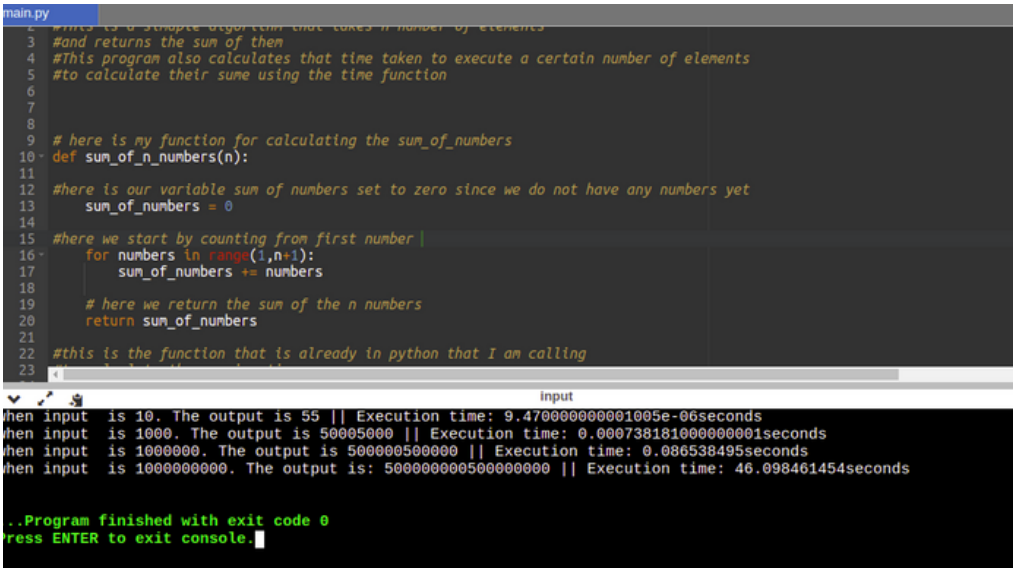
HEATHER MATARUSE

ADVANCED ALGORITHMS SUMMATIVE

GITHUB CODE HERE

DECEMBER 16 , 2021

Question 1



Pseudo-code

- create a function that calculates the sum of the **n** numbers
- Inside our function, we have a variable sum of numbers at zero that tracks the number of numbers
- Then you use a for loop to loop through the numbers you are adding together
- The function should make sure that it returns the sum of the n items

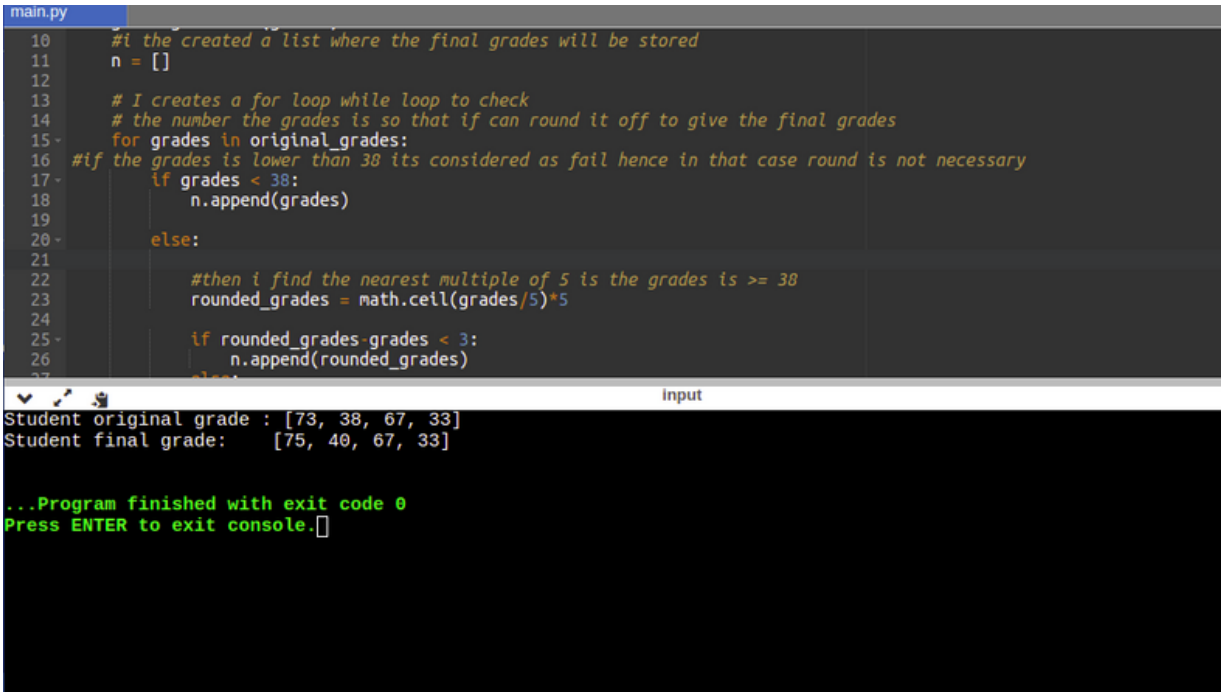
Time Complexity

- As the algorithm runs it takes O(n) time because as we can see from the above output the time increases as the number of the numbers increases

