Day 4: Programming with Python

Data Visualization

Data visualization is a way to show complex data in a form that is graphical and easy to understand. This can be especially useful when one is trying to explore the data.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is one of the most widely used, if not the most popular data visualization library in Python.

Figure

Every plot that matplotlib makes is drawn on something called `Figure`. `Figure` object is as a canvas that holds all the subplots and other plot elements inside it. It is the top level container for all the plot elements. Each Figure can have multiple Axes.

pyplot

The pyplot module contains command-style functions and each of them makes some changes to the `Figure` object. For example, the `pyplot` has functions for the creation of a plotting area and decoration of the plot with a label.

`Pyplot` tracks the status of the current figure and its plotting area. The functions act on the current figure.

0. Import matplotlib

Let's start by importing all the required libraries. In particular, we will need `matplotlib`, and `Pandas`

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import random
```

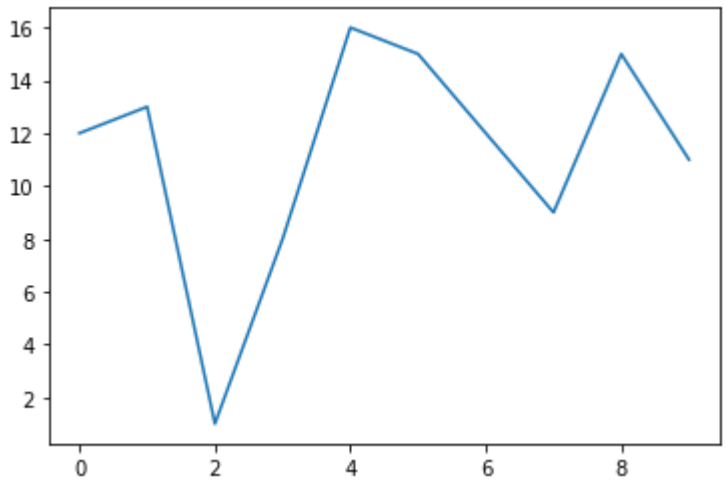
1. Basic plotting

Key points:

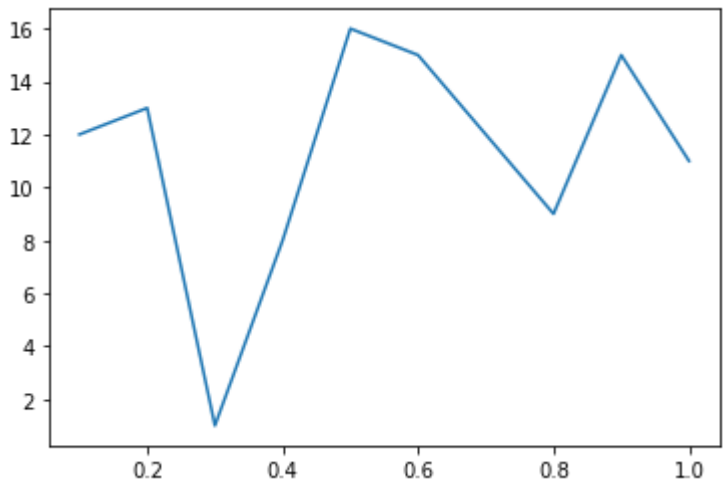
- **plt.plot()** - Creates a plot
- **plt.title()** - Sets a title to the plot
- **plt.xlabel()** - Sets a label at the x-axis
- **plt.ylabel()** - Sets a label at the y-axis
- **plt.figure(figsize=(x,y))** - Creates a figure of width x and height y
- **plt.subplot(nrows, ncols, index)** - Creates a subplot
- **fig.suptitle()** - Sets a title to the figure

The simplest method we can use for a plot is the `plot()` method. To see our plot we use the `show()` method. Let's create an array with and plot it

In [2]:
```
random.seed(0)
y = []

for i in range(10):
    y.append(random.randint(0, 20))

plt.plot(y)
plt.show()
```



In [3]:
```
x = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]


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



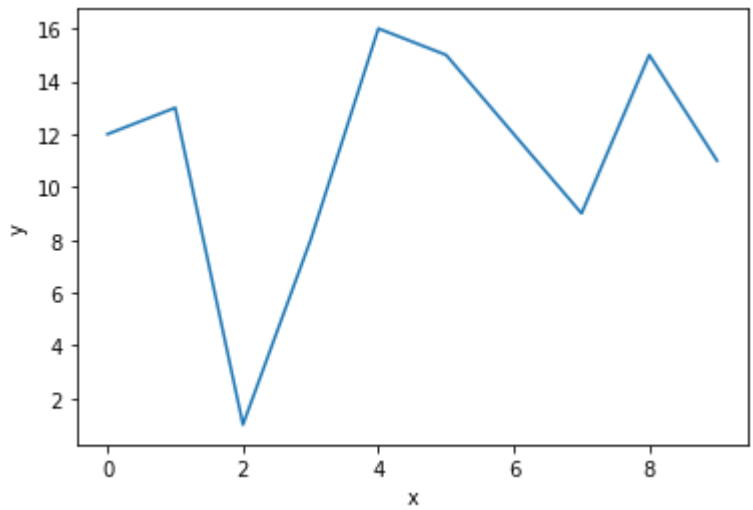
Adding elements to the chart

We notice that the plot does not have any information. We can start by adding the title.

In [4]:
```
plt.title('My first plot')
plt.plot(y)
plt.show()
```

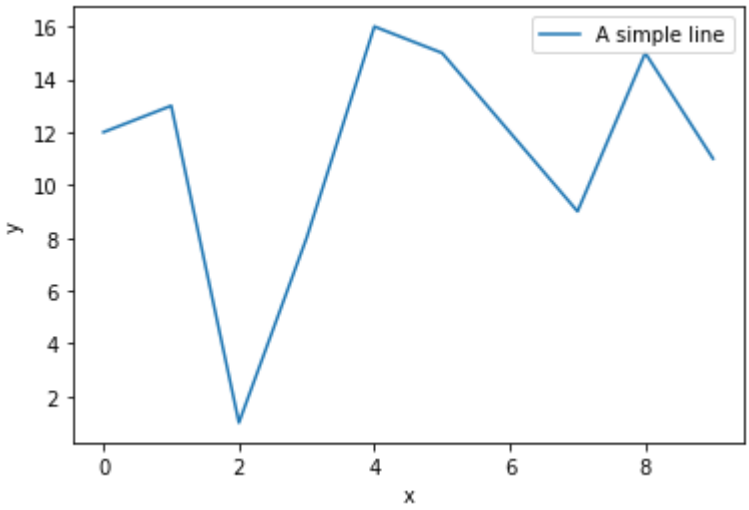
Another important element is the labels at the axes. We can add those with the `xlabel()` and `ylabel()` .

In [5]:
```
plt.xlabel('x')
plt.ylabel('y')
plt.plot(y)
plt.show()
```



Finally, let's add a legend

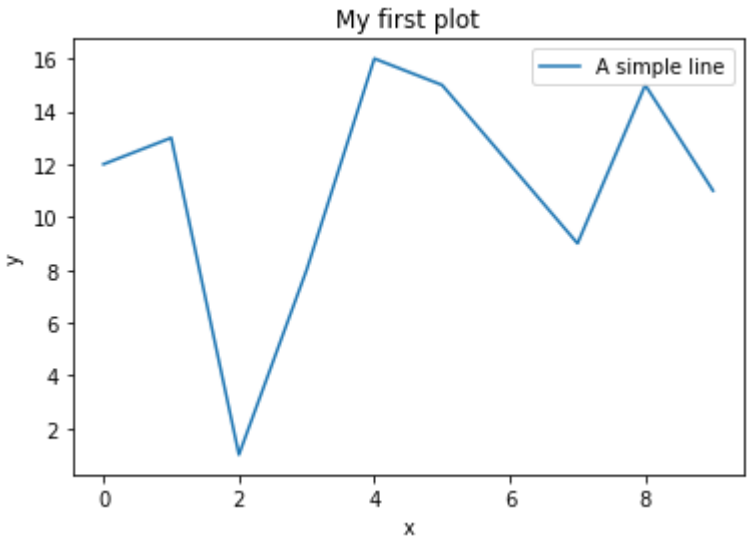
In [6]:
```
plt.xlabel('x')
plt.ylabel('y')
plt.plot(y)

plt.legend(['A simple line'])
plt.show()
```



And all together

In [7]:

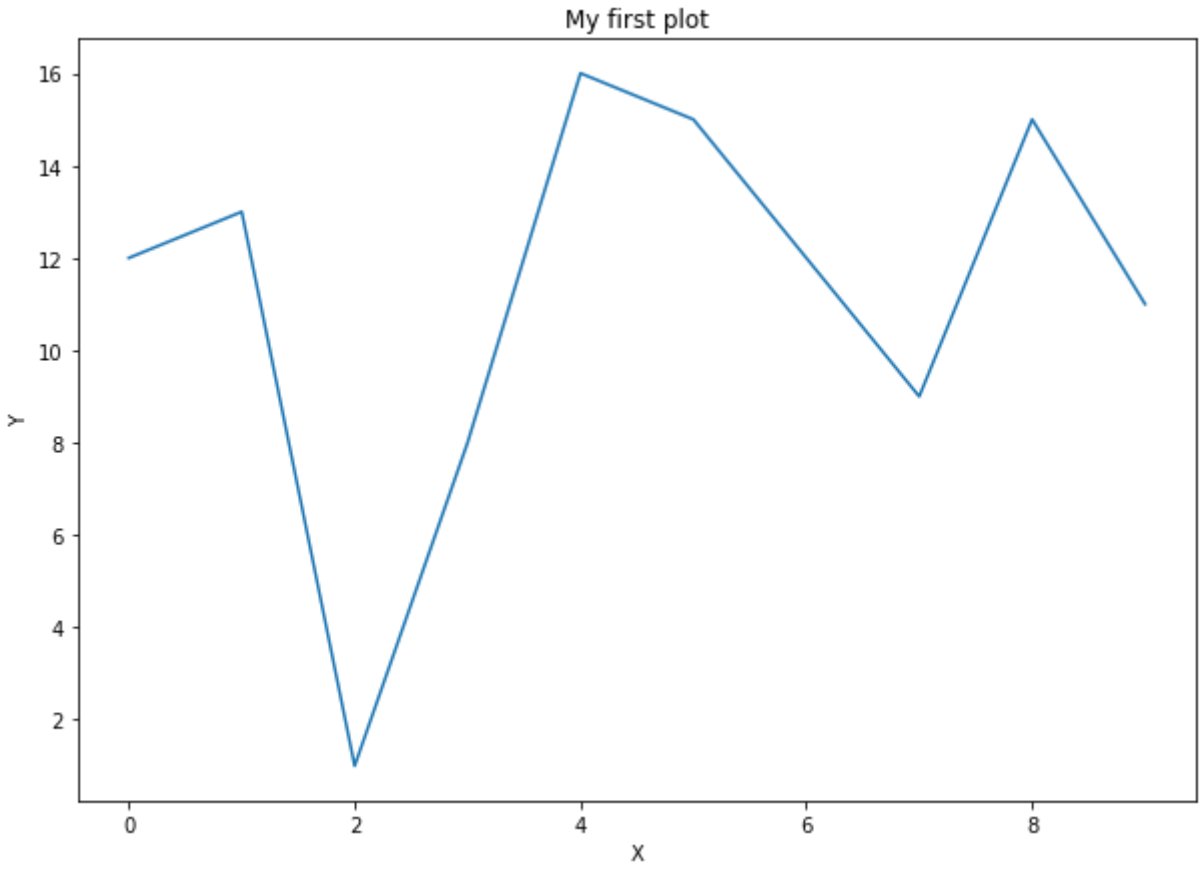
```
plt.title('My first plot')
plt.xlabel('x')
plt.ylabel('y')
plt.plot(y)
plt.legend(['A simple line'])
plt.show()
```



Let's say we want to change the size of the figure. For that I have to create a Figure object

In [8]:

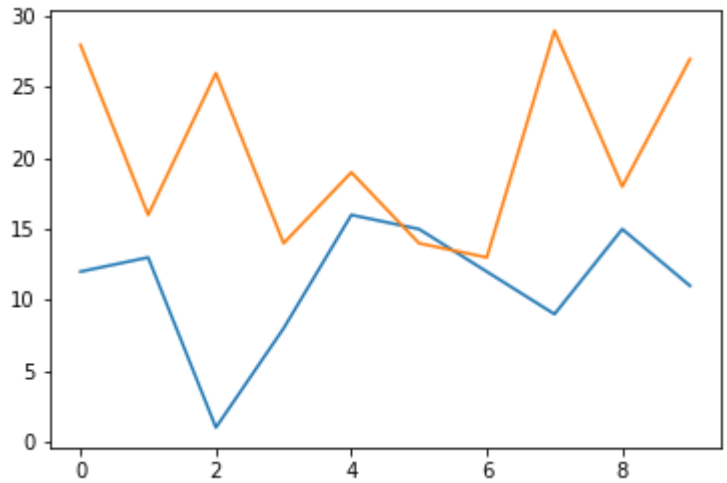
```
plt.figure(figsize=(10,7))
plt.title('My first plot')
plt.xlabel('X')
plt.ylabel('Y')
plt.plot(y)
plt.show()
```



Let's say now we want to get in the same plot two lines

In [9]:

```
y2 = []
for i in range(10):
    y2.append(random.randint(10, 30))

plt.plot(y)
plt.plot(y2)

plt.show()
```

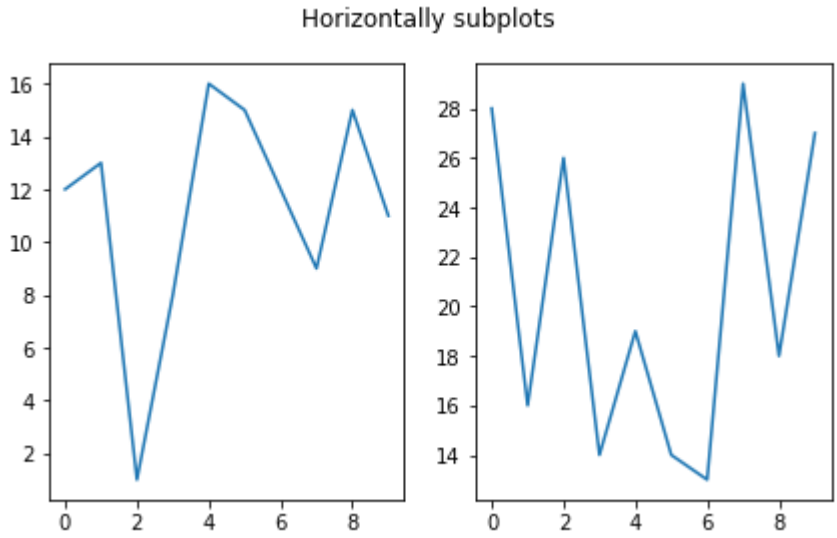


Subplots

Sometimes we want to show different plots and place them next to each other. We do that with the `subplot()` function. The `subplot()` takes as arguments the number of rows, the number of colums and the index of the current plot.

With the `suptitle()` I can set the title of the figure.

In [10]:
```
fig = plt.figure(figsize=(7,4))
ax1 = plt.subplot(1, 2, 1)
ax2 = plt.subplot(1, 2, 2)
ax1.plot(y)
ax2.plot(y2)

fig.suptitle('Horizontally subplots')
plt.show()
```



2. Scatter plots

- **plt.scatter()** - Plots a scatter plot

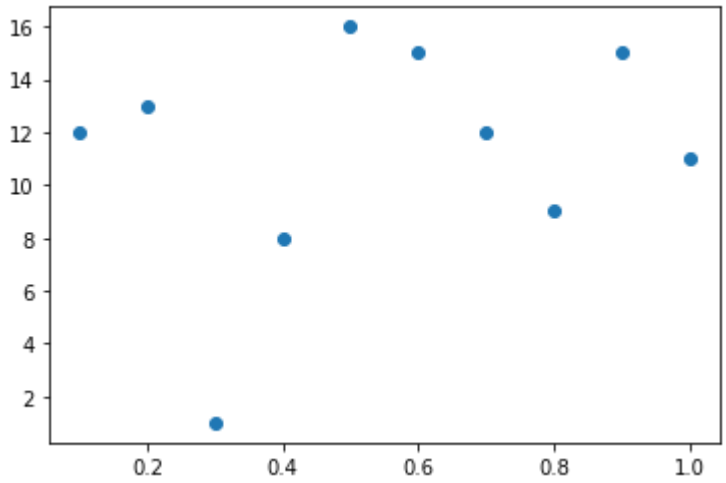
Scatterplot is used to visualise the relationship between two columns/series of data. The graph displays the collection of data points without connecting. The chart needs two variables, one variable shows X-position and the second variable shows Y-position.

Let's create a simple scatter plot with some data.

In [11]:
```
print(x)
print(y)
plt.scatter(x, y)
```

```
[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
[12, 13, 1, 8, 16, 15, 12, 9, 15, 11]
```

Out[11]:
```
<matplotlib.collections.PathCollection at 0x121d02d00>
```

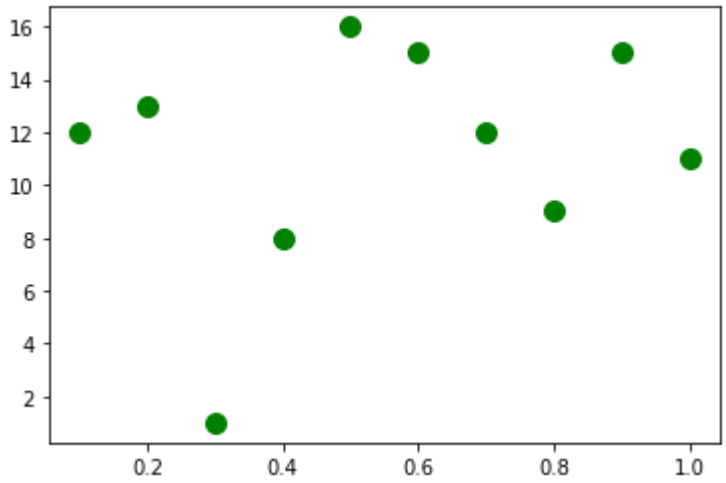


Style of scatter plots

There is a range of different arguments that we can use to change the style of the scatter plot. For example, we can change the size with the `s` parameter and the color of the dots with `color` parameter.

In [12]:
```
plt.scatter(x, y, s=100, color='green')
```

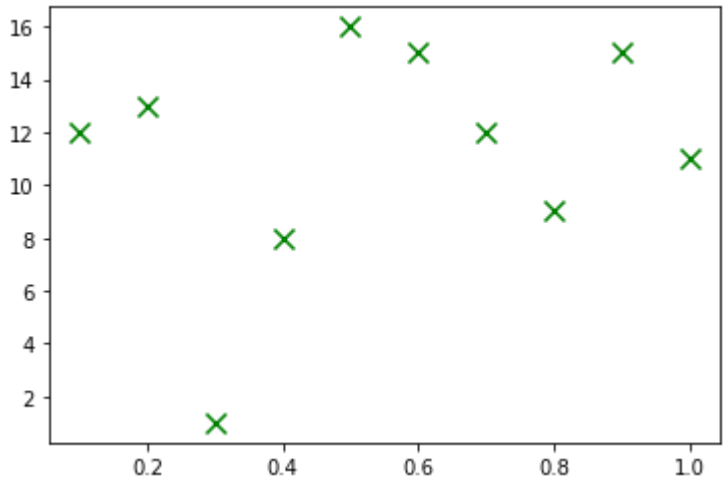
Out[12]:
```
<matplotlib.collections.PathCollection at 0x121d349d0>
```



With the `marker` parameter we can change the shape of the points.

In [13]:
```
plt.scatter(x, y, s=100, color='green', marker='x')
```

Out[13]:
```
<matplotlib.collections.PathCollection at 0x121c3b790>
```



3. Bar graphs

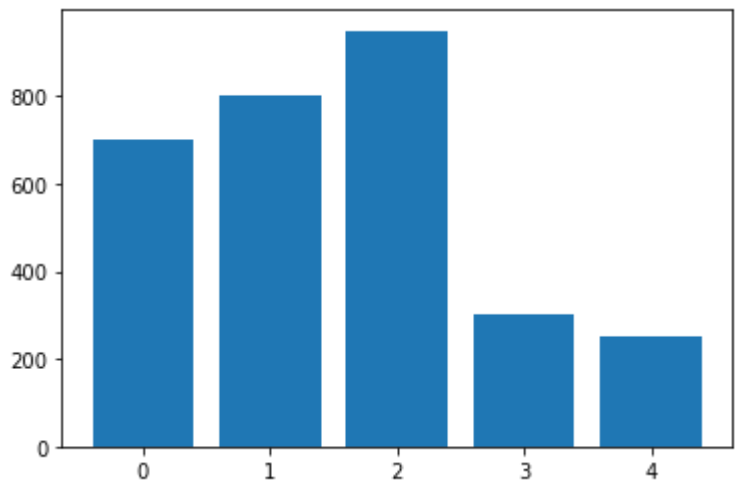
- **plt.bar()** - Plots a bar graph

Bar graph represents the data using bars either in horizontal or vertical directions. Bar graphs are used to show two or more values and typically the x-axis should be categorical data. The length of the bar is proportional to the counts of the categorical variable on x-axis. The function used to show bar graph is `plt.bar()` . The `bar()` function expects two lists of values one on x-coordinate and another on y-coordinate.

Let's assume we have products and number of units sold in March and April in a specific store.

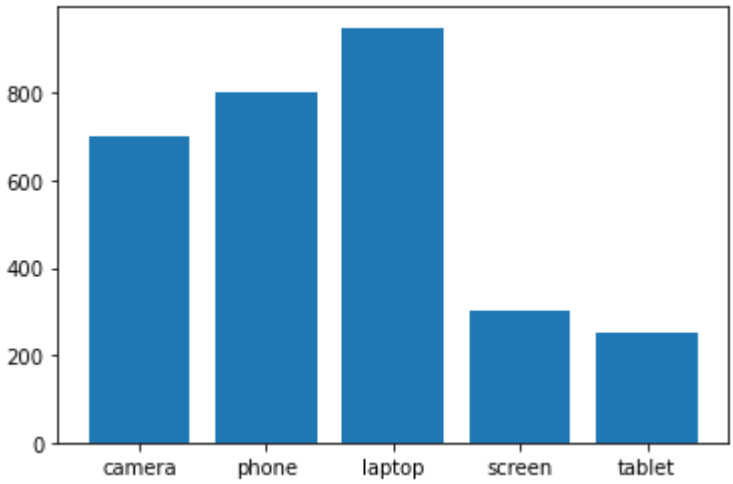
Let's plot the sales of March

In [14]:
```
product = ['camera','phone','laptop','screen','tablet']
March =[700, 800, 950, 300, 250]
April =[600, 900, 920, 130, 210]

x = range(len(product))
df = pd.DataFrame({'product': product, 'March': March, 'April': April})

plt.bar(x, df['March'])
plt.show()
```



I have to assign xticks to see my products on x-axis

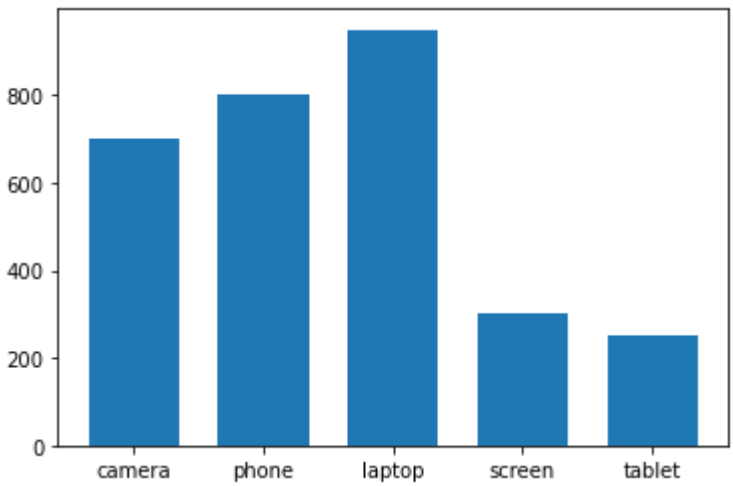
In [15]:
```
x = range(len(product))

plt.bar(x, df['March'])
plt.xticks(x, product)
plt.show()
```



With the argument `width` we can set the width of the columns

In [16]:
```
plt.bar(x, df['March'], width = 0.7)
plt.xticks(x, product)
```

Out[16]:
```
([<matplotlib.axis.XTick at 0x1223d9e80>,
  <matplotlib.axis.XTick at 0x1223d9e50>,
  <matplotlib.axis.XTick at 0x12235ca00>,
  <matplotlib.axis.XTick at 0x12240f460>,
  <matplotlib.axis.XTick at 0x12240f970>],
 [Text(0, 0, 'camera'),
  Text(1, 0, 'phone'),
  Text(2, 0, 'laptop'),
  Text(3, 0, 'screen'),
  Text(4, 0, 'tablet')])
```

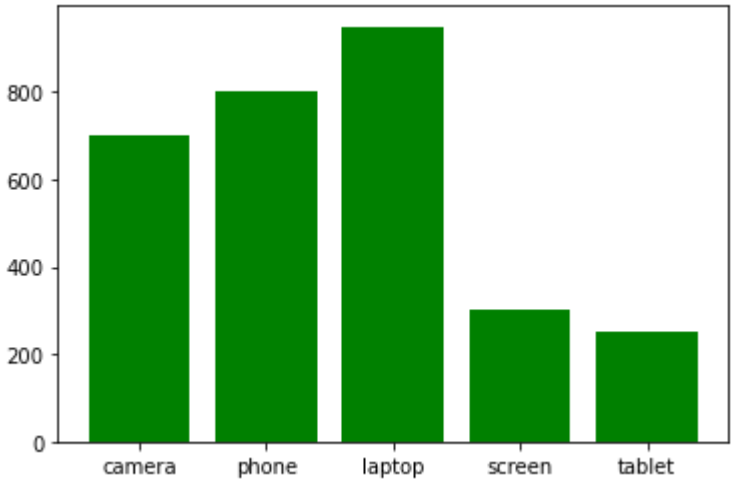


We use the `color` argument to change the color of the bars:

In [17]:
```
plt.bar(x, df['March'], color ='green')
plt.xticks(x, product)
```

Out[17]:
```
([<matplotlib.axis.XTick at 0x122452e50>,
  <matplotlib.axis.XTick at 0x122452e20>,
  <matplotlib.axis.XTick at 0x12244b9a0>,
  <matplotlib.axis.XTick at 0x1224e14c0>,
  <matplotlib.axis.XTick at 0x1224e19d0>],
```

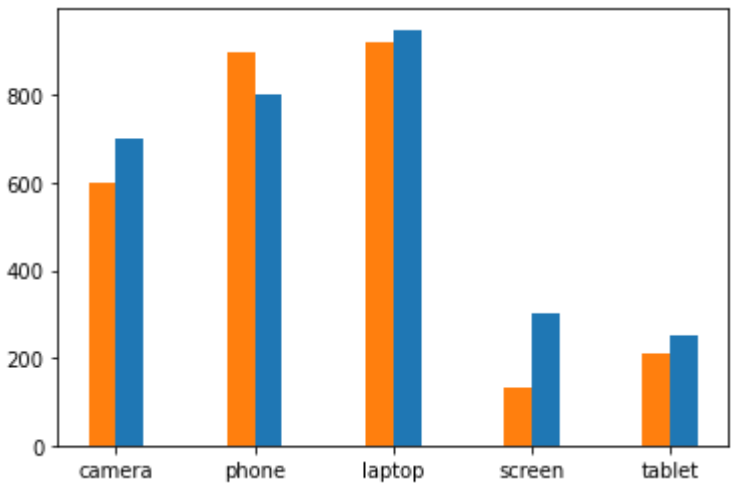
```
[Text(0, 0, 'camera'),
 Text(1, 0, 'phone'),
 Text(2, 0, 'laptop'),
 Text(3, 0, 'screen'),
 Text(4, 0, 'tablet')])
```



And now we also get data for April and we want to visualise those data with multiseries bars. I have to do a slight trick for that that is to move the March on the left on the x-axis and move April on the right

In [18]:

```
product = ['camera','phone','laptop','screen','tablet']
March =[700, 800, 950, 300, 250]
April =[600, 900, 920, 130, 210]

df = pd.DataFrame({'product': product, 'March': March, 'April': April})

x = range(len(product))

x_March = [0.1, 1.1, 2.1, 3.1, 4.1]
x_April = [-0.1, 0.9, 1.9, 2.9, 3.9]

plt.bar(x_March, df['March'], width=0.2, label = 'March')
plt.bar(x_April, df['April'], width= 0.2, label = 'April')
plt.xticks(x, product)
plt.show()
```



4. Pie chart

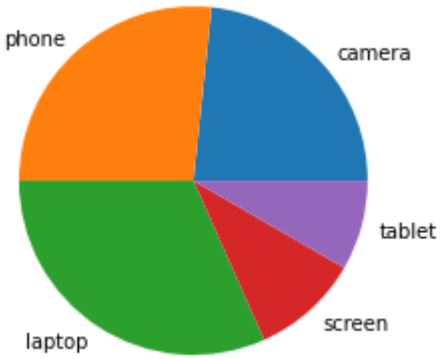
- **plt.pie()** - Plots a pie chart

A pie chart is a circular statistical graphic, which is divided into slices to illustrate numerical proportion.

To create a pie chart we use the `plt.pie()` . Let's see in an example using the same data as before:

In [19]:
```
plt.pie(df['March'], labels = product)
```

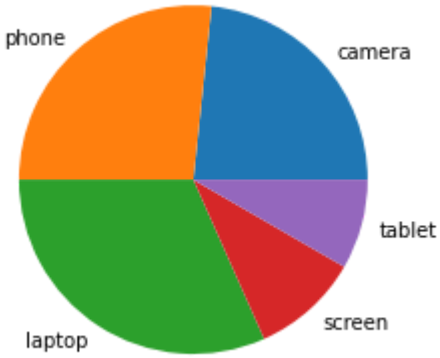
Out[19]:
```
([<matplotlib.patches.Wedge at 0x122356f40>,
  <matplotlib.patches.Wedge at 0x121c15a90>,
  <matplotlib.patches.Wedge at 0x12196f5b0>,
  <matplotlib.patches.Wedge at 0x12196f940>,
  <matplotlib.patches.Wedge at 0x121cd4460>],
 [Text(0.817459305728021, 0.7360436695459462, 'camera'),
  Text(-0.7360437078139774, 0.8174592712713291, 'phone'),
  Text(-0.5991028636588829, -0.9225376733530866, 'laptop'),
  Text(0.7360437269479921, -0.8174592540429828, 'screen'),
  Text(1.0625184333522957, -0.2847008584226318, 'tablet')])
```



Here we see the pie chart of the total sales in March. Now we want to see the sales percentage per product:

In [20]:
```
plt.pie(df['March'], labels = product)
```

Out[20]:
```
([<matplotlib.patches.Wedge at 0x12229e100>,
  <matplotlib.patches.Wedge at 0x122356a00>,
  <matplotlib.patches.Wedge at 0x12229ebe0>,
  <matplotlib.patches.Wedge at 0x10dc9e910>,
  <matplotlib.patches.Wedge at 0x10dc9e640>],
 [Text(0.817459305728021, 0.7360436695459462, 'camera'),
  Text(-0.7360437078139774, 0.8174592712713291, 'phone'),
  Text(-0.5991028636588829, -0.9225376733530866, 'laptop'),
  Text(0.7360437269479921, -0.8174592540429828, 'screen'),
  Text(1.0625184333522957, -0.2847008584226318, 'tablet')])
```



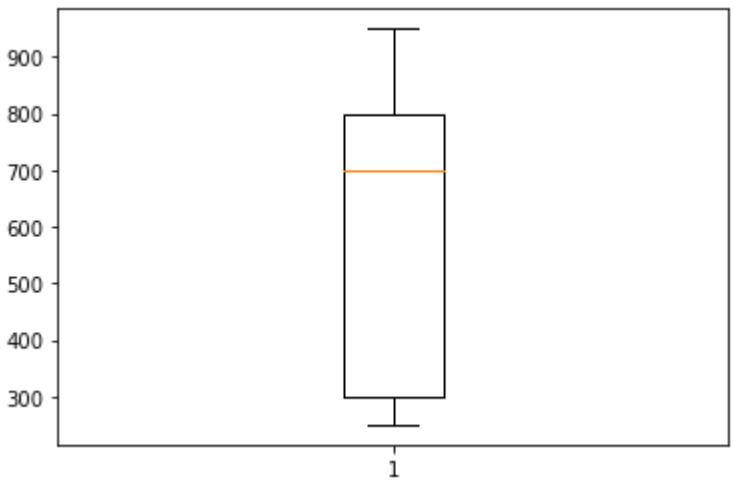
5. Box plots

- **plt.boxplot()** - Plots a boxplot

Box plots visually show the distribution of numerical data and skewness through displaying the data quartiles (or percentiles) and averages. A boxplot is a graph that gives you a good indication of how the values in the data are spread out.

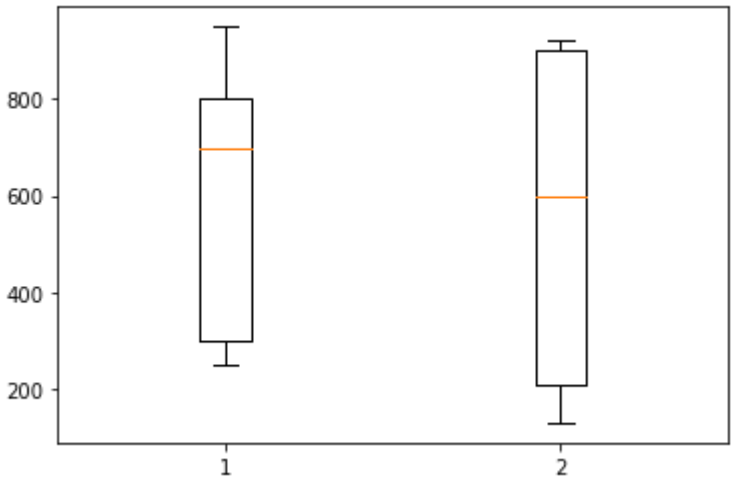
It is very easy and straightforward to create a boxplot with the matplotlib. We only have to call the `boxplot()` function. Let's create the boxplot of the sales in March

In [21]:
```
plt.boxplot(df['March'])
plt.show()
```



To add more boxplots we pass a list of lists as arguments to the `boxplot()` function.

In [22]:
```
plt.boxplot([df['March'],df['April']])
plt.show()
```



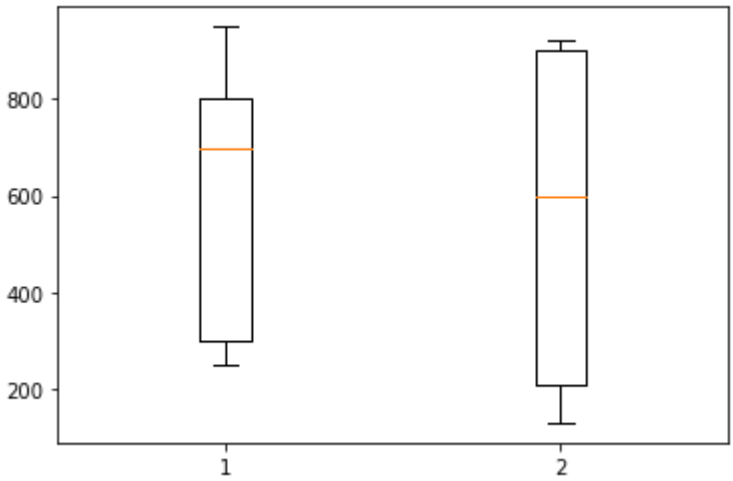
6. Save a plot

Key point:

- **plt.savefig(path)** - Saves the plot to an image at path

Finally, let's try to save the plot. To do that we call the `savefig()` function of figure object

In [23]:
```
plt.boxplot([df['March'],df['April']])
plt.savefig('../data/savedBoxplot.png')
```



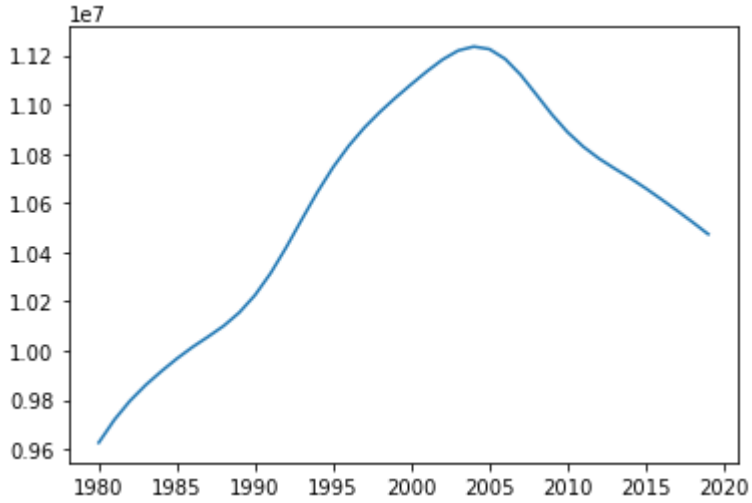
Exercises

1. Read the file greece-population-2021-06-09.csv and save it in a DataFrame df_GR

In [24]:
```
df_GR = pd.read_csv("../data/greece-population-2021-06-09.csv")
```

2. Plot the distribution of the population growth in Greece (greece-population-2021-06-09.csv) as changed in years (date and population columns)

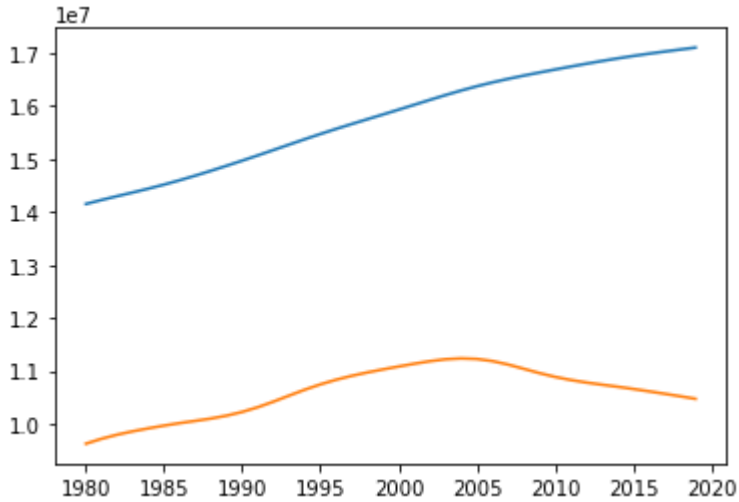
In [25]:
```
plt.plot(df_GR.date, df_GR.population)
plt.show()
```



3. Load the netherlands-population-2021-06-09.csv as well. Plot the distribution of the population growth in Greece (greece-population-2021-06-09.csv) and Netherlands (netherlands-population-2021-06-09.csv) in the same plot

In [26]:

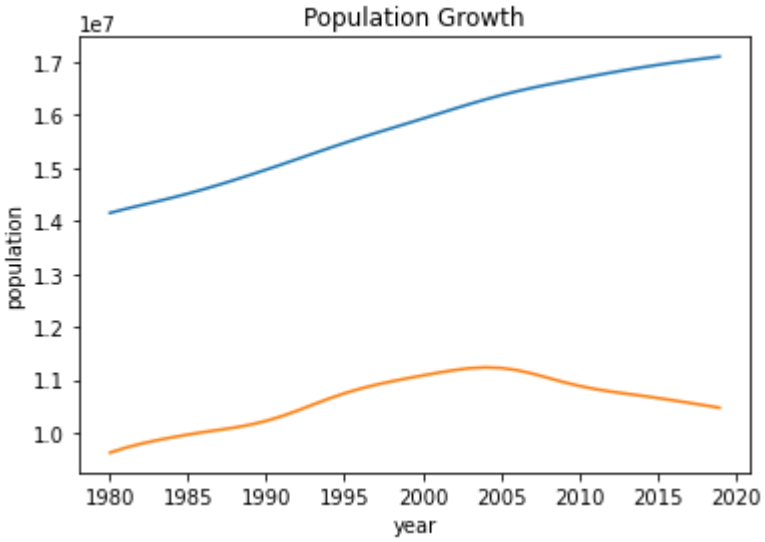
```
df_NL = pd.read_csv("../data/netherlands-population-2021-06-09.csv")
df_GR = pd.read_csv("../data/greece-population-2021-06-09.csv")

plt.plot(df_NL.date, df_NL.population)
plt.plot(df_GR.date, df_GR.population)
plt.show()
```



1. Add title, xlabel and ylabel to the plot

In [27]:

```
plt.title("Population Growth")
plt.xlabel("year")
plt.ylabel("population")

plt.plot(df_NL.date, df_NL.population, label='Netherlands')
plt.plot(df_GR.date, df_GR.population, label='Greece')
plt.show()
```

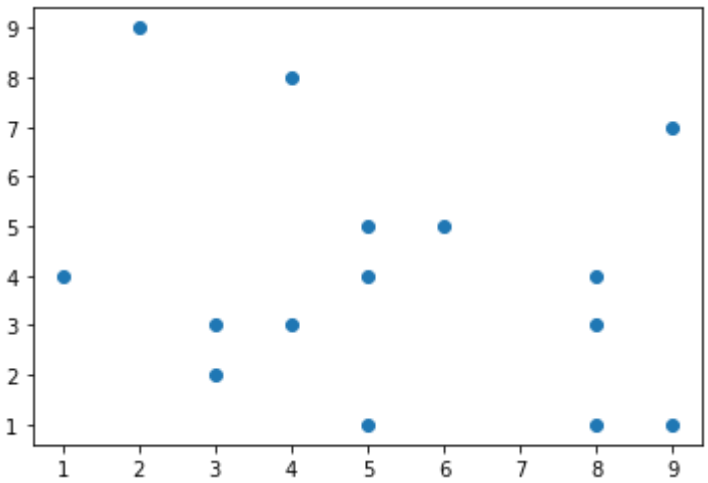


5. Let's say we have some ratings from users towards some movies.

a. Make a scatter plot of x (ratings) and y (movies_ids)
b. Increase the size of the points to 250
c. We also have colors that show the satisfaction regarding the inital expectations. Change the color of the points in order to reflect the satisfaction (search online for colorbar)

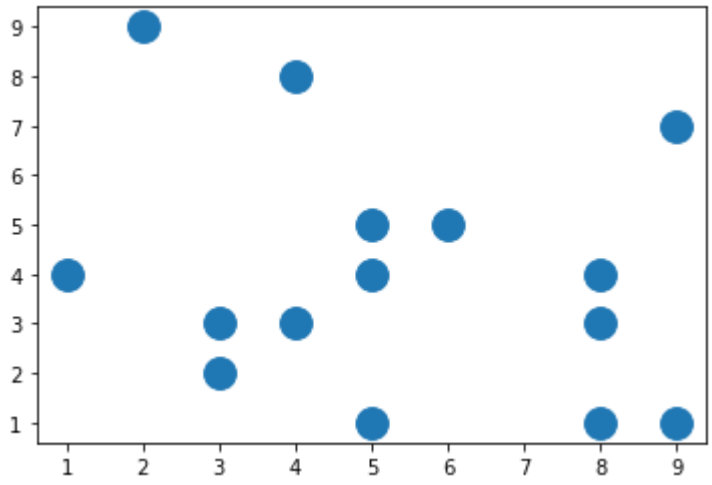
In [28]:
```
x = [5, 8, 3, 5, 8, 9, 1, 5, 9, 3, 6, 4, 2, 4, 8]
y = [4, 3, 2, 1, 1, 1, 4, 5, 7, 3, 5, 8, 9, 3, 4]

colors = [1, 4, 1, 3, 4, 1, 6, 7, 1, 9, 7, 3, 8, 4, 9]

plt.scatter(x, y)
```

Out[28]: <matplotlib.collections.PathCollection at 0x122781100>

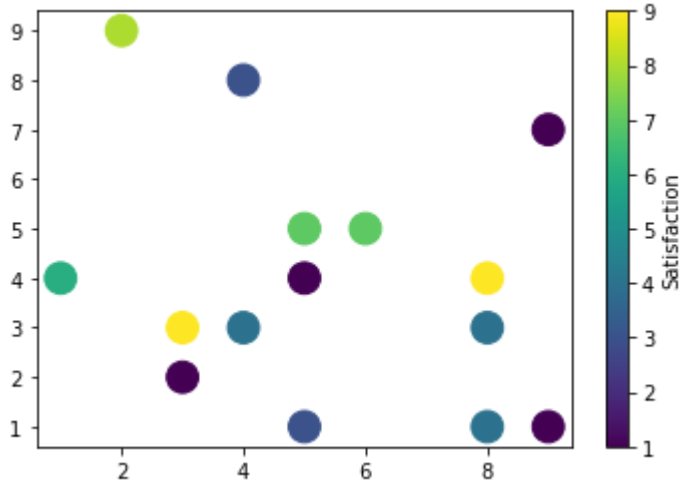


In [29]:
```
plt.scatter(x, y, s=250)
```

Out[29]: <matplotlib.collections.PathCollection at 0x122802ee0>



In [30]:
```
plt.scatter(x, y, c=colors, s=250)

cbar = plt.colorbar()
cbar.set_label("Satisfaction")
plt.show()
```



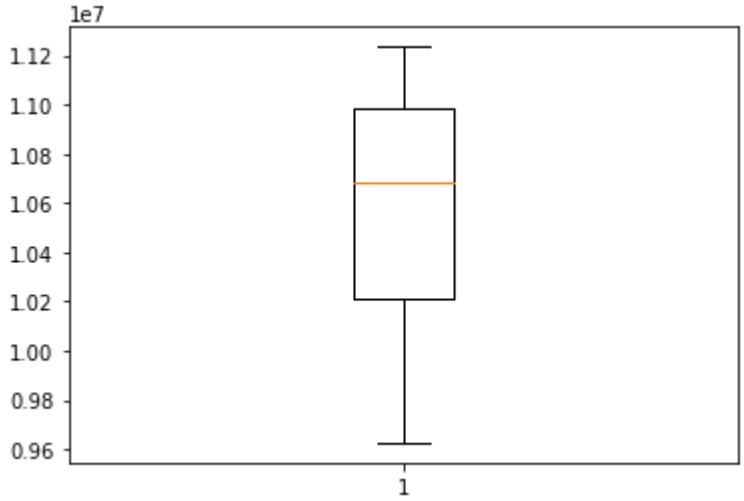
6. Load the greece-population-2021-06-09.csv file and create a boxplot of the Population

In [31]:
```
df_GR = pd.read_csv("../data/greece-population-2021-06-09.csv")
df_GR.head()
```

Out[31]:

	date	population	growth_rate
0	1980	9627002	1.17
1	1981	9719841	0.96
2	1982	9796829	0.79
3	1983	9860739	0.65
4	1984	9916482	0.57

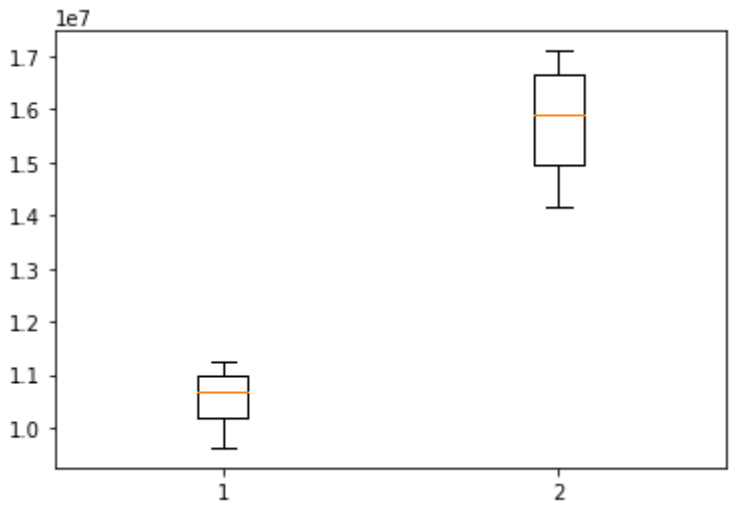
In [32]:
```
populationGreece = df_GR['population']
plt.boxplot(populationGreece)
plt.show()
```



In [ ]:

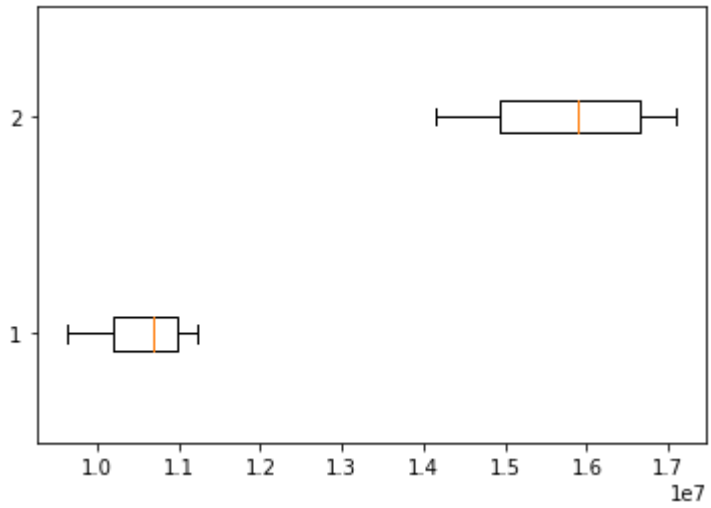
7. Load the netherlands-population-2021-06-09.csv file as well and plot the boxplots of population in Greece and the Netherlands

In [33]:
```
df_NL = pd.read_csv("../data/netherlands-population-2021-06-09.csv")

populationGreece = df_GR['population']
populationNetherlands = df_NL['population']

plt.boxplot([populationGreece, populationNetherlands])
plt.show()
```



1. Make the boxplots of the previous exercise horizontal

In [34]:
```
plt.boxplot([populationGreece, populationNetherlands], vert = 0)
plt.show()
```



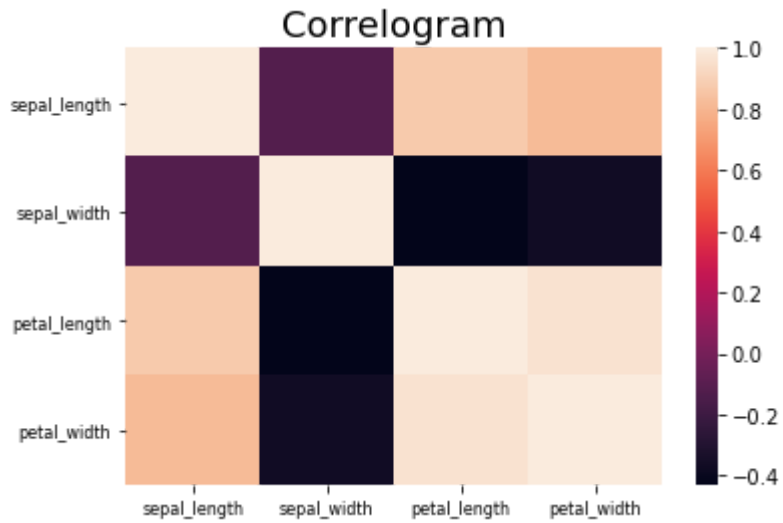
9 Bonus.

Seaborn is an open-source Python library built on top of matplotlib and is used for data visualization and exploratory data analysis.

a. Import seaborn as sns and load the iris dataset (use the load_dataset("iris"))
b. Create the Correlogram of iris features

In [35]:

```
import seaborn as sns

df = sns.load_dataset("iris")
sns.heatmap(df.corr())

plt.title('Correlogram', fontsize=18)
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.show()
```



In [ ]: