**CODE**:-

# Huffman Coding in python

string = 'BCAADDDCCACACAC'

# Creating tree nodes

class NodeTree(object):

def \_\_init\_\_(self, left=None, right=None):

self.left = left

self.right = right

def children(self):

return (self.left, self.right)

def nodes(self):

return (self.left, self.right)

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

return '%s\_%s' % (self.left, self.right)

# Main function implementing huffman coding

def huffman\_code\_tree(node, left=True, binString=''):

if type(node) is str:

return {node: binString}

(l, r) = node.children()

d = dict()

d.update(huffman\_code\_tree(l, True, binString + '0'))

d.update(huffman\_code\_tree(r, False, binString + '1'))

return d

# Calculating frequency

freq = {}

for c in string:

if c in freq:

freq[c] += 1

else:

freq[c] = 1

freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

nodes = freq

while len(nodes) > 1:

(key1, c1) = nodes[-1]

(key2, c2) = nodes[-2]

nodes = nodes[:-2]

node = NodeTree(key1, key2)

nodes.append((node, c1 + c2))

nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman\_code\_tree(nodes[0][0])

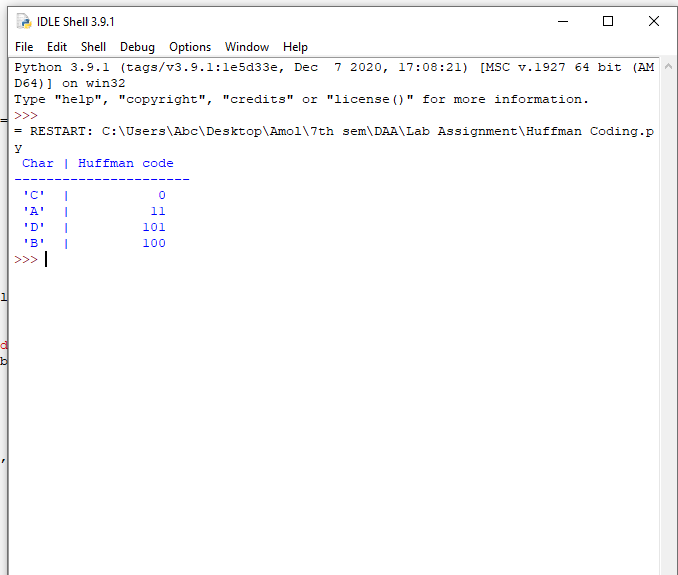
print(' Char | Huffman code ')

print('----------------------')

for (char, frequency) in freq:

print(' %-4r |%12s' % (char, huffmanCode[char]))

**OUTPUT:-**

****