**Lab Assignment - 3**

**Github Repository:** <https://github.com/daksharma3163/Assignment-3>

**Q1:**

def find\_longest\_word(text):

words = text.split()

longest\_word = max(words, key=len)

return longest\_word

text = "Python programming is powerful"

print(f"The longest word is: {find\_longest\_word(text)}")



**Q2:**

def replace\_middle\_with\_star(s):

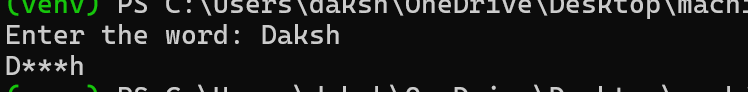
if len(s) <= 2:

return s

return s[0] + '\*' \* (len(s) - 2) + s[-1]

text = input("Enter the word: ")

print(replace\_middle\_with\_star(text))



**Q3:**

def is\_isogram(word):

word = word.lower()

seen = {}

for char in word:

if char in seen:

return False

seen[char] = True

return True

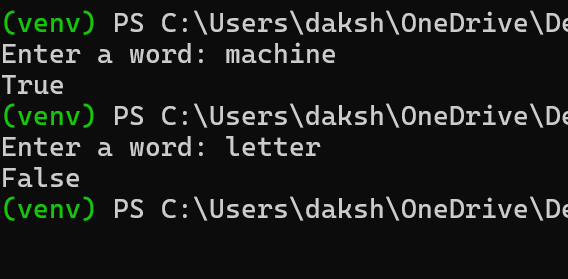
word = input("Enter a word: ")

if(is\_isogram(word)):

print(“True”)

else:

print(“False”)



**Q4:**

def every\_nth\_char(s, n):

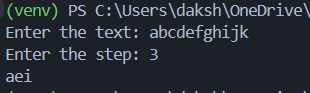
for i in range(0,len(s),n+1):

print(s[i],end='')

string = input("Enter the text: ")

n = int(input("Enter the step: "))

every\_nth\_char(string, n)



**Q5:**

def reverse\_by\_words(s):

word\_list = list(map(str, s.split()))

result = ""

for word in word\_list:

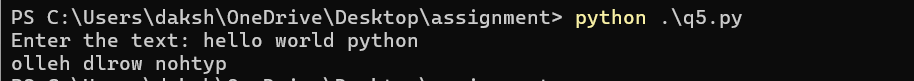
word = word[::-1]

result = result + word + " "

return result

s = input("Enter the text: ")

print(reverse\_by\_words(s))



**Q6:**

def compress\_consecutive(s):

if not s:

return []

count = 1

compressed = ""

for i in range(1, len(s)):

if s[i] == s[i-1]:

count += 1

else:

compressed += s[i-1] + str(count)

count = 1

compressed += s[-1] + str(count)

return compressed

s = input("Enter the text: ")

print(compress\_consecutive(s))



**Q7:**

def count\_islands(grid):

def dfs(i, j):

if i < 0 or i >= 10 or j < 0 or j >= 10 or grid[i][j] != '1':

return

grid[i][j] = '0'

dfs(i+1, j) # Down

dfs(i-1, j) # Up

dfs(i, j+1) # Right

dfs(i, j-1) # Left

island\_count = 0

for i in range(10):

for j in range(10):

if grid[i][j] == '1':

dfs(i, j)

island\_count += 1

return island\_count

data = [

"1100000111",

"1000000111",

"0000000111",

"0010001000",

"0000011100",

"0000111110",

"0001111111",

"1000111110",

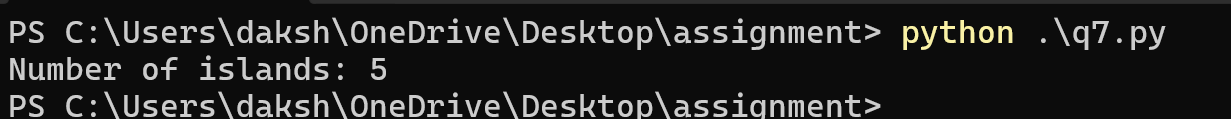
"1100011100",

"1110010000"

]

grid = [list(row) for row in data]

print("Number of islands:", count\_islands(grid)) # Output: 5



**Q8:**

def count\_above\_average(nums):

if not nums:

return 0

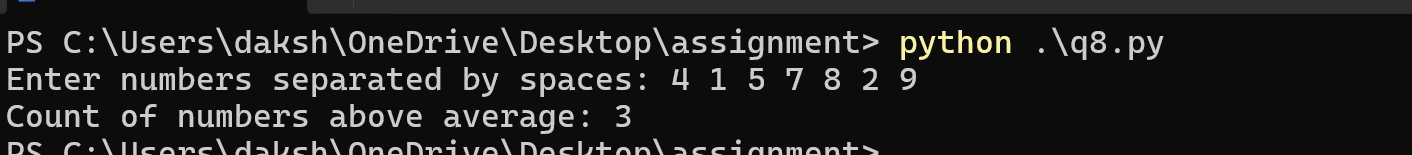
average = sum(nums) / len(nums)

count = sum(1 for num in nums if num > average)

return count

nums = [int(x) for x in input("Enter numbers separated by spaces: ").split()]

print("Count of numbers above average:", count\_above\_average(nums))



**Q9:**

def toggle\_cast\_count(s):

toggled = ""

counts = {'upper':0,'lower':0}

for char in s:

if(char.isupper()):

counts['upper'] += 1

toggled += char.lower()

else:

counts['lower'] += 1

toggled += char.upper()

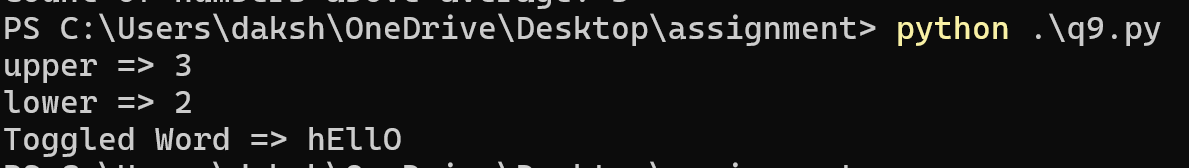
return counts, toggled

counts\_dict, toggled\_word = toggle\_cast\_count("HeLLo")

for key, value in counts\_dict.items():

print(f"{key} => {value}")

print(f"Toggled Word => {toggled\_word}")



**Q10:**

def binary\_search\_recursive(arr, target, low, high):

if low > high:

return -1

mid = (low + high) // 2

if arr[mid] == target:

return mid

elif arr[mid] < target:

return binary\_search\_recursive(arr, target, mid + 1, high)

else:

return binary\_search\_recursive(arr, target, low, mid - 1)

def binary\_search\_iterative(arr, target):

low = 0

high = len(arr) - 1

while low <= high:

mid = (low + high) // 2

if arr[mid] == target:

return mid

elif arr[mid] < target:

low = mid + 1

else:

high = mid - 1

return -1

nums = input("Enter the list: ")

nums = list(map(int,nums.split()))

target = int(input("Enter the element to find: "))

if binary\_search\_iterative(nums, target) != -1:

print("Found")

else:

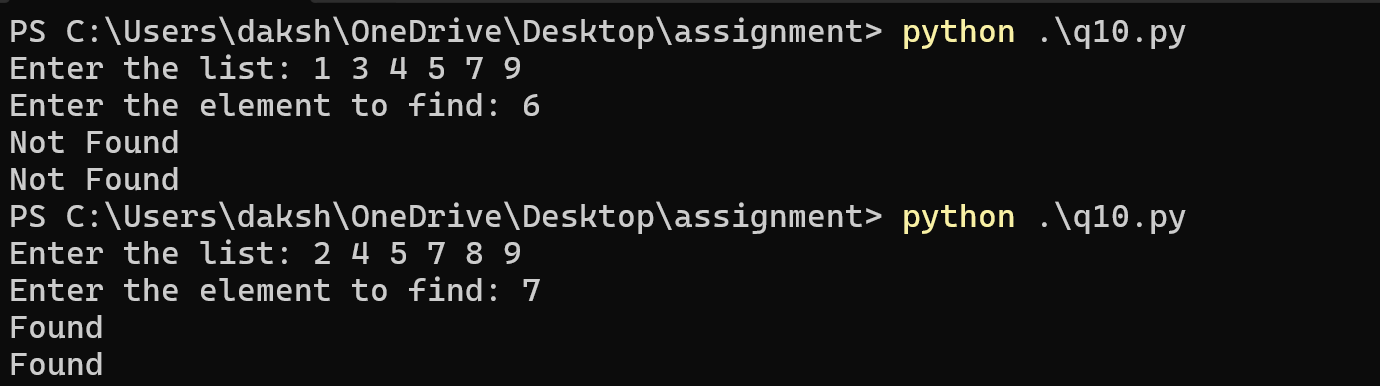
print("Not Found")

if binary\_search\_recursive(nums, target, 0, len(nums) - 1) != -1:

print("Found")

else:

print("Not Found")



**Q11:**

n = int(input("Enter the number : "))

end = int(n \*\* 0.5)

for i in range(2, end + 1):

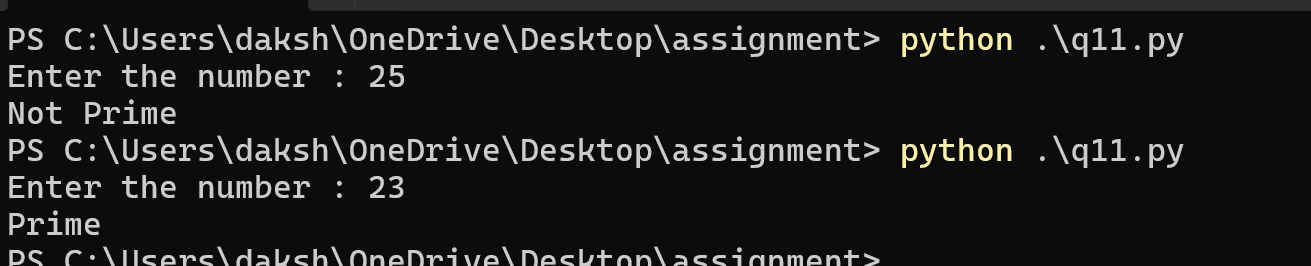
if n % i == 0:

print("Not Prime")

break

else:

print("Prime")



**Q12:**

def flatten\_matrix(matrix):

flattened = []

rows = len(matrix)

cols = len(matrix[0])

for i in range(rows):

for j in range(cols):

flattened.append(matrix[i][j])

return flattened

input\_str = input("Enter matrix rows separated by commas: ")

rows = input\_str.strip().split(',')

matrix = [list(map(int, row.strip().split())) for row in rows]

flattened = flatten\_matrix(matrix)

for element in flattened:

print(element,end=', ')

