In [ ]: Using numpy

Creating Numpy Arrays.

In [2]: *#1. Create the evenly spaced array using arrange?>*
```python
import numpy as np
a=np.arange(1,20)
print(a)
x=range(1,20)
print(x)
print(list(x))
x=np.arange(20.4)
print(x)
x=np.arange(0.8,12.5,0.3)
print(x)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
range(1, 20)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17.
 18. 19. 20.]
[ 0.8  1.1  1.4  1.7  2.   2.3  2.6  2.9  3.2  3.5  3.8  4.1  4.4  4.7
  5.   5.3  5.6  5.9  6.2  6.5  6.8  7.1  7.4  7.7  8.   8.3  8.6  8.9
  9.2  9.5  9.8 10.1 10.4 10.7 11.  11.3 11.6 11.9 12.2]
```

In [3]: *#2. Create the evenly spaced array using linespace?*
```python
import numpy as np
print("50 values between 1 to 20",np.linspace(1,20))
print("7 values between 1 to 20",np.linspace(1,20,7))
print("excluding end points",np.linspace(1,20,7,endpoint=False))
```

```
50 values between 1 to 20 [ 1.         1.3877551  1.7755102  2.16326531 2.55102041 2.93877551
  3.32653061  3.71428571  4.10204082  4.48979592  4.87755102  5.26530612
  5.65306122  6.04081633  6.42857143  6.81632653  7.20408163  7.59183673
  7.97959184  8.36734694  8.75510204  9.14285714  9.53061224  9.91836735
 10.30612245 10.69387755 11.08163265 11.46938776 11.85714286 12.24489796
 12.63265306 13.02040816 13.40816327 13.79591837 14.18367347 14.57142857
 14.95918367 15.34693878 15.73469388 16.12244898 16.51020408 16.89795918
 17.28571429 17.67346939 18.06122449 18.44897959 18.83673469 19.2244898
 19.6122449  20.        ]
7 values between 1 to 20 [ 1.          4.16666667  7.33333333 10.5        13.66666667 16.83333333
 20.        ]
excluding end points [ 1.          3.71428571  6.42857143  9.14285714 11.85714286 14.57142857
 17.28571429]
```

In [4]: *#3.Create a Zero dimension array in Numpy also print its type and dimension?*
```python
import numpy as np
x = np.array(40)
print("x: ", x)
print("The type of x: ", type(x))
print("The dimension of x:", np.ndim(x))
```

```
x:  40
The type of x:  <class 'numpy.ndarray'>
The dimension of x: 0
```

In [5]: *#4. Create two One dimension array in Numpy also print its type and dimension?*
```python
a = np.array([1,5,6,3,8,10,23])
b = np.array([1.2,4.5,3.0,6.8])
print("a: ", a)
print("b: ", b)
print("Type of a: ", a.dtype)
print("Type of b: ", b.dtype)
print("Dimension of a: ", np.ndim(a))
print("Dimension of b: ", np.ndim(b))
```

```
a:  [ 1  5  6  3  8 10 23]
b:  [1.2 4.5 3.  6.8]
Type of a:  int32
Type of b:  float64
Dimension of a:  1
Dimension of b:  1
```

In [7]: *#5. Write a code to create a two dimension array in Numpy and print its dimension?*
```python
x = np.array([ [1.2,5.6,3.4],
               [1.1,-0.2,-5.6],
               [5.6,3.4,8.6]])
print(x)
print(x.ndim)
```

```
[[ 1.2  5.6  3.4]
 [ 1.1 -0.2 -5.6]
 [ 5.6  3.4  8.6]]
2
```

In [8]: *#6. Create a multi dimension array in Numpy and print its dimension?*

```python
x = np.array([ [[11,45], [56,34]],
```

```
                [[67,12], [89,34]],
                [[12,61], [98,45]] ])
        print(x)
        print(x.ndim)

[[[11 45]
  [56 34]]

 [[67 12]
  [89 34]]

 [[12 61]
  [98 45]]]
3
```

In [10]:
```
#7. Write a code to return the shape of an array
x = np.array([ [12,23,34],
              [45,56,67],
              [78,89,90],
              [98,87,76],
              [65,54,43],
              [32,21,10]])

print(np.shape(x))
```

```
(6, 3)
```

In [11]:
```
#8. Write a code  to change the shape of the array
x.shape=(3, 6)
print(x)
```

```
[[12 23 34 45 56 67]
 [78 89 90 98 87 76]
 [65 54 43 32 21 10]]
```

In [12]:
```
#Indexing and Sorting
#9. Write a code  to print the numbers in an array using its index value?

F = np.array([1, 1, 2, 3, 5, 8, 13, 21])
# print the first element of F
print(F[0])
# print the last element of F
print(F[-1])
```

```
1
21
```

In [13]:
```
#10. Write a code  to print the number using index value in multidimensional array?
A = np.array([ [3.4, 8.7, 9.9],
              [1.1, -7.8, -0.7],
              [4.1, 12.3, 4.8]])

print(A[1][0])
```

```
1.1
```

In [14]:
```
#11.Print the index 1 of array A and return the 0th index in the 1st index of A?
tmp = A[1]
print(tmp)
print(tmp[0])
```

```
[ 1.1 -7.8 -0.7]
1.1
```

In [15]:
```
#12.Perform slicing in single dimensional array?
S = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print(S[2:5])
print(S[:4])
print(S[6:])
print(S[:])
```

```
[2 3 4]
[0 1 2 3]
[6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
```

In [16]:
```
#13. Write a code  to Perform slicing in multi dimensional array?
A = np.array([
[11, 12, 13, 14, 15],
[21, 22, 23, 24, 25],
[31, 32, 33, 34, 35],
[41, 42, 43, 44, 45],
[51, 52, 53, 54, 55]])

print(A[:3, 2:])
print(A[3:, :])
print(A[:, 4:])
```

```
[[13 14 15]
 [23 24 25]
 [33 34 35]]
[[41 42 43 44 45]
 [51 52 53 54 55]]
[[15]
 [25]
 [35]
 [45]
 [55]]
```

In [17]: #14.Given A= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] where the values are taken from index 2 to index 5 is stored in S and change the index 0 to 22 and index 1 to
```
A = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
S = A[2:6]
S[0] = 22
S[1] = 23
print(A)
```

```
[ 0  1 22 23  4  5  6  7  8  9]
```

In [18]: #15.Write a code to check whether the two array A and S share the same memory space?
```
np.may_share_memory(A, S)
```

Out[18]:True

In [19]: #16.Create an array of 3 rows and 4 columns using numpy.arange function and change its 0 index value to be 42
```
A = np.arange(12)
B = A.reshape(3, 4)
A[0] = 42
print(B)
```

```
[[42  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [20]: #Numerical Operation
#17.A list is given lst = [2,3, 7.9, 3.3, 6.9, 0.11, 10.3, 12.9] , add 2 to eachelement , multiply each element with 2.2 and subtract each element with1.38
```
import numpy as np
lst = [2,3, 7.9, 3.3, 6.9, 0.11, 10.3, 12.9]
v = np.array(lst)
v = v + 2
print(v)
print(v * 2.2)
print(v - 1.38)
```

```
[ 4.   5.   9.9  5.3  8.9  2.11 12.3 14.9 ]
[ 8.8  11.   21.78 11.66 19.58  4.642 27.06 32.78 ]
[ 2.62  3.62  8.52  3.92  7.52  0.73 10.92 13.52]
```

In [21]: #Arithmethic Operation of two arrays
#18.Create two arrays named A and B and add them, add 1 with each element in B and find the product of two arrays
```
import numpy as np
A = np.array([ [11, 12, 13], [21, 22, 23], [31, 32, 33] ])
B = np.ones((3,3))
print("Adding to arrays: ")
print(A + B)
print("\nMultiplying two arrays: ")
print(A * (B + 1))
```

```
Adding to arrays:
[[12. 13. 14.]
 [22. 23. 24.]
 [32. 33. 34.]]

Multiplying two arrays:
[[22. 24. 26.]
 [42. 44. 46.]
 [62. 64. 66.]]
```

In [22]: #Matrices Vs Two Dimensional Arrays
#19. Write a code to perform multiplication of two arrays
```
import numpy as np
A = np.array([ [1, 2, 3], [2, 2, 2], [3, 3, 3] ])
B = np.array([ [3, 2, 1], [1, 2, 3], [-1, -2, -3] ])
R = A * B
print(R)
```

```
[[ 3  4  3]
 [ 2  4  6]
 [-3 -6 -9]]
```

In [23]: #Comparison Operators
#20.Check whether two arrays are equal using comparison operator '==',where A=[[11, 12, 13], [21, 22, 23], [31, 32, 33]] and B=[[11, 102, 13], [201,
```
import numpy as np
A = np.array([ [11, 12, 13], [21, 22, 23], [31, 32, 33] ])
B = np.array([ [11, 102, 13], [201, 22, 203], [31, 32, 303] ])
A == B
```

```
Out[23]:array([[ True, False,  True],
                [False,  True, False],
                [ True,  True, False]])
```

In [24]: #21. Check whether two arrays are equal using numpy.array_equal(),using the previously created array "A" and "B"
         print(np.array_equal(A, B))

```
False
```

In [25]: #Logical Operators
         #22.Given a=([[True, True], [False, False]]) and b=([[True, False], [True, False]]) check the arrays using logical_or and logical_and.
         a = np.array([[True, True], [False, False]])
         b = np.array([[True, False], [True, False]])
         print(np.logical_or(a, b))
         print(np.logical_and(a, b))

```
[[ True  True]
 [ True False]]
[[ True False]
 [False False]]
```

In [26]: #Operations on Arrays with its Symbols
         #23.Write a code to create multidimensional array A and a single dimensional array B .
         #Multiply A and B
         #Add A and B
         import numpy as np
         A = np.array([[11, 12, 13], [21, 22, 23], [31, 32, 33]])
         B = np.array([1, 2, 3])
         print("Multiplication with broadcasting: ")
         print(A * B)
         print("... and now addition with broadcasting: ")
         print(A + B)

```
Multiplication with broadcasting:
[[11 24 39]
 [21 44 69]
 [31 64 99]]
... and now addition with broadcasting:
[[12 14 16]
 [22 24 26]
 [32 34 36]]
```

In [27]: #24.Given B = [1, 2, 3] , print array B with 3 rows and 3 columns
         B = np.array([[1, 2, 3],] * 3)
         print(B)

```
[[1 2 3]
 [1 2 3]
 [1 2 3]]
```

In [28]: #25.Consider the output of previously given array B and print the transpose of B
         np.array([[1, 2, 3],] * 3).transpose()

```
Out[28]:array([[1, 1, 1],
                [2, 2, 2],
                [3, 3, 3]])
```

In [29]: #26. Given B = [1, 2, 3],consider the rows of B as columns and print the output.
         B = np.array([1, 2, 3])
         B[:, np.newaxis]

```
Out[29]:array([[1],
                [2],
                [3]])
```

In [30]: #Flatten and Reshape arrays
         #27.Write a code for flattening using the values "C", "F" and "A"
         import numpy as np
         A = np.array([[[0, 1],
                 [2, 3],
                 [4, 5],
                 [6, 7]],
                [[8, 9],
                 [10, 11],
                 [12, 13],
                 [14, 15]],
                [[16, 17],
                 [18, 19],
                 [20, 21],
                 [22, 23]]])
         Flattened_X = A.flatten()
         print(Flattened_X)
         print(A.flatten(order="C"))
         print(A.flatten(order="F"))
         print(A.flatten(order="A"))

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
[ 0  8 16  2 10 18  4 12 20  6 14 22  1  9 17  3 11 19  5 13 21  7 15 23]
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

In [31]: #28.Write a code to create an array named X using reshape ()
```python
X = np.array(range(24))
Y = X.reshape((3,4,2))
Y
```

Out[31]:array([[[0, 1],
               [2, 3],
               [4, 5],
               [6, 7]],

              [[8, 9],
               [10, 11],
               [12, 13],
               [14, 15]],

              [[16, 17],
               [18, 19],
               [20, 21],
               [22, 23]]])

In [32]: #Random Number
```python
#29.Write a code to print a random number between 1 and 6 using random.randint function
import random
outcome = random.randint(1,6)
print(outcome)
```

1

In [33]: #30.Write a code to print 10 random numbers between 1 and 6 using random.randint function
```python
import random
[ random.randint(1, 6) for _ in range(10) ]
```

Out[33]:[5, 6, 2, 4, 4, 2, 5, 3, 5, 2]

In [34]: #31.Using random.randint()
```python
#Random number between 1 to 7
#Random number between 1 to 7 with size=1
#Random number between 1 to 7 with size =10
#Random number between 1 to 7 with 5 rows and 4 columns
import numpy as np
print(np.random.randint(1, 7))
print(np.random.randint(1, 7, size=1))
print(np.random.randint(1, 7, size=10))
print(np.random.randint(1, 7, size=(5, 4)))
```

1
[2]
[5 5 6 2 3 1 6 5 2 6]
[[2 5 5 3]
 [3 3 6 3]
 [1 4 1 2]
 [1 1 1 1]
 [5 1 1 6]]

In [35]: #32.Create a list using string values and by using random choice function print the random string value from the list created

```python
from random import choice
possible_destinations = ["Berlin", "Hamburg", "Munich",
            "Amsterdam", "London", "Paris",
            "Zurich", "Heidelberg", "Strasbourg",
            "Augsburg", "Milan", "Rome"]

print(choice(possible_destinations))
```

Hamburg

In [36]: #33.Using random.random_sample function print the random float values with 3 rows and 4 columns
```python
import numpy as np
x = np.random.random_sample((3, 4))
print(x)
```

[[0.02668763 0.3424312 0.45238071 0.28821201]
 [0.50450609 0.68041332 0.00960145 0.43129564]
 [0.19510998 0.84109866 0.61018697 0.46652433]]

In [ ]: #Boolean Indexing

In [37]: #34.Given B = np.array([[42,56,89,6],[99,88,42,12],[55,42,17,18]]) print False for the values which are greater than 42 using Boolean indexing
```python
B = np.array([[42,56,89,65],
        [99,88,42,12],
        [55,42,17,18]])

print(B>=42)
```

[[True True True True]
 [True True True False]
 [True True False False]]

In [ ]: #Fancy Indexing

In [38]: #35.Write a program of one array to select the corresponding index of another array A contains are of C where the corresponding value of (A<=5) is tr

```
C = np.array([123,188,190,99,77,88,100])
A = np.array([4,7,2,8,6,9,5])
R = C[A<=5]
print(R)
```

[123 190 100]

In [39]:
```
#36.Extract from the array np.array([3,4,6,10,24,89,45,43,46,99,100]) with Boolean masking all the number
#which are not divisible by 3
#which are divisible by 5
#which are divisible by 3 and 5
#which are divisible by 3 and set them to 42
import numpy as np
A = np.array([3,4,6,10,24,89,45,43,46,99,100])
div3 = A[A%3!=0]
print("Elements of A not divisible by 3:")
print(div3)
div5 = A[A%5==0]
print("Elements of A divisible by 5:")
print(div5)
print("Elements of A, which are divisible by 3 and 5:")
print(A[(A%3==0) & (A%5==0)])
print("------------------")
A[A%3==0] = 42
print("""New values of A after setting the elements of A,
which are divisible by 3, to 42:""")
print(A)
```

Elements of A not divisible by 3:
[4 10 89 43 46 100]
Elements of A divisible by 5:
[10 45 100]
Elements of A, which are divisible by 3 and 5:
[45]

New values of A after setting the elements of A,
which are divisible by 3, to 42:
[42   4 42 10 42 89 42 43 46 42 100]

In [ ]:
```
#DATA MANIPULATION WITH PANDAS
#Series
```

In [40]:
```
#1. Write the program to define a simple Series object in the following example by instatiating a Pandas Series object with a list?
import pandas as pd
S = pd.Series([11, 28, 72, 3, 5, 8])
S
```

Out[40]:
```
0    11
1    28
2    72
3     3
4     5
5     8
dtype: int64
```

In [41]:
```
#2. What is the code that can directly access the index and the values of our Series S?
print(S.index)
print(S.values)
```

RangeIndex(start=0, stop=6, step=1)
[11 28 72 3 5 8]

In [42]:
```
#3. If two series are taken S and S2 then write the code for addition of these two series with printing the same index?
fruits = ['apples', 'oranges', 'cherries', 'pears']
S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits)
print(S + S2)
print("sum of S: ", sum(S))
```

```
apples     37
oranges    46
cherries   83
pears      42
dtype: int64
sum of S:  115
```

In [43]:
```
#4. What will be the program code for the above question if the indices do not have to be the same for the Series addition If an index doesn't occur in b
fruits = ['peaches', 'oranges', 'cherries', 'pears']
fruits2 = ['raspberries', 'oranges', 'cherries', 'pears']
S = pd.Series([20, 33, 52, 10], index=fruits)
S2 = pd.Series([17, 13, 31, 32], index=fruits2)
print(S + S2)
```

```
cherries      83.0
oranges       46.0
peaches       NaN
pears         42.0
raspberries   NaN
dtype: float64
```

In [44]: *#Q5. Write the code function to extract some elements from the given series object based on the actual positions of# importing pandas as pd*
```python
import pandas as pd
# importing re for regular expressions
import re
# Creating the Series
sr = pd.Series(['New_York', 'Lisbon', 'Tokyo', 'Paris', 'Munich'])
# Creating the index
idx = ['City 1', 'City 2', 'City 3', 'City 4', 'City 5']
# set the index
sr.index = idx
# Print the series
print(sr)
```

```
City 1   New_York
City 2     Lisbon
City 3      Tokyo
City 4      Paris
City 5     Munich
dtype: object
```

In [45]: *#Indexing*
#Q6. Using indexing write the code to access single values of a Series ?
```python
import pandas as pd
fruits = ['apples', 'oranges', 'cherries', 'pears']
S = pd.Series([20, 33, 52, 10], index=fruits)
print(S['apples'])
```

```
20
```

In [46]: *#7. Write a code for Manipulating Pandas Data frame using Applying lambda function to a column?*
```python
import pandas as pd
values = [['Rohan', 455], ['Elvish', 250], ['Deepak', 495],
          ['Sai', 400], ['Radha', 350], ['Vansh', 450]]

df = pd.DataFrame(values, columns=['Name', 'Univ_Marks'])
df = df.assign(Percentage=lambda x: (x['Univ_Marks'] / 500 * 100))
df
```

Out[46]:

	Name	Univ_Marks	Percentage
0	Rohan	455	91.0
1	Elvish	250	50.0
2	Deepak	495	99.0
3	Sai	400	80.0
4	Radha	350	70.0
5	Vansh	450	90.0

In [ ]: *#Creating Series object from Dictionary*

In [47]: *#8. How to create a Series object in pandas for the resulting Series to contain the dict's keys as the indices and the values as the values?*
```python
cities = {"London":    8615246,
          "Berlin":    3562166,
          "Madrid":    3165235,
          "Rome":      2874038,
          "Paris":     2273305,
          "Vienna":    1805681,
          "Bucharest": 1803425,
          "Hamburg":   1760433,
          "Budapest":  1754000,
          "Warsaw":    1740119,
          "Barcelona": 1602386,
          "Munich":    1493900,
          "Milan":     1350680}
city_series = pd.Series(cities)
print(city_series)
```

```
London      8615246
Berlin      3562166
Madrid      3165235
Rome        2874038
Paris       2273305
Vienna      1805681
Bucharest   1803425
Hamburg     1760433
Budapest    1754000
Warsaw      1740119
Barcelona   1602386
Munich      1493900
Milan       1350680
dtype: int64
```

In [49]: *#DataFrame#9. Three series are defined using pandas write the code to concantenate and show the output display ?*

```python
import pandas as pd
years = range(2014, 2018)
shop1 = pd.Series([2409.14, 2941.01, 3496.83, 3119.55], index=years)
shop2 = pd.Series([1203.45, 3441.62, 3007.83, 3619.53], index=years)
shop3 = pd.Series([3412.12, 3491.16, 3457.19, 1963.10], index=years)
pd.concat([shop1, shop2, shop3])
```

Out[49]:
```
2014    2409.14
2015    2941.01
2016    3496.83
2017    3119.55
2014    1203.45
2015    3441.62
2016    3007.83
2017    3619.53
2014    3412.12
2015    3491.16
2016    3457.19
2017    1963.10
dtype: float64
```

In [50]: *#DataFrame from Dictionary*
#10. Give an example to derive a dataframe from a dictionary using pandas library function ?

```python
cities = {"name": ["London", "Berlin", "Madrid", "Rome",
            "Paris", "Vienna", "Bucharest", "Hamburg",
            "Budapest", "Warsaw", "Barcelona",
            "Munich", "Milan"],
        "population": [8615246, 3562166, 3165235, 2874038,
                2273305, 1805681, 1803425, 1760433,
                1754000, 1740119, 1602386, 1493900,
                1350680],
        "country": ["England", "Germany", "Spain", "Italy",
                "France", "Austria", "Romania",
                "Germany", "Hungary", "Poland", "Spain",
                "Germany", "Italy"]}
city_frame = pd.DataFrame(cities)
city_frame
```

Out[50]:

	name	population	country
0	London	8615246	England
1	Berlin	3562166	Germany
2	Madrid	3165235	Spain
3	Rome	2874038	Italy
4	Paris	2273305	France
5	Vienna	1805681	Austria
6	Bucharest	1803425	Romania
7	Hamburg	1760433	Germany
8	Budapest	1754000	Hungary
9	Warsaw	1740119	Poland
10	Barcelona	1602386	Spain
11	Munich	1493900	Germany
12	Milan	1350680	Italy

In [51]: *#11. Give an example to derive a dataframe from a dictionary using pandas function?*

```python
import pandas
data = {'Ojaswi': {'Age': 15, 'subject': 'java', 'Address': 'Hyderabad'},
    'Rohith':  {'Age': 9, 'subject': 'python', 'Address': 'Hyderabad'},
```

```
'Gnanesh': {'Age': 15, 'subject': 'c/c++', 'Address': 'Guntur'},
'divya': {'Age': 21, 'subject': 'html', 'Address': 'ponnur'},
'ramya': {'Age': 15, 'subject': 'c/c++', 'Address': 'delhi'}}

data = pandas.DataFrame(data)
data
```

Out[51]:

	Ojaswi	Rohith	Gnanesh	divya	ramya
Age	15	9	15	21	15
subject	java	python	c/c++	html	c/c++
Address	Hyderabad	Hyderabad	Guntur	ponnur	delhi

In [52]:
```
#12. Give the program to change both the column order and the ordering of the index with the function reindex ?
city_frame.reindex(index=[0, 2, 4, 6,  8, 10, 12, 1, 3, 5, 7, 9, 11],
columns=['country', 'name', 'population'])
```

Out[52]:

	country	name	population
0	England	London	8615246
2	Spain	Madrid	3165235
4	France	Paris	2273305
6	Romania	Bucharest	1803425
8	Hungary	Budapest	1754000
10	Spain	Barcelona	1602386
12	Italy	Milan	1350680
1	Germany	Berlin	3562166
3	Italy	Rome	2874038
5	Austria	Vienna	1805681
7	Germany	Hamburg	1760433
9	Poland	Warsaw	1740119
11	Germany	Munich	1493900

In [53]:
```
#13. Write the code to rename a dataframe using pandas library function?
city_frame.rename(columns={"name":"Soyadı",
                "country":"Ülke",
                "population":"Nüfus"},
        inplace=True)
city_frame
```

Out[53]:

	Soyadı	Nüfus	Ülke
0	London	8615246	England
1	Berlin	3562166	Germany
2	Madrid	3165235	Spain
3	Rome	2874038	Italy
4	Paris	2273305	France
5	Vienna	1805681	Austria
6	Bucharest	1803425	Romania
7	Hamburg	1760433	Germany
8	Budapest	1754000	Hungary
9	Warsaw	1740119	Poland
10	Barcelona	1602386	Spain
11	Munich	1493900	Germany
12	Milan	1350680	Italy

In [54]:
```
#14.  Write the program for accessing row via indexing value ie select the  German cities in the following example by using 'loc' ?
city_frame = pd.DataFrame(cities,
                columns=("name", "population"),
                index=cities["country"])
print(city_frame.loc["Germany"])
```

```
         name  population
Germany  Berlin    3562166
Germany  Hamburg   1760433
Germany  Munich    1493900
```

In [55]:
```
#Pandas Pivot Table
#15.Write a program to perform a pivot table format by reshaping a dataframe in pandas library function?
import pandas as pd
d = {'A': ['kırmızı', 'yeşil', 'mavi', 'kırmızı', 'yeşil', 'mavi'],
```

```python
         'B': ['bir', 'iki', 'bir',  'iki',  'bir', 'iki'],
         'C': [345, 325, 898, 989, 23, 143],
         'D': [1, 2, 3, 4, 5, 6]}
df = pd.DataFrame(d)
df
```

Out[55]:

	A	B	C	D
0	kırmızı	bir	345	1
1	yeşil	iki	325	2
2	mavi	bir	898	3
3	kırmızı	iki	989	4
4	yeşil	bir	23	5
5	mavi	iki	143	6

In [56]: *#16. Write a program where a Series object with an index of size nvalues. The index will not be unique, because the strings for the index are taken fro*

```python
import pandas as pd
import numpy as np
import random
nvalues = 30
values = np.random.randint(1, 20, (nvalues,))
fruits = ["bananas", "oranges", "apples", "clementines", "cherries", "pears"]
fruits_index = np.random.choice(fruits, (nvalues,))
s = pd.Series(values, index=fruits_index)
print(s[:10])
```

```
apples        9
oranges      12
pears         9
apples        2
cherries     17
bananas      11
apples       13
clementines   6
cherries      8
cherries      8
dtype: int32
```

In [57]: *#Pandas Groupby*
#17. Write a program to get the given series in sorted label form using groupby function so that the solution is gropby iterable form ?

```python
grouped = s.groupby(s.index)
for fruit, s_obj in grouped:
    print(f"===== {fruit} =====")
    print(s_obj)
```

```
===== apples =====
apples    9
apples    2
apples    13
apples    2
apples    6
apples    13
dtype: int32
===== bananas =====
bananas   11
bananas   10
bananas   14
bananas   13
bananas    2
bananas   16
bananas    8
bananas   19
dtype: int32
===== cherries =====
cherries   17
cherries    8
cherries    8
cherries   19
cherries    7
cherries    8
dtype: int32
===== clementines =====
clementines    6
clementines   10
clementines    1
clementines   18
dtype: int32
===== oranges =====
oranges   12
oranges    4
oranges    5
dtype: int32
===== pears =====
pears    9
pears    2
pears   12
dtype: int32
```

In [58]: *#18. The DataFrame has two columns one containing names Name and the other one coffee contains integers which are the number of cups of coffe*

```python
import pandas as pd
beverages = pd.DataFrame({'Name': ['Robert', 'Melinda', 'Brenda',
                'Samantha', 'Melinda', 'Robert',
                'Melinda', 'Brenda', 'Samantha'],
            'Coffee': [3, 0, 2, 2, 0, 2, 0, 1, 3],
            'Tea':    [0, 4, 2, 0, 3, 0, 3, 2, 0]})
beverages
```

Out[58]:

	Name	Coffee	Tea
0	Robert	3	0
1	Melinda	0	4
2	Brenda	2	2
3	Samantha	2	0
4	Melinda	0	3
5	Robert	2	0
6	Melinda	0	3
7	Brenda	1	2
8	Samantha	3	0

In [59]: *#19. Write the program to calculate the average number of coffee and tea cups the persons had using gropby function in pandas?*

```python
beverages.groupby(['Name']).mean()
```

Out[59]:

	Coffee	Tea
Name		
Brenda	1.5	2.000000
Melinda	0.0	3.333333
Robert	2.5	0.000000
Samantha	2.5	0.000000

In [60]: *#Binning*
#Q20. Write a program with function 'find bin' with two list or tuple of bins of two elements to find the index

```python
    def create_bins(lower_bound, width, quantity):
        bins = []
        for low in range(lower_bound,
                    lower_bound + quantity*width + 1, width):
            bins.append((low, low+width))
        return bins
    bins = create_bins(lower_bound=10,
                width=10,
                quantity=5)
    bins
    def find_bin(value, bins):
        for i in range(0, len(bins)):
            if bins[i][0] <= value < bins[i][1]:
                return i
        return -1
    from collections import Counter
    bins = create_bins(lower_bound=50,
                width=4,
                quantity=10)
    print(bins)
    weights_of_persons = [73.4, 69.3, 64.9, 75.6, 74.9, 80.3,
                78.6, 84.1, 88.9, 90.3, 83.4, 69.3,
                52.4, 58.3, 67.4, 74.0, 89.3, 63.4]
    binned_weights = []
    for value in weights_of_persons:
        bin_index = find_bin(value, bins)
        print(value, bin_index, bins[bin_index])
        binned_weights.append(bin_index)
    frequencies = Counter(binned_weights)
    print(frequencies)
```

```
[(50, 54), (54, 58), (58, 62), (62, 66), (66, 70), (70, 74), (74, 78), (78, 82), (82, 86), (86, 90), (90, 94)]
73.4 5 (70, 74)
69.3 4 (66, 70)
64.9 3 (62, 66)
75.6 6 (74, 78)
74.9 6 (74, 78)
80.3 7 (78, 82)
78.6 7 (78, 82)
84.1 8 (82, 86)
88.9 9 (86, 90)
90.3 10 (90, 94)
83.4 8 (82, 86)
69.3 4 (66, 70)
52.4 0 (50, 54)
58.3 2 (58, 62)
67.4 4 (66, 70)
74.0 6 (74, 78)
89.3 9 (86, 90)
63.4 3 (62, 66)
Counter({4: 3, 6: 3, 3: 2, 7: 2, 8: 2, 9: 2, 5: 1, 10: 1, 0: 1, 2: 1})
```

In [61]:
```python
#Multilevel indexing
#Q21. Write the code for multilevel indexing using Pandas data structures. It's an efficient way to store and manipuimport pandas as pd
cities = ["Vienna", "Vienna", "Vienna",
"Hamburg", "Hamburg", "Hamburg",
"Berlin", "Berlin", "Berlin",
"Zürich", "Zürich", "Zürich"]
index = [cities, ["country", "area", "population",
"country", "area", "population",
"country", "area", "population",
"country", "area", "population"]]
print(index)
```

```
[['Vienna', 'Vienna', 'Vienna', 'Hamburg', 'Hamburg', 'Hamburg', 'Berlin', 'Berlin', 'Berlin', 'Zürich', 'Zürich', 'Zürich'], ['country', 'area', 'population', 'country', 'area'
, 'population', 'country', 'area', 'population', 'country', 'area', 'population']]
```

In [62]:
```python
#Q22. Write the program to sort the index using slicing operation for the given series in pandas function ?
city_series = city_series.sort_index()
print("city_series with sorted index:")
print(city_series)
print("\n\nSlicing the city_series:")
city_series["Berlin":"Vienna"]
```

city_series with sorted index:
```
Barcelona    1602386
Berlin       3562166
Bucharest    1803425
Budapest     1754000
Hamburg      1760433
London       8615246
Madrid       3165235
Milan        1350680
Munich       1493900
Paris        2273305
Rome         2874038
Vienna       1805681
Warsaw       1740119
dtype: int64
```

Slicing the city_series:
```
Out[62]:Berlin       3562166
        Bucharest    1803425
        Budapest     1754000
        Hamburg      1760433
        London       8615246
        Madrid       3165235
        Milan        1350680
        Munich       1493900
        Paris        2273305
        Rome         2874038
        Vienna       1805681
        dtype: int64
```

In [63]: #Q23. Write a program to perform swapping multidex levels using pandas library function ?
```python
import pandas as pd
city_series.sort_index(inplace=True)
city_series
```

```
Out[63]:Barcelona    1602386
        Berlin       3562166
        Bucharest    1803425
        Budapest     1754000
        Hamburg      1760433
        London       8615246
        Madrid       3165235
        Milan        1350680
        Munich       1493900
        Paris        2273305
        Rome         2874038
        Vienna       1805681
        Warsaw       1740119
        dtype: int64
```

In [64]: #Data Visualization
#24 . If given a tuple data =[100, 120, 140, 180, 200, 210, 214] ,using pandas   series function plot a line plot for the data ?
```python
import pandas as pd
data = [100, 120, 140, 180, 200, 210, 214]
s = pd.Series(data, index=range(len(data)))
s.plot()
```

Out[64]:<Axes: >
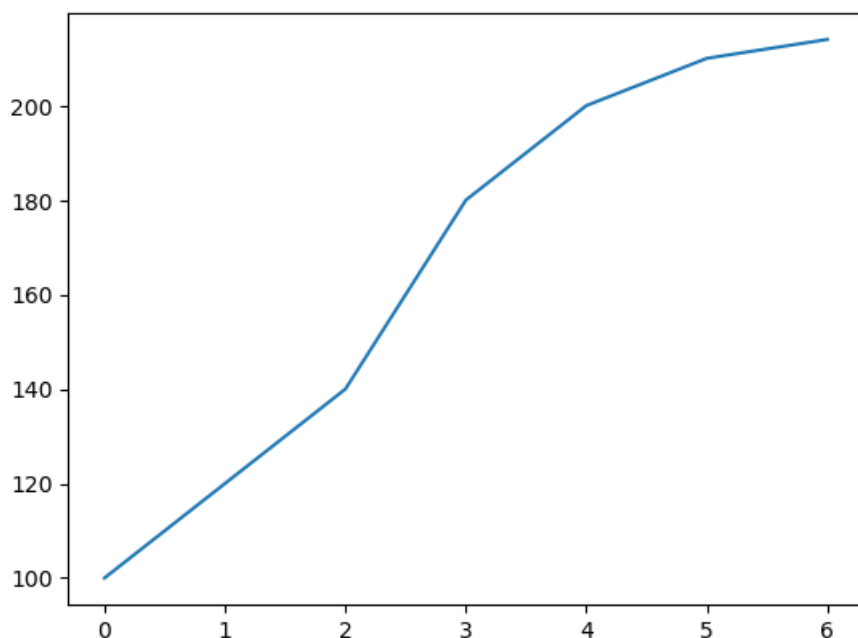
In [65]: *#25. For the defined dcitionary with the population and area figures. This dictionary can be used to create the DataFrame, which we want to use for p*

```python
import pandas as pd
cities = {"name": ["London", "Berlin", "Madrid", "Rome",
                   "Paris", "Vienna", "Bucharest", "Hamburg",
                   "Budapest", "Warsaw", "Barcelona",
                   "Munich", "Milan"],
          "population": [8615246, 3562166, 3165235, 2874038,
                         2273305, 1805681, 1803425, 1760433,
                         1754000, 1740119, 1602386, 1493900,
                         1350680],
          "area" : [1572, 891.85, 605.77, 1285,
                    105.4, 414.6, 228, 755,
                    525.2, 517, 101.9, 310.4,
                    181.8]
         }
city_frame = pd.DataFrame(cities,
                          columns=["population", "area"],
                          index=cities["name"])
print(city_frame)
```

```
          population    area
London       8615246  1572.00
Berlin       3562166   891.85
Madrid       3165235   605.77
Rome         2874038  1285.00
Paris        2273305   105.40
Vienna       1805681   414.60
Bucharest    1803425   228.00
Hamburg      1760433   755.00
Budapest     1754000   525.20
Warsaw       1740119   517.00
Barcelona    1602386   101.90
Munich       1493900   310.40
Milan        1350680   181.80
```

In [66]: *#26. Wite a program for pie chart diagram in pandas for the given series using plot function ?*

```python
import pandas as pd
fruits = ['apples', 'pears', 'cherries', 'bananas']
series = pd.Series([20, 30, 40, 10],
                   index=fruits,
                   name='series')
series.plot.pie(figsize=(6, 6))
```

Out[66]:<Axes: ylabel='series'>
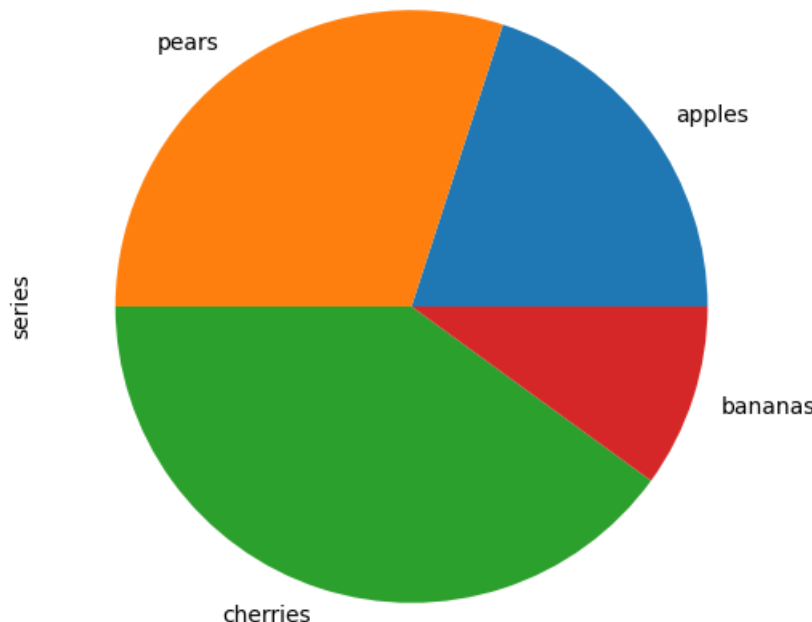
In [67]: *#Date and Time*
#27. Write a program to print the date and time using pandas date-time function?

```python
from datetime import date
x = date(1993, 12, 14)
print(x)
```

```
1993-12-14
```

In [68]: *#28. Write a program to instantiate dates in the range from January 1, 1 to December 31, 9999. This can be inquired from the attributes min and max*

```python
from datetime import date
print(date.min)
print(date.max)
```

0001-01-01
9999-12-31

In [69]: *#29. Write a program to show an output a dataframe of data and time ?*

```python
import pandas as pd
data = pd.date_range('1/1/2011', periods = 10, freq ='H')

data
```

Out[69]:DatetimeIndex(['2011-01-01 00:00:00', '2011-01-01 01:00:00',
               '2011-01-01 02:00:00', '2011-01-01 03:00:00',
               '2011-01-01 04:00:00', '2011-01-01 05:00:00',
               '2011-01-01 06:00:00', '2011-01-01 07:00:00',
               '2011-01-01 08:00:00', '2011-01-01 09:00:00'],
              dtype='datetime64[ns]', freq='H')

In [70]: *#Time Series*
30. Using time series **in** pandas write the program to find the index consisting of time stamps**?**

```python
import numpy as np
import pandas as pd
from datetime import datetime, timedelta as delta
ndays = 10
start = datetime(2017, 3, 31)
dates = [start - delta(days=x) for x in range(0, ndays)]
values = [25, 50, 15, 67, 70, 9, 28, 30, 32, 12]
ts = pd.Series(values, index=dates)
ts
```

Object `stamps` not found.
Out[70]:2017-03-31   25
        2017-03-30   50
        2017-03-29   15
        2017-03-28   67
        2017-03-27   70
        2017-03-26   9
        2017-03-25   28
        2017-03-24   30
        2017-03-23   32
        2017-03-22   12
        dtype: int64

In [71]: 31. Write a program to using a function to **return** the values lying **in** the given time duration**?**

```python
import pandas as pd
sr = pd.Series([11, 21, 8, 18, 65, 18, 32, 10, 5, 32, None])
index_ = pd.date_range('2010-10-09 08:45', periods = 11, freq ='H')
sr.index = index_
print(sr)
```

Object `duration` not found.
2010-10-09 08:45:00   11.0
2010-10-09 09:45:00   21.0
2010-10-09 10:45:00   8.0
2010-10-09 11:45:00   18.0
2010-10-09 12:45:00   65.0
2010-10-09 13:45:00   18.0
2010-10-09 14:45:00   32.0
2010-10-09 15:45:00   10.0
2010-10-09 16:45:00   5.0
2010-10-09 17:45:00   32.0
2010-10-09 18:45:00   NaN
Freq: H, dtype: float64

In [72]: *#Unit V*
#VISUALIZATION WITH MATPLOTLIB
#Simple linne and scatter plot

In [73]: *#1.Write a python code for simple line plot .*

```python
import matplotlib.pyplot as plt
plt.plot([-1, -4.5, 16, 23, 78, 22, 3])
plt.show()
```
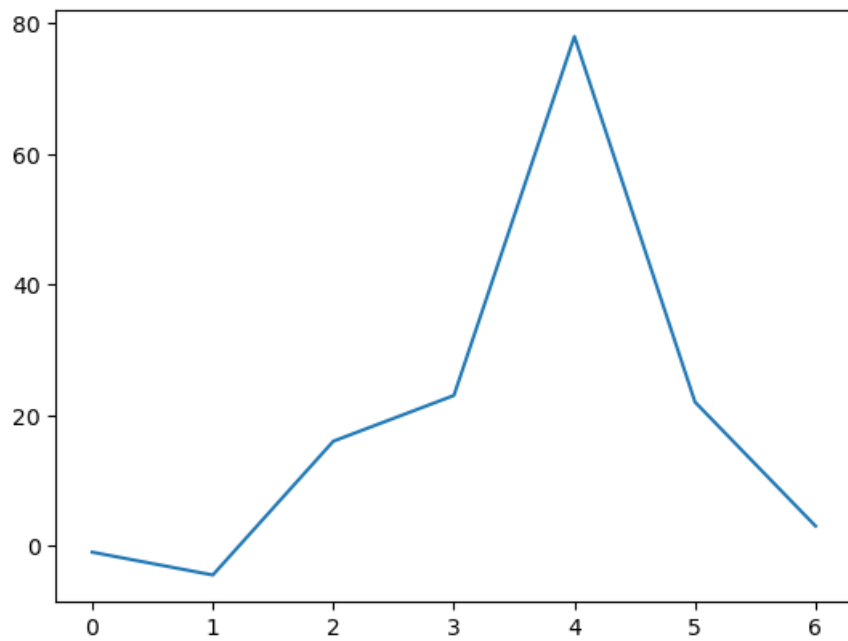
In [74]: *#2.Write a python code for scatter plot .*
```python
import matplotlib.pyplot as plt
plt.plot([-1, -4.5, 16, 23, 78, 22, 3], "ob")
plt.show()
```
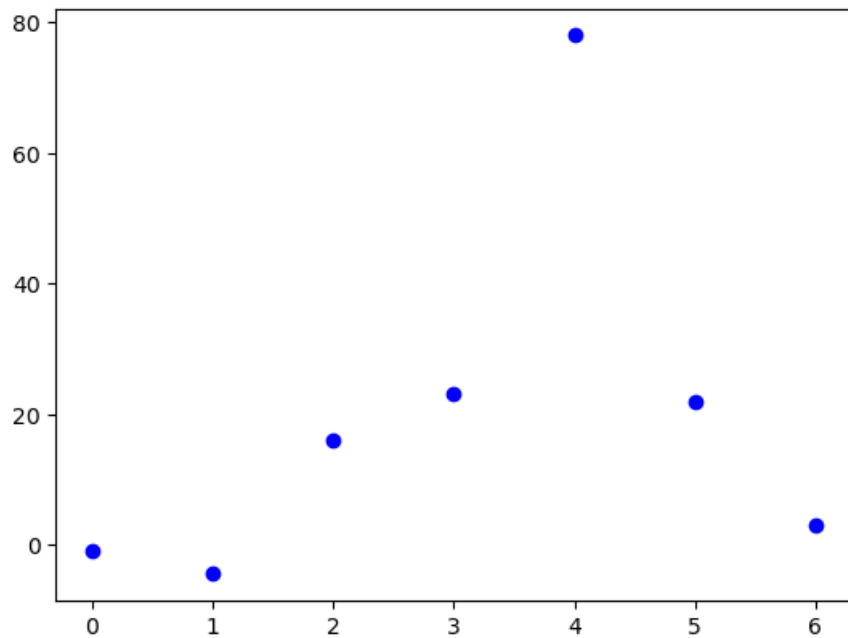

In [76]: *#3.Write a python code for line plot and display with axis name with title.*
#(i)x-axis named as "day"
#(ii)y-axis named as "Temperature in Celsius."
#(iii) title named as "Temperature Graph."

```python
import matplotlib.pyplot as plt
days = range(1, 9)
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
fig, ax = plt.subplots()
ax.plot(days, celsius_values)
ax.set(xlabel='Day',
       ylabel='Temperature in Celsius',
       title='Temperature Graph')
```
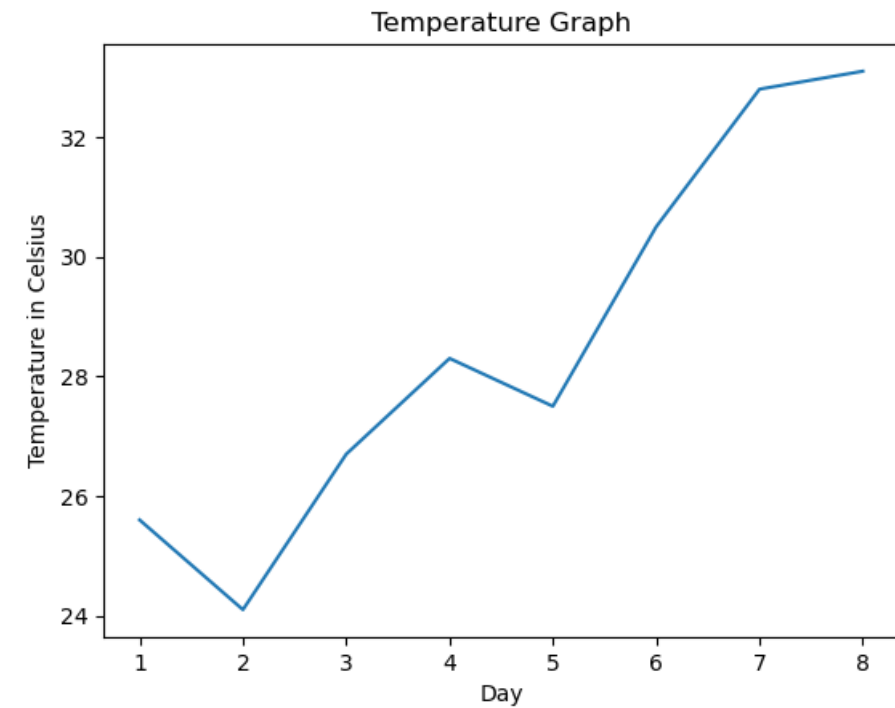
In [77]: *#4.Write a python code for multiple plot(scatter,line).*

```python
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_min = [19.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
celsius_max = [24.8, 28.9, 31.3, 33.0, 34.9, 35.6, 38.4, 39.2]
fig, ax = plt.subplots()
ax.set(xlabel='Day',
    ylabel='Temperature in Celsius',
    title='Temperature Graph')
ax.plot(days, celsius_min,
    days, celsius_min, "oy",
    days, celsius_max,
    days, celsius_max, "or")
```
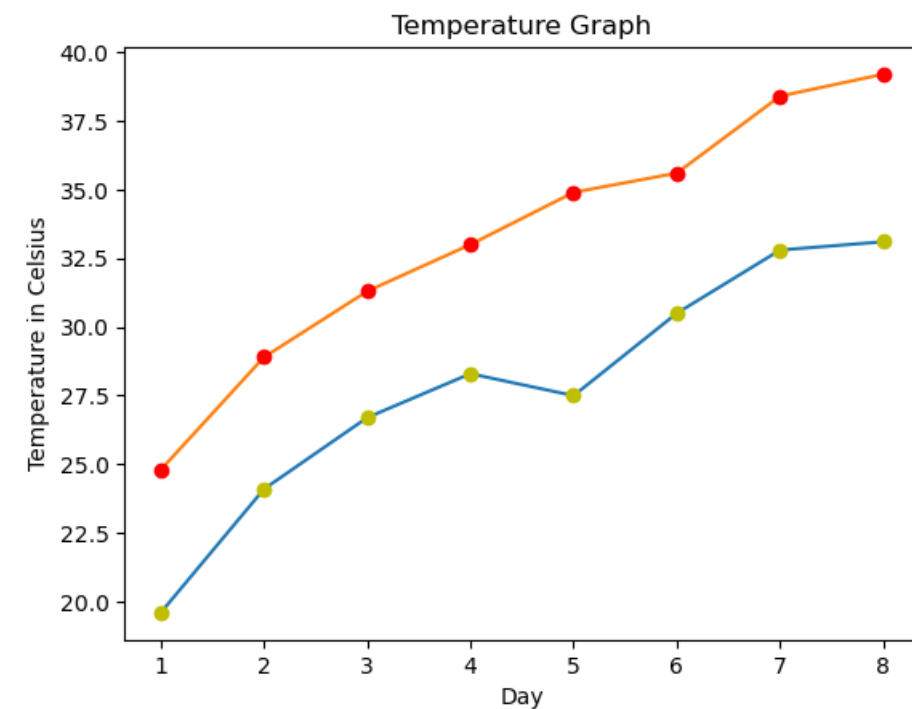
In [78]: *#Bar Chart*
#5.Write a python code to display the bar plot.

```python
import matplotlib.pyplot as plt
import numpy as np
years = [str(year) for year in range(2010, 2021)]
visitors = (1241, 50927, 162242, 222093,
        665004, 2071987, 2460407, 3799215,
```

```
                5399000, 5474016, 6003672)
  plt.bar(years, visitors, color="green")
  plt.xlabel("Years")
  plt.ylabel("Values")
  plt.title("Bar Chart ")
  plt.plot()
  plt.show()
```

Bar Chart
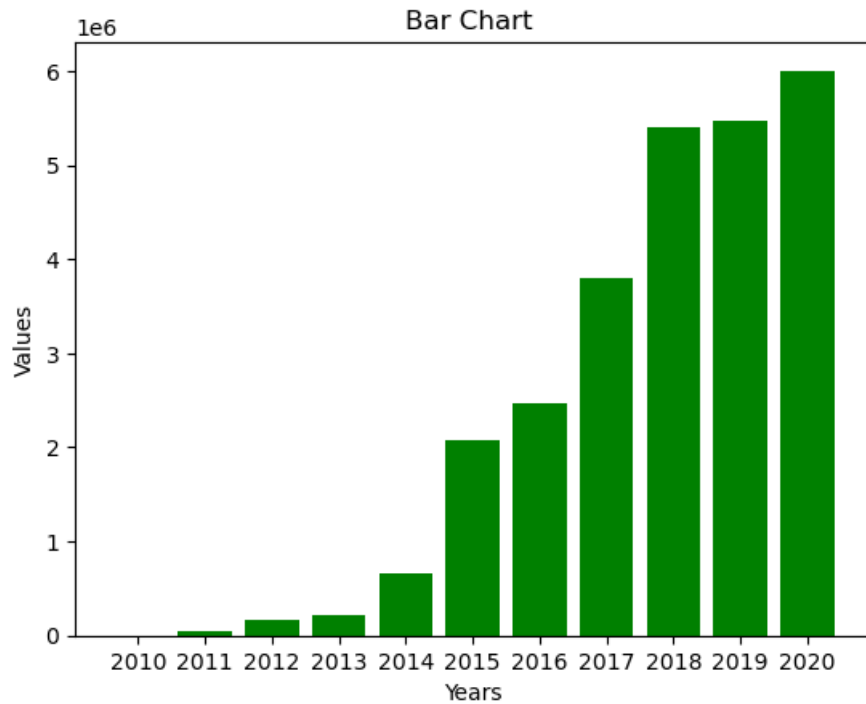
In [79]: *#7.write a python code to display the bar plot in vertical order .*
```
  import matplotlib.pyplot as plt
  import numpy as np
  import matplotlib.pyplot as plt
  # restore default parameters:
  plt.rcdefaults()
  fig, ax = plt.subplots()
  personen = ('Michael', 'Dorothea', 'Robert', 'Bea', 'Uli')
  y_pos = np.arange(len(personen))
  cups = (15, 22, 24, 39, 12)
  ax.barh(y_pos, cups, align='center',
        color='green', ecolor='black')
  ax.set_yticks(y_pos)
  ax.set_yticklabels(personen)
  ax.invert_yaxis()
  ax.set_xlabel('Cups')
  ax.set_title('Coffee Consumption')
  plt.show()
```

Coffee Consumption
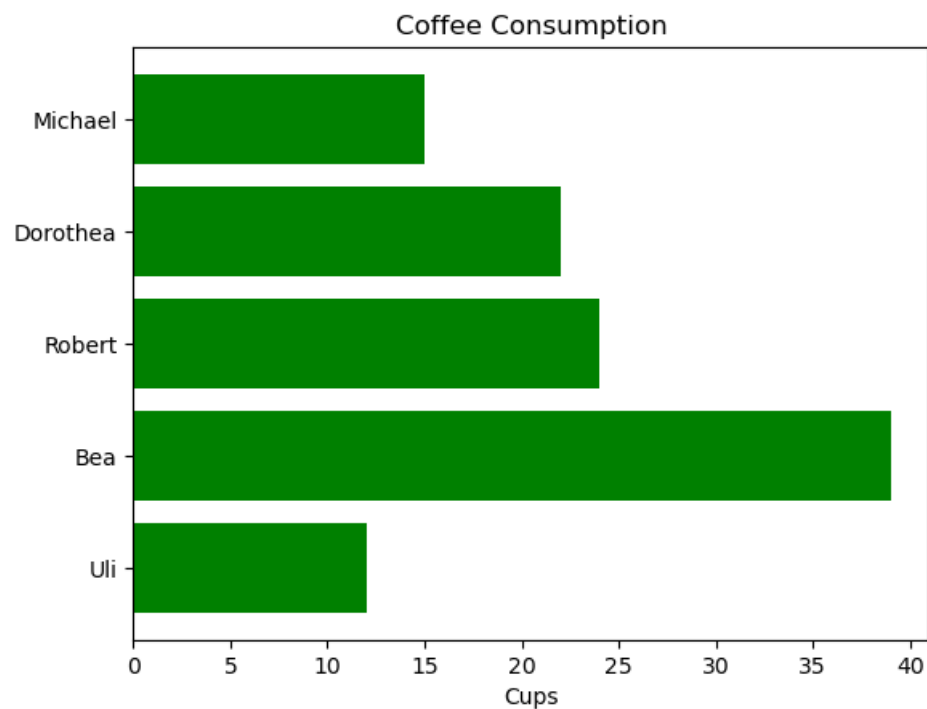
In [80]: *#8.Write a python code for grouped bar charts .*

```python
import matplotlib.pyplot as plt
import numpy as np

last_week_cups = (20, 35, 30, 35, 27)
this_week_cups = (25, 32, 34, 20, 25)
names = ['Mary', 'Paul', 'Billy', 'Franka', 'Stephan']

fig = plt.figure(figsize=(6,5), dpi=200)
left, bottom, width, height = 0.1, 0.3, 0.8, 0.6
ax = fig.add_axes([left, bottom, width, height])

width = 0.35
ticks = np.arange(len(names))
ax.bar(ticks, last_week_cups, width, label='Last week')
ax.bar(ticks + width, this_week_cups, width, align="center",
    label='This week')

ax.set_ylabel('Cups of Coffee')
ax.set_title('Coffee Consummation')
ax.set_xticks(ticks + width/2)
ax.set_xticklabels(names)

ax.legend(loc='best')
plt.show()
```
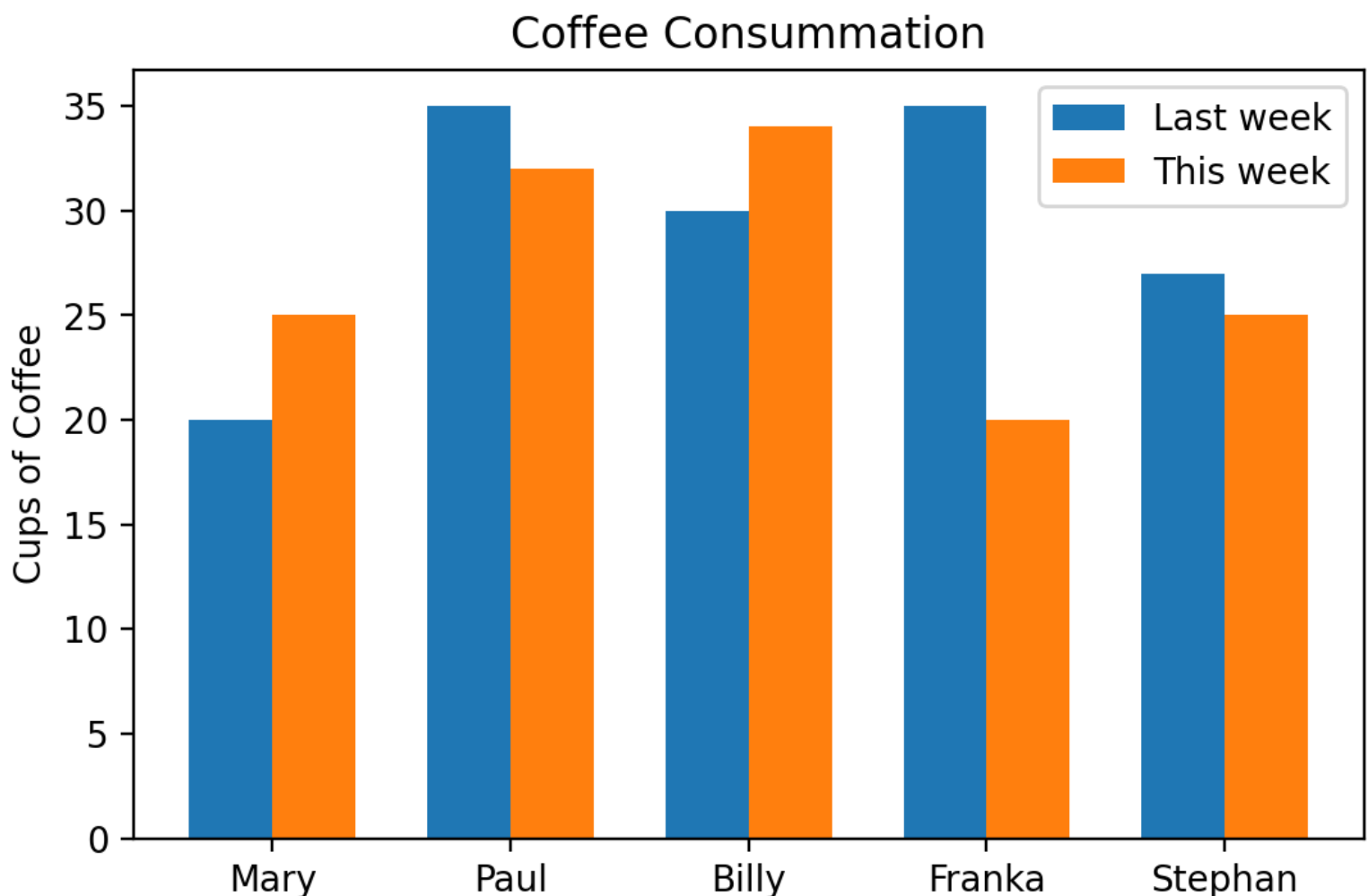

In [81]:
```python
#9.Write a python code to display the stacked bar  chart .
import matplotlib.pyplot as plt
import numpy as np

coffee = np.array([5, 5, 7, 6, 7])
tea = np.array([1, 2, 0, 2, 0])
water = np.array([10, 12, 14, 12, 15])
names = ['Mary', 'Paul', 'Billy', 'Franka', 'Stephan']

fig = plt.figure(figsize=(6,5), dpi=200)
left, bottom, width, height = 0.2, 0.1, 0.7, 0.8
ax = fig.add_axes([left, bottom, width, height])

width = 0.35
ticks = np.arange(len(names))
ax.bar(ticks, tea, width, label='Coffee', bottom=water+coffee)
ax.bar(ticks, coffee, width, align="center", label='Tea',
    bottom=water)
ax.bar(ticks, water, width, align="center", label='Water')
```
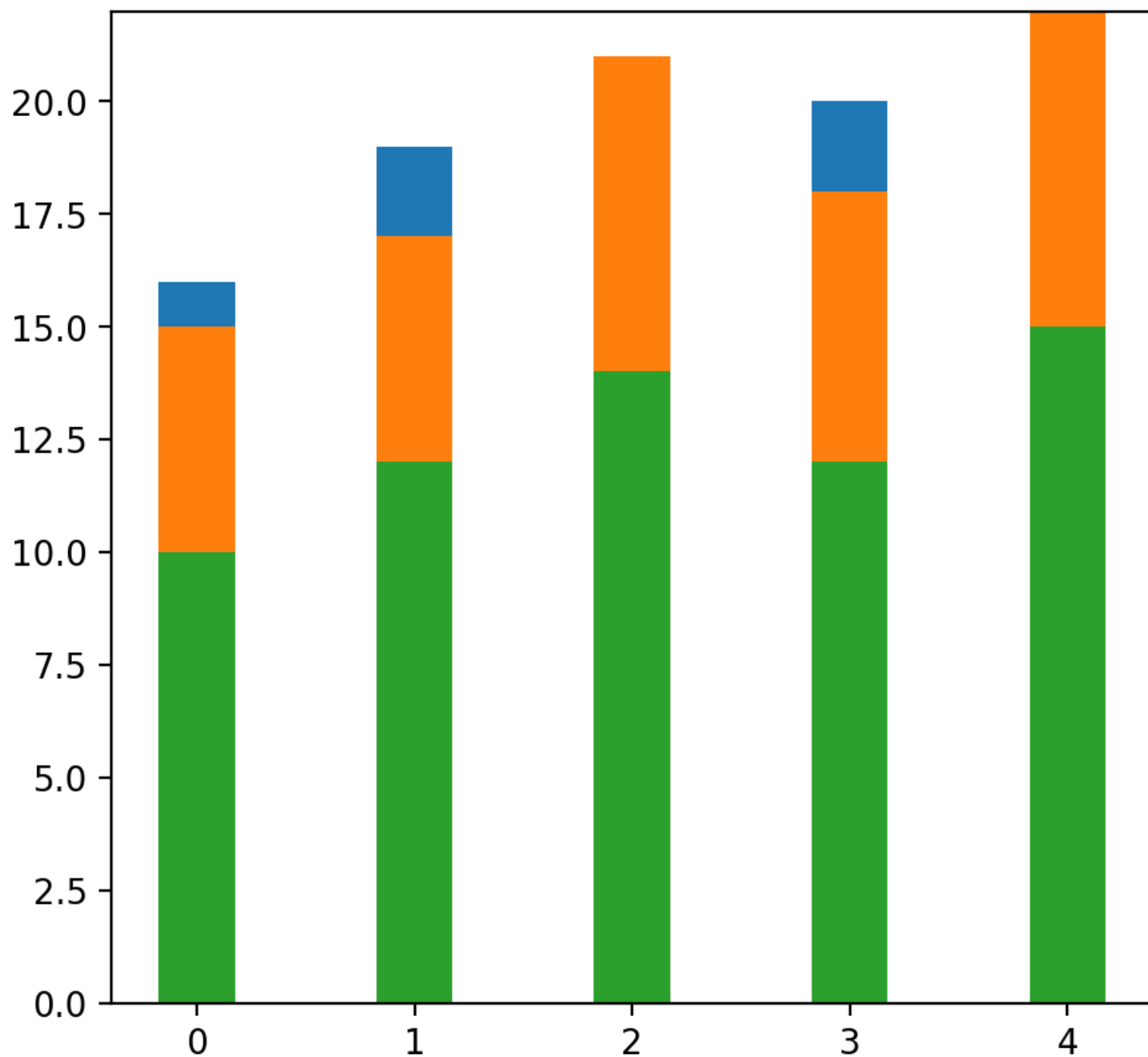
In [82]: *#10.Write a python code for histogram .*

```python
import matplotlib.pyplot as plt
import numpy as np
gaussian_numbers = np.random.normal(size=10000)
gaussian_numbers
plt.hist(gaussian_numbers, bins=20)
plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```
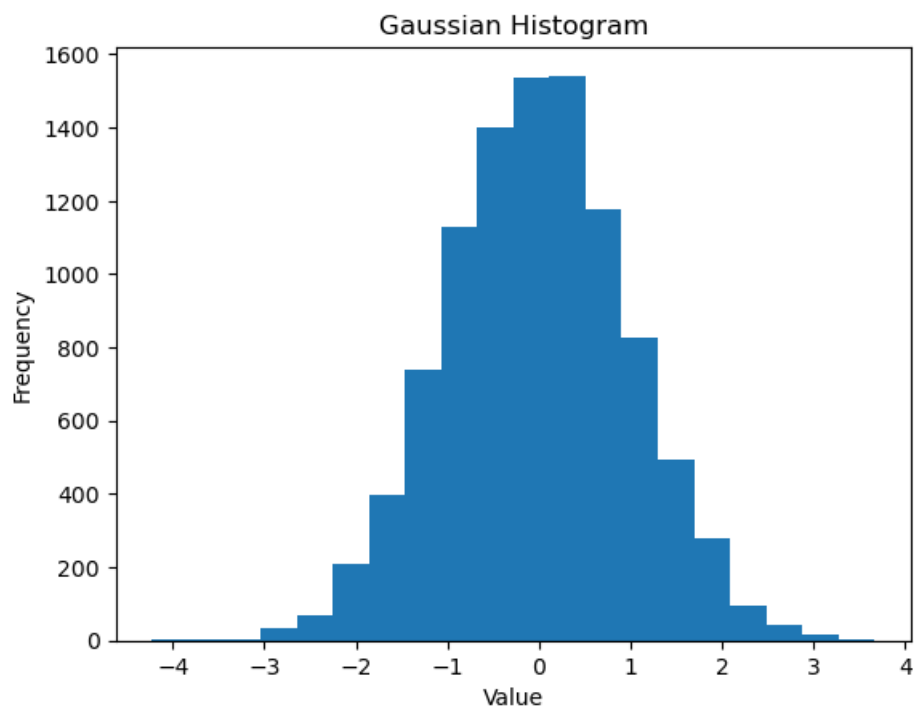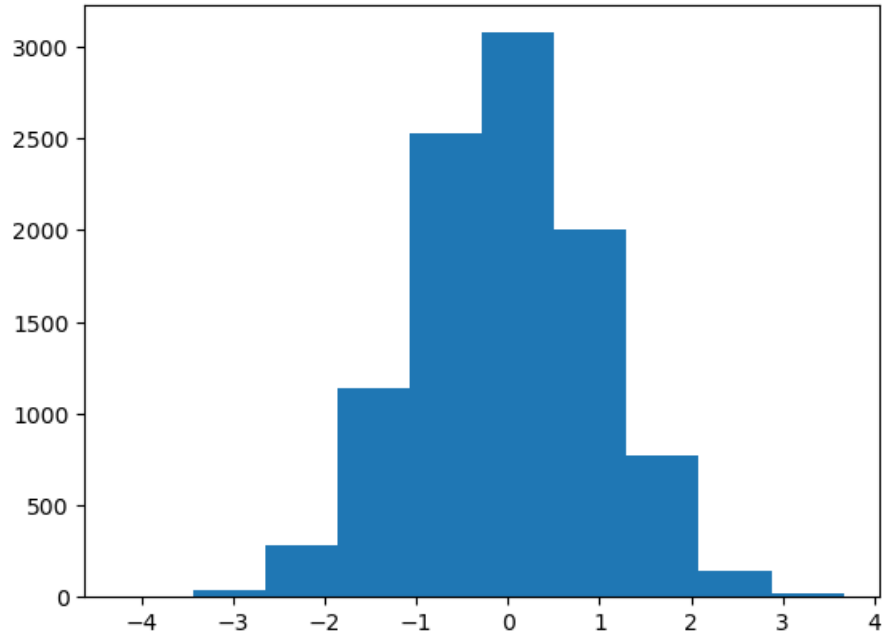
Gaussian Histogram

#11.Write a python code to display the histogram using binnings .

```python
n, bins, patches = plt.hist(gaussian_numbers)
print("n: ", n, sum(n))
print("bins: ", bins)
for i in range(len(bins)-1):
    print(bins[i+1] -bins[i])
print("patches: ", patches)
print(patches[1])
print(patches[2])
```

```
n: [   6.   38.  278. 1136. 2530. 3077. 2002.  774.  139.   20.] 10000.0
bins: [-4.22652293 -3.43746393 -2.64840493 -1.85934593 -1.07028693 -0.28122793
 0.50783107  1.29689007  2.08594907  2.87500807  3.66406707]
0.7890589994226826
0.7890589994226826
0.7890589994226831
0.7890589994226826
0.7890589994226826
0.7890589994226831
0.7890589994226822
0.7890589994226831
0.7890589994226831
0.7890589994226826
patches:  <BarContainer object of 10 artists>
Rectangle(xy=(-3.43746, 0), width=0.789059, height=38, angle=0)
Rectangle(xy=(-2.6484, 0), width=0.789059, height=278, angle=0)
```


#12.Write a python code to display histogram to increase the no.of.binnings

```python
plt.hist(gaussian_numbers, bins=100)
plt.show()
```
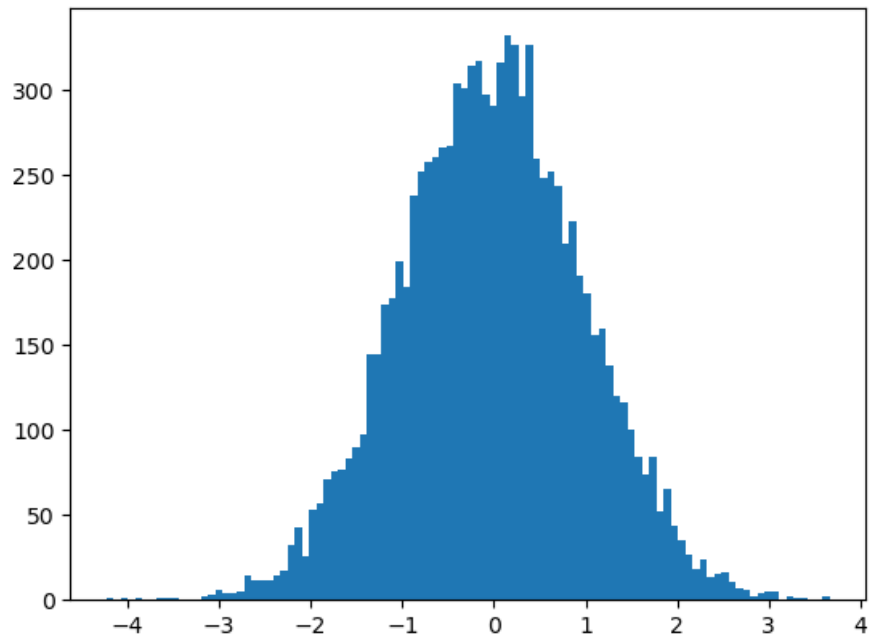
#13.Write a python code to display the histogram in horizontal orientation.

```python
plt.hist(gaussian_numbers,
    bins=100,
    orientation="horizontal")
plt.show()
```
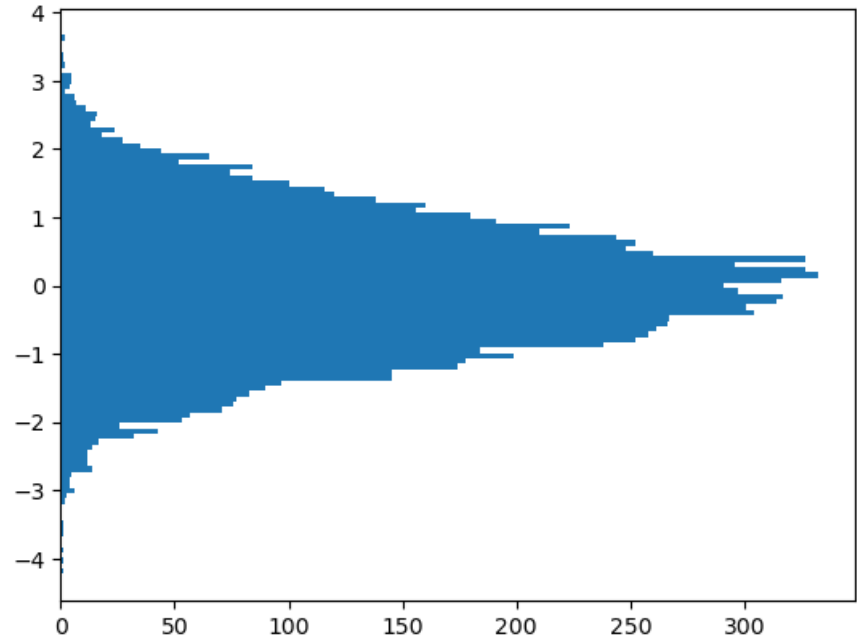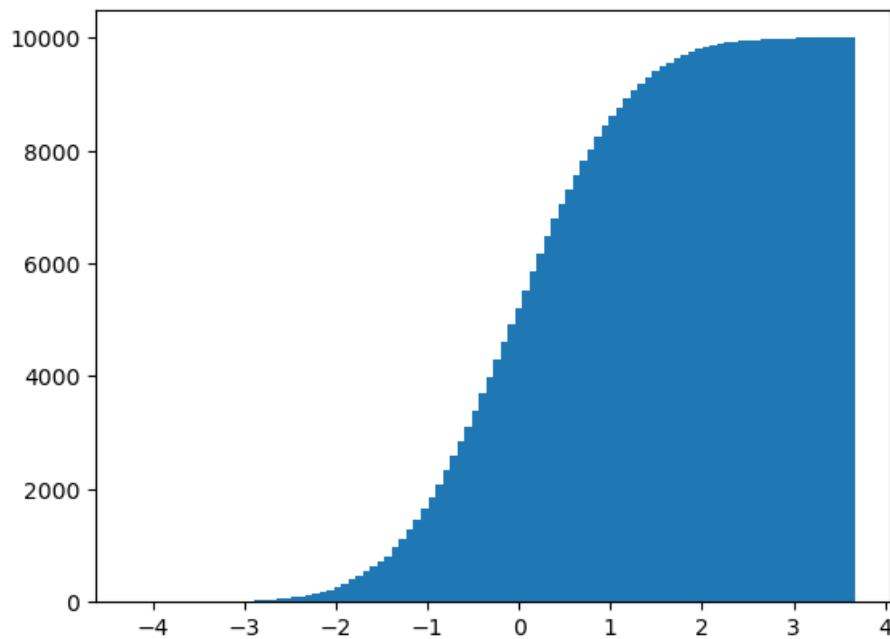

#14.Write a python code to display the histogram with edge color.

```python
plt.hist(gaussian_numbers,
    bins=100,
    density=True,
    stacked=True,
    edgecolor="#6A9662",
    color="#DDFFDD")
plt.show()
```
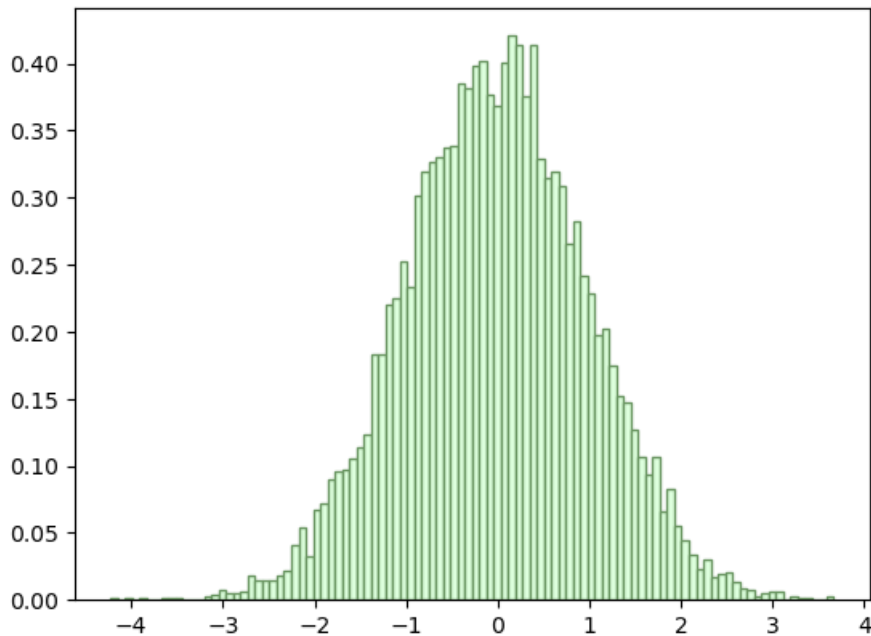
In [88]: *#15.Write a python code to display the histogram in cumulative using bins .*
```
plt.hist(gaussian_numbers,
    bins=100,
    stacked=True,
    cumulative=True)
plt.show()
```


In [89]: *#16.Write a python code to display scatter plot using markers.*
```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0, 11)
y1 = np.random.randint(2, 7, (11,))
y2 = np.random.randint(9, 14, (11,))
y3 = np.random.randint(15, 25, (11,))
plt.scatter(x, y1)
plt.scatter(x, y2, marker='v', color='r')
plt.scatter(x, y3, marker='^', color='m')
plt.title('Scatter Plot Example')
plt.show()
```

Scatter Plot Example
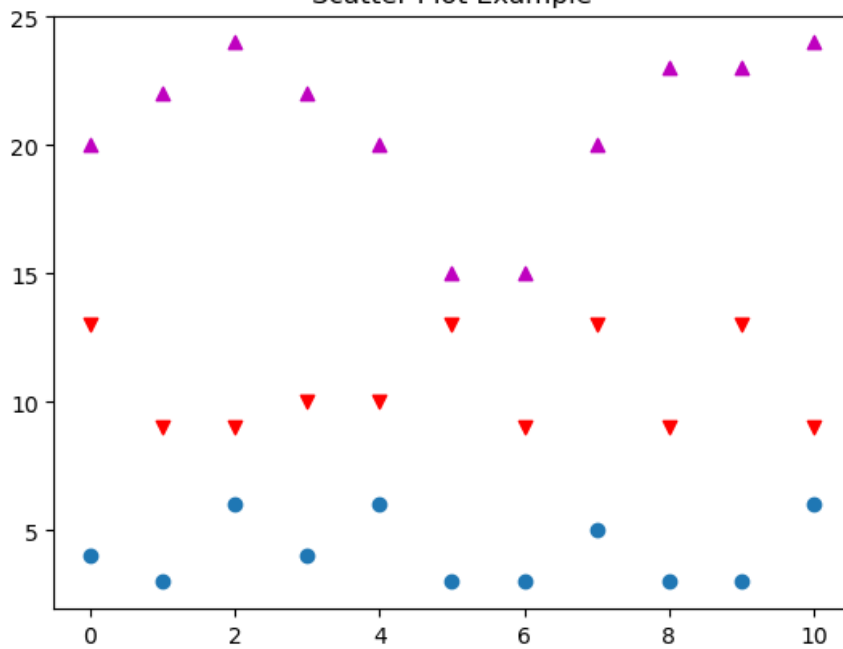
In [90]:
```
#17.Write a python code for contour plot without charts .
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
n, m = 7, 7
start = -3
x_vals = np.arange(start, start+n, 1)
y_vals = np.arange(start, start+m, 1)
X, Y = np.meshgrid(x_vals, y_vals)
print(X)
print(Y)
```

```
[[-3 -2 -1  0  1  2  3]
 [-3 -2 -1  0  1  2  3]
 [-3 -2 -1  0  1  2  3]
 [-3 -2 -1  0  1  2  3]
 [-3 -2 -1  0  1  2  3]
 [-3 -2 -1  0  1  2  3]
 [-3 -2 -1  0  1  2  3]]
[[-3 -3 -3 -3 -3 -3 -3]
 [-2 -2 -2 -2 -2 -2 -2]
 [-1 -1 -1 -1 -1 -1 -1]
 [ 0  0  0  0  0  0  0]
 [ 1  1  1  1  1  1  1]
 [ 2  2  2  2  2  2  2]
 [ 3  3  3  3  3  3  3]]
```

In [91]:
```
#Q18.Write a python code to display the scatter diagram using contour plot.
#(i) X axis named as "x"
#(ii) Y-axis named as "y"
#(iii) Titled named as "Scatter using contour plot"
fig, ax = plt.subplots()
ax.scatter(X, Y, color="green")
ax.set_title('Scatter using contour plot')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```
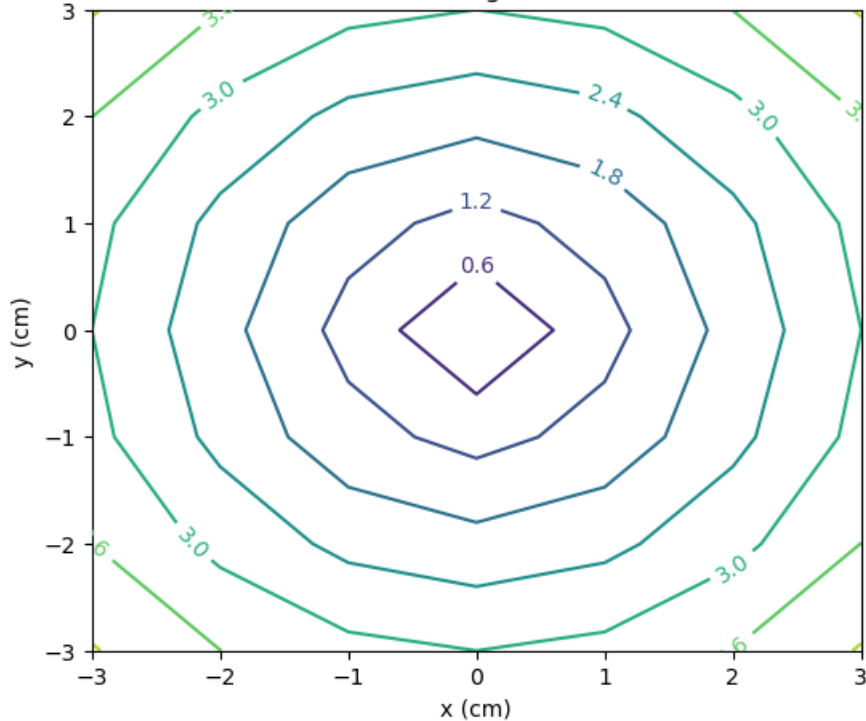
Scatter using contour plot
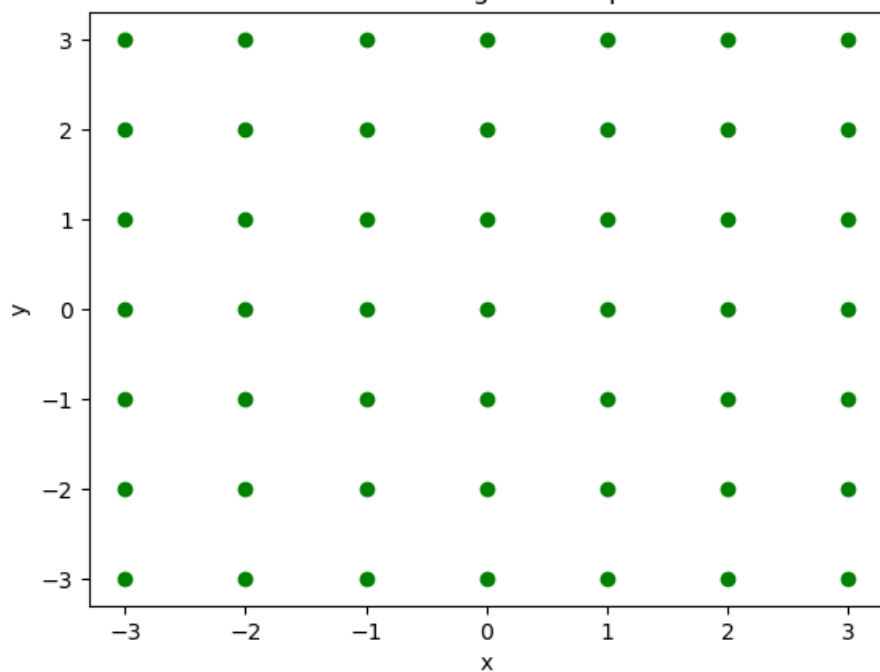
In [92]:
```
#Q19.Write a python code to display contour plot using clabel method.
fig = plt.figure(figsize=(6,5))
left, bottom, width, height = 0.1, 0.1, 0.8, 0.8
ax = fig.add_axes([left, bottom, width, height])
Z = np.sqrt(X**2 + Y**2)
cp = ax.contour(X, Y, Z)
ax.clabel(cp, inline=True,
fontsize=10)
ax.set_title('Contour Plot Using clabel method')
ax.set_xlabel('x (cm)')
ax.set_ylabel('y (cm)')
plt.show()
```
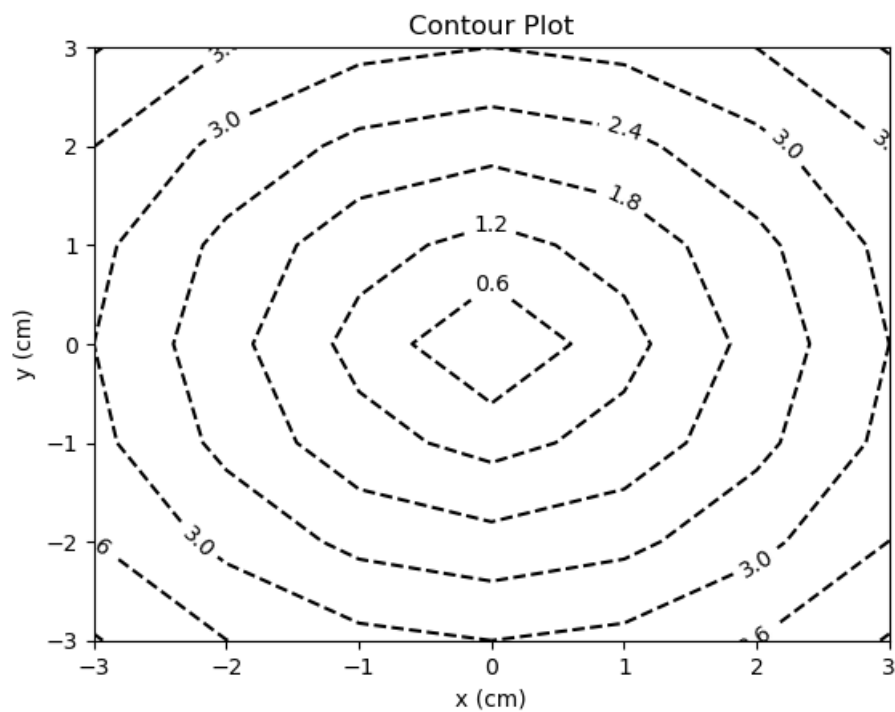
Contour Plot Using clabel method

In [93]:
```
#Q20.Write a python code for changing the linestyle= "dashed line" and color="black" in contour plot.
import matplotlib.pyplot as plt
plt.figure()
cp = plt.contour(X, Y, Z, colors='black', linestyles='dashed')
plt.clabel(cp, inline=True,
fontsize=10)
plt.title('Contour Plot')
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.show()
```

Contour Plot

#Q21.Write a python code for filled color using contour plot .

```python
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure(figsize=(6,5))
left, bottom, width, height = 0.1, 0.1, 0.8, 0.8
ax = fig.add_axes([left, bottom, width, height])
start, stop, n_values = -8, 8, 800
x_vals = np.linspace(start, stop, n_values)
y_vals = np.linspace(start, stop, n_values)
X, Y = np.meshgrid(x_vals, y_vals)
Z = np.sqrt(X**2 + Y**2)
cp = plt.contourf(X, Y, Z)
plt.colorbar(cp)
ax.set_title('Contour Plot')
ax.set_xlabel('x (cm)')
ax.set_ylabel('y (cm)')
plt.show()
```
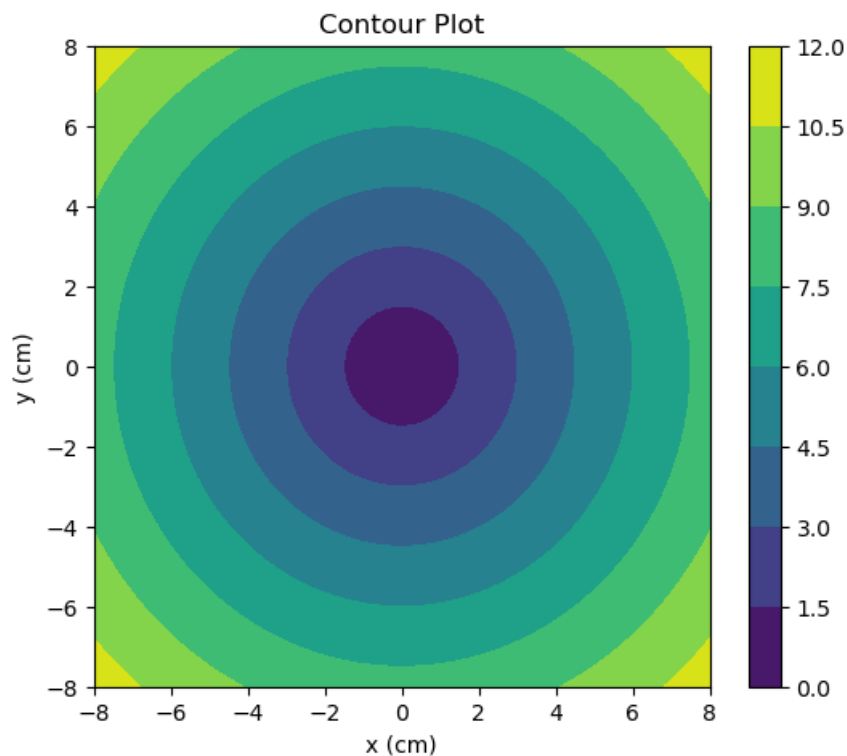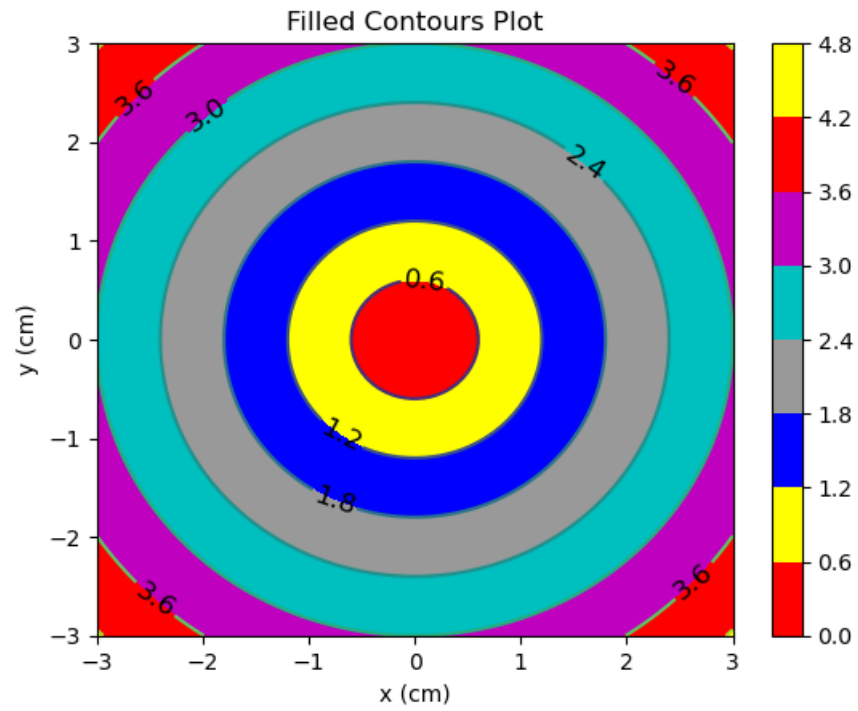

Contour Plot

#Q22.Write a python code for to displaty individual color using contour plot.

```python
import numpy as np
import matplotlib.pyplot as plt
xlist = np.linspace(-3.0, 3.0, 100)
ylist = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X**2 + Y**2)
plt.figure()
```

```
contour = plt.contour(X, Y, Z)
plt.clabel(contour, colors = 'k', fmt = '%2.1f', fontsize=12)
c = ('#ff0000', '#ffff00', '#0000FF', '0.6', 'c', 'm')
contour_filled = plt.contourf(X, Y, Z, colors=c)
plt.colorbar(contour_filled)
plt.title('Filled Contours Plot')
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.savefig('contourplot_own_colours.png', dpi=300)
plt.show()
```


In [ ]: