**Print list in reverse using while loop**

**Using while:**

numbers=[1,2,3,4,5]

result=[]

i=len(numbers)-1

while i>0:

result.append(numbers[i])

i=i-1

print(result)

**Using for:**

num=[1,2,3,4]

result=[]

for i in range(len(num)-1,-1,-1):

result.append(num[i])

print(result)

**Using slicing:**

# Original list

my\_list = [1, 2, 3, 4, 5]

# Reverse the list using slicing

reversed\_list = my\_list[::-1]

# Output the reversed list

print(reversed\_list) # Output: [5, 4, 3, 2, 1]

**Changing String**

s="Hello World"

modified\_s="h"+s[1:6]+"w"+s[7:11]

print(modified\_s)

original = "banana"

# Replace 'a' with 'o'

modified = original.replace('a', 'o')

print(modified) # Output: bonono

original = "hello"

print(original.upper()) # Output: HELLO

print(original.capitalize()) # Output: Hello

print(original.swapcase()) # Output: HELLO

**Iterating a dict**

my\_dict = {"name": "Alice", "age": 25, "city": "New York"}

for key in my\_dict:

print(key)

for value in my\_dict.values():

print(value)

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

print(f"{key}: {value}")

**Set Operations**

# Example sets

students\_a = {"Alice", "Bob", "Charlie"}

students\_b = {"Bob", "David", "Ella"}

# Union

print("Union:", students\_a | students\_b) # {'Alice', 'Bob', 'Charlie', 'David', 'Ella'}

# Intersection

print("Intersection:", students\_a & students\_b) # {'Bob'}

# Difference (Elements in A but not in B)

print("Difference (A - B):", students\_a - students\_b) # {'Alice', 'Charlie'}

# Difference (Elements in B but not in A)

print("Difference (B - A):", students\_b - students\_a) # {'David', 'Ella'}

# Symmetric Difference (Elements in either set but not both)

print("Symmetric Difference:", students\_a ^ students\_b) # {'Alice', 'Charlie', 'David', 'Ella'}

# Subset

print("A is subset of B:", students\_a.issubset(students\_b)) # False

# Superset

print("A is superset of B:", students\_b.issuperset(students\_a)) # False

# Disjoint

students\_c = {"George", "Hannah"}

print("A and C are disjoint:", students\_a.isdisjoint(students\_c)) # True

# Add an element

students\_a.add("David")

print("After adding David:", students\_a) # {'Alice', 'Bob', 'Charlie', 'David'}

# Remove an element

students\_a.remove("Bob")

print("After removing Bob:", students\_a) # {'Alice', 'Charlie', 'David'}

# Discard an element (won't raise error if element is not found)

students\_a.discard("Eve")

print("After discarding Eve:", students\_a) # {'Alice', 'Charlie', 'David'}

# Pop an element

popped\_element = students\_a.pop()

print("Popped element:", popped\_element) # Could be any of the elements

print("After popping:", students\_a)

# Clear the set

students\_a.clear()

print("After clearing:", students\_a) # set()

# Copy the set

students\_copy = students\_b.copy()

print("Copy of students\_b:", students\_copy) # {'Bob', 'David', 'Ella'}

# Create a frozenset

frozen\_students = frozenset({"Alice", "Bob", "Charlie"})

print("Frozen students:", frozen\_students) # frozenset({'Alice', 'Bob', 'Charlie'})

**Filter dicts that don’t have location key using list comprehension**

dicts = [

{'name': 'Alice', 'age': 25, 'location': 'New York'},

{'name': 'Bob', 'age': 30},

{'name': 'Charlie', 'age': 22, 'location': 'Los Angeles'},

{'name': 'David', 'age': 35}

]

filtered\_dicts=[d for d in dicts if "location" not in d]

print(filtered\_dicts)

**Find highest value in dict and remove corresponding key**

# Sample dictionary

my\_dict = {'a': 10, 'b': 25, 'c': 15, 'd': 30}

# Find the key with the highest value

max\_key = max(my\_dict, key=my\_dict.get)

# Remove the key with the highest value

del my\_dict[max\_key]

print(f"Updated dictionary: {my\_dict}")

**Given a dictionary , interchange the key => as values and values as => keys**

**#using another dict**

my\_dict = {'a': 10, 'b': 25, 'c': 15, 'd': 30}

new\_dict={}

for key,value in my\_dictt.items():

new\_dict[value]=key

print(new\_dict)

**#without using another dict**

def interchange\_key\_value(my\_dict):

for key in list(my\_dict.keys()):

value=my\_dict.pop(key)

my\_dict[value]=key

return my\_dict

my\_dict = {'a': 10, 'b': 25, 'c': 15, 'd': 30}

print(interchange\_key\_value(my\_dict))

**Why Use list(d.keys())?**

If you directly iterate over d.keys() while modifying the dictionary, Python raises a RuntimeError because the size of the dictionary changes during iteration. By using list(d.keys()), we avoid this issue.

**Reverse a number without converting it into string**

def reverse\_num(number):

temp=number

res=0

while temp>0:

last=temp%10

res=res\*10+last

temp=temp//10

return res

print(reverse\_num(1234))

**Swap first and last number without converting into string**

def swap\_first\_last(number):

# Handle single-digit numbers

if number < 10:

return number

# Find the last digit

last\_digit = number % 10

# Find the total number of digits

num\_digits = 0

temp = number

while temp > 0:

temp //= 10

num\_digits += 1

# Find the first digit

first\_digit = number // 10\*\*(num\_digits - 1)

# Remove the first and last digits to get the middle part

middle\_part = number % 10\*\*(num\_digits - 1) // 10

# Reconstruct the number with swapped digits

swapped\_number = (last\_digit \* 10\*\*(num\_digits - 1)) + (middle\_part \* 10) + first\_digit

return swapped\_number

# Example

print(swap\_first\_last(12345)) # Output: 52341

print(swap\_first\_last(9876)) # Output: 6879

**Find the combination of numbers in a given array that product a target sum**

def find\_combinations(nums, target):

result = []

**# Helper function to explore combinations**

def backtrack(start, target, current\_combination):

**# If the target becomes 0, add the current combination to the result**

if target == 0:

result.append(list(current\_combination))

return

**# If the target is negative, stop further exploration**

if target < 0:

return

**# Explore each number starting from the current index**

for i in range(start, len(nums)):

current\_combination.append(nums[i]) **# Add the number to the current** **combination**

backtrack(i, target - nums[i], current\_combination) **# Recur with the updated target**

current\_combination.pop() **# Remove the number to backtrack**

**# Start the backtracking from the first index**

backtrack(0, target, [])

return result

**# Example Usage**

nums = [2, 3, 5]

target = 8

combinations = find\_combinations(nums, target)

print(combinations)

**Combine two sorted array in O(n) or less time**

def merge\_sorted\_arrays(arr1, arr2):

**# Initialize pointers for both arrays and the result array**

i, j = 0, 0

merged\_array = []

**# Traverse both arrays and merge them**

while i < len(arr1) and j < len(arr2):

if arr1[i] < arr2[j]:

merged\_array.append(arr1[i])

i += 1

else:

merged\_array.append(arr2[j])

j += 1

**# If there are remaining elements in arr1**

while i < len(arr1):

merged\_array.append(arr1[i])

i += 1

**# If there are remaining elements in arr2**

while j < len(arr2):

merged\_array.append(arr2[j])

j += 1

return merged\_array

**# Example usage**

arr1 = [1, 3, 5, 7]

arr2 = [2, 4, 6, 8]

result = merge\_sorted\_arrays(arr1, arr2)

print(result)

**Create a list and delete duplicate without using another list**

def remove\_dup(arr):

seen=set()

i=len(arr)-1

while i>=0:

if arr[i] != seen:

seen.add(arr[i])

else:

del arr[i]

i=i-1

return seen

arr = [1,1,1,1,1,2,3,4,]

print(remove\_dup(arr))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def rem(lst):

i=0

while i<len(lst):

j=i+1

while j<len(lst):

if lst[i]==lst[j]:

lst.pop(j)

else:

j=j+1

i=i+1

return lst

lst = [1,1,1,1,1,2,3,4,]

print(rem(lst))

**Create a new list using list comprehension this list should include string that starts with vowels**

words = ['rahul', 'rajeev', 'abhijith', 'ajith', 'elen']

NewList=[word for word in words if word[0] in {'a','e','i','o','u'}]

print(NewList)

**Flat a nested list**

def flatten\_list(nested\_list):

flat\_list=[]

for item in nested\_list:

if isinstance(item,list):

flat\_list.extend(flatten\_list(item))

else:

flat\_list.append(item)

return flat\_list

nested\_list = [1, [2, [3, 4], 5], [6, 7], 8, [9, [10, 11]]]

flattened = flatten\_list(nested\_list)

print(flattened)

**Find the longest consecutive subarray from a given array of integers**

def longest\_subarr(arr):

res=[]

temp=[arr[0]]

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

if temp[-1]==arr[i]-1:

temp.append(arr[i])

else:

temp=arr[i]

if len(temp)>len(res):

res=temp

return res

arr = [1,2,3,5,6,8,9,10,11,12]

print(longest\_subarray(arr))

**Reverse a list in place**

l = [1,2,3,4,6]

l.reverse()

print(l)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

l = l[::-1]

print(l)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def reverse\_list(arr):

left=0

right=len(arr)-1

while left<right:

arr[left],arr[right]=arr[right],arr[left]

left=left+1

right=right-1

return arr

arr = [1,2,3,5,6,8,9,10,11,12]

print(reverse\_list(arr))

**write a recursive function to find sum of all elements in a list**

def sum\_of\_array(arr: list) -> int :

# base case

if len(arr) == 1:

return arr[0]

return arr[0] + sum\_of\_array(arr[1:])

print(sum\_of\_array(l))

**Reverse nested list**

def reverse\_nested\_list(lst):

# Reverse the outer list first

lst.reverse()

# Now reverse each element inside the list

for i in range(len(lst)):

if isinstance(lst[i], list): # If the element is a list

lst[i] = reverse\_nested\_list(lst[i]) # Recursively reverse the inner list

return lst

# Example Usage

l = [1, 2, [3, 4], [6, [7, 8]]]

reversed\_l = reverse\_nested\_list(l)

print(reversed\_l)

**find the non repeating element in list**

**(unique)**

my\_list2 = [3, 1, 2, 3, 1, 2, 4, 5,6,7,99,999,9,9,0,9]

l = [i for i in my\_list2 if my\_list2.count(i) == 1]

print(l)

**Find first 100 prime numbers**

def is\_prime(num):

if num < 2:

return False

for i in range(2, int(num \*\* 0.5) + 1):

if num % i == 0:

return False

return True

def first\_100\_primes():

primes = []

num = 2 # Starting from the first prime number

while len(primes) < 100:

if is\_prime(num):

primes.append(num)

num += 1

return primes

# Example Usage

prime\_numbers = first\_100\_primes()

print(prime\_numbers)

**Find the longest consecutive subarray from a given array of integers**

def longest\_consecutive\_subsequence(s):

**# Initialize the result as an empty string**

result = s[-1] # Start with the last character as the initial subsequence

current\_subseq = s[-1] # To track the current consecutive subsequence

**# Iterate backward through the string starting from the second-last character**

for i in range(len(s)-2, -1, -1):

if ord(s[i]) == ord(s[i+1]) - 1**: # Check if current char is consecutive to the next**

current\_subseq = s[i] + current\_subseq **# Append to the subsequence**

else:

break **# If consecutive condition is not met, break the loop**

return current\_subseq

**# Example usage:**

input\_string = "abcdfghijk"

result = longest\_consecutive\_subsequence\_from\_end(input\_string)

print(result) # Output: "fghijk"

**Print priciest fruit in dict**

fruit\_prices = {

"apple": 1.5,

"banana": 0.5,

"cherry": 2.0,

"date": 3.0,

"elderberry": 4.5

}

# Find the fruit with the maximum price

priciest\_fruit = max(fruit\_prices, key=fruit\_prices.get)

print(priciest\_fruit)

**Print second largest odd number**

def second\_largest\_odd(lst):

odd\_numbers = [num for num in lst if num % 2 != 0]

if len(odd\_numbers) < 2:

return None

odd\_numbers.sort(reverse=True)

return odd\_numbers[1]

# Example usage

lst = [1, 2, 3, 4, 5, 7, 9, 11]

print(second\_largest\_odd(lst)) # Output: 9

**lambda function to return random number between 1 and 100**

import random

a = lambda : random.randint(1,100)

**Count of palindromes**

def count\_palindromes(text: str) -> int:

unique\_palindromes = set()

for i in range(len(text)):

for j in range(i + 1, len(text) + 1):

substring = text[i:j]

if substring == substring[::-1]:

unique\_palindromes.add(substring)

return len(unique\_palindromes)

string = "aabaa"

print(count\_palindromes(string)) # Output: 6 (unique palindromes: a, b, aa, aba, aabaa)

**Reverse String**

s = "hello world"

reversed\_s = ""

for char in s:

reversed\_s = char + reversed\_s

print("Reversed String:", reversed\_s)

**def reverse\_string(s):**

**if len(s) == 0:**

**return s**

**return s[-1] + reverse\_string(s[:-1])**

**s = "hello world"**

**reversed\_s = reverse\_string(s)**

**print("Reversed String:", reversed\_s)**

**create a string with multiple words with spaces, all in lower case. convert the first and last char to uppercase**

s = 'my name is rahul Rajeev'

words=s.split()

result=[]

for word in words:

if len(word)>1:

modified=word[0].upper()+word[1:-1]+word[-1].upper()

else:

modified=word[0].upper()

result.append(modified)

result\_str=" ".join(result)

print(result\_str)

**Count of small and upper count**

s = "RahuLRAJeev"

lower\_count = 0

upper\_count = 0

for char in s:

if char.islower():

lower\_count += 1 # Increment count for lowercase letters

elif char.isupper():

upper\_count += 1 # Increment count for uppercase letters

print("Lowercase letters:", lower\_count)

print("Uppercase letters:", upper\_count)

**You are give a string with items separated by commas convert them into elements in list**

s = "apple, banana, 24, cherry,,fig"

l = s.split(',')

print(l)

**Find the date 45 days before today**

from datetime import datetime, timedelta

today = datetime.today()

print(today)

days\_before = today - timedelta(days=45)

print(days\_before)

# formated date

print("Today's date:", today.strftime("%Y-%m-%d"))

print("Date 45 days before today:", days\_before.strftime("%Y-%m-%d"))

**Replace the char in string without using built-in replace function**

def replace\_char(s,old\_char,new\_char):

result=""

for char in s:

if char==old\_char:

result=result+new\_char

else:

result=result+char

return result

print(replace\_char("hi","h","i"))

**Find Common Keys**

# Two example dictionaries

dict1 = {"a": 1, "b": 2, "c": 3}

dict2 = {"b": 4, "c": 5, "d": 6}

# Find common keys

common\_keys = dict1.keys() & dict2.keys()

print(common\_keys) # Output: {'b', 'c'}

**Combine 3 tuples**

tuple1 = (1, 2, 3)

tuple2 = (4, 5, 6)

tuple3 = (7, 8, 9)

combined\_tuple = tuple1 + tuple2 + tuple3

print(combined\_tuple)

# Output: (1, 2, 3, 4, 5, 6, 7, 8, 9)

**Generate a Random Integer**

import random

# Lambda to generate a random integer between 1 and 100

random\_int = lambda: random.randint(1, 100)

print(random\_int()) # Output: Random integer, e.g., 42

**Capitalize first occurrence of character in string**

def capitalize\_first\_occurrences(sentence):

seen = set() # Set to track the characters we have already seen (case insensitive)

result = [] # List to store the final result

for char in sentence:

if char.isalpha(): # Process only alphabetic characters

if char.lower() not in seen:

# First occurrence: capitalize it and add to seen

result.append(char.upper())

seen.add(char.lower())

else:

# Subsequent occurrence: make it lowercase

result.append(char.lower())

else:

# Non-alphabetic characters are added as-is

result.append(char)

return ''.join(result)

# Example usage

sentence = "MMmYyY"

output = capitalize\_first\_occurrences(sentence)

print(output) # Output: "MmmYyy"