Assignment 4 - Print Pretty

1) How to execute the code?

```
python print_neatly_test.py
```

Note: The text files should be in the same folder as the python file.

2) Reflection

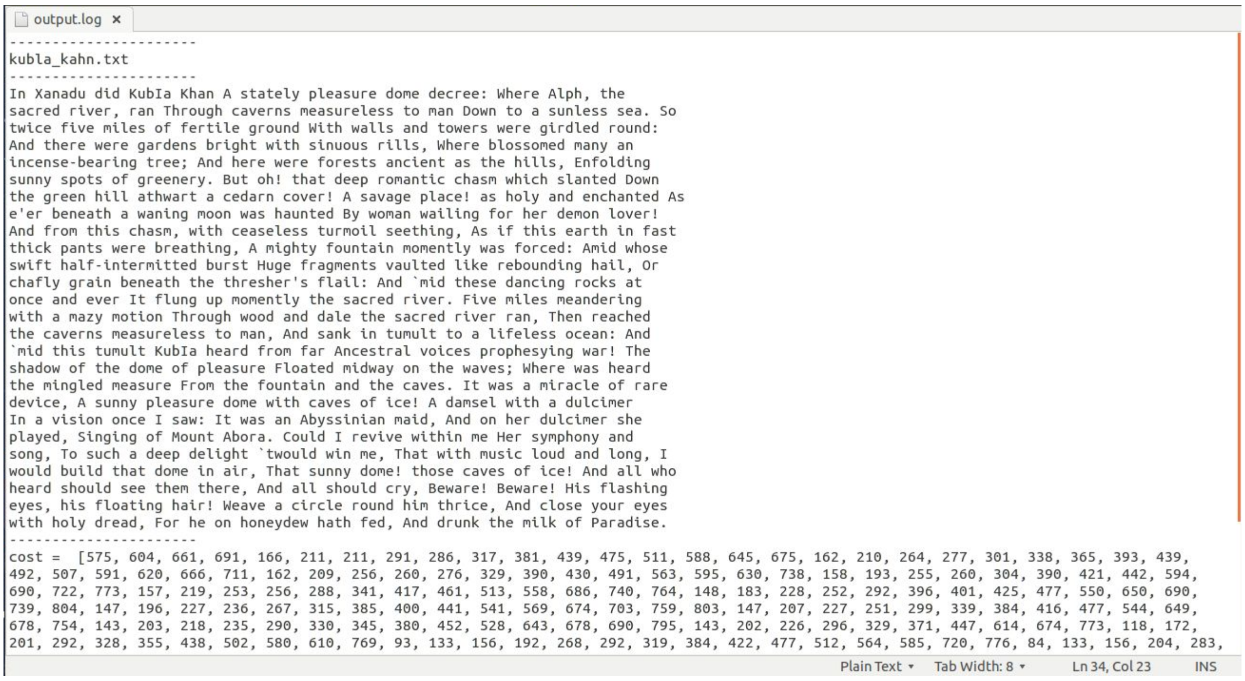
In this program the aim is to format the text using dynamic programming so that the final output is pretty to see.

The input given is a text file containing words and the no of characters a line can hold. We have to arrange the words in such a way that the no of empty characters in the end of line is minimum and also to arrange the words so that each line has similar no of empty characters in the end.

We do this using dynamic programming. We first calculate the word length and hold it in an array. And then build the cost matrix which holds the square of empty spaces in the end of line. This is calculated by subtracting the length of all the words in a line + the empty spaces between words by the maximum characters a line can hold and then squaring it.

We then build a cost array which is the cost of including a particular word in a line. By using cost array and cost matrix we find the minimum cost. We hold the index of that word in path array which tells which all words go in a particular line.

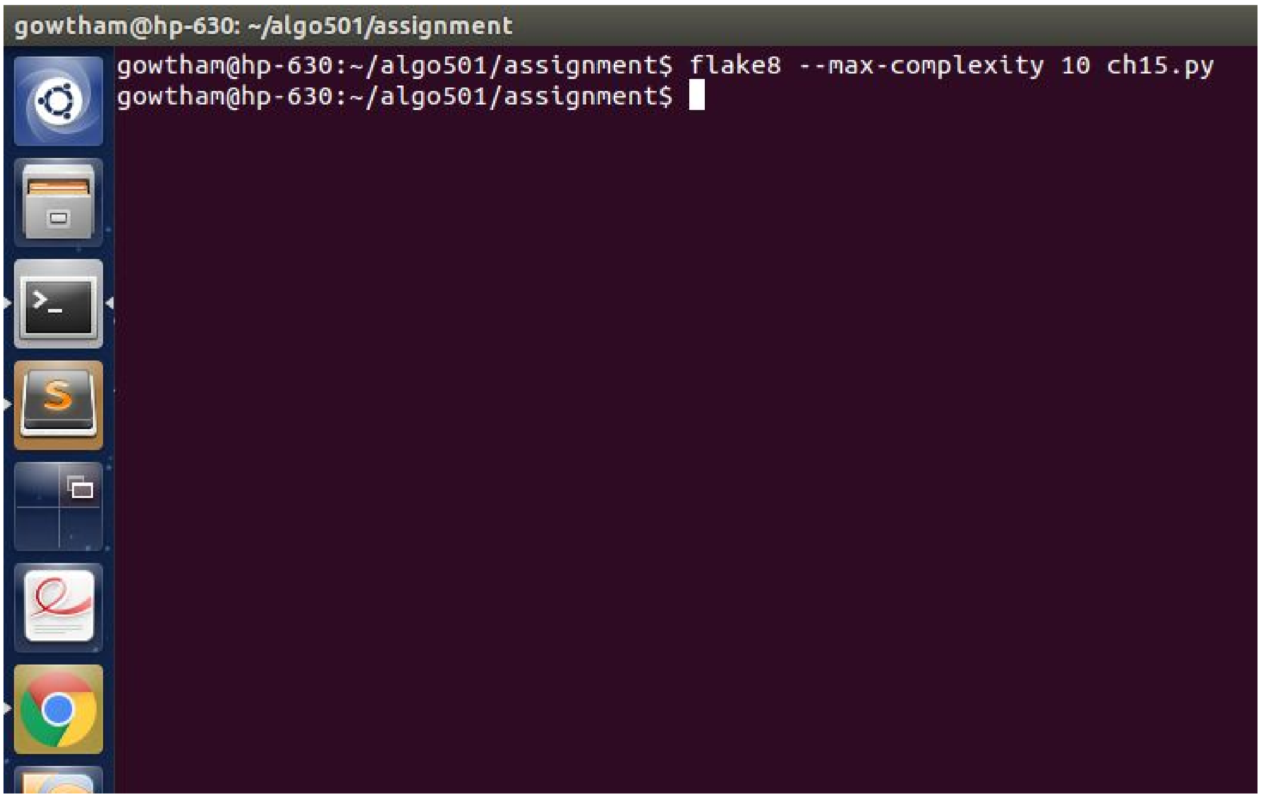
3) Testing Output



```
----------------------
kubla_kahn.txt
----------------------
In Xanadu did KubIa Khan A stately pleasure dome decree: Where Alph, the
sacred river, ran Through caverns measureless to man Down to a sunless sea. So
twice five miles of fertile ground With walls and towers were girdled round:
And there were gardens bright with sinuous rills, Where blossomed many an
incense-bearing tree; And here were forests ancient as the hills, Enfolding
sunny spots of greenery. But oh! that deep romantic chasm which slanted Down
the green hill athwart a cedarn cover! A savage place! as holy and enchanted As
e'er beneath a waning moon was haunted By woman wailing for her demon lover!
And from this chasm, with ceaseless turmoil seething, As if this earth in fast
thick pants were breathing, A mighty fountain momently was forced: Amid whose
swift half-intermitted burst Huge fragments vaulted like rebounding hail, Or
chafly grain beneath the thresher's flail: And `mid these dancing rocks at
once and ever It flung up momently the sacred river. Five miles meandering
with a mazy motion Through wood and dale the sacred river ran, Then reached
the caverns measureless to man, And sank in tumult to a lifeless ocean: And
`mid this tumult KubIa heard from far Ancestral voices prophesying war! The
shadow of the dome of pleasure Floated midway on the waves; Where was heard
the mingled measure From the fountain and the caves. It was a miracle of rare
device, A sunny pleasure dome with caves of ice! A damsel with a dulcimer
In a vision once I saw: It was an Abyssinian maid, And on her dulcimer she
played, Singing of Mount Abora. Could I revive within me Her symphony and
song, To such a deep delight `twould win me, That with music loud and long, I
would build that dome in air, That sunny dome! those caves of ice! And all who
heard should see them there, And all should cry, Beware! Beware! His flashing
eyes, his floating hair! Weave a circle round him thrice, And close your eyes
with holy dread, For he on honeydew hath fed, And drunk the milk of Paradise.
----------------------
cost =  [575, 604, 661, 691, 166, 211, 211, 291, 286, 317, 381, 439, 475, 511, 588, 645, 675, 162, 210, 264, 277, 301, 338, 365, 393, 439,
492, 507, 591, 620, 666, 711, 162, 209, 256, 260, 276, 329, 390, 430, 491, 563, 595, 630, 738, 158, 193, 255, 260, 304, 390, 421, 442, 594,
690, 722, 773, 157, 219, 253, 256, 288, 341, 417, 461, 513, 558, 686, 740, 764, 148, 183, 228, 252, 292, 396, 401, 425, 477, 550, 650, 690,
739, 804, 147, 196, 227, 236, 267, 315, 385, 400, 441, 541, 569, 674, 703, 759, 803, 147, 207, 227, 251, 299, 339, 384, 416, 477, 544, 649,
678, 754, 143, 203, 218, 235, 290, 330, 345, 380, 452, 528, 643, 678, 690, 795, 143, 202, 226, 296, 329, 371, 447, 614, 674, 773, 118, 172,
201, 292, 328, 355, 438, 502, 580, 610, 769, 93, 133, 156, 192, 268, 292, 319, 384, 422, 477, 512, 564, 585, 720, 776, 84, 133, 156, 204, 283,
```

4) Static Analysis / Compilation Output

Below is the output from flake8



```
gowtham@hp-630: ~/algo501/assignment
gowtham@hp-630:~/algo501/assignment$ flake8 --max-complexity 10 ch15.py
gowtham@hp-630:~/algo501/assignment$
```

5) Source Code

```python
import math
INFINITY = float('inf')

cost_matrix = []
cost_array = []
word_length = []
path_array = []
maxWidth = 0
noofwords = 0
words_array = []


def calculate_word_length(words):
    for word in words:
        word_length.append(len(word))


def print_neatly(words, M):
    global noofwords
    global maxWidth
    global cost_matrix
    global cost_array
    global word_length
    global path_array
    global words_array

    noofwords = len(words)
    maxWidth = M
    cost_matrix = [[0 for j in range(noofwords)]for i in range(noofwords)]
    cost_array = [0 for j in range(noofwords)]
    word_length = []
    path_array = [0 for j in range(noofwords)]
    words_array = words

    calculate_word_length(words)
    build_cost_matrix()

    i = j = noofwords-1

    for i in reversed(range(noofwords)):
        j = noofwords - 1
        min_cost = cost_matrix[i][j]
        min_index = j
        if cost_matrix[i][j] is not INFINITY:
            cost_array[i] = cost_matrix[i][j]
            path_array[i] = j + 1
        else:
            while j > i:
                if cost_matrix[i][j-1] is not INFINITY:
                    temp_cost = cost_array[j] + cost_matrix[i][j-1]

                    if min_cost > temp_cost:
                        min_cost = temp_cost
                        min_index = j
                    j -= 1
                else:
                    j -= 1

            cost_array[i] = min_cost
            path_array[i] = min_index
    string_text = string_builder()
    return cost_array, string_text


def build_cost_matrix():
    for i in range(noofwords):
        for j in range(i, noofwords):
            if i != j:
                if cost_matrix[i][j-1] != INFINITY:
                    sub_total_length = maxWidth - int(math.sqrt(cost_matrix[i][j-1]))
                    total_length = word_length[j] + sub_total_length + 1
                    empty_space = maxWidth - total_length
                else:
                    empty_space = -1
            else:
                total_length = word_length[j]
                empty_space = maxWidth - total_length

            if empty_space < 0:
                cost_matrix[i][j] = INFINITY
            else:
                cost_matrix[i][j] = empty_space * empty_space


def string_builder():
    stringer = ''
    i = 0
    while i < noofwords:
        end_index = path_array[i]
        for j in range(i, end_index):
            stringer += words_array[j]
            if j < end_index-1:
                stringer += ' '
        i = path_array[i]
        if i < noofwords:
            stringer += '\n'
    return stringer
```