Coding Assignment 3: Seam Carving Write-up

1. Reflection

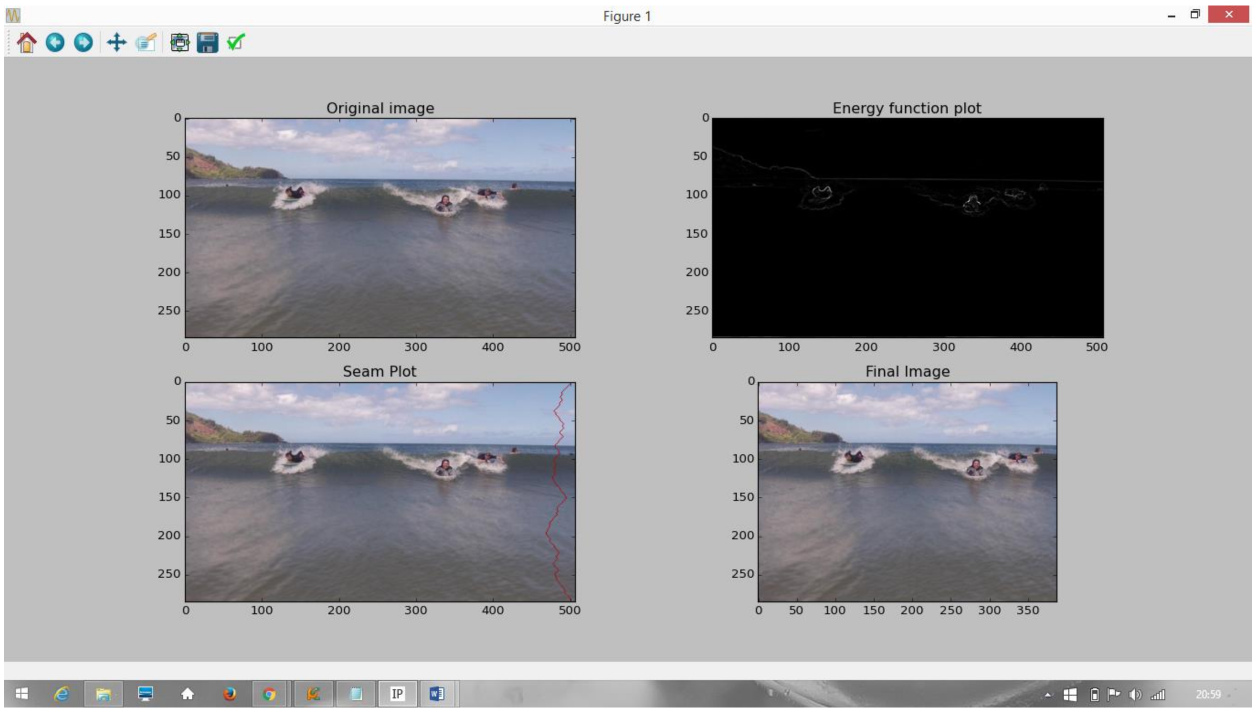
Seam carving assignment facilitated me to relate and implement the dynamic programming concepts to the real word scenarios, where I developed a data-type that manipulates the aspect ratio of the image with the help of the several sub-modules. It helped me to understand with a practical hands-on experience that for certain scenarios in which sub-problems has repeated sub-sub-problem can be solved with a more efficient approach of dynamic programming than divide and conquer. The trade-off is between time and memory, where memorization complements recursion by storing the cost of the paths and path indices for backtracking. On implementation grounds, assignment 3 also gives me exposure to the set of tools available in python like scipy and skimage, which has excellent capability to handle large amount of data, here the data was pixel information of the input image.

I implemented seam carving by finding the energy of the image pixel with the help of dual energy gradient function and then calculating the minimum cost path, by taking pixels as the nodes and energy of pixels as the weight of taking corresponding path. Once the path costs were calculated and saved in a matrix, a path with minimum cost was selected. Finally, the pixels that are in the path of identified minimum cost path called as seam was removed from the image to create a new image. The above process was repeated a number of times to achieved the desired image aspect ratio.

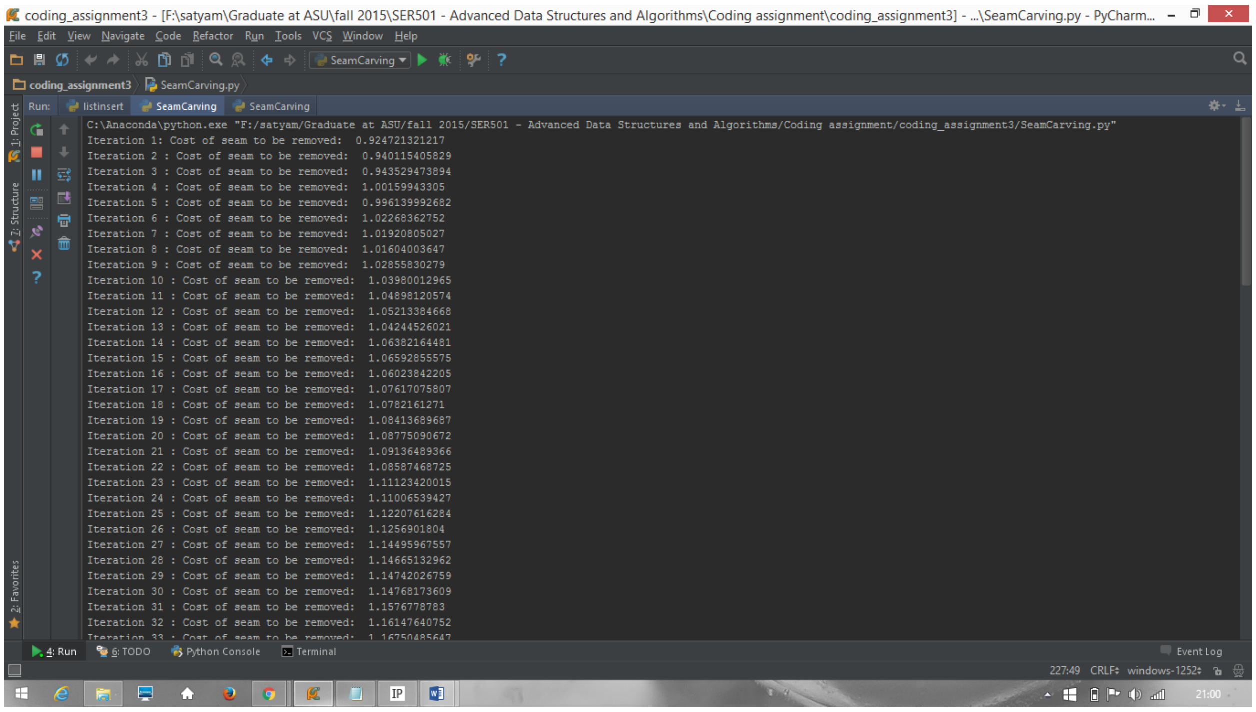
Hence, Seam Carving assignment was a great learning experience.

2. Testing Output

a. The below image shows the figure tray of scikit-image showing:
   The input image,
   Energy of each pixel,
   Seam plot of first seam found,
   Final reduced image after a series of seam removal

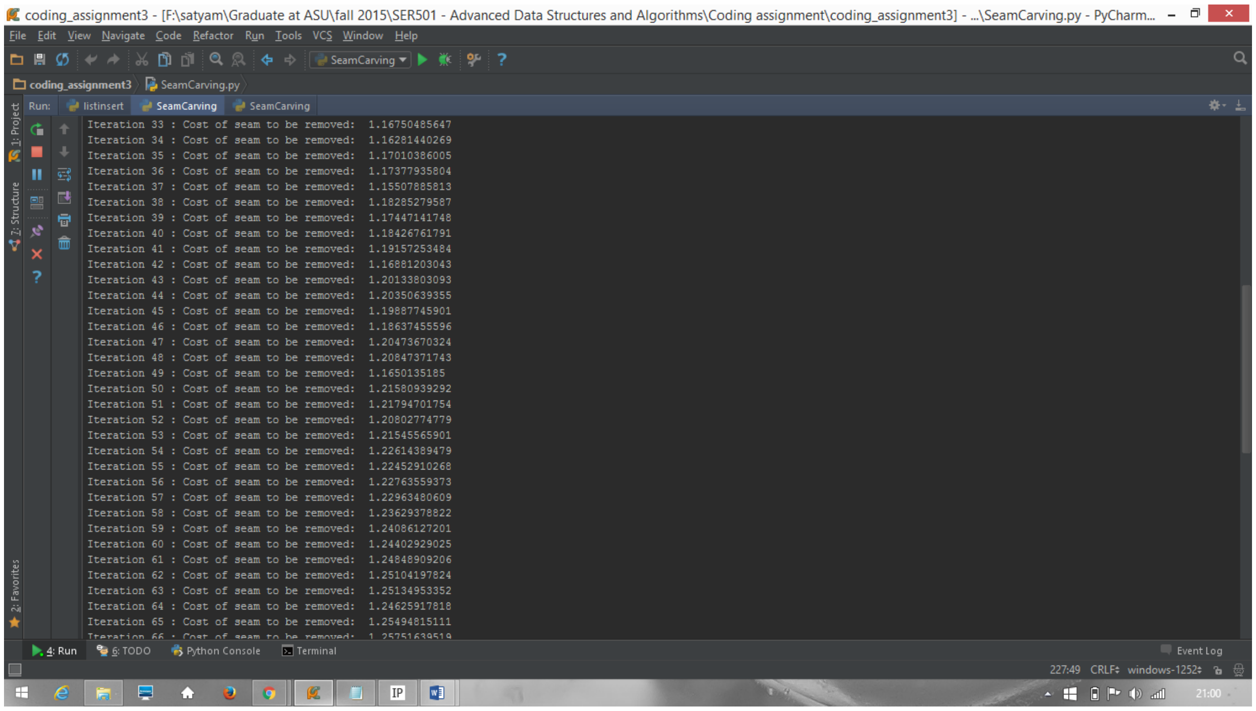


b. Below image shows the path cost of the seam selected for removal from iteration 1 to 33



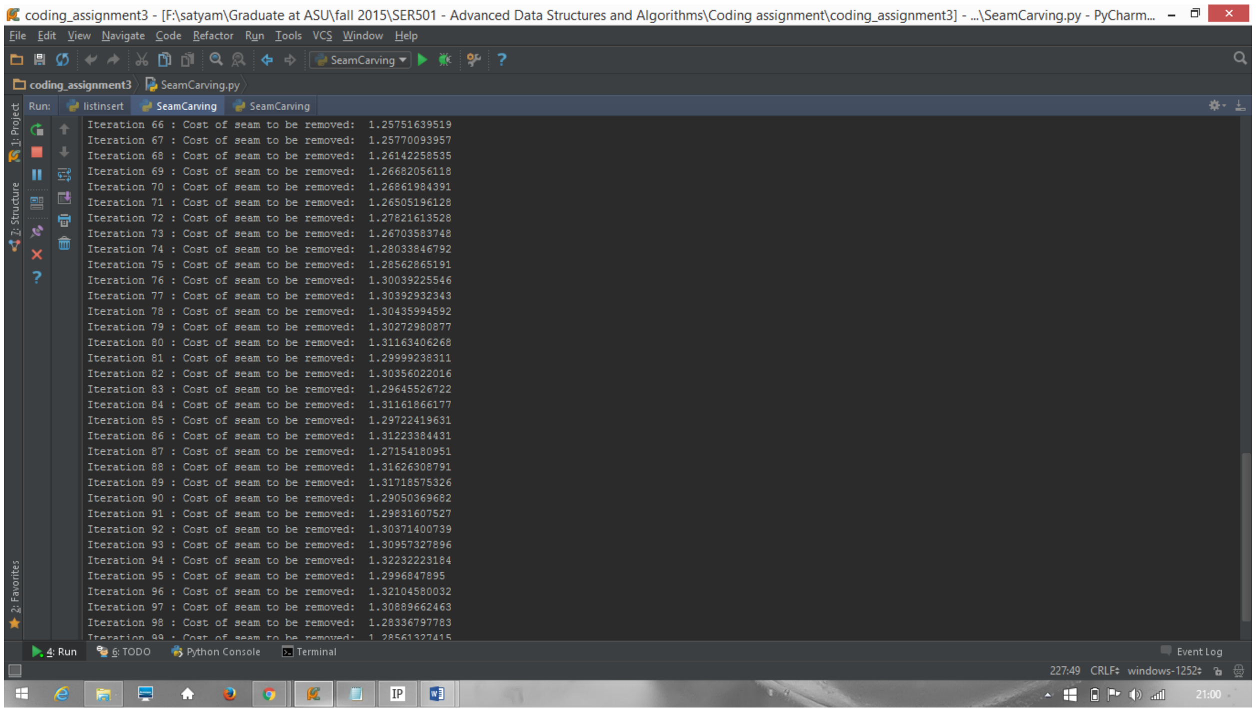
```
C:\Anaconda\python.exe "F:/satyam/Graduate at ASU/fall 2015/SER501 - Advanced Data Structures and Algorithms/Coding assignment/coding_assignment3/SeamCarving.py"
Iteration 1: Cost of seam to be removed:  0.924721321217
Iteration 2 : Cost of seam to be removed:  0.940115405829
Iteration 3 : Cost of seam to be removed:  0.943529473894
Iteration 4 : Cost of seam to be removed:  1.00159943305
Iteration 5 : Cost of seam to be removed:  0.996139992682
Iteration 6 : Cost of seam to be removed:  1.02268362752
Iteration 7 : Cost of seam to be removed:  1.01920805027
Iteration 8 : Cost of seam to be removed:  1.01604003647
Iteration 9 : Cost of seam to be removed:  1.02855830279
Iteration 10 : Cost of seam to be removed:  1.03980012965
Iteration 11 : Cost of seam to be removed:  1.04898120574
Iteration 12 : Cost of seam to be removed:  1.05213384668
Iteration 13 : Cost of seam to be removed:  1.04244526021
Iteration 14 : Cost of seam to be removed:  1.06382164481
Iteration 15 : Cost of seam to be removed:  1.06592855575
Iteration 16 : Cost of seam to be removed:  1.06023842205
Iteration 17 : Cost of seam to be removed:  1.07617075807
Iteration 18 : Cost of seam to be removed:  1.0782161271
Iteration 19 : Cost of seam to be removed:  1.08413689687
Iteration 20 : Cost of seam to be removed:  1.08775090672
Iteration 21 : Cost of seam to be removed:  1.09136489366
Iteration 22 : Cost of seam to be removed:  1.08587468725
Iteration 23 : Cost of seam to be removed:  1.11123420015
Iteration 24 : Cost of seam to be removed:  1.11006539427
Iteration 25 : Cost of seam to be removed:  1.12207616284
Iteration 26 : Cost of seam to be removed:  1.1256901804
Iteration 27 : Cost of seam to be removed:  1.14495967557
Iteration 28 : Cost of seam to be removed:  1.14665132962
Iteration 29 : Cost of seam to be removed:  1.14742026759
Iteration 30 : Cost of seam to be removed:  1.14768173609
Iteration 31 : Cost of seam to be removed:  1.1576778783
Iteration 32 : Cost of seam to be removed:  1.16147640752
Iteration 33 : Cost of seam to be removed:  1.16750485647
```

c. Below image shows the path cost of the seam selected for removal from iteration 33 to 65



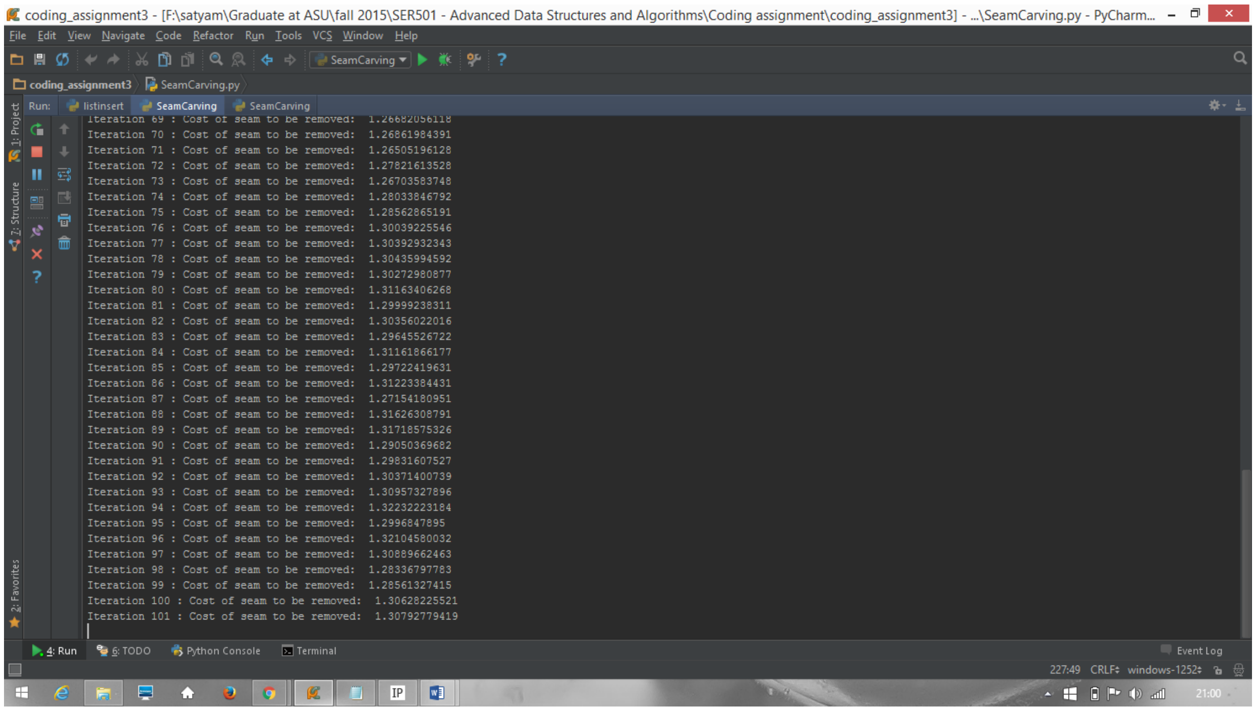
```
Iteration 33 : Cost of seam to be removed:  1.16750485647
Iteration 34 : Cost of seam to be removed:  1.16281440269
Iteration 35 : Cost of seam to be removed:  1.17010386005
Iteration 36 : Cost of seam to be removed:  1.17377935804
Iteration 37 : Cost of seam to be removed:  1.15507885813
Iteration 38 : Cost of seam to be removed:  1.18285279587
Iteration 39 : Cost of seam to be removed:  1.17447141748
Iteration 40 : Cost of seam to be removed:  1.18426761791
Iteration 41 : Cost of seam to be removed:  1.19157253484
Iteration 42 : Cost of seam to be removed:  1.16881203043
Iteration 43 : Cost of seam to be removed:  1.20133803093
Iteration 44 : Cost of seam to be removed:  1.20350639355
Iteration 45 : Cost of seam to be removed:  1.19887745901
Iteration 46 : Cost of seam to be removed:  1.18637455596
Iteration 47 : Cost of seam to be removed:  1.20473670324
Iteration 48 : Cost of seam to be removed:  1.20847371743
Iteration 49 : Cost of seam to be removed:  1.1650135185
Iteration 50 : Cost of seam to be removed:  1.21580939292
Iteration 51 : Cost of seam to be removed:  1.21794701754
Iteration 52 : Cost of seam to be removed:  1.20802774779
Iteration 53 : Cost of seam to be removed:  1.21545565901
Iteration 54 : Cost of seam to be removed:  1.22614389479
Iteration 55 : Cost of seam to be removed:  1.22452910268
Iteration 56 : Cost of seam to be removed:  1.22763559373
Iteration 57 : Cost of seam to be removed:  1.22963480609
Iteration 58 : Cost of seam to be removed:  1.23629378822
Iteration 59 : Cost of seam to be removed:  1.24086127201
Iteration 60 : Cost of seam to be removed:  1.24402929025
Iteration 61 : Cost of seam to be removed:  1.24848909206
Iteration 62 : Cost of seam to be removed:  1.25104197824
Iteration 63 : Cost of seam to be removed:  1.25134953352
Iteration 64 : Cost of seam to be removed:  1.24625917818
Iteration 65 : Cost of seam to be removed:  1.25494815111
Iteration 66 : Cost of seam to be removed:  1.25751639519
```

