# Experiment -10

Python – Database connectivity

**Aim:** Python program to demonstrate mysql database connectivity

**Theory:**

**Install a Pip Python Module**

To connect MySQL to Python, you'd need **to install the Pip Python module**.
A Pip module is the *standard package manager for Python*. It allows you to install and manage additional packages and extensions that are part of the standard Python library distribution.
**Install at least one** of these packages in a virtual environment to connect MySQL using Python.

- **Mysql-client:** this client package allows you to connect to a MySQL server and access the command-line program. It comes with utilities that enable you to easily backup data, restore and administer the server. This package contains the MySQLdb module.
- **Mysql-connector-python:** this package is a MySQL driver that enables Python programs to access **MySQL databases using an API**. The package contains the *mysql.connector* module.
- **PyMySQL:** this package provides an interface for connecting to the MySQL database server. It contains the pymysql module.

**Installing The MySQL Packages**

After creating and activating the python virtual environment, **the next thing is to install the MySQL packages**. It's advisable to install all three packages.
Installing multiple modules allows you **to switch between modules anytime**. The modules use the portable SQL database API interface; this will enable you to reuse codes without any modification.
Run these codes to install the packages.

- Run this code to install the mysqlclient package *pip install mysqlclient*
- Use this code to install the mysql-connector-python package *pip install mysql-connector-python*
- And, this to install the pymysql packages *pip install pymysql*

**Creating The Connection**

After installing the packages, **you can connect to your MySQL databases** and run commands through any of those modules.

Run this code to establish a MySQL Python connection using the MySQL connector module.

```
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
print(mydb)
```

Replace *yourusername* and *yourpassword* with those of the database you want to connect to.

To set up series of Python connection that opens the same database using the different MySQL packages, then **run the below sample code**.

```
#!/usr/bin/python
from __future__ import print_function
hostname = 'localhost'
username = 'yourusername'
password = 'yourpassword'
database = 'yourdbname'
```

Replace *yourusername* with the username of the MySQL database you want to connect to, *yourpassword* with the database user's password, and *yourdbname* with the database's name.

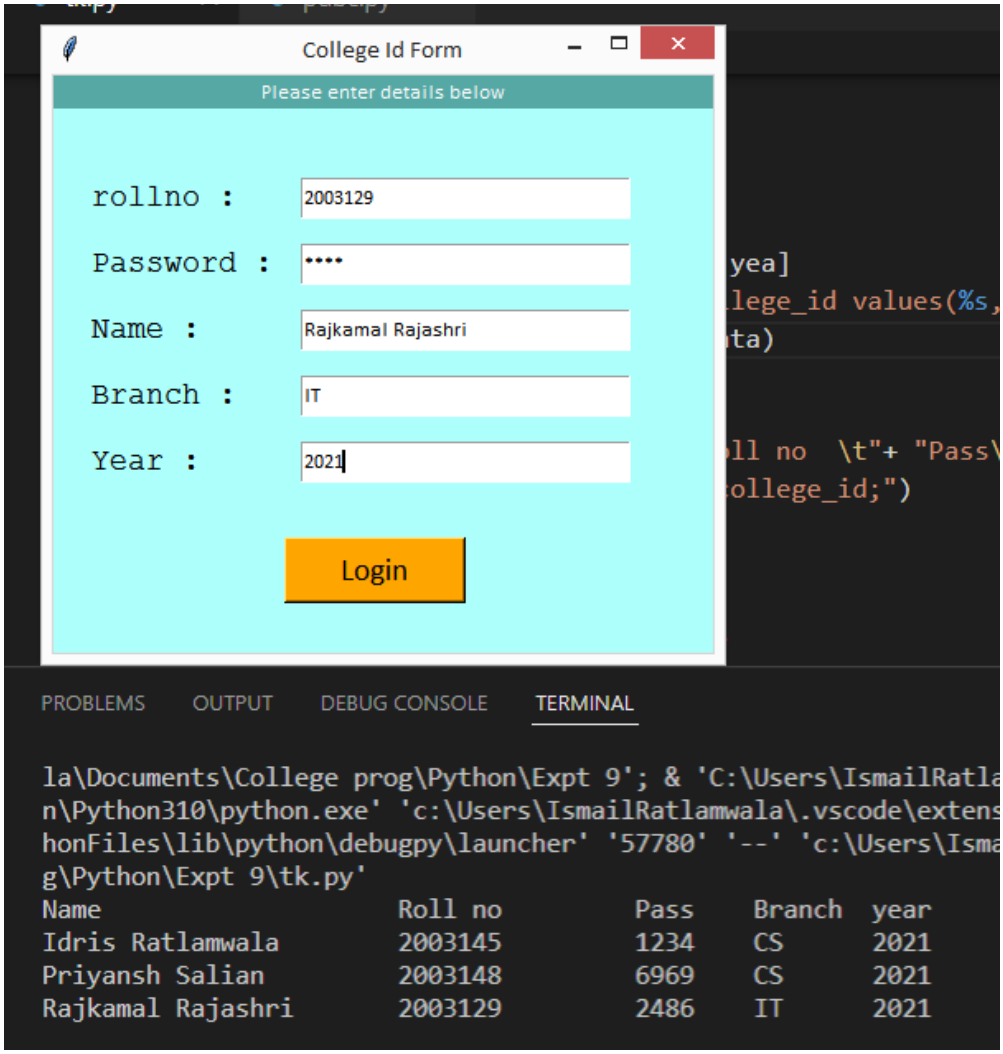
## Program :

```
from tkinter import *
from tkinter import messagebox
def login():
   uname=rollno.get()
   pwd=password.get()
   nam=name.get()
   yea=year.get()
   bran=branch.get()

   print("Roll no\t"+"Pass\t" +"Name\t"+"Year\tBranch")
   print(uname +"\t"+ pwd +"\t"+ nam +"\t"+ yea +"\t"+ bran)
   if uname=='' or pwd==''or nam==''or yea==''or bran==':
     messagebox.showerror('Error', 'Plese enter all details')
```

```
    else:
      if uname=="2003145" and pwd=="1234":
       messagebox.showinfo('Login success', 'You have logged in\nsuccessfully !')
      else:
       messagebox.showinfo('Login failed', 'Wrong roll no or password')

def Loginform():
    global login_screen
    login_screen = Tk()
    login_screen.title("College Id Form")
    bgCol='#adfffc'
    login_screen.geometry("400x350")
    login_screen.configure(bg=bgCol)
    global  message
    global rollno
    global password
    global name
    global branch
    global year
    rollno = StringVar()
    password = StringVar()
    message=StringVar()
    name=StringVar()
    branch=StringVar()
    year=StringVar()
    Label(login_screen,width="300", text="Please enter details below",
bg="#56a8a5",fg="white").pack()
    yoff=20
    llogin = Label(login_screen, text="rollno :")
    llogin.config(font=("Courier", 14),bg=bgCol)
    llogin.place(x=20,y=40+yoff)
    Entry(login_screen, textvariable=rollno).place(
        x=150,y=42+yoff,width=200,height=25)

    lpass = Label(login_screen, text="Password :")
    lpass.config(font=("Courier", 14),bg=bgCol)
    lpass.place(x=20,y=80+yoff)
    Entry(login_screen, textvariable=password ,show="*").place(
        x=150,y=82+yoff,width=200,height=25)

    lname = Label(login_screen, text="Name :")
    lname.config(font=("Courier", 14),bg=bgCol)
    lname.place(x=20,y=120+yoff)
    Entry(login_screen, textvariable=name ).place(
        x=150,y=122+yoff,width=200,height=25)

    lbra = Label(login_screen, text="Branch :")
    lbra.config(font=("Courier", 14),bg=bgCol)
    lbra.place(x=20,y=160+yoff)
    Entry(login_screen, textvariable=branch ).place(
        x=150,y=162+yoff,width=200,height=25)

    lyear = Label(login_screen, text="Year :")
```

```
    lyear.config(font=("Courier", 14),bg=bgCol)
    lyear.place(x=20,y=200+yoff)
    Entry(login_screen, textvariable=year ).place(
        x=150,y=202+yoff,width=200,height=25)

    but = Button(login_screen, text="Login", width=10, height=1, bg="orange",command=login)
    but.config(font=("Calibri", 14))
    but.place(x=140,y=260+yoff)
    login_screen.mainloop()

Loginform()
```

## Output :




```
1  use stud_db;
2
3  SET SQL_SAFE_UPDATES = 0;
4  select * from college_id;
```

| stud_name | username | passw | branch | year_ |
|---|---|---|---|---|
| Idris Ratlamwala | 2003145 | 1234 | CS | 2021 |
| Priyansh Salian | 2003148 | 6969 | CS | 2021 |
| Rajkamal Rajashri | 2003129 | 2486 | IT | 2021 |

# Experiment-11

**Aim:** Django Web Framework

## Program :

Creating web application using Django web framework

- Installing Django
- Creating project
- Creating App and Views
- Creating and activating model
- Admin interface -Modify database from admin interface **Functions used:**

## Theory:

### 1. Django:

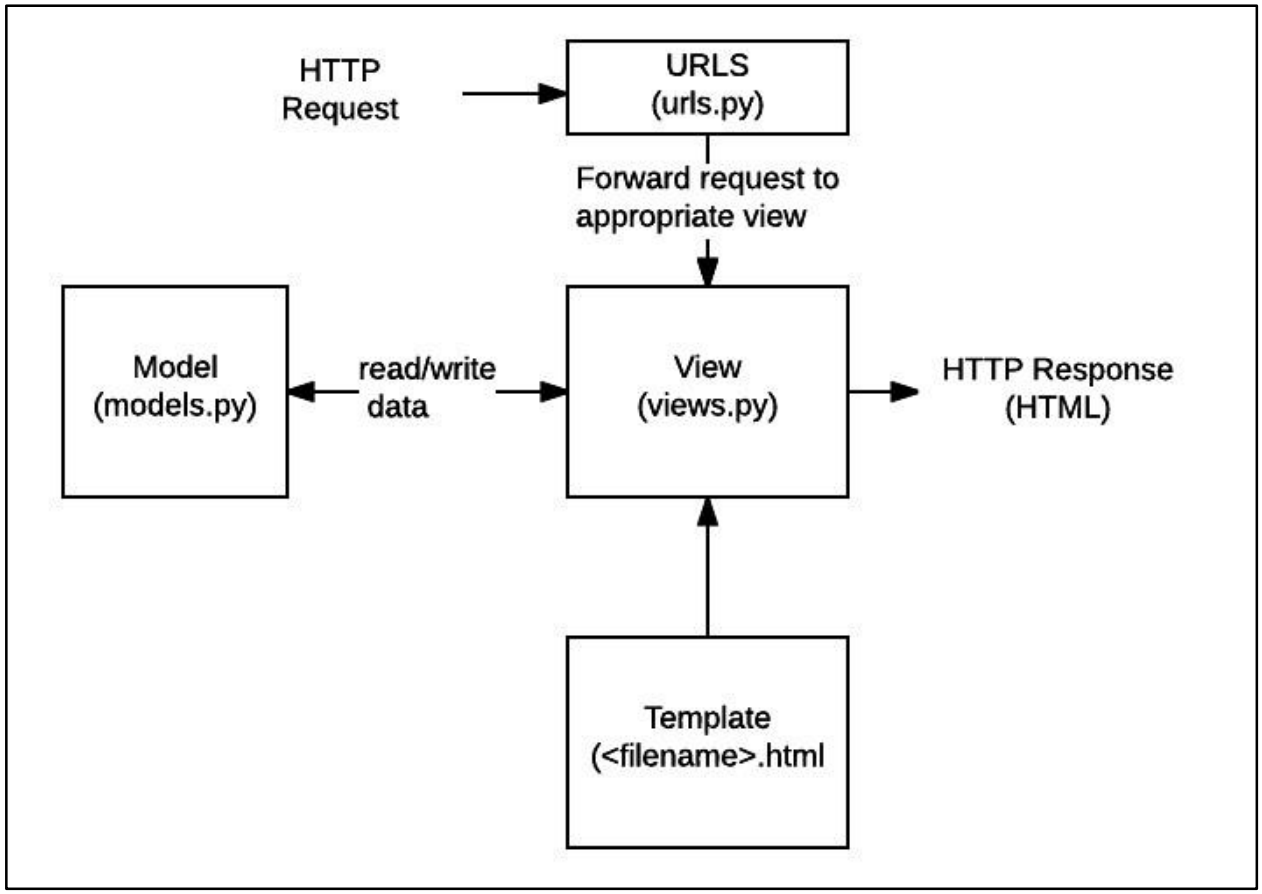
Django is a high-level Python Web framework that encourages rapid development and clean pragmatic design. A Web framework is a set of components that provide a standard way to develop websites fast and easily. Django's primary goal is to ease the creation of complex database-driven websites. Some well known sites that use Django include PBS, Instagram, Disqus, Washington Times, Bitbucket and Mozilla.

In a traditional data-driven website, a web application waits for HTTP requests from the web browser (or other client). When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template.

Django web applications typically group the code that handles each of these steps into separate files:

- URLs: While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in a URL and pass these to a view function as data.
- View: A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via models, and delegate the formatting of the response to templates.
- Models: Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
- Templates: A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A view can dynamically create an HTML page using an HTML template, populating it with data

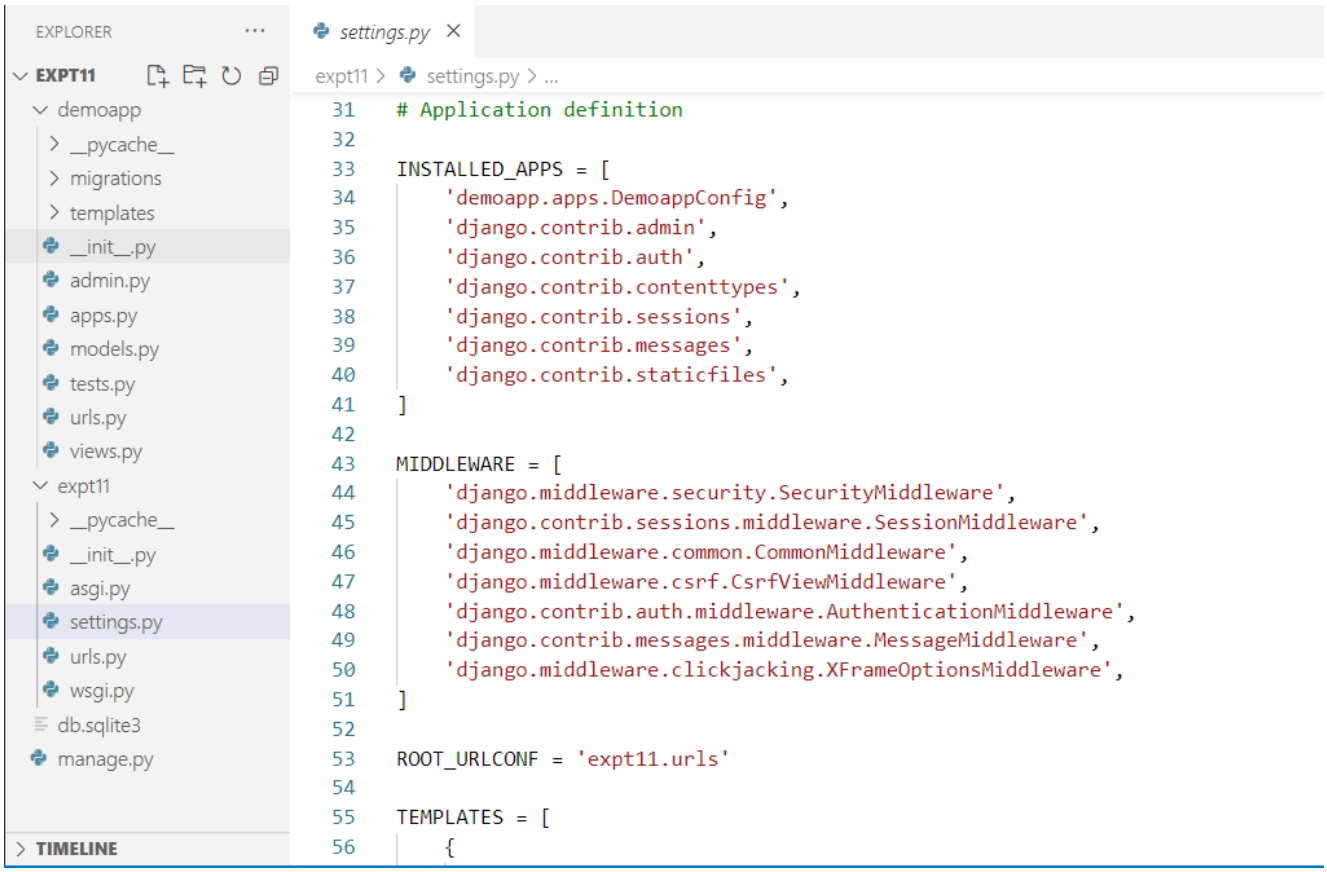
from a model. A template can be used to define the structure of any type of file; it doesn't have to be HTML!



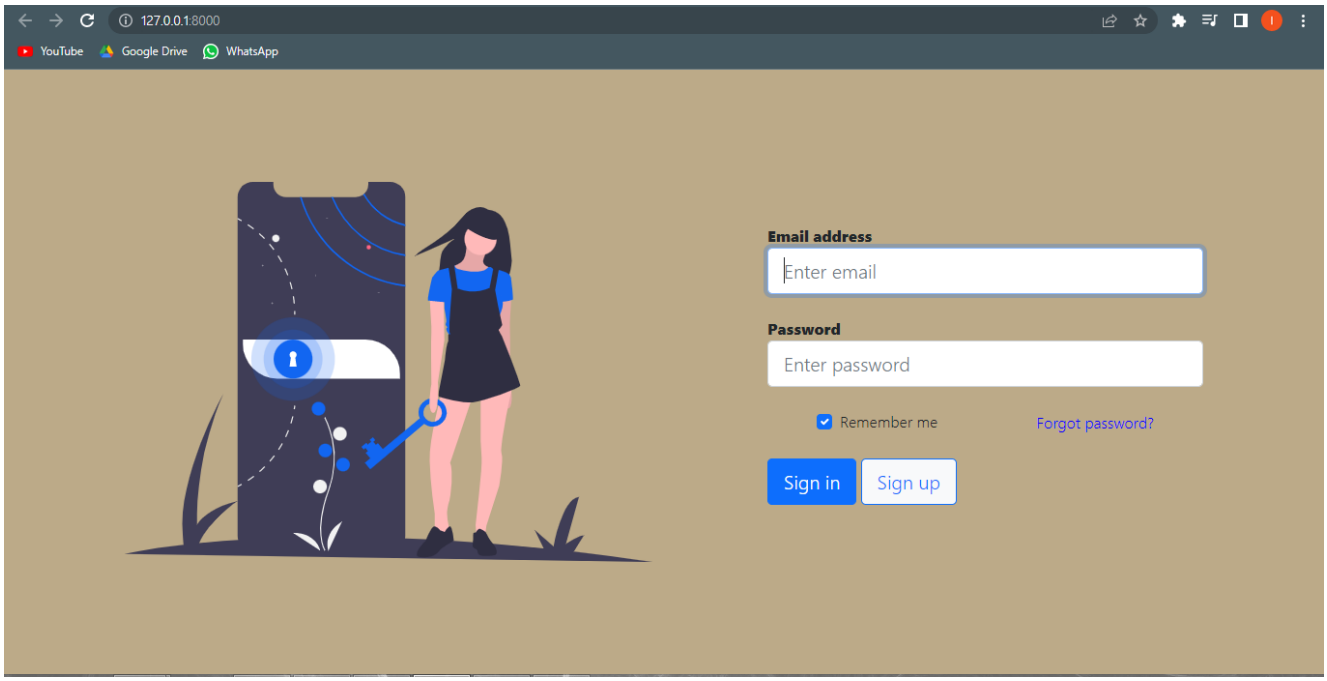

**Program :**

mange.py :

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys

def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'expt11.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()
```


EXPLORER
EXPT11
demoapp
__pycache__
migrations
templates
__init__.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py
expt11
__pycache__
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
db.sqlite3
manage.py
TIMELINE
settings.py
expt11 > settings.py > ...
31 # Application definition
32
33 INSTALLED_APPS = [
34 'demoapp.apps.DemoappConfig',
35 'django.contrib.admin',
36 'django.contrib.auth',
37 'django.contrib.contenttypes',
38 'django.contrib.sessions',
39 'django.contrib.messages',
40 'django.contrib.staticfiles',
41 ]
42
43 MIDDLEWARE = [
44 'django.middleware.security.SecurityMiddleware',
45 'django.contrib.sessions.middleware.SessionMiddleware',
46 'django.middleware.common.CommonMiddleware',
47 'django.middleware.csrf.CsrfViewMiddleware',
48 'django.contrib.auth.middleware.AuthenticationMiddleware',
49 'django.contrib.messages.middleware.MessageMiddleware',
50 'django.middleware.clickjacking.XFrameOptionsMiddleware',
51 ]
52
53 ROOT_URLCONF = 'expt11.urls'
54
55 TEMPLATES = [
56 {

## Output :


127.0.0.1:8000
YouTube
Google Drive
WhatsApp
Email address
Enter email
Password
Enter password
Remember me
Forgot password?
Sign in
Sign up

# Experiment No: 12

**Aim :** Pandas in Python

**Question 1-**

Write a pandas program to:

i) add, subtract, multiple and divide two pandas series ii) compare the elements ofthe two Pandas Series. iii) convert a dictionary to a Pandas series. iv) convert a NumPy array to a Pandas series.

**Function Used**:

1. **Pandas:** Pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal. pandas is well suited for many different kinds of data:
   > Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
   > Ordered and unordered (not necessarily fixed-frequency) time series data.
   >Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
   >Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

2. **Pandas Series:** Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.
   Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

3. **Operations on Series:** Basic arithmetic operations like addition, subtraction, multiplication, and division on two Pandas Series can be performed.
   Perform the required arithmetic operation using the respective arithmetic operator between the two Series
   ![image.png](attachment:image.png)
   Result can be assigned the to another Series.
   Similarly Relation Operators can be used to compare two Series. The result is obtained as a new series with boolean values by element to element comparison
   ![image-2.png](attachment:image-2.png)
   ![image-3.png](attachment:image-3.png)

4. **Numpy:** NumPy is a Python library used for working with arrays.It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was

created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

1)
Code:

```
import pandas as pd
s1=pd.Series({'one':6,'two':2,'three':3},index=['one','two','four'])
s2=pd.Series({'six':4,'two':9,'one':7})  print(f'Series s1 :\n',s1)
print(f'\nSeries s2 :\n',s2)

