Unit 14

Packages in Python

Table of Contents

9.1. INTRODUCTION

In Python programming so far, you have been introduced to the basics of the programming language. You have learned how to define and use variables and functions, the various operations and coding parameters applicable in programming, and how to use conditions to direct the program flow. The conditional statements (*if* and *else*) and loops (*for* and *while*) are used to set parameters to perform specific functions based on the given conditions. You learned about the various data types that provide a construct to store data and make accessing them more efficient. These data structures may be mutable (list, dictionary, set) or immutable (string, tuple, range, etc.).

Python is a multi-paradigm programming language and supports many programming approaches. The Object-Oriented Programming (OOP) approach is one of the most famous ones where programming problems are solved by creating objects. An **object** is a data that has attributes and shows specific behaviour. It is an instance of a class. A **class** is a blueprint of an object and defines the description of its objects. The Python OOPS follows the concept of inheritance to define new classes based on a pre-existing **base class**.

In the more recent units, you also learned about the different types of errors that arise during programming. If there is an error during program execution, Python throws a message known as an **exception**. Common errors like NameError or IndentationError can be resolved through simple changes in the code. However, logical or run-time errors are more difficult to detect. Programmers use the process of **exception handling** to test potential error and provide code to handle them.

In this unit, you will learn about the packages used in Python. As an application program size grows, it becomes more difficult to manage all the files we need to access. For efficient file organisation, we use packages in Python. A Python **package** can be understood as a set of directories and modules of files.

A **module** is any file that contains Python statements and definitions. We use modules to divide large programs into small manageable and organized files. Saving code in modules also allows a programmer to reuse them in different programs. Defined functions that are used often can simply be store in modules and imported into a program by using the import keyword. In packaging, similar modules are placed under one package and different modules are placed under different packages.

LEARNING OBJECTIVES

At the end of this unit, you will be able to:

- Use and import the Pandas package in Python. Use the pandas.Series, pandas.Dataframe, pandas.Panel classes. Use Pandas in descriptive statistics.
- Use and import the Matplotlib package in python. Use Matplotlib package in making a simple plot, bar plot, and histogram.
- Use and import the Django package in python. Use Apache, Sessions and Cooking, along with form processing inDjango.

14.2. PANDAS PACKAGE

Pandas is an open-source Python package or library that provides fast, flexible, and expressive data structures that help in efficient data manipulation and analysis. Before Pandas, Python was mostly used for **data munging** and preparation. It had nothing to do with data analysis. In 2008, Wes McKinney started developing Pandas with the intention to provide high performance, flexible tools for data analysis.

STUDY NOTE

A directory must have a file with the name "__init__.py".

Packages contain modules that are used to simplify and divide large programs into small manageable files. The Pandas library architecture consists of the following modules:

- **pandas/core:** Contains the data structures of the Pandas library.
- **pandas/src:** Contains basic functionality of Pandas based on certain algorithms.
- **pandas/io:** Contains the tools used for input and output, files, data, etc.
- **pandas/tools:** Codes and algorithms for various functions and operations in Pandas are stored here.
- **pandas/sparse:** Contains the spares versions, that is, those versions that handle missing values of various Data Structures in Pandas.
- **pandas/stats:** The functions related to statistics, like linear regression, are stored here.
- **pandas/util:** Consist of testing tools and various other utilities for library debugging.
- **pandas/rpy:** Contains the R2Py interface used to convert data to R.

Another open-source package that is similar to Pandas is the NumPy package. Unlike Pandas which works on tabular data, NumPy provides multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation. NumPy stores similar data (integers) in the form of homogeneous multidimensional arrays.

14.2.1. USE OF PANDAS PACKAGE

The five typical steps in the processing and analysis of data--load, prepare, manipulate, model, and analyze--can be achieved through Pandas. Data analysis is a crucial part of many industries. Python with Pandas is used in a wide range of academic and commercial fields like finance, economics, statistics, analytics, etc.

STUDY NOTE

The word "Pandas" is derived from the term "panel data" which is a econometrics term for data sets.

One of the most essential uses of Pandas is in data analysis. The Pandas library can handle huge sets of data. It helps users analyse them easily by providing the tools to clean and filter data. Some of the sectors that use Pandas for data analysis include--economics (to find trends and similarities), statistics (to perform various statistical operations), web analysis (to read and analyze the traffic of a website), etc.

The Pandas library is also used in Machine Learning as it helps to provide data for a model to learn and predict results. In financing, machine learning is used to predict stocks (Pandas is used to handle data of previous stock market dealings). Machine learning also contributes to Natural Language Processing (NLP) that is used to understand the human language and its intricacies.). Using the functions and features in Pandas, programmers can also manipulate data that are present in datasets. Other features of Pandas package include:

- Handling missing data values in data sets.
- Tools to load data into in-memory data objects from various formats.
- Slicing, indexing and subsetting of large data sets based on data labels.
- Ordered and unordered time series data.
- Provide functions for merging, concatenating, grouping, and munging data.

14.2.2. IMPORT INTO PYTHON

As Pandas is a third-party library, the standard Python distribution does not come with an in-built Pandas package. To use the Pandas package, users have to install it into the Python directory.

The easiest way to do this is by installing the **Anaconda distribution**, which is a cross-platform distribution for data analysis and scientific computing. Through this method, users can not only install Pandas but also other popular packages that make up the SciPy stack (IPython, NumPy, Matplotlib, etc.) often used in Python. The cross-platform distribution supports installation on Linux, macOS, and Windows.

To use the Pandas package in Python, users have to use the import keyword as follows:

```
import pandas as pd
```

"pd" is a term assigned to the Pandas library so users can access it easily as 'pd.<command>' in a program.

STUDY NOTE

SciPy is an open-source Python library used to solve scientific and mathematical problems. It is built on the NumPy extension.

Handling ImportError

Users may sometimes encounter an ImportError when trying to access the Pandas library. Python internally has a list of directories it searches through, to find packages. However, if users have multiple Python installations on their system, they may need to check if they are using the installation with the Pandas library installed in it. Users can do this by running "which python" on their terminal.

14.2.3. PANDAS.SERIES

Series is defined as a one-dimensional array that is capable of storing data of any data type (integer, float, string, etc.). The elements or values of a Pandas series are mutable (can be changed), however, the size of a series cannot be changed. All the elements in a

series are indexed, where the first element has the index "0". A Pandas series is created using the following syntax and parameters:

```
pandas.Series( data, index, dtype, copy)
```

- **data -** It can take any form like ndarray, list, constants.
- **index -** The index values must be unique, hashable, and the same length as the data. "np.arrange(n)" is passed by default if no index is passed.
- **dtype -** This is the data type for the output Series. If not specified, it is inferred from the data.
- **copy -** Copy input data. It is "False" by default.

STUDY NOTE

A Pandas series can be created by using an array, dictionary, scalar value or a constant.

Creating an Empty Series

A basic series in Pandas, the empty series, is created as follows:

```
# import the Pandas library and alias as "pd"
import pandas as pd
s = pd.Series()
print (s)

Output:
Series([], dtype: float64)
```

Creating a Series from ndarray

If a data is of type ndarray, the index passed must be of the same length as the data. If no index is passed, the default index will be range(n), where n is the length of the array.

```
# array to series without index passed
import pandas as pd
import numpy as np

data = np.array(['p', 'b', 'q', 'd'])
s = pd.Series(data)
print (s)

Output:
0 p
1 b
2 q
3 d
dtype: object
```

The NumPy library is often used along with the Pandas library as they are both very useful for scientific computing with Python.

Accessing Data from Series

We can access the data in a series based on the in-built index as follows:

```
import pandas as pd
s = pd.Series([1, 2, 3, 4, 5],index = ['a','b','c','d','e'])

#retrieve the first element
print (s[0])

Output:
4

# Range of elements
import pandas as pd
s = pd.Series([1, 2, 3, 4, 5],index = ['a','b','c','d','e'])

#retrieve the first element
print (s[2:4])

Output:
c 3
d 4
dtype: int64
```

Retrieve Data Using Labels

The index labels given to the series elements can be used as follows:

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve a single element
print (s['d'])

Output:
4

# multiple elements
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve multiple elements
print (s[['a','c','d']])

Output:
a 1
c 3
d 4
dtype: int64
```

14.2.4. PANDAS.DATAFRAME

A Pandas Dataframe is a two-dimensional structure that organises data in the form of rows and columns. The columns of a Dataframe can be of different data types and each row and column can be labeled. Pandas Dataframes are mutable and users can perform various arithmetic operations on the rows and columns.

A Pandas dataframe can be created by the following construct:

pandas.DataFrame(data, index, columns, dtype, copy)

STUDY NOTE

A pandas DataFrame is created by using Lists, dict, Series, Numpy ndarrays, or another DataFrame as input.

- **data -** Can be of various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
- **index -** For the row labels, "np.arrange(n)" is passed by default if no index is passed.
- **columns -** For column labels, "np.arrange(n)" is passed by default if no index is passed.
- **dtype -** The data type of each column.
- **copy -** Command used for copying data. It is "False" by default.

Creating an Empty DataFrame

```
import pandas as pd
df = pd.DataFrame()
print (df)

Output:
Empty DataFrame
Columns: []
Index: []
>
```

Creating a DataFrame from a Dictionary of ndarrays / Lists

To create this type of Dataframe, all the arrays must be of the same length and the index (if passes) should be the same length as arrays.

```
# without passing index
import pandas as pd
d = {'Name':['Tommy', 'Jill', 'Stone', 'Nicky'],'Age':[28,34,29,42]}
df = pd.DataFrame(d)
print (df)

Output:
Name Age
0 Tommy 28
1 Jill 34
2 Stone 29
3 Nicky 42
```

```
# by passing index
import pandas as pd
d = {'Name':['Tommy', 'Jill', 'Stone', 'Nicky'],'Age':[28,34,29,42]}
df = pd.DataFrame(d, index=['rank1','rank2','rank3','rank4'])
print (df)

Output:
Name Age
rank 1 Tommy 28
rank 2 Jill 34
rank 3 Stone 29
rank 4 Nicky 42
```

Selecting a Column

```
import pandas as pd

d = {'One' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'Two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

#selecting column two from dataframe
df = pd.DataFrame(d)
print (df ['One'])

Output:
a 1.0
b 2.0
c 3.0
d NaN
Name: One, dtype: float64
>
```

Adding and Deleting a Column

```
# adding a column
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

# Adding a new column to an existing DataFrame object with a column label by passing new series

print ("Adding a new column by passing a Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print (df)
```

```
print ("Adding a new column using the existing columns in DataFrame:")
df['four']=df['one']+df['three']

print (df)
```

Output:
Adding a new column by passing a Series:
one two three
a 1.0 1 10.0
b 2.0 2 20.0
c 3.0 3 30.0
d NaN 4 NaN

Adding a new column using the existing columns in DataFrame:
one two three four
a 1.0 1 10.0 11.0
b 2.0 2 20.0 22.0
c 3.0 3 30.0 33.0
d NaN 4 NaN NaN

```
# deleting a column
# delete a column using the del function
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
   'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)
print ("Our dataframe is:")
print (df)

# using del function
print ("Deleting the first column using DEL function:")
del (df['one'])
print (df)
```

Output:
Our dataframe is:
one two three
a 1.0 1 10.0
b 2.0 2 20.0
c 3.0 3 30.0
d NaN 4 NaN

Deleting the first column using DEL function:
two three
a 1 10.0
b 2 20.0

```
c 3 30.0
d 4 NaN
```

Selecting, Adding, and Deleting Rows

The loc() function is used to select rows

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print (df.loc['c'])
```

Output:

```
one 3.0
two 3.0
Name: c, dtype: float64
```

The iloc() function can be used to pass an integer location to select a row.

Example:

```
print (df.iloc[2])
```

The ':' operator is used to select multiple rows where the index values are passed.

Example:

```
print (df[2:4])
```

The append() function can be used to add new rows to the Dataframe. The row is appended at the end.

Example:

```
df = df.append(df2)
print (df)
```

The drop() function is used to delete rows by using the label index. If multiple rows have the same label index, all these rows are deleted.

Example:

```
df = df.drop(0)
print (df)
```

14.2.5. PANDAS.PANEL

A Panel is a three-dimensional data structure. The 3 axes that are used to give a Panel a semantic meaning to describing operations are:

- **items** – axis 0, each item corresponds to a DataFrame contained inside.
- **major_axis** – axis 1, the index (rows) of each of the DataFrames.
- **minor_axis** – axis 2, the columns of each of the DataFrames.

A Panel is created using the following construct:

```
pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)
```

Creating a Panel from a ndarray and a Dict of Dataframes

```
# from ndarray
import pandas as pd
import numpy as np

data = np.random.rand(2,4,5)
p = pd.Panel(data)
print (p)

Output:
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 4 (major_axis) x 5 (minor_axis)
Items axis: 0 to 1
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 4

# from dict of Dataframes
import pandas as pd
import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
   'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print (p)
```

Selecting Data From a Panel

We can select data in a Panel by using the three axes.

By Using Item

```
import pandas as pd
import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
   'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print (p['Item1'])

Output:
      0             1             2
0    0.488224     -0.128637      0.930817
1    0.417497    0.896681    0.576657
2    -2.775266     0.571668  0.290082
3    -0.400538   -0.144234   1.110535
```

In the above code, we have two items and we retrieved one of them. The result is a DataFrame with 4 rows and 3 columns, which are the **Major_axis** and **Minor_axis** dimensions.

By Using major_axis

We can use the method **panel.major_axis(index)**.

```
import pandas as pd
import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
   'Item2' : pd.DataFrame(np.random.randn(4, 2))}

p = pd.Panel(data)
print (p.major_xs(1))
```

Output:

```
      Item1         Item2
0    0.417497    0.748412
1    0.896681    -0.557322
2    0.576657    NaN
```

By Using minor_axis

We can use the method **panel.minor_axis(index)**.

```
import pandas as pd
import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
   'Item2' : pd.DataFrame(np.random.randn(4, 2))}

p = pd.Panel(data)
print (p.minor_xs(1))
```

Output:

```
       Item1           Item2
0    -0.128637   -1.047032
1    0.896681    -0.557322
2    0.571668    0.431953
3    -0.144234   1.302466
```

14.2.6. DESCRIPTIVE STATISTICS

Descriptive statistics is defined as the study of data analysis to describe, show or summarize data in a meaningful manner. A vast number of methods and operations collectively help us compute descriptive statistics on a Dataframe. These methods usually

take an **axis** argument, similar to *ndarray.{sum, std, ...}*, however, the axis can be specified by name or integer.

For this section, we will be using the descriptive statistics methods on the following Dataframe:

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])}

#Create a DataFrame
df = pd.DataFrame(d)
print (df)
```

Sum() - Returns the sum of the values for the requested axis.

```
print df.sum(1)

Output:
0 29.23
1 29.24
2 28.98
3 25.56
4 33.20
5 33.60
6 26.80
7 37.78
8 42.98
9 34.80
10 55.10
11 49.65
dtype: float64
```

STUDY NOTE

'include' is the argument used to pass information regarding what columns need to be considered for summarizing (object, number, all).

Column 2 is added.

Mean() - Gives the average value of the columns.

```
print df.mean()

Output:
Age 31.833333
Rating 3.743333
dtype: float64
```

The **describe()** function gives a summary of the statistics related to the Dataframe columns. It gives the **mean, std** and **IQR** values.

Other Functions:

Function	Description
count()	Number of non-null observations
median()	Median of Values
mode()	Mode of values
std()	Standard Deviation of the Values
min()	Minimum Value
max()	Maximum Value
abs()	Absolute Value
prod()	Product of Values
cumsum()	Cumulative Sum
cumprod()	Cumulative Product

SELF-ASSESSMENT QUESTIONS

1) _____ component of Pandas stored data in a tabular form.

2) A _____ is any file that contains Python statements and definitions.

3) The _____ library is often used along with the Pandas library as they are both very useful for scientific computing with Python.

4) The data in a Series can be ndarray, list, constants. [True/False]

5) The loc() function is used to select rows by passing an integer in Pandas Dataframe. [True/False]

6) Which function is used to delete a row in Dataframe?

A. drop() B. delete() C. both a and b D. none

7) _____ axes shows the columns of each of the DataFrames.

A. items B. major_axis C. minor_axis D. all

8) _____ function gives the standard deviation of the values in descriptive statistics.

ACTIVITY 1

Create a Dataframe that have the following attributes of diamonds: price, carat, cut, colour, clarity, length, width, and depth. Write a Python program to rename two of the columns and remove the second column of the diamonds Dataframe.

14.3. MATPLOTLIB PACKAGE

The matplotlib is one of the most famous Python packages used for data visualisation. It is a cross-platform library used to make 2D plots from array data. Matplotlib has an interface known as Pylab that has been designed to resemble the MATLAB programming language.

14.3.1. USE OF MATPLOTLIB PACKAGE

Matplotlib was originally written by John D. Hunter in 2003 and offers an object-oriented API that helps to embed plots in applications that use Python GUI toolkits, such as PyQt, and WxPythonotTkinter.

Matplotlib is used to visually represent data in the form of bar graphs, pie charts, images, etc. in various designs. Using matplotlib users can represent statistical data as visual plots. The library is also used to create animations and widgets to help understand complex data.

STUDY NOTE

Matplotlib along with NumPy is considered the open-source equivalent of MATLAB.

It finds use in a wide range of sectors such as finance, economics, science, academics, etc. It is also used in Machine Learning to understand data visually.

14.3.2. IMPORT INTO PYTHON

Matplotlib and its related packages (available in the form of wheel packages) can be installed into Python by using the pip package manager.

```
pip install matplotlib
```

Once it is installed, it can be imported into Python as shown:

```
import matplotlib.pyplot as pt
```

STUDY NOTE

If a single list or array is provided to the plot() command, matplotlib automatically assigns it to the v-axis.

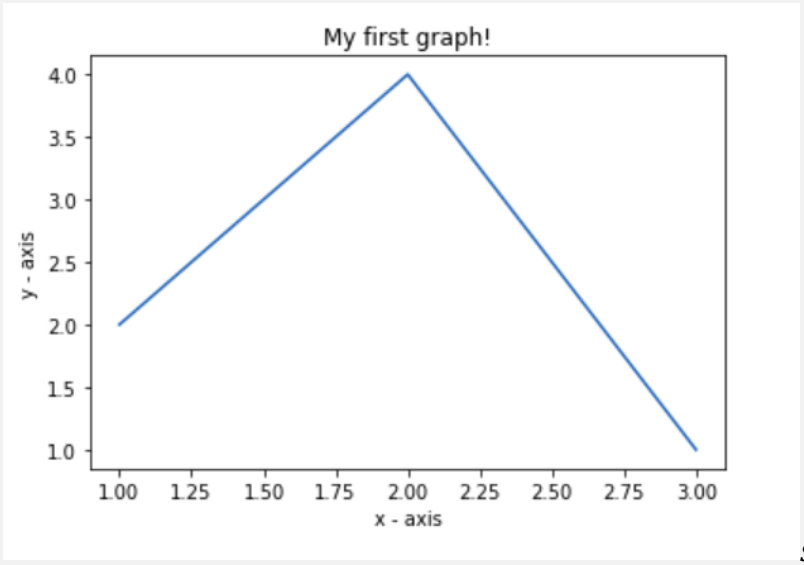
The module we wish to import can be specified by appending ".pyploy" to matplotlib. The library is abbreviated to "pt" for easier use.

14.3.3. SIMPLE PLOT

The matplotlib.pyplot is a collection of functions that help matplotlib work like MATLAB. Each function in the pyplot brings changes in the plot. Let us see how a simple plot is created using matplotlib through an example.

```
# importing the required module
import matplotlib.pyplot as plt

# x axis values
x = [1,2,3]
# corresponding y axis values
y = [2,4,1]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('My first graph!')

# function to show the plot
plt.show()
```

Output:



***Source**:geeksforgeeks.org*

Fig 1: Simple Plot

The lines in a simple plot have many attributes that can be changed such as line width, colour, shape, etc. The easiest way to do this is by using keyword arguments as shown:

```
# sets the line width to 2.0 for both x and y
plt.plot(x, y, linewidth=2.0)
```

You can set multiple properties of a line by using the MATLAB style set() command.

```
# line colour is red and width is 2.0
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

Some of the basic functions used in matplotlib include:

Method	Description
plot()	Creates a plot in the background but does not display it.
show()	displays the created plot
xlabel()	labels x-axis
ylabel()	labels y-axis
title()	gives title to the plot.
gca()	helps to get access over all the four axes.
gca().spines['right/left/top/bottom']. set_visible(True/False)	helps to access the individual boundaries and helps to change their visibility.
xticks()	decides how marking should be made on the x-axis.
yticks()	decides how marking should be made on the y-axis.
annotate()	used to write commands at specified positions on the graph.
subplot(r, c, i)	used to create multiple plots in the same figure with 'r' signifies the no of rows in the figure, 'c' signifies no of columns in a figure and 'i' specifies the positioning of the particular plot.

14.3.4. BAR PLOT

A bar graph or plot is used to present and compare data in discrete categories. The matplotlib API provides the **bar()** function that can be used in MATLAB as well as the object-oriented API format. The bar() function is used in the format and parameters:

```
ax.bar(x, height, width, bottom, align)
```

x - A sequence of scalars that represent the x-coordinates of the bars.
height - scalar(s) representing the height(s) of the bars.
width - scalar or array-like sequences. These are optional. The default value is 0.8.
bottom - scalar or array-like sequences. These are optional. Represent the y-coordinate of the bars. The default value is None.
align - {'center', 'edge'}, optional, default is 'center'.

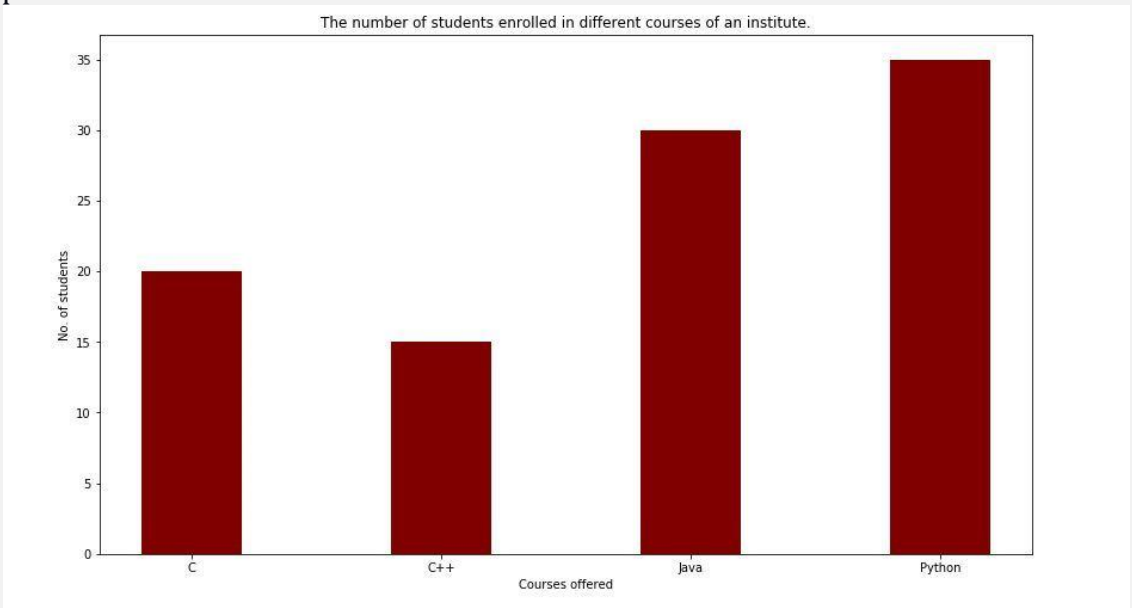
STUDY NOTE

The optional bottom parameter of the pyplot.bar() function allows you to specify a starting value for a bar.

Example:

```
import numpy as np
import matplotlib.pyplot as plt

# creating the dataset
data = {'C':20, 'C++':15, 'Java':30,
'Python':35}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(courses, values, color ='maroon',
width = 0.4)

plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```

Output:



***Source**: geeksforgeeks.org*

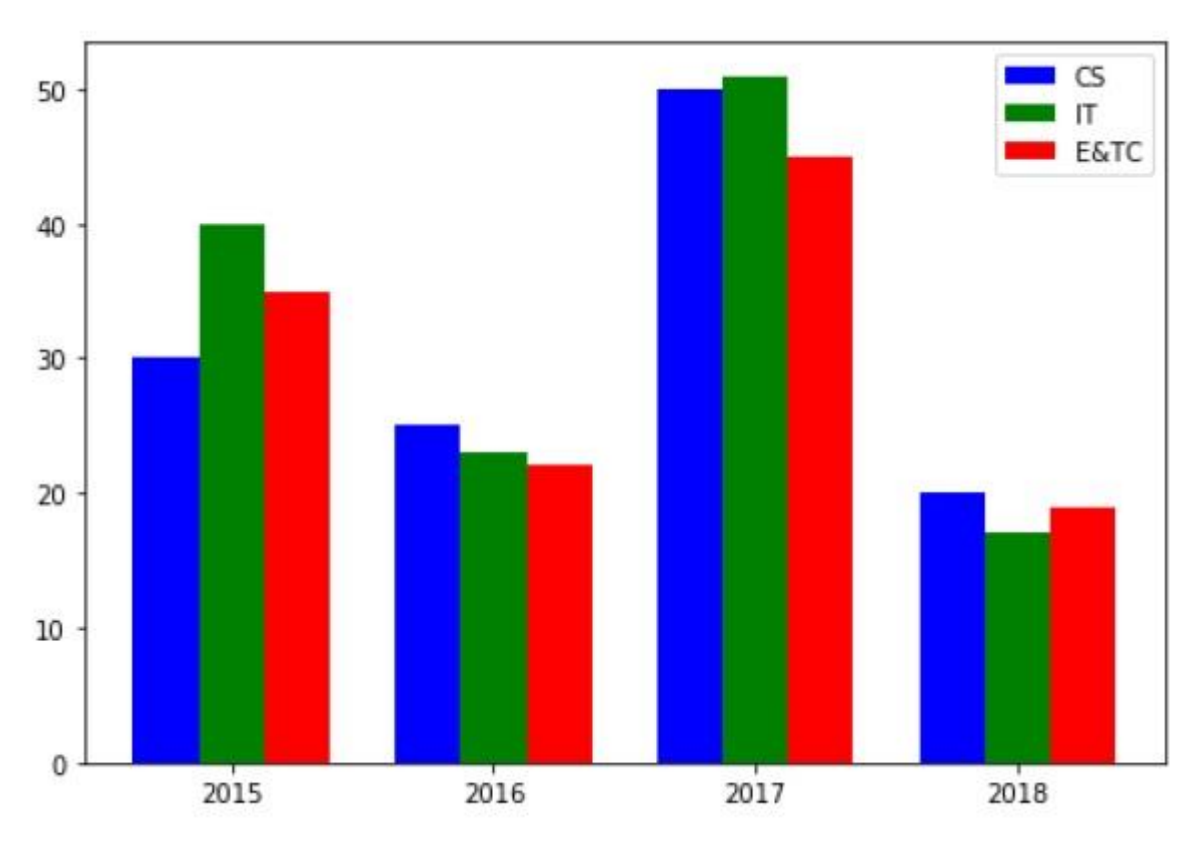
Fig 2: Simple Bar Plot

We can arrange and compare several quantities as a bar plot as shown:

```
import numpy as np
import matplotlib.pyplot as plt

# values of the bars
data = [[30, 25, 50, 20],
[40, 23, 51, 17],
[35, 22, 45, 19]]
```

```
X = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(X + 0.00, data[0], color = 'b', width = 0.25)
ax.bar(X + 0.25, data[1], color = 'g', width = 0.25)
ax.bar(X + 0.50, data[2], color = 'r', width = 0.25)
```

Output:



***Source**: tutorialspoint.com*

Fig 3: Multiple Bar Plots in One

The above example shows a series of four bar plots with three values each. Each bar has a thickness of 0.25 units and each bar chart is shifted 0.25 units away from the previous one. The plot has bars of one color for one quantity value.

14.3.5. HISTOGRAM

A histogram is a type of bar graph that presents an accurate representation of numerical data. It is used to show an estimate of the probability distribution of a continuous variable.

STUDY NOTE

The bins are usually specified as consecutive, non-overlapping intervals of a variable.

To build a histogram, first, you need to Bin the range of values. Divide the range into a series of intervals, and then count how many values fall into each interval. A histogram is plotted with the help of

the **matplotlib.pyplot.hist()** function. The function computes and draws the histogram of x.

Parameter of a histogram:

x - Array(s)
bins - integer or sequence or 'auto', optional

Other optional parameters:

range - Sets the lower and upper range of the bin.
density - If set as "True", the first element of the return tuple is used to form a probability density.
cumulative - If set as "True", a histogram is computed in which each bin gives the counts in that bin along with all the bins of smaller values.
histtype - Defines the type of histogram to draw [bar, barstacked, step, stepfilled]. Bar is the default type.

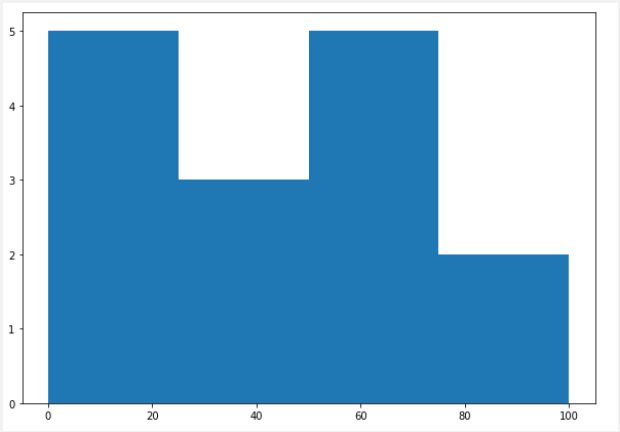
STUDY NOTE

Thematplotlib.pyplot.hist() function provides many attributes using which we can modify a histogram.

Example of a histogram:

```
from matplotlib import pyplot as plt
import numpy as np
# Creating dataset
a = np.array([22, 87, 5, 43, 56,
73, 55, 54, 11,
20, 51, 5, 79, 31,
27])

# Creating histogram
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(a, bins = [0, 25, 50, 75, 100])

# Show plot
plt.show()
```

Output:



***Source**: geekforheeks.com*

Fig 4: A Histogram

ACTIVITY 2:

Create a bar plot that shows scores by group and gender. Use different x values for men and women in the same chart.

SELF-ASSESSMENT QUESTIONS

9) _____ library is used to represent data arrays as visual plots.

10) Matplotlib along with _____ is considered the open-source equivalent of MATLAB.

11) _____ function offers a patches object which gives us access to the properties of the created objects.

12) Matplotlib and its related packages can be installed into Python by using the _____ package manager.

13) "r" in subplot(r, c, i) specifies the positioning of the particular plot. [True/False]

14) The pyplot.bar() function allows you to specify a starting value for a bar. [True/False]

15) Which parameter in a bar plot represent the y-coordinate of the bars.

a) height b) bottom c) ylabel d)align

16) _____ is the default type of histogram.

14.4. DJANGO PACKAGE

14.4.1. USE OF DJANGO PACKAGE

Django is a web development framework that helps users develop and maintain web applications. It eliminates the need for repetitive tasks which makes web development an easy and time-saving process. It provides a high-level framework that encourages rapid development and displays a clean, pragmatic design in its code. Each element in a Django stack is independent of the other.

STUDY NOTE

Django is a registered trademark of the Django Software Foundation and is licensed under BSD License.

Django's basic role is to ease the process of creating complex database-driven websites. It provides a bridge between the data model and the database engine. It supports a vast collection of database systems like MySQL, Postgres, Oracle, etc. Its built-in internationalization system also supports multilingual websites.

STUDY NOTE

The hist() function offers a patches object which gives us access to the properties of the created objects.

Django has a built-in for frameworks like Ajax, RSS, Caching, etc. and also provide an easy-to-use user interface for administrative activities. Users can also develop and test their applications as Django comes with a lightweight web server.

Django works based on the Model-View-Template (MVT) pattern which is slightly different from the Model-View-Controller (MVC) pattern used in a web framework. Django takes care of the Controller part (Software Code that controls the interactions between the Model and View), while users are left with the template. The template is an HTML file mixed with Django Template Language (DTL).

14.4.2. IMPORT INTO PYTHON

Django can be installed into Python by using the pip or easy_install package managers (similar to the matplotlib package installation).

STUDY NOTE

A Web framework is a set of components that offer a standard way to develop websites quickly and easily.

First, go to the Python Package Index (PyPI) to install easy_install (You will have to download setuptools, which includes easy_install). Download the package egg (.egg), and then install it directly from the file.

The pip package manager can be installed by the following command:

```
easy_install pip
```

Every web application we create is called a project which is a sum of applications (a set of code file that relies on the MVT pattern). For example, if you want to create a website, the website itself is your project while the forum, new, contact engine are the applications (every application is independent).

STUDY NOTE

The DEBUG option lets you set if your project is in debug mode or not. Never set it to 'True' for a live project unless you want the Django light server to serve static files.

You can create a project by using:

```
django-admin startproject myproject
```

This will create a project named "myproject" with the following structure:

```
myproject/
   manage.py
   myproject/
      __init__.py
      settings.py
      urls.py
      wsgi.py
```

14.4.3. APACHE IN DJANGO

The Django **dev** web server is used only for testing and is not suitable for a production environment. For this, you will need to use a real server like Apache.

Example:

To be able to share our project "myproject" with Apache, we need to set Apache to access our folder. If we place our project in the default "/var/www/html", the project will be accessed via 127.0.0.1/myproject. In this case, Apache will simply list the folder as shown:

STUDY NOTE

The wsgi.py in the project structure takes care of the link between Django and Apache.

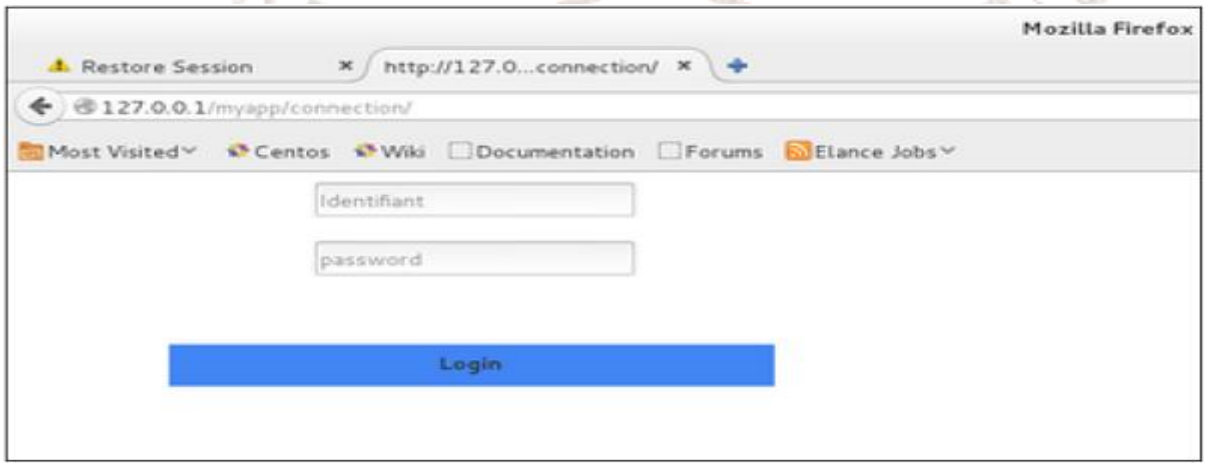
***Source**: tutorialspoint.com*

Fig 5: Project Access Via 127.0.0.1/myproject

For Apache to handle Django the way we want, we need to configure Apache in httpd.conf. To do this open the httpd.conf and add the following command:

```
WSGIScriptAlias / /var/www/html/myproject/myproject/wsgi.py

WSGIPythonPath /var/www/html/myproject/

<Directory /var/www/html/myproject/>

<Files wsgi.py>

        Order deny,allow

        Allow from all

</Files>

</Directory>
```

This will give you the following output when the page is accessed through 127.0.0.1/myapp/connection.



***Source**: tutorialspoint.com*

Fig 6: Project Access Via 127.0.0.1/myapp/connection

14.4.4. DJANGO FORM PROCESSING

We can create forms by using Django by inheriting the from Django class. The class attributes will become the form fields. To get started, we need to create a **forms.py** file in the main project folder ("myapp" for this example) that will contain all our app forms. You can create the login page in the following manner.

```
from django import forms

class LoginForm(forms.Form):
   user = forms.CharField(max_length = 100)
   password = forms.CharField(widget = forms.PasswordInput())
```

We want the password to be hidden and hence the password field can take "widget" arguments for html rendering.

GET and POST are the two HTTP methods used when dealing with forms in Django. The login form is returned using the POST method. In this, the browser bundles up the form data; encodes it for transmission, sends it to the server, and then receives its response.

On the other hand, GET bundles the data into a string, and uses this to create a URL. The URL contains the address where the data must be sent, as well as the data keys and values.

Let us create a login view for the myapp/views.py:

```
from myapp.forms import LoginForm

def login(request):
   username = "not logged in"

   if request.method == "POST":
      #Get the posted form
      MyLoginForm = LoginForm(request.POST)

      if MyLoginForm.is_valid():
         username = MyLoginForm.cleaned_data['username']
   else:
      MyLoginForm = Loginform()
   return render(request, 'loggedin.html', {"username" : username})
```

The above code will display the result of the login form posted through the **loggedin.html.** However, to test this, we first need the login form template ("login.html" in this example).

```
<html>
```

```
  <body>
     <form name = "form" action = "{% url "myapp.views.login" %}"
       method = "POST">{% csrf_token %}
              <div style = "max-width:470px;">
         <center>
            <input type = "text" style = "margin-left:20%;"
              placeholder = "Identifiant" name = "username" />
         </center>
       </div>
       <br>
                 <div style = "max-width:470px;">
         <center>
            <input type = "password" style = "margin-left:20%;"
              placeholder = "password" name = "password" />
         </center>
       </div>
       <br>

       <div style = "max-width:470px;">
         <center>
           <button style = "border:0px; background-color:#4285F4; margin-top:8%;
              height:35px; width:80%;margin-left:19%;" type = "submit"
              value = "Login">
              <strong>Login</strong>
            </button>

         </center>
       </div>
      </form>
    </body>
</html>
```

The above template will display a login form and post the result to our login view.
Now that we have the login template, we need the loggedin.html template that will be given after form treatment.

STUDY NOTE

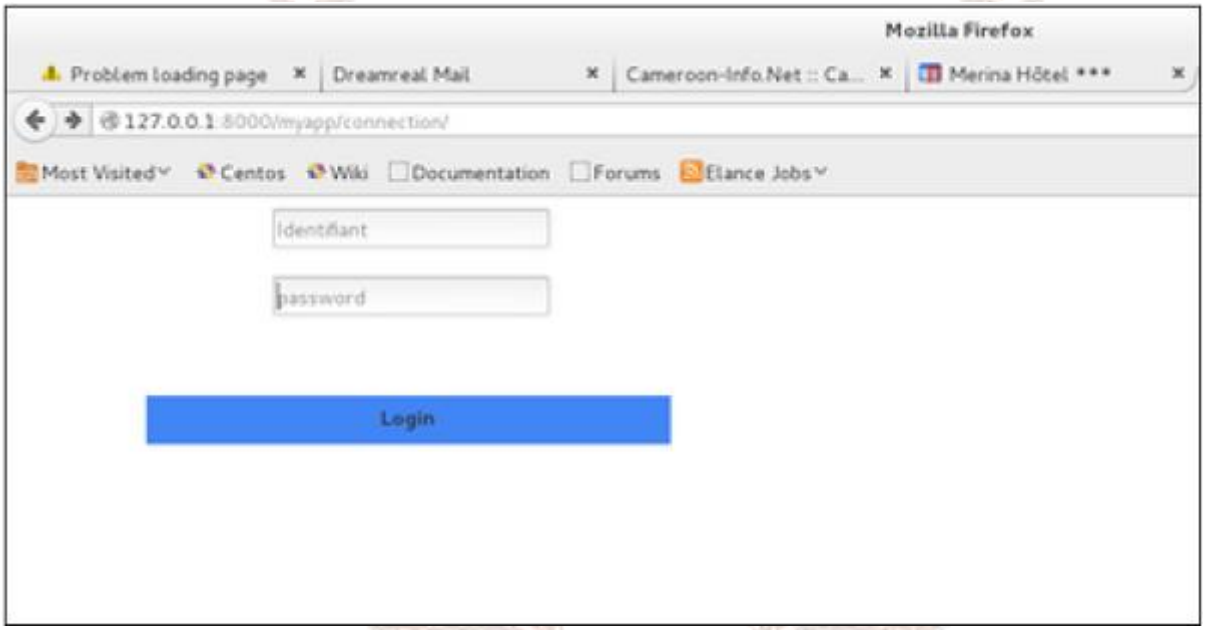
The {% csrf_token %} tag is used to prevent a Cross-site Request Forgery (CSRF) attack on a website.

```
<html>

<body>

        You are : <strong>{{username}}</strong>

</body>

</html>
```

Now, to get started, we need a pair of URLs (myapp/urls.py)

```
from django.conf.urls import patterns, url
from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views',
    url(r'^connection/',TemplateView.as_view(template_name = 'login.html')),
    url(r'^login/', 'login', name = 'login'))
```

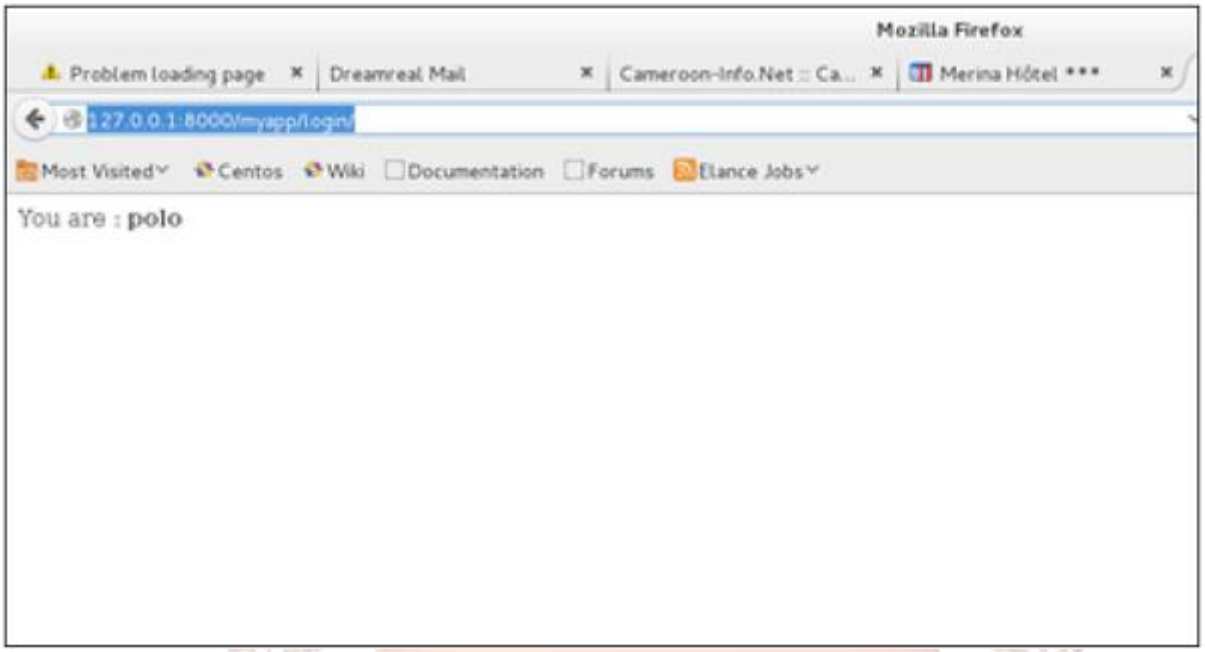
The following login template is rendered, when you try to access "/myapp/connection"



***Source**: tutorialspoint.com*

Fig 7: Accessing login Template

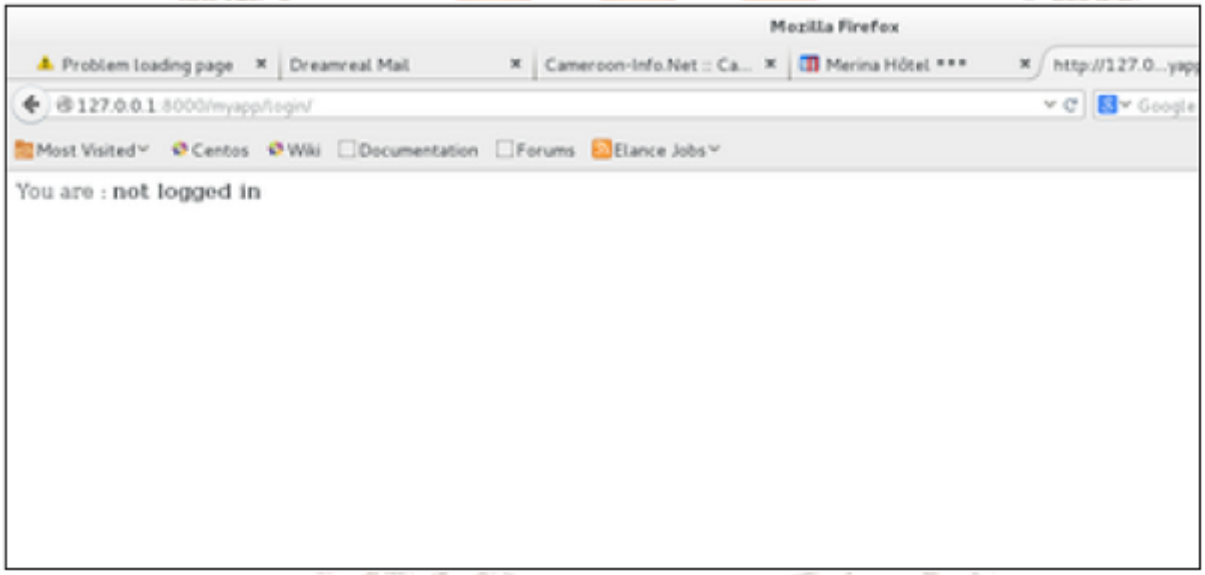
When you fill the two fields in the login form, you will get the following result:



***Source**: tutorialspoint.com*

Fig 8: Correct Login

If you forget the password:



***Source**: tutorialspoint.com*

Fig 9: Forgot Password

14.4.5. COOKIES

Sometimes we wish to save data on a per-site-visitor basis on our website. They are stored in the form of HTTP cookies that are stored locally by a browser.

Django provides methods like set_cookie() (to create a cookie) which help users bypass a lot of work that is usually required for cookies.
The set_cookie() method has the following attributes:

name: Specifies the name of the cookie.
value: Specifies the text or variable you want to store in the cookie.
max_age: The time period of a cookie in seconds, after which it will expire.

STUDY NOTE

HTTP is a stateless protocol and when a request is sent to the server, it cannot distinguish whether the user is new or has visited the site previously.

Creating a cookie

```
def setcookie(request):
html = HttpResponse("<h1>Dataflair Django Tutorial</h1>")
html.set_cookie('dataflair', 'Hello this is your Cookies', max_age = None)
return html
```

Retrieve Data From a Cookie

(i) Using request.COOKIES[]

```
request.COOKIES['cookie_name']
```

Example:

```
def showcookie(request):
show = request.COOKIES['dataflair']
html = "<center> New Page <br>{0}</center>".format(show)
return HttpResponse(html)
```

(ii) Using request.COOKIES.get()

```
COOKIES.get('cookie_name', 'value')
```

Example:

```
from django.shortcuts import render, redirect
from django.http import HttpResponse

def setcookie(request):
html = HttpResponse("<h1>Dataflair Django Tutorial</h1>")
    if request.COOKIES.get('visits'):
        html.set_cookie('dataflair', 'Welcome Back')
        value = int(request.COOKIES.get('visits'))
        html.set_cookie('visits', value + 1)
    else:
        value = 1
        text = "Welcome for the first time"
```

```
            html.set_cookie('visits', value)
            html.set_cookie('dataflair', text)
        return html
# it will check whether the request has cookies or not, then set the cookies according to
that
```

14.4.6. SESSIONS

Storing cookies can often lead to security holes in a website. Django offers a Sessions framework that abstracts the receiving and sending of cookies, saves data on the server-side (like in a database), and the client-side cookie has a session ID for identification.

STUDY NOTE

Django saves sessions information in a database by default. However, we can save this information in a file or a cache as well.

Sessions are enabled in a Django project by adding a few lines to the MIDDLEWARE_CLASSES and the INSTALLED_APPS options. **MIDDLEWARE_CLASSES** should contain:

```
'django.contrib.sessions.middleware.SessionMiddleware'
```

and **INSTALLED_APPS** should contain:

```
'django.contrib.sessions'
```

Let us look at an example where we save the username in a cookie and if not signed out, you won't see the login form when you try to access the login page.

```
def login(request):
    username = 'not logged in'

    if request.method == 'POST':
        MyLoginForm = LoginForm(request.POST)

        if MyLoginForm.is_valid():
            username = MyLoginForm.cleaned_data['username']
            request.session['username'] = username
        else:
            MyLoginForm = LoginForm()
    return render(request, 'loggedin.html', {"username" : username}
```

Create formView view for the login form

```
def formView(request):
    if request.session.has_key('username'):
        username = request.session['username']
        return render(request, 'loggedin.html', {"username" : username})
    else:
```

```
        return render(request, 'login.html', {})
# the form is not displayed if cookie is set

# change the url.py file to change the url so it pairs with the new view
from django.conf.urls import patterns, url
from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views',
   url(r'^connection/','formView', name = 'loginform'),
   url(r'^login/', 'login', name = 'login'))

# logout view that deletes the cookie
def logout(request):
   try:
      del request.session['username']
   except:
      pass
   return HttpResponse("<strong>You are logged out.</strong>")
```

ACTIVITY 3

Create a Django template that automates the process of sending emails.You can have a list of email addresses and their respective names. The massages can be modified and sent to the target audience automatically.

SELF-ASSESSMENT QUESTIONS

17) Django works based on the _____ pattern used in a web framework.

18) The _____ file in a project structure takes care of the link between Django and Apache.

19) _____ and _____are the two HTTP methods used when dealing with forms in Django.

20) POST bundles the data into a string, and uses this to create a URL. [True/False]

21) The {% csrf_token %} tag is used to prevent a Cross-site Request Forgery (CSRF) attack on a website. [True/False]

22) Which attribute of the set_cookie() method helps to set the time period of a cookie.

A. value
B. name
C. max_age
D. age

23) Information from Django sessions can be saved as:

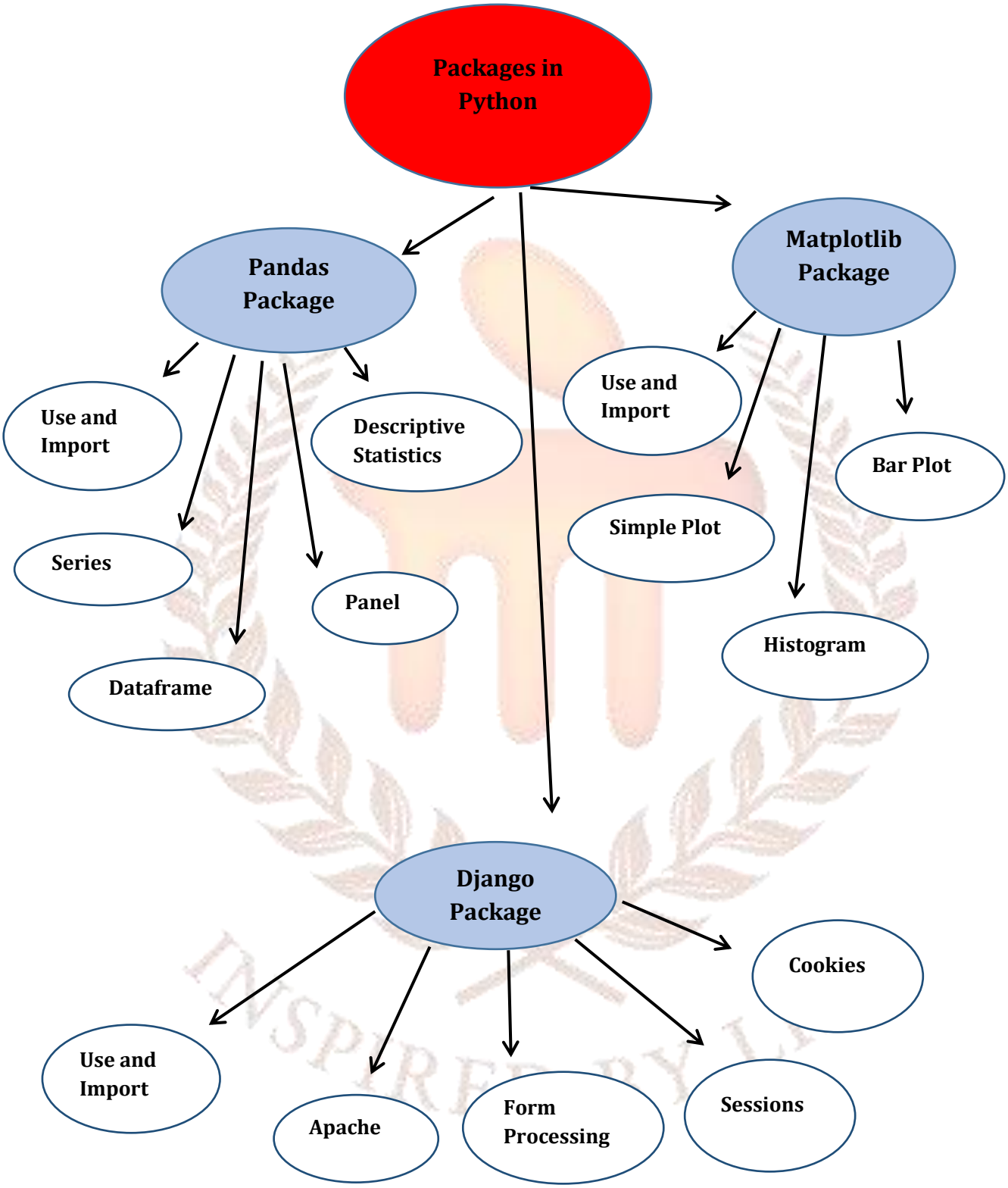
A. database
B. Cache
C. file
D. all

14.5. SUMMARY

- A module is any file that contains Python statements and definitions. They are used to divide a huge code into small manageable and organized files.
- A Python package is a set of directories and modules of files.
- Pandas package is used to efficiently manipulate data and analyse it.
- Anaconda distribution is a cross-platform distribution for data analysis and scientific computing. It contains the NumPy and Pandas packages along with others.
- Series is a Pandas module and is a one-dimensional array that is capable of storing data of any data type.
- The elements or values of a Pandas series are mutable and indexed.
- A series can be created by using an array, dictionary, scalar value or a constant.
- Pandas Dataframe is used to organise data in a tabular form. Users can perform various arithmetic operations on the rows and columns.
- Pandas Panel represents data in a 3D structure. Three axes are used to describing operations--items, major_axis, and minor_axis. These can be used to select data from a panel.
- Descriptive Statistics is used to study data analysis and describe, show or summarize data in a meaningful manner.
- The Matplotlib package of Python is used to visually represent data as plots.
- We can create simple plots, bar graphs, animations, histograms, etc. using matplotlib. The **bar()** function is used to create bar plots.
- Django is a web development framework that helps users develop and maintain web applications.
- Django works based on the Model-View-Template (MVT) pattern of the web framework. Apache is a server that offers a production environment.
- We can create forms in Django and save user data in the form of cookies. Sessions are used to provide a layer of security to the cookies.

14.6. GLOSSARY

- **Module -** files that contain Python definitions and parameters.
- **Data munging -** Process of transforming and mapping **data** from raw data form into another format to make it appropriate and valuable for downstream purposes like analytics.
- **ndarray -** array
- **dtype -** data type
- **loc() -** function used to select rows in a Pandas Dataframe.
- **axis 0 -** rows in a Pandas Dataframe.
- **axis 1 -** columns in a Pandas Dataframe.
- **Histogram -** a type of bar graph that presents an accurate representation of numerical data.
- **Webserver -** software and hardware that uses HTTP (Hypertext Transfer Protocol) and other protocols to respond to client requests made over the World Wide Web.
- **Cookies -** units that store data on a browser.
- **Cache-** short-term computer memory where information is stored for easy retrieval.


Packages in Python
Pandas Package
Use and Import
Descriptive Statistics
Series
Panel
Dataframe
Matplotlib Package
Use and Import
Bar Plot
Simple Plot
Histogram
Django Package
Cookies
Use and Import
Apache
Form Processing
Sessions

14.7. CASE STUDY

L'Oreal Men Expert

Moccu Digital Agency was assigned the task of creating a website to help L'Oreal relaunch the Men Expert project digitally. The agency decided to use the Django CMS to create the website. A Content Management System (CMS) helps users add, modify and delete the content as per their will.

It provides an interface that can be easily used by clients to maintain their websites, without any prior knowledge of back-end coding.

The web pages were not based on set templates and were made using the freely combinable elements in Django CMS. The CMS interface lets the content writers work directly on the pages where media can directly be inserted from a media database (with provisions for different formats and resolutions).

Source: *django-cms.org*

Fig 11: Website Created Using Django CMS

Discussion Questions

1) Discuss why Django CMS is preferred over Django.

2) Which possible tools did the company use in Django CMS to create their website?

14.8. TERMINAL QUESTIONS

SHORT ANSWER QUESTIONS

Q1. What is the difference between Pandas and NumPy?
Q2. What are sub-plots in matplotlib?
Q3. What is the difference between del, remove and pop functions in Python? Explain with examples.
Q4. What is Descriptive statistics?
Q5. What is your understanding on matplotlib?

LONG ANSWER QUESTIONS

Q1. What is the architecture of Django?
Q2. How do you set up a database in Django?
Q3. Show the different ways a Dataframe can be created with Python Pandas.
Q4. Write a code to add new column using Pandas.Dataframe?

ANSWERS

SELF-ASSESSMENT QUESTIONS:

1) Dataframe
2) Module
3) NumPy
4) True
5) False
6) A. drop(). drop() function is used to delete rows by using the label index.
7) C. minor_axis. axis 2, the columns of each of the DataFrames.
8) B. std(). Standard Deviation of the Values
9) Matplotlib
10) NumPy
11) hist()
12) Pip
13) False
14) True
15) B. bottom. scalar or array-like sequences that represent the y coordinate of the bars
16) B. bar is the default histogram.
17) Model-View-Template (MVT)
18) wsgi.py
19) GET, POST
20) False
21) True
22) C. max_age. The time period of a cookie in seconds, after which it will expire.
23) D. all. We can save sessions information as a database, a file or a cache.

TERMINAL QUESTIONS

SHORT ANSWER QUESTIONS:

Answer 1:

PANDAS	NUMPY
Used to work with tabular data	Used to work with numerical data
tools of pandas are Data frame and Series.	tool of numpy is Arrays.
consume more memory.	memory efficient.
Shows better performance when number of rows is 500K or more.	Shows better performance when number of rows is 50K or less.
Indexing of series is very slow	Indexing of Arrays is very fast.

Answer 2: Subplots are grids of plots within a single figure. Subplots can be plotted using *the subplots()* function from the matplotlib.pyplot module.

```
x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x ** 2)

# one figure with one subplot
fig, ax = plt.subplots(ncols=1,nrows=1)
ax.plot(x, y)
plt.plot()
```

Answer 3: remove() removes the first matching value in a given list
del() removes the item at a specific index
pop() removes the item at a specific index and returns it.

Answer 4: Descriptive statistics is defined as the study of data analysis to describe, show or summarize data in a meaningful manner. A vast number of methods and operations collectively help us compute descriptive statistics on a Dataframe.

These methods usually take an **axis** argument, similar to *ndarray.{sum, std, ...}*, however, the axis can be specified by name or integer

Answer 5: The matplotlib is one of the most famous Python packages used for data visualisation. It is a cross-platform library used to make 2D plots from array data. Matplotlib has an interface known as Pylab that has been designed to resemble the MATLAB programming language.

LONG ANSWER QUESTIONS:

Answer 1: Django is based on **MVT** architecture. It contains the following layers:

Models: Describes the database scheme and data structure.

Views: The view layer is the user interface. It controls what a user sees and retrieves data from appropriate models to execute any calculation made to the data and pass it to the template.

Templates: It determines how the user sees. It describes how the data received from the views should be changed or formatted for display on the page.

Controller: Controller is the heart of the system. It handles requests and responses, sets up database connections and loads add-ons. It specifies the Django framework and URL parsing.

Answer 2: To set up a database in Django, you can use the command edit mysite/setting.py.

By default, Django uses SQLite database. It is easy for Django users because it doesn't require any other type of installation. In the case of another database, you can use the following keys in the DATABASE 'default' item to match your database connection settings.

Engines: you can change database by using 'django.db.backends.sqlite3' , 'django.db.backeneds.mysql', 'django.db.backends.postgresql_psycopg2', 'django.db.backends.oracle' and so on

Name: The name of your database. In case you are using SQLite as your database, it will be a file on your computer. The name should be a full path, including the file name.

Note: You have to add setting like Password, Host, User, etc. in your database if you are not using SQLite as your database.

Answer 3: We can create a DataFrame using lists and Dict
Examaple:

```
# from lists
import pandas as pd
# a list of strings
a = ['Python', 'Pandas']
# Calling DataFrame constructor on list
info = pd.DataFrame(a)
print(info)

Output:

     0
0    Python
1    Pandas

# from dict
```

```
import pandas as pd
info = {'ID' :[101, 102, 103],'Department' :['B.Sc','B.Tech','M.Tech',]}
info = pd.DataFrame(info)
print (info)
```

Output:

	ID	Department
0	101	B.Sc
1	102	B.Tech
2	103	M.Tech

Answer 4: Code to add a column

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)

# Adding a new column to an existing DataFrame object with a column label by passing new series

print ("Adding a new column by passing a Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print (df)

print ("Adding a new column using the existing columns in DataFrame:")
df['four']=df['one']+df['three']

print (df)
```

Output:

```
Adding a new column by passing a Series:
one two three
a 1.0 1 10.0
b 2.0 2 20.0
c 3.0 3 30.0
d NaN 4 NaN

Adding a new column using the existing columns in DataFrame:
one two three four
a 1.0 1 10.0 11.0
b 2.0 2 20.0 22.0
c 3.0 3 30.0 33.0
d NaN 4 NaN NaN
```

14.9. SUGGESTED BOOKS AND E-REFERENCES

BOOKS:

- Gupta, A. (2020). Python Programming: Problem Solving, Packages and Libraries. McGraw Hill Education Pvt.
- Miller, C. (2015). Data Analysis with NumPy and Pandas. Packt Publishng. 2nd Edn.

E-REFERENCES:

- Matplotlib, viewed on 3 March 2021, < https://matplotlib.org/2.0.2/index.html >
- Pandas, viewed on 3 March 2021. < https://pandas.pydata.org/pandas-docs/stable/index.html >
- Tango with Django, Matplotlib, viewed on 3 March 2021, <https://www.tangowithdjango.com/>