d. Below image shows the path cost of the seam selected for removal from iteration 66 to 98



```
Iteration 66 : Cost of seam to be removed:  1.25751639519
Iteration 67 : Cost of seam to be removed:  1.25770093957
Iteration 68 : Cost of seam to be removed:  1.26142258535
Iteration 69 : Cost of seam to be removed:  1.26682056118
Iteration 70 : Cost of seam to be removed:  1.26861984391
Iteration 71 : Cost of seam to be removed:  1.26505196128
Iteration 72 : Cost of seam to be removed:  1.27821613528
Iteration 73 : Cost of seam to be removed:  1.26703583748
Iteration 74 : Cost of seam to be removed:  1.28033846792
Iteration 75 : Cost of seam to be removed:  1.28562865191
Iteration 76 : Cost of seam to be removed:  1.30039225546
Iteration 77 : Cost of seam to be removed:  1.30392932343
Iteration 78 : Cost of seam to be removed:  1.30435994592
Iteration 79 : Cost of seam to be removed:  1.30272980877
Iteration 80 : Cost of seam to be removed:  1.31163406268
Iteration 81 : Cost of seam to be removed:  1.29999238311
Iteration 82 : Cost of seam to be removed:  1.30356022016
Iteration 83 : Cost of seam to be removed:  1.29645526722
Iteration 84 : Cost of seam to be removed:  1.31161866177
Iteration 85 : Cost of seam to be removed:  1.29722419631
Iteration 86 : Cost of seam to be removed:  1.31223384431
Iteration 87 : Cost of seam to be removed:  1.27154180951
Iteration 88 : Cost of seam to be removed:  1.31626308791
Iteration 89 : Cost of seam to be removed:  1.31718575326
Iteration 90 : Cost of seam to be removed:  1.29050369682
Iteration 91 : Cost of seam to be removed:  1.29831607527
Iteration 92 : Cost of seam to be removed:  1.30371400739
Iteration 93 : Cost of seam to be removed:  1.30957327896
Iteration 94 : Cost of seam to be removed:  1.32232223184
Iteration 95 : Cost of seam to be removed:  1.2996847895
Iteration 96 : Cost of seam to be removed:  1.32104580032
Iteration 97 : Cost of seam to be removed:  1.30889662463
Iteration 98 : Cost of seam to be removed:  1.28336797783
Iteration 99 : Cost of seam to be removed:  1.28561327415
```

e. Below image shows the path cost of the seam selected for removal from iteration 98 to 101

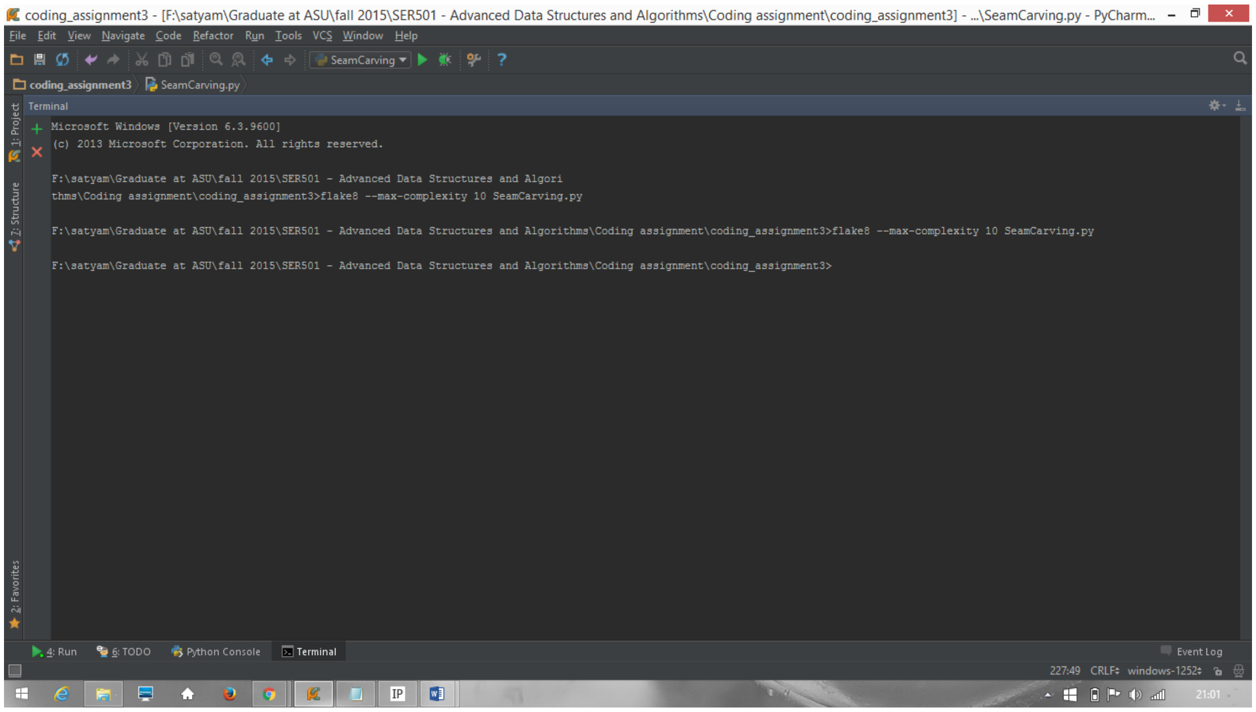


```
Iteration 69 : Cost of seam to be removed:  1.26682056118
Iteration 70 : Cost of seam to be removed:  1.26861984391
Iteration 71 : Cost of seam to be removed:  1.26505196128
Iteration 72 : Cost of seam to be removed:  1.27821613528
Iteration 73 : Cost of seam to be removed:  1.26703583748
Iteration 74 : Cost of seam to be removed:  1.28033846792
Iteration 75 : Cost of seam to be removed:  1.28562865191
Iteration 76 : Cost of seam to be removed:  1.30039225546
Iteration 77 : Cost of seam to be removed:  1.30392932343
Iteration 78 : Cost of seam to be removed:  1.30435994592
Iteration 79 : Cost of seam to be removed:  1.30272980877
Iteration 80 : Cost of seam to be removed:  1.31163406268
Iteration 81 : Cost of seam to be removed:  1.29999238311
Iteration 82 : Cost of seam to be removed:  1.30356022016
Iteration 83 : Cost of seam to be removed:  1.29645526722
Iteration 84 : Cost of seam to be removed:  1.31161866177
Iteration 85 : Cost of seam to be removed:  1.29722419631
Iteration 86 : Cost of seam to be removed:  1.31223384431
Iteration 87 : Cost of seam to be removed:  1.27154180951
Iteration 88 : Cost of seam to be removed:  1.31626308791
Iteration 89 : Cost of seam to be removed:  1.31718575326
Iteration 90 : Cost of seam to be removed:  1.29050369682
Iteration 91 : Cost of seam to be removed:  1.29831607527
Iteration 92 : Cost of seam to be removed:  1.30371400739
Iteration 93 : Cost of seam to be removed:  1.30957327896
Iteration 94 : Cost of seam to be removed:  1.32232223184
Iteration 95 : Cost of seam to be removed:  1.2996847895
Iteration 96 : Cost of seam to be removed:  1.32104580032
Iteration 97 : Cost of seam to be removed:  1.30889662463
Iteration 98 : Cost of seam to be removed:  1.28336797783
Iteration 99 : Cost of seam to be removed:  1.28561327415
Iteration 100 : Cost of seam to be removed:  1.30628225521
Iteration 101 : Cost of seam to be removed:  1.30792779419
```

3. Static Analysis / Compilation Output

For static analysis, I have used Flake8.
Below is the Flake8 output which shows that

- All the functions written have code complexity less than 10 and
- The code is free of warnings


coding_assignment3 - [F:\satyam\Graduate at ASU\fall 2015\SER501 - Advanced Data Structures and Algorithms\Coding assignment\coding_assignment3] - ...\SeamCarving.py - PyCharm...
File Edit View Navigate Code Refactor Run Tools VCS Window Help
SeamCarving
coding_assignment3 SeamCarving.py
Terminal
4: Run 6: TODO Python Console Terminal Event Log
227:49 CRLF windows-1252

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

F:\satyam\Graduate at ASU\fall 2015\SER501 - Advanced Data Structures and Algori
thms\Coding assignment\coding_assignment3>flake8 --max-complexity 10 SeamCarving.py

F:\satyam\Graduate at ASU\fall 2015\SER501 - Advanced Data Structures and Algorithms\Coding assignment\coding_assignment3>flake8 --max-complexity 10 SeamCarving.py

F:\satyam\Graduate at ASU\fall 2015\SER501 - Advanced Data Structures and Algorithms\Coding assignment\coding_assignment3>
```

4. Source Code

This section shows the formatted code to achieve the seam carving functionality.
Each function includes a doc-string explaining the input the function takes, the output of the function and a short description of the working of the function.

```python
import pylab
import math
from skimage import img_as_float


class SeamCarving:

  def dual_gradient_energy(self, img_path):
      """
      Input: Path of the image
      Output:
      a W x H array of floats, the energy at each pixel in input img.
```

```
W is width of the image
H is height of the image
Working:
Slices the image into its R, G and B components.
Calculates energy of each pixel by dual gradient energy function.
"""

img = pylab.imread(img_path)
img = img_as_float(img)
height, width = img.shape[:2]
r = img[:, :, 0]
g = img[:, :, 1]
b = img[:, :, 2]

energy = [[-1 for i in range(width)] for j in range(height)]

for i in range(height):
    for j in range(width):
        if i == 0:
            ry = r[i+1][j] - r[height-1][j]
            gy = g[i+1][j] - g[height-1][j]
            by = b[i+1][j] - b[height-1][j]
        elif i == height - 1:
            ry = r[0][j] - r[i-1][j]
            gy = g[0][j] - g[i-1][j]
            by = b[0][j] - b[i-1][j]
        else:
            ry = r[i+1][j] - r[i-1][j]
            gy = g[i+1][j] - g[i-1][j]
            by = b[i+1][j] - b[i-1][j]

        if j == 0:
            rx = r[i][j+1] - r[i][width-1]
            gx = g[i][j+1] - g[i][width-1]
            bx = b[i][j+1] - b[i][width-1]
        elif j == width - 1:
            rx = r[i][0] - r[i][j-1]
            gx = g[i][0] - g[i][j-1]
            bx = b[i][0] - b[i][j-1]
        else:
            rx = r[i][j+1] - r[i][j-1]
            gx = g[i][j+1] - g[i][j-1]
            bx = b[i][j+1] - b[i][j-1]

        delta_x = math.pow(rx, 2) + math.pow(gx, 2) + math.pow(bx, 2)
        delta_y = math.pow(ry, 2) + math.pow(gy, 2) + math.pow(by, 2)

        energy[i][j] = delta_x + delta_y
        return energy

    def find_seam(self, energy):
        """
        Input: Array of energy of each pixel
        Output:
        1. A List having column numbers for each row of the image,
        depicting the seam with lowest cost.
        2. Cost of the seam found with lowest cost.
        Working:
        Performs memoization by saving the solution of sub-problems in the
        form of matrices cost and path.
        Cost matrix helps us to find minimum cost at iteration without
        recomputing it.
        Path matrix helps us to find the seam co-ordinates without
        recomputing it.
        """

        energy = img_as_float(energy)
        height, width = energy.shape

        cost = [[0 for i in range(width)] for j in range(height)]
        path = [[-1 for i in range(width)] for j in range(height)]

        for i in range(width):
            cost[height-1][i] = energy[height-1][i]

        for i in range(height-2, -1, -1):
            for j in range(width):
                if j == 0:
                    adj = [cost[i+1][j], cost[i+1][j+1]]
                    cost[i][j] = energy[i][j] + min(adj)
                    path[i][j] = j + adj.index(min(adj))
                elif j == width-1:
                    adj = [cost[i+1][j-1], cost[i+1][j]]
                    cost[i][j] = energy[i][j] + min(adj)
                    path[i][j] = j + adj.index(min(adj)) - 1
                else:
                    adj = [cost[i+1][j-1], cost[i+1][j], cost[i+1][j+1]]
                    cost[i][j] = energy[i][j] + min(adj)
                    path[i][j] = j + adj.index(min(adj)) - 1

        min_cost = cost[0][0]
        start_index = 0
        for i in range(width):
            if cost[0][i] < min_cost:
                min_cost = cost[0][i]
                start_index = i
        seam = [start_index]
        next_index = start_index

        for i in range(height):
            seam = seam + [path[i][next_index]]
            next_index = path[i][next_index]
        return seam, min_cost

    def plot_seam(self, image, seam):
        """
        Input:
        1. image: Original image in the first instance and then
        reduced image in subsequent iteration
        2. seam: list having column numbers for each row of the image,
        depicting the seam to be removed.
        Output:
        Input image with the seam drawn, showing the seam visualization
        Working:
        Replaces the RGB values of the seam pixel co-ordinates identified by
        list - seam with (0.7, 0, 0) which RGB value for red
         """

        seam_plot = pylab.imread(image)
        seam_plot = img_as_float(seam_plot)

        height, width = seam_plot.shape[0:2]

        for i in range(height):
            for j in range(width):
                if seam[i] == j:
                    seam_plot[i][j][0] = 0.7
                    seam_plot[i][j][1] = 0
                    seam_plot[i][j][2] = 0
        pylab.imsave("SeamPlot", seam_plot)
        return seam_plot

    def remove_seam(self, img, seam):
        """
        Input:
        1. img: Original image in the first instance and then
        reduced image in subsequent iteration
        2. seam: list having column numbers for each row of the image,
        depicting the seam to be removed.
        Output:
        NewImage.png: New image saved having the seam pixels removed
        new_img: New image having the seam pixels removed from input image
        Working:
        We copy the non-seam pixels from img to a new image.
        We adopt this method since nd array is immutable.
        """

        img = pylab.imread(img)
        img = img_as_float(img)

        height = img.shape[0]
        width = img.shape[1]

        new_img = [[[0 for k in range(3)] for i in range(width-1)]
                   for j in range(height)]

        new_img = img_as_float(new_img)

        y = 0
        for i in range(height):
            for j in range(width):
                if j != seam[i]:
                    new_img[i][y][0] = img[i][j][0]
                    new_img[i][y][1] = img[i][j][1]
                    new_img[i][y][2] = img[i][j][2]
                    y = (y + 1) % (width - 1)

        pylab.imsave('NewImage.png', new_img)
        return new_img


if __name__ == '__main__':

    sc = SeamCarving()

    img = pylab.imread("HJoceanSmall.png")
    img = img_as_float(img)

    pylab.figure()
    pylab.gray()

    # Plotting the input image
    pylab.subplot(2, 2, 1)
    pylab.imshow(img)
    pylab.title('Original image')

    # Plotting the energy function of the image
    energy_image = sc.dual_gradient_energy('HJoceanSmall.png')
    pylab.subplot(2, 2, 2)
    pylab.imshow(energy_image)
    pylab.title('Energy function plot')

    seam, mininum_cost = sc.find_seam(energy_image)
    print "Iteration 1: Cost of seam to be removed: ", mininum_cost
```

```python
# Plotting the seam found on the input image
seam_plot = sc.plot_seam('HJoceanSmall.png', seam)
pylab.subplot(2, 2, 3)
pylab.imshow(seam_plot)
pylab.title('Seam Plot')

new_image = sc.remove_seam('HJoceanSmall.png', seam)

# Iterating the entire process 100 times to see significant reduction.
for i in range(100):
    energy_image = sc.dual_gradient_energy('NewImage.png')
    seam, mininum_cost = sc.find_seam(energy_image)
    print "Iteration", (i+2), ": Cost of seam to be removed: ",\
        mininum_cost
    seam_plot = sc.plot_seam('SeamPlot.png', seam)
    new_image = sc.remove_seam('NewImage.png', seam)

# Plotting the final image after computation
pylab.subplot(2, 2, 4)
pylab.imshow(new_image)
pylab.title('Final Image')
pylab.show()
```

Revised rubric for coding assignments.

This is a 5-point rubric for coding projects. Graders should refrain from using fractional points (they are a pain to defend), choose the one one number that best reflects the assignment. For assignments with multiple parts, choose the lowest scoring part.

This rubric is based on the idea that students submit PDF write-ups with their coding assignment. Write-ups *must* be PDF's with the source code so that graders can quickly view them annotate them using blackboard. The rubric does not address specific learning objectives — the assumption is that by completing the assignment the student has implicitly demonstrated some set objectives in addition to coding.

0 points — Student does not submit **all** parts of the assignment, meaning *both* a **PDF** writeup (all sections) that includes source code and output of testing, as well as a .**zip** file with source code.
2 points — The code does not run or does not *appear* to be able to run. The code it much longer than it should be, or does not appear to follow the scaffolding provided. The grader can but **does not have to verify that it does not run**, it is the student's responsibility to provide a writeup that is sufficiently convincing. Student may not appeal by coming after the fact and showing that code runs on their machine.
When grading, the grader should indicate portions of the code by annotating the writeup that are suspicious.
3 points — The code runs or looks like it would run, but the student has not shown via their writeup that it produces the correct result on reasonable inputs. Or, the student has implemented algorithms using approaches other than the ones indicated in the assignment, or the implementation has the wrong asymptotic complexity or that demonstrates a lack of understanding of the assignments objectives. The grader can, but **does not have to run the code to verify correctness** — it is the student's responsibility to make a convincing case that the output and the algorithm is correct.
When grading, the grader should indicate by annotating the write-up where results
4 points — The code runs or appears to run correctly, but has readability or style issues. The student has not demonstrated that their code has passed style guidelines, or the student's implementation appears to be unnecessarily complex (even though it looks like it works).
When grading, the grader should indicate the style problems.
5 points — No issues that we can spot.