#Arithmetic operations on series

s3=s1+s2  s4=s1-s2  s5=s1*s2
s6=s1/s2print(f'\n\nAddition of series :\n{s3}')  print(f'\nSubtraction of
series :\n{s4}')  print(f'\nMultiplication of series :\n{s5}')
print(f'\nDivision of series :\n{s6}')
```

Output:

```
Series s1 :
 one     6.0
two     2.0
four    NaN
dtype: float64

Series s2 :
 six    4
two    9
one    7
dtype: int64


Addition of series :
four     NaN
one     13.0
six      NaN
two     11.0
dtype: float64

Subtraction of series :
four    NaN
one    -1.0
six     NaN
two    -7.0
dtype: float64
```

```
Multiplication of series :
four     NaN
one     42.0
six      NaN
two     18.0
dtype: float64

Division of series :
four         NaN
one     0.857143
six          NaN
two     0.222222
dtype: float64
```

**2)**
**Code:**

```
s1=pd.Series({'name':'rio','roll':80,'sem':'2'})
s2=pd.Series({'name':'tokyo','roll':45,'sem':'4'})  print(f'Series
s1 :\n',s1)  print(f'\nSeries s2 :\n',s2)  #Comparision
operators  s3=s1>s2
s4=s1>=s2  s5=s1<s2  s6=s1<=s2
s7=s1==s2  s8=s1!=s2
```

```
s9=s1.equals(s2)
s10=s1.compare(s2)
print(f'\n\nGreater than :\n{s3}') print(f'\nGreater than
or equal to :\n{s4}') print(f'\nLess than :\n{s5}')
print(f'\nLess than or equal to:\n{s6}') print(f'\nEqual
to:\n{s7}') print(f'\nNot equal to:\n{s8}')
print(f'\nequal() :{s9}') print(f'\ncompare():\n{s10}')
```

**Output:**

```
Series s1 :
 name    rio
roll     80
sem       2
dtype: object

Series s2 :
 name    tokyo
roll       45
sem         4
dtype: object


Greater than :
name    False
roll     True
sem     False
dtype: bool

Greater than or equal to :
name    False
roll     True
sem     False
dtype: bool

 compare():
      self  other
 name  rio  tokyo
 roll   80     45
 sem     2      4
```

**3)**

**Code:**

```
#Converting dictionary to pandas series
details=dict(name='Monica', roll=55, stream='Computer_Science') s1=pd.Series(details)
print(f'Dictionary t0 Pandas Series :\n',s1)
```

**Output:**

```
Dictionary t0 Pandas Series :
 name                 Monica
roll                     50
stream    Computer_Science
dtype: object
```

**4)**

**Code:**

```
import numpy as np
#Converting numpy arrays to pandas series nd_arr=np.array(['Lily',"CS",'TSEC'])
s1=pd.Series(nd_arr)
print(f'Numpy arrays to Pandas Series :\n',s1)
```

**Output:**

```
Numpy arrays to Pandas Series :
 0    Lily
1      CS
2    TSEC
dtype: object
```

**Question2**-

Write a program to read csv file in a dataframe, replace missing values with anyvalue, drop the row if all values are missing or contain null values.

**Function Used:**

1. **Pandas DataFrame:** Pandas DataFrame is two-dimensional sizemutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

2. **CSV file:** A comma-separated values (CSV) file is a plaintext file with a .csv extension that holds tabular data. This is one of the most popular file formats for storing large amounts of data. Each row of the CSV file represents a single table row. The values in the same row are by default separated with commas, but you could change the separator to a semicolon, tab, space, or some other character.

3. **read_csv():** Pandas read_csv() function imports a CSV file to DataFrame format.
   header: this allows you to specify which row will be used as column names for your dataframe. Expected an int value or a list of int values.
   Default value is header=0 , which means the first row of the CSV file will be treated as column names.

4. **head():** The head() function is used to get the first n rows.
   This function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

5. **isnull():** isnull() function detect missing values in the given series object. It return a boolean same-sized object indicating if the values are NA. Missing values gets mapped to True and non-missing value gets mapped to False .

6. **fillna():** DataFrame.fillna() method fills(replaces) NA or NaN values in the DataFrame with the specified values. fillna() method can be used to fill NaN values in the whole DataFrame, or specific columns, or modify inplace, or limit on the number of fillings, or choose an axis along which filling has to take place etc.

7. **dropna():** Pandas dropna() method allows the user to analyze and drop Rows/Columns with Null values in different ways. Parameters: axis: axis takes int or string value for rows/columns. Input can be 0 or 1 for Integer and 'index' or 'columns' for String.

**Code:**

```
import pandas as pd  import numpy as
np  #read csv file
df=pd.read_csv('flights_data.csv')  df=df.head(50)
print("\033[1m Dataframe of flights_data.csv file:::: \033[0m")  print(df)
#finding all the values with NAN
print('\n\n\033[1m  Boolean Dataframe of movies.csv file having values
NAN :::: \033[0m')
print(df.isnull())

#filling all NAN values with 999999
print('\033[1m Dataframe with all values filled with Indigo ::::\033[0m')  print(df.fillna('Indigo'))
#Droping all the rows with NAN values
print('\n\n  \033[1m Dataframe neglecting all rows with NAN :::: \033[0m') print(df.dropna())
```

**Output:**

```
 Dataframe of flights_data.csv file::::
              Airline Date_of_Journey  ...               Additional_Info  Price
0              IndiGo      24/03/2019  ...                       No info   3897
1           Air India       1/05/2019  ...                       No info   7662
2         Jet Airways       9/06/2019  ...                       No info  13882
3              IndiGo      12/05/2019  ...                       No info   6218
4              IndiGo      01/03/2019  ...                       No info  13302
5            SpiceJet      24/06/2019  ...                       No info   3873
6         Jet Airways      12/03/2019  ...   In-flight meal not included  11087
7         Jet Airways      01/03/2019  ...                       No info  22270
8         Jet Airways      12/03/2019  ...   In-flight meal not included  11087
9   Multiple carriers      27/05/2019  ...                       No info   8625
10          Air India       1/06/2019  ...                       No info   8907
11             IndiGo      18/04/2019  ...                       No info   4174
12          Air India      24/06/2019  ...                       No info   4667
13        Jet Airways       9/05/2019  ...   In-flight meal not included   9663
14             IndiGo      24/04/2019  ...                       No info   4804
15          Air India       3/03/2019  ...                       No info  14011
16           SpiceJet      15/04/2019  ...                       No info   5830
17        Jet Airways      12/06/2019  ...   In-flight meal not included  10262
18          Air India      12/06/2019  ...                       No info  13381
19        Jet Airways      27/05/2019  ...   In-flight meal not included  12898
20              GoAir       6/03/2019  ...                       No info  19495
21          Air India      21/03/2019  ...                       No info   6955
22             IndiGo       3/04/2019  ...                       No info   3943
```

Dataframe with all values filled with Indigo ::::

| | Airline | Date_of_Journey | ... | Additional_Info | Price |
|---|---|---|---|---|---|
| 0 | IndiGo | 24/03/2019 | ... | No info | 3897 |
| 1 | Air India | 1/05/2019 | ... | No info | 7662 |
| 2 | Jet Airways | 9/06/2019 | ... | No info | 13882 |
| 3 | IndiGo | 12/05/2019 | ... | No info | 6218 |
| 4 | IndiGo | 01/03/2019 | ... | No info | 13302 |
| 5 | SpiceJet | 24/06/2019 | ... | No info | 3873 |
| 6 | Jet Airways | 12/03/2019 | ... | In-flight meal not included | 11087 |
| 7 | Jet Airways | 01/03/2019 | ... | No info | 22270 |
| 8 | Jet Airways | 12/03/2019 | ... | In-flight meal not included | 11087 |
| 9 | Multiple carriers | 27/05/2019 | ... | No info | 8625 |
| 10 | Air India | 1/06/2019 | ... | No info | 8907 |
| 11 | IndiGo | 18/04/2019 | ... | No info | 4174 |
| 12 | Air India | 24/06/2019 | ... | No info | 4667 |
| 13 | Jet Airways | 9/05/2019 | ... | In-flight meal not included | 9663 |
| 14 | IndiGo | 24/04/2019 | ... | No info | 4804 |
| 15 | Air India | 3/03/2019 | ... | No info | 14011 |
| 16 | SpiceJet | 15/04/2019 | ... | No info | 5830 |
| 17 | Jet Airways | 12/06/2019 | ... | In-flight meal not included | 10262 |
| 18 | Air India | 12/06/2019 | ... | No info | 13381 |
| 19 | Jet Airways | 27/05/2019 | ... | In-flight meal not included | 12898 |
| 20 | GoAir | 6/03/2019 | ... | No info | 19495 |
| 21 | Air India | 21/03/2019 | ... | No info | 6955 |
| 22 | IndiGo | 3/04/2019 | ... | No info | 3943 |

## Question 3-

Write a program to demonstrate merging of Frames: i) on the basis of id
ii) using how

**Function Used:**

1. **Merge DataFrames:** Pandas DataFrame merge() function is used to merge two DataFrame objects with a database-style join operation. **merge() arguments**- Pandas provides a single function, merge, as the entry point for all standard database join operations between DataFrame objects − pd.merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=True) Here, we have used the following parameters −

   **left** − A DataFrame object.

   **right** − Another DataFrame object.

   **on** − Columns (names) to join on. Must be found in both the left and right DataFrame

   **objects.left_on** − Columns from the left DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame.

   **right_on** − Columns from the right DataFrame to use as keys. Can either be column names or arrays with length equal to the length of the DataFrame.

   **left_index** − If True, use the index (row labels) from the left DataFrame as its join key(s). In case of a DataFrame with a MultiIndex (hierarchical), the number of levels must match the number of join keys from the right DataFrame.

   **right_index** − Same usage as left_index for the right DataFrame.

**how** − One of 'left', 'right', 'outer', 'inner'. Defaults to inner. Each method has been described below.

**sort** − Sort the result DataFrame by the join keys in lexicographical order. Defaults to True, setting to False will improve the performance substantially in many cases.

**1)**

**Code:**

```
import pandas as pd df_left=pd.DataFrame({
      'Id':[79,78,77,76,75],
      'Name':['Isha','Aanchal','Nishita','Laveena','Muskan'],
       'Subject':['Python','Java','App_Dev','AOA','Web_Dev']
       })
#df_left=df_left.set_index("Id") df_right=pd.DataFrame({
      'Id':[79,78,77,76,75],
       'Name':['Dash','Lily','Noah','Sara','Joe'],
'Subject':['Gamer','Python','App_Dev','AOA','PM']
       })
#df_right=df_right.set_index('Id')
print(f'First Dataframe :::: \n{df_left}')
print(f'\n\nSecond Dataframe ::::
\n{df_right}')
#using on=''
print(f'\n\n\033[1m Merging Dataframe using on argument ::::\033[0m')
print(pd.merge(df_left,df_right,on=['Id','Subject']))
```

**Output:**

```
First Dataframe ::::
   Id     Name  Subject
0  79     Isha   Python
1  78  Aanchal     Java
2  77  Nishita  App_Dev
3  76  Laveena      AOA
4  75   Muskan  Web_Dev


Second Dataframe ::::
   Id  Name  Subject
0  79  Dash    Gamer
1  78  Lily   Python
2  77  Noah  App_Dev
3  76  Sara      AOA
4  75   Joe       PM


 Merging Dataframe using on argument ::::
   Id   Name_x  Subject Name_y
0  77  Nishita  App_Dev   Noah
1  76  Laveena      AOA   Sara
```

**2)**

**Code:**

```
#using how='outer'
```

```
print(f'\033[1m Merging Dataframe using how="outer" argument ::::\033[0m')
print(pd.merge(df_left,df_right,on='Subject',how='outer'))
#using how='inner'
print(f'\n\n\033[1m  Merging Dataframe using how="inner" argument ::::\033[0 m')
print(pd.merge(df_left,df_right,on='Subject',how='inner'))
#using how='left'
print(f'\n\n\033[1m  Merging Dataframe using how="left" argument :::\033[0m')
print(pd.merge(df_left,df_right,on='Subject',how='left'))
#using how='right'
print(f'\n\n\033[1m  Merging Dataframe using how="right" argument ::::\033[0m'
)
print(pd.merge(df_left,df_right,on='Subject',how='right'))
```

**Output:**

```
 Merging Dataframe using how="outer" argument ::::
   Id_x   Name_x  Subject  Id_y Name_y
0  79.0     Isha   Python  78.0   Lily
1  78.0  Aanchal     Java   NaN    NaN
2  77.0  Nishita  App_Dev  77.0   Noah
3  76.0  Laveena      AOA  76.0   Sara
4  75.0   Muskan  Web_Dev   NaN    NaN
5   NaN      NaN    Gamer  79.0   Dash
6   NaN      NaN       PM  75.0    Joe


 Merging Dataframe using how="inner" argument ::::
   Id_x   Name_x  Subject  Id_y Name_y
0    79     Isha   Python    78   Lily
1    77  Nishita  App_Dev    77   Noah
2    76  Laveena      AOA    76   Sara


 Merging Dataframe using how="left" argument :::
   Id_x   Name_x  Subject  Id_y Name_y
0    79     Isha   Python  78.0   Lily
1    78  Aanchal     Java   NaN    NaN
2    77  Nishita  App_Dev  77.0   Noah
3    76  Laveena      AOA  76.0   Sara
4    75   Muskan  Web_Dev   NaN    NaN


 Merging Dataframe using how="right" argument ::::
   Id_x   Name_x  Subject  Id_y Name_y
0   NaN      NaN    Gamer    79   Dash
1  79.0     Isha   Python    78   Lily
2  77.0  Nishita  App_Dev    77   Noah
3  76.0  Laveena      AOA    76   Sara
4   NaN      NaN       PM    75    Joe
```