Observations

- For the 10 , 1000, and 100000 elements that running time was less but for the 1000000000 it took a while for the output to be presented as we can see it took 46 seconds

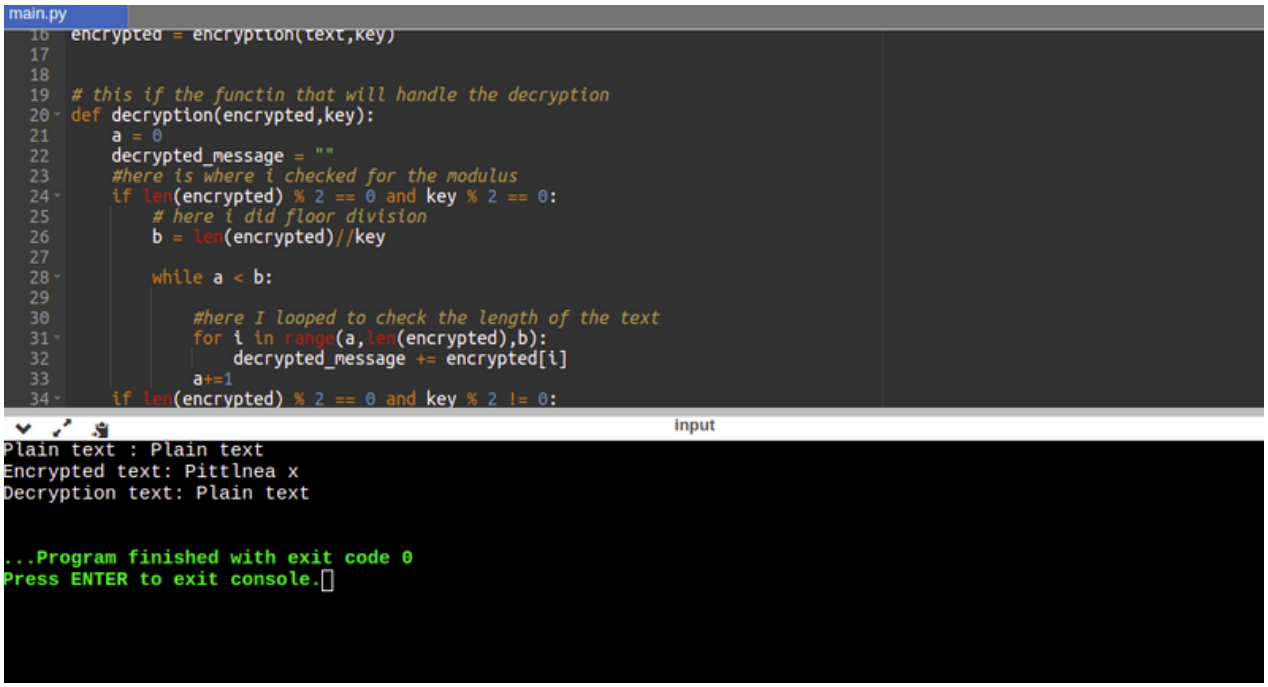
Question 2



Pseudocode

- I created a function called gradingStudents that takes in a list of the students' grades.
- After that,I created an empty list called final grade to hold the rounded grades.
- For the input array of the student's grades .
- If the grade is less than 38, it should be added to the final grade.
- If your grade is higher than or equal to 38,
- you then find the nearest multiple of 5 by dividing the grade by 5 and rounding the result to the nearest whole number.
- and then multiply it by 5 to save the result as rounded grade
- Append the rounded grade to the final grade array if the difference between the rounded grade and the original grade is less than 3.
- Otherwise, add the original grade to the final grade.

