In [1]:

```
# CSV files are an important source of data. The NumPy library contains
# functions that allow you to store NumPy arrays in the form of CSV files. The
# NumPy module also allows you to load CSV files into NumPy arrays.
# To save a NumPy array in the form of a CSV file, you can use the tofile()
# method and pass it your NumPy array.
# The following script stores a two-dimensional array of three rows and four
# columns to a CSV file. You can see that you need to pass a comma “,” as the
# value for the sep parameter.

import numpy as np
my_array = np.random.randint(1,20, size = (3,4))
print(my_array)
my_array.tofile('D:\CarPrice.csv', sep = ',')

# The following NumPy array will be stored in the form of a CSV file.
# If you open the CSV file, you will seethe NumPy array you stored had
# two dimensions, it is flattened and  stored as a single row in your CSV file.
# This is the default behavior of the tofile() function.
```

```
[[14  4  4 11]
 [13 12 13 12]
 [10 16  8 16]]
```

In [7]:

```
# To store a two-dimensional NumPy array in the form of multiple rows in a
# CSV file, you can use the savetxt() method from the NumPy module, as
# shown in the following script.

import numpy as np
my_array = np.random.randint(1,20, size = (3,4))
print(my_array)
np.savetxt('D:\CarPrice.csv', my_array, delimiter=',')

# open your newly created CSV file and check the difference.
```

```
[[ 7 17 16  7]
 [13 19 14  9]
 [ 8 13 13 17]]
```

In [8]:

```
# To load a CSV file into a NumPy array, you can use the genfromtxt() method
# and pass it the CSV file along with a comma “,” as the value for the delimiter
# attribute of the genfromtxt() method. Here is an example:

import numpy as np
loaded_array = np.genfromtxt('D:\CarPrice.csv', delimiter=',')
print(loaded_array)
```

```
[[ 7. 17. 16.  7.]
 [13. 19. 14.  9.]
 [ 8. 13. 13. 17.]]
```

In [10]:

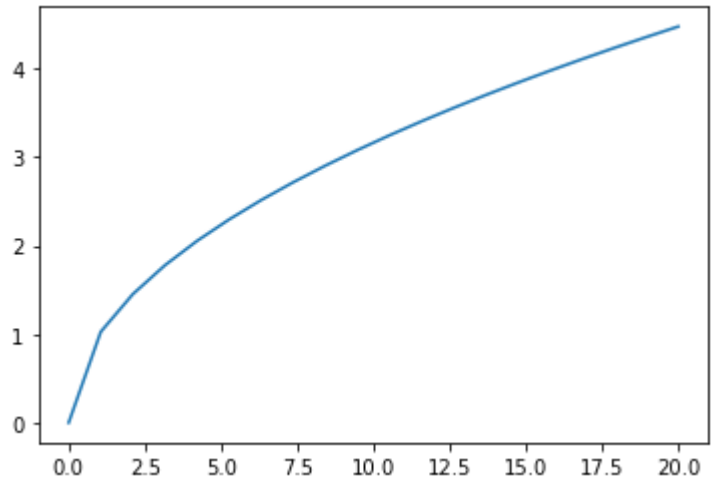
```
# You can use NumPy arrays along with Matplotlib plotting functionalities to
# plot various types of graphs. You will see some examples in this section.
# The following script creates two NumPy arrays. The first array contains 20
# equidistant numbers between 0 and 20, and the second array contains the
# square of these values. Next, the plot() method from the pyplot module of the
# Matplotlib library is used to plot a line plot using the two input arrays.

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import math
x_vals = np.linspace(0, 20, 20)
print(x_vals)
y_vals = [math.sqrt(i) for i in x_vals]
plt.plot(x_vals, y_vals)
```

```
[ 0.          1.05263158  2.10526316  3.15789474  4.21052632  5.26315789
  6.31578947  7.36842105  8.42105263  9.47368421 10.52631579 11.57894737
 12.63157895 13.68421053 14.73684211 15.78947368 16.84210526 17.89473684
 18.94736842 20.        ]
```

Out[10]:

```
[<matplotlib.lines.Line2D at 0x92e6e05190>]
```



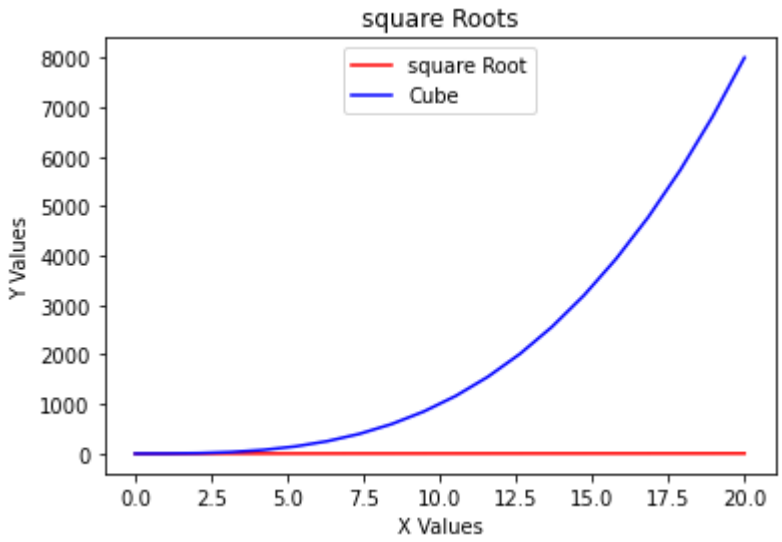
In [12]:

```
# Similarly, you can draw graphs using multiple NumPy arrays. The following
# script plots a line plot that displays squares and cubes of 20 equidistant
# numbers between 0 and 20.

import matplotlib.pyplot as plt
import numpy as np
import math
x_vals = np.linspace(0, 20, 20)
y_vals = [math.sqrt(i) for i in x_vals]
y2_vals = x_vals ** 3
plt.xlabel('X Values')
plt.ylabel('Y Values')
plt.title('square Roots')
plt.plot(x_vals, y_vals, 'r', label = 'square Root')
plt.plot(x_vals, y2_vals, 'b', label = 'Cube')
plt.legend(loc='upper center')
```

Out[12]:

```
<matplotlib.legend.Legend at 0x92e6e59190>
```

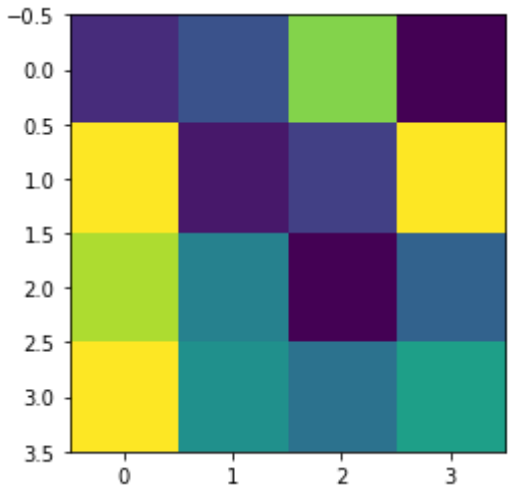


In [13]:

```
# Finally, you can also plot heatmaps using a two-dimensional NumPy array by
# passing the array to the imshow() method of the pyplot module of the
# Matplotlib library.

import numpy as np
import matplotlib.pyplot as plt
my_array = np.random.randint(1,20, size = (4,4))
print(my_array)
plt.imshow(my_array)
plt.show()
```

```
[[ 5  7 16  3]
 [19  4  6 19]
 [17 10  3  8]
 [19 11  9 12]]
```



In [14]:

```
# NumPy arrays provide a variety of functions to perform arithmetic
# operations. Some of these functions are explained in this section.
# The sqrt() function is used to find the square roots of all the elements in a
# list, as shown below:

import numpy as np
nums = [10,20,30,40,50]
np_sqr = np.sqrt(nums)
print(np_sqr)
```

```
[3.16227766 4.47213595 5.47722558 6.32455532 7.07106781]
```

In [15]:

```
# The log() function is used to find the logs of all the elements in a list, as
# shown below:

import numpy as np
nums = [10,20,30,40,50]
np_log = np.log(nums)
print(np_log)
```

```
[2.30258509 2.99573227 3.40119738 3.68887945 3.91202301]
```

In [16]:

```
# The exp() function takes the exponents of all the elements in a list, as shown
# below:

import numpy as np
nums = [10,20,30,40,50]
np_exp = np.exp(nums)
print(np_exp)
```

```
[2.20264658e+04 4.85165195e+08 1.06864746e+13 2.35385267e+17
 5.18470553e+21]
```

In [17]:

```
# You can find the sines and cosines of items in a list using the sine and cosine
# function, respectively, as shown in the following script.

import numpy as np
nums = [10,20,30,40,50]
np_sine = np.sin(nums)
print(np_sine)
nums = [10,20,30,40,50]
np_cos = np.cos(nums)
print(np_cos)
```

```
[-0.54402111  0.91294525 -0.98803162  0.74511316 -0.26237485]
[-0.83907153  0.40808206  0.15425145 -0.66693806  0.96496603]
```