1 –

A -

Mistery(3)

First loop:

First print -> “ \*” – 2 \* ‘ ‘ + 1 \* ‘ \*’

Second print -> “ \* \*” – 1 \* ‘ ‘ + 2 \* ‘ \*’

Third print -> “ \* \* \*” – 0 \* ‘ ‘ + 3 \* ‘ \*’

Second loop:

Forth print -> “ \* \*” – 1 \* ‘ ‘ + 2 \* ‘ \*’

Fifth loop -> “ \* - 2 \* ‘ ‘ + 1 \* ‘ \*’

Sixth loop -> “ “ – 3 \* ‘ ‘ + 0 \* ‘ \*’

Output:

\*

\* \*

\* \* \*

\* \*

\*

B –

m.v = 12

m.k = [23]

m.both = {11:[23]}

C –

Return [[x for x in range(z, 0, y)] for z, y in zip(range(100, -1, -10), range(-20, -1, 2))] + [[0] \*5]

- > [[x for x in range(100, -1, -20)] + [x for x in range(90, -1, -18)] + [x for x in range(80, -1, -16)] + [x for x in range(70, -1, -14)] + [x for x in range(60, -1, -12)] + [x for x in range(50, -1, -10)] + [x for x in range(40, -1, -8)] + [x for x in range(30, -1, -6)] + [x for x in range(20, -1, -4)] + [x for x in range(10, -1, -2)] + [[0] \* 5]

3 –

A-

Def sum\_of\_two(arr1,arr2,value):

For i in range(len(arr1)):

For j in range(len(arr2)):

If arr1[i] + arr2[j] == value:

Return True

return false

B –

Def sum\_of\_two(arr1,arr2,value):

Dic = {}

For v in arr1:

Dic[abs(value-v)]=i

For v in arr2:

If dic.get(v):

Return True

Return False

C –

Def sum\_of\_two(arr1,arr2,value):

For i in range(len(arr1)):

For j in range(len(arr2)):

If arr1[i] + arr2[j] == value:

Yield (arr1[i], arr2[j])

yield None

3 –

def remove\_all\_adjacent\_duplicates(text):

stack = myStack()

for ch in text:

if stack.isEmpty():

stack.push(ch)

elif stack.look() == ch:

stack.pop()

else:

stack.push(ch)

#from here, we already have a Stack without the adjacent duplicates

#from down here we return a new string but now fixed

Return stack.to\_string() #correct answer using the function from the Stack class

Return stack

#or you can return the Stack and call the to\_string also

#my solution (it works but it’s the wrong one since the class Stack had a function for it)

fixed\_text = “”

while stack.len > 0:

e = stack.pop()

fixed\_test += e

return fixed\_text[::-1]

4-

A-

class Person:

def \_\_init\_\_(self, name):

self.name = name.lower().capitalize()

self.right = None

self.left = None

def \_\_eq\_\_(self, person):

if self.name == other.name:

return True

else:

return self.\_\_ne\_\_()

def \_\_ne\_\_(self, person):

if self.name != person.name:

return True

else:

return self.\_\_eq\_\_(person)

def \_\_add\_\_(self,person):

self.right = person

person.left = self

return person

def \_\_str\_\_(self):

return '[{}]'.format(self.name)

def display(self):

return '{}<--{}-->{}'.format(self.left,self,self.right)

def \_\_next\_\_(self):

while self.right != None:

return self.right

B-

class DanceLine:

def \_\_init\_\_(self,root):

self.root = root

self.len = self.init\_root()

self.tail = self.get\_tail()

def init\_root(self):

self.len = 0

itr = self.root

while itr:

self.len += 1

itr = itr.right

def get\_tail(self):

cur = self.root

while cur.right != None:

cur = cur.right

return cur

def \_\_add\_\_(self,dance\_line):

self.tail + dance\_line.root #Person object + Person object (add at Person class)

self.tail = dance\_line.tail #Update the tail

return self

def \_\_iter\_\_(self):

return self.root

def \_\_str\_\_(self):

s = ''

itr = iter(self)

while itr:

s += '{} --> '.format(itr)

itr = next(itr)

s += '||'

return s