotation=45)
plt.legend(title="Case Status")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

Output:

case_status	Certified	Denied
education_of_employee		
Bachelor's	6367	3867
Doctorate	1912	280
High School	1164	2256
Master's	7575	2059



INTERPRETATION:

It provides insights into how education levels affect visa application outcomes. This can be useful for analysing trends in visa approval rates for different educational qualifications.

4) Windows application- Create Windows Application using Library tkinter

Create a windows application with textbox Button and label. Once Enter number in textbox and then click on button then is that number Armstrong Number/Not will display on label.

This assignment demonstrates how to create a **Windows GUI application** using the **Tkinter** library in Python. The application includes:

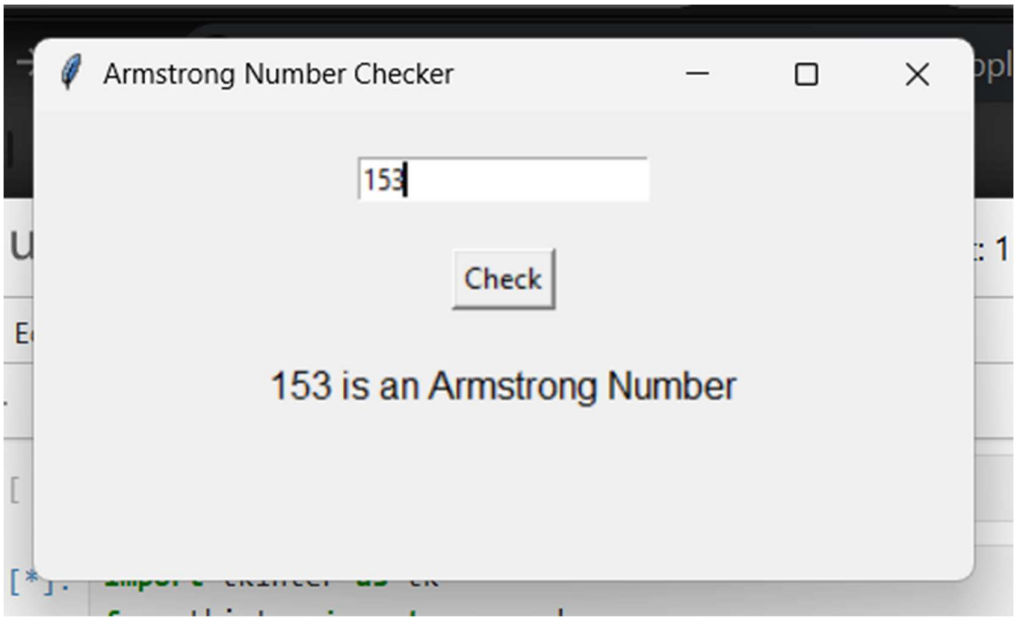
- A **textbox** for user input.
- A **button** to trigger the Armstrong number check.
- A **label** to display the result.

Code:

```
import tkinter as tk
from tkinter import messagebox
def is_armstrong(number):
    try:
        num = int(number)
        sum_of_digits = sum(int(digit) ** len(number) for digit in number)
        return num == sum_of_digits
    except ValueError:
        return False
def check_armstrong():
    number = entry.get()
    if number.isdigit():
        if is_armstrong(number):
            result_label.config(text=f"{number} is an Armstrong Number")
        else:
            result_label.config(text=f"{number} is NOT an Armstrong Number")
    else:
        messagebox.showerror("Invalid Input", "Please enter a valid number")
# Create main application window
root = tk.Tk()
root.title("Armstrong Number Checker")
```

```
root.geometry("400x200")
# Create input field
entry = tk.Entry(root, width=20)
entry.pack(pady=20)
# Create button
check_button = tk.Button(root, text="Check", command=check_armstrong)
check_button.pack()
# Create label for result
result_label = tk.Label(root, text="", font=("Arial", 12))
result_label.pack(pady=20)
# Run the application
root.mainloop()
```

Output:



INTERPRETATION:

- **User Input Handling**: The user enters a number in the textbox (entry). The is_armstrong() function retrieves this input.
- **Armstrong Number Calculation:** Converts the input to an integer. Splits it into individual digits and calculates the sum of digits raised to the power of their count. Compares the result with the original number.
- **Displaying the Result:** If the number is Armstrong, a green success message is displayed. Otherwise, a red failure message appears. If the input is invalid (non-numeric), an error message is shown.

5) Sqlserver- Using Sqlserver database Learn queries like Select, Insert, Update, Delete

Get data from DB and insert data into DB

- **SQL operations demonstrated:**

✅ **Creating** a database & table.
✅ **Inserting** records into the table.
✅ **Selecting** all records.
✅ **Updating** employee information.
✅ **Deleting** specific records

SQL Code:

```
CREATE DATABASE SampleDB;
GO
USE SampleDB;
GO
CREATE TABLE Employees (
    EmployeeID INT IDENTITY(1,1) PRIMARY KEY,
    Name VARCHAR(100),
    Age INT,
    Department VARCHAR(50)
);
GO
INSERT INTO Employees (Name, Age, Department)
VALUES
    ('Alice Johnson', 30, 'HR'),
    ('Bob Smith', 28, 'IT'),
    ('Charlie Brown', 35, 'Finance');
GO
SELECT * FROM Employees;
GO
UPDATE Employees
SET Age = 32
```

```
WHERE Name = 'Alice Johnson';
GO
SELECT * FROM Employees;
GO
DELETE FROM Employees
WHERE Name = 'Charlie Brown';
GO
SELECT * FROM Employees;
GO
Delete From Employees where EmployeeID=1033
```

Output:

Results | Messages

	EmployeeID	Name	Age	Department
1	1023	Alice Johnson	32	HR
2	1024	Bob Smith	28	IT

INTERPRETATION:

The final dataset contains only **two employees** (Alice Johnson and Bob Smith) if all deletions were successful.

Python Code:

```
pip install pyodbc
```

Output:

```
Requirement already satisfied: pyodbc in c:\users\lahar\appdata\local\programs\python\pytho
n312\lib\site-packages (5.2.0)
Note: you may need to restart the kernel to use updated packages.
```

Code:

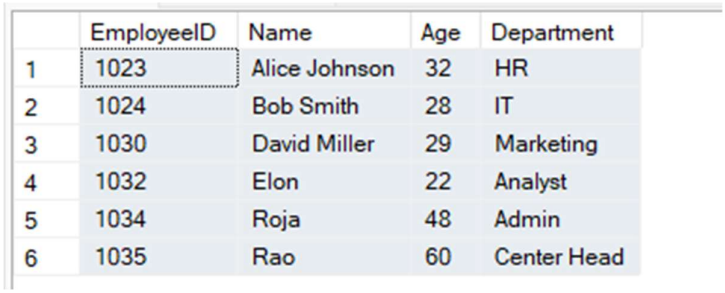
```
import pyodbc
# Database connection
conn = pyodbc.connect('DRIVER={SQL
Server};SERVER=Lahari;DATABASE=SampleDB;Trusted_Connection=yes;')
cursor = conn.cursor()
# Select Data
```

```
cursor.execute("SELECT * FROM Employees")

for row in cursor.fetchall():

    print(row)

# Insert Data

cursor.execute("INSERT INTO Employees (Name, Age, Department) VALUES ('Rao', 60,
'Center Head')")

conn.commit()

# Close the connection

cursor.close()

conn.close()
```

Output:

```
(1023, 'Alice Johnson', 32, 'HR')
(1024, 'Bob Smith', 28, 'IT')
(1030, 'David Miller', 29, 'Marketing')
(1032, 'Elon', 22, 'Analyst')
(1034, 'Roja', 48, 'Admin')
```

SQL Output:



	EmployeeID	Name	Age	Department
1	1023	Alice Johnson	32	HR
2	1024	Bob Smith	28	IT
3	1030	David Miller	29	Marketing
4	1032	Elon	22	Analyst
5	1034	Roja	48	Admin
6	1035	Rao	60	Center Head

INTERPRETATION:

- **Purpose:** The script connects Python to an SQL Server database using pyodbc, retrieves existing employee records, and inserts a new record.
- **Operations Performed:**
  - ✅ Database **Connection**
  - ✅ Data **Retrieval (SELECT * FROM Employees)**
  - ✅ Data **Insertion (INSERT INTO Employees)**
  - ✅ Proper **Resource Cleanup** (close () functions)

Final Output: After execution, a new record for Rao (Center Head) is added to the Employees table.