

What is Python?

Python is a high-level, general-purpose, and very popular programming language. Python programming language is being used in web development, Machine Learning applications, along with all cutting-edge technology in Software Industry.



Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

1a) Write a python program to find the best of two test average marks out of three test's marks accepted from the user.

```
m1 = int (input ("Enter the marks in the first test: "))
m2 = int (input ("Enter the marks in second test: "))
m3 = int (input ("Enter the marks in third test: "))

if (m1 > m2):
    if (m2 > m3):
        total = m1 + m2
    else:
        total = m1 + m3
elif (m1 > m3):
    total = m1 + m2
else:
    total = m2 + m3

Avg = total / 2
print ("The average of the best two test marks is: ",Avg)
```

OUTPUT

```
Enter the marks in the first test: 45
Enter the marks in second test: 78
Enter the marks in third test: 23
The average of the best two test marks is: 61.5
```

1b. Develop a Python program to check whether a given number is palindrome or not and also count the number of occurrences of each digit in the input number.

```
val = int(input("Enter a value : "))
str_val = str(val)
if str_val == str_val[::-1]:
    print("Palindrome")
else:
    print("Not Palindrome")
for i in range(10):
    if str_val.count(str(i)) > 0:
        print(str(i),"appears", str_val.count(str(i)), "times")
```

OUTPUT 1

```
Enter a value : 1222221
Palindrome
1 appears 2 times
2 appears 5 times
```

OUTPUT 2

```
Enter a value : 12223
Not Palindrome
2  appears 1 times
2 appears 3 times
3  appears 1 times
4
```

2 a) Defined as a function F as $F_n = F_{n-1} + F_{n-2}$. Write a Python program which accepts a value for N (where $N > 0$) as input and pass this value to the function. Display suitable error message if the condition for input value is not followed.

Decision Constructs

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

In python, decision making is performed by the following statements.

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.

While loop

The Python while loop allows a part of the code to be executed until the given condition returns false. It is also known as a pre-tested loop.

It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.

The syntax is given below.

While

expression:

statements

Example:

```
count = 0
while (count < 9):
    print 'The count is:',
    count = count + 1
```

PROGRAM :

```
def fn(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fn(n-1) + fn(n-2)

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

if num > 0:
    print("fn(", num, ") = ",fn(num) , sep = "")
else:
    print("Error in input")
```

OUTPUT:

Enter a number : 0
Error in input

Enter a number : 7
fn(7) = 8

2 b) Develop a python program to convert binary to decimal, octal to hexadecimal using functions.

Python Function

Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code, which can be called whenever required. Python allows us to divide a large program into the basic building blocks known as a function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the Python program.

The Function helps to programmer to break the program into the smaller part. It organizes the code very effectively and avoids the repetition of the code. As the program grows, function makes the program more organized.

Python provide us various inbuilt functions like range() or print(). Although, the user can create its functions, which can be called user-defined functions.

There are mainly two types of functions.

User-define functions - The user-defined functions are those define by the user to perform the specific task. Built-in functions - The built-in functions are those functions that are pre-defined in Python.

Advantage of Functions in Python

- Using functions, we can avoid rewriting the same logic/code again and again in a program. We can call Python functions multiple times in a program and anywhere in a program.
- We can track a large Python program easily when it is divided into multiple functions. Reusability is the main achievement of Python functions.

However, Function calling is always overhead in a Python program. Creating a Function. Python provides the def keyword to define the function. The syntax of the define function is given below. Syntax:

```
def
    my_func
    tion(para
    meters):
    function
    _block
return expression
```

The def keyword, along with the function name is used to define the function. The identifier rule must follow the function name.

A function accepts the parameter (argument), and they can be optional.

The function block is started with the colon (:), and block statements must be at the same indentation. The return statement is used to return the value. A function can have only one return Function Calling

In Python, after the function is created, we can call it from another function. A function must be defined before the function call; otherwise, the Python interpreter gives an error. To call the function, use the function name followed by the parentheses

Consider the following example of a simple example that prints the message "Hello World".

```
function
definition  def
hello_world():
    print("helloworld")
#          function      calling
hello_world()
```

Output:

hello world

The return statement

The return statement is used at the end of the function and returns the result of the function. It terminates the function execution and transfers the result where the function is called. The return statement cannot be used outside of the function.

Syntax

```
return [expression_list]
```

Arguments in function

The arguments are types of information which can be passed into the function. The arguments are specified in the parentheses. We can pass any number of arguments, but they must be separate them with a comma. #Python function to calculate the sum of two variables

#defining the function

```
def sum (a,b):
    return a+b;
```

```
#taking values from the user a = int(input("Enter a: "))
```

```
b = int(input("Enter b: "))
```

```
#printing the sum of a and b print("Sum = ",sum(a,b))
```

Program:

```
def binary_to_decimal(binary_str):
    decimal_num = 0
    binary_str = binary_str[::-1] # Reverse the binary string
    for i in range(len(binary_str)):
        if binary_str[i] == '1':
            decimal_num += 2**i
    return decimal_num

def octal_to_hexadecimal(octal_str):
    decimal_num = int(octal_str, 8) # Convert octal string to decimal
    hexadecimal_str = hex(decimal_num).upper()[2:] # Convert decimal to hexadecimal
    return hexadecimal_str

# Main program
while True:
    print("Choose an option:")
    print("1. Convert Binary to Decimal")
    print("2. Convert Octal to Hexadecimal")
    print("3. Exit")
    choice = input("Enter your choice (1/2/3): ")

    if choice == '1':
        binary_str = input("Enter a binary number: ")
        decimal_num = binary_to_decimal(binary_str)
        print(f"Decimal equivalent: {decimal_num}")
    elif choice == '2':
        octal_str = input("Enter an octal number: ")
        hexadecimal_str = octal_to_hexadecimal(octal_str)
        print(f"Hexadecimal equivalent: 0x{hexadecimal_str}")
    elif choice == '3':
        print("Exiting the program. Goodbye!")
        break
    else:
        print("Invalid choice. Please select a valid option.")
```

OUTPUT:

Choose an option:

1. Convert Binary to Decimal
2. Convert Octal to Hexadecimal
3. Exit

Enter your choice (1/2/3): 1

Enter a binary number: 1011

Decimal equivalent: 11

Choose an option:

1. Convert Binary to Decimal
2. Convert Octal to Hexadecimal
3. Exit

Enter your choice (1/2/3): 2

Enter an octal number: 345

Hexadecimal equivalent: 0xE5

Choose an option:

1. Convert Binary to Decimal
2. Convert Octal to Hexadecimal
3. Exit

Enter your choice (1/2/3): 3

Exiting the program. Goodbye!

3a. Write a Python program that accepts a sentence and find the number of words, digits, uppercase letters and lowercase letters.

A String in Python consists of a series or sequence of characters - letters, numbers, and special characters. Strings are marked by quotes:

- single quotes (' ') Eg, 'This a string in single quotes'
- double quotes (" ") Eg, "This a string in double quotes"
- triple quotes(""" """) Eg, This is a paragraph. It is made up of multiple lines and sentences. """ Individual character in a string is accessed using a subscript (index).

Program:

```
sentence = input("Enter a sentence : ")

wordList = sentence.split(" ")
print("This sentence has", len(wordList), "words")

digCnt = upCnt = loCnt = 0

for ch in sentence:
    if '0' <= ch <= '9':
        digCnt += 1
    elif 'A' <= ch <= 'Z':
        upCnt += 1
    elif 'a' <= ch <= 'z':
        loCnt += 1

print("This sentence has", digCnt, "digits", upCnt, "upper case letters", loCnt,
      "lower case letters")
```

OUTPUT:

```
Enter a sentence : cse department at dsatm
This sentence has 4 words
This sentence has 0 digits 0 upper case letters 20 lower case letters
Enter a sentence : Cse department at DSATM 1234
This sentence has 5 words
This sentence has 4 digits 6 upper case letters 14 lower case letters
```

3 b) Write a Python program to find the string similarity between two given strings .

```
str1 = input("Enter String 1 \n")
str2 = input("Enter String 2 \n")
if len(str2) < len(str1):
    short = len(str2)
    long = len(str1)
else:
    short = len(str1)
    long = len(str2)
matchCnt = 0
for i in range(short):
    if str1[i] == str2[i]:
        matchCnt += 1
print("Similarity between two said strings:")
print(matchCnt/long)
```

OUTPUT:

```
Enter String 1
dsatmcse
Enter String 2
dsatmcse
Similarity between two said strings:
1.0
Enter String 1
cse department at dsatm
Enter String 2
ise department at dsatm
Similarity between two said strings:
0.9565217391304348
```

4a. Write a Python program to Demonstrate how to Draw a Bar Plot using Matplotlib.

A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically. The **matplotlib** API in Python provides the **bar()** function which can be used in MATLAB style use or as an object-oriented API.

The syntax of the bar() function to be used with the axes is as follows:-

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

The function creates a bar plot bounded with a rectangle depending on the given parameters.

Program:

```
import matplotlib.pyplot as pyplot

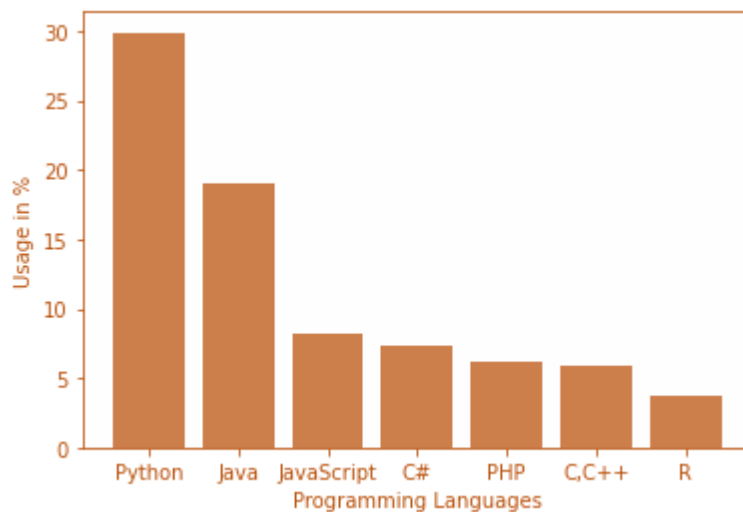
# Manual data setup
labels = ('Python', 'Java', 'JavaScript', 'C#', 'PHP', 'C,C++', 'R')
index = (1, 2, 3, 4, 5, 6, 7) # provides locations on x axis
sizes = [29.9, 19.1, 8.2, 7.3, 6.2, 5.9, 3.7]

# bar chart setup
pyplot.bar(index, sizes, color="#6c3376", tick_label=labels)

# layout configuration
pyplot.ylabel('Usage in %')
pyplot.xlabel('Programming Languages')

# Save the chart file
#pyplot.savefig('filename.png', dpi=300)

# Print the chart
pyplot.show()
```

OUTPUT:

4b. Write a Python program to Demonstrate how to Draw a Scatter Plot using Matplotlib.**matplotlib.pyplot.scatter()**

Scatter plots are used to observe relationship between variables and uses dots to represent the relationship between them. The **scatter()** method in the matplotlib library is used to draw a scatter plot. Scatter plots are widely used to represent relation among variables and how change in one affects the other.

The scatter() method takes in the following parameters:

- **x_axis_data**- An array containing x-axis data
- **y_axis_data**- An array containing y-axis data
- **s**- marker size (can be scalar or array of size equal to size of x or y)
- **c**- color of sequence of colors for markers
- **marker**- marker style
- **cmap**- cmap name
- **linewidths**- width of marker border
- **edgecolor**- marker border color
- **alpha**- blending value, between 0 (transparent) and 1 (opaque)

Program:

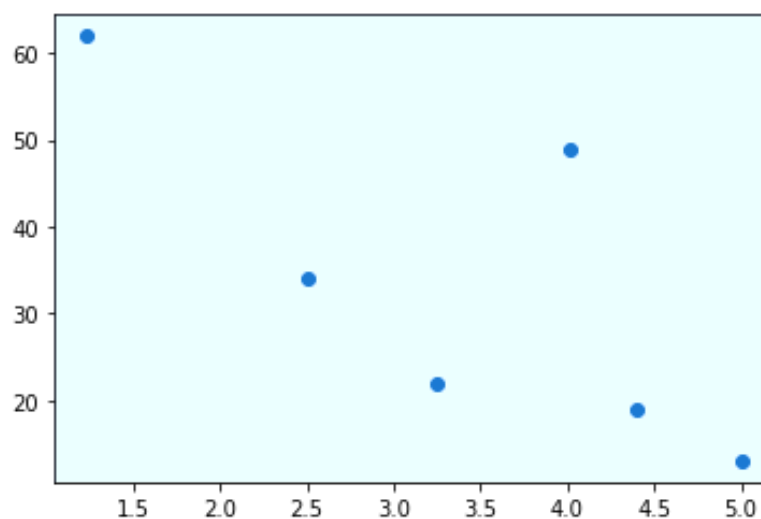
```
import matplotlib.pyplot as plt
```

```
price = [2.50, 1.23, 4.02, 3.25, 5.00, 4.40]
```

```
sales_per_day = [34, 62, 49, 22, 13, 19]
```

```
plt.scatter(price, sales_per_day)
```

```
plt.show()
```

OUTPUT:

5 a) Write a Python program to Demonstrate how to Draw a Histogram Plot using Matplotlib.

A histogram is a graph showing *frequency* distributions.
It is a graph showing the number of observations within each given interval.

In Matplotlib, we use the `hist()` function to create histograms.

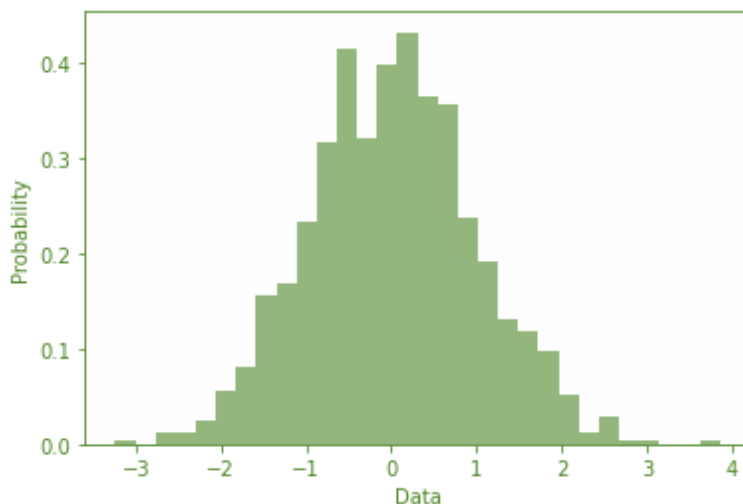
The `hist()` function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

PROGRAM:

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

np.random.seed(42)
x = np.random.normal(size=1000)

plt.hist(x, density=True, bins=30) # density=False would make counts
plt.ylabel('Probability')
plt.xlabel('Data');
```

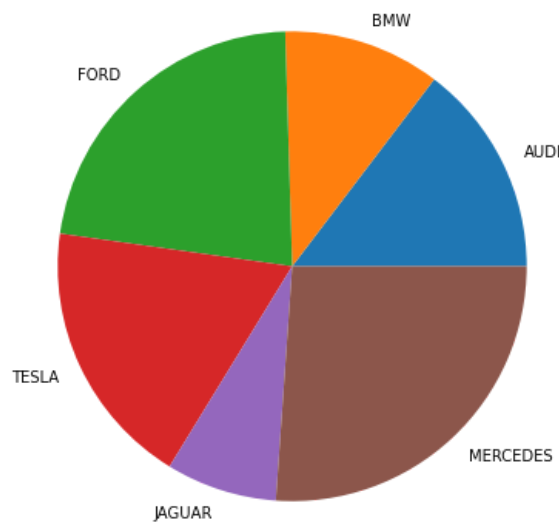
OUTPUT:

5 b) Write a Python program to Demonstrate how to draw a pie chart

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

# Creating dataset
cars = ['AUDI', 'BMW', 'FORD', 'TESLA', 'JAGUAR', 'MERCEDES']
data = [23, 17, 35, 29, 12, 41]
# Creating plot
fig = plt.figure(figsize=(10, 7))
plt.pie(data, labels = cars)
# show plot
plt.show()
# Draw a Pie Chart using Matplotlib
```

OUTPUT:



6. a) Write a Python program to illustrate Linear Plotting using Matplotlib.

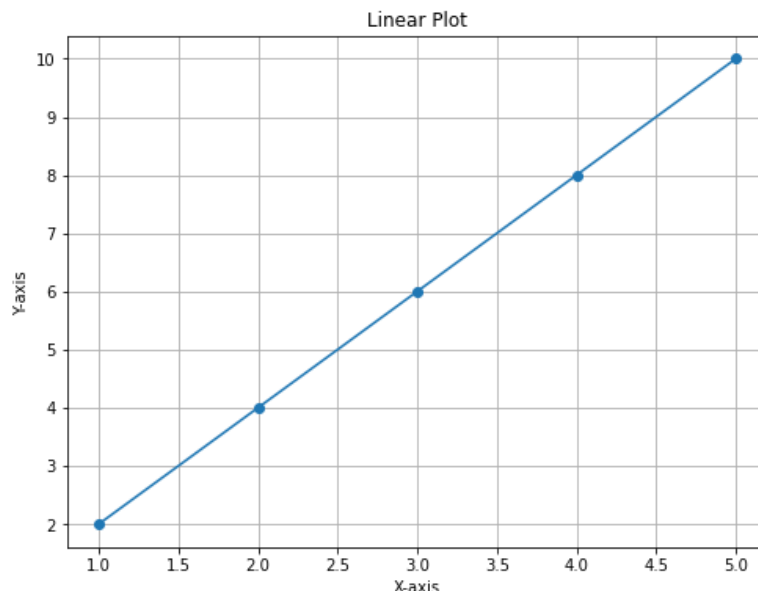
```
import matplotlib.pyplot as plt

# Step 2: Prepare your data
x = [1, 2, 3, 4, 5] # X-axis data
y = [2, 4, 6, 8, 10] # Y-axis data

# Step 3: Create the plot
plt.figure(figsize=(8, 6)) # Optional: Set the figure size
plt.plot(x, y, marker='o', linestyle='-') # Plotting with markers and lines

# Step 4: Customize the plot (optional)
plt.title("Linear Plot") # Set the title
plt.xlabel("X-axis") # Label for the X-axis
plt.ylabel("Y-axis") # Label for the Y-axis
plt.grid(True) # Add a grid

# Step 5: Display the plot
plt.show()
```

OUTPUT:

6 b) Write a Python program to illustrate liner plotting with line formatting using Matplotlib.

```
import matplotlib.pyplot as plt

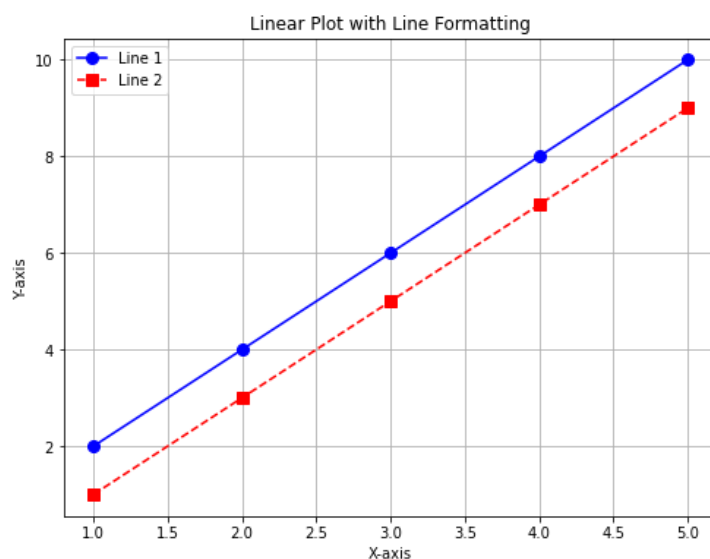
# Prepare your data
x = [1, 2, 3, 4, 5] # X-axis data
y1 = [2, 4, 6, 8, 10] # Y1-axis data
y2 = [1, 3, 5, 7, 9] # Y2-axis data

# Create the plot with line formatting
plt.figure(figsize=(8, 6)) # Optional: Set the figure size
# Plot Y1 with a blue solid line and circle markers
plt.plot(x, y1, color='blue', linestyle='-', marker='o', markersize=8, label='Line 1')

# Plot Y2 with a red dashed line and square markers
plt.plot(x, y2, color='red', linestyle='--', marker='s', markersize=8, label='Line 2')

# Customize the plot
plt.title("Linear Plot with Line Formatting")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.legend() # Display legend to distinguish between lines

# Display the plot
plt.show()
```

OUTPUT:

7. Write a Python program which explains uses of customizing seaborn plots with Aesthetic functions.

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

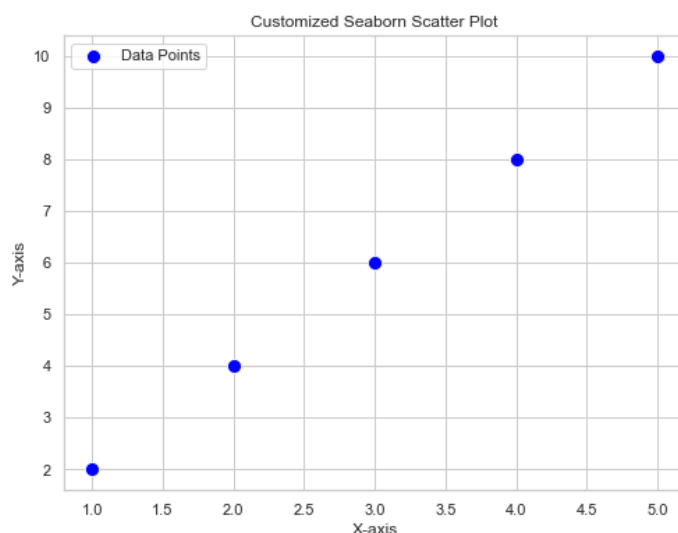
# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a scatter plot
sns.set(style="whitegrid") # Set the style
plt.figure(figsize=(8, 6)) # Set the figure size

# Customize the plot using aesthetic functions
sns.scatterplot(x=x, y=y, color='blue', marker='o', s=100, label='Data Points')

# Customize labels and title
plt.title("Customized Seaborn Scatter Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

# Show the plot
plt.legend()
plt.show()
```

OUTPUT:

8 a) Write a Python program to explain working with bokeh line graph using Annotations and Legends.

Bokeh is a Python interactive data visualization. It renders its plots using HTML and javascript. It targets modern web browsers for presentation providing elegant, concise construction of novel graphics with high-performance interactivity.

Bokeh can be used to **plot a line graph**. Plotting a line graph can be done using the `line()` method of the plotting module.

- ***x** : x-coordinates of the points to be plotted*
- ***y** : y-coordinates of the points to be plotted*
- ***line_alpha** : percentage value of line alpha, default is 1*
- ***line_cap** : value of line cap for the line, default is butt*
- ***line_color** : color of the line, default is black*
- ***line_dash** : value of line dash such as :*
 - *solid*
 - *dashed*
 - *dotted*
 - *dotdash*
 - *dashdot**default is solid*
- ***line_dash_offset** : value of line dash offset, default is 0*
- ***line_join** : value of line join, default in bevel*
- ***line_width** : value of the width of the line, default is 1*
- ***name** : user-supplied name for the model*
- ***tags** : user-supplied values for this model*

Other Parameters :

- ***alpha** : sets all alpha keyword arguments at once*
- ***color** : sets all color keyword arguments at once*
- ***legend_field** : name of a column in the data source that should be used*
- ***legend_group** : name of a column in the data source that should be used*
- ***legend_label** : labels the legend entry*
- ***muted** : determines whether the glyph should be rendered as muted or not, default is False*
- ***name** : optional user-supplied name to attach to the renderer*
- ***source** : user-supplied data source*
- ***view** : view for filtering the data source*
- ***visible** : determines whether the glyph should be rendered or not, default is True*
- ***x_range_name** : name of an extra range to use for mapping x-coordinates*
- ***y_range_name** : name of an extra range to use for mapping y-coordinates*
- ***level** : specifies the render level order for this glyph*

Returns : an object of class *GlyphRenderer*

```
from bokeh.plotting import figure, show
from bokeh.models import Label, Legend

# Sample data
x = [1, 2, 3, 4, 5]
y1 = [2, 4, 6, 8, 10]
y2 = [1, 3, 5, 7, 9]

# Create a Bokeh figure
p = figure(title="Bokeh Line Graph with Annotations and Legends", x_axis_label="X-axis",
y_axis_label="Y-axis")

# Add line glyphs for the data
line1 = p.line(x, y1, legend_label="Line 1", line_width=2, line_color="blue")
line2 = p.line(x, y2, legend_label="Line 2", line_width=2, line_color="red")

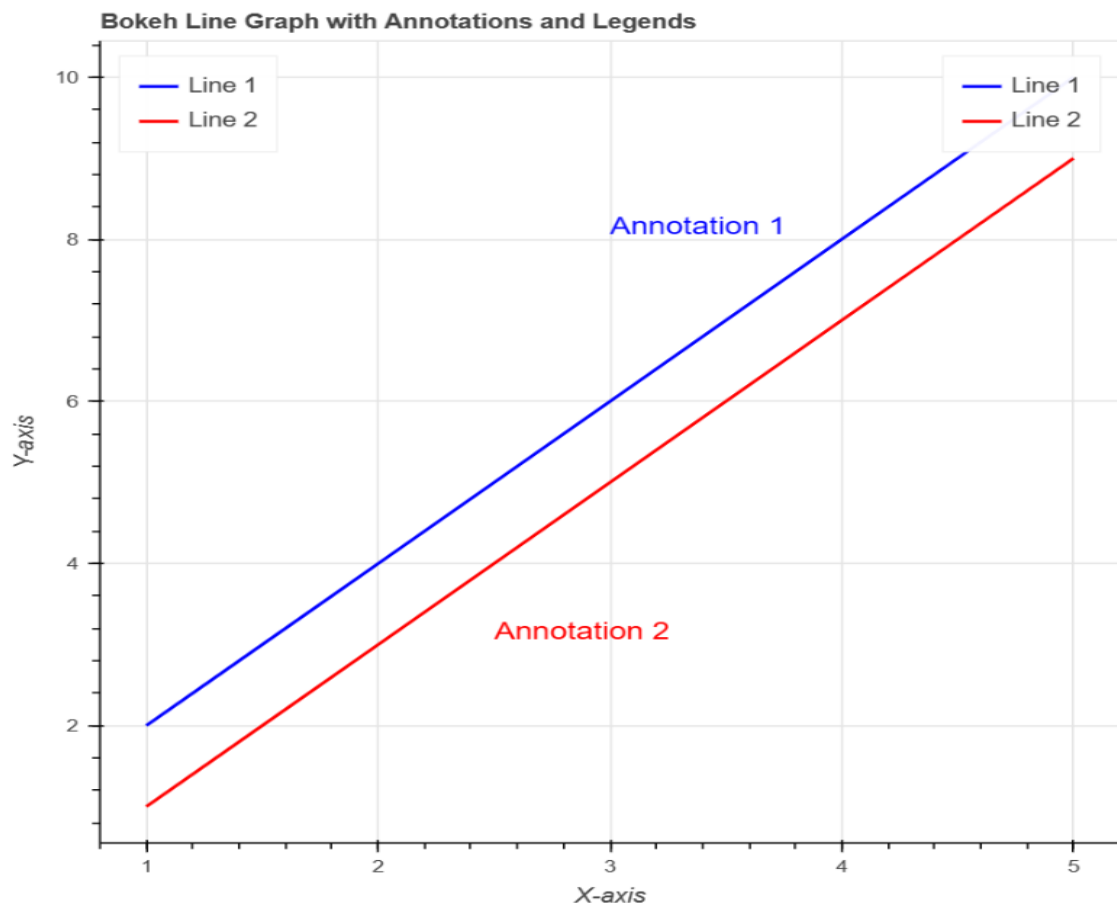
# Add annotations
label1 = Label(x=3, y=8, text="Annotation 1", text_font_size="12pt", text_color="blue")
label2 = Label(x=2.5, y=3, text="Annotation 2", text_font_size="12pt", text_color="red")

p.add_layout(label1)
p.add_layout(label2)

# Create a legend
legend = Legend(items=[("Line 1", [line1]), ("Line 2", [line2])], location="top_left")
p.add_layout(legend)

# Show the plot
show(p)
```

OUTPUT:



8. b) Write a Python program for plotting different types of plots using Bokeh.

Python Bokeh is a Data Visualization library that provides interactive charts and plots. Bokeh renders its plots using HTML and JavaScript that uses modern web browsers for presenting elegant, concise construction of novel graphics with high-level interactivity.

bokeh.plotting: This is the mid-level interface that provides **Matplotlib** or **MATLAB** like features for plotting. It deals with the data that is to be plotted and creating the valid axes, grids, and tools. The main class of this interface is the **Figure** class.

```
from bokeh.plotting import figure, show
from bokeh.layouts import gridplot
from bokeh.models import ColumnDataSource
import numpy as np
```

```
# Create some sample data
x = np.linspace(0, 4 * np.pi, 100)
y = np.sin(x)

# Line plot
p1 = figure(title="Line Plot", width=400, height=300)
p1.line(x, y, line_width=2, color="blue")

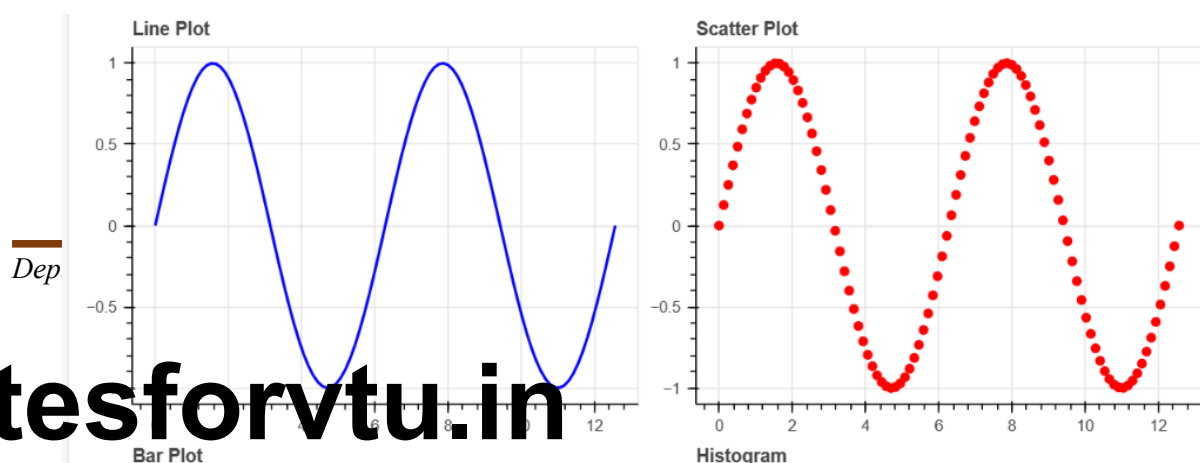
# Scatter plot
source = ColumnDataSource(data=dict(x=x, y=y))
p2 = figure(title="Scatter Plot", width=400, height=300)
p2.circle('x', 'y', source=source, size=6, color="red")

# Bar plot
categories = ['A', 'B', 'C', 'D']
values = [4, 7, 1, 2]
p3 = figure(x_range=categories, title="Bar Plot", width=400, height=300)
p3.vbar(x=categories, top=values, width=0.5, color="green")

# Histogram
data = np.random.normal(0, 1, 1000)
p4 = figure(title="Histogram", width=400, height=300)
hist, edges = np.histogram(data, bins=20)
p4.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:], fill_color="purple",
line_color="black")

# Create a grid of plots
plots = gridplot([[p1, p2], [p3, p4]])

# Show the plots
show(plots)
```

OUTPUT:

9. Write a Python program to draw 3D Plots using Plotly Libraries.

```
import plotly.graph_objects as go
import numpy as np

# Generate data for the 3D surface plot
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

# Create a 3D surface plot
fig = go.Figure(data=[go.Surface(z=Z, x=X, y=Y)])

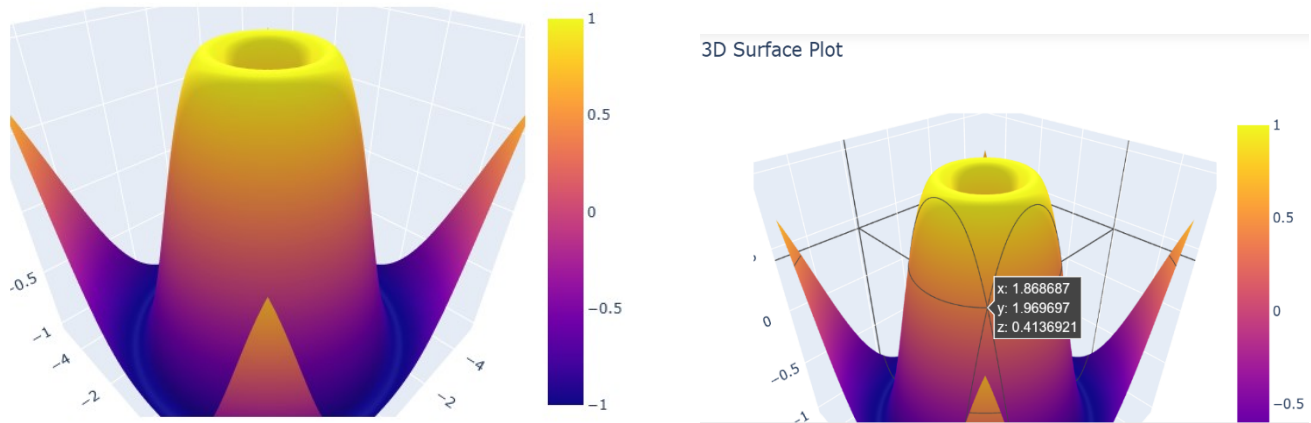
# Customize the layout
fig.update_layout(
    title="3D Surface Plot",
    scene=dict(
        xaxis_title="X-axis",
        yaxis_title="Y-axis",
        zaxis_title="Z-axis",
    )
)

# Show the plot
```

```
fig.show()
```

OUT PUT:

3D Surface Plot



10. a) Write a Python program to draw Time Series using Plotly Libraries

Time series can be represented using either `plotly.express` functions (`px.line`, `px.scatter`, `px.bar` etc) or `plotly.graph_objects` charts objects (`go.Scatter`, `go.Bar` etc). For more examples of such charts, see the documentation of [line and scatter plots](#) or [bar charts](#). Plotly auto-sets the axis type to a date format when the corresponding data are either ISO-formatted date strings or if they're a [date pandas column](#) or [datetime NumPy array](#).

```
import plotly.graph_objects as go
import pandas as pd

# Sample time series data (you can replace this with your own data)
data = {
    'Date': pd.date_range(start='2023-01-01', periods=10, freq='D'),
    'Value': [10, 12, 15, 20, 18, 22, 25, 28, 30, 32]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Create a time series line chart
fig = go.Figure()

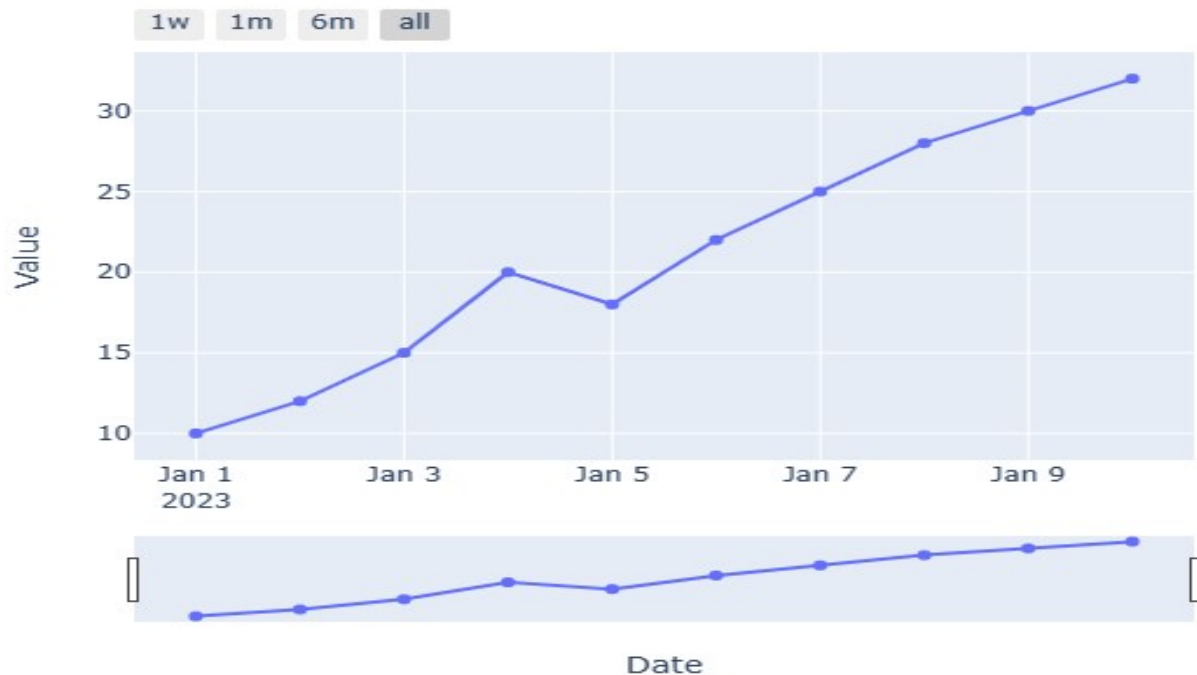
fig.add_trace(go.Scatter(x=df['Date'], y=df['Value'], mode='lines+markers', name='Time Series'))
```

```
# Customize the layout
fig.update_layout(
    title="Time Series Line Chart",
    xaxis_title="Date",
    yaxis_title="Value",
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=7, label="1w", step="day", stepmode="backward"),
                dict(count=1, label="1m", step="month", stepmode="backward"),
                dict(count=6, label="6m", step="month", stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(visible=True),
        type="date"
    )
)

# Show the plot
fig.show()
```

OUTPUT :

Time Series Line Chart



10 b . Write a Python program for creating Maps using Plotly Libraries.

Plotly is a Python library that is very popular among data scientists to create interactive data visualizations. One of the visualizations available in Plotly is Choropleth Maps. Choropleth maps are used to plot maps with shaded or patterned areas which are proportional to a statistical variable. They are composed of colored polygons. They are used for representing spatial variations of a quantity.

To create them, we require two main types of inputs –

- Geometric information –
 - this can be a GeoJSON file (here each feature has an *id* or some identifying value in properties, or
 - this can be built-in geometries of plotly – US states and world countries
- A list of values with feature identifier as index

```
import plotly.graph_objects as go
```

```
# Sample data for Indian states (replace with your own data)
```

```
data = {
```

```
    'State': ['Delhi', 'Maharashtra', 'Tamil Nadu', 'Karnataka', 'Kerala'],
```

```
    'Latitude': [28.6139, 19.7515, 11.1271, 15.3173, 10.8505],
```

```
    'Longitude': [77.2090, 75.7139, 78.6569, 75.7139, 76.2711],
```

```

'Value': [100, 200, 300, 400, 500] # Sample value associated with each state
}

# Create a map using Plotly
fig = go.Figure()

# Add scatter points for each state
for i in range(len(data['State'])):
    fig.add_trace(go.Scattergeo(
        lon=[data['Longitude'][i]],
        lat=[data['Latitude'][i]],
        text=data['State'][i] + '<br>Value: ' + str(data['Value'][i]),
        mode='markers',
        marker=dict(
            size=10,
            opacity=0.7,
            color=data['Value'][i],
            colorscale='Viridis',
            colorbar=dict(title='Value')
        ),
        name=data['State'][i]
    ))

# Set map layout
fig.update_geos(
    scope='asia',
    showcoastlines=True,
    coastlinecolor="Black",
    showland=True,
    landcolor="lightgray"
)

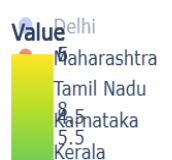
# Set the title
fig.update_layout(title='Indian States Map')

# Show the map
fig.show()

```

OUTPUT:

Indian States Map



QUESTIONS

- 1 What is data visualization, and why is it important in data analysis and interpretation?
- 2 Can you explain the key libraries in Python commonly used for data visualization?
- 3 How do you import and load data for visualization in Python, and which libraries or methods do you use for this purpose?
- 4 What are the differences between Matplotlib and Seaborn, and when would you choose one over the other for data visualization?
- 5 Explain the basic structure of a Matplotlib or Seaborn plot. What are the essential components?
- 6 How do you create different types of plots, such as line plots, scatter plots, bar charts, and histograms, in Matplotlib or Seaborn?
- 7 Discuss the concept of customizing and styling visualizations in Python.
- 8 What kind of customizations can you apply to plots?
- 9 What is the importance of choosing the right color palette for data visualizations?
- 10 How do you select an appropriate one in Seaborn?
- 11 How can you handle missing data or outliers when preparing data for visualization?
- 12 What is the role of data preprocessing in data visualization, and can you provide examples of common preprocessing tasks?
- 13 How can you create interactive visualizations in Python, and which libraries would you use for this purpose?
- 14 What is the role of data aggregation and summarization in creating meaningful visualizations?

- 15 Describe the process of creating subplots and multi-panel visualizations in Matplotlib or Seaborn.
- 16 How do you save visualizations as image files or interactive web applications in Python?
- 17 Can you explain the concept of geospatial data visualization in Python?
- 18 What libraries and tools are commonly used for this purpose?
- 19 What are the best practices for labeling and annotating visualizations to make them more informative and readable?
- 20 How can you create time series visualizations in Python, and what libraries are suitable for this task?
- 21 Describe the steps involved in creating 3D or volumetric visualizations in Python.
- 22 Discuss the advantages and limitations of using Python for data visualization compared to other programming languages or tools.
- 23 Provide examples of real-world projects or applications where Python data visualization played a crucial role