Question 1:

1. Write a Python program to swap two variables,

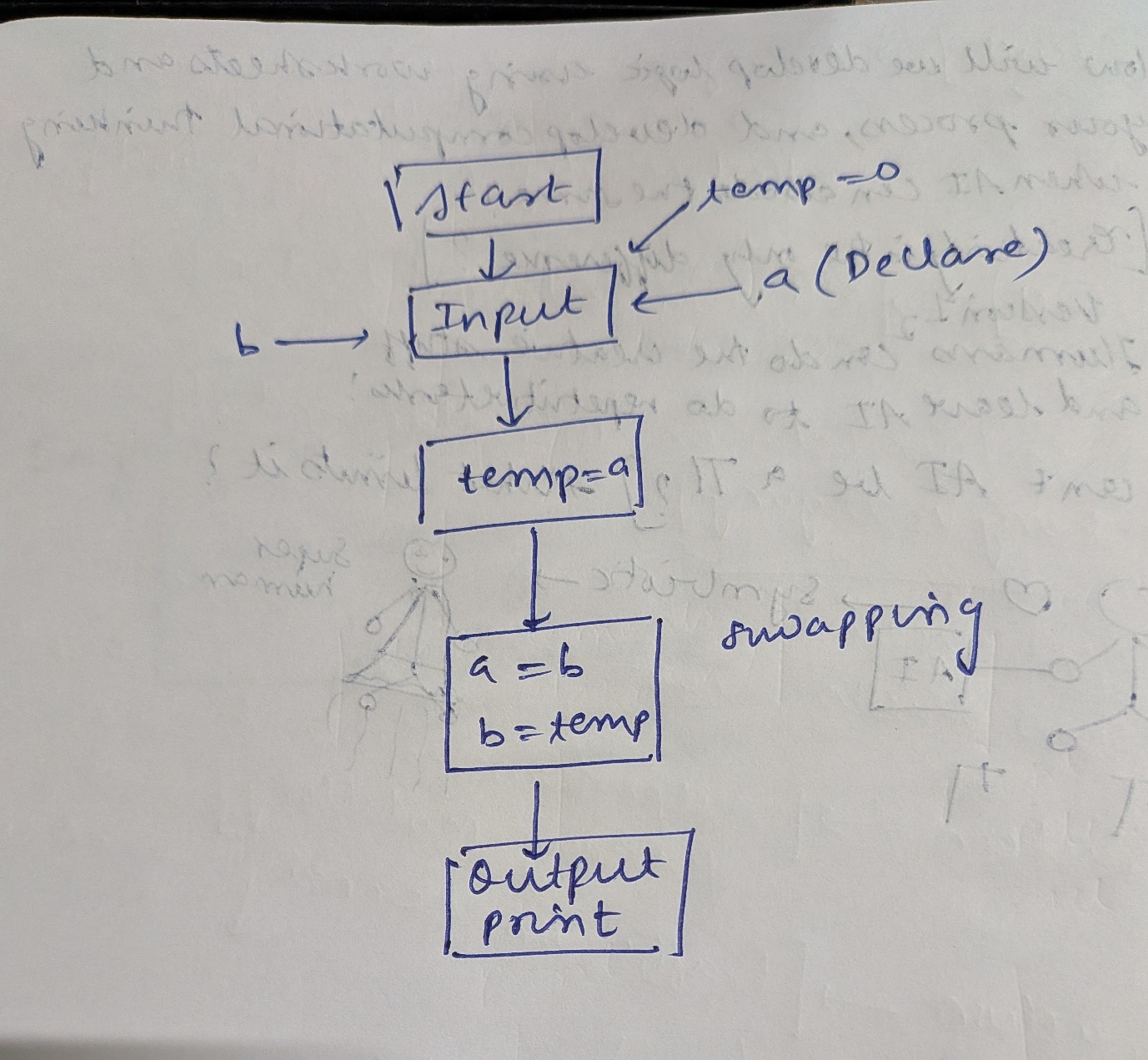
(i) with temp,

(ii) without temp.

Solution:

(i)With temp:

* Initialise two integer variables with names a and b
* Initialise a third variable called temp and use it to store one value, which gets erased when interchanged
* Use this third value to get back the value which was lost and then swap them

Flow Chart:

Code (Formatted):

1. a=5

2. #Initialise a

3. print(a)

4. #printing a for reference

5. b=10

6. #initialise b

7. print(b)

8. #print b for reference

9. temp=a

10. #initialising temp and setting it to a

11. a=b

12. #making a as b

13. b=temp

14. #retrieving value of a

15. print(a)

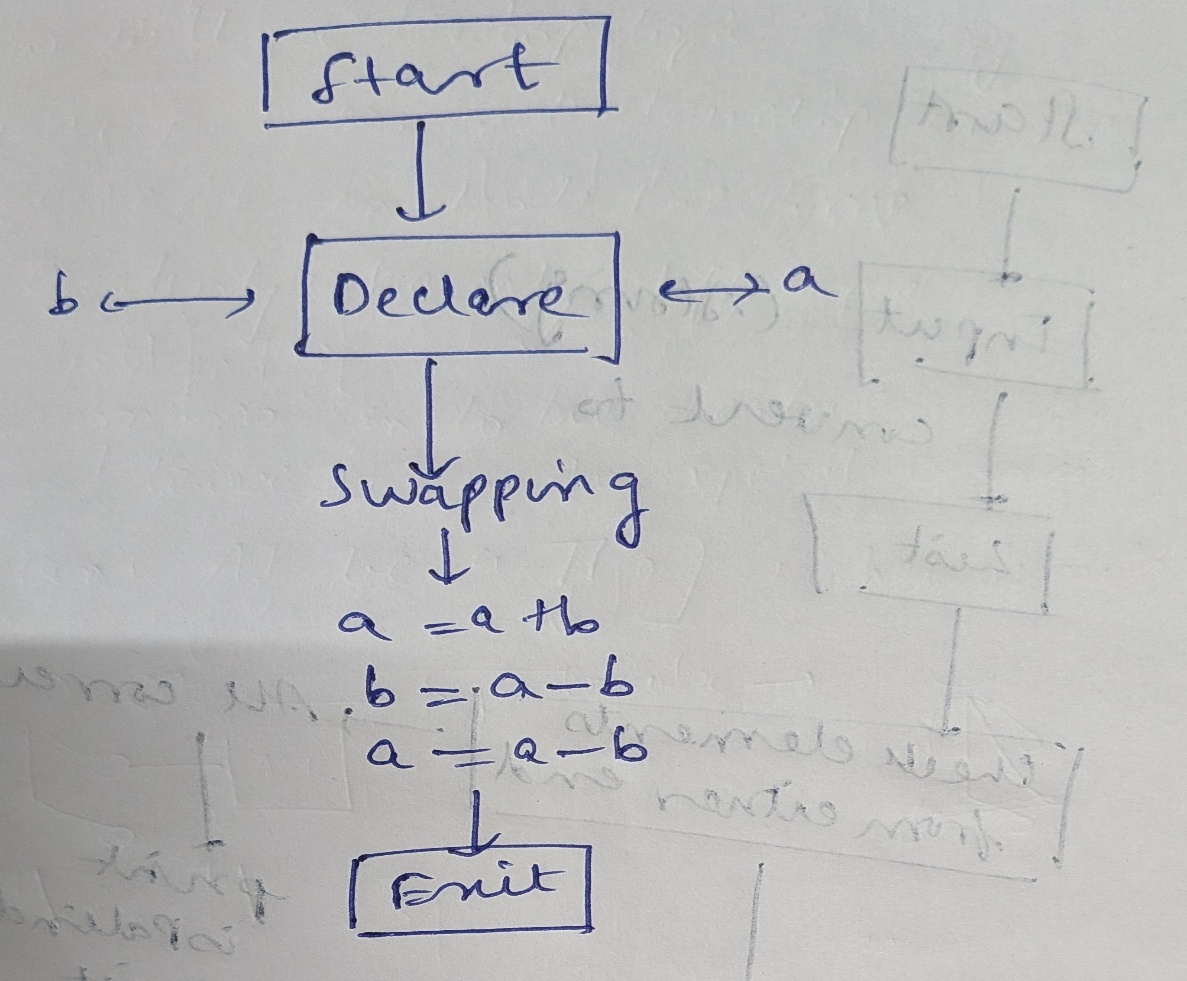
16. #printing a

17. print(b)

18. #printing b

(ii)Without temp:

* Initialise two integer variables named a and b
* Use either one of them to store their sum, the idea is to subtract one element from there in order to get the other
* Like making a to a+b, then removing b from there to get back a

Flow Chart:

Code (Formatted):

1. a=5

2. #Initialise a

3. print(a)

4. #printing a for reference

5. b=10

6. #initialise b

7. print(b)

8. #print b for reference

9. a=a+b

10. #using without temp by storing the sum

11. b=a-b

12. #retrieving value of a

13. a=a-b

14. #retrieving value of b

15. print(a)

16. #printing a

17. print(b)

18. #printing b

**Question 2:**

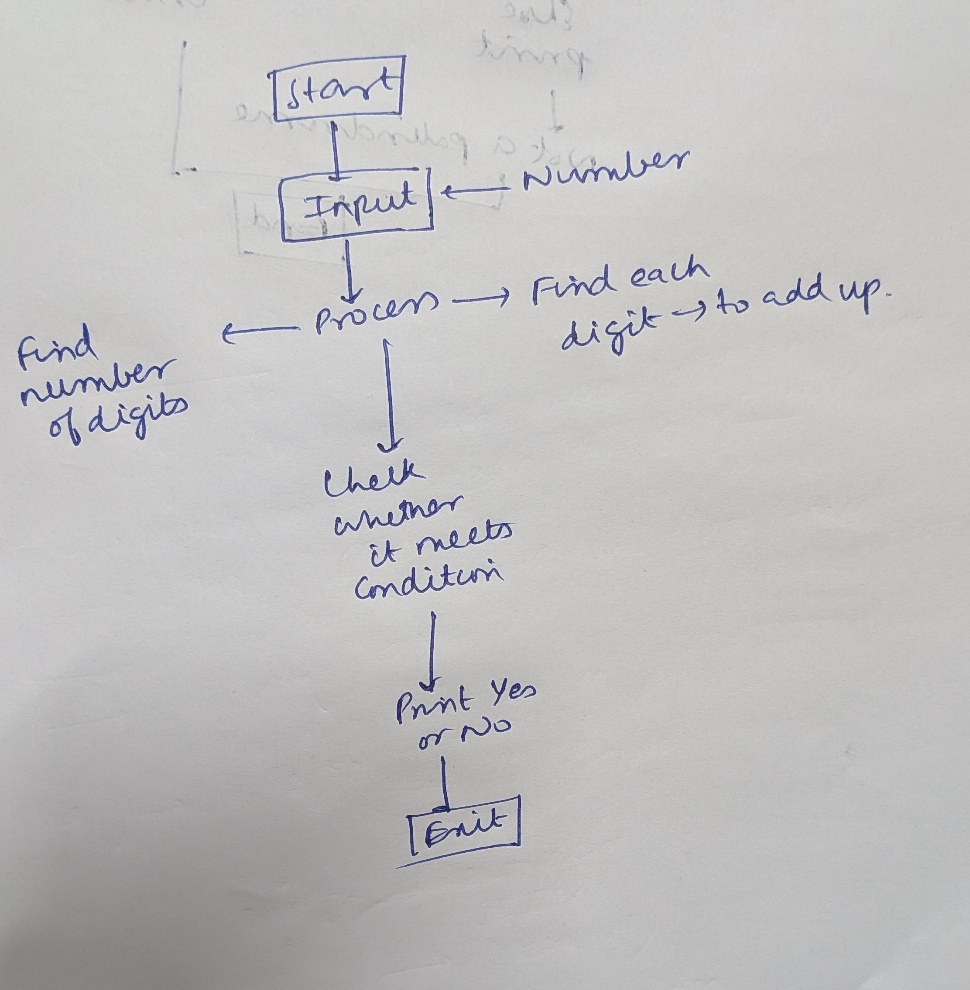
2. To check if a number that is given is an Armstrong number:

**Armstrong Number:**

A number that equals the sum of its digits each raised to the number of digits in the number is called an Armstrong number

Solution:

* Take a number as input
* Retrieve the digits in the number
* Check if it satisfies that condition

Flow Chart:

Code:

Method 1:

1. x=input("Enter a number to check: ")

2. #taking an input

3. list\_of\_digits=[]

4. #initialising a list to store all digits

5.

6. for element in x:

7.     list\_of\_digits.append(element)

8. x=int(x)

9. #converting x to an integer

10. divisor=int(1)

11. #initialising divisor as 1, later we increment it by 10 each time

12. quotient=1

13. #initialise quotient as something greater than or equal to 1

14. count=0

15. while quotient>=1:

16.     quotient=x/divisor

17.     divisor=divisor\*10

18.     count+=1

19. #while loop runs until quotient becomes less than 1

20. # although we can rely on list length, it will fail if person enters a number starting with zeroes

21. # so using while loop

22. no\_of\_digits=count-1

23. #Now we have no\_of\_digits

24. print(no\_of\_digits)

25. #now we calculate the sum

26. sum=0

27. for element in list\_of\_digits:

28.     sum=sum+(int(element)\*\*no\_of\_digits)

29. print(sum)

30. #printing sum for reference

31. if sum==x:

32.     print("Yes")

33. else:

34.     print("No")

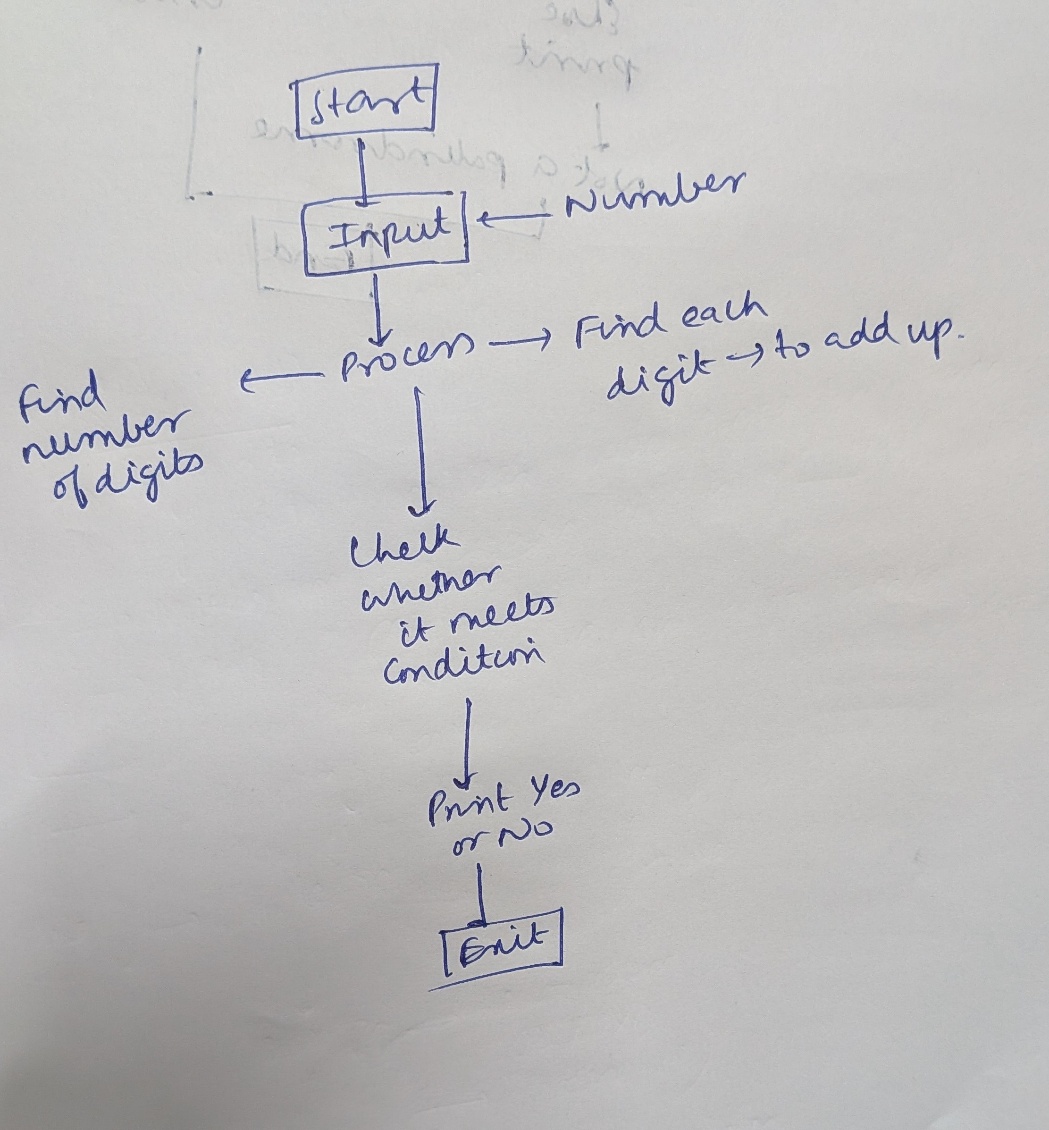
Question 3:

To check if a given number is a palindrome

Solution:

To check whether it is a palindrome:

* First convert input from string into a list
* Then check whether elements equidistant from the end match up or not for each such pair
* Print the boolean value of the variable is\_Palindrome

Flow Chart: 1

Code:

Iteration 1:

I assume that palindrome as strings not as a number

1. Which fails if there are zeroes before the first non zero digit:

2. from math import \*

3. #I need to use the floor function in case the palindrome has odd number of digits

4. x=input("Enter a number to check: ")

5. #taking input

6. string\_to\_list=[]

7. #initialising empty list to fill up with elements

8. for element in x:

9.     string\_to\_list.append(element)

10. length=len(string\_to\_list) . #storing the length of list in a separate variable

12. is\_Palindrome=bool(True)

13. for index in range(0,floor(length/2)):

14.     if string\_to\_list[index]==string\_to\_list[-index-1]:

15.         continue

16.     else:

17.         is\_Palindrome=False

18.         break

19. print(is\_Palindrome)for loop compares the equidistant elements at either ends 20. # and breaks even if one fails and then prints whether it is a palindrome or not

Output:



Iteration 2:

Additional code to remove all the zeroes before the first non-zero digit

1. from math import \*

2.# I need to use the floor function in case the palindrome has odd number of digits

4. x=input("Enter a number to check: ")

5. # taking input

6. string\_to\_list=[]

7. copy\_list=[]

8. # initialising empty list to fill up with elements

9. for element in x:

10.     string\_to\_list.append(int(element))

11.     #appending all elements to the list makes it[0,0,0,0,1,2,2,1]

12. first\_nonzero=bool (False)

13.     #setting to false

14. first\_nonzero\_index=0;

15. # intialising randomly the index checker

16. for element in string\_to\_list:

17.     if element! =0:

18.         first\_nonzero=True

19.         copy\_list.append(element)

20.     if element==0:

21.         if (first\_nonzero==True):

22.             copy\_list.append(element)

23. print(string\_to\_list)

25. print(copy\_list)

26. #probably some error due to reading and removing from the same list

27. #maybe copy will work

28. length=len(copy\_list)

30. # storing the length of list in a separate variable

31. is\_Palindrome=bool (True)

32. for index in range (0, floor(length/2)):

33.     if copy\_list[index]==copy\_list[-index-1]:

34.         continue

35.     else:

36.         is\_Palindrome=False

37.         break

38. # for loop compares the equidistant elements at either ends and breaks even if one fails print(is\_Palindrome)

Output:



Question 4:

To add and print the sum of three distinct integers, if at least two are found equal, printing 0

Solution:

* Take 3 inputs from the user
* Check the value of (1-2)\*(2-3)\*(3-1), the three inputs
* If it is zero then give 0, else give back the sum of the three distinct integers

Flow Chart:

Code:

1. x=int(input("Enter first number: "))

2. y=int(input("Enter second number: "))

3. z=int(input("Enter third number: "))

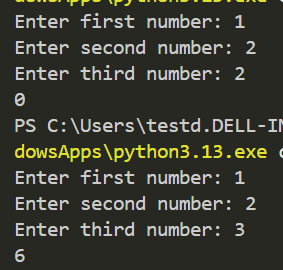
4. if (x-y)\*(y-z)\*(z-x)==0:

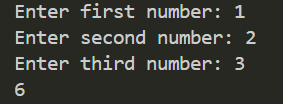
5.     print(0)

6. else:

7.     print(x+y+z)

Output:





Question 5:

To calculate the total wholesale cost for buying 60 books, when given the price, discount, conditional shipping charges

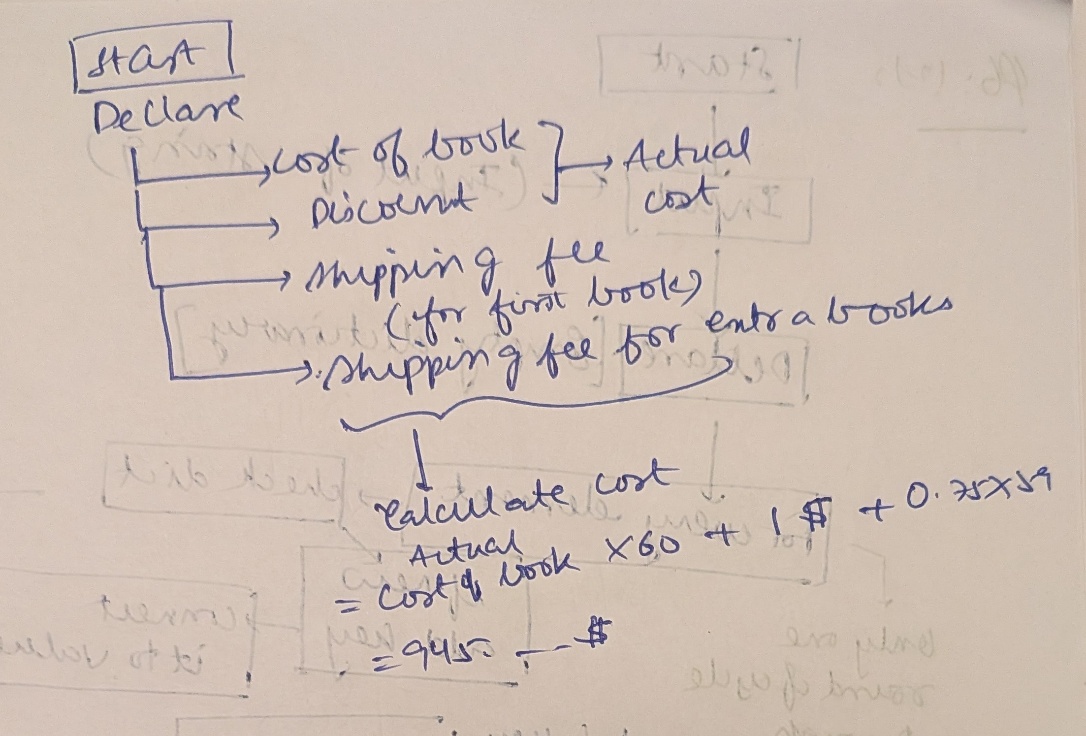
Solution:

Declare all the constants

* The initial cost
* Discount percentage
* Discounted cost
* Shipping for first copy
* Shipping for additional copies

Declare the variables:

* Number of books to be bought
* Final price

Flow Chart:

Code:

1. MRP=24.95

2. DISCOUNT=40

3. Actual\_MRP=MRP\*(1-(DISCOUNT/100))

4. FIRST\_COPY\_SHIPPING=3

5. ADDITIONAL\_COPY\_SHIPPING=0.75

6. NUMBER=60

7. COST=0

8. repeats=0

9. while repeats<NUMBER:

10.     if (repeats==0):

11.         COST=Actual\_MRP+FIRST\_COPY\_SHIPPING+COST

12.         repeats+=1

13.

14.     elif(repeats>0):

15.         COST=Actual\_MRP+ADDITIONAL\_COPY\_SHIPPING+COST

16.         repeats+=1

17.

18.

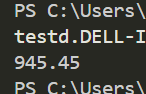
19. print(f"{COST:.2f}")

20. #One way to reduce the number of digits is to use an f string

21. # Use f"{:.2f}""

22.

Output:



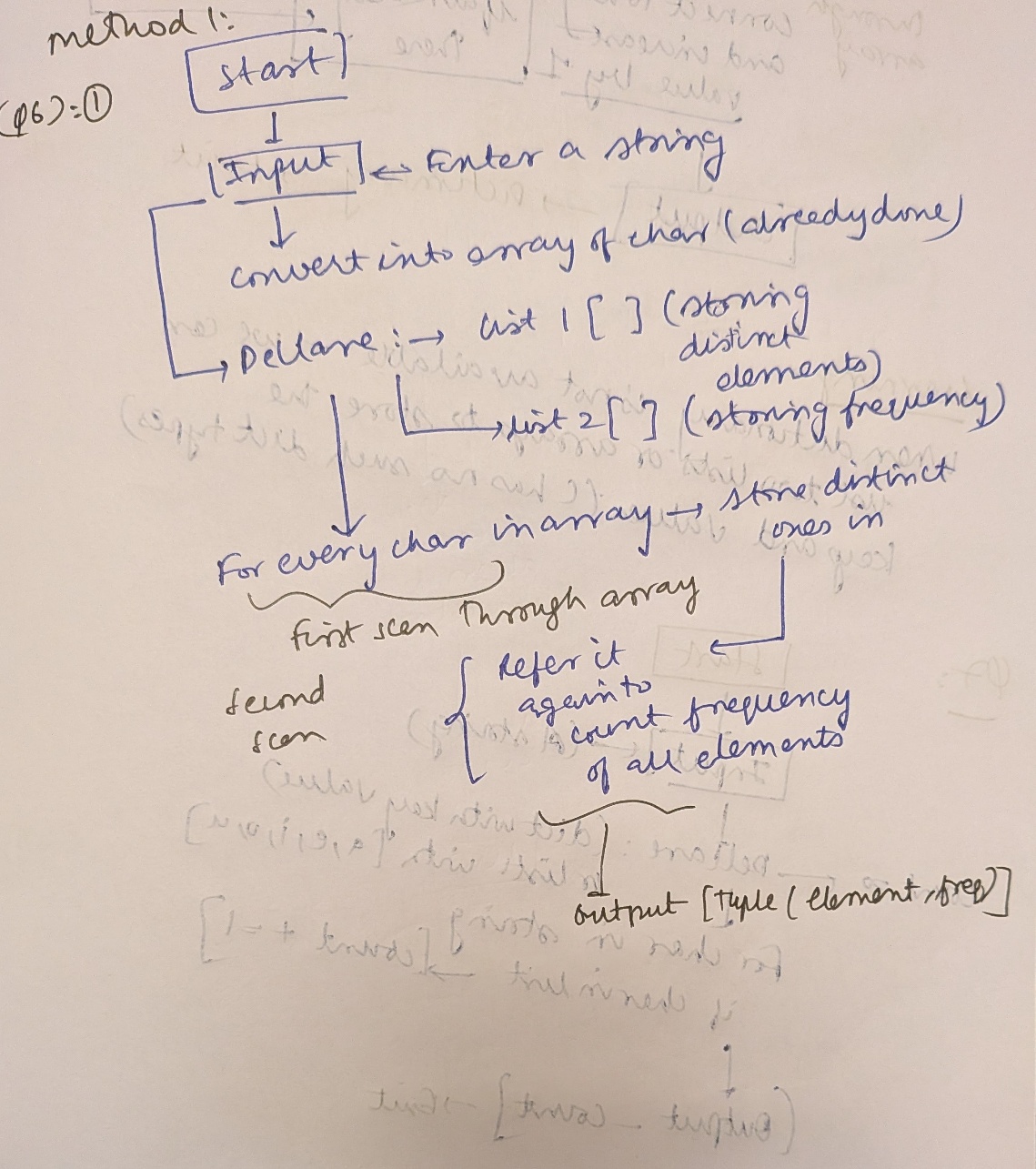
Question 6:

To make a program which lists out all elements with the number of times that each element has repeated

Method 1:

* Iterating through the array (string is an array of characters) and preparing a list of the distinct elements
* Using the list of distinct elements to compare with other elements and increase their count, formatting the chars and frequency into a tuple

This method involves seeing through the array for each and every distinct variable, hence it is less efficient

Flow Chart:

Code:

1. x=input("Enter the string:")

2. #Taking a random string

3. list1=[]

4. #One list to store characters which are read already

5. list2=[]

6. #One list to store characters and frequency

7. for ch in x:

8.     if ch in list1:

9.         continue

10.     else:

11.         list1.append(ch)

12. #Build list with all distinct elements to compare it later

13. for element in list1:

14.     times=0

15.     for ch in x:

16.         if ch==element:

17.             times+=1

18.     list2.append((element,times))

19. #Build list with frequency also along with the element itself

20. print(list2)

21. #2 for loops right

Method 2:

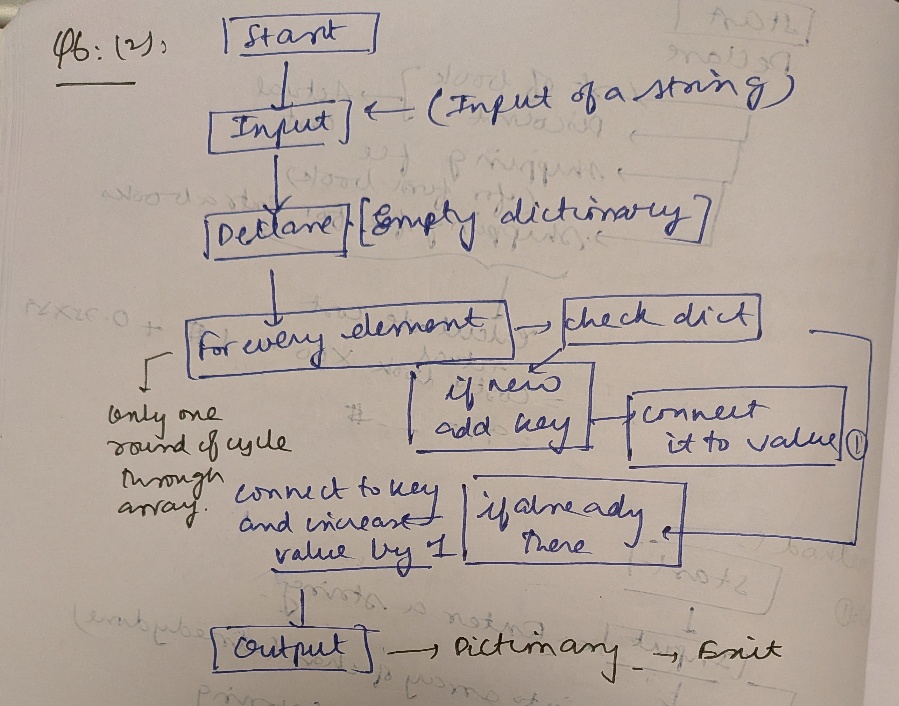
Using dictionaries

Dictionaries are like functions; you map a key to its value inside a dictionary

Then, you can ask it to return the value mapped, when you pass that key through it.

The advantage of this is that, instead of first seeing distinct elements, preparing a list, and then scanning again, I can use a feature of the dictionary in python, i.e. it will create a new key if that doesn’t exist already

* This means whenever I see an element that is new, I can assign the value 1 to that key, and if I see an element that I have already seen, I can increase its count by one
* By this method, I need to go through the entire string only once, 2 for loops vs only one

Flow Chart:

Code:

1. x=input("Enter the string: ")

2. x=x.lower()#removing anomalies like lower and upper cases

3. x=x.strip()#removing spaces, which are usually counted by python as separate element

4.#Creating empty dictionary

5. dict={}#empty dictionary

6. for ch in x:

7.     if ch in dict:

8.         dict[ch]+=1

9.     else:

10.         dict[ch]=1

11. print(dict)

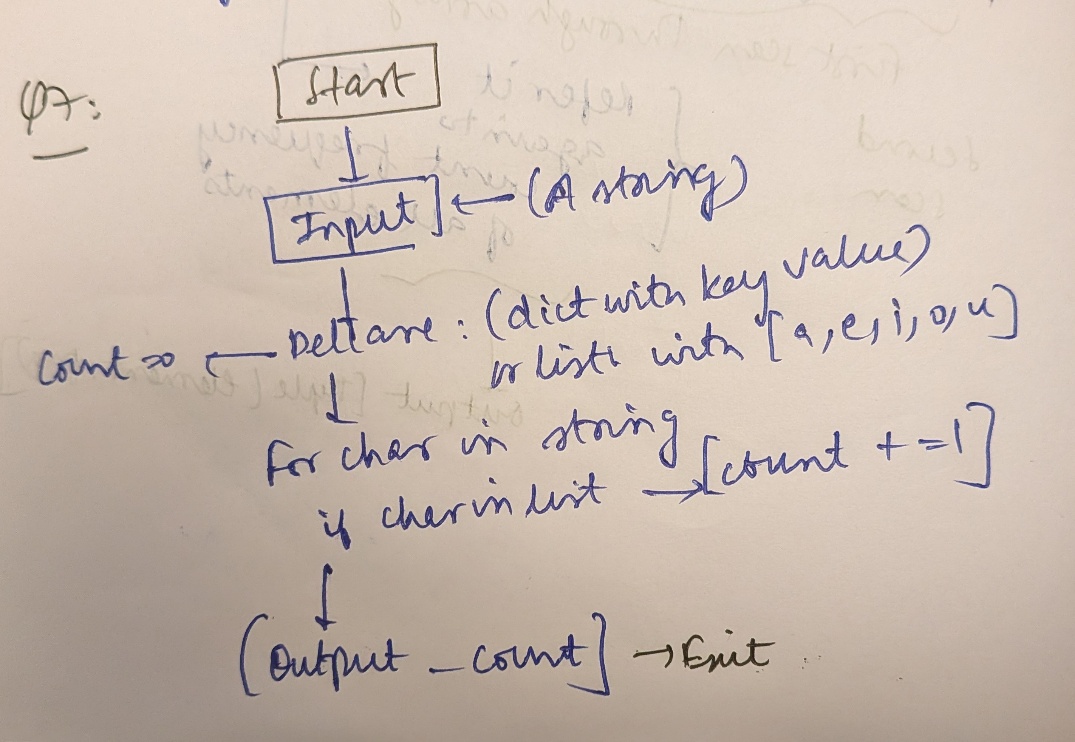
Question 7:

To make a function which takes in a string as input and gives the number of vowels as the output

Solution:

Method 1:

* Initialise number of vowels to 0
* Iterate through string and compare with the set of vowels, if you find any such element, then increment the counter of vowels by 1 each time
* You can use either individually checking whether it matches a vowel or simply check if it is in a list of the vowels. Second method is better, but why?-because it again requires only one complete iteration through string, as compared to five iterations through string by comparing each alphabet
* So it is better in terms of time

Flow Chart:

Code:

1. y=input("Enter a string: ")

2. def no\_of\_vowels(x):

3.     list\_vowels=['a','e','i','o','u']

4.     no\_of\_vowels=0;

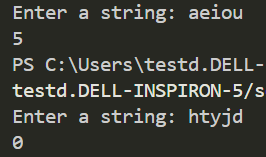
5.     for element in x:

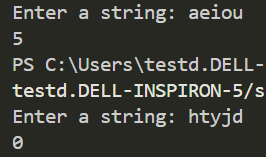
6.        if element in list\_vowels:

7.            vowels+=1;

8.     return no\_of\_vowels

9. print(no\_of\_vowels(y))

Output:



Question 8:

To write a python program , which takes three integers and determines if they satisfy any of the three conditions

* A\*B=C
* A+B=C
* A-B=C

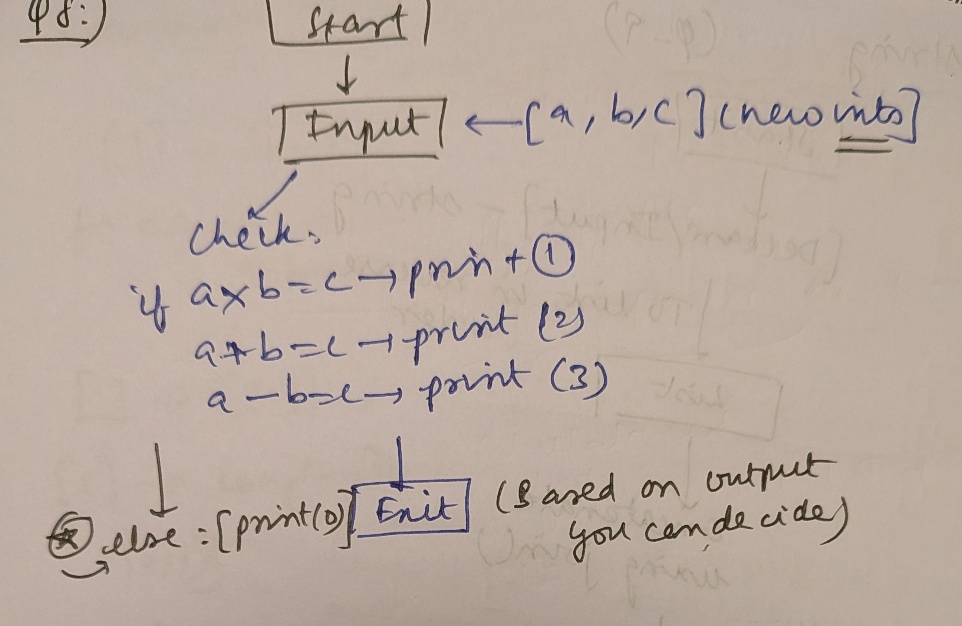
Solution:

Take 3 inputs, which are in the form of strings, because all inputs are treated as strings by python

Convert the strings into integers

If any one is satisfied, print the number in the terminal

If none are satisfied, then print that None are satisfied, then exit

Flow Chart:

Code:

1. x=int(input("Enter the first integer: "))

2. y=int(input("Enter second integer: "))

3. z=int(input("Enter the third integer: "))

4. if (x+y==z):

5.     print(1)

6. if (x+z==y):

7.     print(2)

8. if (x\*y==z):

9.     print(3)

10. else:

11.     print("0")

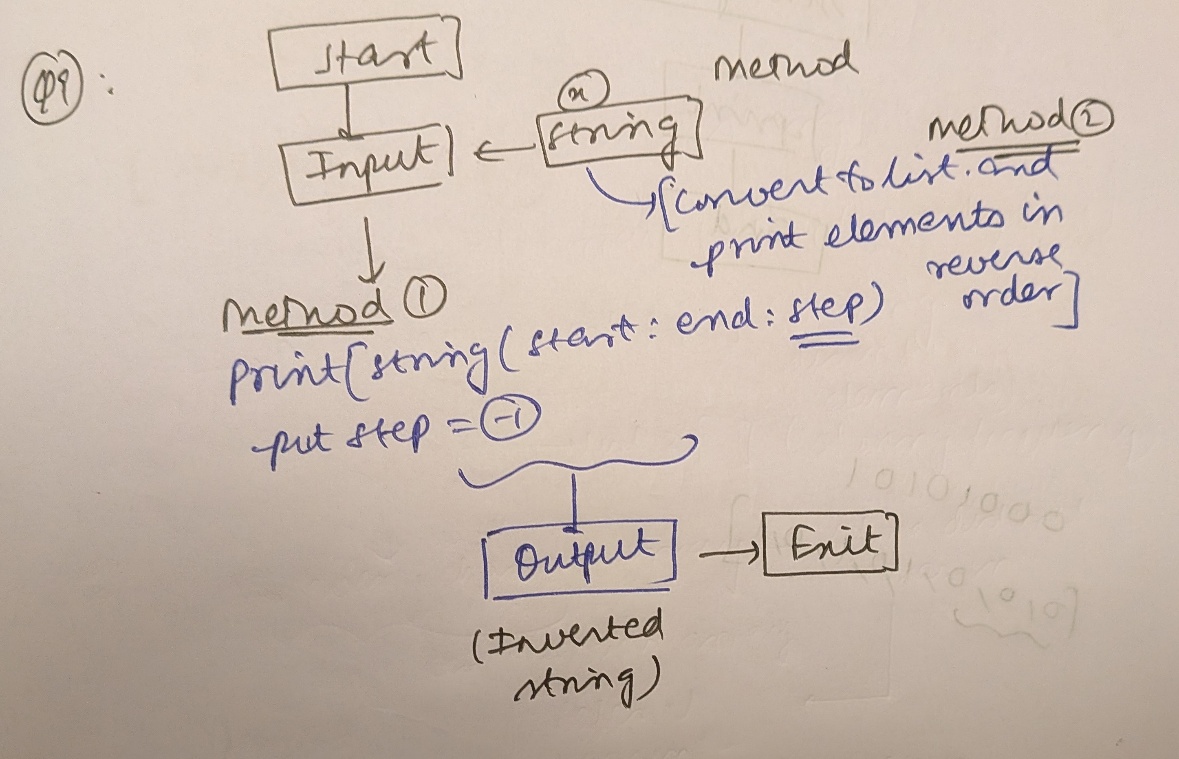
Question 9:

Given a string as input, return or print to console the reversed string as output

Solution:

Method 1:

Use the string reverser using negative steps

Flow Chart:

Code:

1. x=input("Enter a string: ")

2. print(z[::-1])

Output:



Method 2:

Putting string into list in reverse order and then join it back to get the reversed string

Flow Chart:

Included in the first picture shown above

Code:

1. z=input("Enter a string: ")

2. string\_to\_list=[]

3. index=1

4. for element in z:

5.     string\_to\_list.insert(0,element)

6.

7. x="".join(string\_to\_list)

8. print(x)

Output:



Question 10:

To determine if two strings are isomorphic:

Isomorphic:

It is like checking whether a function is possible by mapping elements at a particular index for one string to elements at same index in the other string, if yes, they are isomorphic, else they are not

Procedure:

Start going through both of the strings, while doing so, also:

* Map elements of first string to that of the second string, using a dictionary
* When any single mismatch is detected, then return False and exit immediately as there is no need to check further

Flow Chart:

Code:

1. string\_in1=input("Enter a string: ")

2. string\_in2=input("Enter another string with same length:")

3. #These are my test cases for checking whether it works or not

4. #P1:

5. #Being able to create a function or mapping using dictionaries

6. #Throwing an error when two strings which are given don't follow this rule

7. #P2:

8. #Comparing two strings,both or one of which may have spaces

9. #test cases are hello world & ifllg wgrld

10. #P3:

11. #Throwing error when two strings are not of the same length, probably will already be

12. #taken care of

13. dict1={}

14. for char in string\_in1:

15.     print("Element in is: "+char)

16.     index\_comp=string\_in1.index(char)

17.     #if a character repeats this goes back

18.     print(index\_comp)

19.     compared\_element=string\_in2[index\_comp]

20.     print("Element compared to is: "+compared\_element)

21.     #element at same index

22.     if char in dict1:

23.        print("Character already read before, checking...")

24.        #If element that was read now is already in the dictionary, then check if it is still

25.        #unique mapping

26.        if dict1[char]!=compared\_element:

27.            print("Mapping not proper")

28.            exit(1)

29.            #Returns that mapping is not proper and exits

30.     else:

31.         dict1[char]=compared\_element

32.         #Otherwise adds a new key to the dictionary

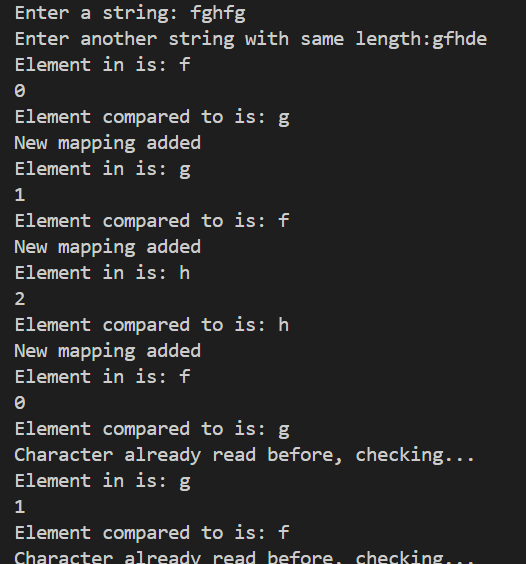
33.         print("New mapping added")

34.

Errors encountered:

Trying to read the index of the character in a string, which has repeated characters, will always return the index for the first occurrence of that element, not moving ahead

Output(While debugging):



Notice, how index should have gone to 4 instead it went to 3, so it is better to use manual incrementing only

Method 2:

1. string\_in1=input("Enter a string: ")

2. string\_in2=input("Enter another string with same length:")

3. #These are my test cases for checking whether it works or not

4. #P1:

5. #Being able to create a function or mapping using dictionaries

6. #Throwing an error when two strings which are given don't follow this rule

7. #P2:

8. #Comparing two strings,both or one of which may have spaces

9. #test cases are hello world & ifllg wgrld

10. #P3:

11. #Throwing error when two strings are not of the same length, probably will already be

12. #taken care of

13. dict1={}

14. index=0

15. is\_isomorphic=True

16. for char in string\_in1:

17.     index\_comp=index

18.     #if a character repeats this goes back

19.     compared\_element=string\_in2[index\_comp]

20.     #element at same index

21.     if char in dict1:

22.        #If element that was read now is already in the dictionary, then check if it is still

23.        #unique mapping

24.        if dict1[char]!=compared\_element:

25.            print("Mapping not proper")

26.            is\_isomorphic=False

27.            exit(1)

28.            #Returns that mapping is not proper and exits

29.     else:

30.         dict1[char]=compared\_element

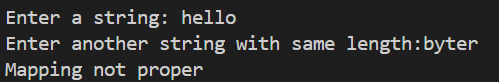
31.         #Otherwise adds a new key to the dictionary

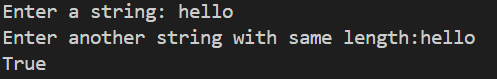
32.     index+=1

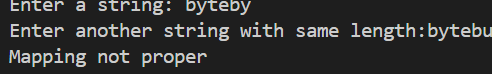
33. print(is\_isomorphic)

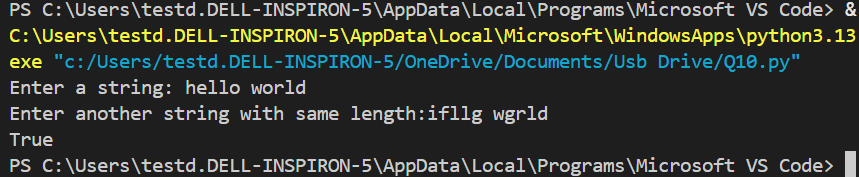
34.

Output:







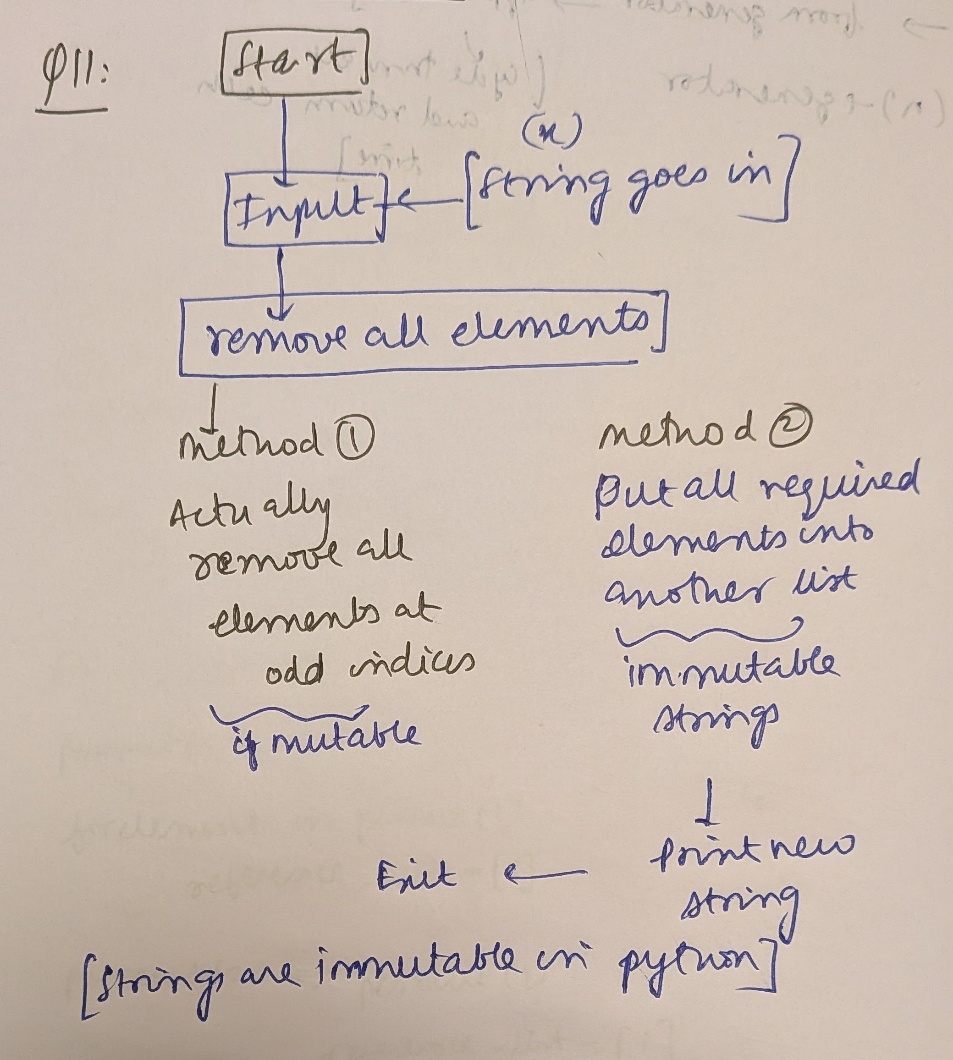


Question 11:

To remove all the elements at odd-indices, base 1 from a given string and print it out

Solution:

* Elements at a particular index in a string can be accessed by using string[i], because string is an array of characters
* Using this, we append all the elements at even indices into a new list and join them back.
* Writing to a string is not allowed because strings are immutable in Python

Flow Chart:

Code:

1. from math import \*

2. x=input("Enter a string: ")

3. list\_out=[]

4. index=0

5. length=len(x)

6. for integer in range(0,floor(len(x)/2)):

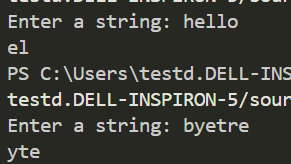
7.     index=2\*integer+1

8.     list\_out.append(x[index])

9. output=''.join(list\_out)

10. print(output)

Output:





Question 12:

To print out all possible sequences with c,a,t,d,o,g

Solution:

* Generate permutations of the list [1,2,3,4,5,6]
* Then map each number uniquely to one of c,a,t,d,o,g
* Print out the result

Why use numbers?

We use numbers, not characters for permutation because, given a certain permutation of numbers, we can easily determine, without much thinking that the next element to be going in, is length+1, where length is length of the given array. But for characters, we must decide which character to put in

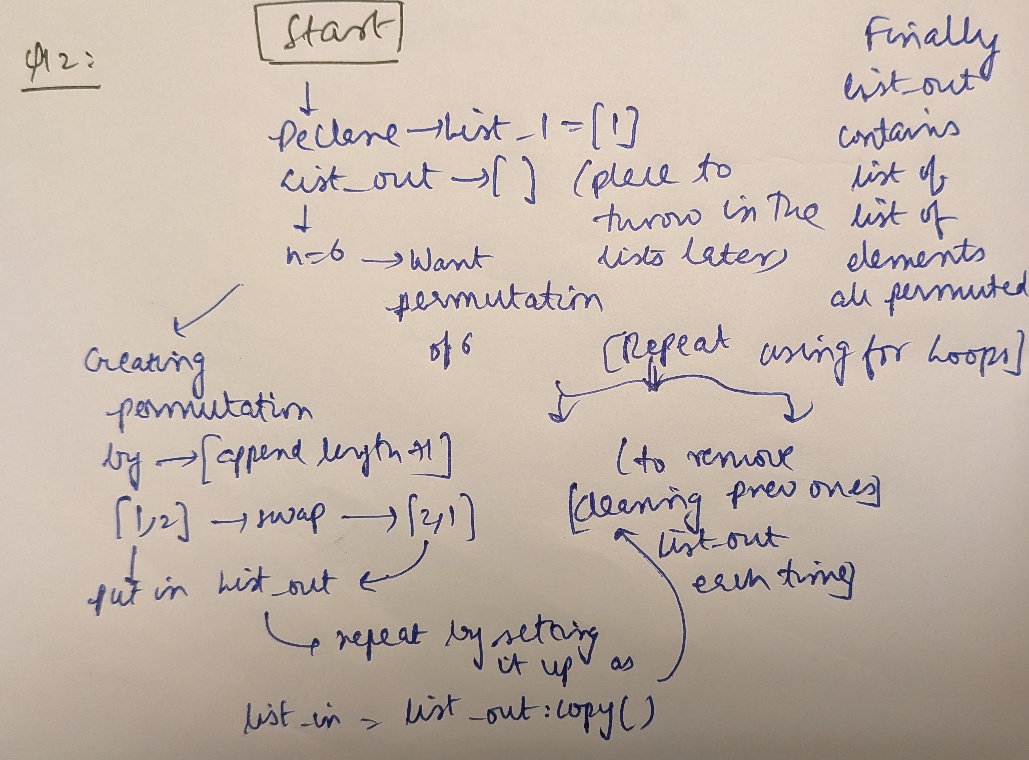
How to generate permutations?

Method 1:

Permutations can be generated by starting with [], adding, each time, an element=length of list+1 to it at all possible positions, then storing them up and repeating. However, running even with 6 elements, is very-very memory heavy, so a better solution is required

This method is called iterative method, not recursion.

It relies on first storing all the permutations of previous case before proceeding to the next one

Flow Chart:

Code:

1. list1=[1]

2. list\_out=[]

3. n=6

4. length=len(list1)

5. for index in range(0,length+1):

6.     list1.insert(index,length+1)

7.     list\_out.append(list1.copy())

8.     list1.remove(length+1)

9. if n<=1:

10.     print(list\_out);

11.     exit(0)

12. if n>1:

13.     repeats=0;

14.     while repeats<n-2:

15.         list1=list\_out.copy()

16.         list\_out.clear()

17.         for element in list1:

18.             length=len(element)

19.             for index in range(0,length+1):

20.                 element.insert(index,length+1)

21.                 list\_out.append(element.copy())

22.                 element.remove(length+1)

23.         repeats+=1

24. print(list\_out)

25. for element in list\_out:

26.     element=[str(x) for x in element]

27.     string\_out=''.join(element)

28.     for num in string\_out:

29.                 if num=='1':

30.                     string\_out=string\_out.replace('1','c')

31.                     #replace creates a new string, so without reassignment it does nothing

32.                 if num=='2':

33.                     string\_out=string\_out.replace('2','a')

34.                 if num=='3':

35.                     string\_out=string\_out.replace('3','t')

36.                 if num=='4':

37.                     string\_out=string\_out.replace('4','d')

38.                 if num=='5':

39.                     string\_out=string\_out.replace('5','o')

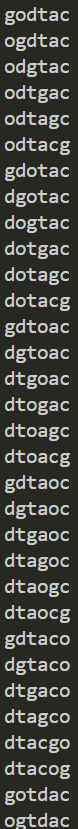
40.                 if num=='6':

41.                     string\_out=string\_out.replace('6','g')

42.     print(string\_out)

Output:

Putting all 720 of them here is quite difficult, but I included few screenshots



For full output, refer to end of document

Method 2:

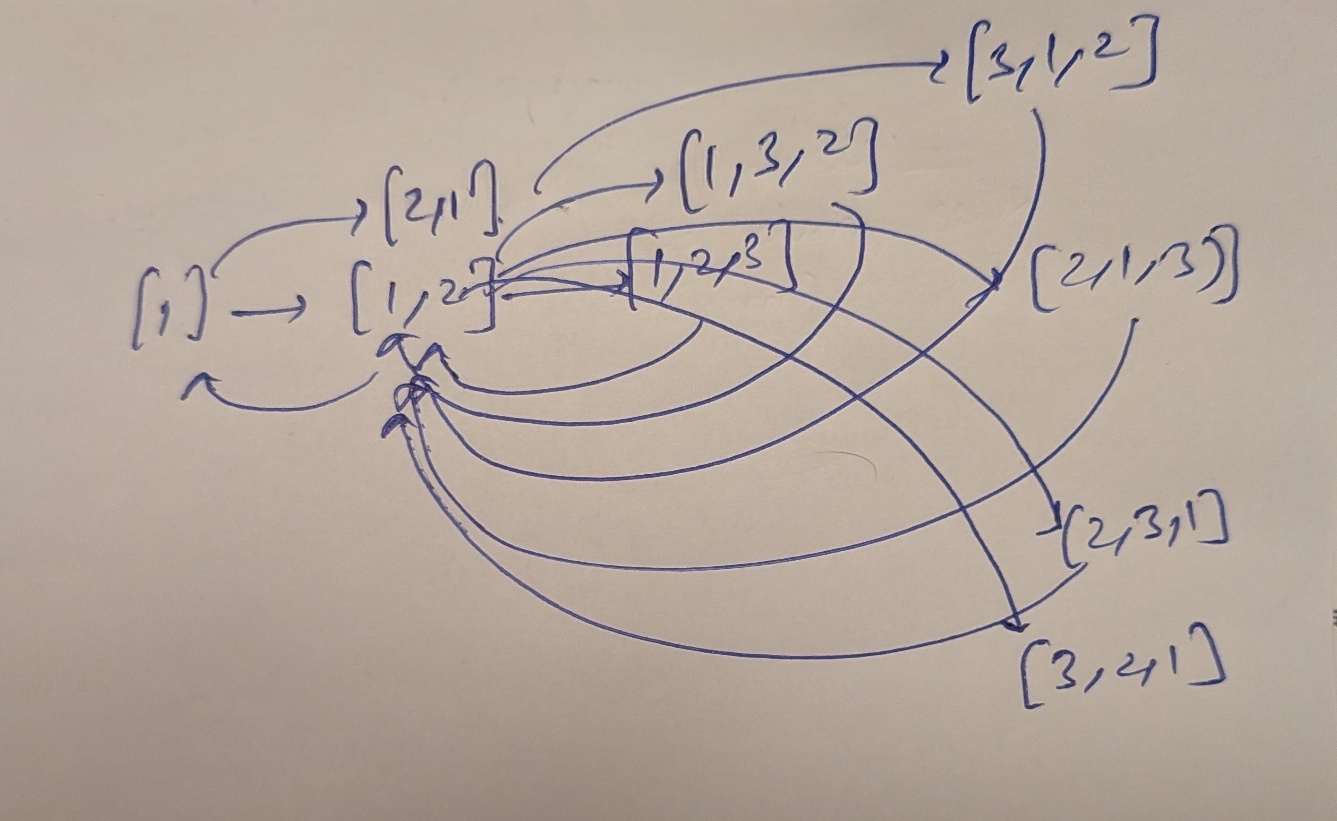
Method of Recursion and back tracking:

Recursion is a method where you determine value of function on the basis of the previous call.

But the difference between this method and the first one is the concept of backtracking

Backtracking:

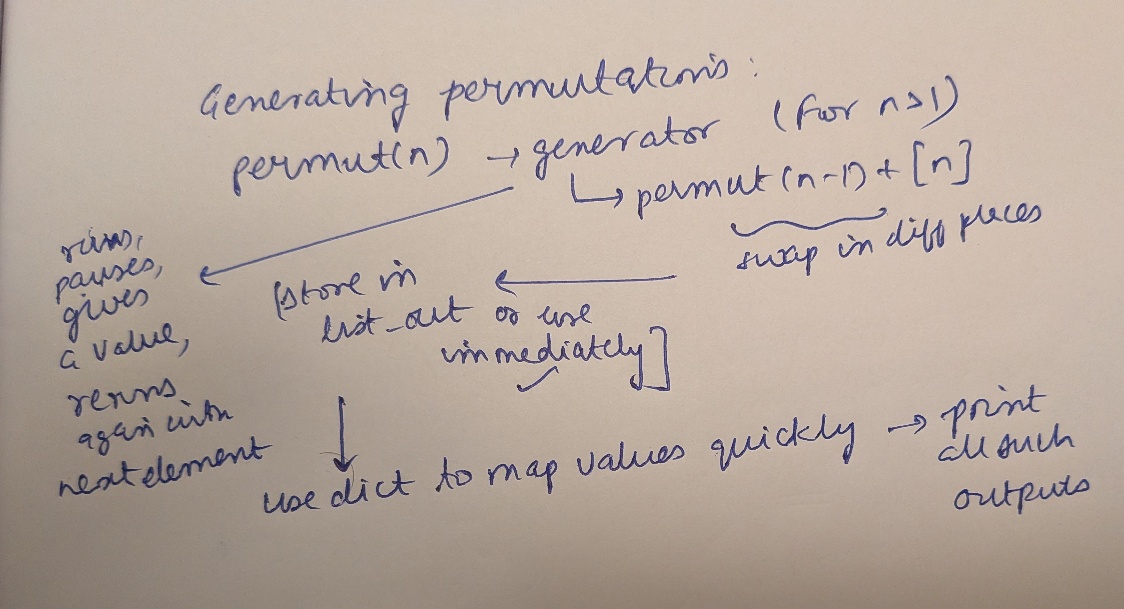
Once the recursion reaches the end, i.e. once it produces one single permutation of six elements, it will not stop. Instead, it moves back to where it generated a 5-element permutation and goes all the way back and starts over again, like this:

Concept of yield

Yield is used in place of return for functions that generate things, like factorials and permutations. Unlike return, which terminates the function after returning whatever it was supposed to, yield pauses the function, returns a value, then starts running again from the next iteration

This means, we can generate all possible permutations without actually storing them up in memory each time, instead generating one each time, making sure they don’t repeat

This will save lot of memory, because storing those many lists and clearing and writing is expensive.

Flow Chart:

Code:

1. #How to write recursion to generate permutations?

2. #The basic concept is that swapping in various ways generates different permutations

3. #Think of it this way

4. #[1], I need to get permutations upto number 6

5. #base case [1]

6. reference={

7.     1:"a",

8.     2:"c",

9.     3:"t",

10.     4:"d",

11.     5:"o",

12.     6:"g"

13. }

14. def permut(n):

15.     if n==1:

16.         yield [1]

17.     if n>1:

18.         for element in permut(n-1):

19.            for index in range(0,len(element)+1):

20.                new\_element=element.copy()

21.                #making sure I don't modify the lists in permut(n-1), becuase it remembers this element until the cycle is complete

22.                new\_element.insert(index,n)

23.                #then inserting at different indices

24.                #this for loop is exited once yield is complete

25.                #hence you don't need to reset new\_element to element.copy() again

26.                yield new\_element

27.

28. for element in permut(6):

29.         for i in element:

30.             element[element.index(i)]=reference[i]

31.         string\_out="".join(element)

32.         print(string\_out)

33.             #replacing numbers by corresponding alphabets

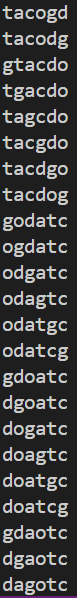
34.

35. #converting them into strings

36. #

37. #the basis for all permutations is swapping

38. # doing it with numbers means that the element to be inserted is very clear

Output:

For full output and solution for all the questions:

Link: