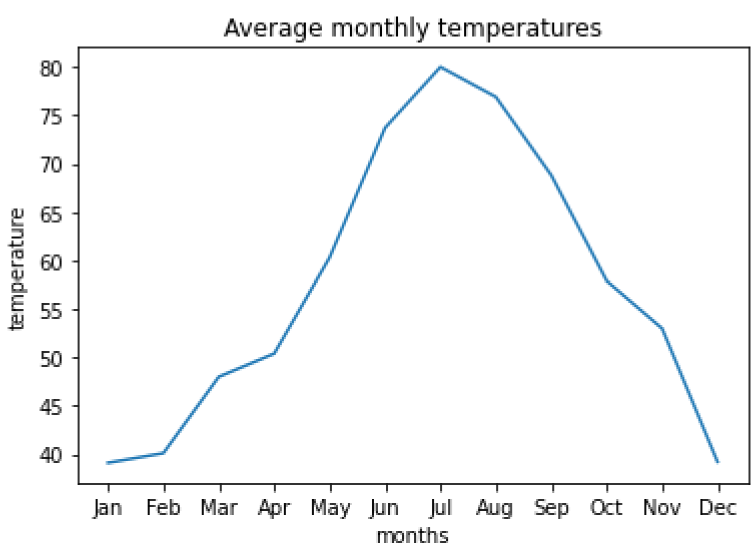
NumPy Essential Training: 2 MatPlotlib and Linear Algebra Capabilities

Chapter 1: Plotting with Matplotlib

In [1]:
```
%matplotlib notebook
%matplotlib inline
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

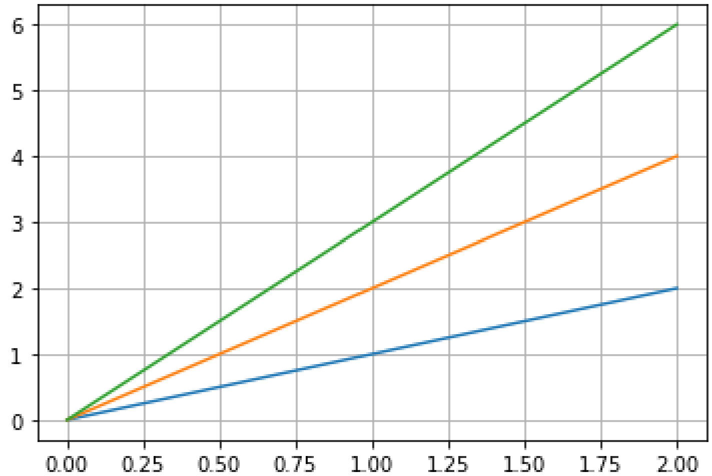
In [3]:
```
average_monthly_temperatures = [39.1, 40.1, 48.0, 50.4, 60.3, 73.7, 80.0, 76.9, 68.8, 5
months=['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
fig = plt.figure()
plt.plot(months,average_monthly_temperatures)
plt.title("Average monthly temperatures")
plt.xlabel("months")
plt.ylabel("temperature")
plt.show()
```



In [5]:
```
fig.savefig('average_monthly_temperatures.png')
```

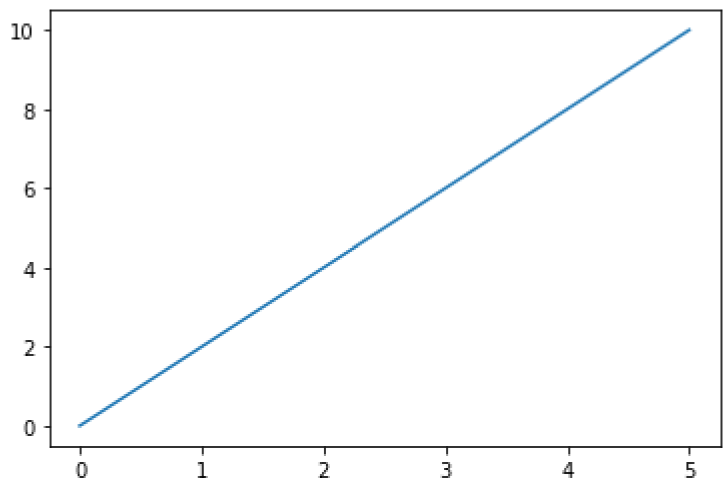
In [6]:
```
fig.savefig('average_monthly_temperatures.pdf')
```

In [8]:
```
x = np.arange(3)
plt.plot(x,x)
plt.plot(x,2*x)
plt.plot(x,3*x)
plt.grid(True)
plt.show()
```

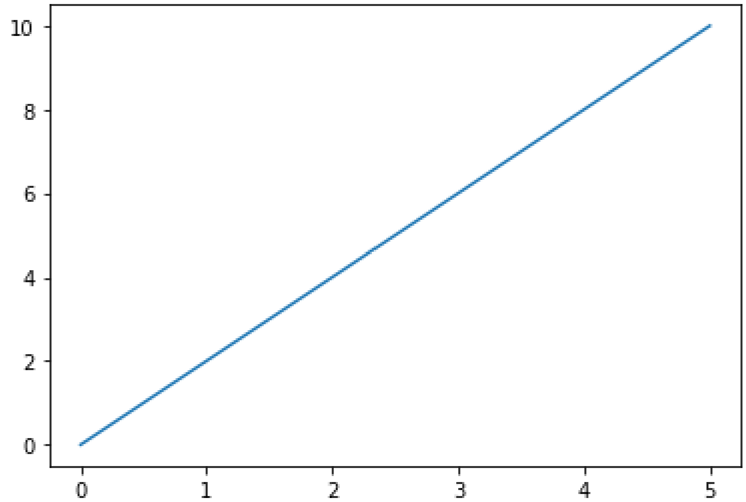


In [9]:
```
x = np.linspace(0,5,5)
y=2*x
plt.plot(x,y)
#plt.show()
```

Out[9]: [<matplotlib.lines.Line2D at 0x2077c9c8610>]

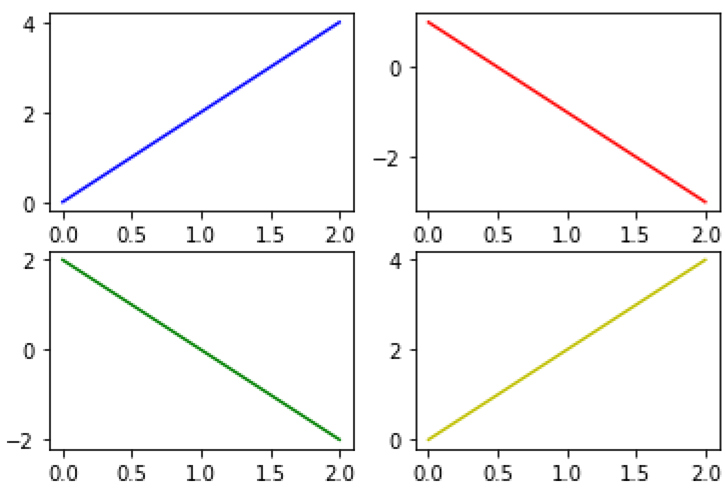


In [10]:
```
fig = plt.figure()
axes = fig.add_axes([0.1,0.1,0.8,0.8])
axes.plot(x,y)
plt.show()
```



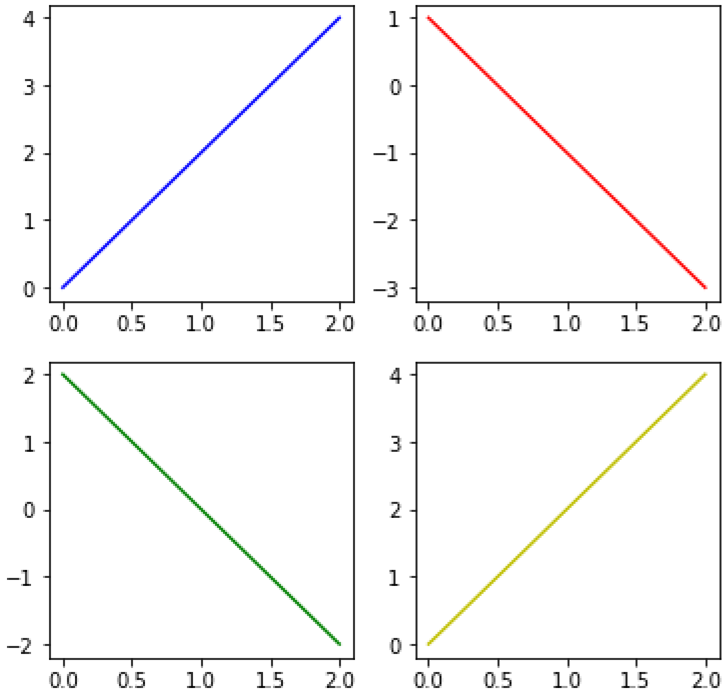
In [11]:

```
fig=plt.figure()
x=np.arange(3)
y=2*x
plt.subplot(2,2,1)
plt.plot(x,y,'b')

plt.subplot(2,2,2)
plt.plot(x,1-y,'r')

plt.subplot(2,2,3)
plt.plot(x,2-y,'g')

plt.subplot(2,2,4)
plt.plot(x,y,'y')

plt.show()
```



In [12]:

```
fig, axs = plt.subplots(2, 2, figsize=(6,6))
axs[0, 0].plot(x, y, 'b')
axs[0, 1].plot(x, 1-y, 'r')
axs[1, 0].plot(x, 2-y, 'g')
axs[1, 1].plot(x, y, 'y')
plt.show()
```

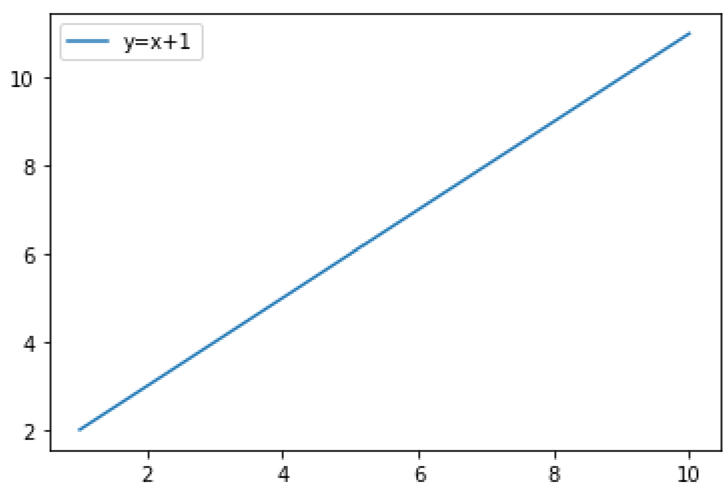


In [13]:

```
x = np.linspace(1,10)
first_line = plt.plot(x, x+1, label= 'y=x+1')
plt.legend()
```

Out[13]:

```
<matplotlib.legend.Legend at 0x2077c8ff4c0>
```

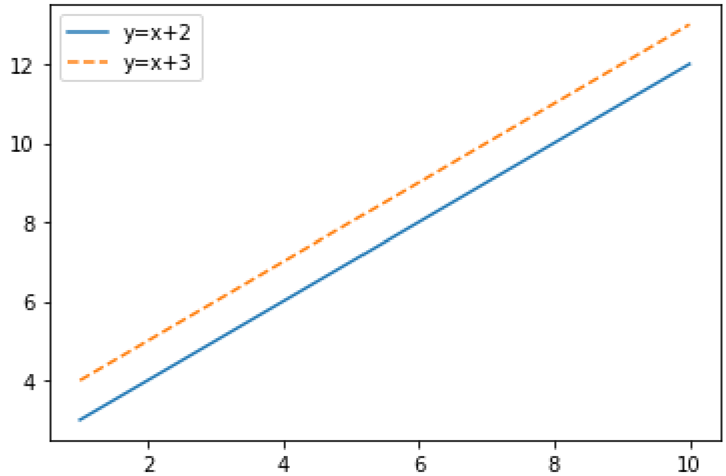


In [14]:

```
second_line, = plt.plot(x,x+2,linestyle='solid')
second_line.set_label('y=x+2')
third_line, = plt.plot(x,x+3,linestyle='dashed')
third_line.set_label('y=x+3')
plt.legend()
```

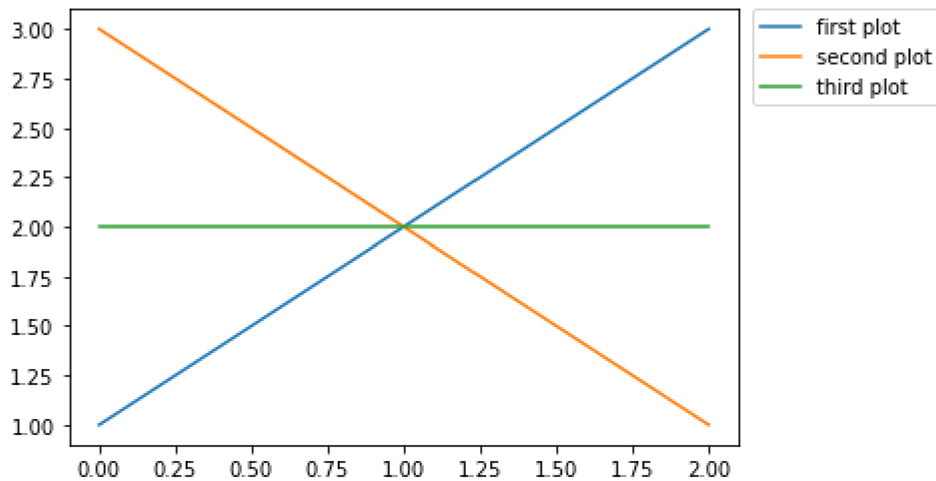
Out[14]:

```
<matplotlib.legend.Legend at 0x2077cb41100>
```



In [15]:

