```
b = 2
print(b)
```

```
2
```

```
string1 = "This is python programming"
print(string1[1:3])
```

```
hi
```

```
n=input("enter the number")
```

```
enter the number5
```

```
print(n)
```

```
5
```

A **tuple** is an ordered, immutable collection of items. Tuples are similar to lists, but once created, their elements cannot be changed, added, or removed.

```
tup=("hello",1,2,"Apple")
tup
```

```
('hello', 1, 2, 'Apple')
```

```
another_tuple = 1, 2, 3
another_tuple
```

```
(1, 2, 3)
```

```
print(tup[3])  # Output: Apple
```

```
Apple
```

```
#  immutable
#  tup[2]="ali"
```

In Python, a list is an ordered, mutable collection of items. Lists are one of the most commonly used data structures in Python.

```
l1=[1,2,"hello"]
l1
```

```
[1, 2, 'hello']
```

```
l1[2]
```

```
'hello'
```

```
#  mutable
l1[2]="ali"
l1[2]
```

```
'ali'
```

```
for item in l1:
    print(item)
```

```
1
2
ali
```

```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
print(fruits)  # Output: ['apple', 'banana', 'cherry', 'orange']
fruits.remove("orange")
print(fruits)
```

```
['apple', 'banana', 'cherry', 'orange']
['apple', 'banana', 'cherry']
```

In Python, a **dictionary** is an unordered collection of key-value pairs. Each key in a dictionary is unique, and it is used to access its corresponding value.

```
my_dict = {
    "name": "Oli",
    "age": 30,
    "profession": "Teacher"
}
```

```
# You can access values using their keys:

print(my_dict["name"])  # Output: Oli
```

```
Oli
```

```
my_dict["country"] = "Bangladesh"  # Add new key-value pair
my_dict["age"] = 31                # Modify existing value
```

```
del my_dict["profession"]  # Remove a key-value pair
```

```
for key, value in my_dict.items():
    print(key, ":", value)
```

```
name : Oli
age : 31
country : Bangladesh
```

```
student = {
    "id": 12345,
    "name": "Alice",
    "grades": [85, 90, 92]
}
print(student["grades"])  # Output: [85, 90, 92]
```

```
[85, 90, 92]
```

In Python, a **set** is an unordered collection of unique elements. It is useful when you only want to store one instance of each item, and it provides efficient operations for checking membership or eliminating duplicates from a sequence.

```
my_set = {1, 2, 3, 4}
another_set = set([3, 4, 5, 6])
```

```
# Add elements:

my_set.add(3)


# Remove elements:

my_set.remove(3)


# Check membership:

print(3 in my_set)  # Output: True or False
```

```
False
```

```
# Set union (combine two sets):

union_set = my_set | another_set


# Set intersection (common elements):

intersection_set = my_set & another_set


# Difference (elements in one set but not the other):

difference_set = my_set - another_set

print(union_set, intersection_set,difference_set)
```

```
{1, 2, 3, 4, 5, 6} {4} {1, 2}
```

```
numbers = {1, 2, 3, 3, 4}
print(numbers)  # Output: {1, 2, 3, 4} (duplicates removed)

a = {1, 2, 3}
b = {3, 4, 5}
print(a | b)  # Union: {1, 2, 3, 4, 5}
print(a & b)  # Intersection: {3}
```

```
{1, 2, 3, 4}
{1, 2, 3, 4, 5}
{3}
```

Control flow IF else

```
a=10
b=20
if(a>b):
  print("a is greater")
else:
  print("b is greater")
```

```
b is greater
```

```
cmd="start"
match cmd:
  case "start":
    print("start the game")
  case "stop":
    print("stop the game")
```

```
start the game
```

for loop

```
# for variable in iterable:
    # code block

for i in range(10):
  print(i)
```

```
0
1
2
3
4
5
6
7
8
```

```
9
```

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

```
apple
banana
cherry
```

while Loop

```
count = 1
while count <= 5:
    print(count)
    count += 1  # Must update condition to avoid infinite loop
```

```
1
2
3
4
5
```

Functions and modules

```
def hello():
  print("hello")
hello()
```

```
hello
```

```
def add(a, b):
    return a + b

result = add(5, 3)
print(result)  # Output: 8
```

```
8
```

```
def greet(name="Guest"):
    print(f"Hello, {name}!")

greet("Oli")   # Output: Hello, Oli!
greet()        # Output: Hello, Guest!
```

```
Hello, Oli!
Hello, Guest!
```

an array is a data structure that holds a collection of items (usually of the same type) in a single variable.

```
import array

arr = array.array('i', [1, 2, 3, 4])  # 'i' means integer type
print(arr)
```

```
array('i', [1, 2, 3, 4])
```

```
import numpy as np
a= np.array([1,2,3,4])
a
```

```
array([1, 2, 3, 4])
```

```
from cmath import sin
a=sin(90)
print(a)
```

```
(0.8939966636005579-0j)
```

```
import math

result = math.sin(math.radians(90))
print(result)  # Output: 1.0
```

```
1.0
```

```
import math
a=math.sqrt(18)
print(a)
print(format(a, ".3f"))  # Output: 4.243
print(f"{a:.3f}")  # Output: 4.243
print(round(a, 3))  # Output: 4.243
```

```
4.242640687119285
4.243
4.243
4.243
```

Error handling

```
a=-2
try:
  if a<0:
    raise ValueError("Cannot find the squareroot of negative number")
  b=math.sqrt(a)
  print(b)
except ValueError as e:
  print(e)
```

```
Cannot find the squareroot of negative number
```

```
try:
  c = 10 / 0 # This will raise a ZeroDivisionError
  raise ValueError("Cannot divide by zero") # This won't execute because of the above error
except ZeroDivisionError as e:
  print("Caught a ZeroDivisionError:", e)
except ValueError as e:
  print("Caught a ValueError:", e)
finally:
  print("The error is of another source or handled completely.")
```

```
Caught a ZeroDivisionError: division by zero
The error is of another source or handled completely.
```

Numerical and scientific computations

```
a=np.array([[1,2,3],[5,6,7]])
a
```

```
array([[1, 2, 3],
       [5, 6, 7]])
```

```
b=np.array([[23,7,3],[8,6,9]])
b
```

```
array([[23,  7,  3],
       [ 8,  6,  9]])
```

```
c=np.array([[4,5],[6,7],[9,0]])
c
```

```
array([[4, 5],
       [6, 7],
       [9, 0]])
```

```
d=np.dot(a,c)
d
```

```
array([[ 43,  19],
       [119,  67]])
```

```
e=a@c
e
```

```
array([[ 43,  19],
       [119,  67]])
```

```
f=a+b
f
```

```
array([[24,  9,  6],
       [13, 12, 16]])
```

```
g=a*2
g
```

```
array([[ 2,  4,  6],
       [10, 12, 14]])
```

```
h=a+c.T
h
```

```
array([[ 5,  8, 12],
       [10, 13,  7]])
```

```
import numpy as np
a = np.array([1, 2, 3])
b = 5 # Scalar
result = a + b
print(result)
```

```
[6 7 8]
```

```
a = np.array([[1, 2, 3],
[4, 5, 6]])
b = np.array([10, 20, 30]) # Shape (3,)
result = a + b
print(result)
```

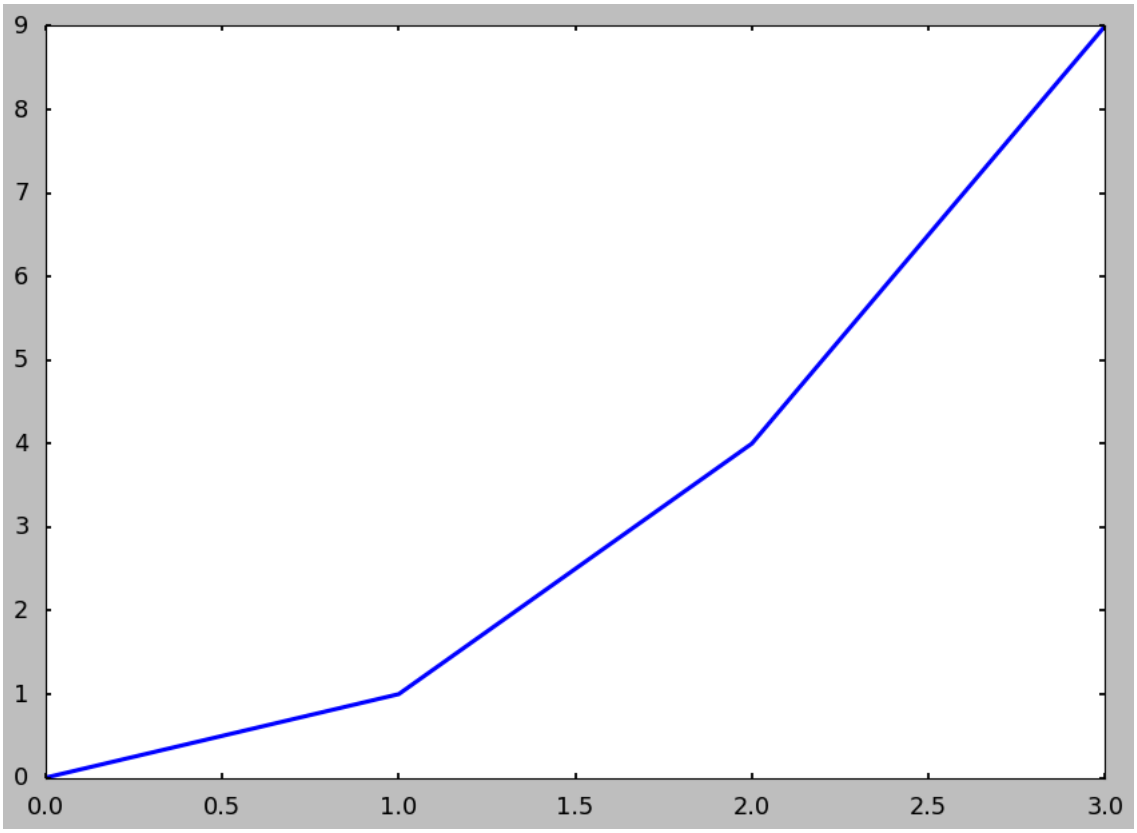
```
[[11 22 33]
 [14 25 36]]
```

```
a = np.array([[1, 2, 3],
[4, 5, 6]])
b = np.array([[10],
[20]]) # Shape (2, 1)
result = a + b
print(result)
```

```
[[11 12 13]
 [24 25 26]]
```

Visualization and Plotting

```
%matplotlib inline
import matplotlib.pyplot as plt
x = [0, 1, 2, 3]
y = [0, 1, 4, 9]
plt.plot(x, y)
plt.show()
```

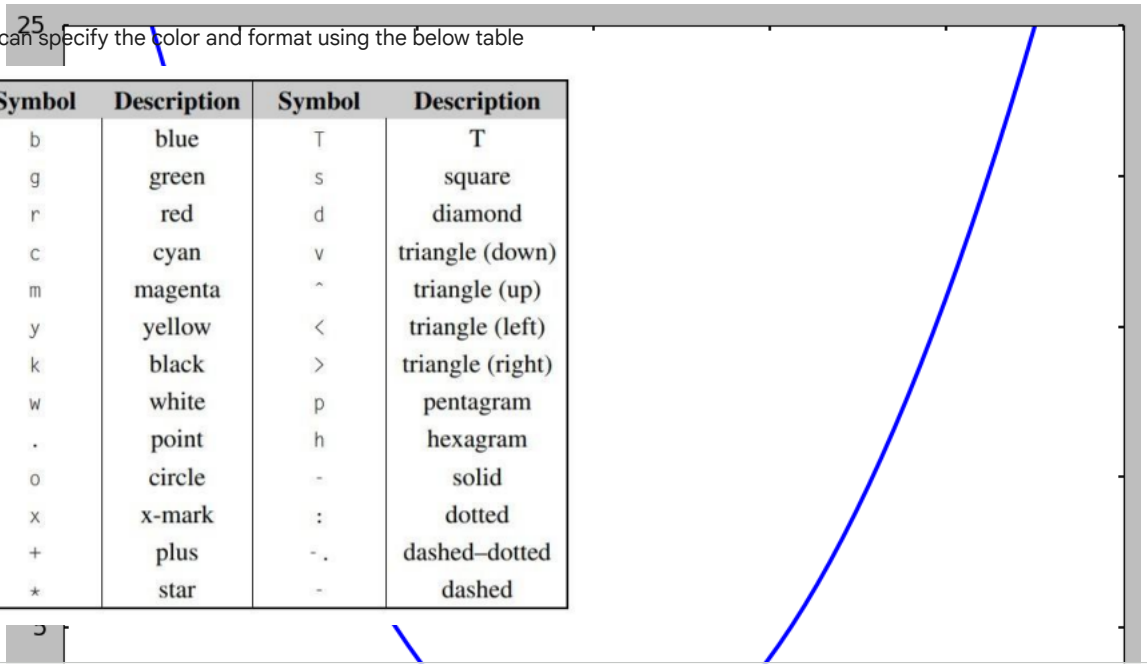


Make a plot of the function f (x) = x^2 for −5 ≤ x ≤ 5

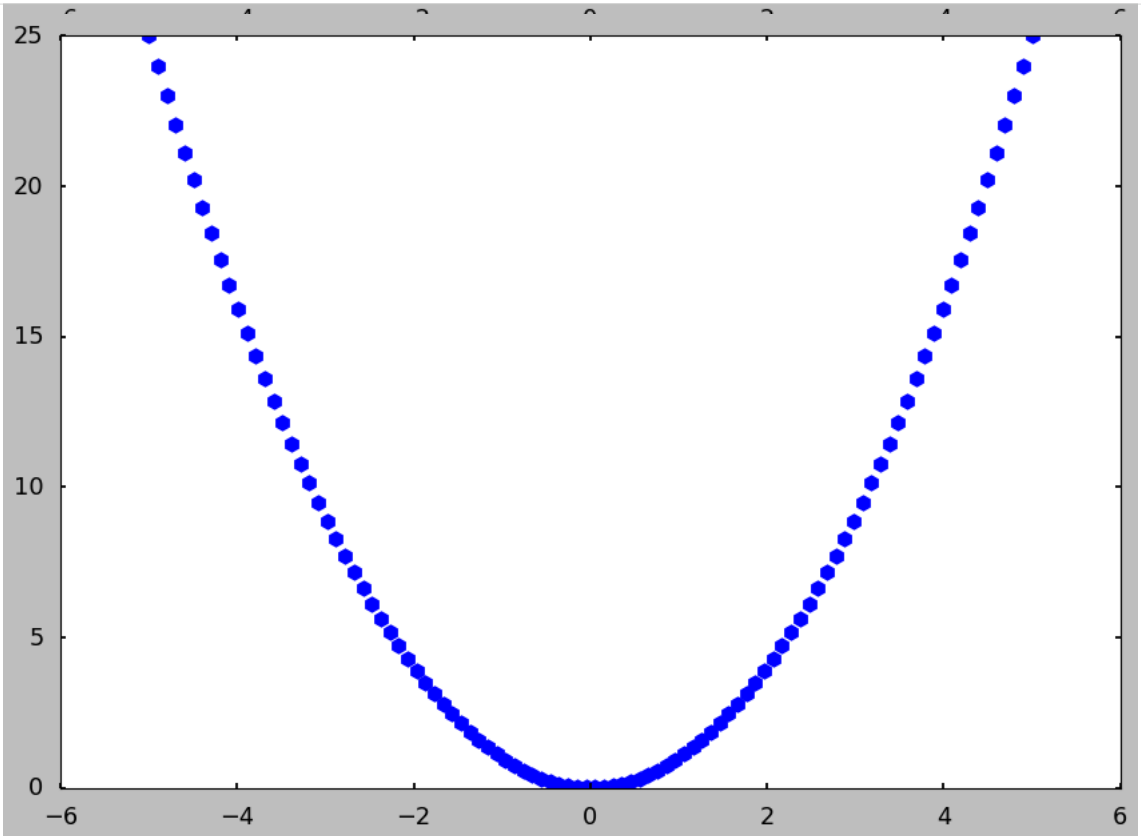
```
%matplotlib inline
x = np.linspace(-5,5, 70)
plt.plot(x, x**2)
plt.show()
#print(x)
```

you can specify the color and format using the below table

Symbol	Description	Symbol	Description
b	blue	T	T
g	green	s	square
r	red	d	diamond
c	cyan	v	triangle (down)
m	magenta	^	triangle (up)
y	yellow	<	triangle (left)
k	black	>	triangle (right)
w	white	p	pentagram
.	point	h	hexagram
o	circle	-	solid
x	x-mark	:	dotted
+	plus	-.	dashed–dotted
*	star	-	dashed

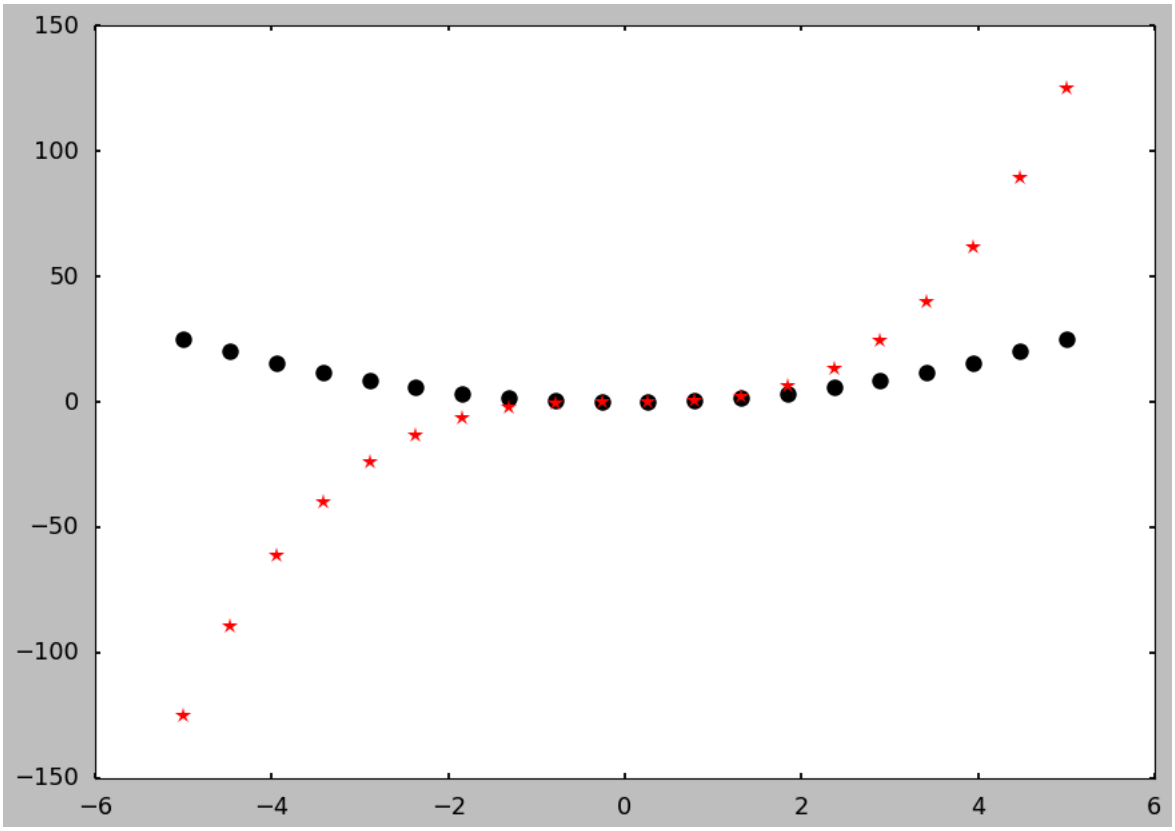


```
%matplotlib inline
x = np.linspace(-5,5, 100)
plt.plot(x, x**2,"h")
plt.show()
```

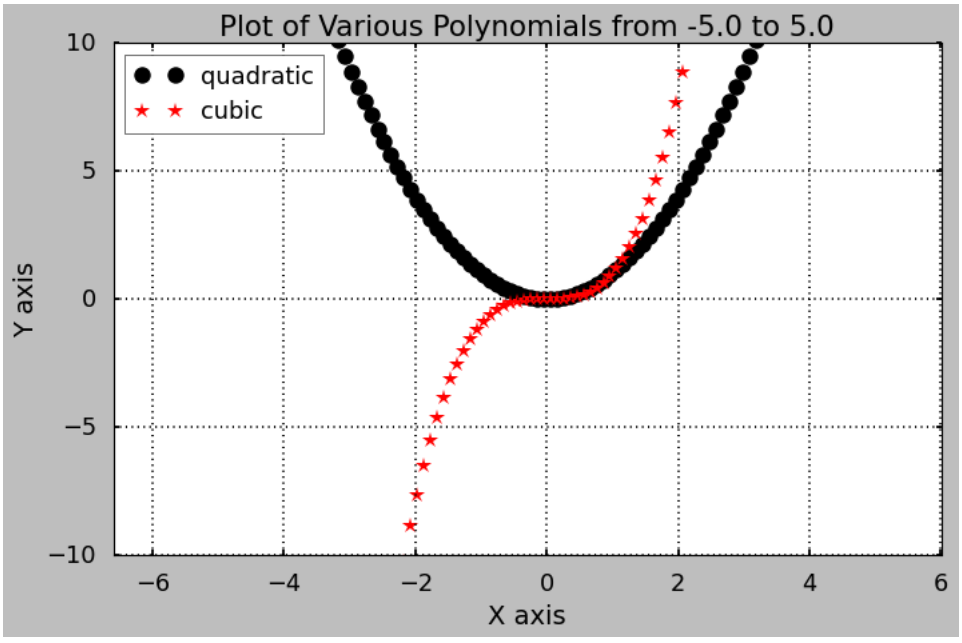


Make a plot of the function f (x) = x^2 and g(x) = x^3 for −5 ≤ x ≤ 5. Use different colors and markers for each function.

```
x = np.linspace(-5,5,20)
plt.plot(x, x**2, "ko")
plt.plot(x, x**3, "r*")
plt.show()
```

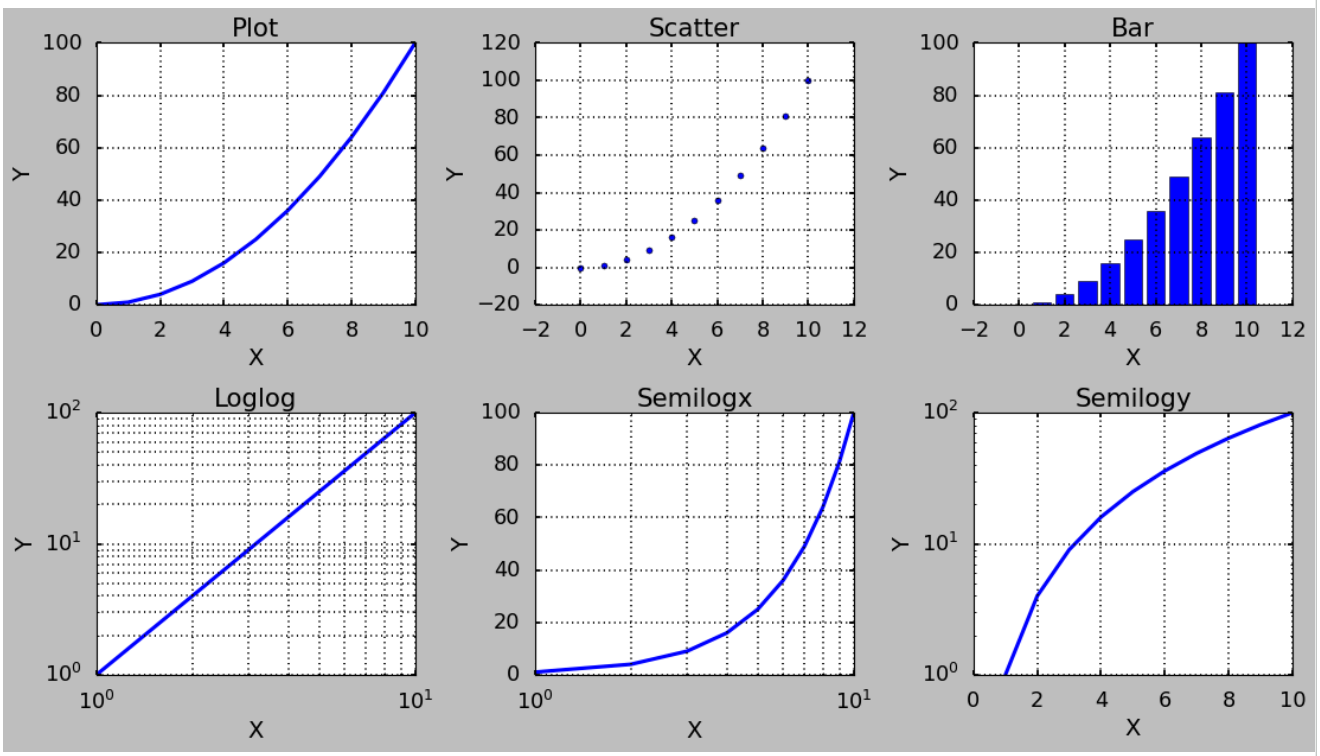



```
plt.figure(figsize = (10,6))
x = np.linspace(-5,5,100)
plt.plot(x, x**2, "ko", label = "quadratic")
plt.plot(x, x**3, "r*", label = "cubic")
plt.title(f"Plot of Various Polynomials from {x[0]} to {x[-1]}")
plt.xlabel("X axis")
plt.ylabel("Y axis")
plt.legend(loc = 2)
plt.xlim(-6.6)
plt.ylim(-10,10)
plt.grid()
plt.show()
```



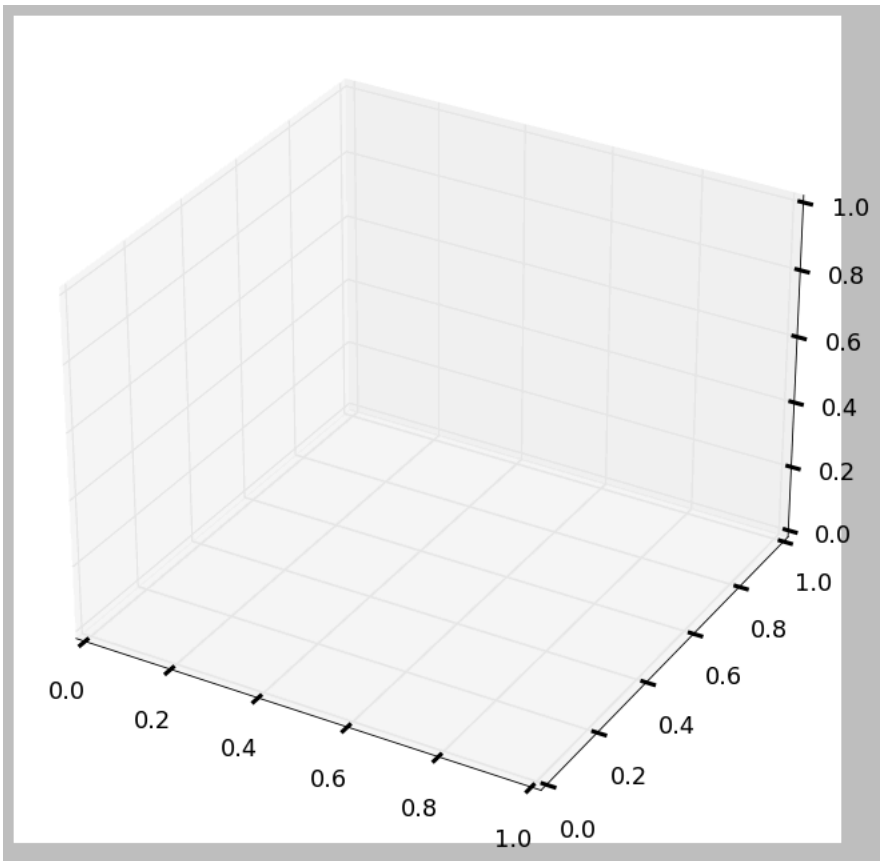
Given the lists x = np.arange(11) and y = x^2, create a 2 × 3 subplot where each subplot plots x versus y using plot, scatter, bar, loglog, semilogx, and semilogy. Title and label each plot appropriately. Use a grid, but a legend is not necessary here.

```
x = np.arange(11)  #np.arange(start, stop, step)
y = x**2
plt.figure(figsize = (14, 8)) # Width: 14 inches, Height: 8 inches
plt.subplot(2, 3, 1)  # R, C, Position
plt.plot(x,y)
plt.title("Plot")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid()
plt.subplot(2, 3, 2)
plt.scatter(x,y)
plt.title("Scatter")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid()
plt.subplot(2, 3, 3)
plt.bar(x,y)
plt.title("Bar")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid()
plt.subplot(2, 3, 4)
plt.loglog(x,y)
plt.title("Loglog")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(which="both")
plt.subplot(2, 3, 5)
plt.semilogx(x,y)
plt.title("Semilogx")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid(which="both")
plt.subplot(2, 3, 6)
plt.semilogy(x,y)
plt.title("Semilogy")
plt.xlabel("X")
plt.ylabel("Y")
plt.grid()
plt.tight_layout()
plt.show()
```



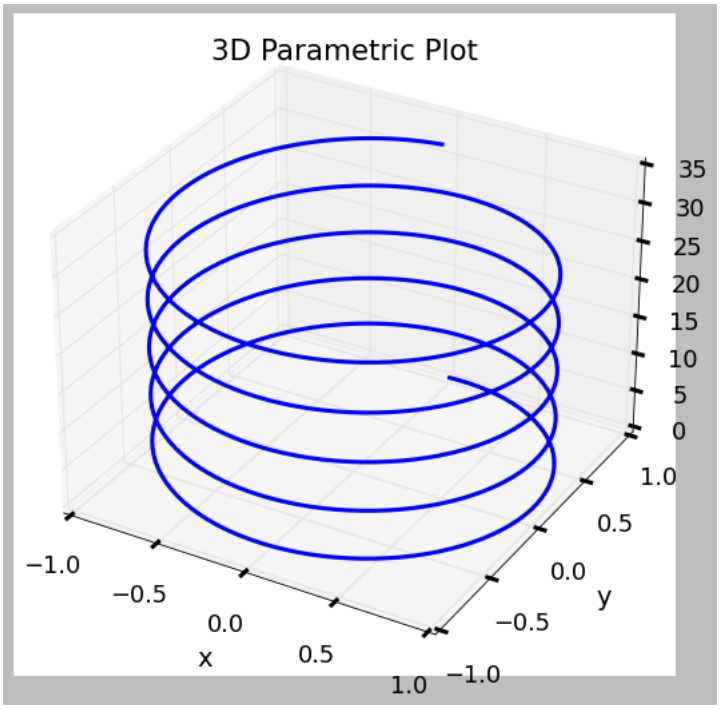
3D plotting

```
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
plt.style.use("seaborn-v0_8-poster")
fig = plt.figure(figsize = (10,10))
ax = plt.axes(projection="3d")
plt.show()
```



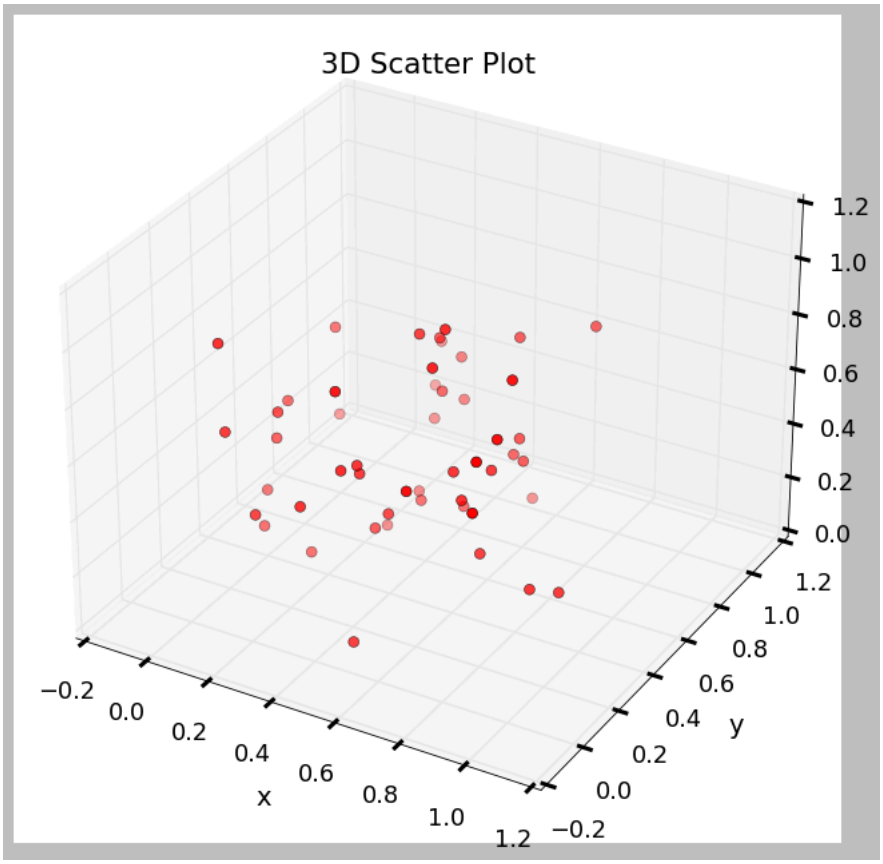
Consider the parameterized dataset: t is a vector from 0 to 10π with a step π/50, x = sin(t), and y = cos(t). Make a 3D plot of the (x, y,t) dataset using plot3. Turn on the grid, make the axis equal, and add axis labels and a title. Activate the i

```
%matplotlib inline
fig = plt.figure(figsize = (8,8))
ax = plt.axes(projection="3d")
ax.grid()
t = np.arange(0, 10*np.pi, np.pi/50)
x = np.sin(t)
y = np.cos(t)
ax.plot3D(x, y, t)
ax.set_title("3D Parametric Plot")
# Set axes label
ax.set_xlabel("x", labelpad=20)
ax.set_ylabel("y", labelpad=20)
ax.set_zlabel("t", labelpad=20)
plt.show()
```



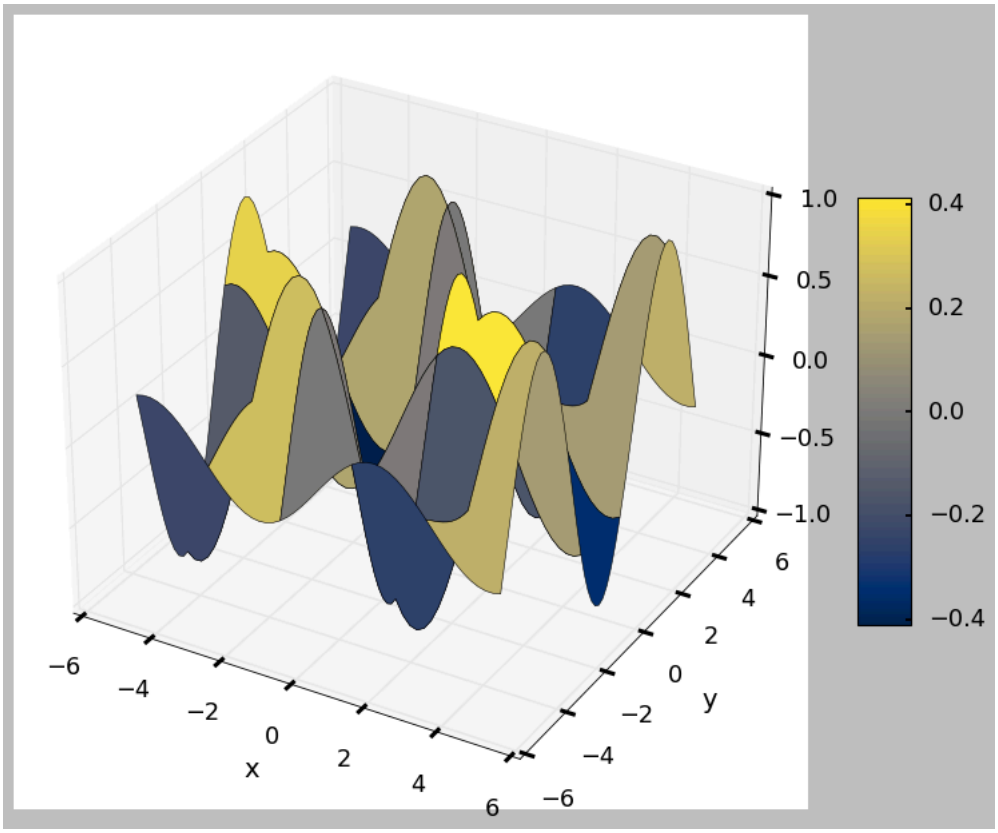
Make a 3D scatter plot with randomly generated 50 data points for x, y, and z. Set the point color as red and size of the point as 50.

```
%matplotlib inline
x = np.random.random(50)
y = np.random.random(50)
z = np.random.random(50)
fig = plt.figure(figsize = (10,10))
ax = plt.axes(projection="3d")
ax.grid()
ax.scatter(x, y, z, c = "r", s = 50)
ax.set_title("3D Scatter Plot")
# Set axes label
ax.set_xlabel("x", labelpad=20)
ax.set_ylabel("y", labelpad=20)
ax.set_zlabel("z", labelpad=20)
plt.show()
```



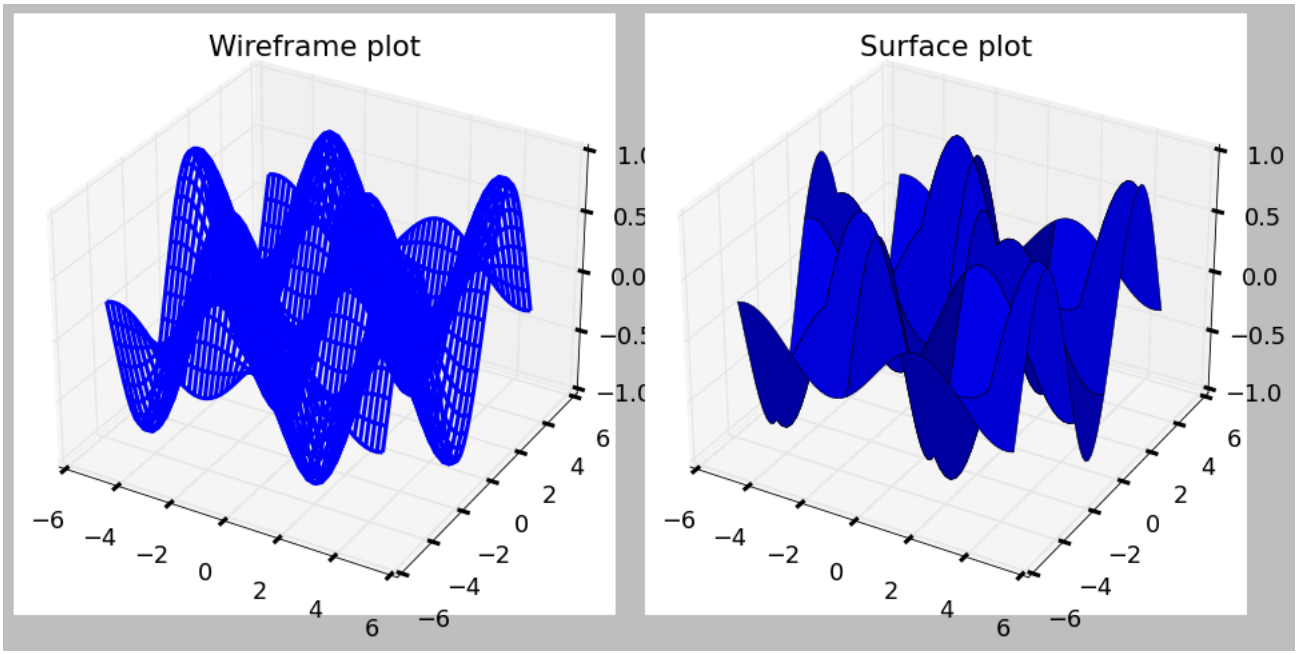
Make a plot of the surface f (x, y) = sin(x) · cos(y) for −5 ≤ x ≤ 5,−5 ≤ y ≤ 5 using the plot_surface function. Take care to use a sufficiently fine discretization in x and y so that the plot looks smooth.

```
fig = plt.figure(figsize = (12,10))
ax = plt.axes(projection="3d")
x = np.arange(-5, 5.1, 0.2)
y = np.arange(-5, 5.1, 0.2)
X, Y = np.meshgrid(x, y) #2D arrays (matrix) of x and y coordinates
Z = np.sin(X)*np.cos(Y)
surf = ax.plot_surface(X, Y, Z, cmap = plt.cm.cividis) #create a 3D surface plot, cividis is a color gradient from blue to yell
# Set axes label
ax.set_xlabel("x", labelpad=20)
ax.set_ylabel("y", labelpad=20)
ax.set_zlabel("z", labelpad=20)
fig.colorbar(surf, shrink=0.5, aspect=8)
plt.show()
```



Make a 1 × 2 subplot to plot the above X, Y, Z data in a wireframe plot and a surface plot

```
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1, 2, 1, projection="3d")
ax.plot_wireframe(X,Y,Z)
ax.set_title("Wireframe plot")
ax = fig.add_subplot(1, 2, 2, projection="3d")
ax.plot_surface(X,Y,Z)
ax.set_title("Surface plot")
plt.tight_layout()
plt.show()
```



Annimations of the graphs:

Create an animation of a red circle following a blue sine wave

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as manimation

n = 1000
x = np.linspace(0, 6*np.pi, n)
```