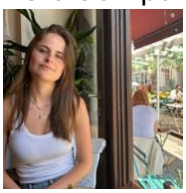
Neta Nakdimon
322216128
BS.c Computer Science



Intro to CV and Python – Ex1

Part 1- basic Python

i. pyramid_case:

I used a for loop with enumerate which provides both the index and the character at that index. I checked whether the index is even or odd using the modulo operation. If the index is even, the character is converted to lowercase using char.lower(). Else converted to uppercase using char.upper(). The modified character is appended to result string during each iteration.

The printed string after running pyramid_case on the input:

```
==== pyramid_case() results:
hElLo
wOrLd

i
aM
lEaRnInG
pYtHoN
```

ii. pyramid_case_one_liner:

To return the pyramid case word in one line of code I used the same logic of the first pyramid, but here I implemented it using list comprehension to make the code a one liner. I used a generator expression (the for loop as above) inside “”. join () which concatenates all the modified characters into a single string, forming the pyramid output.

The printed string after running pyramid_case_one_liner on the input:

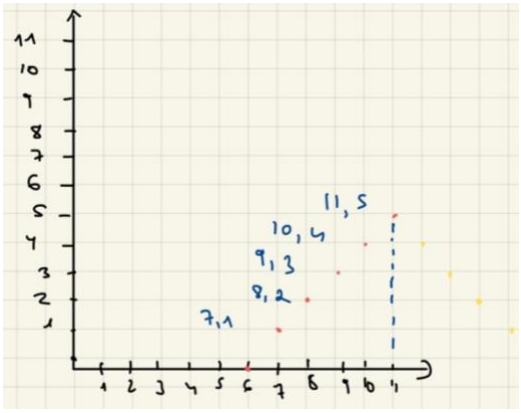
```
==== pyramid_case_one_liner() results:
hElLo
wOrLd

i
aM
lEaRnInG
pYtHoN
```

Part 2 – numpy, matplotlib

i. build_pyramid:

First, I looked for the three mistakes in the code. To get an intuition of how to implement this method I drew to myself half a pyramid on an axis system:

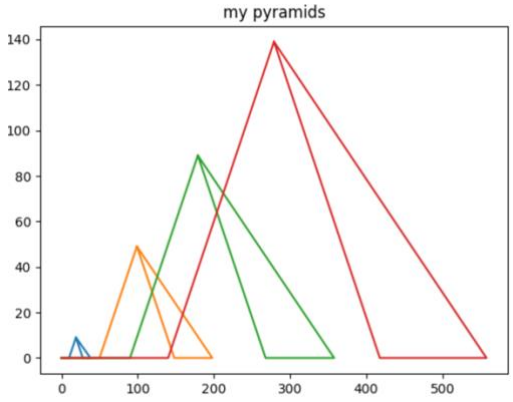


Then I realized that to create the other half we need to reverse the order of the first half x-values and add to each one the maximum x-value. The y-values stays the same.

```
xy_other_half = np.array([xy_first_half[0, ::-1] + xy_first_half[0, :].max(), xy_first_half[1, :]])
```

Another mistake in the original code was that the second half was concatenated first instead of the first one, so I changed to order of the addition to first + other. The third mistake was that the y-values were the first argument to the plt.plot and then the x-values(so the pyramids were presented on the side). I changed it to be xy_Full[0, :] , xy_full[1, :].

The pyramids after running build_pyramid:



Part 3 – OpenCV

The main steps of my code were:

1. **Thresholding** – I applied a binary threshold to the pyramid_img using cv2.theshold. Any pixel value greater than 1 is set to white (255) and all others to black (0). The output is a binary make. I used it to define regions of interest (the pyramids and not their background). I used binary threshold because as we seen in class and I also read online that it creates an unambiguous mask where each pixel is fully included(white) or excluded(black) avoiding blending issues and it makes the overlaying of the two images optimal.
2. **Resizing** – I resized the pyramid_img to match the dimensions of forest_img. The new width and height are determined by forest_img.shape[1] (width) and forest_img.shape[0] (height). Similarly, I resized the mask to match the dimensions of forest_img, ensuring compatibility for subsequent operations. (Used cv2.resize for both).
3. **Conditional pixel replacement** – I used np.where(I knew it from Machine Learning assignments) to perform a pixel by pixel operation – merging the two files such that the pyramids will be over the forest("Inside" the forest). I implemented it by checking if the corresponding pixel in mask is greater than 0, the pixel value from pyramid_img is selected. Else, the pixel value from forest image is used. This overlays the pyrmid_img onto forest_img in the regions spesifired by mask.

   The final image: