In [ ]:
```
import numpy as np
import pandas as pd
```

Finding allocated question

In [ ]:
```
(hash('Kamal')%3)+1
```

Out[ ]:
```
3
```

Paraphrase the problem in your own words

This is a question for finding the missing integer values in the range of 0 to the maximum value of a given list. Based on the explanations for q3 and provided examples, the problem hypothesises that the `given list` is part of the `bigger list with integer members between 0 to the maximum value of a given list`. We want to find integer values from the `bigger list` that are not in the `given list`.

In [ ]:
```
```

In the .md file containing your problem, there are examples that illustrate how the code should work. Create 2 new examples that demonstrate you understand the problem.

In [ ]:
```
# example 1
list = [1,2,7,3,11,9,6]
output = [0,4,5,8,10]
```

In [ ]:
```
# example 2
list = [8,9]
output = [0,1,2,3,4,5,6,7]
```

Code the solution to your assigned problem in Python (code chunk). Try to find the best time and space complexity solution!

First solution

In [ ]:
```
def missing_num1(lst) -> int:
    if lst:
        output = []
        for i in range(0,max(lst)+1):  #O(n)
            if i not in lst:           #O(n)
                output.append(i)     #O(1)

        if not output:             #O(1)
            return -1

        return output

    else:
        raise AssertionError('input list is empty')



# Time complexity is O(n^2)
# Space complexity is O(n)
```

In [ ]:
```
%time
l1 = [1,2,7,7,3,20,9,6,8,10,7,3,11,9,1,2,7,7,3,20,9,6,8,10,7,3,11,9,1,2,7,7,3,20,9,6,8,10,7,3,11,
missing_num1(l1)
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.01 µs
```

Out[ ]:
```
[0, 4, 5, 12, 13, 14, 15, 16, 17, 18, 19]
```

In [ ]:
```
l2 = [0,1,2,4,5,8,10,7,3,11,9,6]
missing_num1(l2)
```

Out[ ]: -1

Second solution

In [ ]:
```
def missing_num2(lst) -> int:
    if lst:
        rlst = np.arange(0,max(lst)+1,1)    #O(n)
        lst = np.array(lst)                 #O(n)

        diff = np.setdiff1d(rlst,lst).tolist() # O(nlog(n)) becuase setdiff1d needs to sort array

        if not diff:
            return -1

        return diff  #O(m) m<n

    else:
        raise AssertionError('input list is empty')


# Time complexity is O(nlog(n))
# Space complexity is O(m) ~ O(n)
```

In [ ]:
```
%time
l1 = [1,2,7,7,3,20,9,6,8,10,7,3,11,9,1,2,7,7,3,20,9,6,8,10,7,3,11,9,1,2,7,7,3,20,9,6,8,10,7,3,11,
missing_num2(l1)
```

```
CPU times: user 1 µs, sys: 0 ns, total: 1 µs
Wall time: 3.1 µs
```

Out[ ]: [0, 4, 5, 12, 13, 14, 15, 16, 17, 18, 19]

In [ ]:

Explain why your solution works

Both missing_num1 and missing_num2 are methods to find missing numbers in a list within a specified range.

- `missing_num1` method iterates through a range of numbers expected to be in the list (from `0` to `max(lst)`), checking for the presence of each number in the input list. Since it checks for the presence of each number within the entire range of values that should be in the list, it can accurately identify which numbers are missing. The method can be inefficient for large lists due to its `O(n^2)` time complexity.
- `missing_num2` method uses NumPy's `np.arange` to generate a complete range array from `0` to `max(lst)`, and then uses `np.setdiff1d` to find the set difference between this range array and the input list, effectively identifying missing numbers. `np.setdiff1d` computes the difference between this complete range and the actual numbers in the list, determining exactly which numbers are missing. It can be more efficient than `missing_num1` regarding execution time for larger lists. The time complexity comes from sorting operations within np.setdiff1d, generally `O(nlogn)`, making it suitable for large datasets.

In [ ]:

Explain the problem's and space complexity

The time and space complexity of these two methods are:

- The time complexity of `missing_num1` is `O(n^2)` becuse of using `for loop` and `if function` for searching missing integers. Its space complexity is `O(m)` becuase the output is a list with m members that it is less than n. However we can assume that the space complexity for this method is `O(n)`.
- Becase `missing_num2` method uses NumPy's `np.setdiff1d` function to find the missing integers, its time complexity is `O(nlogn)`. Similarly, its space complexity is `O(m)` becuase the output is a list with m members that it is less than n.

In [ ]:

Explain the thinking to an alternative solution (no coding required, but a classmate reading this should be able to code it up based off your text)

Follow these steps to implement the alternative solution for finding missing numbers in a given list.

- First, check if the `input list` is empty. If the input list is empty, the function immediately raises an error indicating that the `input list` is empty.
- Use NumPy's `arange function` to create an array with values between `0` and the maximum value in the input list plus one and with step=1.
- To have an efficient execution of array operations, convert the original input list to a NumPy array.
- In this step, use the `setdiff1d function` from NumPy to find the difference between `the array generated in step 2` and `the input list array`.
- Here, you need to convert the resulting array of missing numbers to a Python list.
- Then, you need to check whether the list of missing numbers is empty. If it is empty, your function should return `-1` to indicate no missing numbers. Otherwise, return the list of missing numbers.