E-Record Submission

A Report on the coursework submitted as part of Continuous Internal Assessment for the course Calculus Using Python (MAT111-2)

Submitted by

Vishal Das (2341350)

Under the supervision of

Dr A. John Kaspar

Department of Mathematics
Christ(Deemed to be University)
BENGALURU

April 2024

Contents

Calculus Using Python (MAT111-2)

Vishal Das
2341350
BSc Data Science & Mathematics

April 21, 2024

1 Introduction

[1]:
```
print ("Hello World!")
```

```
Hello World!
```

1.1 Arithmetic Operations of integers, floating points and complex numbers

1.1.1 + (Addition)

[2]:
```
print(2+3)
```

```
5
```

1.1.2 - (Subtraction)

[3]:
```
print(2-3)
```

```
-1
```

1.1.3 * (Multiplication)

[4]:
```
print (2*3)
```

```
6
```

1.1.4 ** (Power)

[5]:
```
print(6**5)
```

```
7776
```

1.1.5 / (Division)

[7]:
```
print(13/8)
```

```
1.625
```

1.1.6 // (Floor Division)

[9]:
```
print(10//4)
```

```
2
```

1.1.7 % (Modulus)-gives only remainder

[10]:
```
print(17%6)
```

```
5
```

1.1.8 abs(real number) -gives absolute value

[12]:
```
print (abs(2-13))
```

```
11
```

1.1.9 Complex Numbers

[13]:
```
print (2+3j)

print (type(2+3j))
```

```
(2+3j)
<class 'complex'>
```

[14]:
```
print ((4+4j)+(3+2j))
```

```
(7+6j)
```

[15]:
```
print ((4+4j)-(3+2j))
```

```
(1+2j)
```

[16]:
```
print ((4+4j)*(3+2j))
```

```
(4+20j)
```

[17]:
```
print ((4+4j)**2)
```

```
32j
```

1.1.10 z.real, z.imaginary, z.conjugate

[18]:
```
print ((2+3j).real,(2+3j).imag,(2+3j).conjugate(),abs(2+3j))
```

```
2.0 3.0 (2-3j) 3.605551275463989
```

1.1.11 float

[21]:
```
print (type(7.312+1.327))
```

```
<class 'float'>
```

1.1.12 Fractions

[22]:
```
from fractions import Fraction
print (Fraction(3,4)+1+1.5)
```

```
3.25
```

[23]:
```
print (Fraction(3,4)+1+Fraction(3,4))
```

```
5/2
```

1.2 Getting Input

[24]:
```
a=input()
```

```
 5
```

[25]:
```
b=input()
print ("The type of data is: ", type(b))
```

```
 3
```

```
The type of data is:  <class 'str'>
```

[27]:
```
x=int(input())
print ("The type of data is: ", type(x))
```

```
 3
```

```
The type of data is:  <class 'int'>
```

[28]:
```
p=int(input())
```

```
 55
```

[29]:
```
q=int(input())
```

```
 100
```

[30]:
```
print (p+q)
print (p-q)
print (p*q)
```

```
155
-45
5500
```

1.3 Excercises

1. Write a Python program that takes a user input (positive integers). The program should then iterate through nubers from 1 to that number(both inclusive) and print "tik" if the current number is divisible by 3; otherwise it should print "tok"

[31]:
```
a=int(input())
```

```
 5
```

[32]:
```
for i in range (1, a+1):
    if i%3==0:
        print ("tik")
    else:
        print ("tok")
```

```
tok
tok
```

```
tik
tok
tok
```

2. Write a program to generate all the prime numbers till 100

[33]:
```
r=100
for a in range(2,r+1):
    k=0
    for i in range(2,a//2+1):
        if(a%i==0):
            k=k+1
    if(k<=0):
        print(a, end = " ");
```

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

2 Unit 1: Fundamentals of Python Programming

2.1 List

Python lists are ordered collections of items, allowing for storage and manipulation of heterogeneous data types within a single variable. Lists are mutable, meaning their elements can be modified after creation, and they support various operations like indexing, slicing, appending, and concatenating.

[34]:
```
# Creating Python List
List=[1,2,3,"apple",2.3,"xyz"]
print("\nA simple Python List, L: ")
print(List)
```

```
A simple Python List, L:
[1, 2, 3, 'apple', 2.3, 'xyz']
```

[35]:
```
#Creating a multidimensional List, that is nesting a list inside a list
#(By nesting a list inside a list)
List2 =␣
 ↪[['Apple','Mango'],['Tomato','Potato','Chillies'],['Rices','Pulses'],['Soap','Brush','Det
print("\nThe multi-dimensional list in Python, Grocery: ")
print(List2)
```

```
The multi-dimensional list in Python, Grocery:
[['Apple', 'Mango'], ['Tomato', 'Potato', 'Chillies'], ['Rices', 'Pulses'],
['Soap', 'Brush', 'Detergent']]
```

[36]:
```
#Accessing a element from the list using index number
print ("Accessing element from the list")
print(List[0])
print(List[2])

print(List2[2])
print(List2[0])
print(List2[0][1])
```

```
Accessing element from the list
1
3
['Rices', 'Pulses']
['Apple', 'Mango']
Mango
```

[38]:
```
#accessing an element using negative indexing
print("Accessing an element using negative indexing")
list3=["a","b",3,"Chocolates", 2.3, "juice"]
print(list3)
#print the second element of list using negative indexing
print(list3[-5])
#print the third last element of the list
print(list3[-3])
```

```
Accessing an element using negative indexing
['a', 'b', 3, 'Chocolates', 2.3, 'juice']
b
Chocolates
```

[39]:
```
#initialise list
list7=[1,2,3,4,5,6,7,8,9]
print("Original List:\n",list7)

#Syntax of slicing is: list[initial:end:indexjump]
print("\nSlicedList:")
```

```
#Display sliced list
print(list7[3:9:2])
#Display sliced list
print(list7[::2])
#Display sliced list
print(list7[::])
```

```
Original List:
 [1, 2, 3, 4, 5, 6, 7, 8, 9]

SlicedList:
[4, 6, 8]
[1, 3, 5, 7, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[40]:
```
#intialise list
lst=[50,70,30,20,90,10,50]

#Display sliced list using negative indexing
print(lst[-7::1])
print(lst[8:1:-2])
```

```
[50, 70, 30, 20, 90, 10, 50]
[50, 90, 30]
```

[41]:
```
#initialise list
List=[1,2,3,4,5,6,7,8,9]

#Show original list
print("Original List: \n",List)

print("\nSliced Lists: ")

#Creating new list
newList = List[:3]+List[7:]

#Display sliced list
print(newList)

#Changing existing list
List=List[::2]+List[1::2]
```

```
#Display Sliced list
print(List)
```

```
Original List:
 [1, 2, 3, 4, 5, 6, 7, 8, 9]

Sliced Lists:
[1, 2, 3, 8, 9]
[1, 3, 5, 7, 9, 2, 4, 6, 8]
```

2.2 Tuple

Similar to a list, a Python tuple is a collection of Python objects; however, once a tuple is generated, its elements cannot be added or withdrawn. A Tuple can have different kinds of elements, much like a List.

Tuples in Python are constructed by grouping a sequence of values together using parentheses or not, and then separating the values with a "comma."

Remark: It is possible to generate tuples with just one element, but it requires some effort. It takes a trailing "comma" to complete the tuple; just one element in the parenthesis is insufficient.

[43]:
```
#Creating a tuple with the use of list
List_1=[2,1,7,0,66,8888,0.5]
print(List_1)
print("\nTuple using list: ")
Tuple = tuple(List_1)
print(Tuple)
```

```
[2, 1, 7, 0, 66, 8888, 0.5]

Tuple using list:
(2, 1, 7, 0, 66, 8888, 0.5)
```

[44]:
```
#Creating a tuple with the use of strings

tuple0=('We','are','learning','python')
print("\nTuple with the use of string: ")
print(tuple0)
```

```
Tuple with the use of string:
('We', 'are', 'learning', 'python')
```

[45]:
```
#Creating a tuple with the use of integers

tuple1 = (1,2,3,4,5)
print("\nTuple with the use of integers: ")
print(tuple1)
```

```
Tuple with the use of integers:
(1, 2, 3, 4, 5)
```

[46]:
```
#Creating a tuple with the use of integers and strings

tuple2=('This', 'is','my','registration','number', 2341350)
print("\nTuple with the use of integers and strings: ")
print(tuple2)
```

```
Tuple with the use of integers and strings:
('This', 'is', 'my', 'registration', 'number', 2341350)
```

[47]:
```
#Creating a tuple with the use of list
list_1=[2,1,7,0,66,8888,0.5]
print(list_1)
print("\nTuple using list: ")
Tuple = tuple(list_1)
print(Tuple)
```

```
[2, 1, 7, 0, 66, 8888, 0.5]

Tuple using list:
(2, 1, 7, 0, 66, 8888, 0.5)
```

[48]:
```
#Accessing element using indexing
print("First element of tuple: ")
print (Tuple[0])

#Accessing element from second last using negative indexing
print("\nSecond last element of tuple")
print(Tuple[-2])
```

```
print("\nLast element of tuple")
print(Tuple[-1])
```

```
First element of tuple:
2

Second last element of tuple
8888

Last element of tuple
0.5
```

[56]:
```
#Concatenation of two tuples
tup1=(1,2,3)
tup2=(4,5,6)
result = tup1 + tup2
print(result)
```

```
(1, 2, 3, 4, 5, 6)
```

[58]:
```
#Adding corresponding elements of two tuples and storing the results in a␣
↪list
tup1=(1,2,3)
tup2=(4,5,6)
list=[]

for i in range (0,len(tup1)):
    list.append(tup1[i]+tup2[i])

print(list)
```

```
[5, 7, 9]
```

2.3 Dictionary

- Dictionaries are used to store data values in key: value pairs
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates
- Dictionaries are written with curly brackets, and have many keys and values:

[50]:
```
#Create and print a dictionary
```

```
thisdict= {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

2.3.1 Dictionary Items

- Dictionary items are ordered, changeable and doesn't allow duplicates
- Dictionary items are presented in key: value pairs, and can be reffered using the key name

[52]:
```
#Print the "brand" value of dictionary
thisdict= {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
print(thisdict["brand"])
```

```
Ford
```

[53]:
```
#Duplicates not allowed. Dictionaries cannot have two items with the same␣
 ↪key
#Example:

thisdict= {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964,
    "year": 2024
}

print(thisdict) #Duplicate values will overwrite existing values:
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2024}
```

[54]:
```
print(len(thisdict))
```

```
3
```

[55]:
```python
#Example: String, int, boolean, and list data types:

thisdict={
    "brand": "Ford",
    "electric": False,
    "year": 1964,
    "colours": ["red","white", "blue"]
}
print(type(thisdict))
```

```
<class 'dict'>
```

2.4 Functions

A function is a block of code that performs a specific task

2.4.1 Types of Functions

Types of Functions in Python Programming:

1. **Standard Library Functions:** These are built-in functions in Python that are readily available for use.
2. **User-Defined Functions:** We can create our own functions based on our specific requirements.

2.4.2 Standard Library Functions

- Examples include print(), sqrt(), pow(), etc.
- These functions are defined within modules, necessitating their inclusion in our program.
- For instance, the sqrt() function is defined within the "math" module.

[60]:
```python
import math

#sqrt computes the square root
square_root= math.sqrt(4)

print("Square root of 4 is", square_root)
```

```
Square root of 4 is 2.0
```

2.4.3 User-defined functions

- The syntax to declare a function is as follows:

“‘python def function_name(arguments): # function body return

- In this syntax:
  - def is the keyword used to declare a function.
  - function_name is any name assigned to the function.
  - arguments are the input parameters passed to the function (optional).
  - return statement (optional) is used to return a value from the function.

[62]:
```
def greet():
    print("Hello World!")
```

[63]:
```
greet()
```

```
Hello World!
```

[64]:
```
def greet():
    print("Hello World!")

#----> Start
greet()
print("Outside Function")
```

```
Hello World!
Outside Function
```

[65]:
```
#function with two arguments
def add_numbers(num1, num2):
    sum = num1+num2
    print("Sum: ", sum)
```

[66]:
```
add_numbers(5,4)
```

```
Sum:  9
```

[67]:
```
# we can also use
add_numbers(num1=5,num2=4)
```

```
Sum:  9
```

- If we want our function to return some value to a function call, we use the return statement

Note: This return statement also denotes that the function has ended. Any code after return is not executed

[68]:
```
#function definition
def find_square(num):
    result = num*num
    return result
```

[69]:
```
#function call
square = find_square(3)

print('Square: ', square)
```

```
Square:  9
```

2.5 Python Lambda/Anonymous function

- We use the lambda keyword instead of def to create a lambda function.
- Syntax to declare the lambda function

```
lambda argument(s): expression
```

- Here, arguments(s) - any value passed to the lambda function * expression - expression is executed and returned

[70]:
```
greet = lambda : print('Hello World')
```

[71]:
```
greet()
```

```
Hello World
```

[72]:
```
# Lambda that accepts one argument
greet_user = lambda name: print('Hey there: ', name)
```

[73]:
```
#Lambda Call
greet_user("Alice")
```

```
Hey there:  Alice
```

[74]:
```
def greet():

    #Local Variable
    message="Hello, 'I am a local variable'"
```

```
    print('Local', message)

greet()
```

```
Local Hello, 'I am a local variable'
```

2.6 Excercises

1. Reverse a list in python

[75]:
```
lst=[6,10,9,33,55,39,22]
```

[76]:
```
lst.reverse()
```

[77]:
```
lst
```

[77]:
```
[22, 39, 55, 33, 9, 10, 6]
```

2. Concatenate two lists index-wise : Write a program to add two lists index-wise. Create a new list that contains the 0th index item from both the list, then the 1st index item, and so on till the last element. any leftover items will get added at the end of the new list.

[78]:
```
#Approach 1: When two lists are of equal length
lst1=[10,20,30,40,50]
lst2=[90,80,70,60,50]
sumlst=[]
```

for i in range(0,(len(lst1))): sum=lst1[i]+lst2[i] sumlst.append(sum)

[80]:
```
sumlst
```

[80]:
```
[100, 100, 100, 100, 100]
```

[81]:
```
# Approach 2: When the lists are of different length
lst1=[11,36,58,97,2,60]
lst2=[16,23,25,6]
sumlst=[]
```

[82]:
```
min_len=min(len(lst1),len(lst2))
```

[83]:
```
for i in range (0,min_len):
  sum=lst1[i]+lst2[i]
  sumlst.append(sum)
```

[84]:
```
sumlst += lst1[min_len:] + lst2[min_len:]
```

[85]:
```
sumlst
```

[85]:
```
[27, 59, 83, 103, 2, 60]
```

3. Turn every item of a list into its square :Given a list of numbers. write a program to turn every item of a list into its square.

[86]:
```
lst=[2,3,4,5,15]
sqrlst=[]
```

[87]:
```
for i in range (0, (len(lst))):
    square=lst[i]**2
    sqrlst.append(square)
```

[88]:
```
sqrlst
```

[88]:
```
[4, 9, 16, 25, 225]
```

4. Concatenate two lists in the following order

[89]:
```
a=["1","2","3","4","5"]
b=["9","8","7","6","5"]
```

[90]:
```
c=[]
```

[91]:
```
for i in range(len(a)):
    c.append(a[i]+b[i])
```

[92]:
```
c
```

[92]:
```
['19', '28', '37', '46', '55']
```

5. Iterate both lists simultaneously : Given a two Python list. Write a program to iterate both lists simultaneously and display items from listl in original order and items from list2 in reverse order.

[93]:
```
l=[2,3,4,19,36,23]
m=[45,78,26,33,2]
```

[94]:
```
l.reverse()
```

[95]:
```
for (i,j) in zip(l,m):
    print(i,j)
```

```
23 45
36 78
19 26
4 33
3 2
```

6. Remove empty strings from the list of strings

[96]:
```
s=[" ","vishal","ruthvik"," ","arnnav"," "]
```

[97]:
```
s
```

[97]:
```
[' ', 'vishal', 'ruthvik', ' ', 'arnnav', ' ']
```

[98]:
```
while (" " in s):
    s.remove(" ")
```

[99]:
```
s
```

[99]:
```
['vishal', 'ruthvik', 'arnnav']
```

7. Add new item to list after a specified item

[100]:
```
list1 = [1, 2, 3, 4, 5, 6, 7]
```

[101]:
```
specified_item = 4
```

[102]:
```
index=list1.index(specified_item)
```

[103]:
```
new_item=55
```

[104]:
```
list1.insert(index+1,new_item)
```

[105]:
```
list1
```

[105]: [1, 2, 3, 4, 55, 5, 6, 7]

8. Extend nested list by adding the sublist: You have given a nested list. Write a program to extend it by adding the sublist [“h”, “i”, “j”] in such a way that it will look like the following list.

list1 = [“a”,“b”,[“c”,[“d”,“e”,[“f”,“g”],“k”],“l”],“m”,“n”]

Expected Output: [‘a’, ‘b’, [‘c’, [‘d’, ‘e’, [‘f’, ‘g’, ‘h’, ‘i’, ‘j’], ‘k’], ‘l’], ‘m’, ‘n’]

[106]:
```
list1 = ["a", "b", ["c", ["d", "e", ["f", "g"], "k"], "l"], "m", "n"]
```

[107]:
```
sublist=["h", "i", "j"]
```

[108]:
```
list1[2][1][2].extend(sublist)
```

[109]:
```
list1
```

[109]:
```
['a', 'b', ['c', ['d', 'e', ['f', 'g', 'h', 'i', 'j'], 'k'], 'l'], 'm',␣
↪'n']
```

9. Replace list’s item with new value if found: You have given a Python list. Write a program to find value 20 in the list, and if it is present, replace it with 200. Only update the first occurrence of an item.

[110]:
```
a=[50,256,36,12,87,20,98,20,300]
```

[111]:
```
if 20 in a:
    index = a.index(20)
    a[index]=200
```

[112]:
```
a
```

[112]:
```
[50, 256, 36, 12, 87, 200, 98, 20, 300]
```

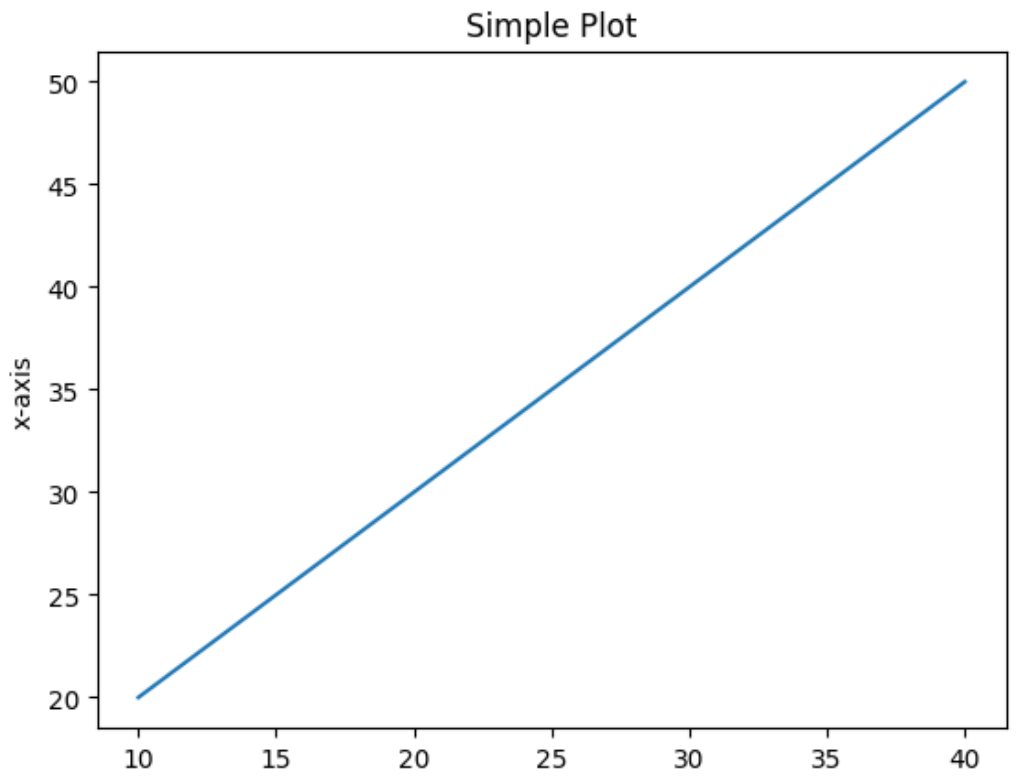
2.7 Standard plots (2D, 3D)

- Pyplot is a Matplotlib module that provides a MATLAB-like interface.
- Pyplot provides functions that interact with the figure i.e creates a figure, decorates the plot with labels, creates a plotting area in a figure

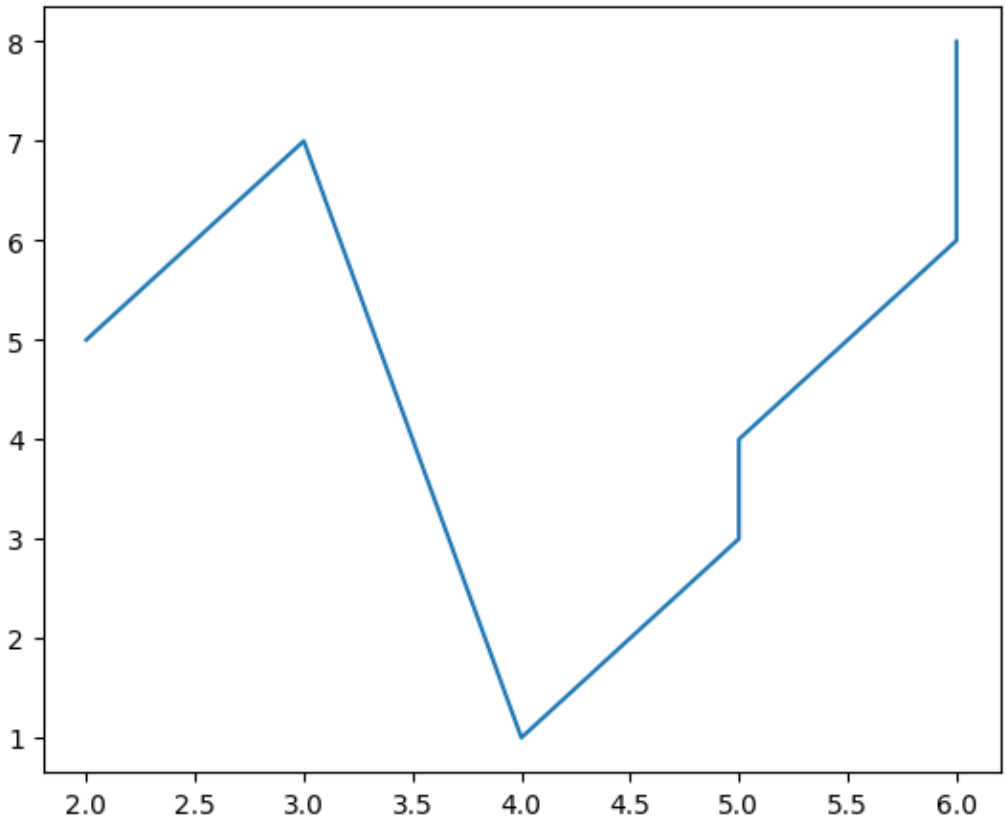
The basic steps to creating plots with matplotlib are: 1. Prepare Data 2. Create Plot 3. Plot 4. Customised Plot 5. Save Plot 6. Show Plot

2.7.1 Simple Plot

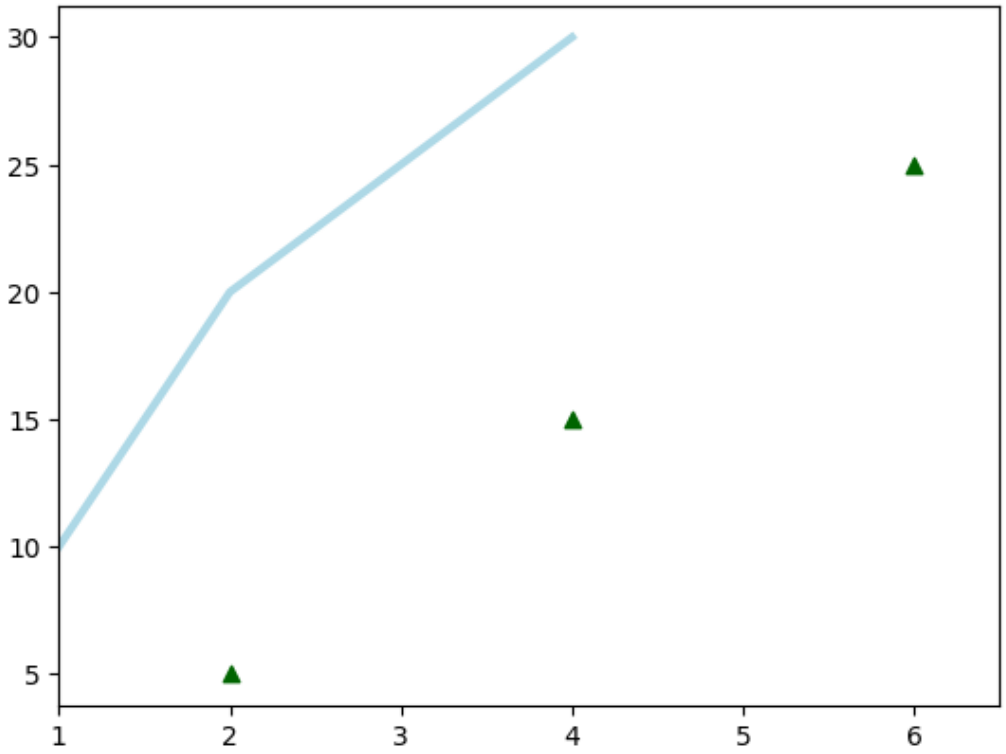
[135]:
```
import matplotlib.pyplot as plt
#initializing the data
x = [10,20,30,40]
y = [20,30,40,50]

# plotting the data
plt.plot(x, y)

# adding the title
plt.title("Simple Plot")

# adding the labels
plt.ylabel("y-axis")
plt.ylabel("x-axis")
plt.show()
```



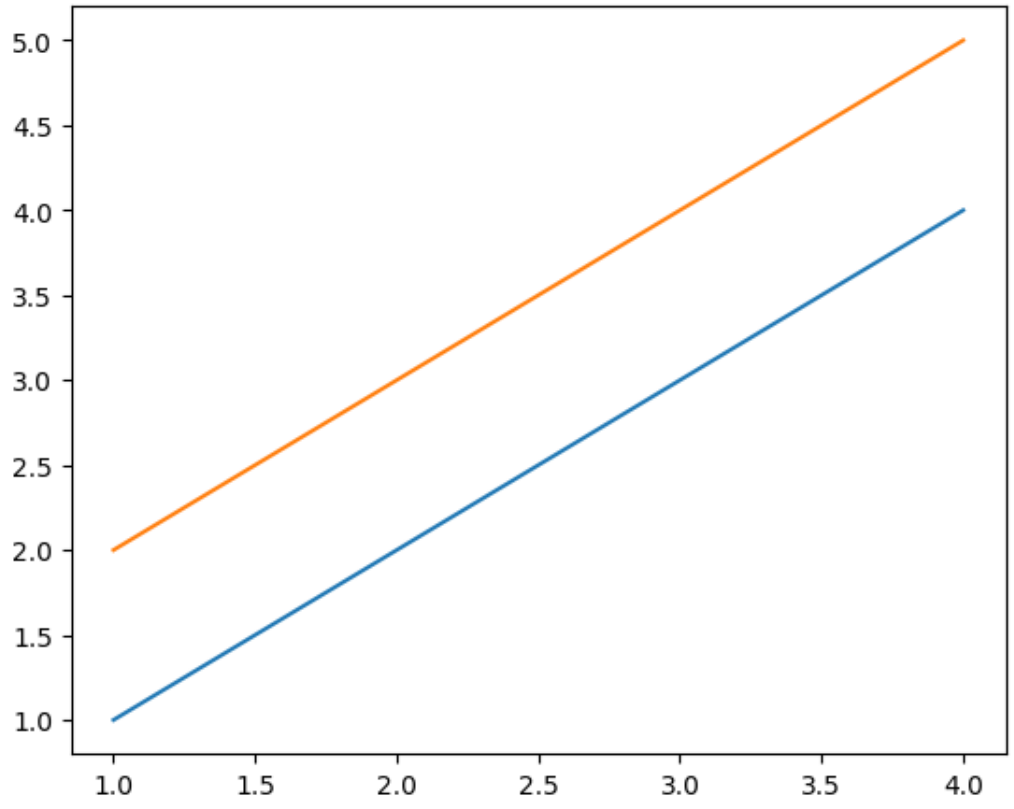
[136]:
```
# Python program to show pyplot module
import matplotlib.pyplot as plt
from matplotlib.figure import Figure

#figure with width = 5inches and height = 4inches
fig = plt.figure(figsize=(5,4))

#axes for the figure
ax= fig.add_axes([1,1,1,1])

#Adding the data to be plotted
ax.plot([2,3,4,5,5,6,6],
        [5,7,1,3,4,6,8])
plt.show()
```



[138]:
```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[10,20,25,30]
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(x,y,color='lightblue',linewidth=3)
ax.scatter([2,4,6],
           [5,15,25],
           color='darkgreen',
           marker='^')
ax.set_xlim(1,6.5)
plt.savefig('foo.png')
plt.show()
```

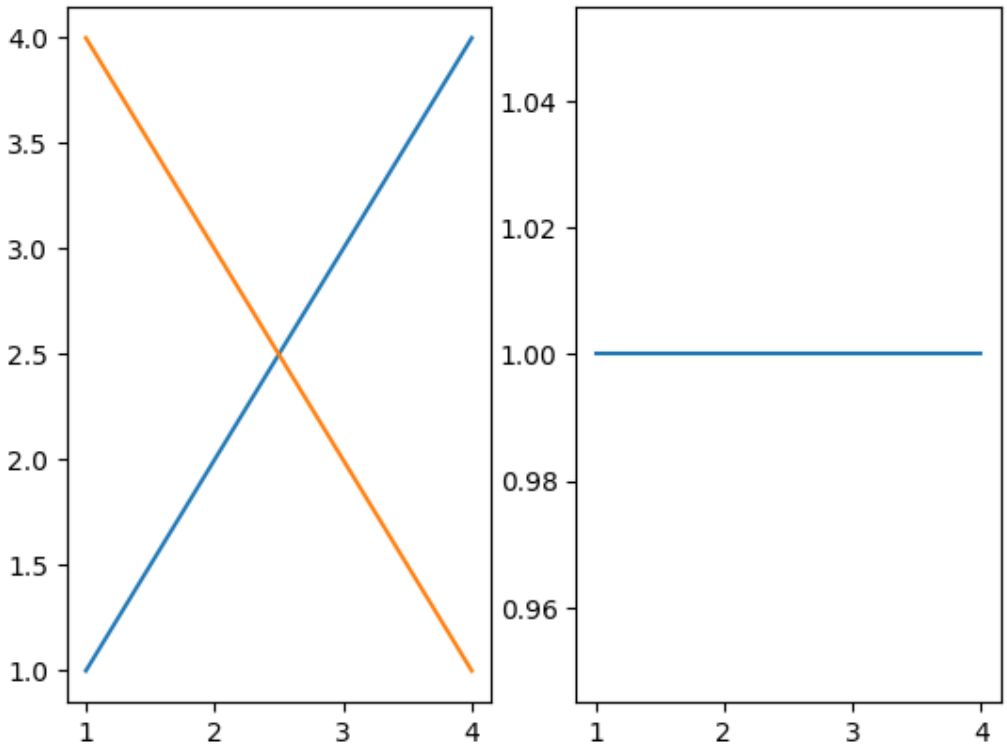


[139]:
```
#Python program to show two data in one figure
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
fig = plt.figure(figsize = (5,4))

#Adding the axes to the figure
ax = fig.add_axes([1,1,1,1])

#Plotting the 1st dataset to figure
ax1 = ax.plot([1,2,3,4], [1,2,3,4])

#Plotting the 2nd dataset to figure
ax2 = ax.plot([1,2,3,4], [2,3,4,5])
plt.show()
```



[140]:
```
import matplotlib.pyplot as plt

#Creating the figure ans subplots
fig, axes =plt.subplots(1,2)

#plotting the data in the 1st subplot
axes[0].plot([1,2,3,4],[1,2,3,4])

#plotting the data in the 1st subplot only
axes[0].plot([1,2,3,4],[4,3,2,1])

#plotting the data in the 2nd subplot only
axes[1].plot([1,2,3,4],[1,1,1,1])
```

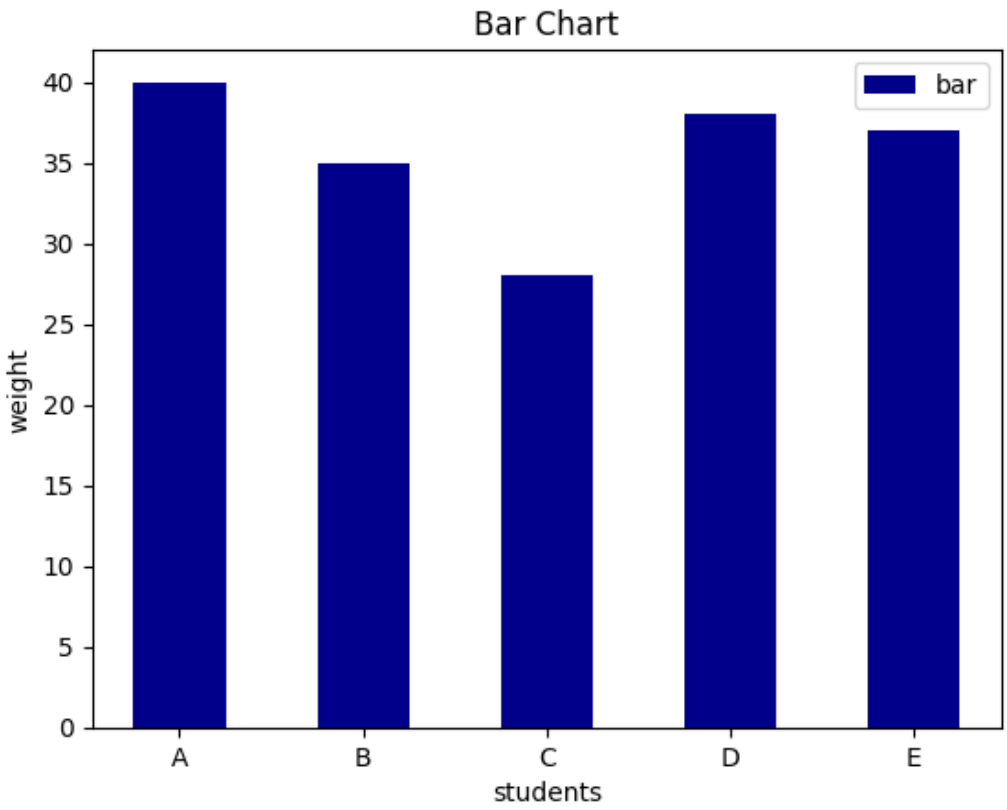
[140]: [<matplotlib.lines.Line2D at 0x159d68a2cd0>]



2.7.2 Bar Charts

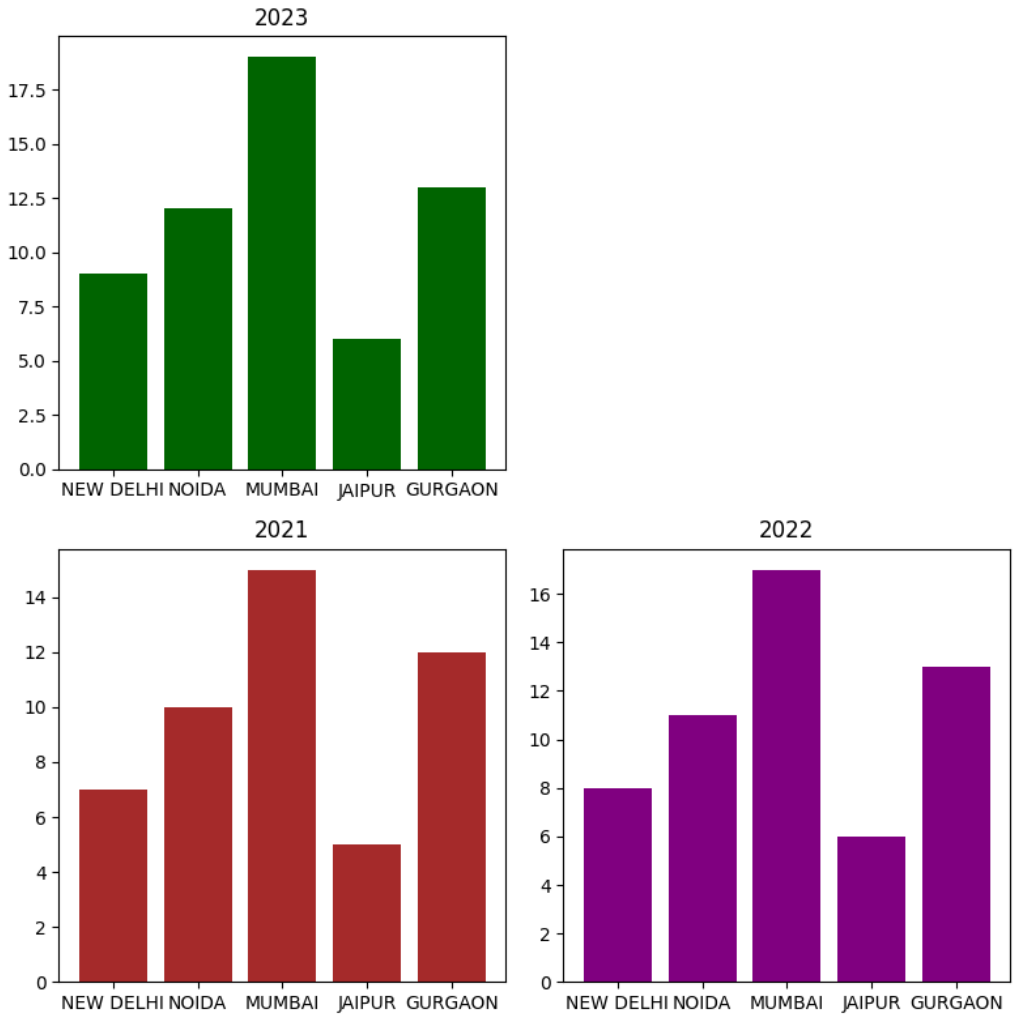
[118]:
```
import matplotlib.pyplot as plt
from matplotlib.figure import Figure

x=["A","B","C","D","E"]
y=[40,35,28,38,37]

plt.bar(x,y, color="darkblue", width=0.5)
plt.title("Bar Chart")

#Adding the legends
plt.legend(["bar"])
```

```
plt.xlabel("students")
plt.ylabel("weight")
plt.show()
```



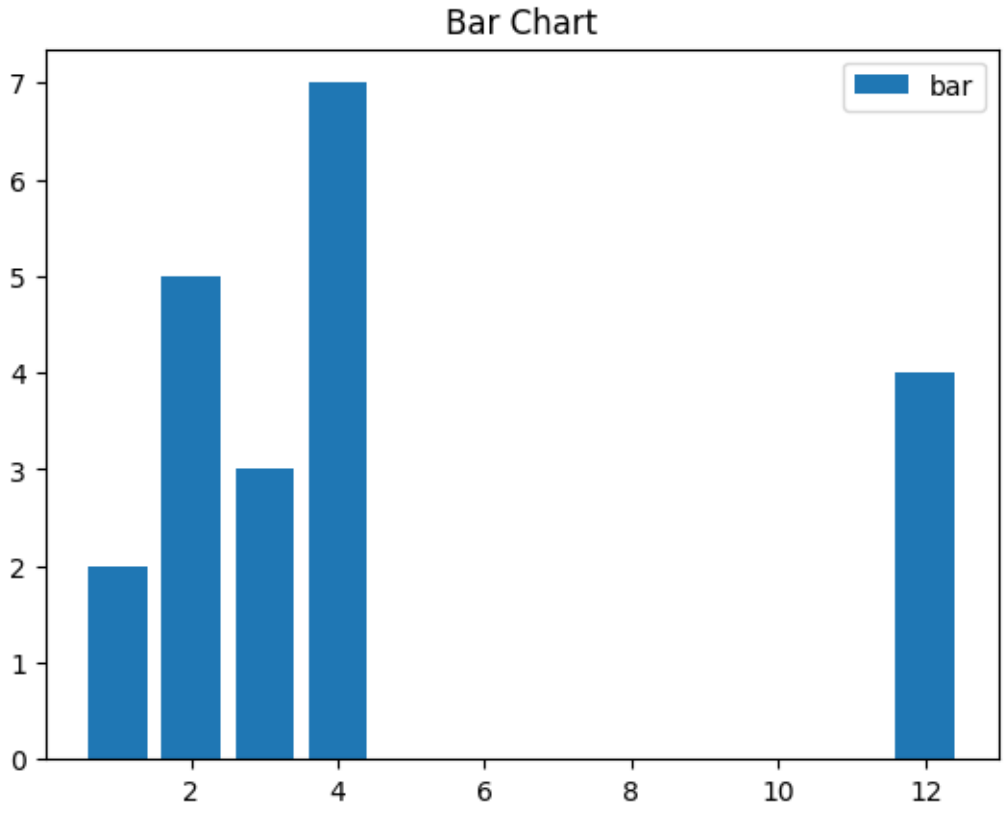
[121]:
```
import matplotlib.pyplot as plt
import numpy as no

city=["NEW DELHI", "NOIDA", "MUMBAI", "JAIPUR", "GURGAON"]
p_2021=[7,10,15,5,12]
p_2022=[8,11,17,6,13]
p_2023=[9,12,19,6,13]

#subplots
fig,axes=plt.subplots(2,2,figsize=(8,8))
```

```
#subplot1
axes[0,0].bar(city,p_2023,color='darkgreen')
axes[0,0].set_title('2023')
#subplot2
axes[1,0].bar(city,p_2021,color='brown')
axes[1,0].set_title('2021')
#subplot3
axes[1,1].bar(city,p_2022,color='Purple')
axes[1,1].set_title('2022')

#hiding required subplot
axes[0,1].axis('off')

plt.tight_layout() #auto adjust
plt.show()
```



[141]:
```
x= [3,1,3,12,2,4,4]
y= [3,2,1,4,5,6,7]

plt.bar(x,y)
plt.title("Bar Chart")

#Adding the legends
plt.legend(["bar"])
```

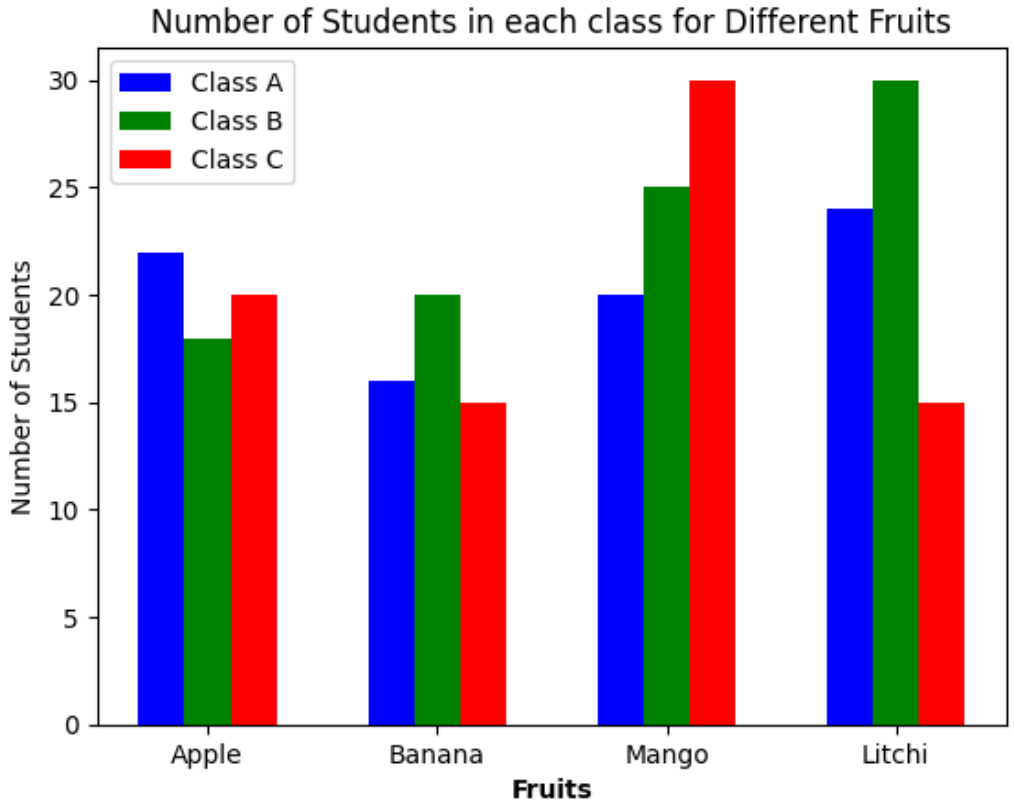
```
plt.show()
```



Grouped Bar Charts

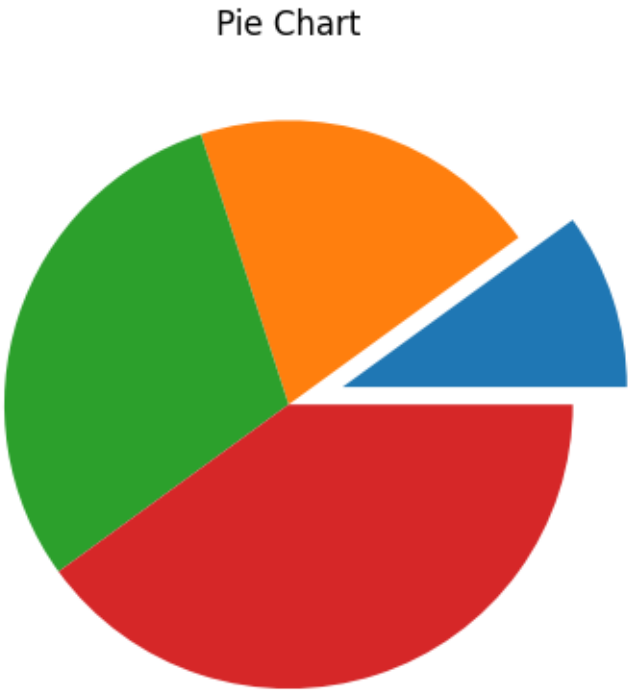
[142]:
```
import matplotlib.pyplot as plt
import numpy as np

# Given data
fruits = ['Apple', 'Banana', 'Mango', 'Litchi']
classes = ['A', 'B', 'C']
data = np.array([
  [22, 18, 20],
  [16, 20, 15],
  [20, 25, 30],
```

```
  [24, 30, 15]
])
# Setting up positions for bars
bar_width = 0.2
r1 = np.arange(len(fruits))
r2 = [x+ bar_width for x in r1]
r3 = [x+ bar_width for x in r2]

# Plotting the grouped bar graph
plt.bar(r1, data[:,0], color='b', width=bar_width, label='Class A')
plt.bar(r2, data[:,1], color='g', width=bar_width, label='Class B')
plt.bar(r3, data[:,2], color='r', width=bar_width, label='Class C')

plt.xlabel('Fruits',fontweight='bold')
plt.xticks([r+bar_width for r in range(len(fruits))],fruits)
plt.ylabel('Number of Students')
plt.title('Number of Students in each class for Different Fruits')
plt.legend()
plt.show()
```



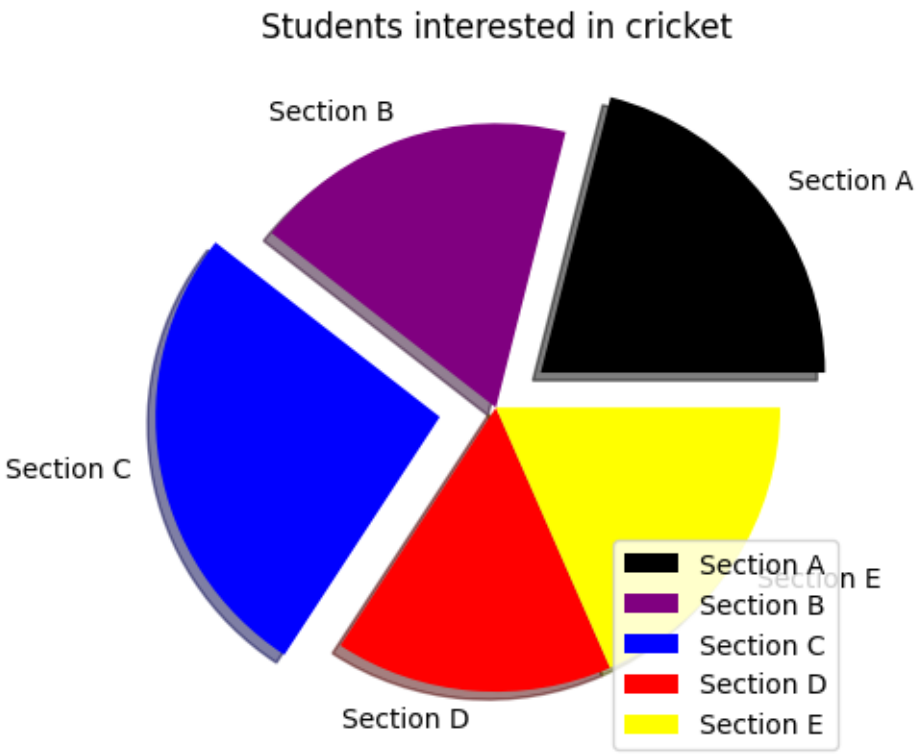
2.7.3 Pie Charts

[144]:
```
import matplotlib.pyplot as plt
x=[1,2,3,4]
#from the chart
e = (0.2,0,0,0)

# This will plot a pie chart
plt.pie(x, explode = e)

plt.title("Pie Chart")
plt.show()
```



[122]:
```
import matplotlib.pyplot as plt
x=[40,35,50,30,35]
mylabel=["Section A", "Section B", "Section C", "Section D", "Section E"]

#from the chart
e=(0.2,0,0.2,0,0)

#this will plot a pie chart
mycolors=["black","purple","b","red","yellow"]
figsize=(6,6)
plt.pie(x,explode=e,labels=mylabel,shadow=True,colors=mycolors)

plt.title("Students interested in cricket")
plt.legend(loc='lower right')
plt.show
```

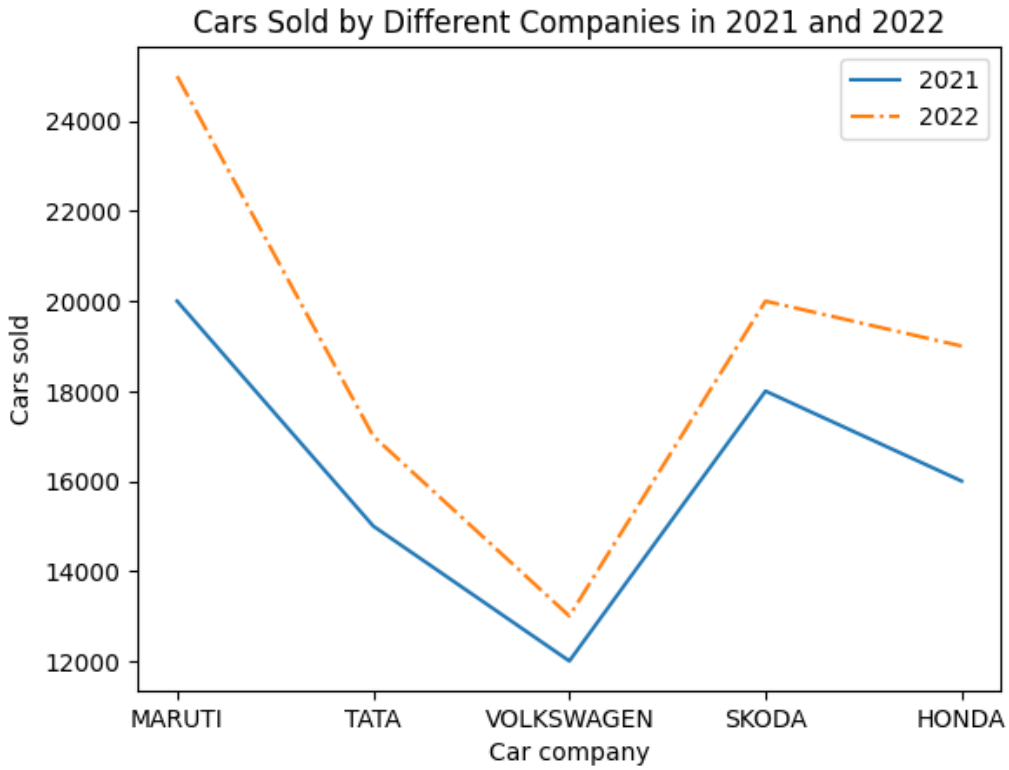
[122]: <function matplotlib.pyplot.show(close=None, block=None)>



2.7.4 Line Plot

[123]:
```
import matplotlib.pyplot as plt

x = ["MARUTI", "TATA", "VOLKSWAGEN", "SKODA", "HONDA"]
y1 = [20000, 15000, 12000, 18000, 16000]
y2 = [25000, 17000, 13000, 20000, 19000]

plt.plot(x, y1)
plt.plot(x, y2, '-.')

plt.xlabel("Car company")
plt.ylabel("Cars sold")
```

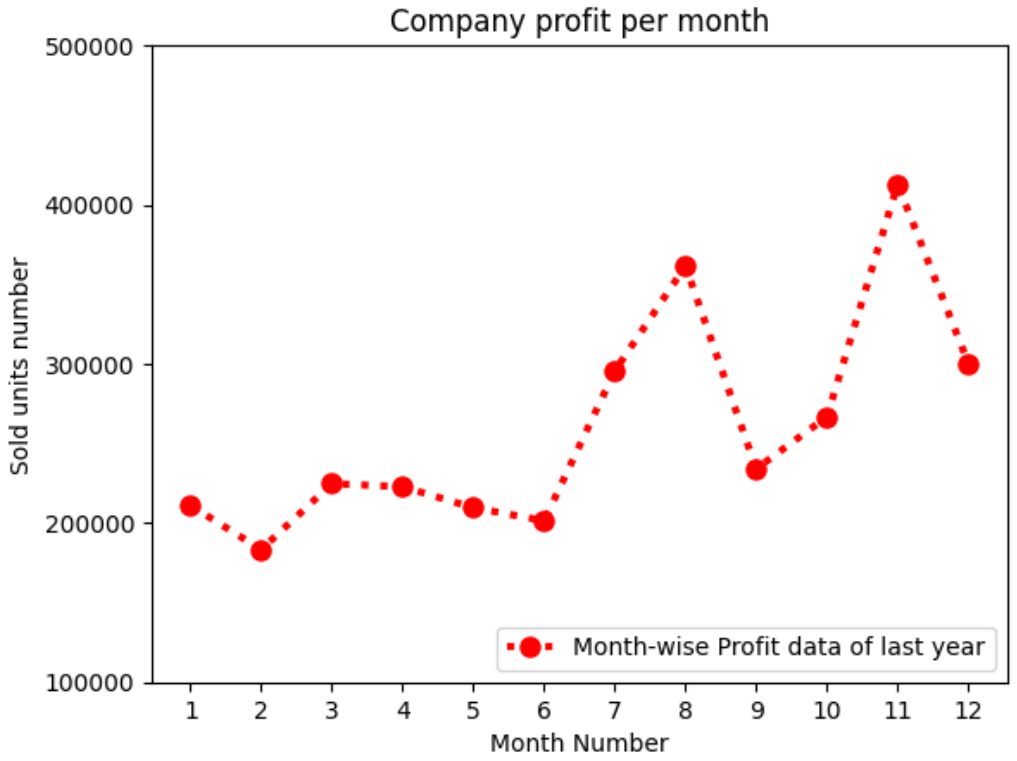
```
plt.title("Cars Sold by Different Companies in 2021 and 2022")
plt.legend(["2021", "2022"])
plt.show()
```



2.7.5 Excercise

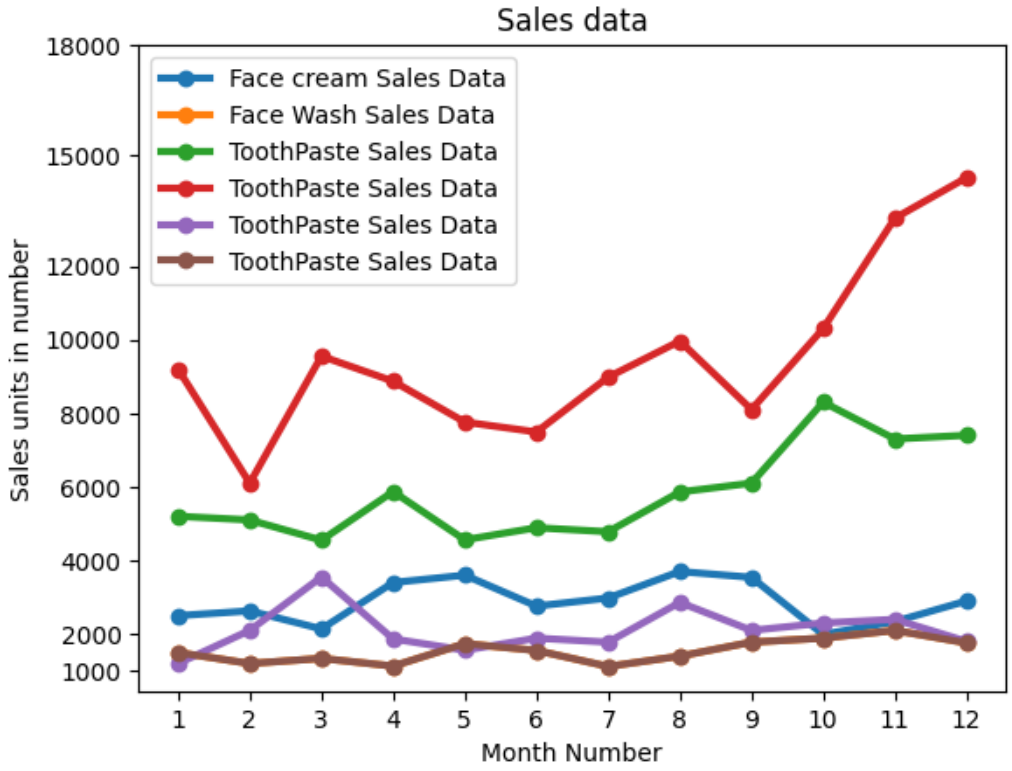
[126]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("company_sales_data - company_sales_data.csv")
profitList = df['total_profit'].tolist()
monthList = df['month_number'].tolist()
plt.plot(monthList, profitList, label='Month-wise Profit data of last␣
↪year', linestyle='dotted', color='red', marker='o',␣
↪markerfacecolor='red', markersize=8, linewidth=3)
```

```
plt.xlabel('Month Number')
plt.ylabel('Sold units number')
plt.xticks(monthList)
plt.title('Company profit per month')
plt.yticks([100000, 200000, 300000, 400000, 500000])
plt.legend(loc='lower right')
plt.show()
```

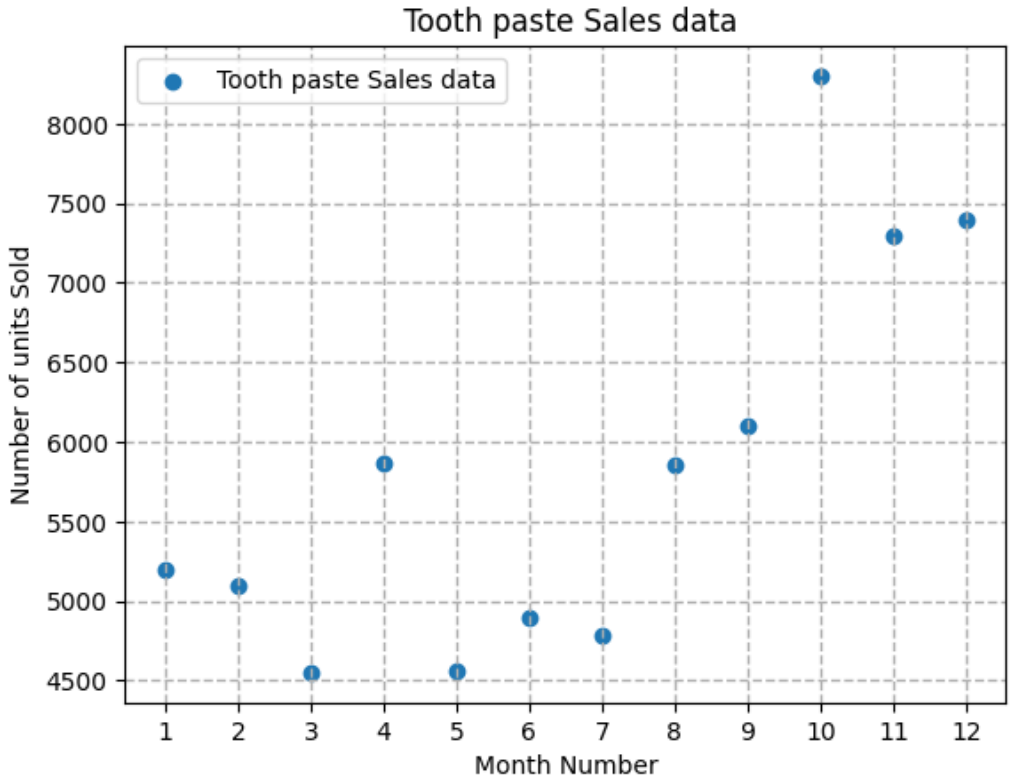


[128]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("company_sales_data - company_sales_data.csv")
monthList  = df ['month_number'].tolist()
faceCremSalesData   = df ['facecream'].tolist()
faceWashSalesData   = df ['facewash'].tolist()
```

```
toothPasteSalesData = df ['toothpaste'].tolist()
bathingsoapSalesData   = df ['bathingsoap'].tolist()
shampooSalesData   = df ['shampoo'].tolist()
moisturizerSalesData = df ['moisturizer'].tolist()

plt.plot(monthList, faceCremSalesData,   label = 'Face cream Sales Data',␣
 ↪marker='o', linewidth=3)
plt.plot(monthList, faceWashSalesData,   label = 'Face Wash Sales Data', ␣
 ↪marker='o', linewidth=3)
plt.plot(monthList, toothPasteSalesData, label = 'ToothPaste Sales Data',␣
 ↪marker='o', linewidth=3)
plt.plot(monthList, bathingsoapSalesData, label = 'ToothPaste Sales␣
 ↪Data', marker='o', linewidth=3)
plt.plot(monthList, shampooSalesData, label = 'ToothPaste Sales Data',␣
 ↪marker='o', linewidth=3)
plt.plot(monthList, moisturizerSalesData, label = 'ToothPaste Sales␣
 ↪Data', marker='o', linewidth=3)

plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.legend(loc='upper left')
plt.xticks(monthList)
plt.yticks([1000, 2000, 4000, 6000, 8000, 10000, 12000, 15000, 18000])
plt.title('Sales data')
plt.show()
```

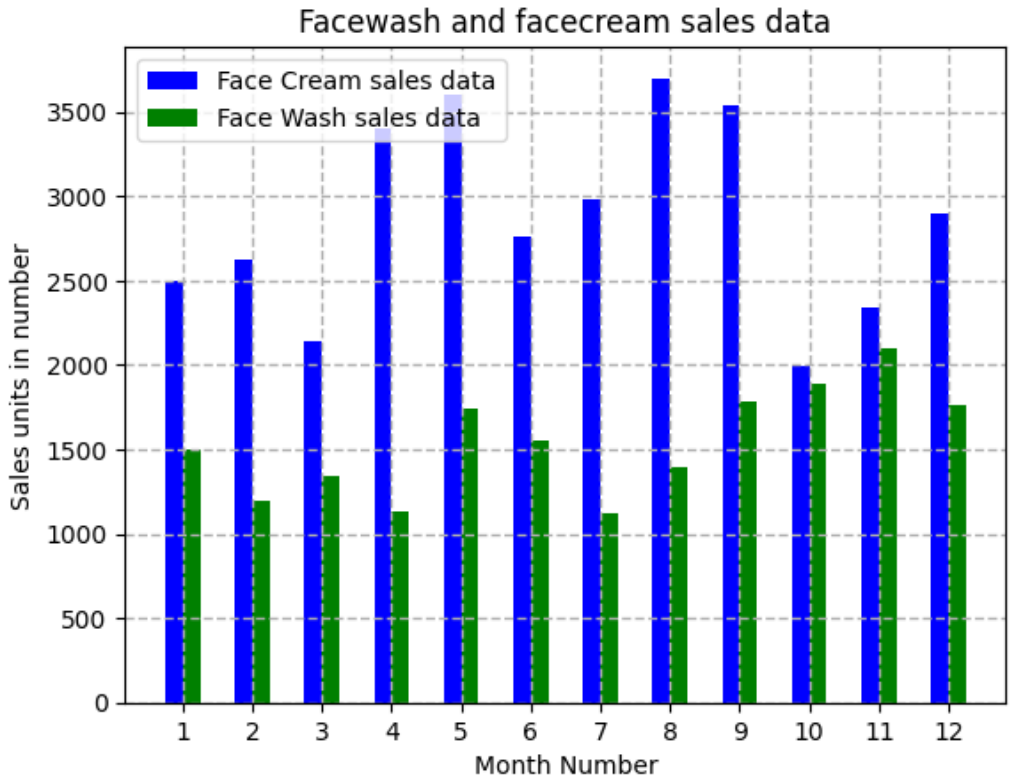


[129]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("company_sales_data - company_sales_data.csv")
monthList  = df ['month_number'].tolist()
toothPasteSalesData = df ['toothpaste'].tolist()
plt.scatter(monthList, toothPasteSalesData, label = 'Tooth paste Sales␣
 ↪data')
plt.xlabel('Month Number')
plt.ylabel('Number of units Sold')
plt.legend(loc='upper left')
plt.title(' Tooth paste Sales data')
plt.xticks(monthList)
plt.grid(True, linewidth= 1, linestyle="--")
plt.show()
```



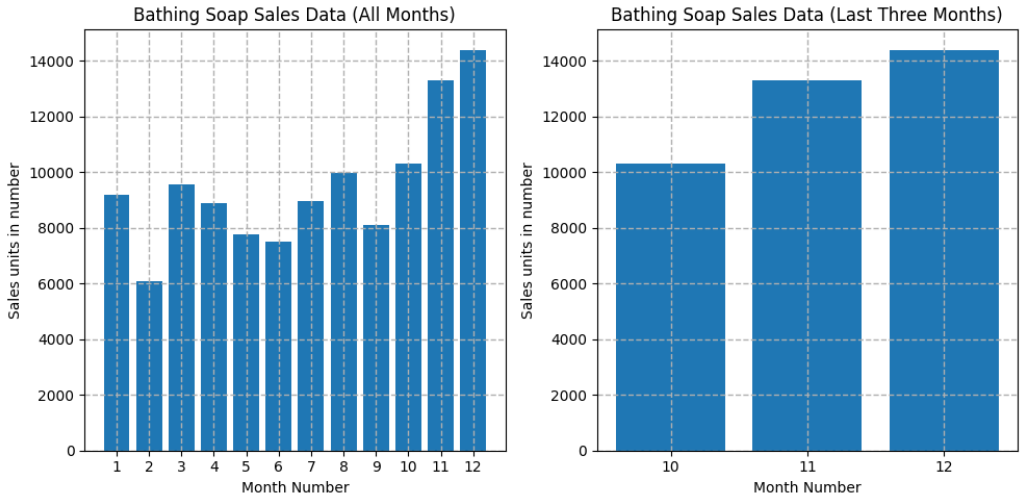
[131]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("company_sales_data - company_sales_data.csv")
monthList  = df['month_number'].tolist()
faceCremSalesData = df['facecream'].tolist()
faceWashSalesData = df['facewash'].tolist()

plt.bar([a-0.25 for a in monthList], faceCremSalesData, width=0.25,␣
 ↪label='Face Cream sales data', align='edge', color='blue', linewidth=1.
 ↪5)
plt.bar([a+0.25 for a in monthList], faceWashSalesData, width=-0.25,␣
 ↪label='Face Wash sales data', align='edge', color='green', linewidth=1.
 ↪5)
```

```
plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.legend(loc='upper left')
plt.xticks(monthList)
plt.grid(True, linewidth=1, linestyle="--")
plt.title('Facewash and facecream sales data')
plt.show()
```



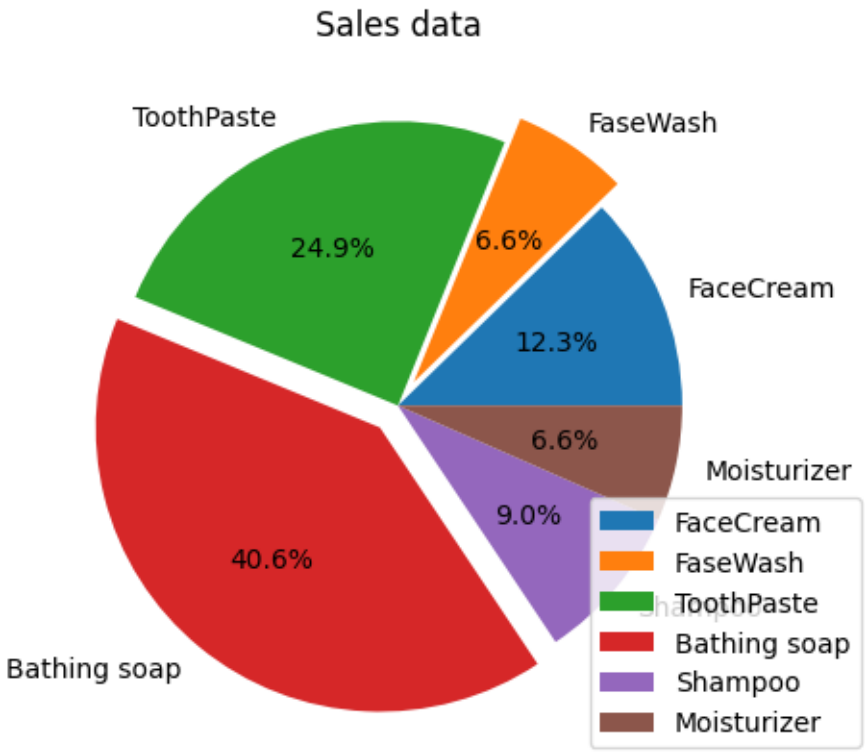
[133]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("company_sales_data - company_sales_data.csv")
monthList = df['month_number'].tolist()
bathingsoapSalesData = df['bathingsoap'].tolist()
```

```
# Plotting sales data of bathing soap for all months
plt.figure(figsize=(10, 5))  # Adjust figure size if needed
plt.subplot(1, 2, 1)  # Subplot for all months
plt.bar(monthList, bathingsoapSalesData)
plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.title('Bathing Soap Sales Data (All Months)')
plt.xticks(monthList)
plt.grid(True, linewidth=1, linestyle="--")

# Subplot for last three months
plt.subplot(1, 2, 2)
plt.bar(monthList[-3:], bathingsoapSalesData[-3:])
plt.xlabel('Month Number')
plt.ylabel('Sales units in number')
plt.title('Bathing Soap Sales Data (Last Three Months)')
plt.xticks(monthList[-3:])
plt.grid(True, linewidth=1, linestyle="--")

# Adjust layout to prevent overlap
plt.tight_layout()

# Save the plot
plt.savefig('bathingsoap_sales_subplot.png')

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



[134]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("company_sales_data - company_sales_data.csv")
monthList = df['month_number'].tolist()

labels = ['FaceCream', 'FaseWash', 'ToothPaste', 'Bathing soap',␣
↪'Shampoo', 'Moisturizer']
salesData = [df['facecream'].sum(), df['facewash'].sum(),␣
↪df['toothpaste'].sum(),
             df['bathingsoap'].sum(), df['shampoo'].sum(),␣
↪df['moisturizer'].sum()]

# Find index of the lowest and highest sales
lowest_index = salesData.index(min(salesData))
highest_index = salesData.index(max(salesData))

# Prepare explode list
explode = [0] * len(labels)
explode[lowest_index] = 0.1  # Explode lowest slice
explode[highest_index] = 0.1  # Explode highest slice
```

```
plt.axis("equal")
plt.pie(salesData, labels=labels, autopct='%1.1f%%', explode=explode)
plt.legend(loc='lower right')
plt.title('Sales data')
plt.show()
```



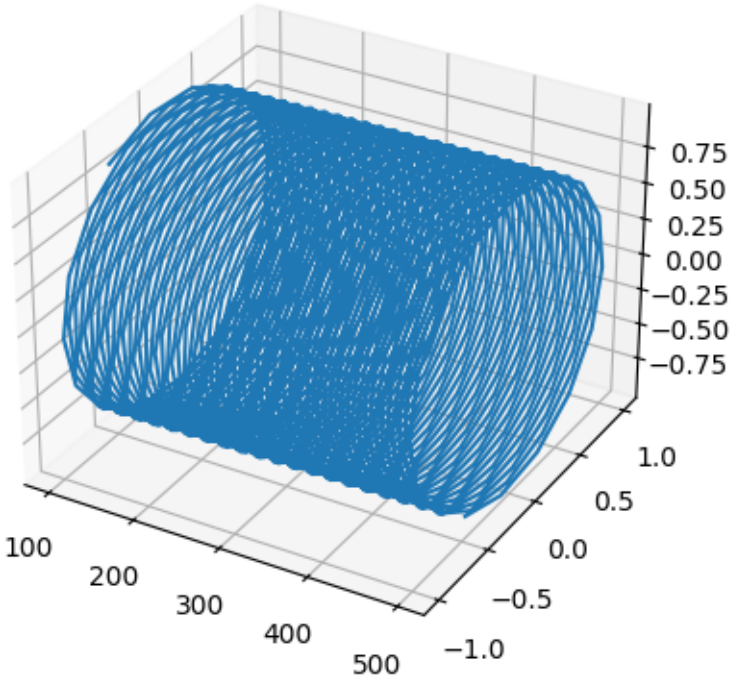
2.8 3D Plots

2.8.1 3D Line Plot

[146]:
```
from mpl_toolkits import mplot3d
from matplotlib.pyplot import *
from numpy import *
sp = axes (projection="3d")
x=linspace(100,500,1000)
```

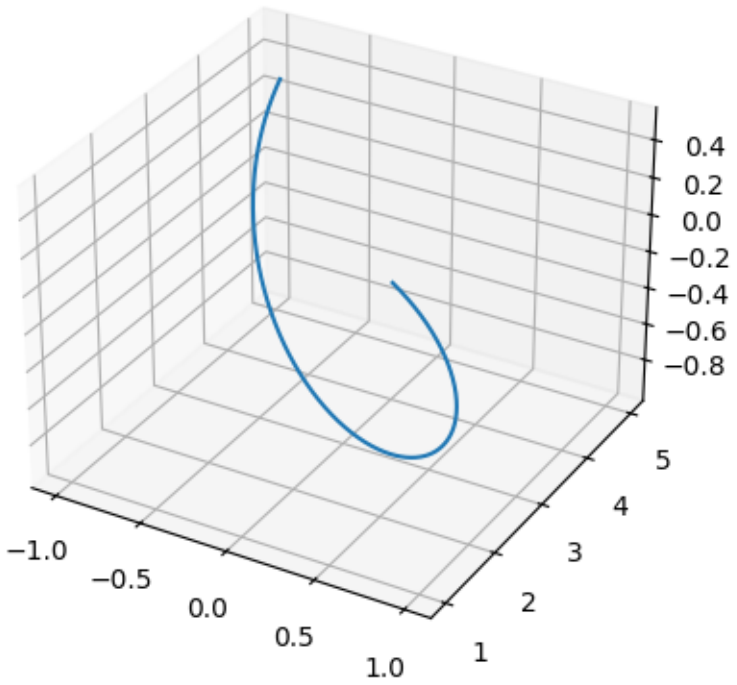
```
y=sin(x)
z=cos(x)
#print(x,y,z)
plot(x,y,z)
```

[146]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x159d64a3d90>]



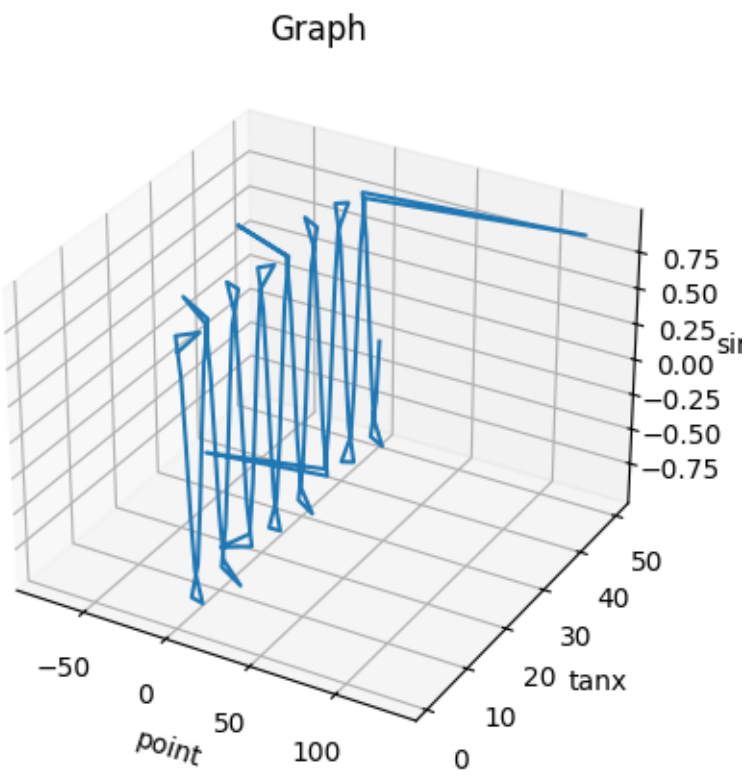
[147]:
```
sp = axes(projection = "3d")
x=linspace(1,5,1000)
y=sin(x)
z=cos(x)
#print(x,y,z)
plot(y,x,z)
```

[147]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x159d64939d0>]

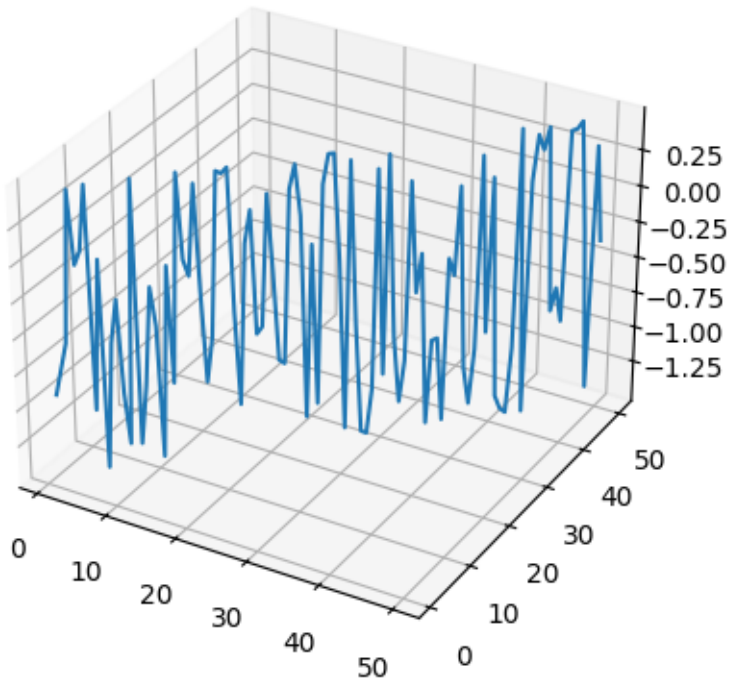


[148]:
```
sp=axes(projection ="3d")
x=linspace(1,50,100)
y=tan(x)
z=sin(x)
plot(y,x,z)
sp.set_title("Graph")
sp.set_xlabel("point")
sp.set_ylabel("tanx")
sp.set_zlabel("sinx")
```

[148]:
```
Text(0.5, 0, 'sinx')
```



[149]:
```
sp = axes(projection = "3d")
x=linspace(1,50,100)
y=linspace(1,50,100)
z=cos(x**2+y**2)-0.5
plot(y,x,z)
show()
```

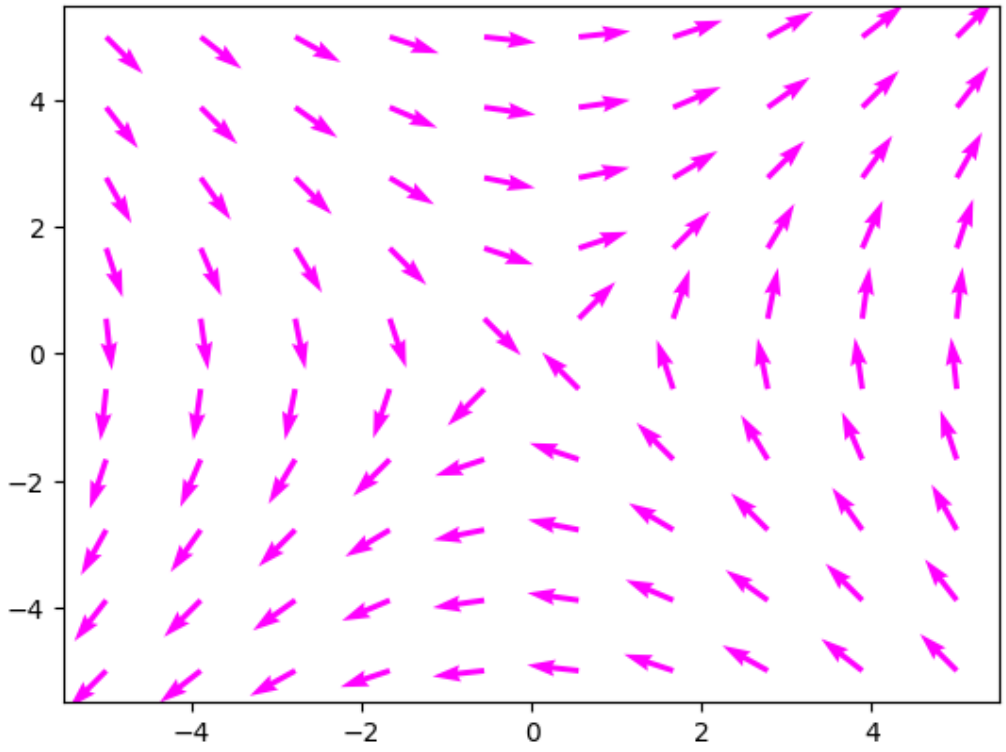


2.8.2 Vector Fields

[150]:
```
x,y=meshgrid(linspace(-5,5,10),linspace(-5,5,10))
u=x/sqrt(x**2+y**2)
v=y/sqrt(x**2+y**2)
quiver(y,x,u,v,color="magenta")
print(u,v)
```

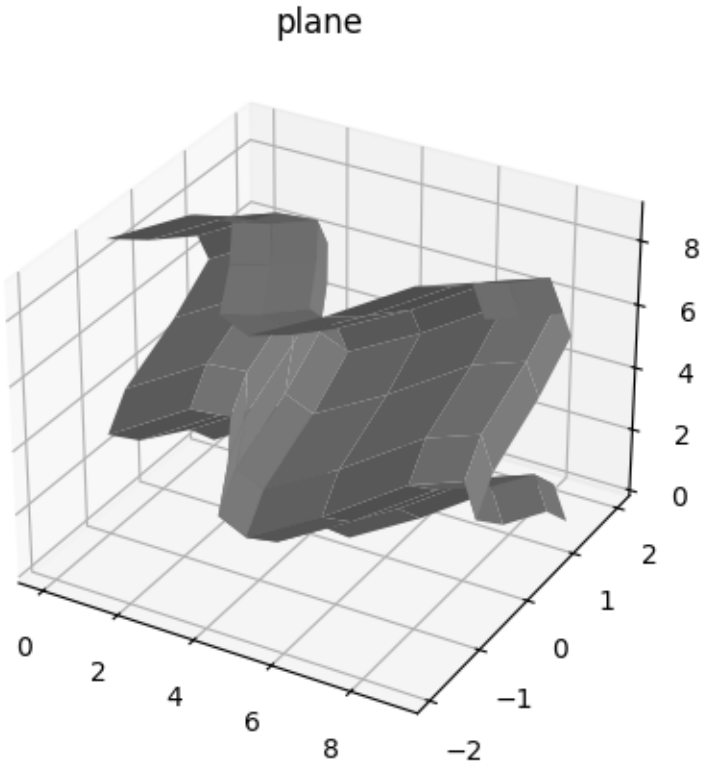
```
[[-0.70710678 -0.61394061 -0.48564293 -0.31622777 -0.11043153  0.11043153
   0.31622777  0.48564293  0.61394061  0.70710678]
 [-0.78935222 -0.70710678 -0.58123819 -0.3939193  -0.14142136  0.14142136
   0.3939193   0.58123819  0.70710678  0.78935222]
 [-0.87415728 -0.81373347 -0.70710678 -0.51449576 -0.19611614  0.19611614
   0.51449576  0.70710678  0.81373347  0.87415728]
 [-0.9486833  -0.91914503 -0.85749293 -0.70710678 -0.31622777  0.31622777
   0.70710678  0.85749293  0.91914503  0.9486833 ]
```

```
 [-0.99388373 -0.98994949 -0.98058068 -0.9486833  -0.70710678  0.70710678
   0.9486833   0.98058068  0.98994949  0.99388373]
 [-0.99388373 -0.98994949 -0.98058068 -0.9486833  -0.70710678  0.70710678
   0.9486833   0.98058068  0.98994949  0.99388373]
 [-0.9486833  -0.91914503 -0.85749293 -0.70710678 -0.31622777  0.31622777
   0.70710678  0.85749293  0.91914503  0.9486833 ]
 [-0.87415728 -0.81373347 -0.70710678 -0.51449576 -0.19611614  0.19611614
   0.51449576  0.70710678  0.81373347  0.87415728]
 [-0.78935222 -0.70710678 -0.58123819 -0.3939193  -0.14142136  0.14142136
   0.3939193   0.58123819  0.70710678  0.78935222]
 [-0.70710678 -0.61394061 -0.48564293 -0.31622777 -0.11043153  0.11043153
   0.31622777  0.48564293  0.61394061  0.70710678]] [[-0.70710678 -0.
↪78935222
-0.87415728 -0.9486833  -0.99388373 -0.99388373
  -0.9486833  -0.87415728 -0.78935222 -0.70710678]
 [-0.61394061 -0.70710678 -0.81373347 -0.91914503 -0.98994949 -0.98994949
  -0.91914503 -0.81373347 -0.70710678 -0.61394061]
 [-0.48564293 -0.58123819 -0.70710678 -0.85749293 -0.98058068 -0.98058068
  -0.85749293 -0.70710678 -0.58123819 -0.48564293]
 [-0.31622777 -0.3939193  -0.51449576 -0.70710678 -0.9486833  -0.9486833
  -0.70710678 -0.51449576 -0.3939193  -0.31622777]
 [-0.11043153 -0.14142136 -0.19611614 -0.31622777 -0.70710678 -0.70710678
  -0.31622777 -0.19611614 -0.14142136 -0.11043153]
 [ 0.11043153  0.14142136  0.19611614  0.31622777  0.70710678  0.70710678
   0.31622777  0.19611614  0.14142136  0.11043153]
 [ 0.31622777  0.3939193   0.51449576  0.70710678  0.9486833   0.9486833
   0.70710678  0.51449576  0.3939193   0.31622777]
 [ 0.48564293  0.58123819  0.70710678  0.85749293  0.98058068  0.98058068
   0.85749293  0.70710678  0.58123819  0.48564293]
 [ 0.61394061  0.70710678  0.81373347  0.91914503  0.98994949  0.98994949
   0.91914503  0.81373347  0.70710678  0.61394061]
 [ 0.70710678  0.78935222  0.87415728  0.9486833   0.99388373  0.99388373
   0.9486833   0.87415728  0.78935222  0.70710678]]
```

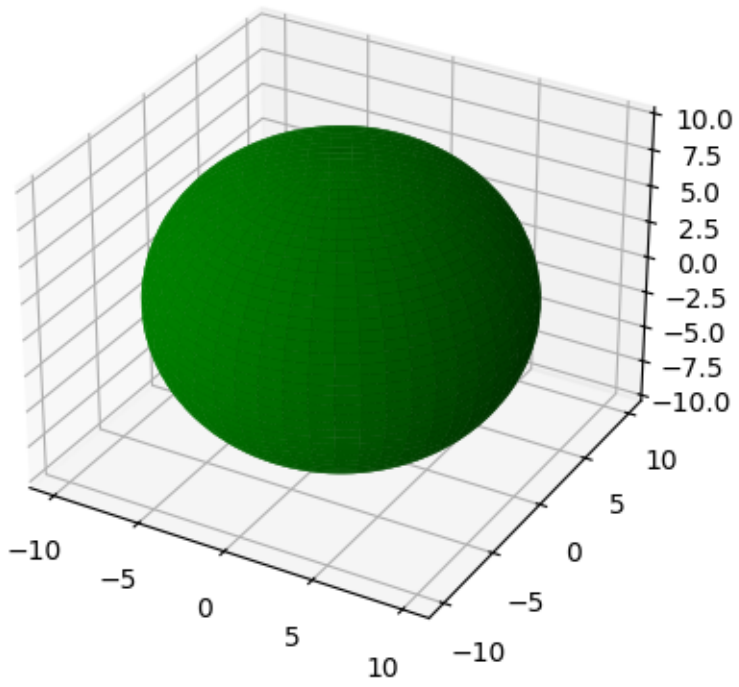


2.8.3 Counter Plots

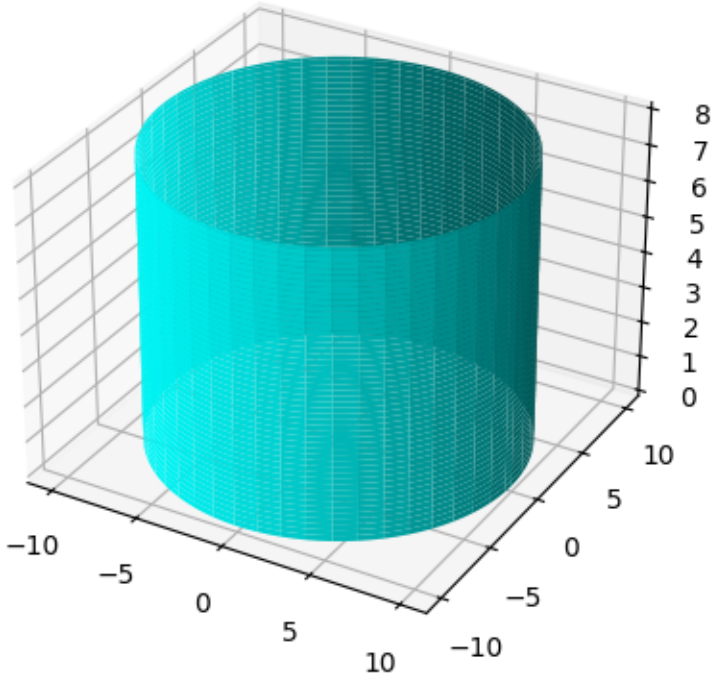
[151]:
```
from mpl_toolkits import mplot3d
from matplotlib.pyplot import *
from numpy import *
#Surface Plot
x,z=meshgrid(range(10),range(10))
y=sin(x)+cos(z)
ax=axes(projection = "3d")
ax.plot_surface(x,y,z, color='grey')
title("plane")
sp.set_xlabel("$X$")
sp.set_ylabel("$Y$")
sp.set_zlabel("$Z$")
show()
```



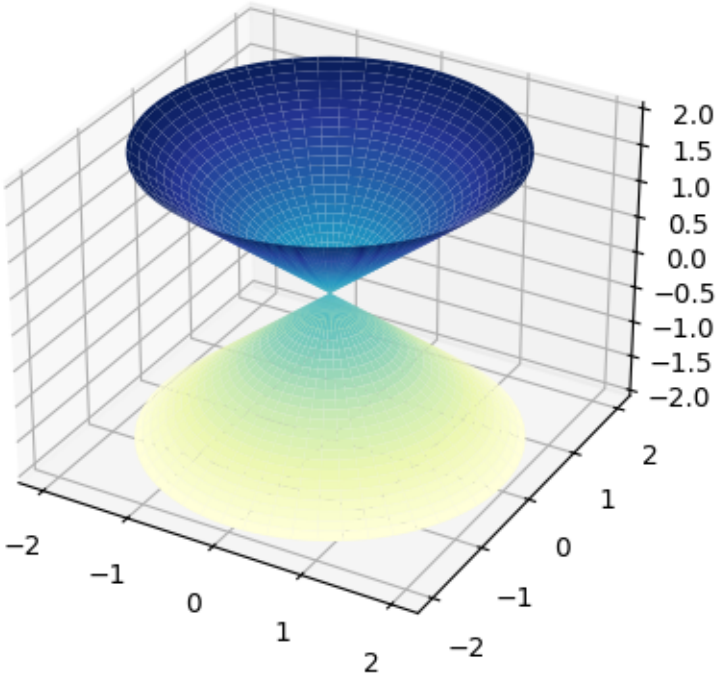
[152]:
```
ax=axes(projection="3d")
u=linspace(0,2*pi,100)
v=linspace(0,pi,100)
x=10*outer(cos(u),sin(v))
y=10*outer(sin(u),sin(v))
z=10*outer((ones(size(v))),cos(v))
ax.plot_surface(x,y,z, color = 'green')
show()
```



[153]:
```
ax=axes(projection="3d")
u=linspace(0,2*pi,100)
v=linspace(0,8,100)
x=10*outer((ones(size(u))),cos(u))
y=10*outer((ones(size(u))),sin(u))
z=outer(v,ones(size(v)))
ax.plot_surface(x,y,z, color = 'cyan')
show()
```



[154]:
```
ax=axes(projection = '3d')
u=linspace(0,2*pi,100)
v=linspace(-2,2,100)
x=outer(v,cos(u))
y=outer(v,sin(u))
z=outer(v,ones(size(u)))
ax.plot_surface(x,y,z, cmap='YlGnBu')
show()
```



3 Unit 2: Symbolic and Numeric Computations

3.1 Use of Sympy and NumPy Packages

1. SymPy: SymPy is a Python library for symbolic mathematics. It provides tools for symbolic computation, including algebraic operations, calculus, equation solving, and more. SymPy allows users to work with mathematical expressions symbolically rather than numerically, making it useful for tasks such as solving equations, simplifying expressions, and performing algebraic manipulations.
2. NumPy: NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is widely used in fields such as mathematics, science, engineering, and data analysis for tasks like numerical computing, linear algebra, statistical analysis, and more. It forms the basis for many other scientific computing libraries in Python.

[155]:
```
from sympy import *
```

3.2 Basic algebraic operations with polynomials/rational functions & Solving Algebraic Equations

3.2.1 Q. Write a program which finds roots of Quadratic equation $x^2 + x + 1$

[157]:
```
import sympy as sp
x=sp.Symbol('x')
eqn=x**2+x+1
roots=sp.solve(eqn,x)
roots
```

[157]:
```
[-1/2 - sqrt(3)*I/2, -1/2 + sqrt(3)*I/2]
```

3.2.2 Q. Write a program which finds roots of Quadratic equation $x^3 + 6x^2 + 11x - 6$

[158]:
```
import sympy as sp
x=sp.Symbol('x')
equation=x**3-6*x**2+11*x-6
roots=sp.solve(equation,x)
print("Roots: ",roots)
```

```
Roots:  [1, 2, 3]
```

3.2.3 Q. Write a program which finds roots of any Quadratic equation

[160]:
```
import cmath

def find_roots(a,b,c):
    val=cmath.sqrt(b**2-4*a*c)
    root1=(-b+val)/(2*a)
    root2=(-b-val)/(2*a)

    return root1, root2

if __name__ == "__main__":
    a=float(input("Enter the coefficient of a: "))
    b=float(input("Enter the coefficient of b: "))
    c=float(input("Enter the coefficient of c: "))
```

```
    roots= find_roots(a,b,c)
    print("Root 1: ",roots[0])
    print("Root 2: ",roots[1])
```

```
Enter the coefficient of a:  1
Enter the coefficient of b:  2
Enter the coefficient of c:  -15

Root 1:  (3+0j)
Root 2:  (-5+0j)
```

3.3 Evaluating Limits

Syntax:

`limit(fn,var,val)`

[162]:
```
limit(x**2,x,3)
```

[162]:
$$9$$

3.3.1 Q. Find the limit of $\frac{x-1}{x^2-1}$ as $x->0$

[164]:
```
limit((x-1)/(x**2-1),x,0)
```

[164]:
$$1$$

3.4 Differentiation

Syntax:

`diff(fn,w.r.t var, nth derivative)`

[165]:
```
diff(x**2+x,x,1)
```

[165]:
$$2x + 1$$

[167]:
```
diff(x**3+x**2,x,2)
```

[167]:
$$2 \cdot (3x + 1)$$

[168]:
```
from sympy import *
x,y=symbols("x,y")
diff(sin(x+y),x,2,y,1)
```

[168]:

$$-\cos(x+y)$$

3.4.1 Q. Find differentiation for the following: $-log(x) - sin(x) - cos(x) - x^3 - e^x$

[170]:
```
diff(-log(x)-sin(x)-cos(x)-x**3-exp(x))
```

[170]:
$$-3x^2 - e^x + \sin(x) - \cos(x) - \frac{1}{x}$$

3.5 Integration

Syntax:

`Integrate(fn,(var,low,lim,upp lim))`

[171]:
```
from sympy import *
```

[173]:
```
import sympy as sp
x=sp.Symbol('x')
integrate(x**2,(x,0x5))
```

[173]:
$$\frac{125}{3}$$

3.5.1 Find $\int_0^1 xy\ dx$

[174]:
```
from sympy import *
x,y=symbols("x,y")
integrate(x*y,(x,0,1))
```

[174]:
$$\frac{y}{2}$$

3.5.2 Find $\int_0^1 \int_0^2 \int_0^3 xyz\ dx\ dy\ dz$

[177]:
```
from sympy import *
x,y,z=symbols("x,y,z")
integrate(x*y*z,(x,0,1),(y,0,2),(z,0,3))
```

[177]:
$$\frac{9}{2}$$

3.6 Find $\int_{y=0}^{1}\int_{z=y^2}^{1}\int_{0}^{1-z} z\ dx\ dy\ dz$

[178]:
```
from sympy import *
x,y,z=symbols("x,y,z")
integrate(z,(x,0,1-z),(z,y**2,1),(y,0,1))
```

[178]: $\frac{4}{35}$

3.7 Arc Length

Arc length is the distance between two points along a section of a curve. Determining the length of an irregular arc segment is also called rectification of a curve. The advent of infinitesimal calculus led to general formula that provides closed-form solutions in some cases

If f' is continuous $[a, b]$, then the arc length of the curve $y = f(x)$ from the point $A = (a, f(a))$ to the point $B = (b, f(b))$ is the value of the integral

To compute the length of a curve on the interval [a,b] we compute the

$L = \int_a^b \sqrt{(1 + |f' = (x)|^2 dx}$

3.7.1 Q.1: Find the arc length of $f(x) = x^{3/2}$ on $|0, 2|$

[181]:
```
from sympy import *
x=Symbol("x")
f=diff(x**(3/2),x)
integrate(sqrt(1+f**2),(x,0,2))
```

[181]: 3.52552395244872

3.7.2 Q2: Find the arc length of $f(x) = x$ on $[1, 2]$

[182]:
```
from sympy import *
x=Symbol('x')
f=diff(x,x,1)
integrate(sqrt(1+f**2),(x,1,2))
```

[182]: $\sqrt{2}$

3.7.3 Q3: Find the arc length of $f(x) = (1/3)(x^2+2)^{3/2}$ on $[0,1]$

[183]:
```
from sympy import *
x=Symbol('x')
f=diff((1/3)*(x**2+2)**(3/2),x)
integrate(sqrt(1+f**2),(x,0,1))
```

[183]:
$$1.0\int_0^1 \sqrt{1.0x^2\left(x^2+2\right)^{1.0}+1.0}\,dx$$

3.8 Double Integration

Double integral of a function of two varieties $f(x,y)$ over a region in the plane is evaluated by integrating the function with respect to each of the yx and y in turn.

It is used to find area of the given region.

$\int\int f(a)dA = \int_{x=a}^{b}\int_{y=c}^{d} f(x)dxdy$

3.8.1 Find $\int_0^1\int_0^1 xy\ dx\ dy$

[185]:
```
from sympy import *
x,y=symbols("x,y")
integrate(x*y,(x,0,1),(y,0,1))
```

[185]:
$$\frac{1}{4}$$

3.8.2 Integrate over the region $f(x,y) = x/y$ in the first quadrant bounded by the lines $y = x, y = 2x, x = 1$ and $x = 2$

[186]:
```
import sympy as sp
x,y=sp.symbols('x,y')
sp.integrate(x/y,(y,x,2*x),(x,1,2))
```

[186]:
$$\frac{3\log(2)}{2}$$

3.8.3 Find the area oer the region R bounded by $y = x$ and $y = x^2$ in the first quadrant for $x = 1$

[187]:
```
sp.integrate(1,(y,x,x*x),(x,0,1))
```

[187]:
$$-\frac{1}{6}$$

3.9 Trigonometric Simplifications

[207]:
```
from math import *

a=pi/6
b=3
c=4

#returning the value of tangent of pi/6
print("The value of tangent of pi/6 is:", end="")
print (tan(a))

#returning the value of hypotenuse of 3 and 4
print("The value of hypotenuse of 3 and 4 is:", end="")
print(hypot(b,c))
```

```
The value of tangent of pi/6 is:0.5773502691896257
The value of hypotenuse of 3 and 4 is:5.0
```

[208]:
```
import math
from math import *
a=pi/6
b=30
c=15

#Returning the converted value from radians to degrees
print("The converted value from radians to degrees is: ", end="")
print(math.degrees(a))

#Returning the converted value from degrees to radians
print("The converted value from degrees to radians is: ", end="")
print(math.radians(a))

print(radians(c))
```

```
The converted value from radians to degrees is: 29.999999999999996
The converted value from degrees to radians is: 0.009138522593601256
0.2617993877991494
```

[209]:
```
from numpy import *
from sympy import *
#Inverse values:
print(asin(1).evalf())
print(acos(0))
print(atan(0))
```

```
1.57079632679490
pi/2
0
```

[210]:
```
x=1.0
y=1.0
z=complex(x,y)
print(z)
print("The arc sine is: ",asin(z))
```

```
(1+1j)
The arc sine is:  0.666239432492515 + 1.06127506190504*I
```

[211]:
```
x=1.0
y=1.0
z=complex(x,y)
print(z)
print("The arc sine is: ", asin(z))
print("The arc cosine is: ", acos(z))
print("The arc hyperbolic sine is: ", sinh(z))
print("The arc hyperbolic cosine is: ", cosh(z))
print("The arc hyperbolic tangent is: ", tanh(z))
print("The arc inverse hyperbolic sine is: ", asinh(z))
print("The arc inverse hyperbolic cosine is: ", acosh(z))
print("The arc inverse hyperbolic tangent is: ", atanh(z))
```

```
(1+1j)
The arc sine is:  0.666239432492515 + 1.06127506190504*I
The arc cosine is:  0.904556894302381 - 1.06127506190504*I
The arc hyperbolic sine is:  0.634963914784736 + 1.29845758141598*I
The arc hyperbolic cosine is:  0.833730025131149 + 0.988897705762865*I
```

```
The arc hyperbolic tangent is:  1.08392332733869 + 0.271752585319512*I
The arc inverse hyperbolic sine is:  1.06127506190504 + 0.666239432492515*I
The arc inverse hyperbolic cosine is:  1.06127506190504 + 0.
↪904556894302381*I
The arc inverse hyperbolic tangent is:  0.402359478108525 + 1.
↪01722196789785*I
```

[212]:
```
x=1.0
y=1.0
z=complex(x,y)
print(z)
print("The arc sine is: ", asin(z).evalf())
print("The arc cosine is: ", acos(z).evalf())
print("The arc hyperbolic sine is: ", sinh(z).evalf())
print("The arc hyperbolic cosine is: ", cosh(z).evalf())
print("The arc hyperbolic tangent is: ", tanh(z).evalf())
print("The arc inverse hyperbolic sine is: ", asinh(z).evalf())
print("The arc inverse hyperbolic cosine is: ", acosh(z).evalf())
print("The arc inverse hyperbolic tangent is: ", atanh(z).evalf())
```

```
(1+1j)
The arc sine is:  0.666239432492515 + 1.06127506190504*I
The arc cosine is:  0.904556894302381 - 1.06127506190504*I
The arc hyperbolic sine is:  0.634963914784736 + 1.29845758141598*I
The arc hyperbolic cosine is:  0.833730025131149 + 0.988897705762865*I
The arc hyperbolic tangent is:  1.08392332733869 + 0.271752585319512*I
The arc inverse hyperbolic sine is:  1.06127506190504 + 0.666239432492515*I
The arc inverse hyperbolic cosine is:  1.06127506190504 + 0.
↪904556894302381*I
The arc inverse hyperbolic tangent is:  0.402359478108525 + 1.
↪01722196789785*I
```

3.10 Exponential and Logarithms

[213]:
```
import math

print ("log value:", math.log(2))
print ("log value of 5000 with base 10:", math.log(5000,10))
print ("log value of 500 with base 2:", math.log(500,2))
```

```
log value: 0.6931471805599453
```

```
log value of 5000 with base 10: 3.6989700043360187
log value of 500 with base 2: 8.965784284662087
```

[214]:
```
a=log(10,10)
print(a)
```

```
1
```

3.11 Problems

Problem 1: Projectile Motion A ball is thrown upwards with an initial velocity of $20m/s$ The acceleration due to gravity is $-9.8m/s^2$. Find the expression for the ball's acceleration and velocity(v) at any time (t).

[191]:
```
#Define the variables
t=sp.symbols('t') #Time
v=sp.symbols('v') #Velocity

#Initial Velocity
v0=20

#Acceleration due to gravity
g=-9.8

#Equation for velocity (v=u+gt)
velocity_eqn = v-v0+g*t

#Find derivative (acceleration)
acceleration=round(sp.diff(velocity_eqn,t),3)

print("Velocity equation:", velocity_eqn)
print("Acceleration:", acceleration)
```

```
Velocity equation: -9.8*t + v - 20
Acceleration: -9.800
```

Problem 2: Rate of change of Population A population of rabbits is modeled by the function $P(t) = 1000/(1+0.01t^2)$ where t is the time in years. Estimate the rate of change of the population at t=5years

[192]:
```
t=sp.symbols("t")
population = 1000/(1+0.01*t**2)
```

```
pop=sp.diff(population,t,1)
print(pop)
res = pop.subs({t:5})
print(res)
```

```
-20.0*t/(0.01*t**2 + 1)**2
-64.0000000000000
```

Problem 3: Jerk in Motion Jerk is the rate of change of acceleration. A car's position (s) is given by $s(t) = t^4 - 2t^3 + 3t^2$ Find the expressoin for the jerk (j) of the car at any time (t)

[193]:
```
import sympy as sp
t,s = sp.symbols("t,s")
eqn=t**4-2*t**3+3*t**2

jerk=sp.diff(eqn,t,3)
print("Jerk is: ", jerk)
```

```
Jerk is:  12*(2*t - 1)
```

Problem 4: Mass Distribution A thin plate has a mass density of $\rho(x, y) = x^2 + y^2$. Find the total mass of the plate if its boundaries are defined by $0 \leq x \leq 1$ and $0 \leq y \leq 1$

[194]:
```
import sympy as sp

#Define variables
x=sp.symbols('x')
y=sp.symbols('y')

#Functions for mass density
mass_density = x**2+y**2

#Integration Limits
lower_x=0
upper_x=1
lower_y=0
upper_y=2

#Set up the double integral (integrate y first, then x)
total_mass = sp.integrate(mass_density, (y,0,2), (x,0,1))
```

```
print("Total mass of the plate: ", round(total_mass.evalf(),3))
```

```
Total mass of the plate:  3.333
```

4 Unit 3: Solving Differential Equations

4.1 Finding solutions and plotting the solution curves of first and second order differential equations

Solve $\frac{dy}{dx} = 300y + x$ with initial condition $y(0) = 30$

[196]:
```
import numpy as np
import sympy as sp
from sympy import *

x=Symbol('x')

f=Function("f")(x)

print("The given Differential Equation is: ")
d=Eq(f.diff(x)-300*f-x,0)
display(d)

print("The General Solution is: ")
gs=dsolve(d)
display(gs)

g=dsolve(d)
#P(0)=379
p1=g.subs({f:30, x:0})
display(p1)

C1=Symbol('C1')

c_1=solve(p1,C1)
display(c_1)

c=float(c_1[0])
display(c)
```

```
final=g.subs({C1:c})
display(final)

sp.plot(final.rhs)
```

The given Differential Equation is:

$$-x - 300f(x) + \frac{d}{dx}f(x) = 0$$

The General Solution is:

$$f(x) = C_1 e^{300x} - \frac{x}{300} - \frac{1}{90000}$$

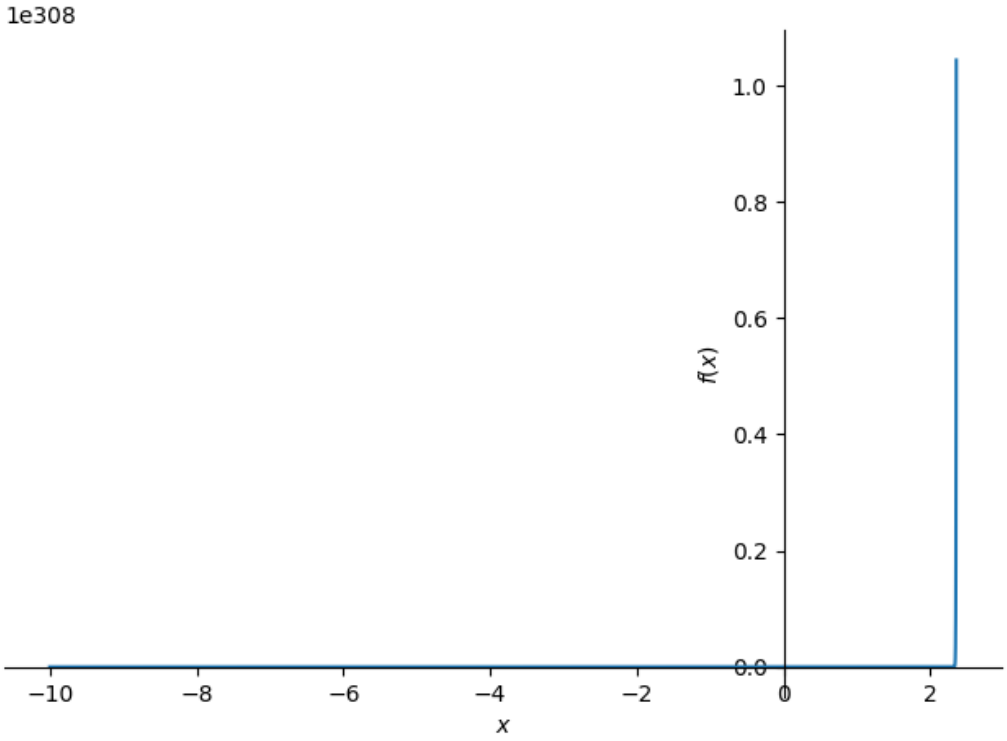
$$30 = C_1 - \frac{1}{90000}$$

[2700001/90000]

30.00001111111111

$$f(x) = -\frac{x}{300} + 30.0000111111111e^{300x} - \frac{1}{90000}$$

```
C:\Users\ASUS\anaconda3\Lib\site-packages\sympy\plotting\plot.py:1634:
RuntimeWarning: invalid value encountered in scalar divide
  cos_theta = dot_product / (vector_a_norm * vector_b_norm)
C:\Users\ASUS\anaconda3\Lib\site-packages\sympy\plotting\plot.py:1629:
RuntimeWarning: invalid value encountered in subtract
  vector_a = (x - y).astype(np.float64)
C:\Users\ASUS\anaconda3\Lib\site-packages\sympy\plotting\plot.py:1630:
RuntimeWarning: invalid value encountered in subtract
  vector_b = (z - y).astype(np.float64)
C:\Users\ASUS\anaconda3\Lib\site-packages\sympy\plotting\plot.py:1482:
RuntimeWarning: overflow encountered in scalar multiply
  pos_bottom = ('data', 0) if yl*yh <= 0 else 'center'
C:\Users\ASUS\anaconda3\Lib\site-packages\matplotlib\ticker.py:2096:
RuntimeWarning: overflow encountered in multiply
  steps = self._extended_steps * scale
```



[196]: <sympy.plotting.plot.Plot at 0x159d9e34c10>

Solve $\frac{dy}{dx} = 20y + e^x$

[197]:
```
import numpy as np
import sympy as sp
from sympy import *

x=Symbol('x')

f=Function("f")(x)

print("The given Differential Equation is: ")
d=Eq(f.diff(x)-20*f-sp.exp(x),0)
display(d)
```

```
print("The General Solution is: ")
gs=dsolve(d)
display(gs)
```

The given Differential Equation is:

$$-20f(x) - e^x + \frac{d}{dx}f(x) = 0$$

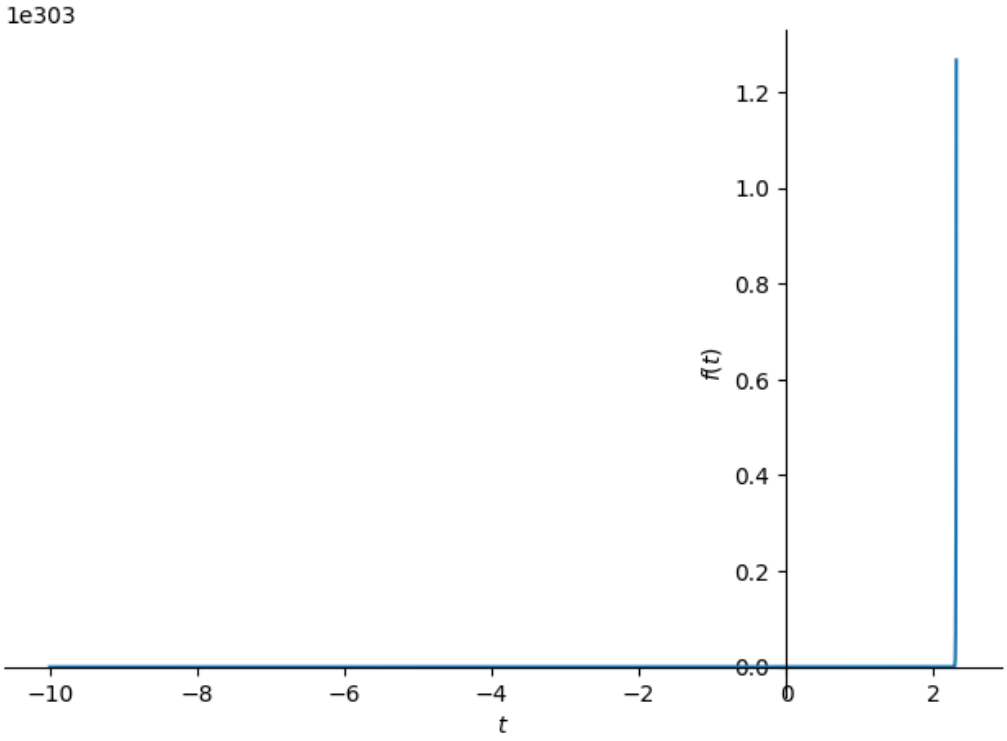
The General Solution is:

$$f(x) = \left(C_1 e^{19x} - \frac{1}{19}\right) e^x$$

[198]:
```
import sympy as sp
from sympy.solvers import ode

t=sp.symbols('t') #symbol
y=sp.Function('y') #symbolic function
eqn=y(t).diff(t)-300*y(t)-t #eqn=0
eqn

choice=ode.classify_ode(eqn)
display(choice)

ode.dsolve(eqn,hint=choice[2])#we solve the Differential Equation


sol=ode.dsolve(eqn,hint=choice[0], ics={y(0):30})
sol

sp.plot(sol.rhs)
```

```
('1st_exact',
 '1st_linear',
 'Bernoulli',
 'almost_linear',
 '1st_power_series',
 'lie_group',
 'nth_linear_constant_coeff_undetermined_coefficients',
 'nth_linear_constant_coeff_variation_of_parameters',
 '1st_exact_Integral',
 '1st_linear_Integral',
```

```
'Bernoulli_Integral',
'almost_linear_Integral',
'nth_linear_constant_coeff_variation_of_parameters_Integral')
```



[198]: `<sympy.plotting.plot.Plot at 0x159da29c4d0>`

4.1.1 Excercises

1) $-6y + 5\frac{dy}{dx} + \frac{d^2y}{dx^2} = 0$

[200]:
```
#Using dsolve find the solution
from sympy.interactive import printing
printing.init_printing(use_latex=True)
from sympy import *

x=Symbol("x")
```

```
y=Function("y")(x)

D=Eq(y.diff(x,x)+5*y.diff(x)-6*y,0)
display(D)
dsolve(D,y)
```

[200]:

$$-6y(x) + 5\frac{d}{dx}y(x) + \frac{d^2}{dx^2}y(x) = 0$$

$$y(x) = C_1 e^{-6x} + C_2 e^{x}$$

2) $\frac{dy}{dx} + \frac{d^2y}{dx^2} = 0$

[202]:

```
#using dsolve find the solution
from sympy.interactive import printing
printing.init_printing(use_latex=True)
from sympy import *
x=Symbol('x')
y=Function("y")(x)
D=Eq(y.diff(x,x)+y.diff(x),0)
display(D)
dsolve(D,y)
```

[202]:

$$\frac{d}{dx}y(x) + \frac{d^2}{dx^2}y(x) = 0$$

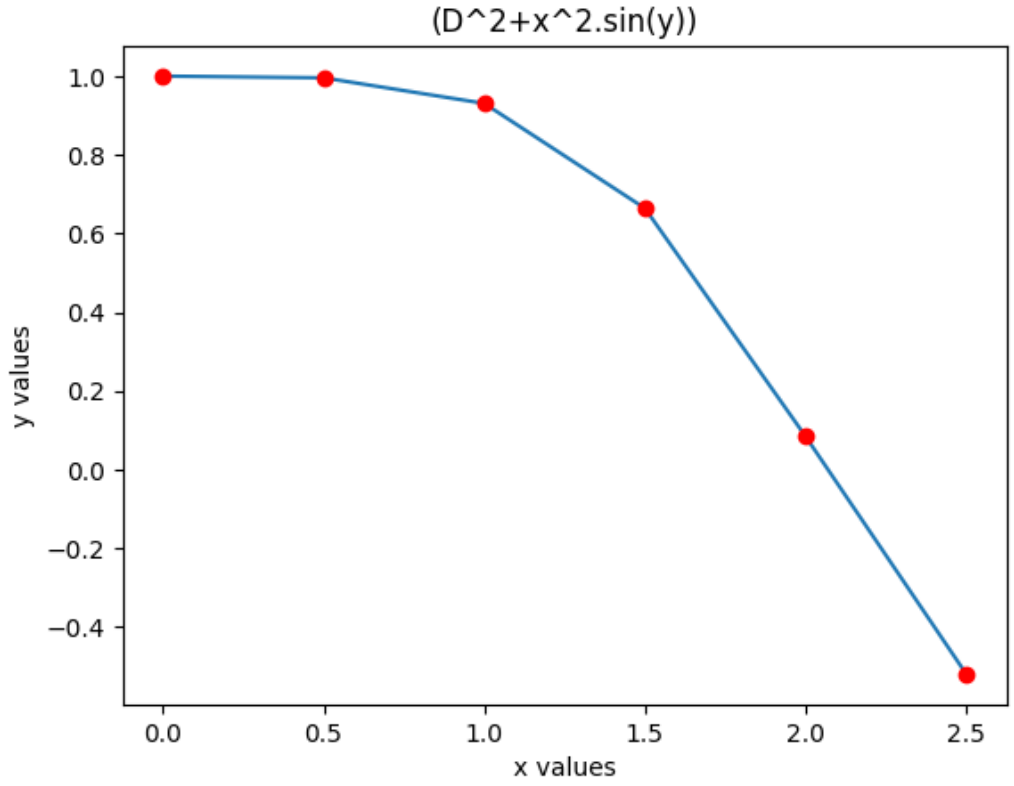
$$y(x) = C_1 + C_2 e^{-x}$$

3) $\frac{d^2y}{dx^2} - x^2 sin(y) = 0$

[203]:

```
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np

def f(u,x):
    return [u[1], -x**2 * np.sin(u[0])]

#Initial Conditions
u0=[1,0] #Initial values for y and y'

#Define the range of x-values
xs=np.arange(0,3,0.5)
```

```
#Solve the differential equation using odeint
us=odeint(f,u0,xs)

#Extract the solution for y
ys=us[:,0]

#Plot the solution
plt.plot(xs,ys, '-') #Solid Line
plt.plot(xs,ys, 'ro') #Red circles at data points

#Label the axes and a title
plt.xlabel('x values')
plt.ylabel('y values')
plt.title('(D^2+x^2.sin(y))')

#Display the plot
plt.show()
```

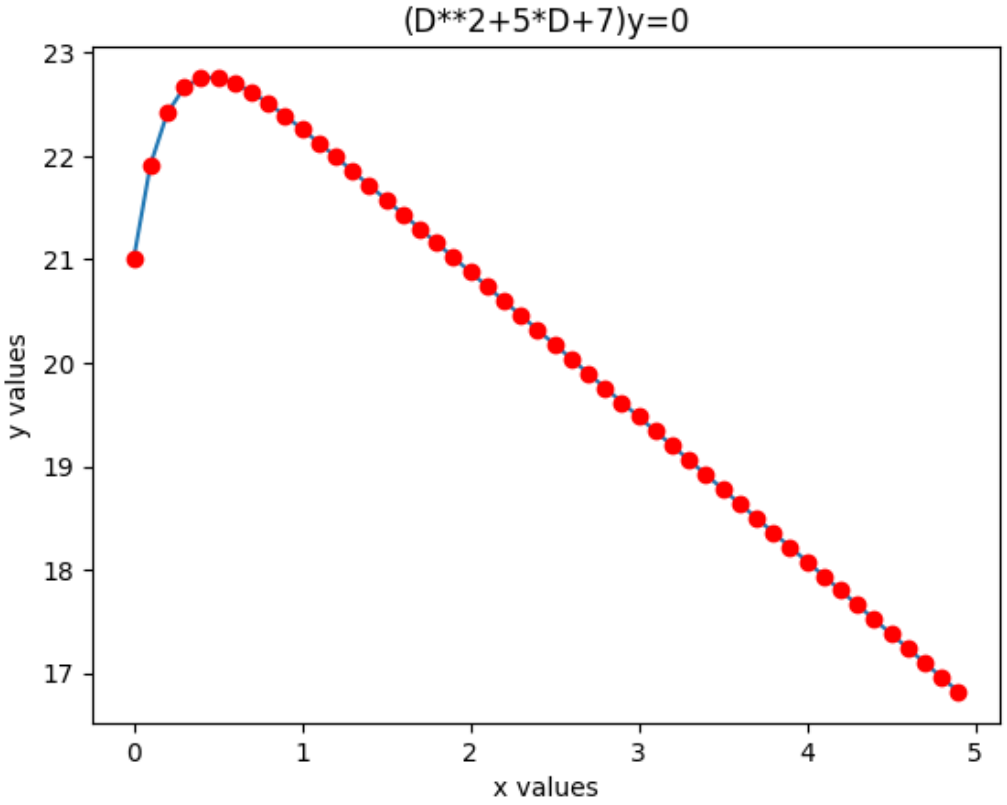


$4)\frac{d^2y}{dx^2}+5\frac{dy}{dx}+7=0$

$y(0)=21, y'(0)=12$

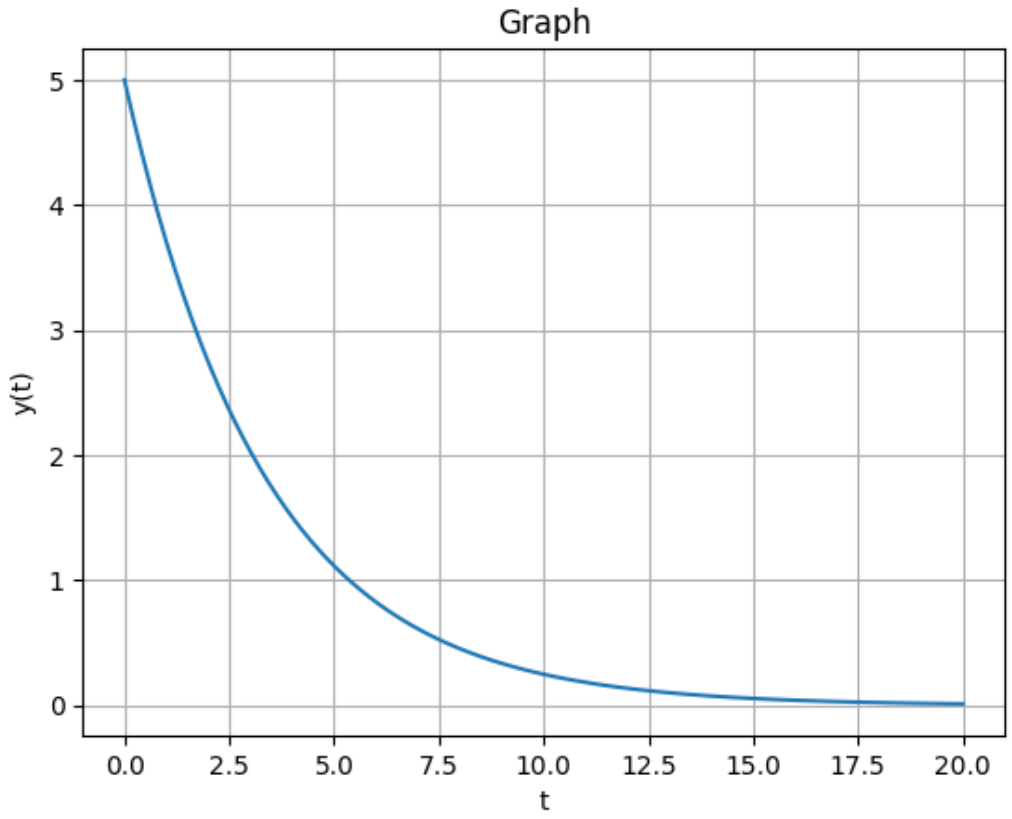
[206]:
```
import matplotlib.pyplot as plt
from scipy.integrate import odeint
import numpy as np

def f(u,x):
    return [u[1], -5*u[1]-7]
y0=[21,12]
xs=np.arange(0,5,0.1)
us=odeint(f,y0,xs)

ys=us[:,0]
```

```
plt.plot(xs, ys, '-')
plt.plot(xs, ys, 'ro')

plt.xlabel('x values')
plt.ylabel('y values')
plt.title('(D**2+5*D+7)y=0')

plt.show()
```



Consider the first-order ordinary differential equation (ODE) given by: $dx/dy = -ky$**, where** $k = 0.3$ **and** $y(0) = 5.0$ Write python code to numerically solve this ODE using odeint function from the scipy.integrate module. Then, plot the solution curve of $y(t)$ over the intergral $0 \leq t \leq 20$

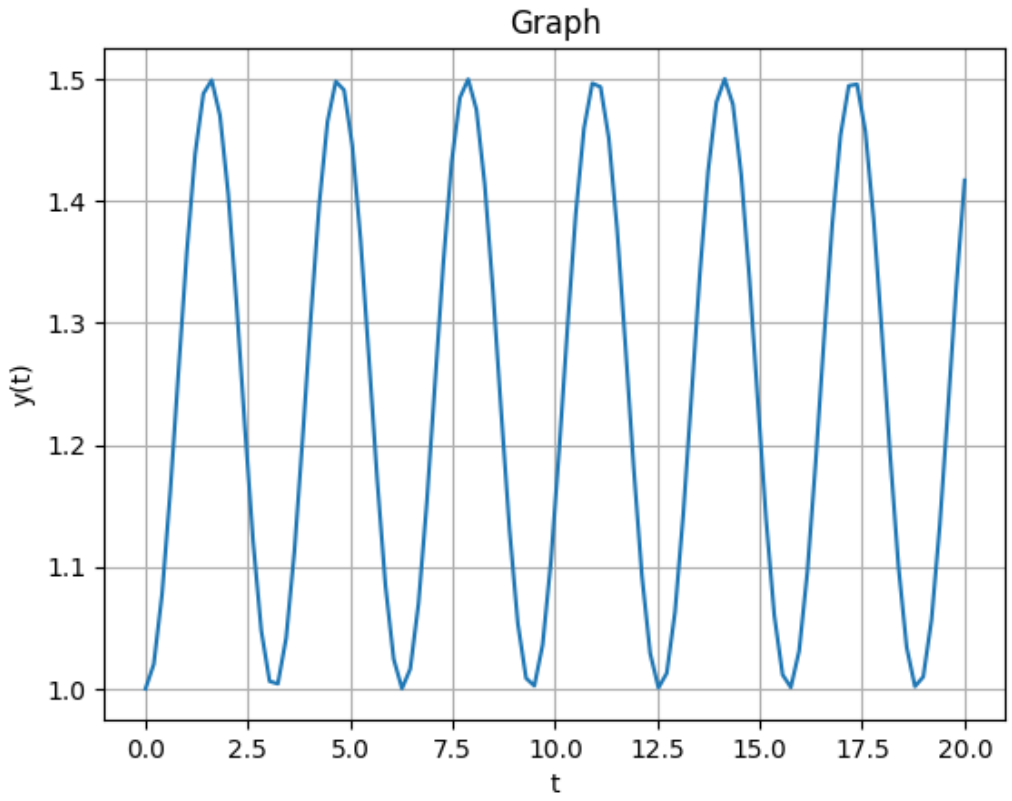
[1]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

#define the ODE function
def dy_dx(y, x):
    k = 0.3
    return -k * y

x = np.linspace(0, 20, 100)
y0 = 5.0

#solve the ODE numerically
y = odeint(dy_dx, y0, x)

#plot curve
plt.plot(x, y)
plt.xlabel("t")
plt.ylabel("y(t)")
plt.title("Graph")
plt.grid(True)
plt.show()
```



$sin(t) * cos(t)$**, with initial condition** $y(0) = 1$ Write python code to numerically solve this ODE using the odeint function from the scipy.integrate module. Then, plot the solution curve of y(t) over the interval $0 \leq t \leq 20$

[2]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def dt_dy(y, t):
    return np.sin(t) * np.cos(t)

t = np.linspace(0, 20, 100)
y0 = 1.0
```

```
# solve the ODE numerically
y = odeint(dt_dy, y0, t)

# plot curve
plt.plot(t, y)
plt.xlabel("t")
plt.ylabel("y(t)")
plt.title("Graph")
plt.grid(True)
plt.show()
```



4.2 Mathematical models of first order differential equations

4.2.1 Predator Prey Model Assumptions

The Lotka-Volterra predator-prey model makes a few important assumptions about the environment and the dynamics of the predator-prey populations:

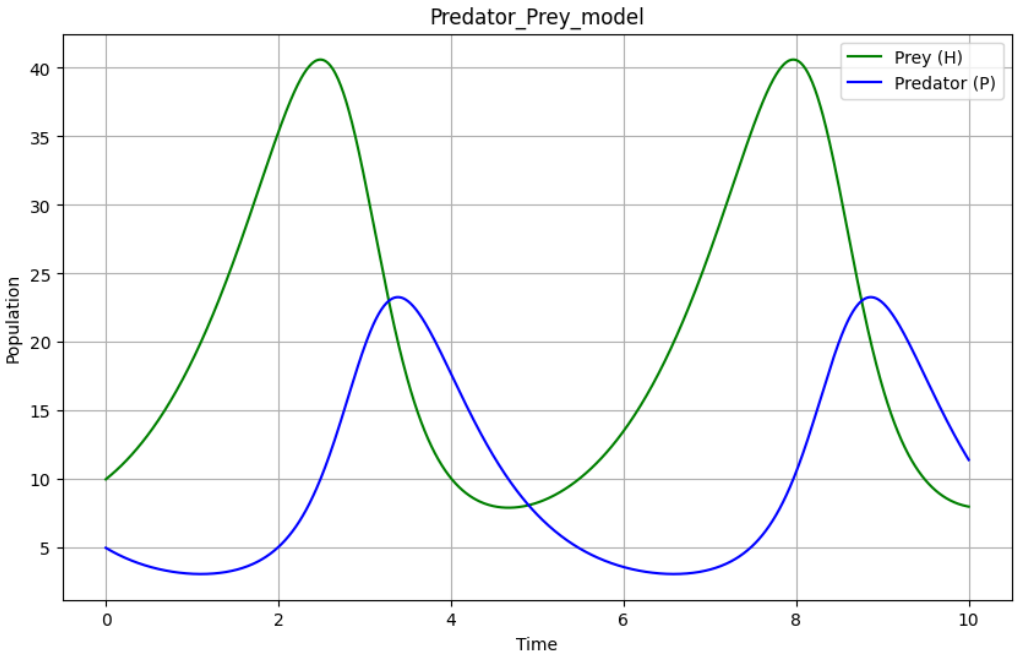
1. The Prey population finds ample food at all times
2. In the absence of a predator, the prey grows at s rate of a proportional to the current the population; thus $dx/dt = αx $, $\alpha > 0$,when $y = 0$.
3. The food supply of the predator population depends entirely on the prey populations.
4. In the absence of a prey, the predator dies out; thus $dy/dt = -βy $, $\beta > 0$, when $x = 0$

The General Formula

Based on the assumptions for this model we have the following two-equation system of autonomous, first-order, nonlinear differential equations: (1) $ $\frac{dx}{dt} = \alpha x - \delta xy = x(\alpha - \delta y)$$

[5]:
```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def predator_prey_model(y, t, a, b, c, d):
    H, P = y
    dHdt = a * H - b * H * P
    dPdt = -c * P + d * b * H * P
    return [dHdt, dPdt]

H0 = 10.0  # initial population of prey
P0 = 5.0   # initial population of predator

# Parameters
a = 1.0
b = 0.1
c = 1.5
d = 0.75

t = np.linspace(0, 10, 1000)

solution = odeint(predator_prey_model, [H0, P0], t, args=(a, b, c, d))
```

```
H = solution[:, 0]
P = solution[:, 1]

plt.figure(figsize=(10, 6))
plt.plot(t, H, label="Prey (H)", color="g")
plt.plot(t, P, label="Predator (P)", color="b")
plt.xlabel("Time")
plt.ylabel("Population")
plt.title("Predator_Prey_model")
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:

5 Conclusion

This course, "Calculus Using Python(MAT111-2)" instructed by Dr. A John Kasper, has equipped students with the necessary skills to bridge the gap between theoretical calculus concepts and their practical applications through Python programming.

We began with a thorough revision, followed by a deep dive into the fundamentals of Python programming, encompassing installation, syntax, data structures, and functionalities. This foundation enabled us to explore various plotting techniques, including standard plots, scatter plots, slope fields, vector fields, contour plots, and streamlines, allowing for effective data visualization.

Unit 2 ventured into symbolic and numeric computations, leveraging the power of SymPy and NumPy packages. We honed our skills in manipulating polynomials, rational functions, trigonometric expressions, exponentials, and logarithms. Furthermore, the course delved into solving algebraic equations, exploring the concepts of limits, derivatives, integrals, and series expansions.

Finally, Unit 3 empowered us to tackle differential equations, a cornerstone of calculus. We learned to solve first and second-order differential equations, visualize their solution curves, and construct mathematical models using these equations.

By effectively combining theoretical knowledge with practical programming skills, this course has provided a valuable foundation for applying calculus to solve real-world problems in various scientific and engineering disciplines.