In [1]:

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

pip install matplotlib

1. Line Plot
2. Scatter Plot
3. Bar Graph
4. Histogram
5. Pie Chart
6. BoxPlot
7. Stem Plot

In [2]:

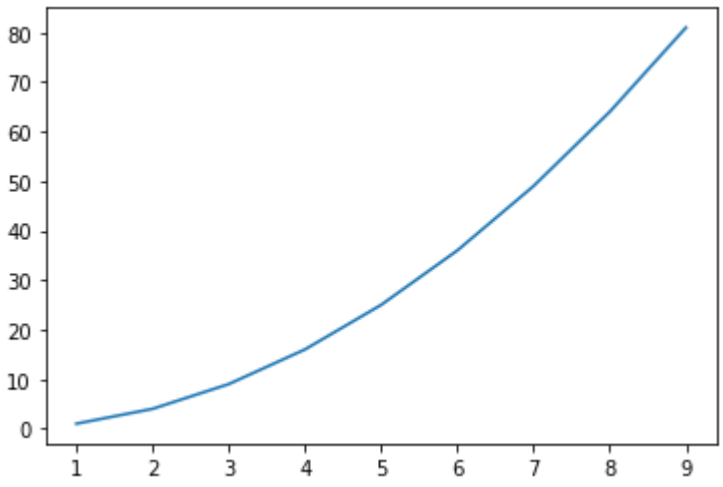
```
import numpy as np
```

In [5]:

```
x = np.arange(1, 10)
y = x ** 2


plt.plot(x, y)

plt.show()
```



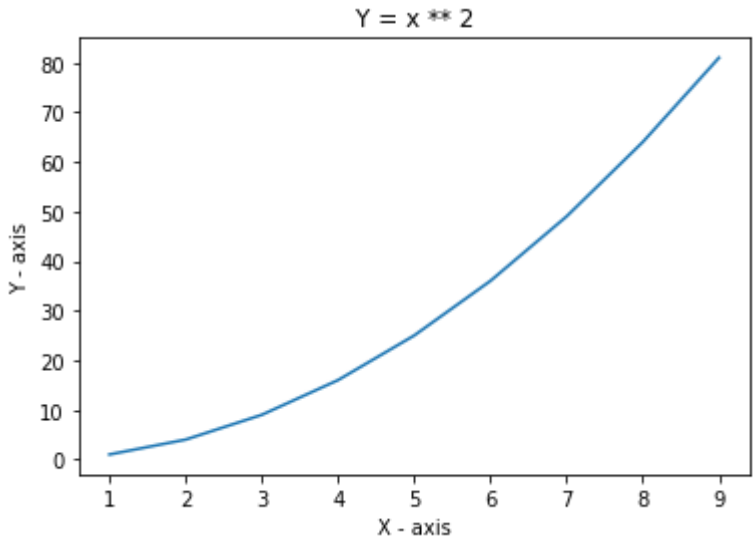
In [4]:

```
print(x, y)
```

```
[1 2 3 4 5 6 7 8 9] [ 1  4  9 16 25 36 49 64 81]
```

In [6]:

```
plt.plot(x, y)
plt.xlabel("X - axis")
plt.ylabel('Y - axis')
plt.title('Y = x ** 2')
plt.show()
```



In [7]:

```
help(plt.plot)
```

```
Help on function plot in module matplotlib.pyplot:

plot(*args, scalex=True, scaley=True, data=None, **kwargs)
    Plot y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], *, data=None, **kwargs)
        plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, *y*.

    The optional parameter *fmt* is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string
    notation described in the *Notes* section below.

    >>> plot(x, y)        # plot x and y using default line style and color
    >>> plot(x, y, 'bo')  # plot x and y using blue circle markers
    >>> plot(y)           # plot y using x as index array 0..N-1
    >>> plot(y, 'r+')     # ditto, but with red plusses

    You can use `.Line2D` properties as keyword arguments for more
    control on the appearance. Line properties and *fmt* can be mixed.
    The following two calls yield identical results:

    >>> plot(x, y, 'go--', linewidth=2, markersize=12)
    >>> plot(x, y, color='green', marker='o', linestyle='dashed',
    ...      linewidth=2, markersize=12)

    When conflicting with *fmt*, keyword arguments take precedence.


    **Plotting labelled data**

    There's a convenient way for plotting objects with labelled data (i.e.
    data that can be accessed by index ``obj['y']``). Instead of giving
    the data in *x* and *y*, you can provide the object in the *data*
    parameter and just give the labels for *x* and *y*::

    >>> plot('xlabel', 'ylabel', data=obj)

    All indexable objects are supported. This could e.g. be a `dict`, a
    `pandas.DataFame` or a structured numpy array.


    **Plotting multiple sets of data**

    There are various ways to plot multiple sets of data.

    - The most straight forward way is just to call `plot` multiple times.
      Example:

      >>> plot(x1, y1, 'bo')
      >>> plot(x2, y2, 'go')

    - Alternatively, if your data is already a 2d array, you can pass it
```

```
  directly to *x*, *y*. A separate data set will be drawn for every
  column.

  Example: an array ``a`` where the first column represents the *x*
  values and the other columns are the *y* columns::

  >>> plot(a[0], a[1:])

- The third way is to specify multiple sets of *[x]*, *y*, *[fmt]*
  groups::

  >>> plot(x1, y1, 'g^', x2, y2, 'g-')

  In this case, any additional keyword argument applies to all
  datasets. Also this syntax cannot be combined with the *data*
  parameter.

By default, each line is assigned a different style specified by a
'style cycle'. The *fmt* and line property parameters are only
necessary if you want explicit deviations from these defaults.
Alternatively, you can also change the style cycle using
:rc:`axes.prop_cycle`.


Parameters
----------
x, y : array-like or scalar
    The horizontal / vertical coordinates of the data points.
    *x* values are optional and default to `range(len(y))`.

    Commonly, these parameters are 1D arrays.

    They can also be scalars, or two-dimensional (in that case, the
    columns represent separate data sets).

    These arguments cannot be passed as keywords.

fmt : str, optional
    A format string, e.g. 'ro' for red circles. See the *Notes*
    section for a full description of the format strings.

    Format strings are just an abbreviation for quickly setting
    basic line properties. All of these and more can also be
    controlled by keyword arguments.

    This argument cannot be passed as keyword.

data : indexable object, optional
    An object with labelled data. If given, provide the label names to
    plot in *x* and *y*.

    .. note::
        Technically there's a slight ambiguity in calls where the
        second label is a valid *fmt*. `plot('n', 'o', data=obj)`
        could be `plt(x, y)` or `plt(y, fmt)`. In such cases,
        the former interpretation is chosen, but a warning is issued.
        You may suppress the warning by adding an empty format string
        `plot('n', 'o', '', data=obj)`.

Other Parameters
----------------
```

```
    scalex, scaley : bool, optional, default: True
        These parameters determined if the view limits are adapted to
        the data limits. The values are passed on to `autoscale_view`.

    **kwargs : `.Line2D` properties, optional
        *kwargs* are used to specify properties like a line label (for
        auto legends), linewidth, antialiasing, marker face color.
        Example::

        >>> plot([1, 2, 3], [1, 2, 3], 'go-', label='line 1', linewidth=2)
        >>> plot([1, 2, 3], [1, 4, 9], 'rs', label='line 2')

        If you make multiple lines with one plot command, the kwargs
        apply to all those lines.

        Here is a list of available `.Line2D` properties:

        Properties:
        agg_filter: a filter function, which takes a (m, n, 3) float array a
nd a dpi value, and returns a (m, n, 3) array
        alpha: float or None
        animated: bool
        antialiased or aa: bool
        clip_box: `.Bbox`
        clip_on: bool
        clip_path: Patch or (Path, Transform) or None
        color or c: color
        contains: callable
        dash_capstyle: {'butt', 'round', 'projecting'}
        dash_joinstyle: {'miter', 'round', 'bevel'}
        dashes: sequence of floats (on/off ink in points) or (None, None)
        data: (2, N) array or two 1D arrays
        drawstyle or ds: {'default', 'steps', 'steps-pre', 'steps-mid', 'ste
ps-post'}, default: 'default'
        figure: `.Figure`
        fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'}
        gid: str
        in_layout: bool
        label: object
        linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq),
...}
        linewidth or lw: float
        marker: marker style
        markeredgecolor or mec: color
        markeredgewidth or mew: float
        markerfacecolor or mfc: color
        markerfacecoloralt or mfcalt: color
        markersize or ms: float
        markevery: None or int or (int, int) or slice or List[int] or float
or (float, float)
        path_effects: `.AbstractPathEffect`
        picker: float or callable[[Artist, Event], Tuple[bool, dict]]
        pickradius: float
        rasterized: bool or None
        sketch_params: (scale: float, length: float, randomness: float)
        snap: bool or None
        solid_capstyle: {'butt', 'round', 'projecting'}
        solid_joinstyle: {'miter', 'round', 'bevel'}
        transform: `matplotlib.transforms.Transform`
        url: str
        visible: bool
```

```
    xdata: 1D array
    ydata: 1D array
    zorder: float

Returns
-------
lines
    A list of `.Line2D` objects representing the plotted data.

See Also
--------
scatter : XY scatter plot with markers of varying size and/or color (
    sometimes also called bubble chart).

Notes
-----
**Format Strings**

A format string consists of a part for color, marker and line::

    fmt = '[marker][line][color]'

Each of them is optional. If not provided, the value from the style
cycle is used. Exception: If ``line`` is given, but no ``marker``,
the data will be a line without markers.

Other combinations such as ``[color][marker][line]`` are also
supported, but note that their parsing may be ambiguous.

**Markers**

=============    ===============================
character        description
=============    ===============================
``'.'``          point marker
``','``          pixel marker
``'o'``          circle marker
``'v'``          triangle_down marker
``'^'``          triangle_up marker
``'<'``          triangle_left marker
``'>'``          triangle_right marker
``'1'``          tri_down marker
``'2'``          tri_up marker
``'3'``          tri_left marker
``'4'``          tri_right marker
``'s'``          square marker
``'p'``          pentagon marker
``'*'``          star marker
``'h'``          hexagon1 marker
``'H'``          hexagon2 marker
``'+'``          plus marker
``'x'``          x marker
``'D'``          diamond marker
``'d'``          thin_diamond marker
``'|'``          vline marker
``'_'``          hline marker
=============    ===============================

**Line Styles**

=============    ===============================
```

character	description
``'-'``	solid line style
``'--'``	dashed line style
``'-.'``	dash-dot line style
``':'``	dotted line style

Example format strings::

```
    'b'    # blue markers with default shape
    'or'   # red circles
    '-g'   # green solid line
    '--'   # dashed line with default color
    '^k:'  # black triangle_up markers connected by a dotted line
```

Colors

The supported color abbreviations are the single letter codes

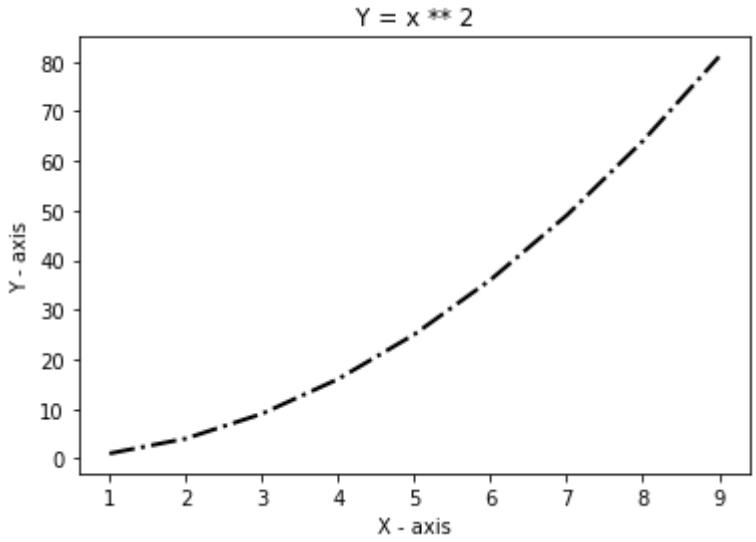
character	color
``'b'``	blue
``'g'``	green
``'r'``	red
``'c'``	cyan
``'m'``	magenta
``'y'``	yellow
``'k'``	black
``'w'``	white

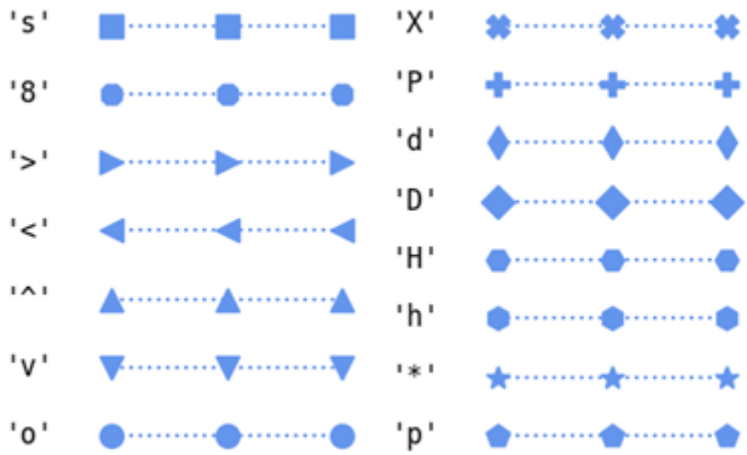
and the ``'CN'`` colors that index into the default property cycle.

If the color is the only part of the format string, you can
additionally use any `matplotlib.colors` spec, e.g. full names
(``'green'``) or hex strings (``'#008000'``).

In [14]:

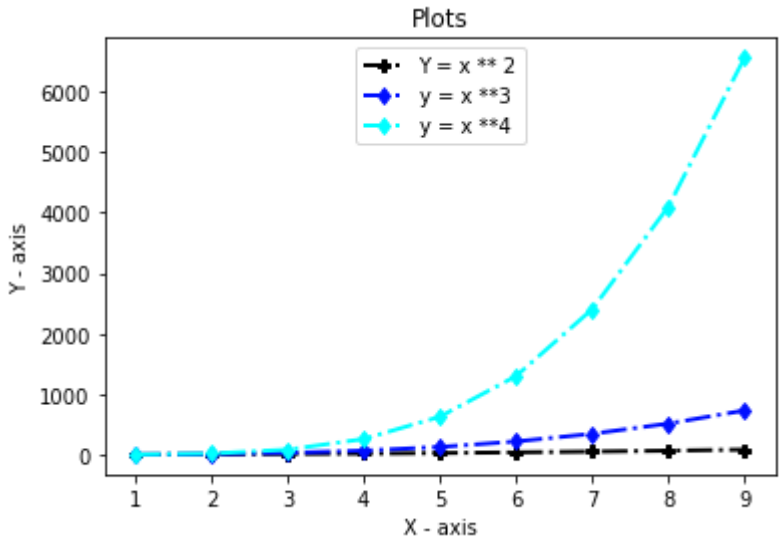
```
plt.plot(x, y, c ='#000000', linestyle = '-.', linewidth = 2)
plt.xlabel("X - axis")
plt.ylabel('Y - axis')
plt.title('Y = x ** 2')
plt.show()
```





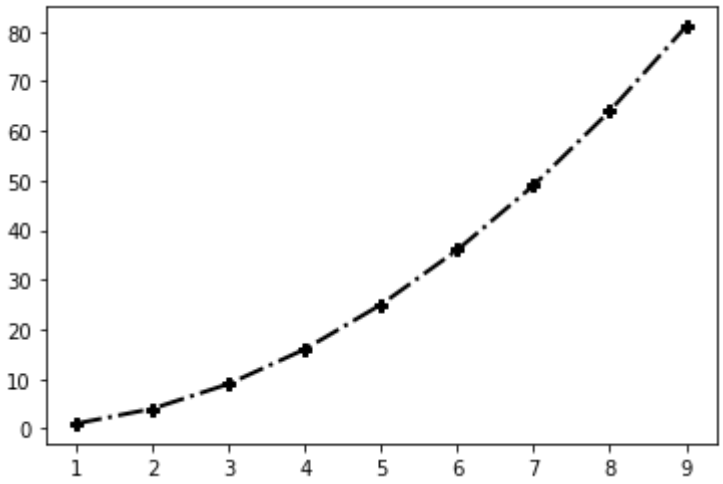
In [24]:

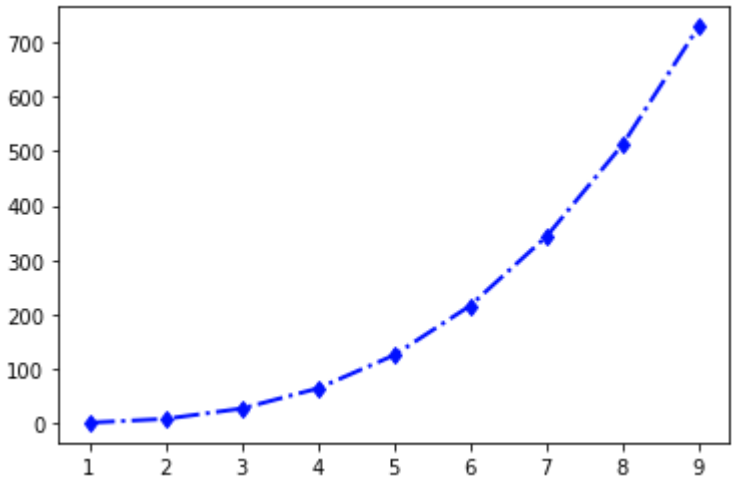
```
plt.plot(x, y, c ='#000000', linestyle = '-.', linewidth = 2, marker = 'P', label = 'Y = x
plt.plot(x, x ** 3, c ='#000fff', linestyle = '-.', linewidth = 2, marker = 'd', label = 'y
plt.plot(x, x ** 4, c ='#00ffff', linestyle = '-.', linewidth = 2, marker = 'd', label = 'y
plt.xlabel("X - axis")
plt.ylabel('Y - axis')
plt.title('Plots')
plt.legend(loc = 'upper center')
plt.show()
```

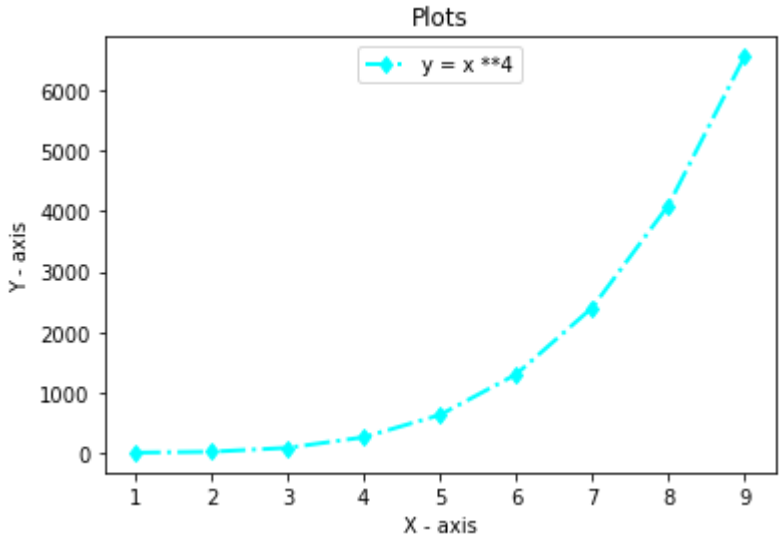


In [25]:

```
plt.plot(x, y, c ='#000000', linestyle = '-.', linewidth = 2, marker = 'P', label = 'Y = x
plt.show()
plt.plot(x, x ** 3, c ='#000fff', linestyle = '-.', linewidth = 2, marker = 'd', label = 'y
plt.show()
plt.plot(x, x ** 4, c ='#00ffff', linestyle = '-.', linewidth = 2, marker = 'd', label = 'y
plt.xlabel("X - axis")
plt.ylabel('Y - axis')
plt.title('Plots')
plt.legend(loc = 'upper center')
plt.show()
```







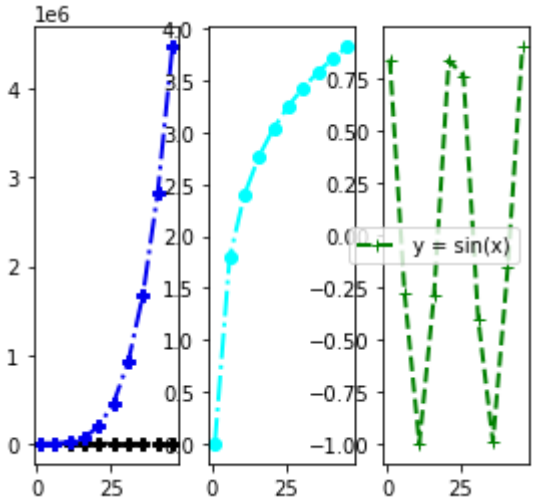
subplots

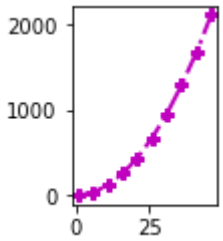
plt.subplot(nrows, ncols, index)

In [41]:

```
plt.subplot(1, 4, 1) # rrggbb
plt.plot(x, y, c ='#000000', linestyle = '-.', linewidth = 2, marker = 'P', label = 'Y = x
plt.plot(x, x **4, c ='#0000f0', linestyle = '-.', linewidth = 2, marker = 'P', label = 'Y
plt.subplot(1, 4, 2)
plt.plot(x, np.log(x), c ='#00ffff', linestyle = '-.', linewidth = 2, marker = 'o', label =
plt.subplot(1, 4, 3)
plt.plot(x, np.sin(x), c ='g', linestyle = '--', linewidth = 2, marker = '+', label = 'y =

plt.legend()
plt.show()
plt.subplot(2, 4, 4)
plt.plot(x, x ** 2, c ='m', linestyle = '-.', linewidth = 2, marker = 'P', label = 'Y = x *
plt.show()
```





Scatter Plot

In [30]:

```
x = np.arange(1, 50, 5)
y = np.random.randint(1,50, size = len(x))
```

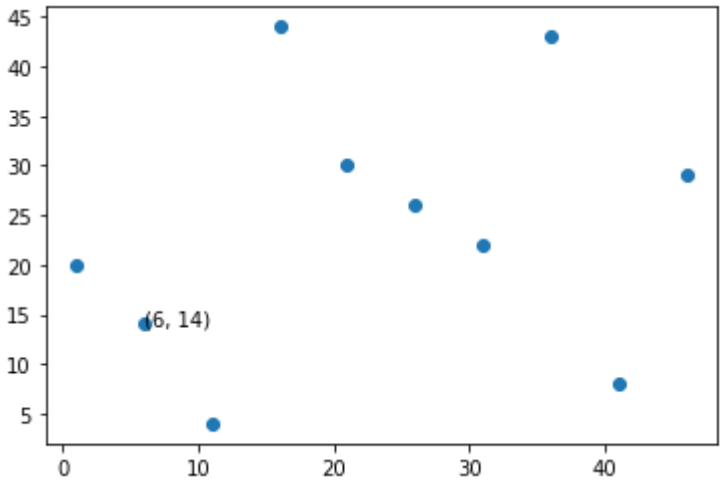
In [31]:

```
print(x, y)
```

```
[ 1  6 11 16 21 26 31 36 41 46] [20 14  4 44 30 26 22 43  8 29]
```

In [43]:

```
plt.scatter(x, y)
plt.text(6, 14, '(6, 14)')
plt.show()
```



In [42]:

```
help(plt.scatter)
```

```
Help on function scatter in module matplotlib.pyplot:

scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=Non
e, vmax=None, alpha=None, linewidths=None, verts=<deprecated parameter>, e
dgecolors=None, *, plotnonfinite=False, data=None, **kwargs)
    A scatter plot of *y* vs. *x* with varying marker size and/or color.

    Parameters
    ----------
    x, y : scalar or array-like, shape (n, )
        The data positions.

    s : scalar or array-like, shape (n, ), optional
        The marker size in points**2.
        Default is ``rcParams['lines.markersize'] ** 2``.

    c : array-like or list of colors or color, optional
        The marker colors. Possible values:

        - A scalar or sequence of n numbers to be mapped to colors using
          *cmap* and *norm*.
        - A 2-D array in which the rows are RGB or RGBA.
        - A sequence of colors of length n.
        - A single color format string.

        Note that *c* should not be a single numeric RGB or RGBA sequence
        because that is indistinguishable from an array of values to be
        colormapped. If you want to specify the same RGB or RGBA value for
        all points, use a 2-D array with a single row.  Otherwise, value-
        matching will have precedence in case of a size matching with *x*
        and *y*.

        If you wish to specify a single color for all points
        prefer the *color* keyword argument.

        Defaults to `None`. In that case the marker color is determined
        by the value of *color*, *facecolor* or *facecolors*. In case
        those are not specified or `None`, the marker color is determined
        by the next color of the ``Axes`` current "shape and fill" color
        cycle. This cycle defaults to :rc:`axes.prop_cycle`.

    marker : `~matplotlib.markers.MarkerStyle`, optional
        The marker style. *marker* can be either an instance of the class
        or the text shorthand for a particular marker.
        Defaults to ``None``, in which case it takes the value of
        :rc:`scatter.marker` = 'o'.
        See `~matplotlib.markers` for more information about marker style
s.

    cmap : `~matplotlib.colors.Colormap`, optional, default: None
        A `.Colormap` instance or registered colormap name. *cmap* is only
        used if *c* is an array of floats. If ``None``, defaults to rc
        ``image.cmap``.

    norm : `~matplotlib.colors.Normalize`, optional, default: None
        A `.Normalize` instance is used to scale luminance data to 0, 1.
```

```
        *norm* is only used if *c* is an array of floats. If *None*, use
        the default `.colors.Normalize`.

    vmin, vmax : scalar, optional, default: None
        *vmin* and *vmax* are used in conjunction with *norm* to normalize
        luminance data. If None, the respective min and max of the color
        array is used. *vmin* and *vmax* are ignored if you pass a *norm*
        instance.

    alpha : scalar, optional, default: None
        The alpha blending value, between 0 (transparent) and 1 (opaque).

    linewidths : scalar or array-like, optional, default: None
        The linewidth of the marker edges. Note: The default *edgecolors*
        is 'face'. You may want to change this as well.
        If *None*, defaults to :rc:`lines.linewidth`.

    edgecolors : {'face', 'none', *None*} or color or sequence of color, o
ptional.
        The edge color of the marker. Possible values:

        - 'face': The edge color will always be the same as the face colo
r.
        - 'none': No patch boundary will be drawn.
        - A Matplotlib color or sequence of color.

        Defaults to ``None``, in which case it takes the value of
        :rc:`scatter.edgecolors` = 'face'.

        For non-filled markers, the *edgecolors* kwarg is ignored and
        forced to 'face' internally.

    plotnonfinite : boolean, optional, default: False
        Set to plot points with nonfinite *c*, in conjunction with
        `~matplotlib.colors.Colormap.set_bad`.

    Returns
    -------
    paths : `~matplotlib.collections.PathCollection`

    Other Parameters
    ----------------
    **kwargs : `~matplotlib.collections.Collection` properties

    See Also
    --------
    plot : To plot scatter plots when markers are identical in size and
        color.

    Notes
    -----
    * The `.plot` function will be faster for scatterplots where markers
      don't vary in size or color.

    * Any or all of *x*, *y*, *s*, and *c* may be masked arrays, in which
      case all masks will be combined and only unmasked points will be
      plotted.

    * Fundamentally, scatter works with 1-D arrays; *x*, *y*, *s*, and *c*
      may be input as N-D arrays, but within scatter they will be
      flattened. The exception is *c*, which will be flattened only if its
```

```
      size matches the size of *x* and *y*.

    .. note::
        In addition to the above described arguments, this function can ta
ke a
        **data** keyword argument. If such a **data** argument is given, t
he
        following arguments are replaced by **data[<arg>]**:

        * All arguments with the following names: 'c', 'color', 'edgecolor
s', 'facecolor', 'facecolors', 'linewidths', 's', 'x', 'y'.

        Objects passed as **data** must support item access (``data[<arg>]
``) and
        membership test (``<arg> in data``).
```

Bar Garph

Histogram

In [44]:

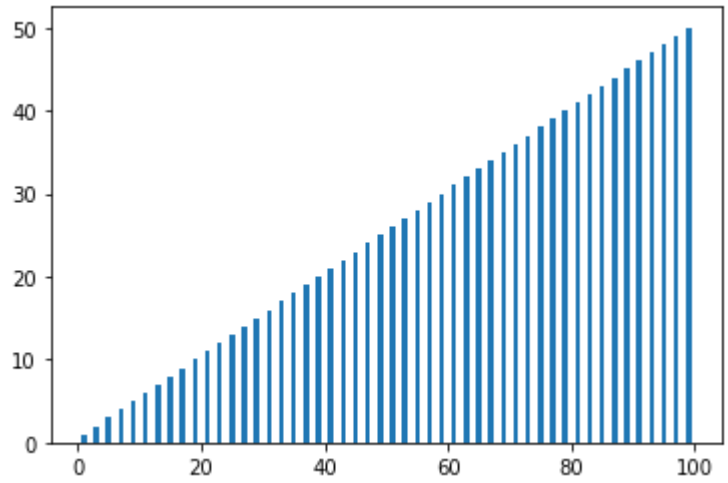
```
x = np.arange(1, 100, 2)
y = np.linspace(1, 50, len(x))
```

In [45]:

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

Out[45]:

```
<BarContainer object of 50 artists>
```



In [48]:

```
print(y)
```

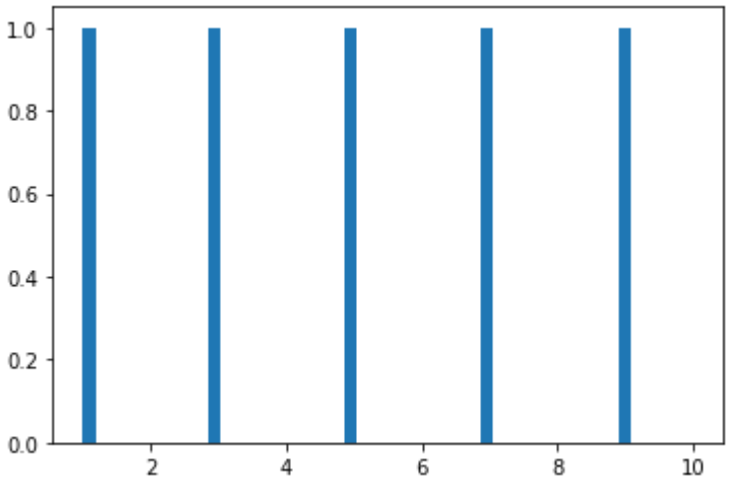
```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.
 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29. 30. 31. 32. 33. 34. 35. 36.
 37. 38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50.]
```

In [49]:

```
x = np.arange(1, 100, 2)
y = np.linspace(1, 10, len(x))
plt.hist(x, y)
```

Out[49]:

```
(array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]),
 array([ 1.        ,  1.18367347,  1.36734694,  1.55102041,  1.73469388,
         1.91836735,  2.10204082,  2.28571429,  2.46938776,  2.65306122,
         2.83673469,  3.02040816,  3.20408163,  3.3877551 ,  3.57142857,
         3.75510204,  3.93877551,  4.12244898,  4.30612245,  4.48979592,
         4.67346939,  4.85714286,  5.04081633,  5.2244898 ,  5.40816327,
         5.59183673,  5.7755102 ,  5.95918367,  6.14285714,  6.32653061,
         6.51020408,  6.69387755,  6.87755102,  7.06122449,  7.24489796,
         7.42857143,  7.6122449 ,  7.79591837,  7.97959184,  8.16326531,
         8.34693878,  8.53061224,  8.71428571,  8.89795918,  9.08163265,
         9.26530612,  9.44897959,  9.63265306,  9.81632653, 10.        ]),
 <a list of 49 Patch objects>)
```



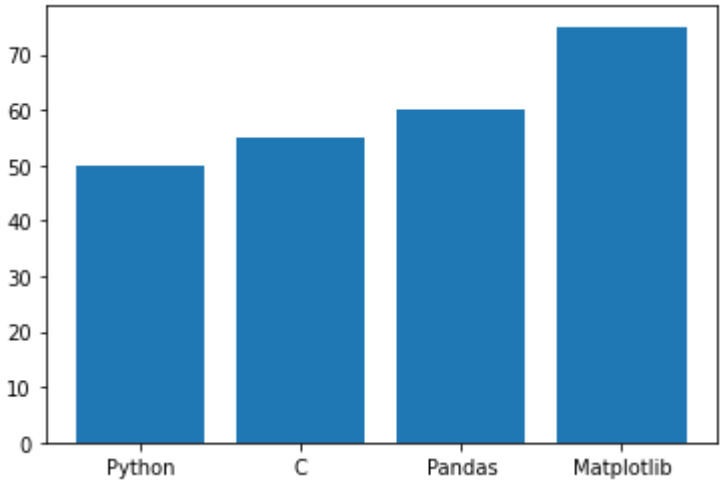
In [53]:

```
x = ['Python', 'C', 'Pandas', 'Matplotlib']

y = [50, 55, 60, 75]
plt.bar(x, y)
```

Out[53]:

```
<BarContainer object of 4 artists>
```



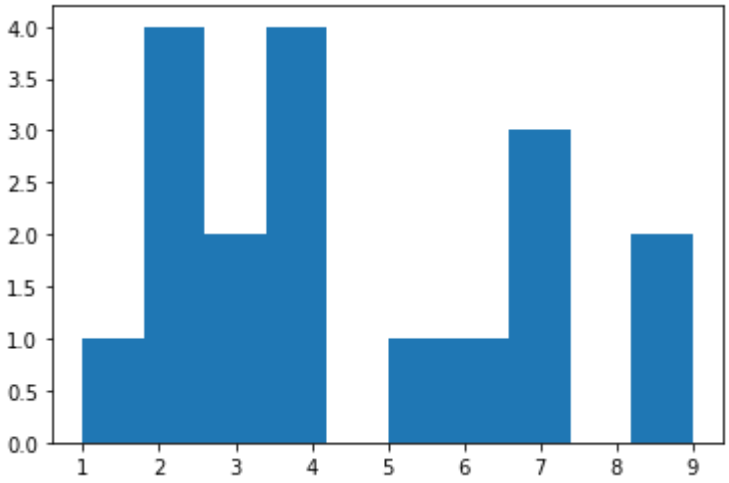
Univarient

In [54]:

```
x =  [1, 2, 2,4,4,5,7,6,9,9,3,3,2,2,4,4,7,7]
plt.hist(x)
```

Out[54]:

```
(array([1., 4., 2., 4., 0., 1., 1., 3., 0., 2.]),
 array([1. , 1.8, 2.6, 3.4, 4.2, 5. , 5.8, 6.6, 7.4, 8.2, 9. ]),
 <a list of 10 Patch objects>)
```

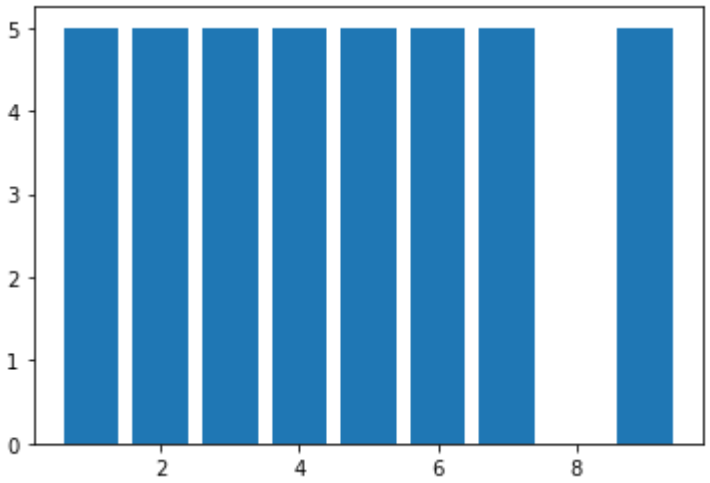


In [56]:

```
plt.bar(x, height = 5)
```

Out[56]:

```
<BarContainer object of 18 artists>
```



In [57]:

```
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation/
```

In [58]:

```
df.head()
```

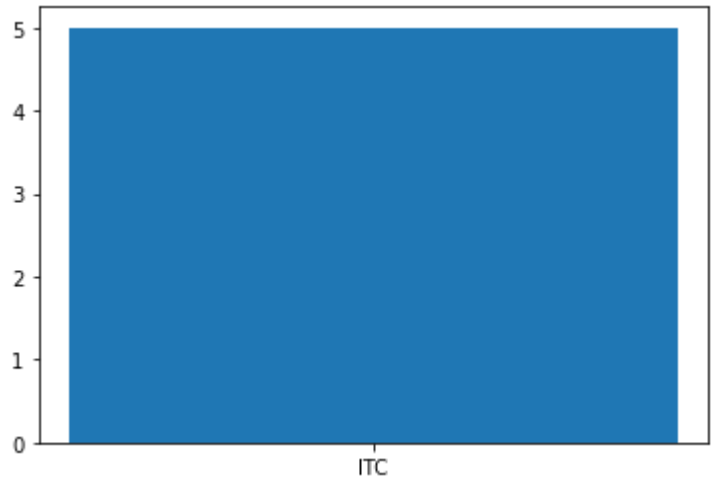
Out[58]:

	Symbol	Series	Date	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price	Total Traded Quantity	
0	ITC	EQ	15-May-2017	274.95	275.90	278.90	275.50	278.50	277.95	277.78	5462855	1.
1	ITC	EQ	16-May-2017	277.95	278.50	284.30	278.00	283.00	283.45	280.93	11204308	3.
2	ITC	EQ	17-May-2017	283.45	284.10	284.40	279.25	281.50	281.65	281.56	8297700	2.
3	ITC	EQ	18-May-2017	281.65	278.00	281.05	277.05	277.65	277.90	278.49	7924261	2.
4	ITC	EQ	19-May-2017	277.90	282.25	295.65	281.95	286.40	286.20	290.08	35724128	1.

In [61]:

```
plt.bar(df['Symbol'], height = 5)

plt.show()
```

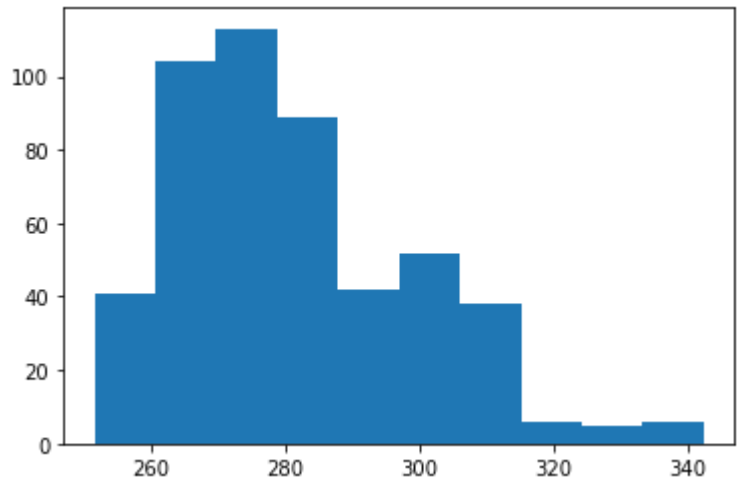


In [63]:

```
plt.hist(df['Close Price'])
```

Out[63]:

```
(array([ 41., 104., 113.,  89.,  42.,  52.,  38.,   6.,   5.,   6.]),
 array([251.6 , 260.69, 269.78, 278.87, 287.96, 297.05, 306.14, 315.23,
        324.32, 333.41, 342.5 ]),
 <a list of 10 Patch objects>)
```

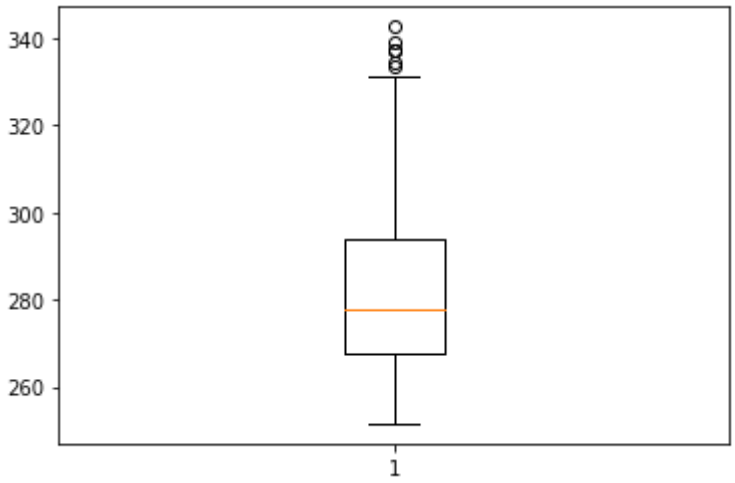


In [64]:

```
plt.boxplot(df['Close Price'])
```

Out[64]:

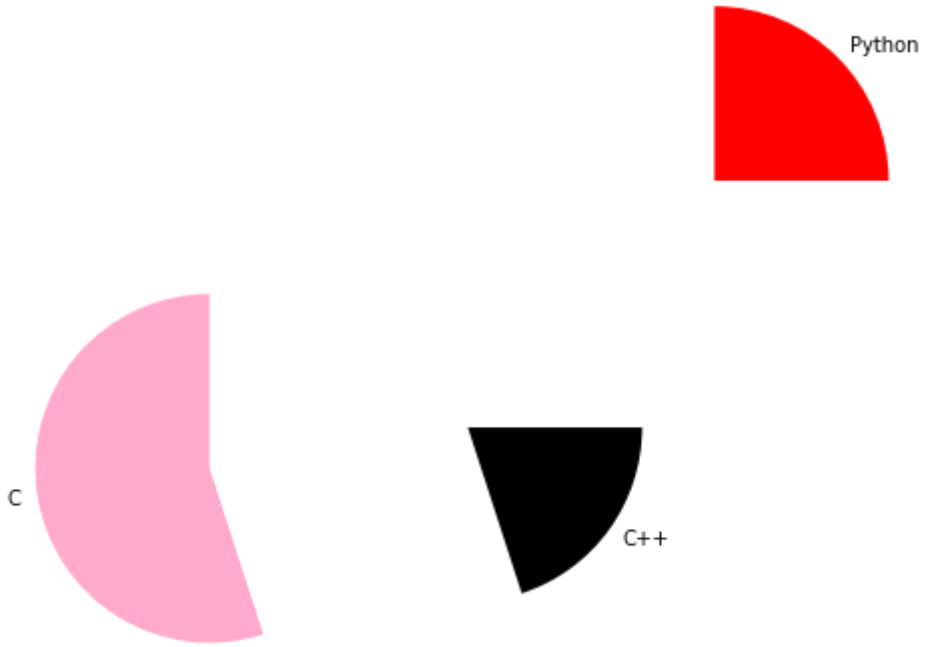
```
{'whiskers': [<matplotlib.lines.Line2D at 0x20e5117e5c8>,
  <matplotlib.lines.Line2D at 0x20e511822c8>],
 'caps': [<matplotlib.lines.Line2D at 0x20e51182c88>,
  <matplotlib.lines.Line2D at 0x20e5077da48>],
 'boxes': [<matplotlib.lines.Line2D at 0x20e5115df08>],
 'medians': [<matplotlib.lines.Line2D at 0x20e51096ec8>],
 'fliers': [<matplotlib.lines.Line2D at 0x20e51150cc8>],
 'means': []}
```



In [76]:

```
sub = ['Python', 'C', 'C++']
marks = [25,55, 20]

plt.pie(marks, colors = ['r','#ffaacc', 'k'], radius = 1, labels = sub, explode = [2, 1.5,
plt.show()
```



In [66]:

```
help(plt.pie)
```

```
Help on function pie in module matplotlib.pyplot:

pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.
6, shadow=False, labeldistance=1.1, startangle=None, radius=None, counterclo
ck=True, wedgeprops=None, textprops=None, center=(0, 0), frame=False, rotate
labels=False, *, data=None)
    Plot a pie chart.

    Make a pie chart of array *x*.  The fractional area of each wedge is
    given by ``x/sum(x)``.  If ``sum(x) < 1``, then the values of *x* give
    the fractional area directly and the array will not be normalized. The
    resulting pie will have an empty wedge of size ``1 - sum(x)``.

    The wedges are plotted counterclockwise, by default starting from the
    x-axis.

    Parameters
    ----------
    x : array-like
        The wedge sizes.

    explode : array-like, optional, default: None
        If not *None*, is a ``len(x)`` array which specifies the fraction
        of the radius with which to offset each wedge.

    labels : list, optional, default: None
        A sequence of strings providing the labels for each wedge

    colors : array-like, optional, default: None
        A sequence of matplotlib color args through which the pie chart
        will cycle.  If *None*, will use the colors in the currently
        active cycle.

    autopct : None (default), str, or function, optional
        If not *None*, is a string or function used to label the wedges
        with their numeric value.  The label will be placed inside the
        wedge.  If it is a format string, the label will be ``fmt%pct``.
        If it is a function, it will be called.

    pctdistance : float, optional, default: 0.6
        The ratio between the center of each pie slice and the start of
        the text generated by *autopct*.  Ignored if *autopct* is *None*.

    shadow : bool, optional, default: False
        Draw a shadow beneath the pie.

    labeldistance : float or None, optional, default: 1.1
        The radial distance at which the pie labels are drawn.
        If set to ``None``, label are not drawn, but are stored for use in
        ``legend()``

    startangle : float, optional, default: None
        If not *None*, rotates the start of the pie chart by *angle*
        degrees counterclockwise from the x-axis.

    radius : float, optional, default: None
```

```
        The radius of the pie, if *radius* is *None* it will be set to 1.

    counterclock : bool, optional, default: True
        Specify fractions direction, clockwise or counterclockwise.

    wedgeprops : dict, optional, default: None
        Dict of arguments passed to the wedge objects making the pie.
        For example, you can pass in ``wedgeprops = {'linewidth': 3}``
        to set the width of the wedge border lines equal to 3.
        For more details, look at the doc/arguments of the wedge object.
        By default ``clip_on=False``.

    textprops : dict, optional, default: None
        Dict of arguments to pass to the text objects.

    center :  list of float, optional, default: (0, 0)
        Center position of the chart. Takes value (0, 0) or is a sequence
        of 2 scalars.

    frame : bool, optional, default: False
        Plot axes frame with the chart if true.

    rotatelabels : bool, optional, default: False
        Rotate each label to the angle of the corresponding slice if true.

    Returns
    -------
    patches : list
        A sequence of :class:`matplotlib.patches.Wedge` instances

    texts : list
        A list of the label :class:`matplotlib.text.Text` instances.

    autotexts : list
        A list of :class:`~matplotlib.text.Text` instances for the numeric
        labels. This will only be returned if the parameter *autopct* is
        not *None*.

    Notes
    -----
    The pie chart will probably look best if the figure and axes are
    square, or the Axes aspect is equal.
    This method sets the aspect ratio of the axis to "equal".
    The axes aspect ratio can be controlled with `Axes.set_aspect`.

    .. note::
        In addition to the above described arguments, this function can take
a
        **data** keyword argument. If such a **data** argument is given, the
        following arguments are replaced by **data[<arg>]**:

        * All arguments with the following names: 'colors', 'explode', 'labe
ls', 'x'.

        Objects passed as **data** must support item access (``data[<arg>]`
`) and
        membership test (``<arg> in data``).
```