1. **For any string str, generate a function to reverse the order of characters in each word   
   within a sentence while still preserving whitespace and initial word order, such as follows  
   Input: str = "Hello, Python"  
   Output: "olleH, nohtyP"**

# Answer of 1

s="Hello Python" # initial string

stringlength=len(s) # calculated length of the list

slicedString=s[stringlength::-1] # slicing

print (slicedString) # printing the reversed string

1. **Write the function to process a list of integers nums and an integer target, return indices   
   of the two numbers such that they add up to target. You may assume that each input   
   would have exactly one solution, and you may not use the same element twice. You can   
   return the answer in any order.  
   Input: nums = [1,2,3,4], target = 7  
   Output: [2,3]  
   Explanation: Because nums[2] + nums[3] == 3+4 == 7, we return [2, 3]**

class answer(object):  
def add(self, num, target):  
ans = []  
for x in range(len(num) - 1):  
for y in range(x + 1, len(num)):  
k = num[x] + num[y]  
if k == target:  
ans.append((x, y))  
return ans  
print(answer().add([1,2,3,4], 7))

1. **For an integer n, write a function to return true if n is palindrome integer. For   
   example, 121 is a palindrome because the reverse sequence of 121 is still 121   
   while 123 is not**

#Answer of 3

def digits(n):

a = (n%10)/1

b = ((n-a)%100)/10

c = ((n-a-10\*b)%1000)/100

return a, b, c

n = 1211

a, b, c = digits(n)

def palindrome(n, a, b, c):

if a==c and b==b and c==a:

return True

print("is a palindrome")

else:

return False

print("is not a palindrome")

palindrome(n, a, b, c)

**4. Create a function to find the longest common prefix string amongst a list of strings.  
If there is no common prefix, return an empty string "". The example is as follows**

**Input: str = ["michael","michelle","mitch"]  
Output: "mi"**

#Answer of 4

def get\_prefix\_str(str\_list):

# taking first string as reference and comparing each char with the corresponding char of the rest of strings

prefix\_str = ''

len\_smallest\_str = min([len(str\_mem) for str\_mem in str\_list])

for i in range(len\_smallest\_str):

if len([0 for ind in range(1, len(str\_list)) if str\_list[0][i] != str\_list[ind][i]]) > 0:

break

prefix\_str += str\_list[0][i]

return prefix\_str

for str\_list in [["tower", "towel", "tow"], ["silicon", "valley", "dream"], ["alex", "aaditya", "aayushma"]]:

print(str\_list)

prefix\_str = get\_prefix\_str(str\_list)

print(f'Prefix is {prefix\_str}')

**5.Write a program that approximates the value of pi by summing the terms of this series: 4  
1  
―4  
3 + 4  
5 ―4  
7 + 4  
9 ― 4  
11 +... The program should prompt the user for n, the number of terms   
to sum, and then output the sum of the first n terms of this series. Have your program   
subtract the approximation from the value of math. pi to see how accurate it is.**

#Answer of 5  
import math  
n = int(input(“Enter the number of terms: “))  
pi = 0  
for r in range(n):  
term = ((-1) \*\* r) \* (4 / (2 \* r + 1))  
pi += term  
print(“The calculated value of pi =“, pi)  
print(“Actual pi value =“, math.pi)  
print(“Difference =“, math.pi - pi)

**6. You have seen that the math library contains a function that computes the square root of   
numbers. In this exercise, you are to write your own algorithm for computing square   
roots. One way to solve this problem is to use a guess-and -check approach. You first   
guess what the square root might be, and then see how close your guess is. You can use   
this information to make another guess and continue guessing until you have found the   
square root (or a close approximation to it). One particularly good way of making guesses   
is to use Newton's method. Suppose x is the number we want the root of, and guess is the   
current guessed answer. The guess can be improved by using computing the next guess as:   
𝑔𝑢𝑒𝑠𝑠+ 𝑥  
𝑔𝑢𝑒𝑠𝑠  
2  
Write a program that implements Newton's method. The program should prompt the user   
for the value to find the square root of (x) and the number of times to improve the guess.   
Starting with a guess value of 𝑥  
2, your program should loop the specified number of times   
applying Newton's method and report the final value of guess. You should also subtract   
your estimate from the value of math. sqrt (x) to show how close it is.**

# Answer of 6  
import math  
def newtonmethod(x,step):  
p=1  
guess = x/2  
while p <= step:  
guess = (guess + (x/guess)) / 2  
p+=1  
return guess  
answer = newtonmethod(49,7 )  
print (answer,’\nDifference with square root of x: ‘, answer - (math.sqrt(49)))