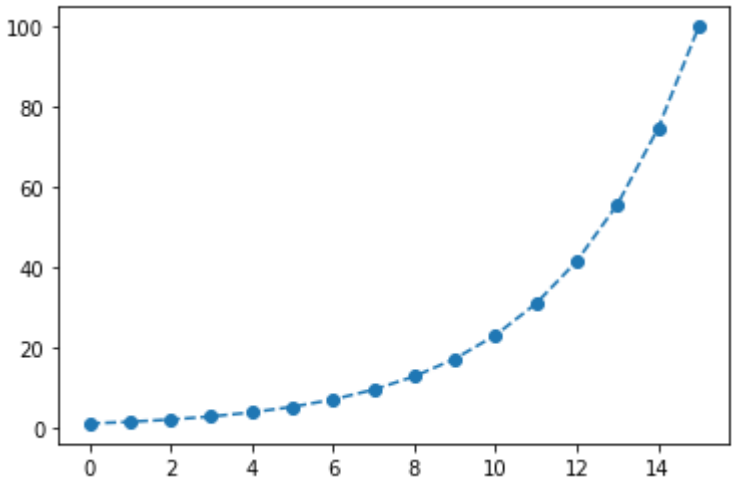
In [3]:

```
# Data Visualization using Matplotlib.
# You can compute and visualize values in the logspace as follows:

import numpy as np
import matplotlib.pyplot as plt
N = 16
x = np.linspace(0, 15, N)
y = np.logspace(0.1, 2, N)
print(y)
plt.plot(x, y, 'o--')
plt.show()
```

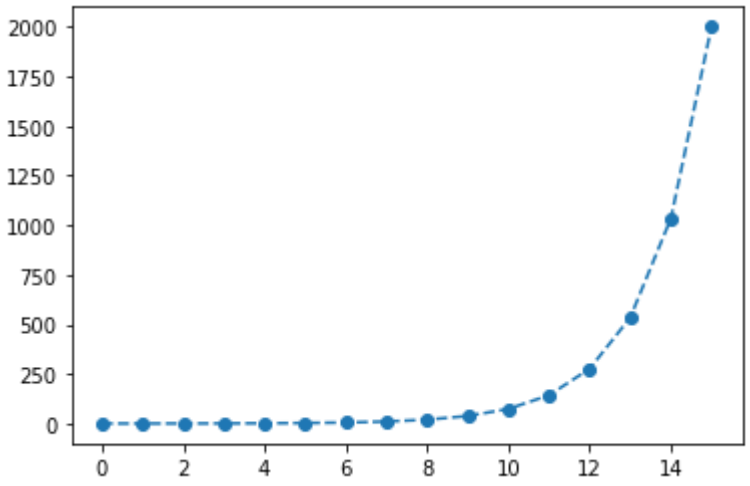
```
[  1.25892541   1.68525904   2.25597007   3.01995172   4.04265487
   5.41169527   7.2443596    9.69765359  12.98175275  17.37800829
  23.26305067  31.14105584  41.68693835  55.80417175  74.70218989
 100.        ]
```



In [4]:

```
# You can even compute a series in the geometric progression as follows:

import numpy as np
import matplotlib.pyplot as plt
N = 16
x = np.linspace(0, 15, N)
y = np.geomspace(0.1, 2000, N)
print(y)
plt.plot(x, y, 'o--')
plt.show()
```

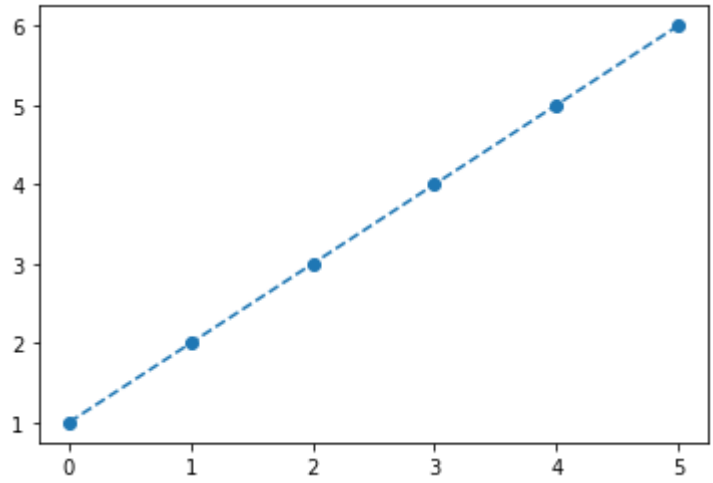
```
[1.00000000e-01 1.93524223e-01 3.74516250e-01 7.24779664e-01
 1.40262421e+00 2.71441762e+00 5.25305561e+00 1.01659351e+01
 1.96735469e+01 3.80730788e+01 7.36806300e+01 1.42589867e+02
 2.75945932e+02 5.34022222e+02 1.03346236e+03 2.00000000e+03]
```



In [5]:

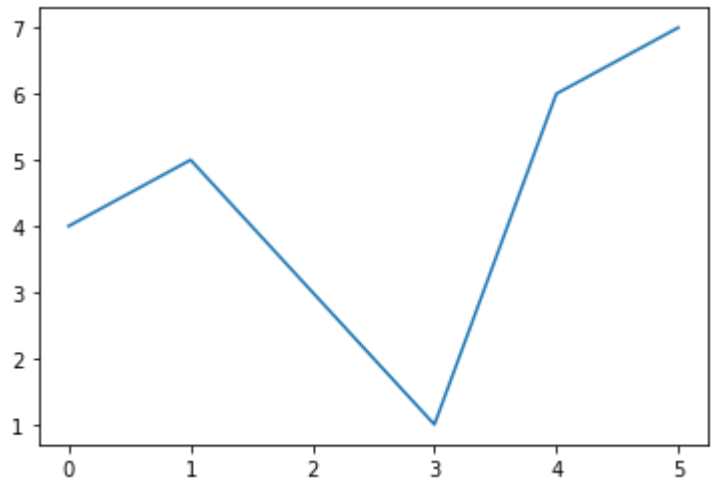
```
# You can use Python's script mode to run the Matplotlib program.

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(6)
print(x)
type(x)
y=x+1
plt.plot(x, y, 'o--')
plt.show()
```

```
[0 1 2 3 4 5]
```



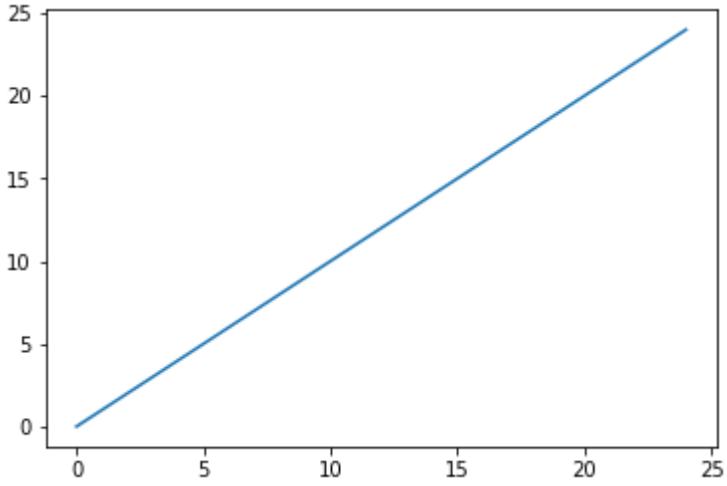
In [6]:

```
# When there is only one visualization in a figure that uses the function plot(), then t
# is known as a single-line plot.

%matplotlib inline
import matplotlib.pyplot as plt
x = [4, 5, 3, 1, 6, 7]
plt.plot(x)
plt.show()

# In this case, the values of the y-axis are assumed.
```



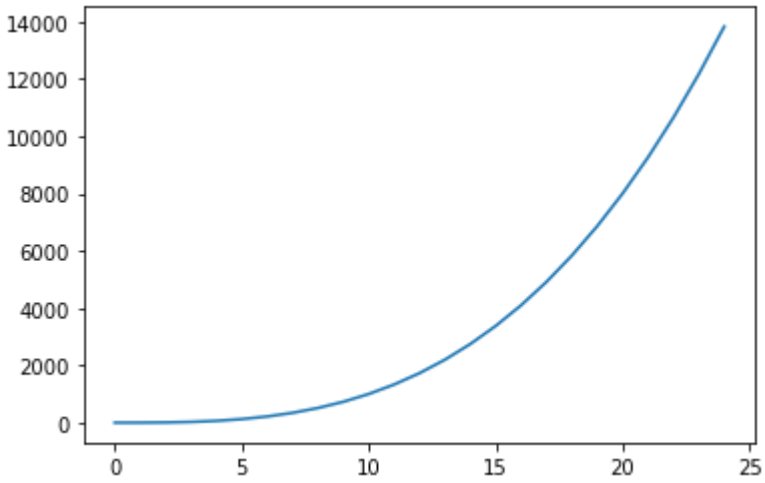
In [7]:

```
# Here's another example of a single-line graph that uses an Ndarray:

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(25)
plt.plot(x)
plt.show()
```



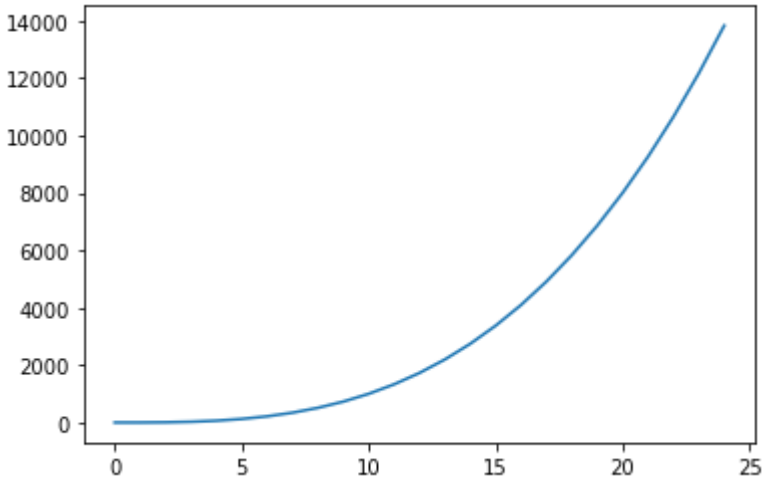
In [8]:

```
# Let's visualize the quadratic graph y = f(x) = x3+1. The code is as follows:

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(25)
plt.plot(x, [(y**3 + 1) for y in x])
plt.show()
```



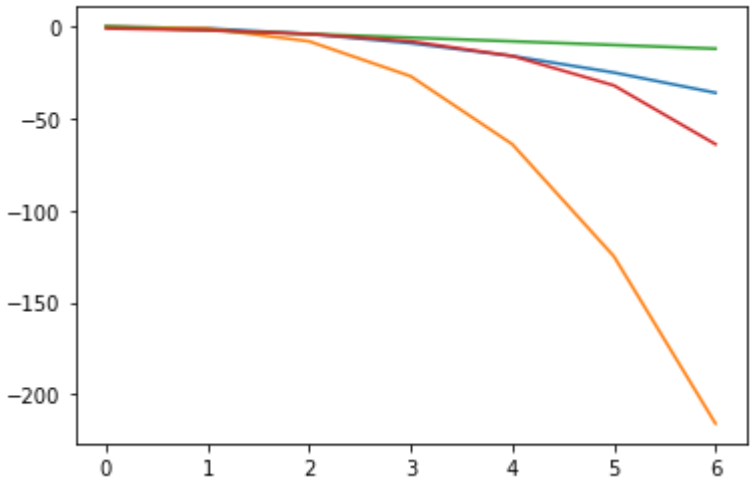
In [9]:

```
# You can write the same code in a simple way as follows:

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(25)
plt.plot(x, x**3 + 1)
plt.show()
```



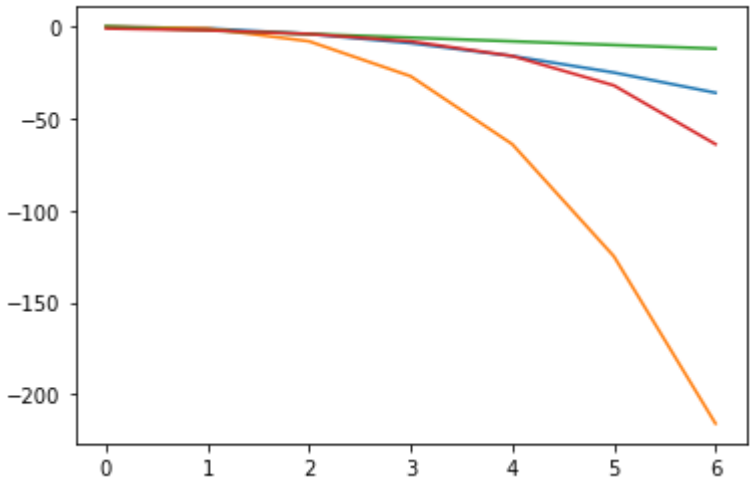
In [10]:

```
# It is possible to visualize multiple plots in the same output. Let's see how to show
# multiple curves in the same visualization. The following is a simple example:

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(7)
plt.plot(x, -x**2)
plt.plot(x, -x**3)
plt.plot(x, -2*x)
plt.plot(x, -2**x)
plt.show()
```



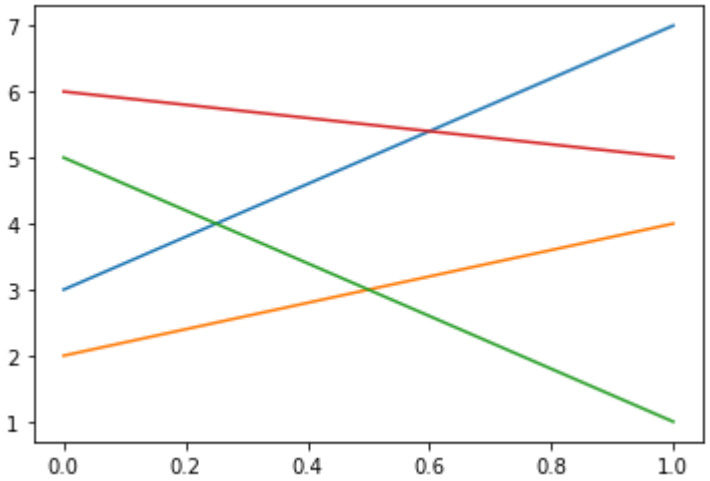
In [11]:

```
# You can write the same code in a simple way as follows:

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(7)
plt.plot(x, -x**2, x, -x**3,x, -2*x, x, -2**x)
plt.show()
```



In [12]:

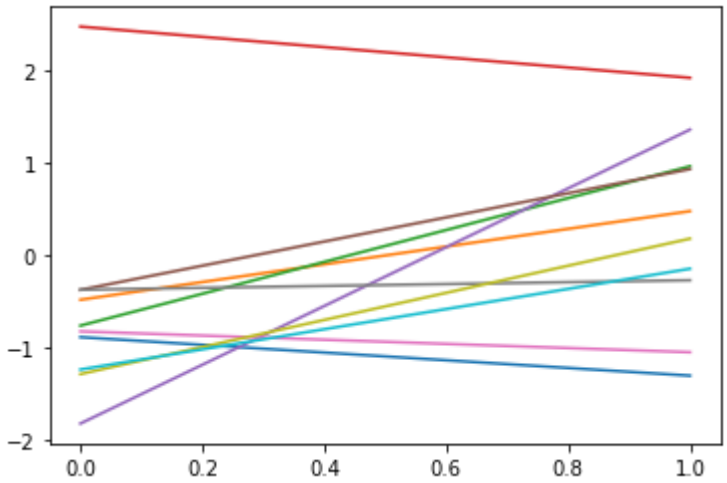
```
# Let’s see another example:

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
x = np.array([[3, 2, 5, 6], [7, 4, 1, 5]])
plt.plot(x)
plt.show()
```



In [13]:

```
# You can also create a multiline graph with random data as follows:

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
data = np.random.randn(2, 10)
print(data)
plt.plot([data[0], data[1]])
plt.show()
```

```
[[-0.88618633 -0.4825752  -0.76178326  2.4657351  -1.81893493 -0.37563486
  -0.82381565 -0.37427682 -1.2849678  -1.23569985]
 [-1.30195569  0.47379691  0.95790941  1.91320096  1.35392727  0.92775103
  -1.04879171 -0.27212365  0.17692582 -0.14772508]]
```



In [14]:

```
# In above example, we generated the data in a random way using the routine np.
# random.randn(). Since this routine will generate the random data, the output will be
# different every time we execute it. So, the output you will see will be different ever
# you execute the code.
```