```
first_plot,=plt.plot([1,2,3],label='first plot')
second_plot,=plt.plot([3,2,1],label='second plot')
third_plot,=plt.plot([2,2,2],label='third plot')
plt.legend(bbox_to_anchor=(1.02, 1.0), borderaxespad=0);
```



Chapter 2: Matplot lib Styling and Advanced Plots

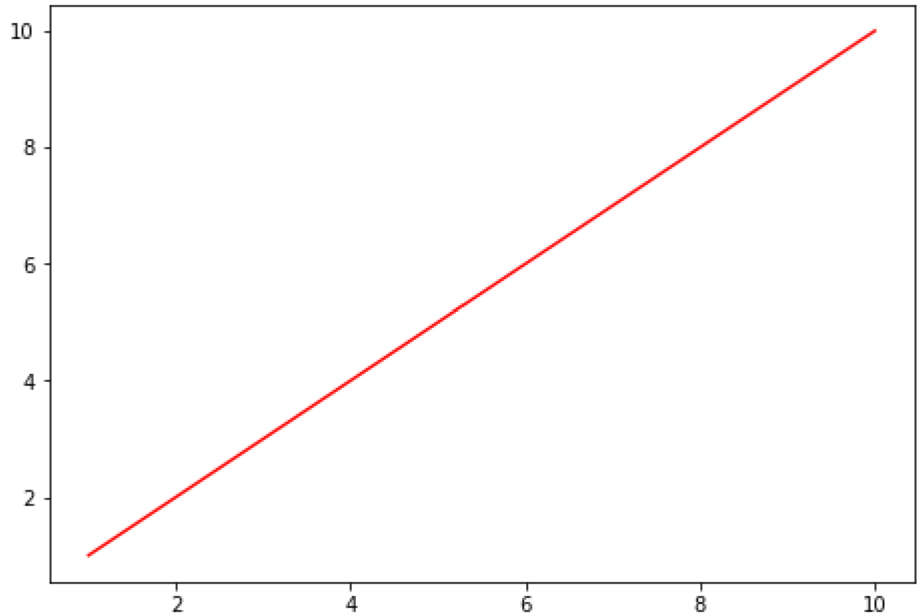
Color and styles

In [16]:

```
first_figure = plt.figure()
x = np.linspace(1, 10)
y = np.linspace(1, 10)
ax=first_figure.add_axes([0,0,1,1])
ax.plot(x,y, color='red')
```

Out[16]:

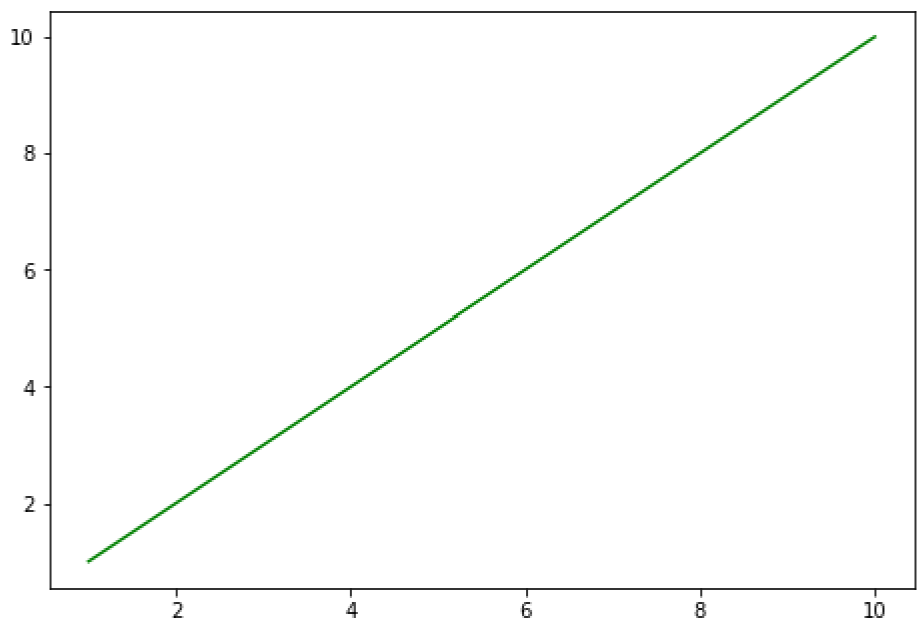
```
[<matplotlib.lines.Line2D at 0x2077bcad0d0>]
```



In [17]:

```
second_figure = plt.figure()
ax=second_figure.add_axes([0,0,1,1])
ax.plot(x,y, color='g')
```

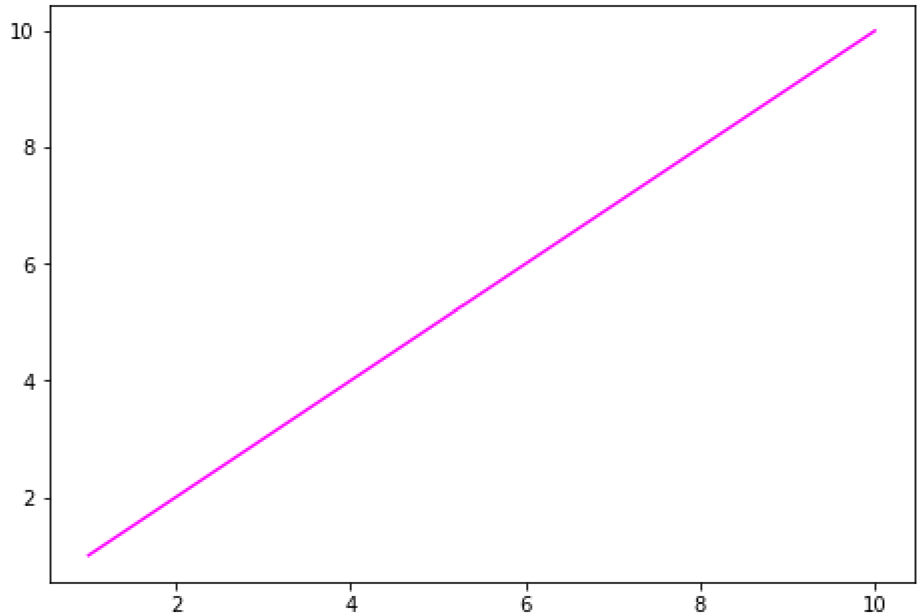
Out[17]: [<matplotlib.lines.Line2D at 0x2077c95b850>]



In [18]:

```
third_figure = plt.figure()
ax=third_figure.add_axes([0,0,1,1])
ax.plot(x,y, color='#FF00FF')
```

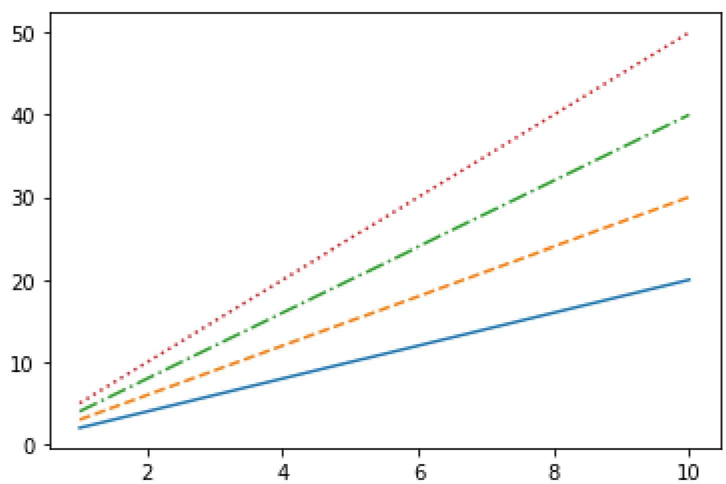
Out[18]: [<matplotlib.lines.Line2D at 0x2077dd79c40>]



In [19]:

```
plt.plot(x,2*x,linestyle='solid')
plt.plot(x,3*x,linestyle='dashed')
plt.plot(x,4*x,linestyle='dashdot')
plt.plot(x,5*x,linestyle='dotted')
```

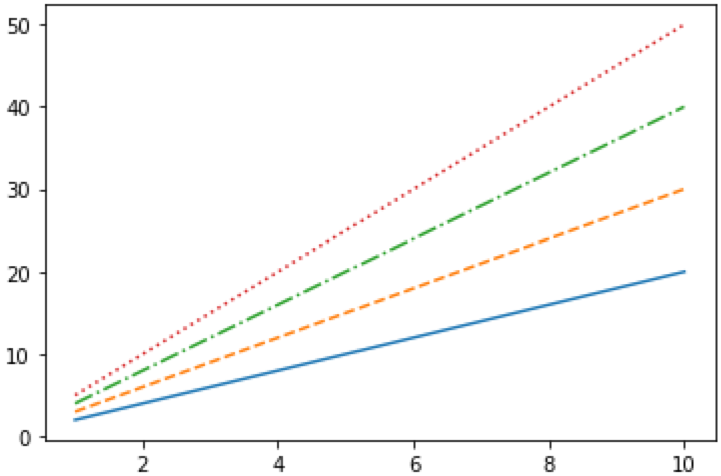
Out[19]: [<matplotlib.lines.Line2D at 0x2077ca9fb50>]



In [20]:

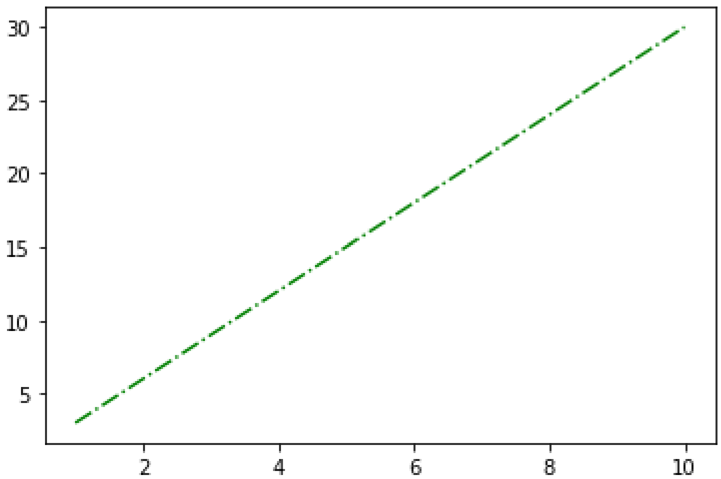
```
plt.plot(x,2*x,linestyle='-')
plt.plot(x,3*x,linestyle='--')
plt.plot(x,4*x,linestyle='-.')
plt.plot(x,5*x,linestyle=':')
```

Out[20]: [<matplotlib.lines.Line2D at 0x2077ddb2e80>]



In [21]:

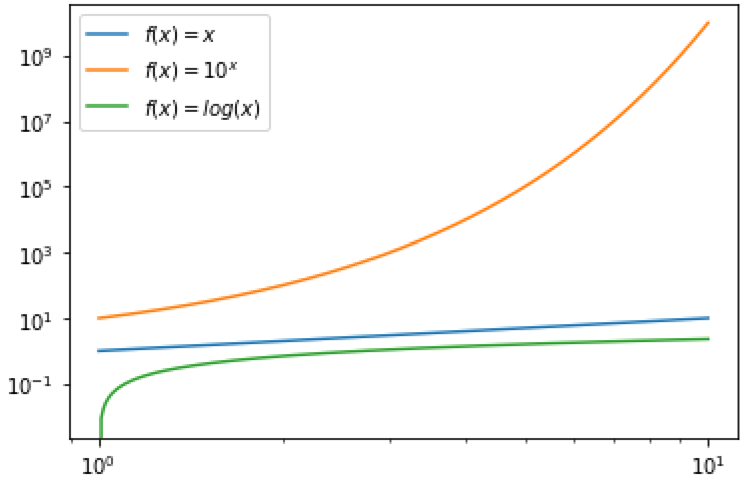
```
plt.plot(x, 3*x ,'-.g');
```



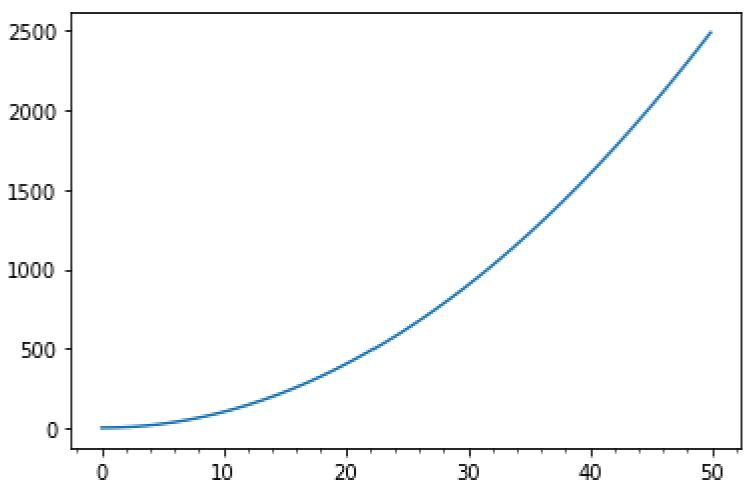
Advanced matplotlib commands

In [22]:

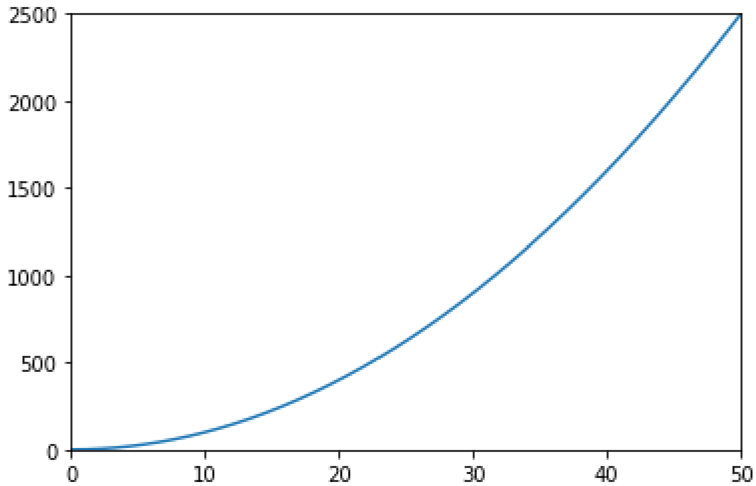
```
x = np.linspace(1, 10, 1024)
plt.xscale('log')
plt.yscale('log')

plt.plot(x, x, label ='$f(x)=x$')
plt.plot(x, 10**x, label ='$f(x)=10^x$')
plt.plot(x, np.log(x),label ='$f(x)=log(x)$')

plt.legend()
plt.show()
```



In [23]:
```
from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)

x = np.arange(0.0, 50.0, 0.1)
y = x**2

fig, ax = plt.subplots()
ax.plot(x,y)

ax.xaxis.set_major_locator(MultipleLocator(10))
ax.xaxis.set_major_formatter('{x:.0f}')

ax.xaxis.set_minor_locator(MultipleLocator(2))

plt.show()
```

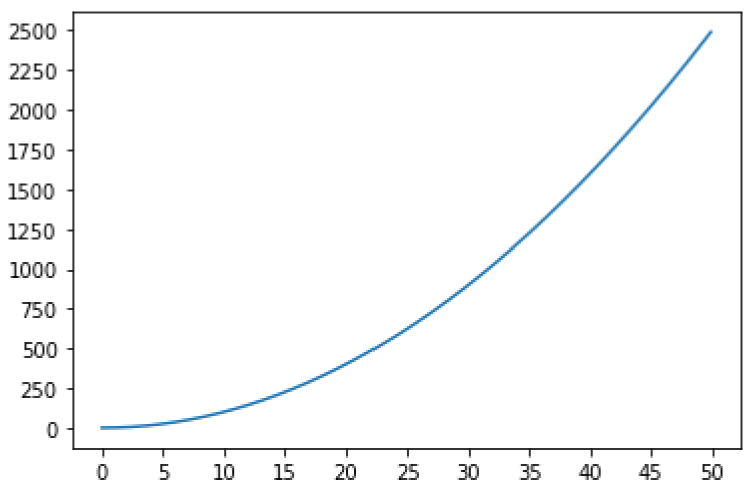


In [24]:
```
x = np.arange(0.0, 50.0, 0.1)
y = x**2
fig, ax = plt.subplots()
ax.plot(x,y)

ax.set_xlim([0, 50])
ax.set_ylim([0, 2500])

plt.show()
```



In [25]:

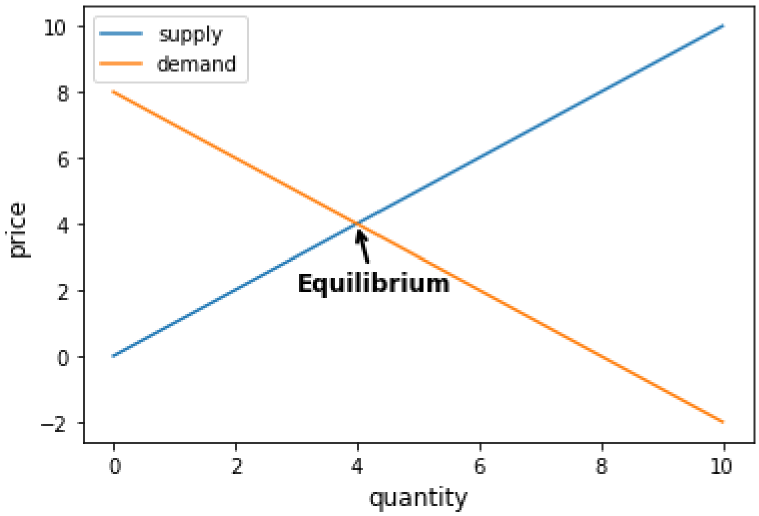
```
x = np.arange(0.0, 50.0, 0.1)
y = x**2

fig, ax = plt.subplots()
ax.plot(x,y)

ax.set_xticks([0,5,10,15,20,25,30,35,40,45,50])
ax.set_yticks([0,250,500,750,1000,1250,1500,1750,2000,2250,2500])

plt.show()
```



In [26]:

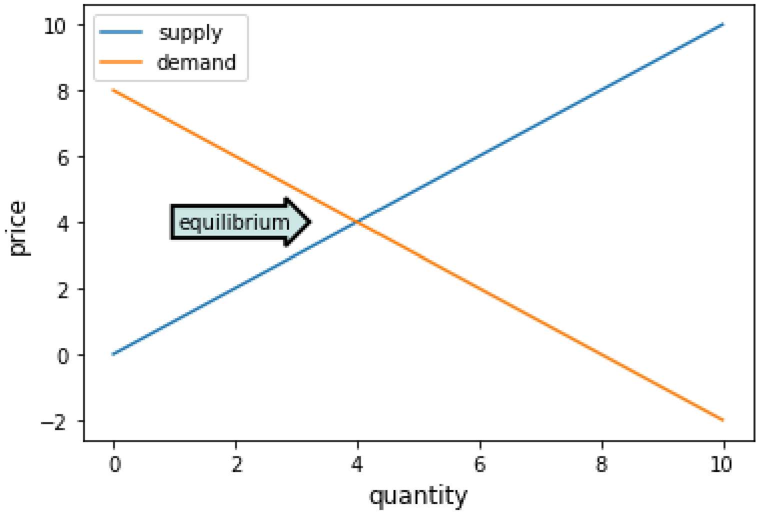
```
x = np.linspace(0, 10)

y1 = x
y2 = 8-x


fig, ax = plt.subplots()
plt.plot(x,y1,label='supply')
plt.plot(x,y2,label='demand')

ax.annotate("Equilibrium", xy=(4,4), xytext=(3,2), \
             fontsize=12, fontweight='semibold',\
             arrowprops=dict(linewidth=2, arrowstyle="->"))
```

```
plt.xlabel('quantity',fontsize=12)
plt.ylabel('price',fontsize=12)


plt.legend()
plt.show()
```



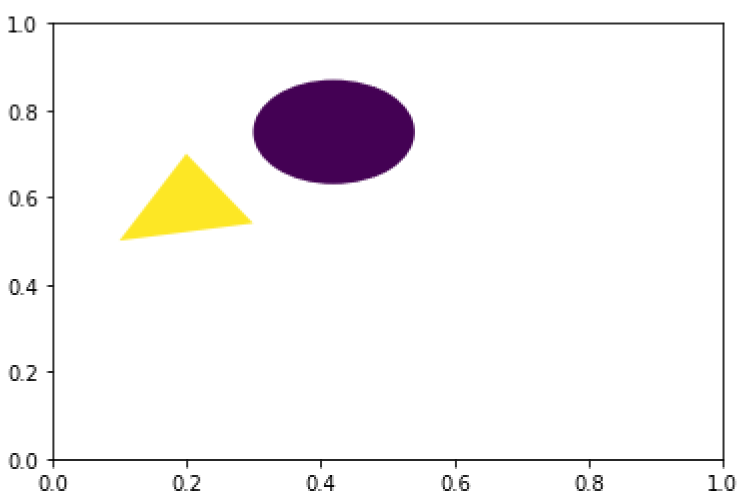
In [27]:

```
x = np.linspace(0, 10)
y1 = x
y2 = 8-x

# Plot the data
fig, ax = plt.subplots()
plt.plot(x,y1,label='supply')
plt.plot(x,y2,label='demand')

# Annotate the equilibrium point with arrow and text
bbox_props = dict(boxstyle="rarrow", fc=(0.8, 0.9, 0.9), lw=2)
t = ax.text(2,4, "equilibrium", ha="center", va="center", rotation=0,
            size=10,bbox=bbox_props)

# Label the axes
plt.xlabel('quantity',fontsize=12)
plt.ylabel('price',fontsize=12)


plt.legend()
plt.show()
```



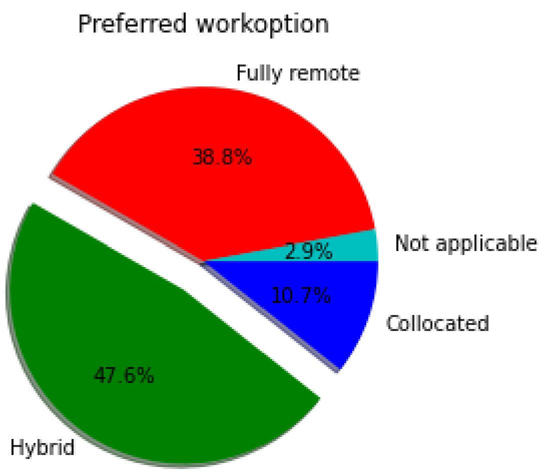
In [28]:

```
from matplotlib.patches import Circle, Polygon
from matplotlib.collections import PatchCollection


fig, ax = plt.subplots()
patches = []

# draw circle and triangle
circle = Circle((.42,.75),0.12)
triangle = Polygon([[.1,.5],[.2,.7],[.3,.54]], True)

patches += [circle,triangle]

# Draw the patches
colors = 100*np.random.rand(len(patches)) # set random colors
p = PatchCollection(patches)
p.set_array(np.array(colors))
ax.add_collection(p)

# Show the figure
plt.show()
```

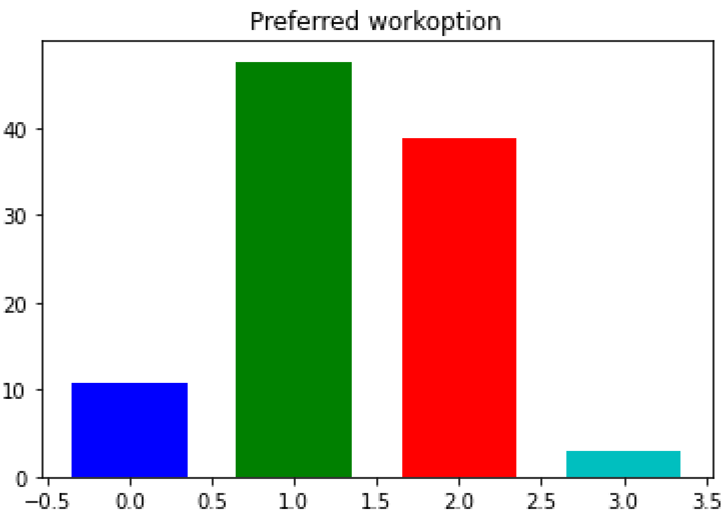


Creating pie charts and bar charts

In [29]:
```
preferred_workoption = [10.7, 47.6, 38.8, 2.9]

colors = ['b', 'g', 'r', 'c']

labels = ['Collocated', 'Hybrid', 'Fully remote', 'Not applicable']

explode = (0, 0.2, 0, 0)


plt.pie(preferred_workoption, colors=colors, labels=labels,

        explode=explode, autopct='%1.1f%%',

        counterclock=False, shadow=True)

plt.title('Preferred workoption')


plt.show()
```



In [30]:
```
preferred_workoption = [10.7, 47.6, 38.8, 2.9]

colors = ['b', 'g', 'r', 'c']

labels = ['Collocated', 'Hybrid', 'Fully remote', 'Not applicable']

widths= [0.7, 0.7, 0.7, 0.7]

plt.bar(range(0, 4), preferred_workoption, width=widths, color=colors, align='center')

plt.title('Preferred workoption')


plt.show()
```



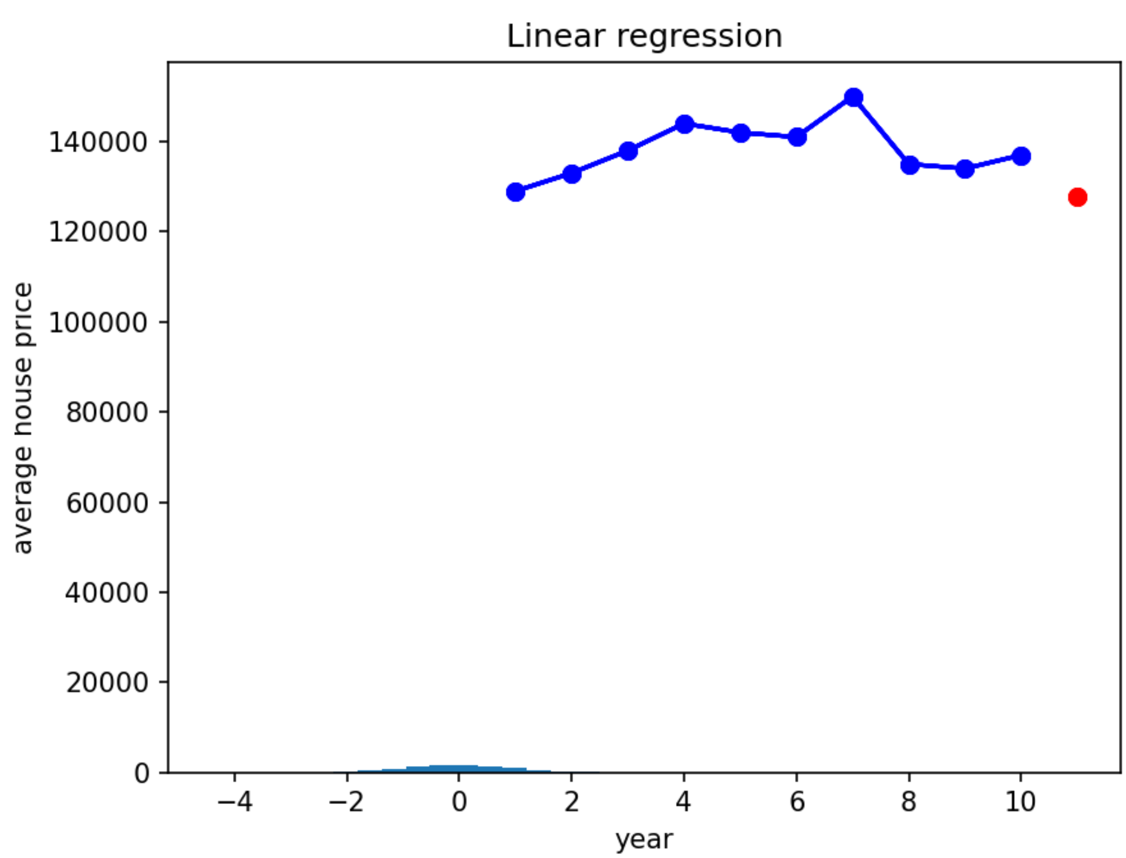
Advanced Plots

In [41]:

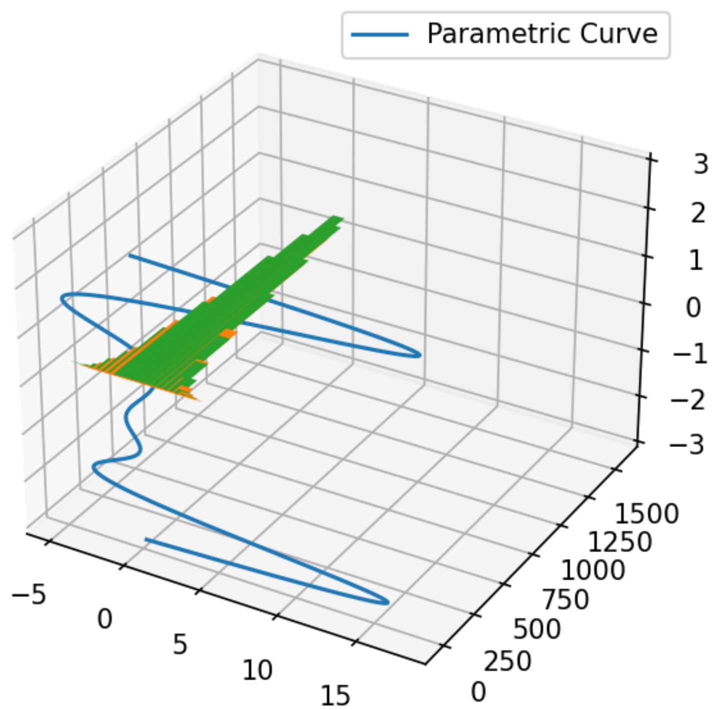
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib notebook
```

In [42]:

```
X = np.random.randn(10000)
plt.hist(X, bins = 20)
plt.show()
```



In [38]:
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
theta = np.linspace(-3 * np.pi, 3 * np.pi, 200)
z = np.linspace(-3, 3, 200)
r = z**3 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)

ax.plot(x, y, z, label='Parametric Curve')
ax.legend()
plt.show()
```



Chapter 3: From Beginner to advance Numpy

In [43]:
```
from __future__ import print_function
import numpy as np
```

In [44]:
```
numbers=np.arange(1,11)
numbers
```

Out[44]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [45]:
```
np.sin(numbers)
```

Out[45]:
```
array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.95892427,
       -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849, -0.54402111])
```

In [46]:
```
np.log(numbers)
```

Out[46]:
```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947, 1.94591015, 2.07944154, 2.19722458, 2.30258509])
```

In [47]:
```
# creating numpy array
integers = np.arange(1, 101)
print("integers     :", *integers)

# creating own function
def modulo(val):
  return (val % 10)

# adding into numpy
mod_10=np.frompyfunc(modulo, 1, 1)

# using function over numpy array
mod_integers=mod_10(integers)
print("mod_integers :", *mod_integers)
```

```
integers     : 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 2
8 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 8
7 88 89 90 91 92 93 94 95 96 97 98 99 100
mod_integers : 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
```

Introducing Strides

In [48]:
```
numbers = np.arange(10, dtype = np.int8)
numbers
```

Out[48]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int8)
```

In [49]:
```
numbers.strides
```

Out[49]:
```
(1,)
```

In [50]:
```
numbers.shape = 2,5
numbers
```

Out[50]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]], dtype=int8)
```

In [51]:
```
numbers.strides
```

Out[51]:
```
(5, 1)
```

In [52]:

```
first_array = np.zeros((100000,))
first_array
```

Out[52]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [53]:
```
second_array = np.zeros((100000 * 100, ))[::100]
second_array
```

Out[53]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [54]:
```
first_array.shape
```

Out[54]:
```
(100000,)
```

In [55]:
```
second_array.shape
```

Out[55]:
```
(100000,)
```

In [56]:
```
first_array.strides
```

Out[56]:
```
(8,)
```

In [57]:
```
second_array.strides
```

Out[57]:
```
(800,)
```

In [58]:
```
%timeit first_array.sum()
```

```
56.9 µs ± 5.16 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

In [59]:
```
%timeit second_array.sum()
```

```
534 µs ± 44.3 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Structures Arrays

In [61]:
```
student_records = np.array([('Lazaro','Oneal', '0526993', 2009, 2.33), ('Dorie','Salina
       dtype=[('name', (np.str_, 10)),('surname', (np.str_, 10)), ('id', (np.str_,7)),(
student_records
```

Out[61]:
```
array([('Lazaro', 'Oneal', '0526993', 2009, 2.33),
       ('Dorie', 'Salinas', '0710325', 2006, 2.26),
       ('Mathilde', 'Hooper', '0496813', 2000, 2.56),
       ('Nell', 'Gomez', '0740631', 2003, 2.22),
       ('Lachelle', 'Jordan', '0490888', 2003, 2.13),
       ('Claud', 'Waller', '0922492', 2004, 3.6 ),
       ('Bob', 'Steele', '0264843', 2002, 2.79),
       ('Zelma', 'Welch', '0885463', 2007, 3.69)],
```

```
      dtype=[('name', '<U10'), ('surname', '<U10'), ('id', '<U7'), ('graduation_year',
'<i4'), ('gpa', '<f8')])
```

In [62]:

```
student_records[['id','graduation_year']]
```

Out[62]:

```
array([('0526993', 2009), ('0710325', 2006), ('0496813', 2000),
       ('0740631', 2003), ('0490888', 2003), ('0922492', 2004),
       ('0264843', 2002), ('0885463', 2007)],
      dtype={'names': ['id', 'graduation_year'], 'formats': ['<U7', '<i4'], 'offsets':
[80, 108], 'itemsize': 120})
```

In [63]:

```
students_sorted_by_surname = np.sort(student_records, order='surname')
print('Students sorted according to the surname :\n', students_sorted_by_surname)
```

```
Students sorted according to the surname :
 [('Nell', 'Gomez', '0740631', 2003, 2.22)
 ('Mathilde', 'Hooper', '0496813', 2000, 2.56)
 ('Lachelle', 'Jordan', '0490888', 2003, 2.13)
 ('Lazaro', 'Oneal', '0526993', 2009, 2.33)
 ('Dorie', 'Salinas', '0710325', 2006, 2.26)
 ('Bob', 'Steele', '0264843', 2002, 2.79)
 ('Claud', 'Waller', '0922492', 2004, 3.6 )
 ('Zelma', 'Welch', '0885463', 2007, 3.69)]
```

In [64]:

```
students_sorted_by_grad_year = np.sort(student_records, order='graduation_year')
print('Students sorted according to the graduation year :\n', students_sorted_by_grad_y
```

```
Students sorted according to the graduation year :
 [('Mathilde', 'Hooper', '0496813', 2000, 2.56)
 ('Bob', 'Steele', '0264843', 2002, 2.79)
 ('Lachelle', 'Jordan', '0490888', 2003, 2.13)
 ('Nell', 'Gomez', '0740631', 2003, 2.22)
 ('Claud', 'Waller', '0922492', 2004, 3.6 )
 ('Dorie', 'Salinas', '0710325', 2006, 2.26)
 ('Zelma', 'Welch', '0885463', 2007, 3.69)
 ('Lazaro', 'Oneal', '0526993', 2009, 2.33)]
```

Date and time in Numpy

In [65]:

```
np.datetime64('2022-03-01')
```

Out[65]:

```
numpy.datetime64('2022-03-01')
```

In [ ]:

```
np.datetime64('2022-03')
```

In [66]:

```
print('Number of weekdays in 2022:')
print(np.busday_count('2022','2023'))
```

```
Number of weekdays in 2022:
260
```

In [67]:

```
print('Number of weekdays in June 2022:')
np.busday_count('2022-06', '2022-07')
```

```
Number of weekdays in June 2022:
```

Out[67]:
```
22
```

In [68]:
```
np.is_busday(np.datetime64('2022-06-05'))
```

Out[68]:
```
False
```

Chapter 4 : Linear Algebra in Numpy

Linear algebra capabilities in NumPy

In [69]:
```
first_array = np.arange(16).reshape(4,4)
first_array
```

Out[69]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [70]:
```
first_matrix = np.matrix(first_array)
first_matrix
```

Out[70]:
```
matrix([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15]])
```

In [71]:
```
second_matrix = np.matrix(np.identity(4))
second_matrix
```

Out[71]:
```
matrix([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]])
```

In [72]:
```
matrix_a=np.random.randint(5,size=(2,3))
matrix_a
```

Out[72]:
```
array([[4, 0, 4],
       [0, 4, 3]])
```

In [73]:
```
matrix_b=np.random.randint(5,size=(3,2))
matrix_b
```

Out[73]:
```
array([[1, 4],
       [0, 0],
       [1, 3]])
```

In [74]:
```
np.matmul(matrix_a,matrix_b)
```

Out[74]:
```
array([[ 8, 28],
       [ 3,  9]])
```

In [75]:
```
matrix_c=np.matrix("0 1 2;1 0 3;4 -3 8")
matrix_c
```

Out[75]:
```
matrix([[ 0,  1,  2],
        [ 1,  0,  3],
        [ 4, -3,  8]])
```

In [76]:
```
inverse = np.linalg.inv(matrix_c)
inverse
```

Out[76]:
```
matrix([[-4.5,  7. , -1.5],
        [-2. ,  4. , -1. ],
        [ 1.5, -2. ,  0.5]])
```

In [78]:
```
print(matrix_c*inverse)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [79]:
```
A =np.mat("1 -2 1;0 2 -8;-4 5 9")
A
```

Out[79]:
```
matrix([[ 1, -2,  1],
        [ 0,  2, -8],
        [-4,  5,  9]])
```

In [80]:
```
b = np.array([0, 16, -18])
b
```

Out[80]:
```
array([  0,  16, -18])
```

In [81]:
```
x = np.linalg.solve(A, b)
print("Solution", x)
```

```
Solution [58. 32.  6.]
```

Decomposition

In [82]:
```
first_matrix=np.matrix([[4,8],[10,14]])
print("Matrix:\n",first_matrix)
```

```
Matrix:
 [[ 4  8]
 [10 14]]
```

In [83]:
```
eigenvalues, eigenvectors = np.linalg.eig(first_matrix)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:", eigenvectors)
```

```
Eigenvalues: [-1.24695077 19.24695077]
Eigenvectors: [[-0.83619408 -0.46462222]
```

```
 [ 0.54843365 -0.885509  ]]
```

In [84]:

```
eigenvalues= np.linalg.eigvals(first_matrix)
print("Eigenvalues:", eigenvalues)
```

```
Eigenvalues: [-1.24695077 19.24695077]
```

In [85]:

```
A = np.mat("3 1 4;1 5 9;2 6 5")
print("A\n", A)

U, Sigma, V = np.linalg.svd(A, full_matrices=False)

print("U: ",U)
print("Sigma : ",Sigma)
print("V : ", V)
```

```
A
 [[3 1 4]
 [1 5 9]
 [2 6 5]]
U:  [[-0.32463251  0.79898436  0.50619929]
 [-0.75307473  0.1054674  -0.64942672]
 [-0.57226932 -0.59203093  0.56745679]]
Sigma :  [13.58235799  2.84547726  2.32869289]
V :  [[-0.21141476 -0.55392606 -0.80527617]
 [ 0.46331722 -0.78224635  0.41644663]
 [ 0.86060499  0.28505536 -0.42202191]]
```

In [86]:

```
print("Product\n", U * np.diag(Sigma) * V)
```

```
Product
 [[3. 1. 4.]
 [1. 5. 9.]
 [2. 6. 5.]]
```

M=Q*R

In [87]:

```
A
```

Out[87]:

```
matrix([[3, 1, 4],
        [1, 5, 9],
        [2, 6, 5]])
```

In [88]:

```
b = np.array([1,2,3]).reshape(3,1)
q, r = np.linalg.qr(A)
x = np.dot(np.linalg.inv(r), np.dot(q.T, b))
x
```

Out[88]:

```
matrix([[ 0.26666667],
        [ 0.46666667],
        [-0.06666667]])
```

In [89]:

```
np.linalg.solve(A,b)
```

Out[89]:
```
array([[ 0.26666667],
       [ 0.46666667],
       [-0.06666667]])
```

Polynomial mathematics

In [90]:
```
import numpy as np
from numpy.polynomial import polynomial
```

In [91]:
```
first_polynomial = np.polynomial.Polynomial([2, -3, 1])
first_polynomial
```

Out[91]: $x \mapsto 2.0 - 3.0\,x + 1.0\,x^2$

In [92]:
```
second_polynomial = np.polynomial.Polynomial.fromroots([1, 2])
second_polynomial
```

Out[92]: $x \mapsto 2.0 - 3.0\,x + 1.0\,x^2$

In [93]:
```
first_polynomial.roots()
```

Out[93]:
```
array([1., 2.])
```

In [94]:
```
second_polynomial.roots()
```

Out[94]:
```
array([1., 2.])
```

$y=x^4+2x^3+3x^2+4x+5$, x=1

y=?

In [95]:
```
np.polyval([5,4,3,2,1], 1)
```

Out[95]:
```
15
```

In [96]:
```
third_polynomial = np.polynomial.Polynomial([1,2,3,4,5])
third_polynomial
```

Out[96]: $x \mapsto 1.0 + 2.0\,x + 3.0\,x^2 + 4.0\,x^3 + 5.0\,x^4$

In [97]:
```
integral=third_polynomial.integ()
integral
```

Out[97]: $x \mapsto 0.0 + 1.0\,x + 1.0\,x^2 + 1.0\,x^3 + 1.0\,x^4 + 1.0\,x^5$

In [98]:
```
integral.deriv()
```

Out[98]: $x \mapsto 1.0 + 2.0\,x + 3.0\,x^2 + 4.0\,x^3 + 5.0\,x^4$

In [99]:
```
derivative=third_polynomial.deriv()
derivative
```

Out[99]: $x \mapsto 2.0 + 6.0\,x + 12.0\,x^2 + 20.0\,x^3$

Application Linear Regression

House market

- Input: Price data from 2012 - 2021
- Output: Avarage house market 2022?
- Asume Relationship - squared
- y=ax^2+bx+c

In [111…
```
import numpy as np
import matplotlib.pyplot as plt
```

In [112…
```
year = np.arange(1,11)
price = np.array([129000, 133000, 138000, 144000, 142000, 141000, 150000, 135000, 13400
year
```

Out[112…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [113…
```
a, b, c = np.polyfit(year, price, 2)
print ("a:",a)
print ("b:",b)
print ("c:",c)
```

```
a: -594.6969696969702
b: 7032.575757575754
c: 122516.66666666669
```

In [114…
```
print("Estimated price for 2022:",a*11**2 + b*11 + c )
```

```
Estimated price for 2022: 127916.66666666658
```

In [118…
```
plt.plot(year,price, color = 'blue')
plt.scatter(year,price, color = 'blue')
plt.scatter(11, a*11**2 + b*11 + c ,color='red')
plt.title('Linear regression')
plt.xlabel('year')
plt.ylabel('average house price')
```

Out[118…

```
Text(17.583333333333336, 0.5, 'average house price')
```

In [ ]: