

Assignment-'1'

Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyse their time and space complexity.

fibonacci.py

```
a = 0
b = 1
n=int(input("Enter the number of terms in the sequence: "))
print(a,b,end=" ")
while(n-2):
    c=a+b
    a,b = b,c
    print(c,end=" ")
    n=n-1
```

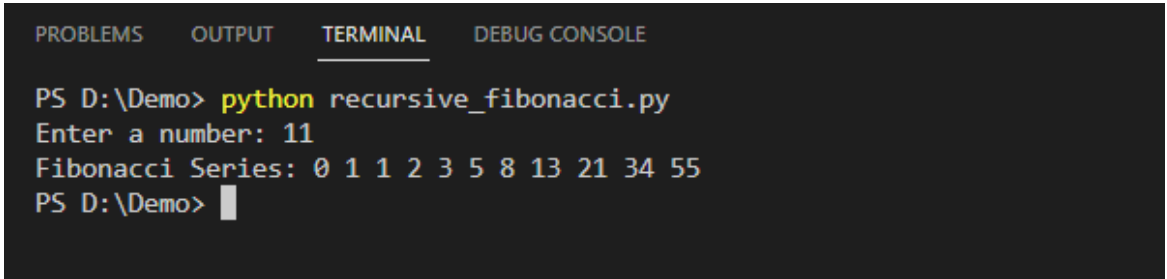
Output:

```
PS D:\Demo> & "C:/Program Files/Python39/python.exe" d:/Demo/fibonacci.py
Enter the number of terms in the sequence: 10
0 1 1 2 3 5 8 13 21 34
PS D:\Demo> █
```

recursive_fibonacci.py

```
def recursive_fibonacci(n):  
    if n <= 1:  
        return n  
    return recursive_fibonacci(n-1) + recursive_fibonacci(n-2)  
  
if __name__ == "__main__":  
    n = int(input("Enter a number: "))  
    if n <= 0:  
        print("Please enter a Positive Number")  
    else:  
        print("Fibonacci Series:", end=" ")  
        for i in range(n):  
            print(recursive_fibonacci(i), end=" ")
```

Output:



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  
PS D:\Demo> python recursive_fibonacci.py  
Enter a number: 11  
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34 55  
PS D:\Demo> 
```

Assignment-'2'

Write a program to implement Huffman Encoding using a greedy strategy.

Hoffman.py

```
string = 'BCAADDDCCACACAC'
```

```
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 (self.left, "_", self.right)
```

```
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
```

```
freq = {}
```

```
for c in string:
```

```
    if c in freq:
```

```

    freq[c] += 1
else:
    freq[c] = 1

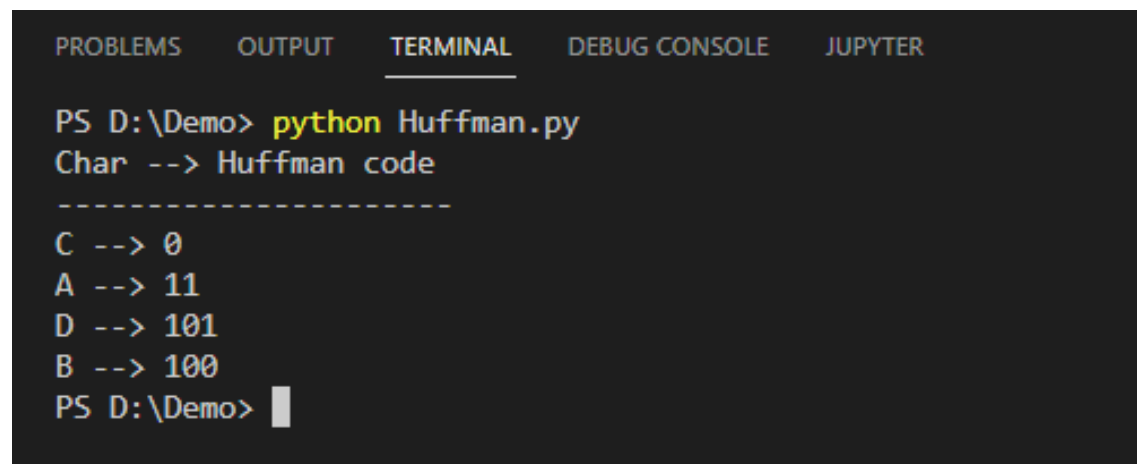
freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

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(char, "-->", huffmanCode[char])

```

Output:



```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  JUPYTER

PS D:\Demo> python Huffman.py
Char --> Huffman code
-----
C --> 
A --> 11
D --> 101
B --> 100
PS D:\Demo> 

```

Assignment-'3'

Write a program to solve a fractional Knapsack problem using a greedy method.

fractional knapsack.py

```
class Item:
```

```
    def __init__(self,value,weight):
```

```
        self.value = value
```

```
        self.weight = weight
```

```
def knapsack(w,arr):
```

```
    arr.sort(key=lambda x: (x.value/x.weight), reverse=True)
```

```
    '''for item in arr:
```

```
        print(item.value, item.weight, item.value/item.weight)'''
```

```
    finalvalue = 0.0
```

```
    for item in arr:
```

```
        if item.weight <= w:
```

```
            w -= item.weight
```

```
            finalvalue += item.value
```

```
        else:
```

```
            finalvalue += item.value * w/item.weight
```

```
            break
```

```
    return finalvalue
```

```
if __name__ == "__main__":
```

```
    # Weight of Knapsack
```

```
    knapsack_weight = 50
```

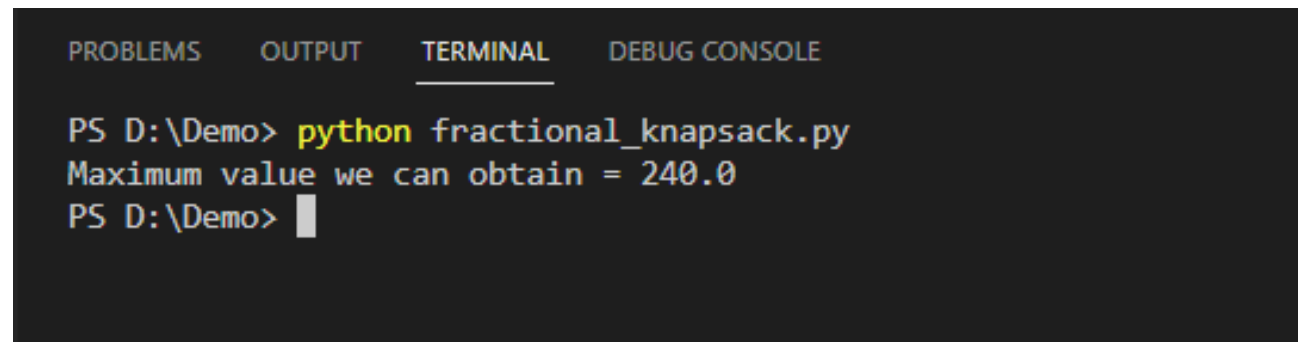
```
    arr = [Item(60, 10), Item(100, 20), Item(120, 30)]
```

Function call

```
max_val = knapsack(knapsack_weight, arr)
```

```
print ('Maximum value we can obtain = {}'.format(max_val))
```

Output:



The screenshot shows a terminal window with a dark background. At the top, there are four tabs: 'PROBLEMS', 'OUTPUT', 'TERMINAL' (which is selected and underlined), and 'DEBUG CONSOLE'. Below the tabs, the terminal shows the following text: 'PS D:\Demo> python fractional_knapsack.py', followed by the output 'Maximum value we can obtain = 240.0', and then 'PS D:\Demo>' with a cursor.

```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

PS D:\Demo> python fractional_knapsack.py
Maximum value we can obtain = 240.0
PS D:\Demo> █
```