1. def find\_the\_difference(in\_one,in\_two):

in\_two\_temp = list(in\_two)

for ele in in\_one:

if ele in in\_two\_temp:

in\_two\_temp.pop(in\_two\_temp.index(ele))

output = ','.join(in\_two\_temp)

print(f'find\_the\_difference{in\_one,in\_two} ➞ {output}')

find\_the\_difference("abcd", "abcde")

find\_the\_difference("", "y")

find\_the\_difference("ae", "aea")

Output:

find\_the\_difference('abcd', 'abcde') ➞ e

find\_the\_difference('', 'y') ➞ y

find\_the\_difference('ae', 'aea') ➞ a

1. from collections import OrderedDict

def count\_datatypes(\*args):

output = OrderedDict({'int':0,'str':0,'bool':0,'list':0,'tuple':0,'dict':0})

for ele in args:

output[type(ele).\_\_name\_\_] += 1

print(f'count\_datatypes{args} ➞ {list(output.values())}')

count\_datatypes(1, 45, "Hi", False)

count\_datatypes([10, 20], ("t", "Ok"), 2, 3, 1)

count\_datatypes("Hello", "Bye", True, True, False, {"1": "One", "2": "Two"}, [1, 3], {"Brayan": 18}, 25, 23)

count\_datatypes(4, 21, ("ES", "EN"), ("a", "b"), False, [1, 2, 3], [4, 5, 6])

Output:

count\_datatypes(1, 45, 'Hi', False) ➞ [2, 1, 1, 0, 0, 0]

count\_datatypes([10, 20], ('t', 'Ok'), 2, 3, 1) ➞ [3, 0, 0, 1, 1, 0]

count\_datatypes('Hello', 'Bye', True, True, False, {'1': 'One', '2': 'Two'}, [1, 3], {'Brayan': 18}, 25, 23) ➞ [2, 2, 3, 1, 0, 2]

count\_datatypes(4, 21, ('ES', 'EN'), ('a', 'b'), False, [1, 2, 3], [4, 5, 6]) ➞ [2, 0, 1, 2, 2, 0]

1. # Approach 1 Using Lists

def fib\_str\_one(in\_num,in\_list):

out\_list = in\_list.copy()

if in\_num > 2:

for ele in range(2,in\_num):

out\_list.append(out\_list[-1]+out\_list[-2])

print(f'fib\_str\_one{in\_num, in\_list} ➞ {", ".join(out\_list)}')

# Approach 2 Without Using Lists

def fib\_str\_two(in\_num,in\_list):

back\_one, back\_two = in\_list[-1],in\_list[-2]

output=", ".join(in\_list)

if in\_num > 2:

for ele in range(2,in\_num):

temp = back\_one+back\_two

output += ", "+temp

back\_one = back\_two

back\_two = temp

print(f'fib\_str\_two{in\_num, in\_list} ➞ {output}')

fib\_str\_one(3, ["j", "h"])

fib\_str\_one(5, ["e", "a"])

fib\_str\_one(6, ["n", "k"])

print()

fib\_str\_two(3, ["j", "h"])

fib\_str\_two(5, ["e", "a"])

fib\_str\_two(6, ["n", "k"])

Output:

fib\_str\_one(3, ['j', 'h']) ➞ j, h, hj

fib\_str\_one(5, ['e', 'a']) ➞ e, a, ae, aea, aeaae

fib\_str\_one(6, ['n', 'k']) ➞ n, k, kn, knk, knkkn, knkknknk

fib\_str\_two(3, ['j', 'h']) ➞ j, h, hj

fib\_str\_two(5, ['e', 'a']) ➞ e, a, ae, eae, aeeae

fib\_str\_two(6, ['n', 'k']) ➞ n, k, kn, nkn, knnkn, nknknnkn

1. def ones\_threes\_nines(in\_num):

in\_num\_clone = in\_num

answer = {'nines':0,'threes':0,'ones':0}

if in\_num > 0 and in\_num < 26:

while in\_num != 0:

if in\_num >= 9:

in\_num -= 9

answer['nines'] += 1

elif in\_num >= 3:

in\_num -= 3

answer['threes'] += 1

elif in\_num >= 1:

in\_num -=1

answer['ones'] +=1

print(f"ones\_threes\_nines({in\_num\_clone}) ➞ \"nines:{answer['nines']}, threes:{answer['threes']}, ones:{answer['ones']}\"")

ones\_threes\_nines(10)

ones\_threes\_nines(15)

ones\_threes\_nines(22)

Output:

ones\_threes\_nines(10) ➞ "nines:1, threes:0, ones:1"

ones\_threes\_nines(15) ➞ "nines:1, threes:2, ones:0"

ones\_threes\_nines(22) ➞ "nines:2, threes:1, ones:1"

1. # Approach 1

def fib\_one(in\_num):

if in\_num < 1:

print("Minimum two digits are required to generate a fibonacci sequence")

else:

output = [0,1]

for ele in range(2,in\_num+1):

output.append(output[-1]+output[-2])

print(f'fib\_one({in\_num}) ➞ {output[-1]}')

# Approach 2

def fib\_two(in\_num):

back\_one,back\_two = 1,0

for ele in range(2,in\_num+1):

temp = back\_one+back\_two

back\_two = back\_one

back\_one = temp

print(f'fib\_two({in\_num}) ➞ {back\_one}')

fib\_one(6)

fib\_one(1)

fib\_one(2)

print()

fib\_two(6)

fib\_two(1)

fib\_two(2)

Output:

fib\_one(6) ➞ 8

fib\_one(1) ➞ 1

fib\_one(2) ➞ 1

fib\_two(6) ➞ 8

fib\_two(1) ➞ 1

fib\_two(2) ➞ 1