# Introduction to Matplotlib

## Matplotlib – Introduction

Matplotlib is one of the most popular Python packages used for data visualization.

It is a cross-platform library for making 2D plots from data in arrays. It is written in Python and makes use of NumPy, the numerical mathematics extension of Python.

It provides an object-oriented API that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPythonotTkinter.

It can be used in Python and IPython shells, Jupyter notebook and web application servers also.

## Matplotlib – Introduction

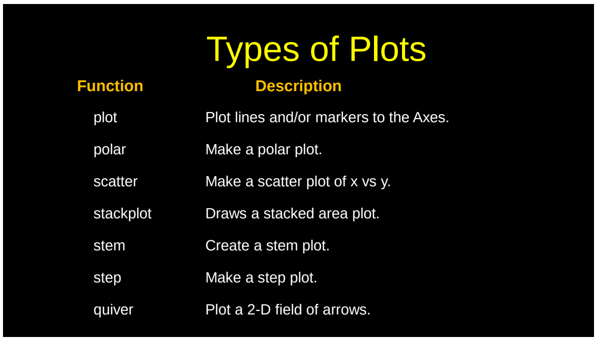
Matplotlib has a procedural interface named the Pylab, which is designed to resemble MATLAB, a proprietary programming language developed by MathWorks.

Matplotlib along with NumPy can be considered as the open source equivalent of MATLAB.

Matplotlib was originally written by John D. Hunter in 2003. The current stable version is 2.2.0 released in January 2018.

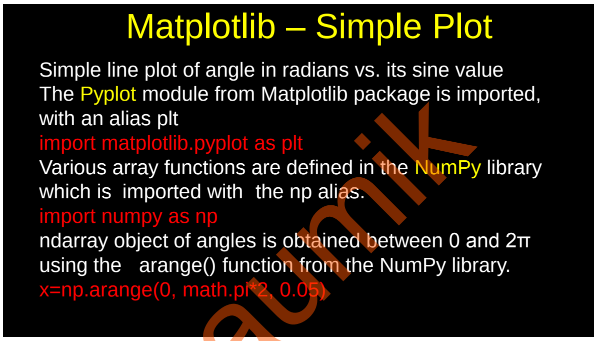
## Types of Plots

| Function | Description |
|---|---|
| bar | Make a bar plot. |
| barh | Make a horizontal bar plot. |
| boxplot | Make a box and whisker plot. |
| hist | Plot a histogram. |
| hist2d | Make a 2D histogram plot. |
| pie | Plot a pie chart. |



## Types of Plots

| Function | Description |
|---|---|
| plot | Plot lines and/or markers to the Axes. |
| polar | Make a polar plot. |
| scatter | Make a scatter plot of x vs y. |
| stackplot | Draws a stacked area plot. |
| stem | Create a stem plot. |
| step | Make a step plot. |
| quiver | Plot a 2-D field of arrows. |



## Matplotlib – Simple Plot

Simple line plot of angle in radians vs. its sine value
The Pyplot module from Matplotlib package is imported, with an alias plt

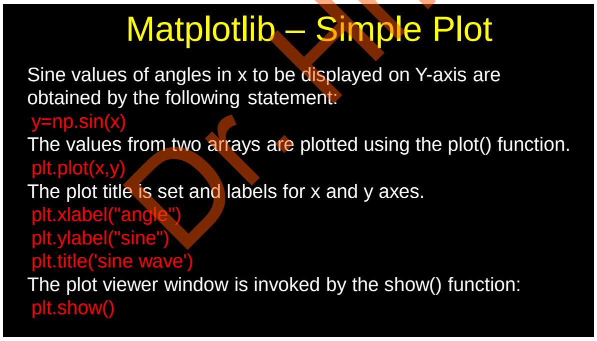
```
import matplotlib.pyplot as plt
```

Various array functions are defined in the NumPy library which is imported with the np alias.

```
import numpy as np
```

ndarray object of angles is obtained between 0 and 2π using the arange() function from the NumPy library.

```
x=np.arange(0, math.pi*2, 0.05)
```



## Matplotlib – Simple Plot

Sine values of angles in x to be displayed on Y-axis are obtained by the following statement:

```
y=np.sin(x)
```

The values from two arrays are plotted using the plot() function.

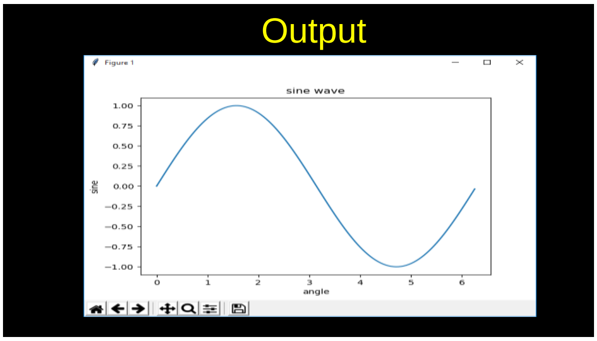
```
plt.plot(x,y)
```

The plot title is set and labels for x and y axes.

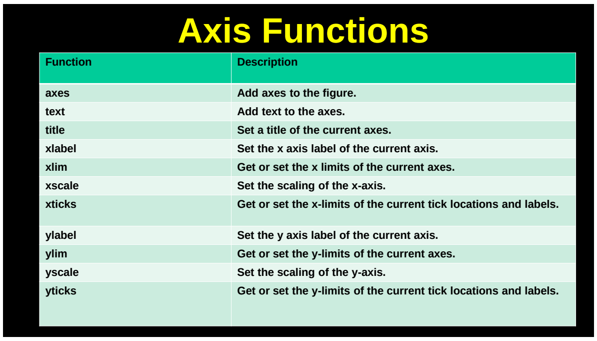
```
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
```

The plot viewer window is invoked by the show() function:

```
plt.show()
```

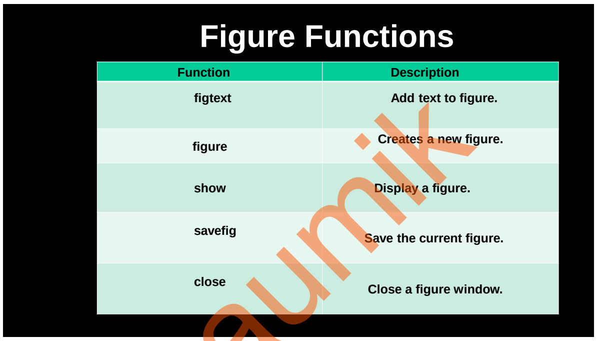


## Output



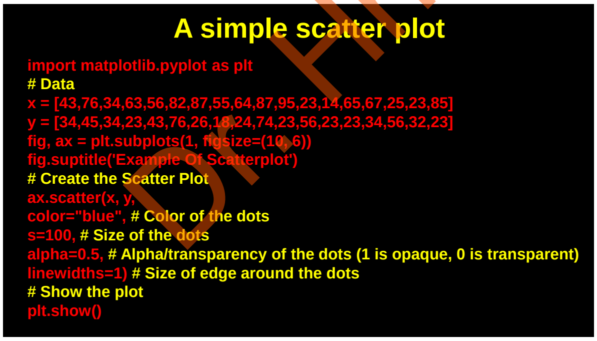
## Axis Functions

| Function | Description |
|---|---|
| axes | Add axes to the figure. |
| text | Add text to the axes. |
| title | Set a title of the current axes. |
| xlabel | Set the x axis label of the current axis. |
| xlim | Get or set the x limits of the current axes. |
| xscale | Set the scaling of the x-axis. |
| xticks | Get or set the x-limits of the current tick locations and labels. |
| ylabel | Set the y axis label of the current axis. |
| ylim | Get or set the y-limits of the current axes. |
| yscale | Set the scaling of the y-axis. |
| yticks | Get or set the y-limits of the current tick locations and labels. |



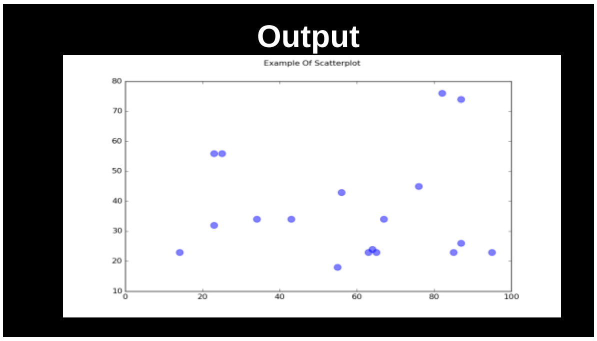
## Figure Functions

| Function | Description |
|---|---|
| figtext | Add text to figure. |
| figure | Creates a new figure. |
| show | Display a figure. |
| savefig | Save the current figure. |
| close | Close a figure window. |



## A simple scatter plot

```
import matplotlib.pyplot as plt
# Data
x = [43,76,34,63,56,82,87,55,64,87,95,23,14,65,67,25,23,85]
y = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]
fig, ax = plt.subplots(1, figsize=(10, 6))
fig.suptitle('Example Of Scatterplot')
# Create the Scatter Plot
ax.scatter(x, y,
color="blue", # Color of the dots
s=100, # Size of the dots
alpha=0.5, # Alpha/transparency of the dots (1 is opaque, 0 is transparent)
linewidths=1) # Size of edge around the dots
# Show the plot
plt.show()
```



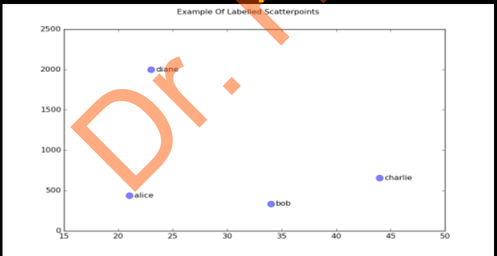
## Output

## A Scatterplot with Labelled Points

```
import matplotlib.pyplot as plt
# Data
x = [21, 34, 44, 23]
y = [435, 334, 656, 1999]
labels = ["alice", "bob", "charlie", "diane"]
# Create the figure and axes objects
fig, ax = plt.subplots(1, figsize=(10, 6))
fig.suptitle('Example Of Labelled Scatterpoints')
# Plot the scatter points
ax.scatter(x, y,
color="blue", # Color of the dots
s=100, # Size of the dots
alpha=0.5, # Alpha/transparency of the dots
linewidths=1) # Size of edge around the dots
```

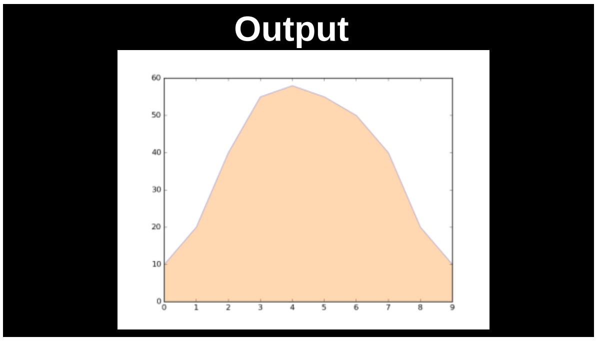
```
# Add the participant names as text labels for each point
for x_pos, y_pos, label in zip(x, y, labels):
    ax.annotate(label, # The label for this point
    xy=(x_pos, y_pos), # Position of the corresponding point
    xytext=(7, 0), # Offset text by 7 points to the right
    textcoords='offset points', # tell it to use offset points
    ha='left', # Horizontally aligned to the left
    va='center') # Vertical alignment is centered
# Show the plot
plt.show()
```

## Output

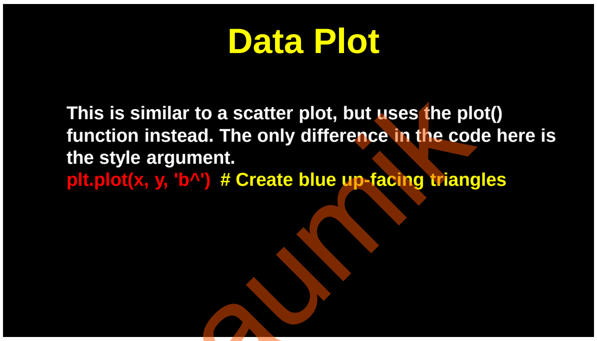


## Shaded region below a line

```
import matplotlib.pyplot as plt
# Data
x = [0,1,2,3,4,5,6,7,8,9]
y1 = [10,20,40,55,58,55,50,40,20,10]
# Shade the area between y1 and line y=0
plt.fill_between(x, y1, 0,
facecolor="orange", # The fill color
color='blue', # The outline color
alpha=0.2) # Transparency of the fill
# Show the plot
plt.show()
```



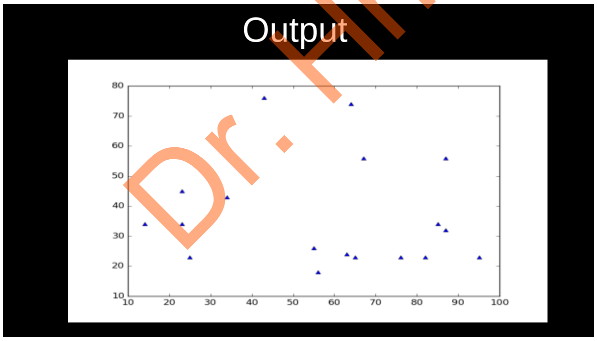
## Output



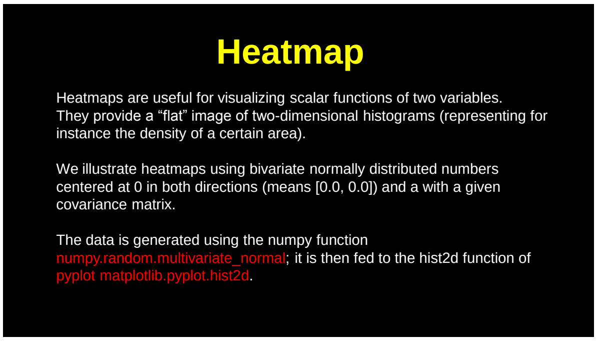
## Data Plot

**This is similar to a scatter plot, but uses the plot() function instead. The only difference in the code here is the style argument.**

**plt.plot(x, y, 'b^')  # Create blue up-facing triangles**



## Output



## Heatmap

Heatmaps are useful for visualizing scalar functions of two variables. They provide a “flat” image of two-dimensional histograms (representing for instance the density of a certain area).

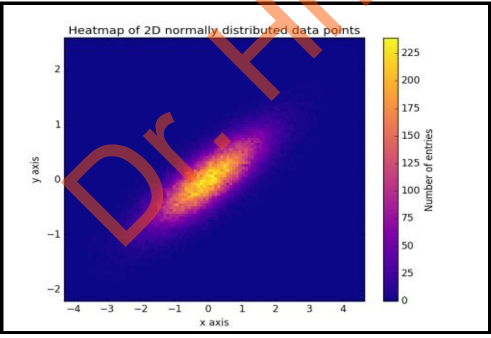
We illustrate heatmaps using bivariate normally distributed numbers centered at 0 in both directions (means [0.0, 0.0]) and a with a given covariance matrix.

The data is generated using the numpy function numpy.random.multivariate_normal; it is then fed to the hist2d function of pyplot matplotlib.pyplot.hist2d.

## Source Code

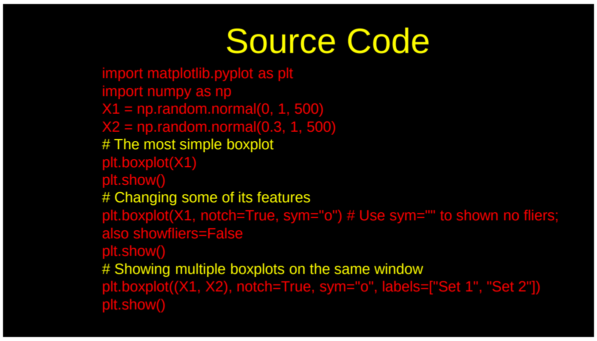
```
import matplotlib
import matplotlib.pyplot as plt
# Define numbers of generated data points and bins per axis.
N_numbers = 100000
N_bins = 100
# set random seed
np.random.seed(0)
# Generate 2D normally distributed numbers.
x, y = np.random.multivariate_normal(
mean=[0.0, 0.0], # mean
cov=[[1.0, 0.4],
[0.4, 0.25]], # covariance matrix
size=N_numbers).T # transpose to get columns
```

```
# Construct 2D histogram from data using the 'plasma' colormap
plt.hist2d(x, y, bins=N_bins, normed=False, cmap='plasma')
# Plot a colorbar with label.
cb = plt.colorbar()
cb.set_label('Number of entries')
# Add title and labels to plot.
plt.title('Heatmap of 2D normally distributed data points')
plt.xlabel('x axis')
plt.ylabel('y axis')
# Show the plot.
plt.show()
```



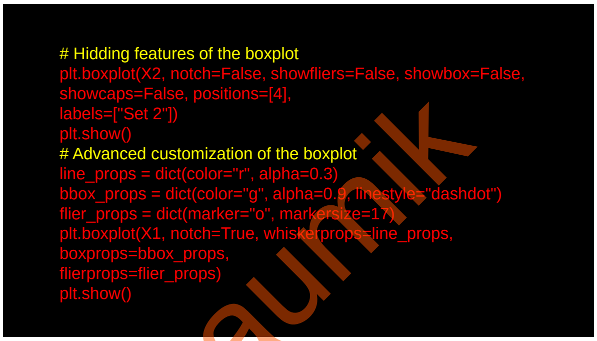
## Boxplots

Boxplots are descriptive diagrams that help to compare the distribution of different series of data. They are *descriptive* because they show measures (e.g. the *median*) which do not assume an underlying probability distribution.

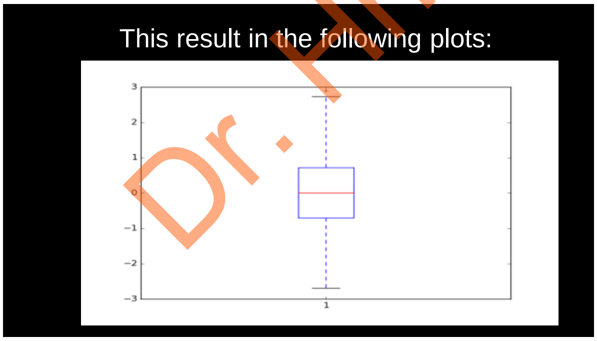


## Source Code

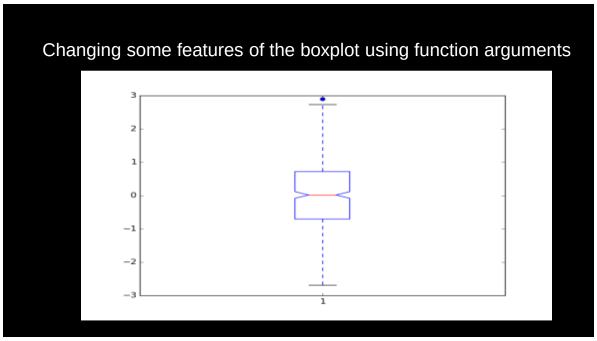
```
import matplotlib.pyplot as plt
import numpy as np
X1 = np.random.normal(0, 1, 500)
X2 = np.random.normal(0.3, 1, 500)
# The most simple boxplot
plt.boxplot(X1)
plt.show()
# Changing some of its features
plt.boxplot(X1, notch=True, sym="o") # Use sym="" to shown no fliers;
also showfliers=False
plt.show()
# Showing multiple boxplots on the same window
plt.boxplot((X1, X2), notch=True, sym="o", labels=["Set 1", "Set 2"])
plt.show()
```



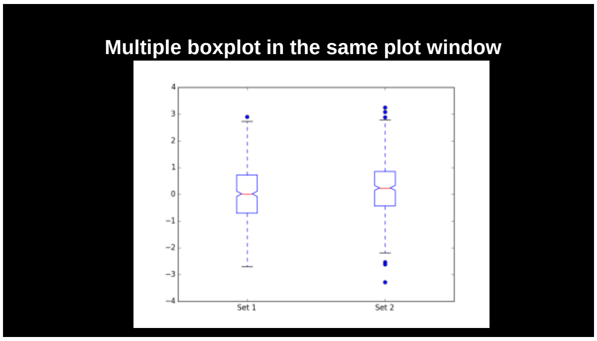
```
# Hidding features of the boxplot
plt.boxplot(X2, notch=False, showfliers=False, showbox=False,
showcaps=False, positions=[4],
labels=["Set 2"])
plt.show()
# Advanced customization of the boxplot
line_props = dict(color="r", alpha=0.3)
bbox_props = dict(color="g", alpha=0.9, linestyle="dashdot")
flier_props = dict(marker="o", markersize=17)
plt.boxplot(X1, notch=True, whiskerprops=line_props,
boxprops=bbox_props,
flierprops=flier_props)
plt.show()
```



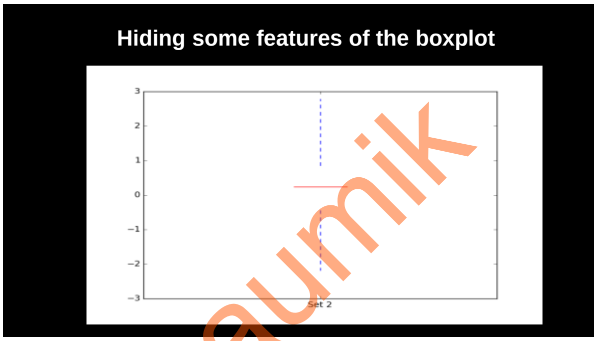
This result in the following plots:



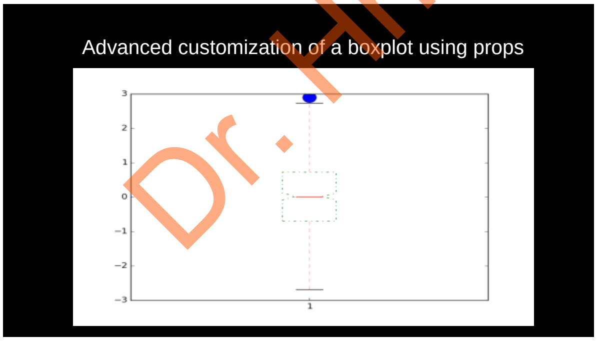
Changing some features of the boxplot using function arguments



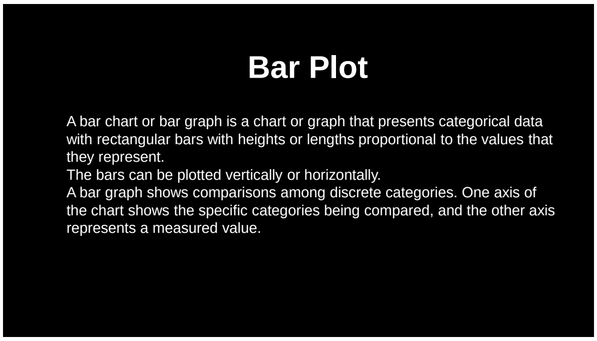
### Multiple boxplot in the same plot window



### Hiding some features of the boxplot



### Advanced customization of a boxplot using props

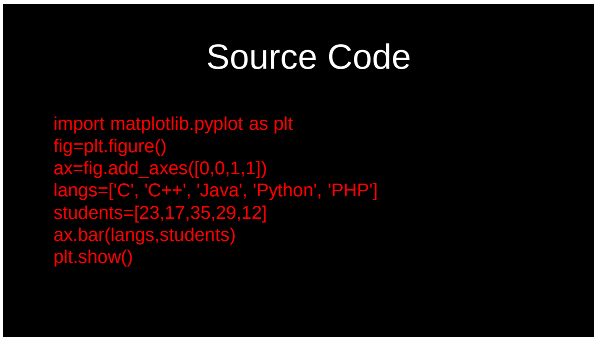


## Bar Plot

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent.

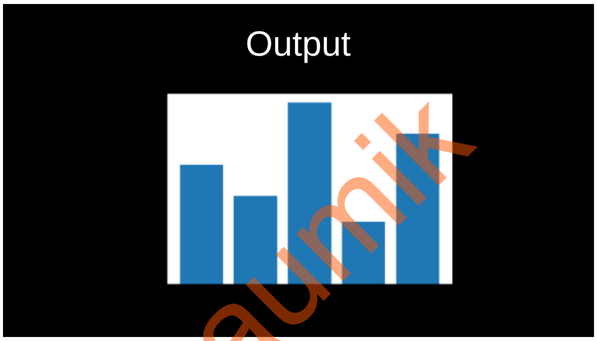
The bars can be plotted vertically or horizontally.

A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value.

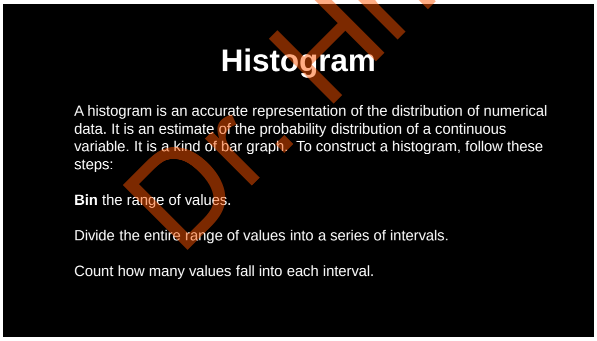


## Source Code

```
import matplotlib.pyplot as plt
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
langs=['C', 'C++', 'Java', 'Python', 'PHP']
students=[23,17,35,29,12]
ax.bar(langs,students)
plt.show()
```



## Output



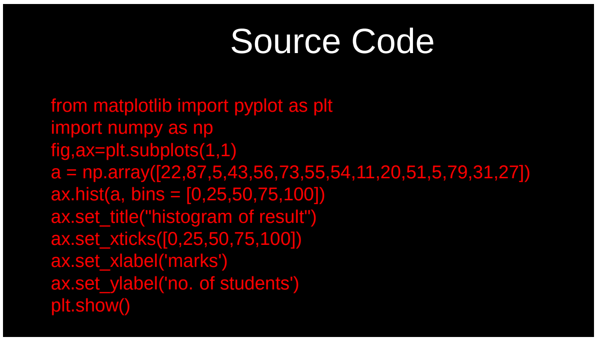
## Histogram

A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable. It is a kind of bar graph. To construct a histogram, follow these steps:

**Bin** the range of values.

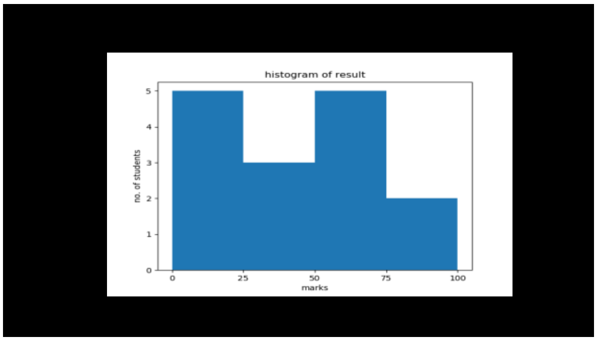
Divide the entire range of values into a series of intervals.

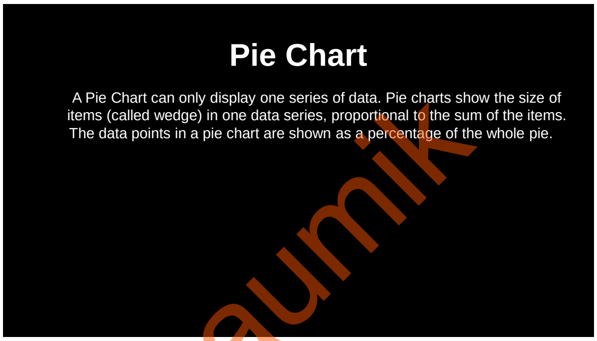
Count how many values fall into each interval.



## Source Code

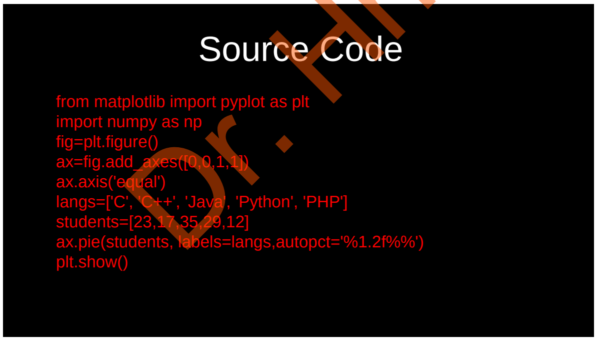
```
from matplotlib import pyplot as plt
import numpy as np
fig,ax=plt.subplots(1,1)
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100])
ax.set_title("histogram of result")
ax.set_xticks([0,25,50,75,100])
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')
plt.show()
```





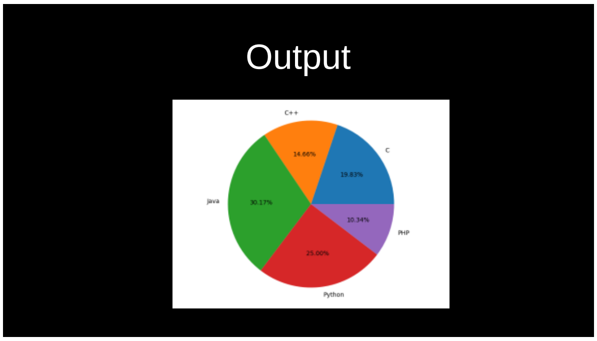
## Pie Chart

A Pie Chart can only display one series of data. Pie charts show the size of items (called wedge) in one data series, proportional to the sum of the items. The data points in a pie chart are shown as a percentage of the whole pie.



## Source Code

```
from matplotlib import pyplot as plt
import numpy as np
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.axis('equal')
langs=['C', 'C++', 'Java', 'Python', 'PHP']
students=[23,17,35,29,12]
ax.pie(students, labels=langs,autopct='%1.2f%%')
plt.show()
```



## Output

# Quiver Plot

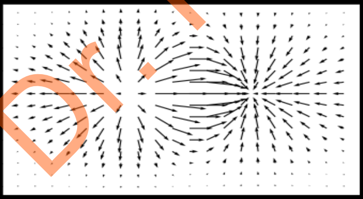
A quiver plot displays the velocity vectors as arrows with components (u,v) at the points (x,y).

quiver(x,y,u,v)

The above function plots vectors as arrows at the coordinates specified in each corresponding pair of elements in x and y.

# Source Code

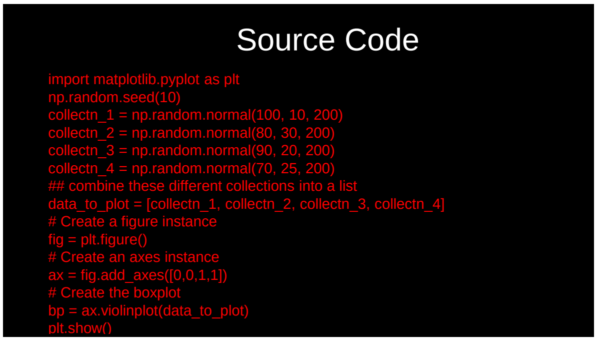
```
import matplotlib.pyplot as plt
import numpy as np
x,y = np.meshgrid(np.arange(-2, 2, .2), np.arange(-2, 2, .25))
z = x*np.exp(-x**2 - y**2)
v, u = np.gradient(z, .2, .2)
fig, ax = plt.subplots()
q = ax.quiver(x,y,u,v)
plt.show()
```



# Violin Plot

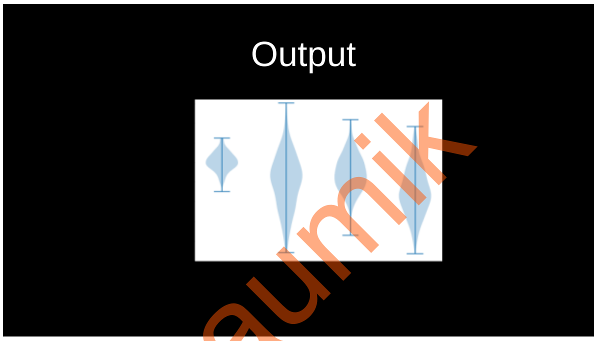
Violin plots are similar to box plots, except that they also show the probability density of the data at different values. These plots include a marker for the median of the data and a box indicating the interquartile range, as in the standard box plots. Overlaid on this box plot is a kernel density estimation. Like box plots, violin plots are used to represent comparison of a variable distribution (or sample distribution) across different "categories".

A violin plot is more informative than a plain box plot. In fact while a box plot only shows summary statistics such as mean/median and interquartile ranges, the violin plot shows the full distribution of the data.



## Source Code

```
import matplotlib.pyplot as plt
np.random.seed(10)
collectn_1 = np.random.normal(100, 10, 200)
collectn_2 = np.random.normal(80, 30, 200)
collectn_3 = np.random.normal(90, 20, 200)
collectn_4 = np.random.normal(70, 25, 200)
## combine these different collections into a list
data_to_plot = [collectn_1, collectn_2, collectn_3, collectn_4]
# Create a figure instance
fig = plt.figure()
# Create an axes instance
ax = fig.add_axes([0,0,1,1])
# Create the boxplot
bp = ax.violinplot(data_to_plot)
plt.show()
```



## Output