Question 3



Pseudo-code for decryption

- Create an encryption function that takes two arguments, the text and the key. Within the function, create an empty string variable called encrypted message to store the encrypted message, as well as an integer variable called start (set to 0), which will help you determine where to start and end based on the key. The algorithm here is to loop through the key and then through the text in any gaps in the key. For example, if the key is 2, we will loop through the text in 2 step increments and add the characters to the storing variable (encrypted message). In the end, we return the encrypted message

Pseudo-code for decryption

- create the function , Check for edges cases if the length of the text is even and the key is even, loop through the length of the word starting at start and take b steps every time. Add the current index of start in the encrypted message character to decrypted_message and then return dencrypted_message

Question 4

```python
def superDigits(x,k):
    s = digsum(x)
    return sup_digit(str(int(s)*k))

def sup_digit(x):
    if len(x) <= 1:
        return x
    else:
        return sup_digit( digsum(x) )

def digsum(x):
    return str(sum((int(i) for i in list(x))))
```

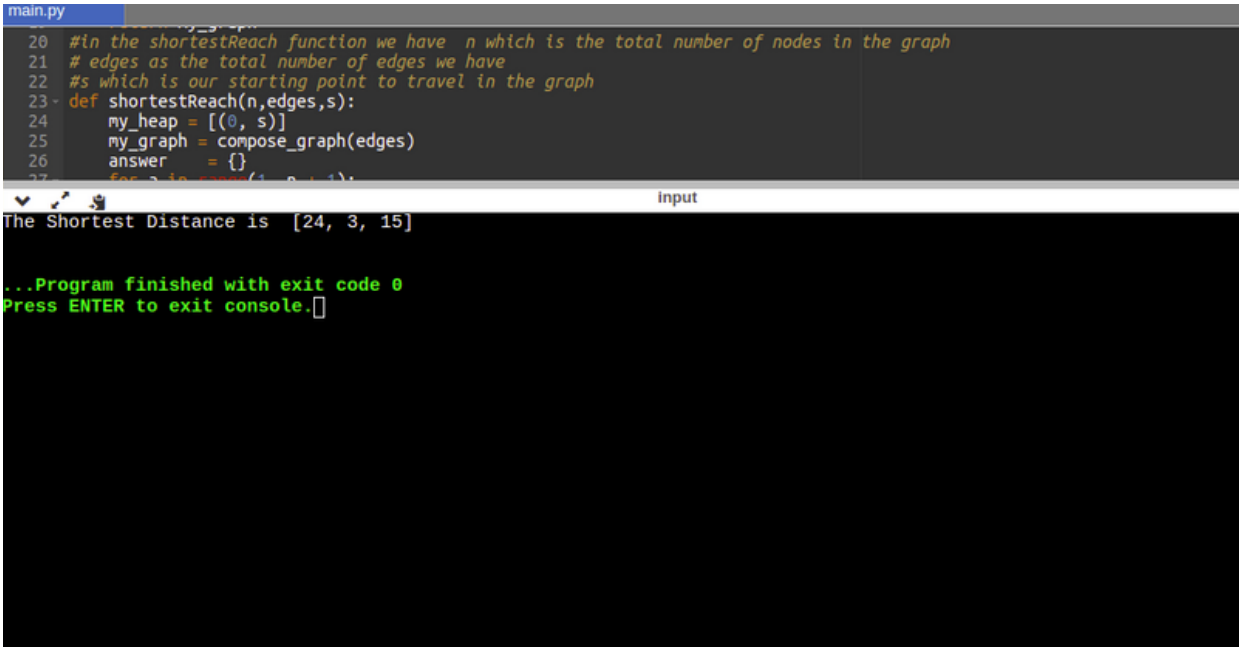
```
input
 = 9875, k = 4, concatenated value = 9875987598759875 superdigit is  8
 = 148, k = 3, concatenated value = 148148148superdigit is   3
 = 56345, k = 6, concatenated value = 563455634556345634556345634556345 superdigit is 3


..Program finished with exit code 0
ress ENTER to exit console.
```

Pseudo-code

- created a function that takes two arguments, n and k which will calculate the super digit
- I also have another function that will calculate the sum
- And another function that checks if the sum is a single number or not
- Lastly I just printed out the output using the test cases

Question 5



Pseudo-code

I wrote a function called compose graph to generate the graph's edges.
I then made a dictionary with the edges as keys and values.
Then I mapped the edges to one another, storing the weights in a tuple because I don't want the value to change.
The function produces a graph, which I will use in the following function.
The next step in the algorithm is to write a function called shortestReach that accepts arguments such as the number of edges, the array of edges, and the starting point, which is s.
In the function where I will use the compose graph function, we will create an answer dictionary to keep track of the shortest reach, with the edges as the key and infinity as the value.
Because we're starting with s, set the value of s in the answer to 0

Lastly, you loop through the dictionary answer with values as the shortest reach and append them to shortest_reach which return shortest_reach the array with the shortest reach of each edge starting from one