Part 3:

Create a Jupyter Notebook, create 6 of the following headings, and complete the following for your partner's assignment 1:

- Paraphrase the problem in your own words.

```
# Your answer here
print('The question 3 is asking to push all the zeros at the end of the list while maintaining the order of the non-zero numbers in the original list.')
```

```
The question 3 is asking to push all the zeros at the end of the list while maintaining the order of the non-zero numbers in the original list.
```

- Create 1 new example that demonstrates you understand the problem. Trace/walkthrough 1 example that your partner made and explain it.

```
# Your answer here
#NEW SAMPLE
nums = [0,6,0,7,0]
# The algorithm iterates and
# * Add 0 to the zeros when ith number is zero
# * Add the ith number to non_zeros when the ith number is non zero
# * Concat the result first the non_zeros followed by the zeros
# Example
# 1st iteration zeros = [0] , non_zeros = []
# 2nd iteration zeros = [0] , non_zeros = [6]
# 3rd iteration zeros = [0,0] , non_zeros = [6]
# 4th iteration zeros =[0,0] , non_zeros = [6,7]
# 5th iteration zeros = [0,0,0] non_zeros = [6,7]
# finally it will concatanate pushing all zeros to de back and solving the problem. [6,7] + [0,0,0] -> [6,7,0,0,0]
print(move_zeros_to_end(nums))

#PARTNER EXAMPLE
#input:
nums= [6,7,0,0,-2, 67, 0, 0, 3, 13, 0, 9, 17, 10, 8,]
res= move_zeros_to_end(nums)
# Following the same iteration logic explained in the new sample at the end of the iteration we will have the following values
# non_zero = [6,7,-2,67,3,13,9,17,10,8] zeros = [0,0,0,0,0]
# after concat will result in the answer [6, 7, -2, 67, 3, 13, 9, 17, 10, 8, 0, 0, 0, 0, 0]
res
```

```
[6, 7, 0, 0, 0]
```

```
[6, 7, -2, 67, 3, 13, 9, 17, 10, 8, 0, 0, 0, 0, 0]
```

- Copy the solution your partner wrote.

```
# Your answer here
from typing import List
nums = [0, 1, 0, 3, 12]
def move_zeros_to_end(nums: List[int]) -> List[int]:
    zeros=[]
    non_zeros=[]
    for x in nums:
        if x==0:
            zeros.append (x)

        else:
            non_zeros.append (x)
    res=non_zeros+zeros
    return res
res= move_zeros_to_end (nums)
print (f"input: nums= ", nums)
print (f"output:", res)
```

```
input: nums=  [0, 1, 0, 3, 12]
output: [1, 3, 12, 0, 0]
```

- Explain why their solution works in your own words.

```
# Your answer here
print('This solution works because it identifies and maintains the order of the numbers in a list while tracking the zeros needed to be added in the end')
```

```
This solution works because it identifies and maintains the order of the numbers in a list while tracking the zeros needed to be added in the end
```

- Explain the problem's time and space complexity in your own words.

```
# Your answer here
print('O(N) running complexity going over all elements and concatenating the arrays. O(N) space complexity saving all values in 2 lists.')
```

```
O(N) running complexity going over all elements and concatenating the arrays. O(N) space complexity saving all values in 2 lists.
```

- Critique your partner's solution, including explanation, and if there is anything that should be adjusted.

```
# Your answer here
print('Is a good solution as it runs in O(N) time which is optimal. ' \
'\nHowever, we could do better in space complexity O(1) with a two-pointer approach')
```

```
Is a good solution as it runs in O(N) time which is optimal.
However, we could do better in space complexity O(1) with a two-pointer approach
```

Reflection

```
# Your answer here
'''
From assignment 1 the problem assigned was to solve a the valid bracket sequence problem.
This problem the trick is to use a queue to track the brackets so you know which one first to close.
```

The time and space complexity were determined by looking at the for loop and worst-case scenario when
we have only opening brackets having O(N) stored in the queue.

For the assignment 2 path to leaves, is a recursive depth first search algorithm.
The only tricky part is handling the copies of the solution to avoid creating multiple lists as much as possible.
The time complexity is O(N) because we have to goto each of the elements in the node to each leaf to return the answer.
In case of the space complexity, the worst case is O(N) because in and scenario where there is only one leaf and many nodes
 in the tree (not balanced binary tree), the recursion stack will have size N before backtracking all the way to the root.

For the review experience, is always good to check how other people come to different solutions to a problem.
The solution was very straight forward and he even correctly identify where the code could be improved. I think that using a 2-pointer approach
to find the zeros and replace with the non-zeros found with the second pointer would be the easiest solution.

For example:

```
[1,2,0,0,3,0,1]
     ^   ^
[1,2,3,0,0,0,1]
       ^     ^
[1,2,3,1,0,0,0]
'''

'\nFrom assignment 1 the problem assigned was to solve a the valid bracket sequence problem. \nThis problem the trick is to use a queue to track the bracke
```