ICT1018 – Data Structures and Algorithms Coursework Documentation

- Question 1
    - Source Code & Explanation

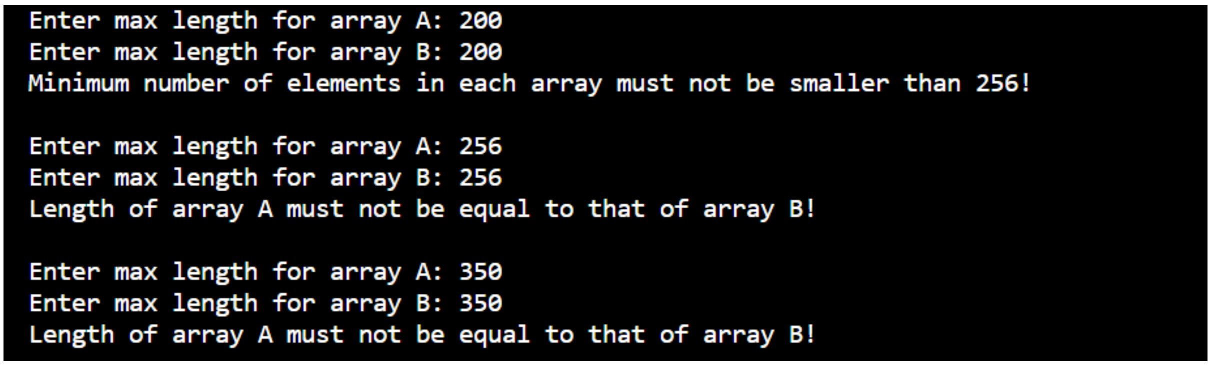
```
# Import random package - used to populate arrays with random integers between
0 and 1024
import random

def main():
    # Declared as global variables in order to be used in q2 program
    global array_A, array_B

    # Declaration of arrays A and B
    array_A = []
    array_B = []

    # Check whether size of both arrays is valid
    isValid = False

    while not isValid:
        sizeA = input("Enter max length for array A: ")
        sizeB = input("Enter max length for array B: ")

        # Check that user enters integers and not anything else
        if (len(sizeA) > 0 and len(sizeB) > 0) and (sizeA.isdigit() and
sizeB.isdigit()):
            sizeA = int(sizeA)
            sizeB = int(sizeB)

            # Check that size of both arrays satisfies the following
conditions
            if (sizeA < 256) or (sizeB < 256):  # size of any one of arrays
must not be less than 256
                isValid = False
                print("Minimum number of elements in each array must not be
smaller than 256!\n")
            elif sizeA == sizeB:    # size of array A must not be equal to
that of array B
                isValid = False
                print("Length of array A must not be equal to that of array
B!\n")
            else: # size of arrays A and B satisfies the above conditions
                isValid = True
        else:
            isValid = False
            print("Please enter integers!\n")
    # Populate arrays with random integers between 0 and 1024
    for index1 in range(0, sizeA):
        number = random.randint(0, 1024)
        array_A.append(number)

    for index2 in range(0, sizeB):
        number = random.randint(0, 1024)
        array_B.append(number)

    # Perform shell sort on array A
    shellSort(array_A, len(array_A))

    # Perform quick sort on array B
    quickSort(array_B, 0, len(array_B)-1)

    # Print sorted arrays after finishing execution
    print("\nArray A sorted using Shell Sort:",array_A)
    print("\nArray B sorted using Quick Sort:",array_B)


# Function which takes an array as a parameter and performs shell sort on it
def shellSort(list, n):
    # Rearrange elements at each n/2, n/4, n/8, ... intervals
    interval = n // 2
    while interval > 0:
        for index in range(interval, n):
            temp = list[index]
            j = index
            while j >= interval and list[j - interval] > temp:
                list[j] = list[j - interval]
                j -= interval

            list[j] = temp
        interval //= 2


# Function which takes an array as a parameter and performs quick sort on it
def quickSort(array, low, high):
    if low < high:

        # Find pivot element such that
        # elements smaller than pivot are on the left
        # elements greater than pivot are on the right
        pivot = partition(array, low, high)

        # Recursive call on the left of pivot (elements smaller than pivot
element)
        quickSort(array, low, pivot-1)
        # Recursive call on the right of pivot (elements larger than pivot
element)
        quickSort(array, pivot+1, high)


# Function to find the partition position
def partition(array, low, high):

    # Choose the rightmost element as pivot element
    pivot = array[high]

    # Index pointer used for greater element
    index1 = low - 1

    # Traverse through all elements
    # compare each element with pivot
    for index2 in range(low, high):
        if array[index2] <= pivot:

            # If element that is smaller than pivot is found
            # Then swap it with the greater element pointed by i
            index1 += 1

            # Swapping element at index i with element at index j
            (array[index1], array[index2]) = (array[index2], array[index1])

    # Swap the pivot element with the greater element specified by index i
    (array[index1 + 1], array[high]) = (array[high], array[index1 + 1])

    # Return the position from where partition is done
    return index1 + 1


# Call main() function to start execution
main()
```

The program starts by declaring 2 variables called array_A and array_B respectively and initialising them as empty lists. Then, it asks the user to input the number of elements for each array. Afterwards, the user input goes through 2 checks. The program first checks if both values entered by the user are integers. If successful, it then checks if both numbers are of unequal value and are greater than or equal to 256. If unsuccessful, an error gets displayed on the console and the user is prompted to enter new values. Otherwise, the program exits the loop.

Once this process is completed, both arrays are then filled with random integers between 0 and 1024 according to the size entered by the user. Then, the program performs shell sort on array A by calling its respective method. The same is then done for array B which is sorted using quick sort. The final arrays are then printed out to the user.

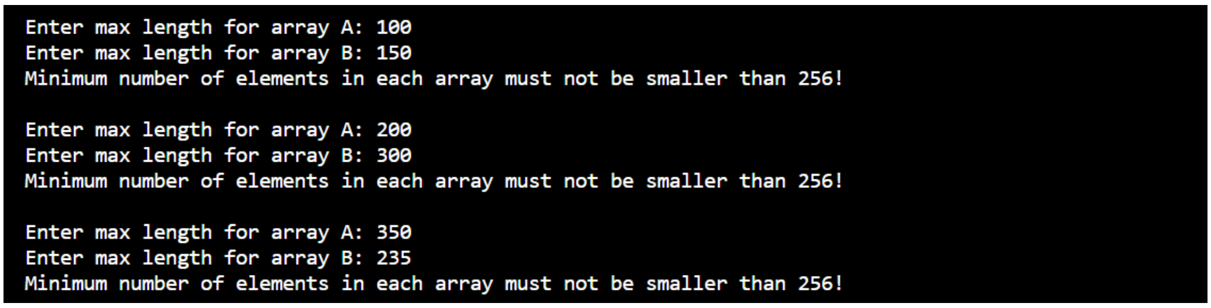
❖ How the Program was Tested + Screen Dumps

The program was first tested by entering numbers that do not pass the first condition, that they must be of unequal size. The numbers tested were 200, 256, and 350. For the first instance, although the numbers entered were equal, another message was printed on the console, stating that the numbers did not pass the second condition. For the last two instances, the correct message was displayed, telling the user that the numbers entered were equal. This can be seen below as follows:



```
Enter max length for array A: 200
Enter max length for array B: 200
Minimum number of elements in each array must not be smaller than 256!

Enter max length for array A: 256
Enter max length for array B: 256
Length of array A must not be equal to that of array B!

Enter max length for array A: 350
Enter max length for array B: 350
Length of array A must not be equal to that of array B!
```

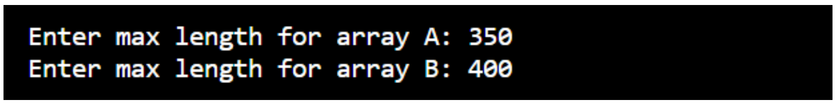
Next, the program was tested by entering numbers that do not pass the second condition, that they must be at least 256. The sets of numbers tested were (100 and 150), (200 and 300), and (350 and 235). This was done to test the program's capability to detect if either one of the numbers is less than 256 or even both. For all instances, a message was displayed on the console, stating to the user that the numbers entered were smaller than 256 and that they must have a minimum of that value. This can be seen below as follows:



```
Enter max length for array A: 100
Enter max length for array B: 150
Minimum number of elements in each array must not be smaller than 256!

Enter max length for array A: 200
Enter max length for array B: 300
Minimum number of elements in each array must not be smaller than 256!

Enter max length for array A: 350
Enter max length for array B: 235
Minimum number of elements in each array must not be smaller than 256!
```

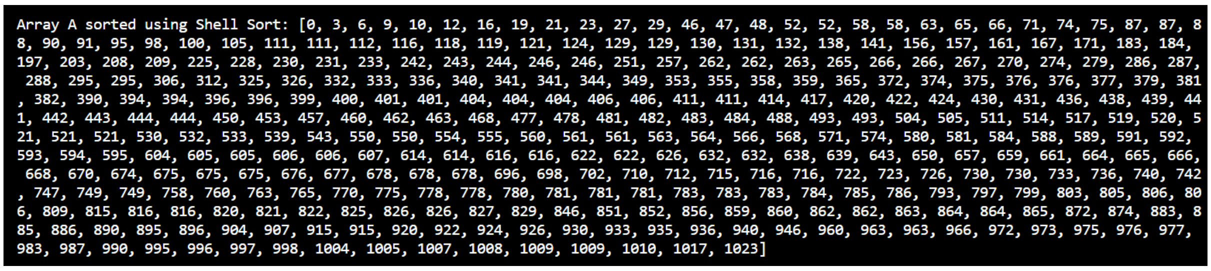
The program was then tested by entering values that pass both conditions, but the first number is smaller than the second number. The numbers tested were 350 and 400. As expected, a message did not display stating that both values did not pass either condition. The program also printed out the sorted versions of arrays A and B after executing their respective sorting algorithms. As expected, array A had a smaller number of elements that array B. This is shown below:

Entering values



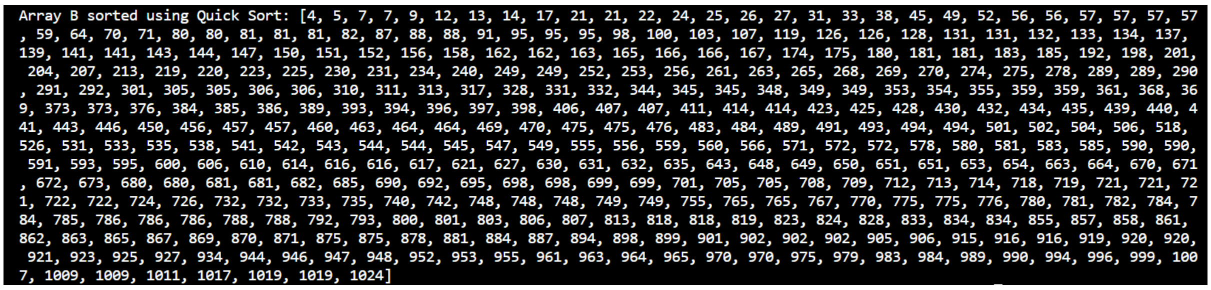
```
Enter max length for array A: 350
Enter max length for array B: 400
```

Final version of Array A displayed:



```
Array A sorted using Shell Sort: [0, 3, 6, 9, 10, 12, 16, 19, 21, 23, 27, 29, 46, 47, 48, 52, 52, 58, 58, 63, 65, 66, 71, 74, 75, 87, 87, 8
8, 90, 91, 95, 98, 100, 105, 111, 111, 112, 116, 118, 119, 121, 124, 129, 129, 130, 131, 132, 138, 141, 156, 157, 161, 167, 171, 183, 184,
197, 203, 208, 209, 225, 228, 230, 231, 233, 242, 243, 244, 246, 246, 251, 257, 262, 262, 263, 265, 266, 266, 267, 270, 274, 279, 286, 287,
 288, 295, 295, 306, 312, 325, 326, 332, 333, 336, 340, 341, 341, 344, 349, 353, 355, 358, 359, 365, 372, 374, 375, 376, 376, 377, 379, 381
, 382, 390, 394, 394, 396, 396, 399, 400, 401, 401, 404, 404, 404, 406, 406, 411, 411, 414, 417, 420, 422, 424, 430, 431, 436, 438, 439, 44
1, 442, 443, 444, 444, 450, 453, 457, 460, 462, 463, 468, 477, 478, 481, 482, 483, 484, 488, 493, 493, 504, 505, 511, 514, 517, 519, 520, 5
21, 521, 521, 530, 532, 533, 539, 543, 550, 550, 554, 555, 560, 561, 561, 563, 564, 566, 568, 571, 574, 580, 581, 584, 588, 589, 591, 592,
593, 594, 595, 604, 605, 605, 606, 606, 607, 614, 614, 616, 616, 622, 622, 626, 632, 632, 638, 639, 643, 650, 657, 659, 661, 664, 665, 666,
 668, 670, 674, 675, 675, 675, 676, 677, 678, 678, 678, 696, 698, 702, 710, 712, 715, 716, 716, 722, 723, 726, 730, 730, 733, 736, 740, 742
, 747, 749, 749, 758, 760, 763, 765, 770, 775, 778, 778, 780, 781, 781, 781, 783, 783, 783, 784, 785, 786, 793, 797, 799, 803, 805, 806, 80
6, 809, 815, 816, 816, 820, 821, 822, 825, 826, 826, 827, 829, 846, 851, 852, 856, 859, 860, 862, 862, 863, 864, 864, 865, 872, 874, 883, 8
85, 886, 890, 895, 896, 904, 907, 915, 915, 920, 922, 924, 926, 930, 933, 935, 936, 940, 946, 960, 963, 963, 966, 972, 973, 975, 976, 977,
983, 987, 990, 995, 996, 997, 998, 1004, 1005, 1007, 1008, 1009, 1009, 1010, 1017, 1023]
```

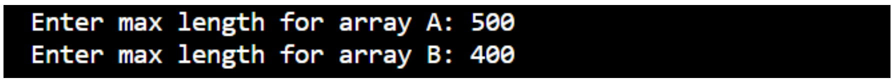
Final version of Array B displayed:



```
Array B sorted using Quick Sort: [4, 5, 7, 7, 9, 12, 13, 14, 17, 21, 21, 22, 24, 25, 26, 27, 31, 33, 38, 45, 49, 52, 56, 56, 57, 57, 57, 57
, 59, 64, 70, 71, 80, 80, 81, 81, 81, 82, 87, 88, 88, 91, 95, 95, 95, 98, 100, 103, 107, 119, 126, 126, 128, 131, 131, 132, 133, 134, 137,
139, 141, 141, 143, 144, 147, 150, 151, 152, 156, 158, 162, 162, 163, 165, 166, 166, 167, 174, 175, 180, 181, 181, 183, 185, 192, 198, 201,
 204, 207, 213, 219, 220, 223, 225, 230, 231, 234, 240, 249, 249, 252, 253, 256, 261, 263, 265, 268, 269, 270, 274, 275, 278, 289, 289, 290
, 291, 292, 301, 305, 305, 306, 306, 310, 311, 313, 317, 328, 331, 332, 344, 345, 345, 348, 349, 349, 353, 354, 355, 359, 359, 361, 368, 36
9, 373, 373, 376, 384, 385, 386, 389, 393, 394, 396, 397, 398, 406, 407, 407, 411, 414, 414, 423, 425, 428, 430, 432, 434, 435, 439, 440, 4
41, 443, 446, 450, 456, 457, 457, 460, 463, 464, 464, 469, 470, 475, 475, 476, 483, 484, 489, 491, 493, 494, 494, 501, 502, 504, 506, 518,
526, 531, 533, 535, 538, 541, 542, 543, 544, 544, 545, 547, 549, 555, 556, 559, 560, 566, 571, 572, 572, 578, 580, 581, 583, 585, 590, 590,
 591, 593, 595, 600, 606, 610, 614, 616, 616, 617, 621, 627, 630, 631, 632, 635, 643, 648, 649, 650, 651, 651, 653, 654, 663, 664, 670, 671
, 672, 673, 680, 680, 681, 681, 682, 685, 690, 692, 695, 698, 698, 699, 699, 701, 705, 705, 708, 709, 712, 713, 714, 718, 719, 721, 721, 72
1, 722, 722, 724, 726, 732, 732, 733, 735, 740, 742, 748, 748, 748, 749, 749, 755, 765, 765, 767, 770, 775, 775, 776, 780, 781, 782, 784, 7
84, 785, 786, 786, 786, 788, 788, 792, 793, 800, 801, 803, 806, 807, 813, 818, 818, 819, 823, 824, 828, 833, 834, 834, 855, 857, 858, 861,
862, 863, 865, 867, 869, 870, 871, 875, 875, 878, 881, 884, 887, 894, 898, 899, 901, 902, 902, 902, 905, 906, 915, 916, 916, 919, 920, 920,
 921, 923, 925, 927, 934, 944, 946, 947, 948, 952, 953, 955, 961, 963, 964, 965, 970, 970, 975, 979, 983, 984, 989, 990, 994, 996, 999, 100
7, 1009, 1009, 1011, 1017, 1019, 1019, 1024]
```

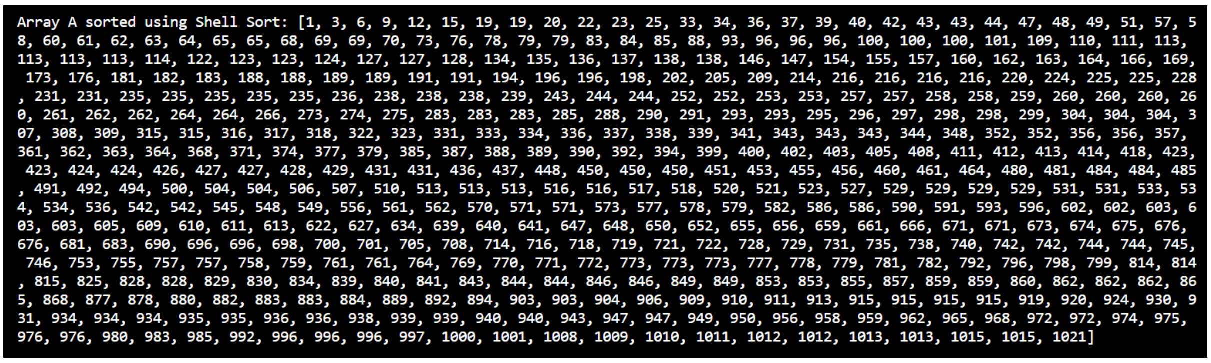
Afterwards, the program was tested by again entering values that pass both conditions, but this time, the first number is greater than the second number. The numbers tested were 500 and 400. As expected, a message did not display stating that both values did not pass either condition. The program also printed out the sorted versions of arrays A and B after executing their respective sorting algorithms. As expected, array A had a greater number of elements that array B. This is shown below:

Entering values



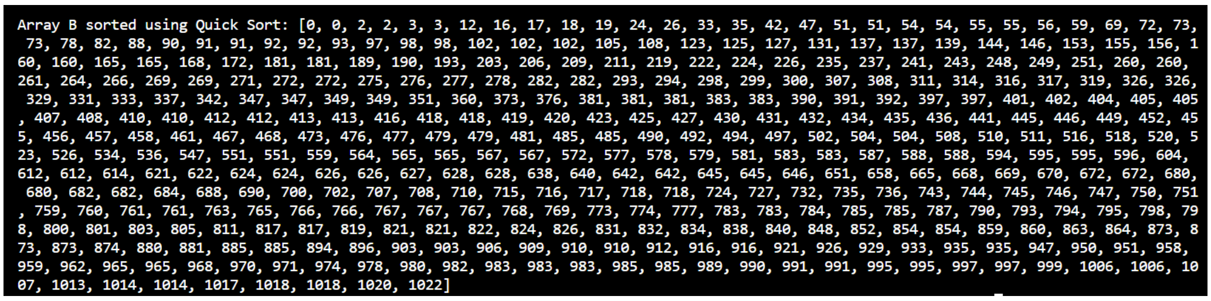
```
Enter max length for array A: 500
Enter max length for array B: 400
```

Final version of Array A displayed:



```
Array A sorted using Shell Sort: [1, 3, 6, 9, 12, 15, 19, 19, 20, 22, 23, 25, 33, 34, 36, 37, 39, 40, 42, 43, 43, 44, 47, 48, 49, 51, 57, 5
8, 60, 61, 62, 63, 64, 65, 65, 68, 69, 69, 70, 73, 76, 78, 79, 79, 83, 84, 85, 88, 93, 96, 96, 96, 100, 100, 100, 101, 109, 110, 111, 113,
113, 113, 113, 114, 122, 123, 123, 124, 127, 127, 128, 134, 135, 136, 137, 138, 138, 146, 147, 154, 155, 157, 160, 162, 163, 164, 166, 169,
 173, 176, 181, 182, 183, 188, 188, 189, 189, 191, 191, 194, 196, 196, 198, 202, 205, 209, 214, 216, 216, 216, 216, 220, 224, 225, 225, 228
, 231, 231, 235, 235, 235, 235, 235, 236, 238, 238, 238, 239, 243, 244, 244, 252, 252, 253, 253, 257, 257, 258, 258, 259, 260, 260, 260, 26
0, 261, 262, 262, 264, 264, 266, 273, 274, 275, 283, 283, 283, 285, 288, 290, 291, 293, 293, 295, 296, 297, 298, 298, 299, 304, 304, 304, 3
07, 308, 309, 315, 315, 316, 317, 318, 322, 323, 331, 333, 334, 336, 337, 338, 339, 341, 343, 343, 343, 344, 348, 352, 352, 356, 356, 357,
361, 362, 363, 364, 368, 371, 374, 377, 379, 385, 387, 388, 389, 390, 392, 394, 399, 400, 402, 403, 405, 408, 411, 412, 413, 414, 418, 423,
 423, 424, 424, 426, 427, 427, 428, 429, 431, 431, 436, 437, 448, 450, 450, 450, 451, 453, 455, 456, 460, 461, 464, 480, 481, 484, 484, 485
, 491, 492, 494, 500, 504, 504, 506, 507, 510, 513, 513, 513, 516, 516, 517, 518, 520, 521, 523, 527, 529, 529, 529, 529, 531, 531, 533, 53
4, 534, 536, 542, 542, 545, 548, 549, 556, 561, 562, 570, 571, 571, 573, 577, 578, 579, 582, 586, 586, 590, 591, 593, 596, 602, 602, 603, 6
03, 603, 605, 609, 610, 611, 613, 622, 627, 634, 639, 640, 641, 647, 648, 650, 652, 655, 656, 659, 661, 666, 671, 671, 673, 674, 675, 676,
676, 681, 683, 690, 696, 696, 698, 700, 701, 705, 708, 714, 716, 718, 719, 721, 722, 728, 729, 731, 735, 738, 740, 742, 742, 744, 744, 745,
 746, 753, 755, 757, 757, 758, 759, 761, 761, 764, 769, 770, 771, 772, 773, 773, 773, 777, 778, 779, 781, 782, 792, 796, 798, 799, 814, 814
, 815, 825, 828, 828, 829, 830, 834, 839, 840, 841, 843, 844, 844, 846, 846, 849, 849, 853, 853, 855, 857, 859, 859, 860, 862, 862, 862, 86
5, 868, 877, 878, 880, 882, 883, 883, 884, 889, 892, 894, 903, 903, 904, 906, 909, 910, 911, 913, 915, 915, 915, 915, 919, 920, 924, 930, 9
31, 934, 934, 934, 935, 935, 936, 936, 938, 939, 939, 940, 940, 943, 947, 947, 949, 950, 956, 958, 959, 962, 965, 968, 972, 972, 974, 975,
976, 976, 980, 983, 985, 992, 996, 996, 996, 997, 1000, 1001, 1008, 1009, 1010, 1011, 1012, 1012, 1013, 1013, 1015, 1015, 1021]
```

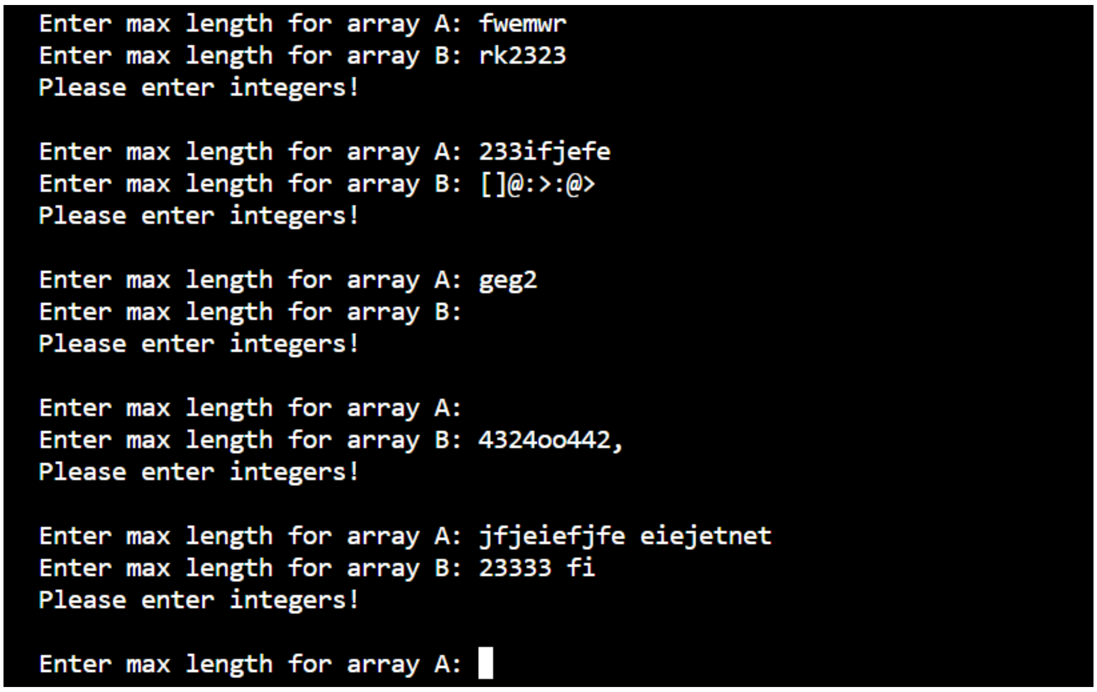
<u>Final version of Array B displayed:</u>



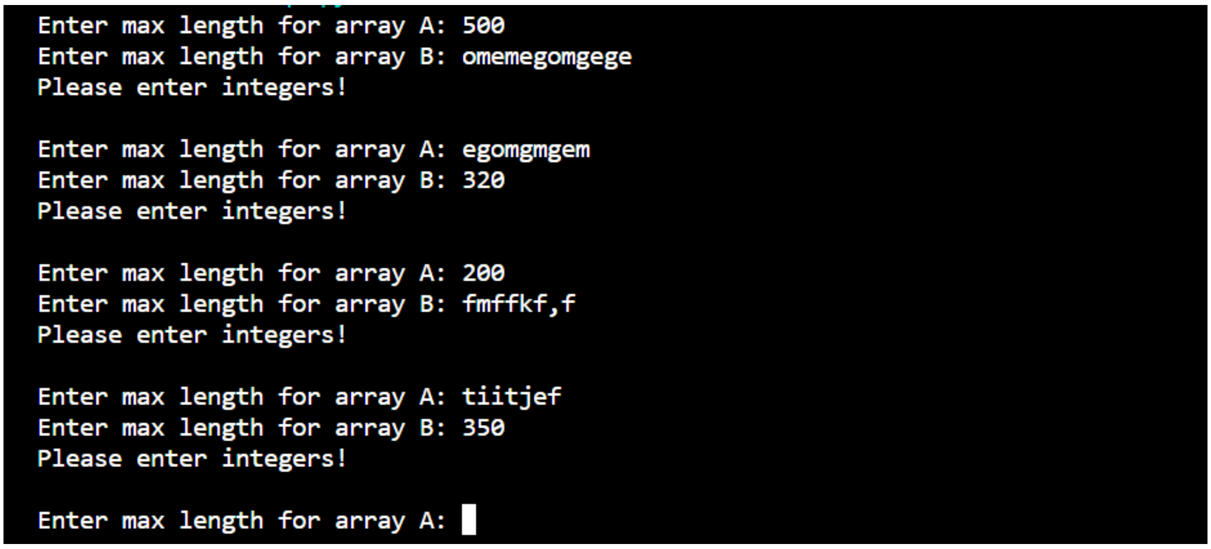
```
Array B sorted using Quick Sort: [0, 0, 2, 2, 3, 3, 12, 16, 17, 18, 19, 24, 26, 33, 35, 42, 47, 51, 51, 54, 54, 55, 55, 56, 59, 69, 72, 73,
 73, 78, 82, 88, 90, 91, 91, 92, 92, 93, 97, 98, 98, 102, 102, 102, 105, 108, 123, 125, 127, 131, 137, 137, 139, 144, 146, 153, 155, 156, 1
60, 160, 165, 165, 168, 172, 181, 181, 189, 190, 193, 203, 206, 209, 211, 219, 222, 224, 226, 235, 237, 241, 243, 248, 249, 251, 260, 260,
261, 264, 266, 269, 269, 271, 272, 272, 275, 276, 277, 278, 282, 282, 293, 294, 298, 299, 300, 307, 308, 311, 314, 316, 317, 319, 326, 326,
 329, 331, 333, 337, 342, 347, 347, 349, 349, 351, 360, 373, 376, 381, 381, 381, 383, 383, 390, 391, 392, 397, 397, 401, 402, 404, 405, 405
, 407, 408, 410, 410, 412, 412, 413, 413, 416, 418, 418, 419, 420, 423, 425, 427, 430, 431, 432, 434, 435, 436, 441, 445, 446, 449, 452, 45
5, 456, 457, 458, 461, 467, 468, 473, 476, 477, 479, 479, 481, 485, 485, 490, 492, 494, 497, 502, 504, 504, 508, 510, 511, 516, 518, 520, 5
23, 526, 534, 536, 547, 551, 551, 559, 564, 565, 565, 567, 567, 572, 577, 578, 579, 581, 583, 583, 587, 588, 588, 594, 595, 595, 596, 604,
612, 612, 614, 621, 622, 624, 624, 626, 626, 627, 628, 628, 638, 640, 642, 642, 645, 645, 646, 651, 658, 665, 668, 669, 670, 672, 672, 680,
 680, 682, 682, 684, 688, 690, 700, 702, 707, 708, 710, 715, 716, 717, 718, 718, 724, 727, 732, 735, 736, 743, 744, 745, 746, 747, 750, 751
, 759, 760, 761, 761, 763, 765, 766, 766, 767, 767, 767, 768, 769, 773, 774, 777, 783, 783, 784, 785, 785, 787, 790, 793, 794, 795, 798, 79
8, 800, 801, 803, 805, 811, 817, 817, 819, 821, 821, 822, 824, 826, 831, 832, 834, 838, 840, 848, 852, 854, 854, 859, 860, 863, 864, 873, 8
73, 873, 874, 880, 881, 885, 885, 894, 896, 903, 903, 906, 909, 910, 910, 912, 916, 916, 921, 926, 929, 933, 935, 935, 947, 950, 951, 958,
959, 962, 965, 965, 968, 970, 971, 974, 978, 980, 982, 983, 983, 983, 985, 985, 989, 990, 991, 991, 995, 995, 997, 997, 999, 1006, 1006, 10
07, 1013, 1014, 1014, 1017, 1018, 1018, 1020, 1022]
```

Afterwards, the program was tested by inputting several strings. This was done to test the data type check for the user input. The tested strings were divided in four parts, whole text, text with numbers included, empty values, and special characters. For all these instances, the console displayed a message telling the user to input integers, due to the values being invalid for the program to continue running.

This was even tested by entering 1 integer value and 1 string value only. The positions of the values are shifted to test the program's ability to detect any user input values of the wrong datatype before setting the values to the size of both arrays A and B. This is shown below:

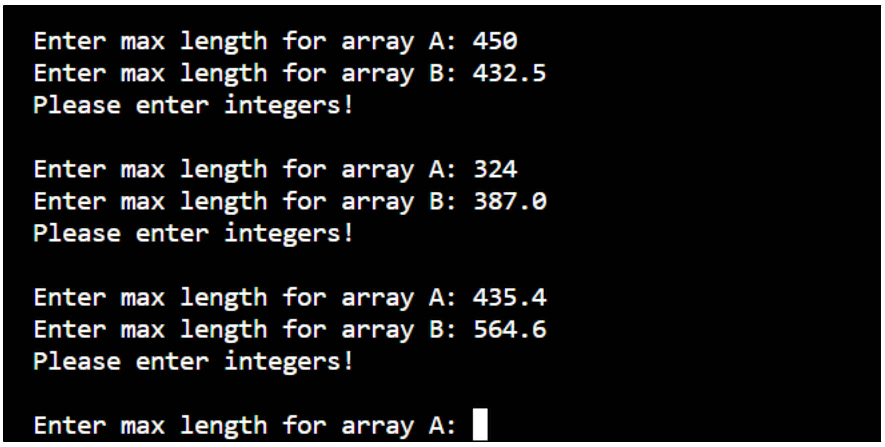


```
Enter max length for array A: fwemwr
Enter max length for array B: rk2323
Please enter integers!

Enter max length for array A: 233ifjefe
Enter max length for array B: []@:>:@>
Please enter integers!

Enter max length for array A: geg2
Enter max length for array B:
Please enter integers!

Enter max length for array A:
Enter max length for array B: 4324oo442,
Please enter integers!

Enter max length for array A: jfjeiefjfe eiejetnet
Enter max length for array B: 23333 fi
Please enter integers!

Enter max length for array A:
```



```
Enter max length for array A: 500
Enter max length for array B: omemegomgege
Please enter integers!

Enter max length for array A: egomgmgem
Enter max length for array B: 320
Please enter integers!

Enter max length for array A: 200
Enter max length for array B: fmffkf,f
Please enter integers!

Enter max length for array A: tiitjef
Enter max length for array B: 350
Please enter integers!

Enter max length for array A:
```

Finally, the program was tested by inputting several numbers of type float. The tested values included numbers with a lot of decimal places and those ending with .0. For all these instances, the console displayed a message telling the user to input an integer, just like the previous test. The results of this test can be seen below:



```
Enter max length for array A: 450
Enter max length for array B: 432.5
Please enter integers!

Enter max length for array A: 324
Enter max length for array B: 387.0
Please enter integers!

Enter max length for array A: 435.4
Enter max length for array B: 564.6
Please enter integers!

Enter max length for array A: 
```

- Question 2
    - ❖ Source Code & Explanation

```
# Import q1 program to use data for q2 - runs q1 program at start of execution
import q1 as question1Program

def main():
    # Retrieve data from arrays A and B after q1 program finishes running
    arrayA = question1Program.array_A
    arrayB = question1Program.array_B

    # Perform merge sort on both arrays
    mergeSort(arrayA, arrayB)


# Function which takes two arrays, merges them into one signle array and sorts
it
def mergeSort(arrayA, arrayB):

    # Array used to store all the elements of arrays A and B in a sorted
manner
    arrayC = []

    # Declare two variables used as the index points for arrays A and B
respectively
    i = j = 0

    # Declare two variables for the size of the two individual arrays
    sizeA = len(arrayA)
    sizeB = len(arrayB)

    # Compare the two elements pointed by the indexes i and j (for array A and
array B respectively)
    # Add the smallest element of the two into the new array C
    while (i < sizeA) and (j < sizeB):
        if arrayA[i] <= arrayB[j]:
            arrayC.append(arrayA[i])
            i += 1
        else:
            arrayC.append(arrayB[j])
            j += 1


    # Checking if any element was left after exitting first while loop
    while i < sizeA:
        arrayC.append(arrayA[i])
        i += 1
    while j < sizeB:
        arrayC.append(arrayB[j])
        j += 1

    # Print out contents of array C after sorting
    print("\nArray C:", arrayC)


# Call main() function to start execution
main()
```

This program begins execution by executing the previous program, where the user is asked to enter values for the individual sizes of both arrays A and B. Once it is executed successfully, the resultant arrays are retrieved and stored in new variables called arrayA and arrayB respectively. The current program will then call the mergeSort() function and pass on both arrays as arguments.

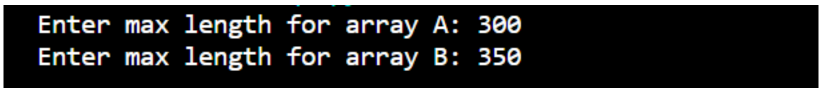
Once called, the function starts by declaring a new variable called arrayC which is initialised to have an empty list. Two counter variables called i and j are then declared and set to 0, which are used to traverse arrays A and B respectively. Afterwards, the program traverses both arrays and adds every possible element from both arrays in ascending order. When the program finishes traversing one whole array, it exits the first iteration and proceeds with the next iteration and adds the remaining elements accordingly. Once this whole process is finished, the final contents of array C are printed to the user.

❖ How the Program was Tested + Screen Dumps

Since this program starts by executing program1 first, the tests made from the previous program led to the same results in terms of output generated when different types of values are imputed when prompted. Thus, whenever the values entered by the user break at least one of the conditions mentioned in program 1, or if the user input is invalid as described previously, the resulting output was the same.

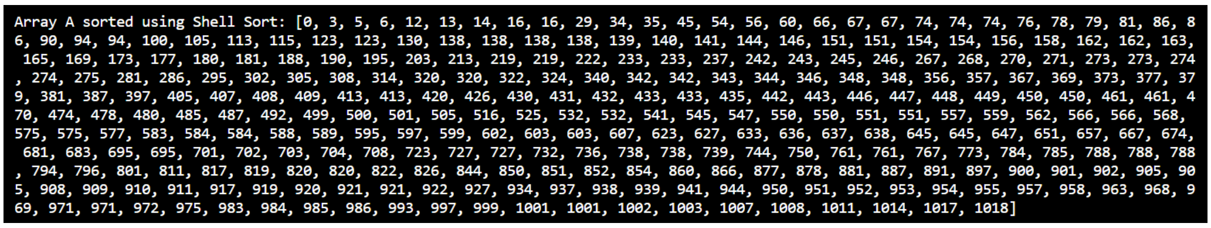
Apart from that, the program was tested by entering values that pass both conditions, but the first number is smaller than the second number. The numbers tested were 300 and 350. As expected, a message did not display stating that both values did not pass either condition or that the user input was invalid. Instead, the program printed out the sorted versions of arrays A and B and even displayed the final version of array C, which is done by merge sorting the first two arrays. As expected, array A had a smaller number of elements that array B. This is shown below:

Entering values



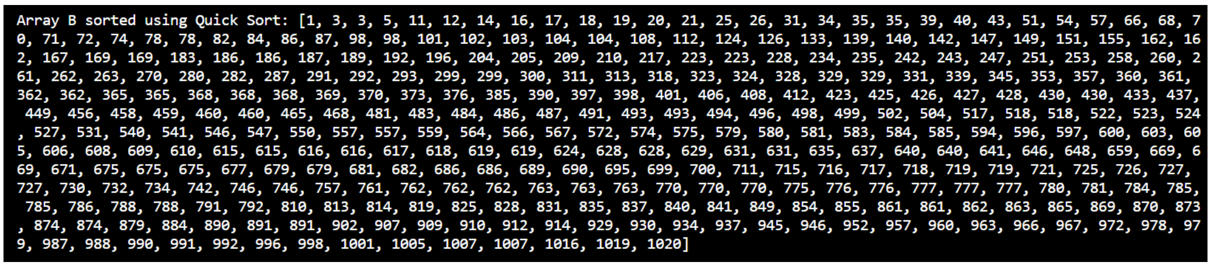
```
Enter max length for array A: 300
Enter max length for array B: 350
```

Final version of Array A displayed:



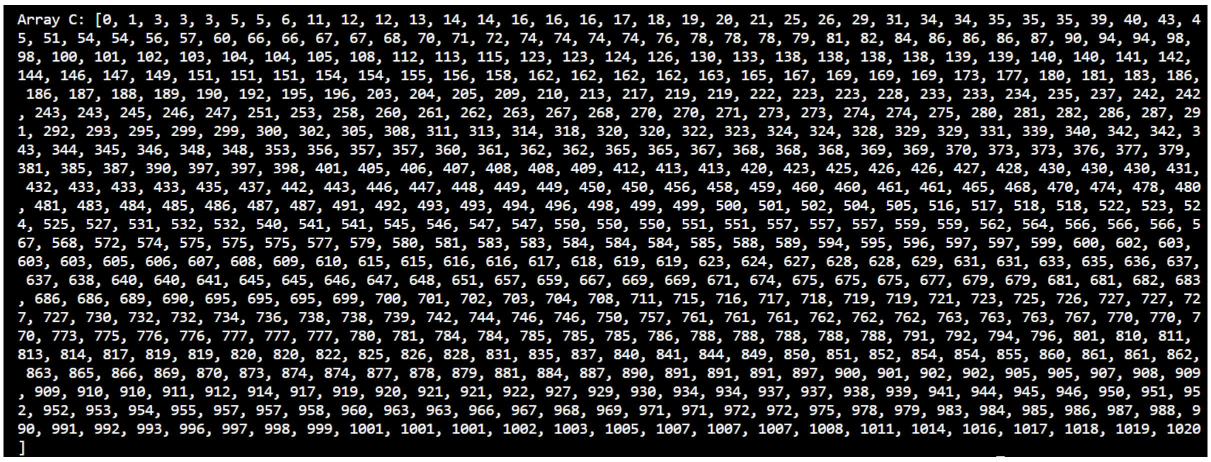
```
Array A sorted using Shell Sort: [0, 3, 5, 6, 12, 13, 14, 16, 16, 29, 34, 35, 45, 54, 56, 60, 66, 67, 67, 74, 74, 74, 76, 78, 79, 81, 86, 8
6, 90, 94, 94, 100, 105, 113, 115, 123, 123, 130, 138, 138, 138, 138, 139, 140, 141, 144, 146, 151, 151, 154, 154, 156, 158, 162, 162, 163,
 165, 169, 173, 177, 180, 181, 188, 190, 195, 203, 213, 219, 219, 222, 233, 233, 237, 242, 243, 245, 246, 267, 268, 270, 271, 273, 273, 274
, 274, 275, 281, 286, 295, 302, 305, 308, 314, 320, 320, 322, 324, 340, 342, 342, 343, 344, 346, 348, 348, 356, 357, 367, 369, 373, 377, 37
9, 381, 387, 397, 405, 407, 408, 409, 413, 413, 420, 426, 430, 431, 432, 433, 433, 435, 442, 443, 446, 447, 448, 449, 450, 450, 461, 461, 4
70, 474, 478, 480, 485, 487, 492, 499, 500, 501, 505, 516, 525, 532, 532, 541, 545, 547, 550, 550, 551, 551, 557, 559, 562, 566, 566, 568,
575, 575, 577, 583, 584, 584, 588, 589, 595, 597, 599, 602, 603, 603, 607, 623, 627, 633, 636, 637, 638, 645, 645, 647, 651, 657, 667, 674,
 681, 683, 695, 695, 701, 702, 703, 704, 708, 723, 727, 727, 732, 736, 738, 738, 739, 744, 750, 761, 761, 767, 773, 784, 785, 788, 788, 788
, 794, 796, 801, 811, 817, 819, 820, 820, 822, 826, 844, 850, 851, 852, 854, 860, 866, 877, 878, 881, 887, 891, 897, 900, 901, 902, 905, 90
5, 908, 909, 910, 911, 917, 919, 920, 921, 921, 922, 927, 934, 937, 938, 939, 941, 944, 950, 951, 952, 953, 954, 955, 957, 958, 963, 968, 9
69, 971, 971, 972, 975, 983, 984, 985, 986, 993, 997, 999, 1001, 1001, 1002, 1003, 1007, 1008, 1011, 1014, 1017, 1018]
```

Final version of Array B displayed:



```
Array B sorted using Quick Sort: [1, 3, 3, 5, 11, 12, 14, 16, 17, 18, 19, 20, 21, 25, 26, 31, 34, 35, 35, 39, 40, 43, 51, 54, 57, 66, 68, 7
0, 71, 72, 74, 78, 78, 82, 84, 86, 87, 98, 98, 101, 102, 103, 104, 104, 108, 112, 124, 126, 133, 139, 140, 142, 147, 149, 151, 155, 162, 16
2, 167, 169, 169, 183, 186, 186, 187, 189, 192, 196, 204, 205, 209, 210, 217, 223, 223, 228, 234, 235, 242, 243, 247, 251, 253, 258, 260, 2
61, 262, 263, 270, 280, 282, 287, 291, 292, 293, 299, 299, 300, 311, 313, 318, 323, 324, 328, 329, 329, 331, 339, 345, 353, 357, 360, 361,
362, 362, 365, 365, 368, 368, 368, 369, 370, 373, 376, 385, 390, 397, 398, 401, 406, 408, 412, 423, 425, 426, 427, 428, 430, 430, 433, 437,
 449, 456, 458, 459, 460, 460, 465, 468, 481, 483, 484, 486, 487, 491, 493, 493, 494, 496, 498, 499, 502, 504, 517, 518, 518, 522, 523, 524
, 527, 531, 540, 541, 546, 547, 550, 557, 557, 559, 564, 566, 567, 572, 574, 575, 579, 580, 581, 583, 584, 585, 594, 596, 597, 600, 603, 60
5, 606, 608, 609, 610, 615, 615, 616, 616, 617, 618, 619, 619, 624, 628, 628, 629, 631, 631, 635, 637, 640, 640, 641, 646, 648, 659, 669, 6
69, 671, 675, 675, 675, 677, 679, 679, 681, 682, 686, 686, 689, 690, 695, 699, 700, 711, 715, 716, 717, 718, 719, 719, 721, 725, 726, 727,
727, 730, 732, 734, 742, 746, 746, 757, 761, 762, 762, 762, 763, 763, 763, 770, 770, 770, 775, 776, 776, 777, 777, 777, 780, 781, 784, 785,
 785, 786, 788, 788, 791, 792, 810, 813, 814, 819, 825, 828, 831, 835, 837, 840, 841, 849, 854, 855, 861, 861, 862, 863, 865, 869, 870, 873
, 874, 874, 879, 884, 890, 891, 891, 902, 907, 909, 910, 912, 914, 929, 930, 934, 937, 945, 946, 952, 957, 960, 963, 966, 967, 972, 978, 97
9, 987, 988, 990, 991, 992, 996, 998, 1001, 1005, 1007, 1007, 1016, 1019, 1020]
```

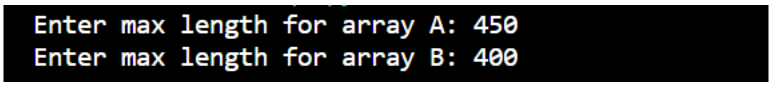
Final version of Array C displayed:



```
Array C: [0, 1, 3, 3, 3, 5, 5, 6, 11, 12, 12, 13, 14, 14, 16, 16, 16, 17, 18, 19, 20, 21, 25, 26, 29, 31, 34, 34, 35, 35, 35, 39, 40, 43, 4
5, 51, 54, 54, 56, 57, 60, 66, 66, 67, 67, 68, 70, 71, 72, 74, 74, 74, 74, 76, 78, 78, 78, 79, 81, 82, 84, 86, 86, 86, 87, 90, 94, 94, 98,
98, 100, 101, 102, 103, 104, 104, 105, 108, 112, 113, 115, 123, 123, 124, 126, 130, 133, 138, 138, 138, 138, 139, 139, 140, 140, 141, 142,
144, 146, 147, 149, 151, 151, 151, 154, 154, 155, 156, 158, 162, 162, 162, 162, 163, 165, 167, 169, 169, 169, 173, 177, 180, 181, 183, 186,
 186, 187, 188, 189, 190, 192, 195, 196, 203, 204, 205, 209, 210, 213, 217, 219, 219, 222, 223, 223, 228, 233, 233, 234, 235, 237, 242, 242
, 243, 243, 245, 246, 247, 251, 253, 258, 260, 261, 262, 263, 267, 268, 270, 270, 271, 273, 273, 274, 274, 275, 280, 281, 282, 286, 287, 29
1, 292, 293, 295, 299, 299, 300, 302, 305, 308, 311, 313, 314, 318, 320, 320, 322, 323, 324, 324, 328, 329, 329, 331, 339, 340, 342, 342, 3
43, 344, 345, 346, 348, 348, 353, 356, 357, 357, 360, 361, 362, 362, 365, 365, 367, 368, 368, 368, 369, 369, 370, 373, 373, 376, 377, 379,
381, 385, 387, 390, 397, 397, 398, 401, 405, 406, 407, 408, 408, 409, 412, 413, 413, 420, 423, 425, 426, 426, 427, 428, 430, 430, 430, 431,
 432, 433, 433, 433, 435, 437, 442, 443, 446, 447, 448, 449, 449, 450, 450, 456, 458, 459, 460, 460, 461, 461, 465, 468, 470, 474, 478, 480
, 481, 483, 484, 485, 486, 487, 487, 491, 492, 493, 493, 494, 496, 498, 499, 499, 500, 501, 502, 504, 505, 516, 517, 518, 518, 522, 523, 52
4, 525, 527, 531, 532, 532, 540, 541, 541, 545, 546, 547, 547, 550, 550, 550, 551, 551, 557, 557, 557, 559, 559, 562, 564, 566, 566, 566, 5
67, 568, 572, 574, 575, 575, 575, 577, 579, 580, 581, 583, 583, 584, 584, 584, 585, 588, 589, 594, 595, 596, 597, 597, 599, 600, 602, 603,
603, 603, 605, 606, 607, 608, 609, 610, 615, 615, 616, 616, 617, 618, 619, 619, 623, 624, 627, 628, 628, 629, 631, 631, 633, 635, 636, 637,
 637, 638, 640, 640, 641, 645, 645, 646, 647, 648, 651, 657, 659, 667, 669, 669, 671, 674, 675, 675, 675, 677, 679, 679, 681, 681, 682, 683
, 686, 686, 689, 690, 695, 695, 695, 699, 700, 701, 702, 703, 704, 708, 711, 715, 716, 717, 718, 719, 719, 721, 723, 725, 726, 727, 727, 72
7, 727, 730, 732, 732, 734, 736, 738, 738, 739, 742, 744, 746, 746, 750, 757, 761, 761, 761, 762, 762, 762, 763, 763, 763, 767, 770, 770, 7
70, 773, 775, 776, 776, 777, 777, 777, 780, 781, 784, 784, 785, 785, 785, 786, 788, 788, 788, 788, 788, 791, 792, 794, 796, 801, 810, 811,
813, 814, 817, 819, 819, 820, 820, 822, 825, 826, 828, 831, 835, 837, 840, 841, 844, 849, 850, 851, 852, 854, 854, 855, 860, 861, 861, 862,
 863, 865, 866, 869, 870, 873, 874, 874, 877, 878, 879, 881, 884, 887, 890, 891, 891, 891, 897, 900, 901, 902, 902, 905, 905, 907, 908, 909
, 909, 910, 910, 911, 912, 914, 917, 919, 920, 921, 921, 922, 927, 929, 930, 934, 934, 937, 937, 938, 939, 941, 944, 945, 946, 950, 951, 95
2, 952, 953, 954, 955, 957, 957, 958, 960, 963, 963, 966, 967, 968, 969, 971, 971, 972, 972, 975, 978, 979, 983, 984, 985, 986, 987, 988, 9
90, 991, 992, 993, 996, 997, 998, 999, 1001, 1001, 1001, 1002, 1003, 1005, 1007, 1007, 1007, 1008, 1011, 1014, 1016, 1017, 1018, 1019, 1020
]
```

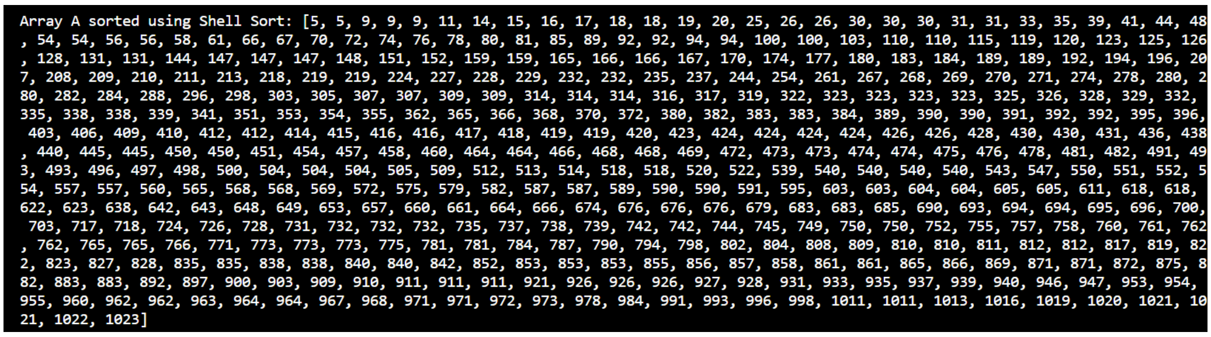
Afterwards, the program was tested by again entering values that pass both conditions, but this time, the first number is greater than the second number. The numbers tested were 450 and 400. As expected, a message did not display stating that both values did not pass either condition. The program also printed out the sorted versions of arrays A and B after executing their respective sorting algorithms. As expected, array A had a greater number of elements that array B. This is shown below:

Entering values



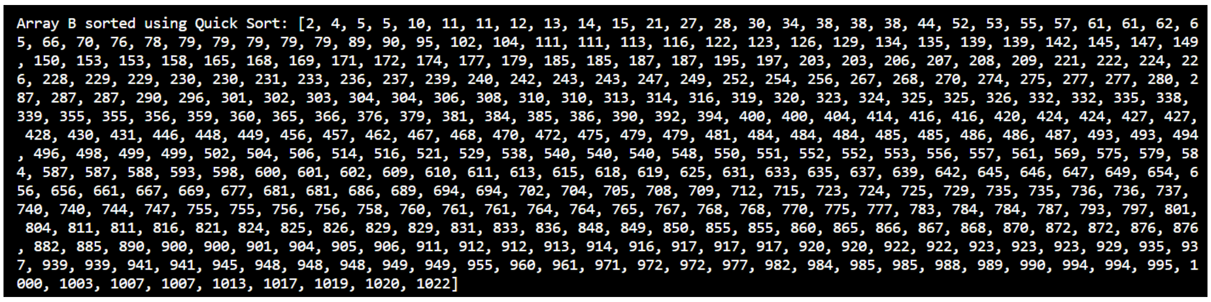
```
Enter max length for array A: 450
Enter max length for array B: 400
```

Final version of Array A displayed:



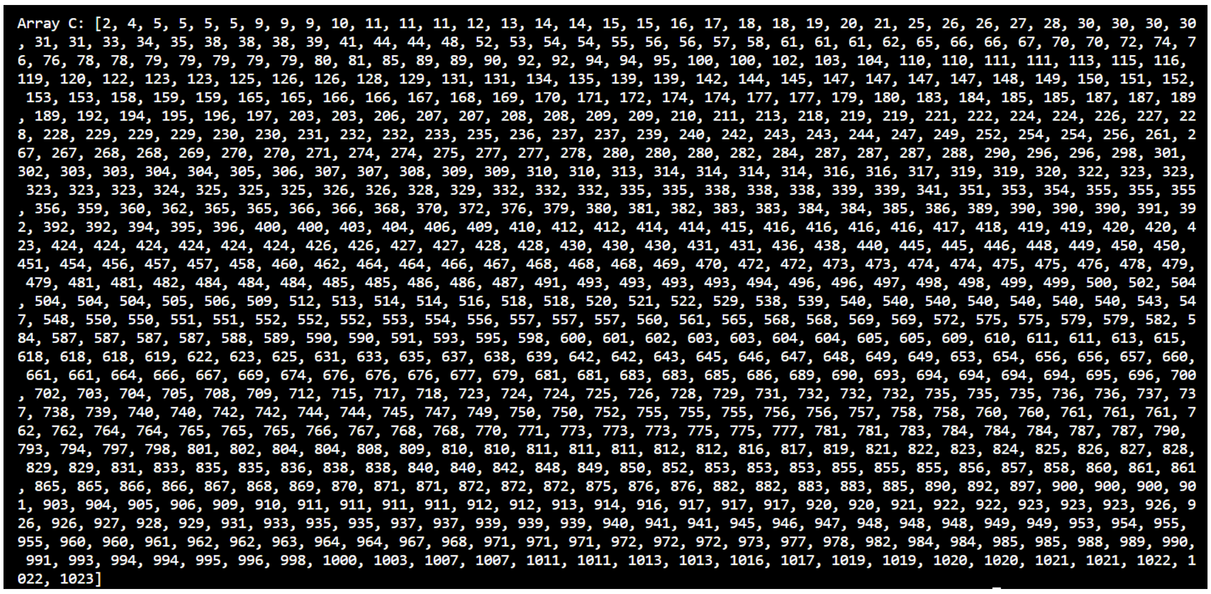
```
Array A sorted using Shell Sort: [5, 5, 9, 9, 9, 11, 14, 15, 16, 17, 18, 18, 19, 20, 25, 26, 26, 30, 30, 30, 31, 31, 33, 35, 39, 41, 44, 48
, 54, 54, 56, 56, 58, 61, 66, 67, 70, 72, 74, 76, 78, 80, 81, 85, 89, 92, 92, 94, 94, 100, 100, 103, 110, 110, 115, 119, 120, 123, 125, 126
, 128, 131, 131, 144, 147, 147, 147, 148, 151, 152, 159, 159, 165, 166, 166, 167, 170, 174, 177, 180, 183, 184, 189, 189, 192, 194, 196, 20
7, 208, 209, 210, 211, 213, 218, 219, 219, 224, 227, 228, 229, 232, 232, 235, 237, 244, 254, 261, 267, 268, 269, 270, 271, 274, 278, 280, 2
80, 282, 284, 288, 296, 298, 303, 305, 307, 307, 309, 309, 314, 314, 314, 316, 317, 319, 322, 323, 323, 323, 323, 325, 326, 328, 329, 332,
335, 338, 338, 339, 341, 351, 353, 354, 355, 362, 365, 366, 368, 370, 372, 380, 382, 383, 383, 384, 389, 390, 390, 391, 392, 392, 395, 396,
 403, 406, 409, 410, 412, 412, 414, 415, 416, 416, 417, 418, 419, 419, 420, 423, 424, 424, 424, 424, 426, 426, 428, 430, 430, 431, 436, 438
, 440, 445, 445, 450, 450, 451, 454, 457, 458, 460, 464, 464, 466, 468, 468, 469, 472, 473, 473, 474, 474, 475, 476, 478, 481, 482, 491, 49
3, 493, 496, 497, 498, 500, 504, 504, 504, 505, 509, 512, 513, 514, 518, 518, 520, 522, 539, 540, 540, 540, 540, 543, 547, 550, 551, 552, 5
54, 557, 557, 560, 565, 568, 568, 569, 572, 575, 579, 582, 587, 587, 589, 590, 590, 591, 595, 603, 603, 604, 604, 605, 605, 611, 618, 618,
622, 623, 638, 642, 643, 648, 649, 653, 657, 660, 661, 664, 666, 674, 676, 676, 676, 679, 683, 683, 685, 690, 693, 694, 694, 695, 696, 700,
 703, 717, 718, 724, 726, 728, 731, 732, 732, 732, 735, 737, 738, 739, 742, 742, 744, 745, 749, 750, 750, 752, 755, 757, 758, 760, 761, 762
, 762, 765, 765, 766, 771, 773, 773, 773, 775, 781, 781, 784, 787, 790, 794, 798, 802, 804, 808, 809, 810, 810, 811, 812, 812, 817, 819, 82
2, 823, 827, 828, 835, 835, 838, 838, 840, 840, 842, 852, 853, 853, 853, 855, 856, 857, 858, 861, 861, 865, 866, 869, 871, 871, 872, 875, 8
82, 883, 883, 892, 897, 900, 903, 909, 910, 911, 911, 911, 921, 926, 926, 926, 927, 928, 931, 933, 935, 937, 939, 940, 946, 947, 953, 954,
955, 960, 962, 962, 963, 964, 964, 967, 968, 971, 971, 972, 973, 978, 984, 991, 993, 996, 998, 1011, 1011, 1013, 1016, 1019, 1020, 1021, 10
21, 1022, 1023]
```

<u>Final version of Array B displayed:</u>



```
Array B sorted using Quick Sort: [2, 4, 5, 5, 10, 11, 11, 12, 13, 14, 15, 21, 27, 28, 30, 34, 38, 38, 38, 44, 52, 53, 55, 57, 61, 61, 62, 6
5, 66, 70, 76, 78, 79, 79, 79, 79, 79, 89, 90, 95, 102, 104, 111, 111, 113, 116, 122, 123, 126, 129, 134, 135, 139, 139, 142, 145, 147, 149
, 150, 153, 153, 158, 165, 168, 169, 171, 172, 174, 177, 179, 185, 185, 187, 187, 195, 197, 203, 203, 206, 207, 208, 209, 221, 222, 224, 22
6, 228, 229, 229, 230, 230, 231, 233, 236, 237, 239, 240, 242, 243, 243, 247, 249, 252, 254, 256, 267, 268, 270, 274, 275, 277, 277, 280, 2
87, 287, 287, 290, 296, 301, 302, 303, 304, 304, 306, 308, 310, 310, 313, 314, 316, 319, 320, 323, 324, 325, 325, 326, 332, 332, 335, 338,
339, 355, 355, 356, 359, 360, 365, 366, 376, 379, 381, 384, 385, 386, 390, 392, 394, 400, 400, 404, 414, 416, 416, 420, 424, 424, 427, 427,
 428, 430, 431, 446, 448, 449, 456, 457, 462, 467, 468, 470, 472, 475, 479, 479, 481, 484, 484, 484, 485, 485, 486, 486, 487, 493, 493, 494
, 496, 498, 499, 499, 502, 504, 506, 514, 516, 521, 529, 538, 540, 540, 540, 548, 550, 551, 552, 552, 553, 556, 557, 561, 569, 575, 579, 58
4, 587, 587, 588, 593, 598, 600, 601, 602, 609, 610, 611, 613, 615, 618, 619, 625, 631, 633, 635, 637, 639, 642, 645, 646, 647, 649, 654, 6
56, 656, 661, 667, 669, 677, 681, 681, 686, 689, 694, 694, 702, 704, 705, 708, 709, 712, 715, 723, 724, 725, 729, 735, 735, 736, 736, 737,
740, 740, 744, 747, 755, 755, 756, 756, 758, 760, 761, 761, 764, 764, 765, 767, 768, 768, 770, 775, 777, 783, 784, 784, 787, 793, 797, 801,
 804, 811, 811, 816, 821, 824, 825, 826, 829, 829, 831, 833, 836, 848, 849, 850, 855, 855, 860, 865, 866, 867, 868, 870, 872, 872, 876, 876
, 882, 885, 890, 900, 900, 901, 904, 905, 906, 911, 912, 912, 913, 914, 916, 917, 917, 917, 920, 920, 922, 922, 923, 923, 923, 929, 935, 93
7, 939, 939, 941, 941, 945, 948, 948, 948, 949, 949, 955, 960, 961, 971, 972, 972, 977, 982, 984, 985, 985, 988, 989, 990, 994, 994, 995, 1
000, 1003, 1007, 1007, 1013, 1017, 1019, 1020, 1022]
```

<u>Final version of Array C displayed:</u>



```
Array C: [2, 4, 5, 5, 5, 5, 9, 9, 9, 10, 11, 11, 11, 12, 13, 14, 14, 15, 15, 16, 17, 18, 18, 19, 20, 21, 25, 26, 26, 27, 28, 30, 30, 30, 30
, 31, 31, 33, 34, 35, 38, 38, 38, 39, 41, 44, 44, 48, 52, 53, 54, 54, 55, 56, 56, 57, 58, 61, 61, 61, 62, 65, 66, 66, 67, 70, 70, 72, 74, 7
6, 76, 78, 78, 79, 79, 79, 79, 79, 80, 81, 85, 89, 89, 90, 92, 92, 94, 94, 95, 100, 100, 102, 103, 104, 110, 110, 111, 111, 113, 115, 116,
119, 120, 122, 123, 123, 125, 126, 126, 128, 129, 131, 131, 134, 135, 139, 139, 142, 144, 145, 147, 147, 147, 147, 148, 149, 150, 151, 152,
 153, 153, 158, 159, 159, 165, 165, 166, 166, 167, 168, 169, 170, 171, 172, 174, 174, 177, 177, 179, 180, 183, 184, 185, 185, 187, 187, 189
, 189, 192, 194, 195, 196, 197, 203, 203, 206, 207, 207, 208, 208, 209, 209, 210, 211, 213, 218, 219, 219, 221, 222, 224, 224, 226, 227, 22
8, 228, 229, 229, 229, 230, 230, 231, 232, 232, 233, 235, 236, 237, 237, 239, 240, 242, 243, 243, 244, 247, 249, 252, 254, 254, 256, 261, 2
67, 267, 268, 268, 269, 270, 270, 271, 274, 274, 275, 277, 277, 278, 280, 280, 280, 282, 284, 287, 287, 287, 288, 290, 296, 296, 298, 301,
302, 303, 303, 304, 304, 305, 306, 307, 307, 308, 309, 309, 310, 310, 313, 314, 314, 314, 314, 316, 316, 317, 319, 319, 320, 322, 323, 323,
 323, 323, 323, 324, 325, 325, 325, 326, 326, 328, 329, 332, 332, 332, 335, 335, 338, 338, 338, 339, 339, 341, 351, 353, 354, 355, 355, 355
, 356, 359, 360, 362, 365, 365, 366, 366, 368, 370, 372, 376, 379, 380, 381, 382, 383, 383, 384, 384, 385, 386, 389, 390, 390, 390, 391, 39
2, 392, 392, 394, 395, 396, 400, 400, 403, 404, 406, 409, 410, 412, 412, 414, 414, 415, 416, 416, 416, 416, 417, 418, 419, 419, 420, 420, 4
23, 424, 424, 424, 424, 424, 424, 426, 426, 427, 427, 428, 428, 430, 430, 430, 431, 431, 436, 438, 440, 445, 445, 446, 448, 449, 450, 450,
451, 454, 456, 457, 457, 458, 460, 462, 464, 464, 466, 467, 468, 468, 468, 469, 470, 472, 472, 473, 473, 474, 474, 475, 475, 476, 478, 479,
 479, 481, 481, 482, 484, 484, 484, 485, 485, 486, 486, 487, 491, 493, 493, 493, 493, 494, 496, 496, 497, 498, 498, 499, 499, 500, 502, 504
, 504, 504, 504, 505, 506, 509, 512, 513, 514, 514, 516, 518, 518, 520, 521, 522, 529, 538, 539, 540, 540, 540, 540, 540, 540, 540, 543, 54
7, 548, 550, 550, 551, 551, 552, 552, 552, 553, 554, 556, 557, 557, 557, 560, 561, 565, 568, 568, 569, 569, 572, 575, 575, 579, 579, 582, 5
84, 587, 587, 587, 587, 588, 589, 590, 590, 591, 593, 595, 598, 600, 601, 602, 603, 603, 604, 604, 605, 605, 609, 610, 611, 611, 613, 615,
618, 618, 618, 619, 622, 623, 625, 631, 633, 635, 637, 638, 639, 642, 642, 643, 645, 646, 647, 648, 649, 649, 653, 654, 656, 656, 657, 660,
 661, 661, 664, 666, 667, 669, 674, 676, 676, 676, 677, 679, 681, 681, 683, 683, 685, 686, 689, 690, 693, 694, 694, 694, 694, 695, 696, 700
, 702, 703, 704, 705, 708, 709, 712, 715, 717, 718, 723, 724, 724, 725, 726, 728, 729, 731, 732, 732, 732, 735, 735, 735, 736, 736, 737, 73
7, 738, 739, 740, 740, 742, 742, 744, 744, 745, 747, 749, 750, 750, 752, 755, 755, 755, 756, 756, 757, 758, 758, 760, 760, 761, 761, 761, 7
62, 762, 764, 764, 765, 765, 765, 766, 767, 768, 768, 770, 771, 773, 773, 773, 775, 775, 777, 781, 781, 783, 784, 784, 784, 787, 787, 790,
793, 794, 797, 798, 801, 802, 804, 804, 808, 809, 810, 810, 811, 811, 811, 812, 812, 816, 817, 819, 821, 822, 823, 824, 825, 826, 827, 828,
 829, 829, 831, 833, 835, 835, 836, 838, 838, 840, 840, 842, 848, 849, 850, 852, 853, 853, 853, 855, 855, 855, 856, 857, 858, 860, 861, 861
, 865, 865, 866, 866, 867, 868, 869, 870, 871, 871, 872, 872, 872, 875, 876, 876, 882, 882, 883, 883, 885, 890, 892, 897, 900, 900, 900, 90
1, 903, 904, 905, 906, 909, 910, 911, 911, 911, 911, 912, 912, 913, 914, 916, 917, 917, 917, 920, 920, 921, 922, 922, 923, 923, 923, 926, 9
26, 926, 927, 928, 929, 931, 933, 935, 935, 937, 937, 939, 939, 939, 940, 941, 941, 945, 946, 947, 948, 948, 948, 949, 949, 953, 954, 955,
955, 960, 960, 961, 962, 962, 963, 964, 964, 967, 968, 971, 971, 971, 972, 972, 972, 973, 977, 978, 982, 984, 984, 985, 985, 988, 989, 990,
 991, 993, 994, 994, 995, 996, 998, 1000, 1003, 1007, 1007, 1011, 1011, 1013, 1013, 1016, 1017, 1019, 1019, 1020, 1020, 1021, 1021, 1022, 1
022, 1023]
```

- Question 3
    - Source Code & Explanation

```
def extreme_points():
    # An array containing list of numbers
    array = []

    # An empty array which is used to add any extreme points
    extremePointsArray = []

    # Enter numbers until user decides to stop
    has_stopped = False

    while not has_stopped:
        # Ask user to input an integer
        num = input("Enter an integer: ")

        # Check if user input is an integer
        if num.lstrip('-+').isnumeric():    # If True, add number to list
            array.append(int(num))

            # Check for valid option
            valid_choice = False

            while not valid_choice:
                # Ask user to decide whether to continue
                choice = input("Do you want to continue adding more numbers
(Y/N)?\n")

                # Check if user entered a value before pressing Enter
                if len(choice) > 0:
                    if choice[0].capitalize() == 'Y':   # valid option,
proceeds with entering next number
                        has_stopped = False
                        valid_choice = True
                    elif choice[0].capitalize() == 'N': # valid option, exits
entire iteration
                        has_stopped = True
                        valid_choice = True
                    else:   # invalid option, asks user to enter a valid
option
                        valid_choice = False
                        print("Please enter Y or N (Yes or No)!")
                else:
                    valid_choice = False
                    print("Please enter a value!")
        else:   # Number not an integer!
```

```
            print('Invalid input! Please enter an integer!')

    # Setting variable length to the number of elements in the variable array
    length = len(array)

    # Go through each element by index and check whether it is an extreme
point
    for index in range(length):
        # Check whether the current element is not the first or the last
element in the list
        if not (index == 0 or index == (length-1)):
            currentElement = array[index]

            # Check whether the current element is an extreme point with the
conditions set below
            # If True, then add the element to the extremePointsArray list
            if (array[index-1] > array[index] and array[index] <
array[index+1]) or (array[index-1] < array[index] and array[index] >
array[index+1]):
                extremePointsArray.append(currentElement)

    # Detects if there are any values in extremePointsArray after iteration of
original list
    if not extremePointsArray:  # If empty, output to the user that original
list was sorted
        print("SORTED")
    else:   # If not empty, print out all the elements from within the
extremePointsArray variable
        print(extremePointsArray)


# Call extreme_points() function to start execution
extreme_points()
```

The program starts execution by declaring two list variables, one which stores the numbers entered by the user and another which stores the extreme points of the first array. It then asks the user to enter a number to be added in the first array. The program checks if the value entered by the user is an integer. If successful, the number is added to the array the user is then prompted to enter whether he/she wants to add any more numbers in the array. The program first checks whether the user enters something in the console before pressing Enter. If that is successful, it then checks whether the first character of the user input is either a ‘Y’ or an ‘N’. If the first character is a ‘Y’, the program redirects the user to enter another number. Otherwise, if it is an ‘N’, the iteration stops and proceeds with the rest of execution. If all checks are unsuccessful, the user is redirected accordingly.

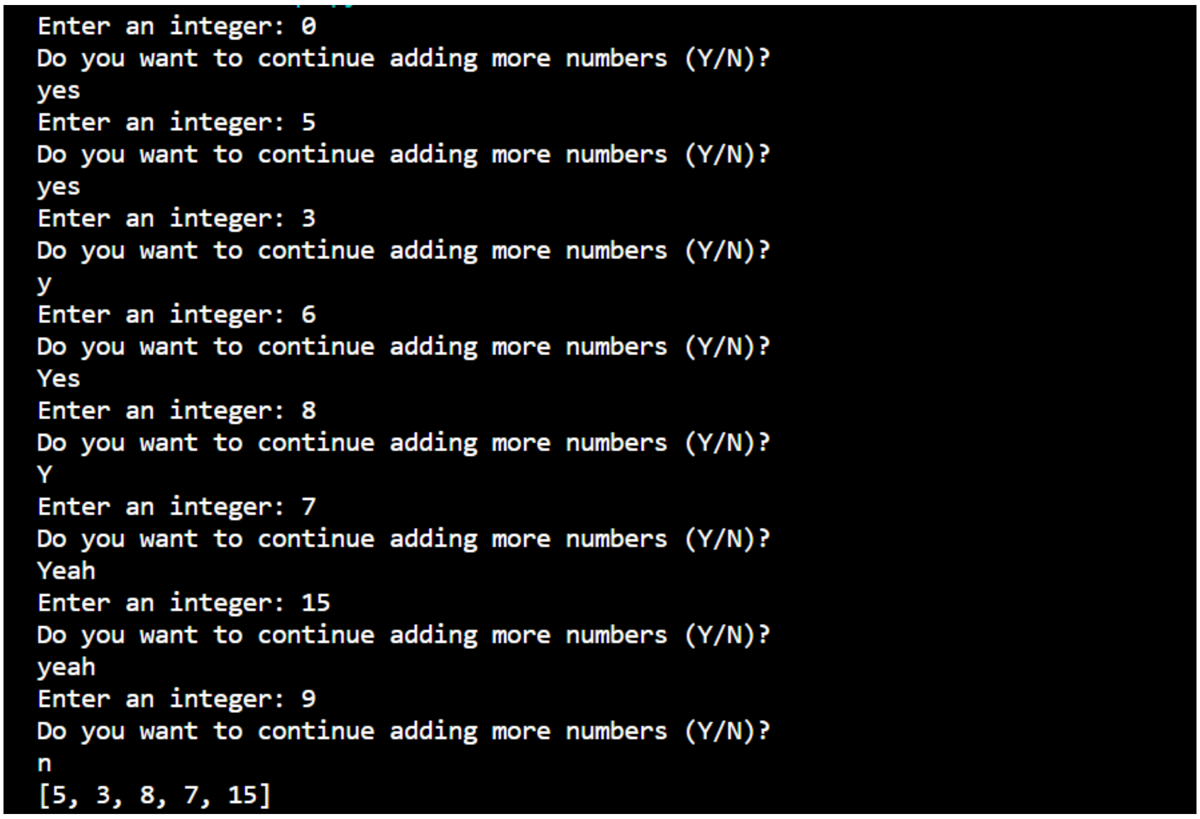
After all the numbers are entered by the user, the program tries to find the extreme points of the array. Any extreme points that are found are added to the extremePointsArray list. Once completed, if there were any extreme points found, the contents of the extremePointsArray are printed to the user. Otherwise, the message “SORTED” gets displayed on the console.

Generally, an array has no extreme points if and only if it is sorted. This is because if the array is sorted, there are no elements that are smaller than the previous element and larger than the next element and vice-versa.

❖ How the Program was Tested + Screen Dumps

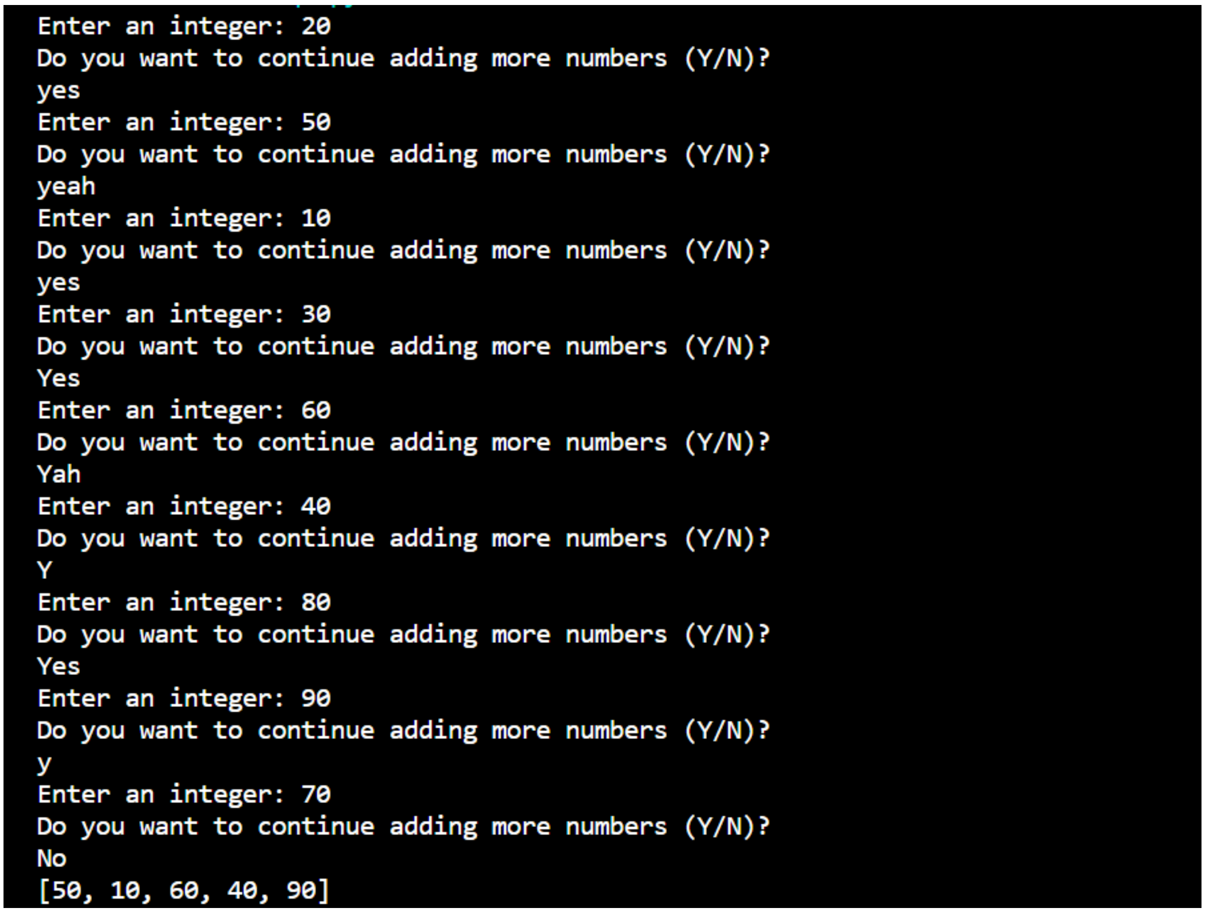
The program was first tested by letting the user enter two different sets of numbers such that they are not sorted. The two lists of valid numbers that were tested were [0, 5, 3, 6, 8, 7, 15, 9] and [20, 50, 10, 30, 60, 40, 80, 90, 70]. As expected, the program detected that the two lists were not sorted, and the result was that a list was displayed with all the extreme values. This can be seen below as follows:

List 1: Inputting values and Answer Output



```
Enter an integer: 0
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: 5
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: 3
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 6
Do you want to continue adding more numbers (Y/N)?
Yes
Enter an integer: 8
Do you want to continue adding more numbers (Y/N)?
Y
Enter an integer: 7
Do you want to continue adding more numbers (Y/N)?
Yeah
Enter an integer: 15
Do you want to continue adding more numbers (Y/N)?
yeah
Enter an integer: 9
Do you want to continue adding more numbers (Y/N)?
n
[5, 3, 8, 7, 15]
```

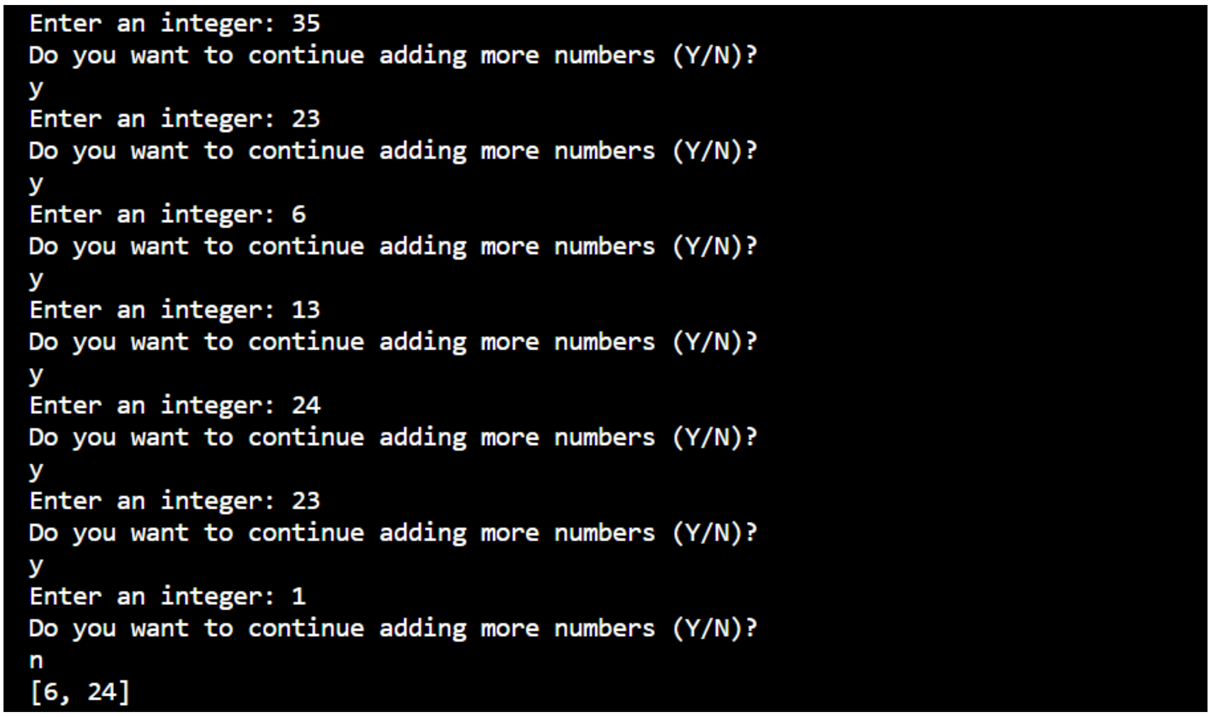
List 2: Inputting values and Answer Output



```
Enter an integer: 20
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: 50
Do you want to continue adding more numbers (Y/N)?
yeah
Enter an integer: 10
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: 30
Do you want to continue adding more numbers (Y/N)?
Yes
Enter an integer: 60
Do you want to continue adding more numbers (Y/N)?
Yah
Enter an integer: 40
Do you want to continue adding more numbers (Y/N)?
Y
Enter an integer: 80
Do you want to continue adding more numbers (Y/N)?
Yes
Enter an integer: 90
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 70
Do you want to continue adding more numbers (Y/N)?
No
[50, 10, 60, 40, 90]
```

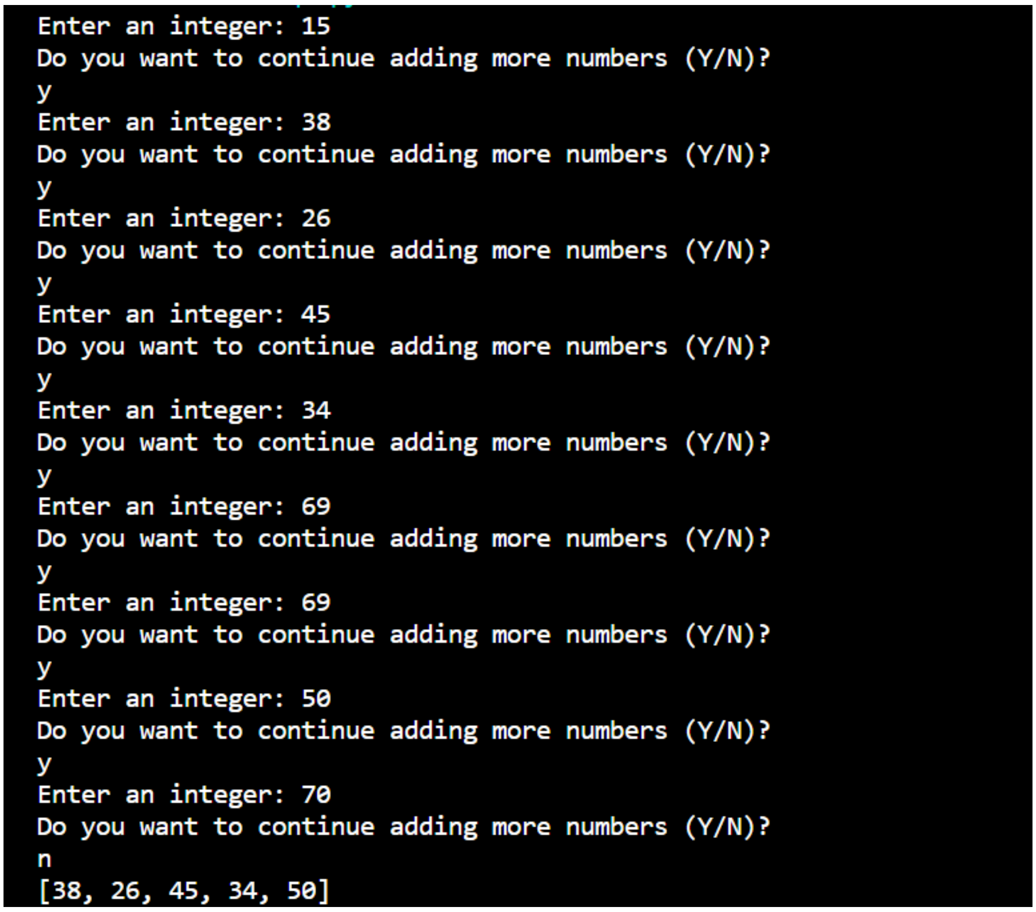
The second test was done to check for any effects in the case that integers are repeated in the list. Once again, two different lists of numbers were tested, one of which does not have any repeated integers next to each other, while the other one does. The end result was, for the first list, it functions as if it is a normal unsorted list. On the other hand, for the second list, the program does not print the repeated integer/s because the conditions for extreme points are not met. This can be seen below as follows:

List 1: Inputting values and Answer Output



```
Enter an integer: 35
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 23
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 6
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 13
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 24
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 23
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 1
Do you want to continue adding more numbers (Y/N)?
n
[6, 24]
```

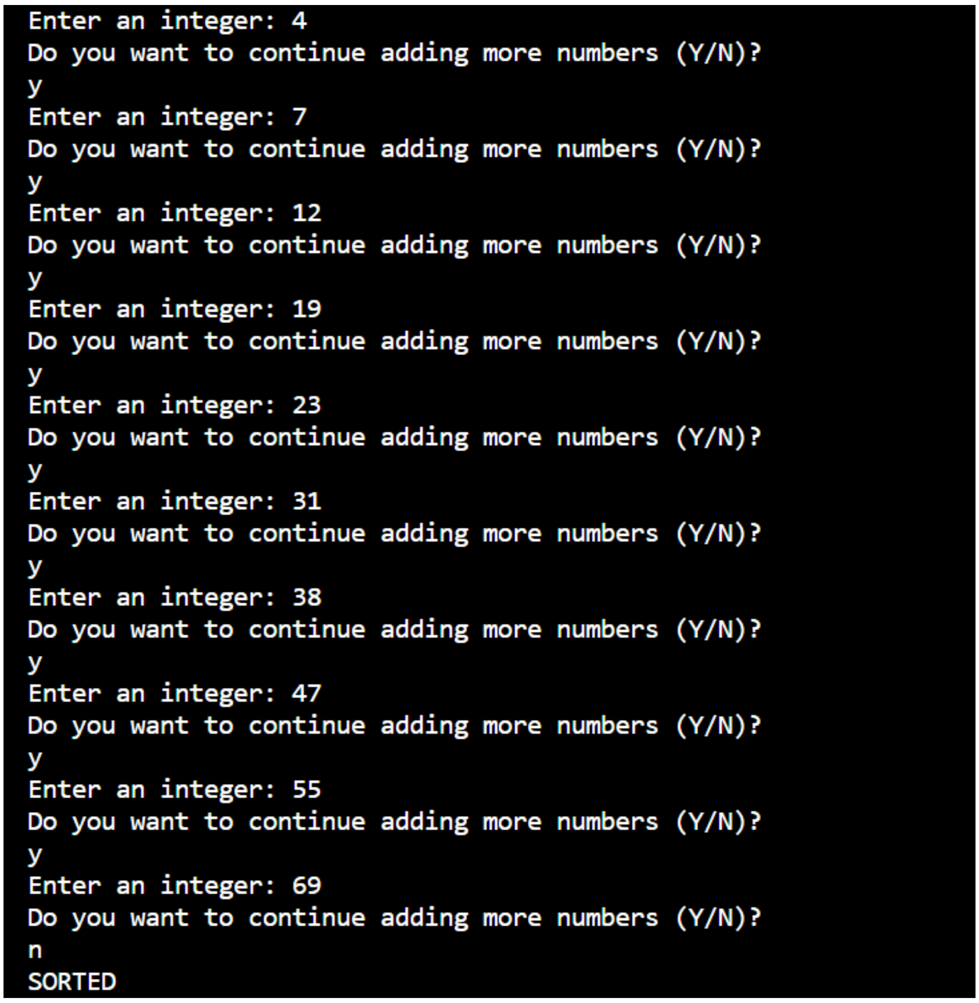
List 2: Inputting values and Answer Output



```
Enter an integer: 15
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 38
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 26
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 45
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 34
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 69
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 69
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 50
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 70
Do you want to continue adding more numbers (Y/N)?
n
[38, 26, 45, 34, 50]
```

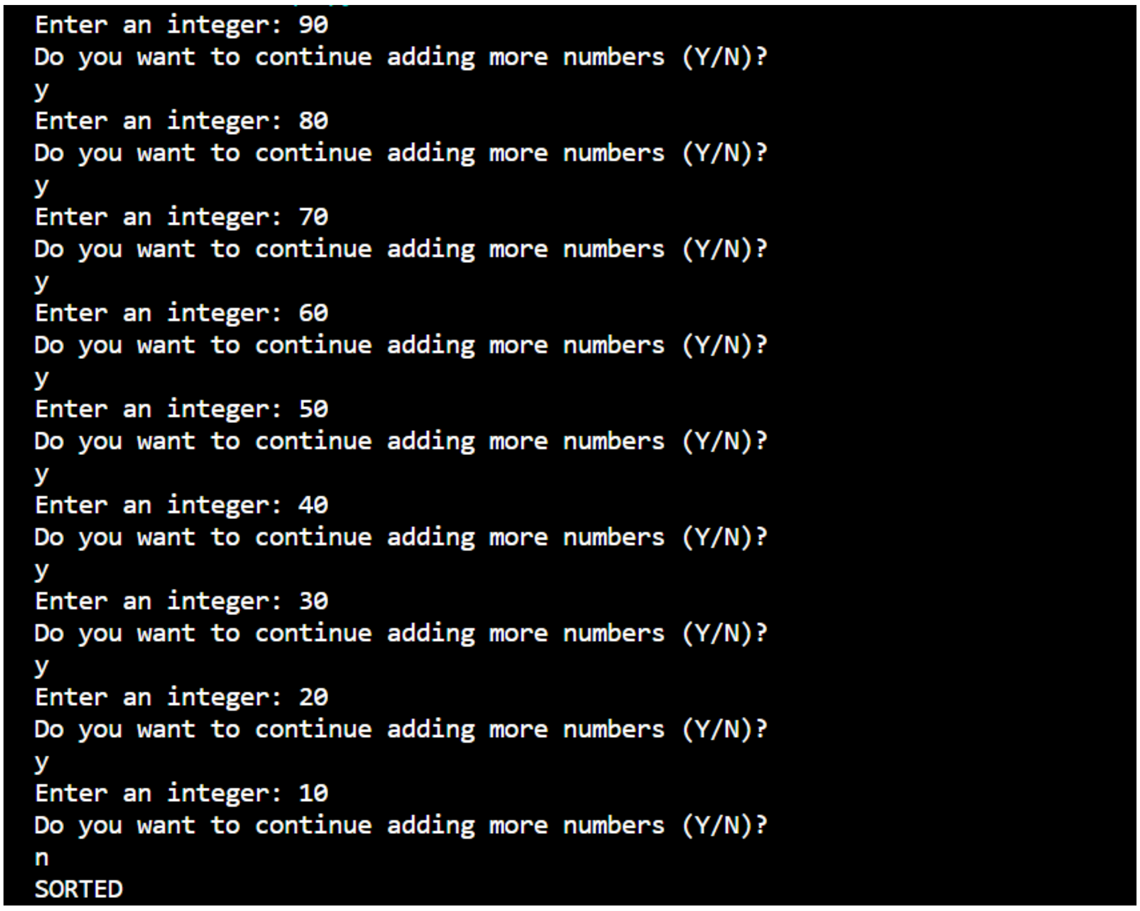
The program was then tested for a third time by entering two different sets of numbers such that they are sorted. One of the lists was sorted in ascending order whilst the other one was sorted in descending order. The two lists that were tested were [4, 7, 12, 19, 23, 31, 38, 47, 55, 69] and [90, 80, 70, 60, 50, 40, 30, 20, 10]. As expected, none of the numbers in both lists meet any of the two conditions for extreme points. Thus a message displayed as 'SORTED' was printed out. This can be seen below:

List 1: Inputting values and Answer Output



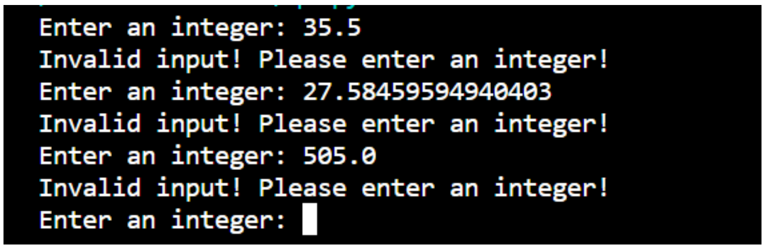
```
Enter an integer: 4
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 7
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 12
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 19
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 23
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 31
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 38
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 47
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 55
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 69
Do you want to continue adding more numbers (Y/N)?
n
SORTED
```

List 2: Inputting values and Answer Output



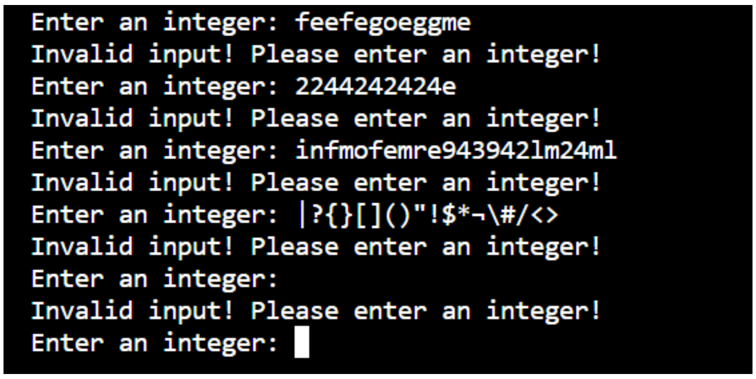
```
Enter an integer: 90
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 80
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 70
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 60
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 50
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 40
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 30
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 20
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 10
Do you want to continue adding more numbers (Y/N)?
n
SORTED
```

Afterwards, the program was tested by inputting a set of float numbers. This was done to test the program's ability to distinguish an integer from a float. As expected, the program saw this user input as invalid and thus, a message was printed out stating that such input is invalid and also prompting the user to enter an integer. This is shown below:



```
Enter an integer: 35.5
Invalid input! Please enter an integer!
Enter an integer: 27.58459594940403
Invalid input! Please enter an integer!
Enter an integer: 505.0
Invalid input! Please enter an integer!
Enter an integer: 
```

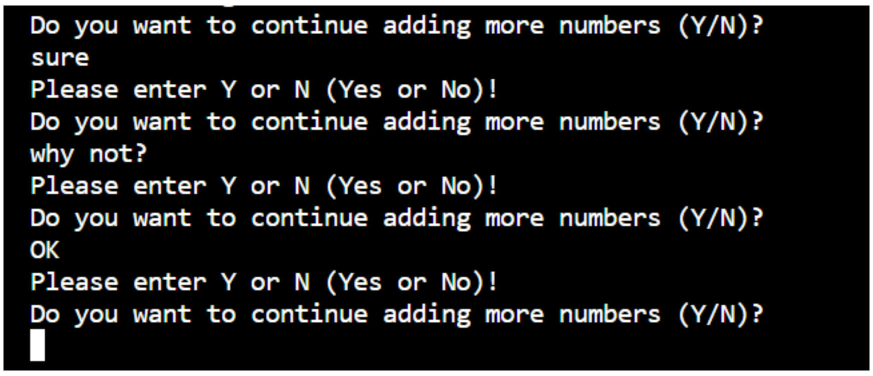
Next, the program was tested by again restricting user input, this time only inputting string values. The program also saw this user input as invalid, as expected. Thus, a message was printed out stating that such input is invalid and also prompted the user to enter an integer. This is shown below:



```
Enter an integer: feefegoeggme
Invalid input! Please enter an integer!
Enter an integer: 2244242424e
Invalid input! Please enter an integer!
Enter an integer: infmofemre9439421m24ml
Invalid input! Please enter an integer!
Enter an integer: |?{}[]()"!$*¬\#/<>
Invalid input! Please enter an integer!
Enter an integer:
Invalid input! Please enter an integer!
Enter an integer: █
```

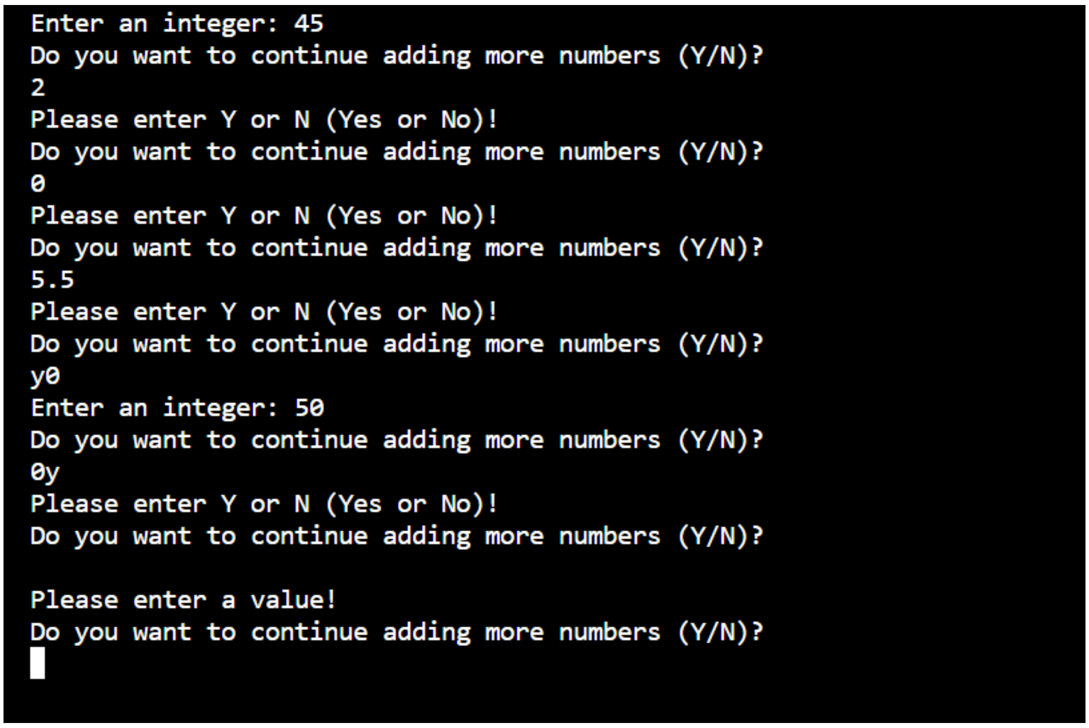
The next tests that were done were made to test the user input whenever the user is asked to choose whether to enter add items to the list or not. Only invalid user input was tested since valid user input was tested previously. For all these tests, the number to be added was valid.

The first test for this part was done by entering a string whose first character is not ‘Y’ or ‘N’, as whole strings are allowed in this program. As expected, the program detected that the user input was invalid and a message was printed out telling the user to enter either ‘Y’ or ‘N’. This can be seen below:



```
Do you want to continue adding more numbers (Y/N)?
sure
Please enter Y or N (Yes or No)!
Do you want to continue adding more numbers (Y/N)?
why not?
Please enter Y or N (Yes or No)!
Do you want to continue adding more numbers (Y/N)?
OK
Please enter Y or N (Yes or No)!
Do you want to continue adding more numbers (Y/N)?
█
```

The second test was done by entering any value of any other data type, such as floats or integers. Even strings which contained numbers or with nothing entered were tested. For almost all inputs, the program saw this user input as invalid and the appropriate message was printed out. However, when the string “y0” was entered, the program saw this as valid, since only the first character is dealt with in the program. As a result, the program redirected the user to input another number to add to the list. This is shown below:



```
Enter an integer: 45
Do you want to continue adding more numbers (Y/N)?
2
Please enter Y or N (Yes or No)!
Do you want to continue adding more numbers (Y/N)?
0
Please enter Y or N (Yes or No)!
Do you want to continue adding more numbers (Y/N)?
5.5
Please enter Y or N (Yes or No)!
Do you want to continue adding more numbers (Y/N)?
y0
Enter an integer: 50
Do you want to continue adding more numbers (Y/N)?
0y
Please enter Y or N (Yes or No)!
Do you want to continue adding more numbers (Y/N)?

Please enter a value!
Do you want to continue adding more numbers (Y/N)?
```

- Question 4
  - Source Code & Explanation

```
# Declare a list which takes in values inputted by the user
numbers_list = []

# Check whether input is integer (a whole number)
has_stopped = False

while not has_stopped:
    num_input = input("Enter number to add to list: ")

    if num_input.lstrip('+-').isnumeric():   # If input in string is a
numerical value, then exit loop
        num_input = int(num_input)

        # Check that number does not exist in numbers_list
        if num_input in numbers_list:
            valid_input = False
            print("Element already exists in list! Please try again!")
        else:
            # Check that number entered by user is between 1 and 1024
            if num_input < 1 or num_input > 1024:
                valid_input = False
                print("Number is not between 1 and 1024! Please try again!")
            else:
                valid_input = True
                numbers_list.append(num_input)

                valid_choice = False

                while not valid_choice:
                    # Ask user to decide whether to continue
                    choice = input("Do you want to continue adding more
numbers (Y/N)?\n")

                    # Check if user entered a value before pressing Enter
                    if len(choice) > 0:
                        if choice[0].capitalize() == 'Y':   # valid option,
proceeds with entering next number
                            has_stopped = False
                            valid_choice = True
                        elif choice[0].capitalize() == 'N': # valid option,
exits entire iteration
                            has_stopped = True
                            valid_choice = True
                        else:   # invalid option, asks user to enter a valid
option
                            valid_choice = False
                            print("Please enter Y or N (Yes or No)!")
                    else:
                        valid_choice = False
                        print("Please enter a value!")

    else:   # If not, ask the user to try again
        valid_input = False
        print("Input has to be a number! Please try again!")

# Print out generated list to user
print("\nList of numbers:")
print(numbers_list)

# Declare a list which stores any distinct pairs of integers found
pairs_list = []

# Check for distinct pairs of integers
for a in range(len(numbers_list)):
    for b in range(a,len(numbers_list)):
        for c in range(a+1,len(numbers_list)):
            for d in range(c,len(numbers_list)):
                # If conditions below are True, then add number to their own
distinct pair and add them to pairs_list
                if (numbers_list[a]*numbers_list[b] ==
numbers_list[c]*numbers_list[d] and numbers_list[a]!=numbers_list[b] and
numbers_list[a]!=numbers_list[c] and numbers_list[a]!=numbers_list[d] and
numbers_list[b]!=numbers_list[c] and numbers_list[b]!=numbers_list[d] and
numbers_list[c]!=numbers_list[d]):
                    pair_1 = (numbers_list[a],numbers_list[b])
                    pair_2 = (numbers_list[c],numbers_list[d])
                    pair_val = [pair_1,pair_2]
                    pairs_list.append(pair_val)

print('\n')

# After exitting loop, program checks if there were any distinct pairs of
integers
if not pairs_list:  # If pairs_list is empty, then there were no distinct
pairs of integers
    print("No distinct pairs of integers!")
else:   # Otherwise, there were distinct pairs of integers
    # Print out distinct pairs
    print('Distinct pairs of integers:')
    for i in pairs_list:
        print(i)
```

```python
print("\nNo more distinct factors!\nThank you for using this program!")
```

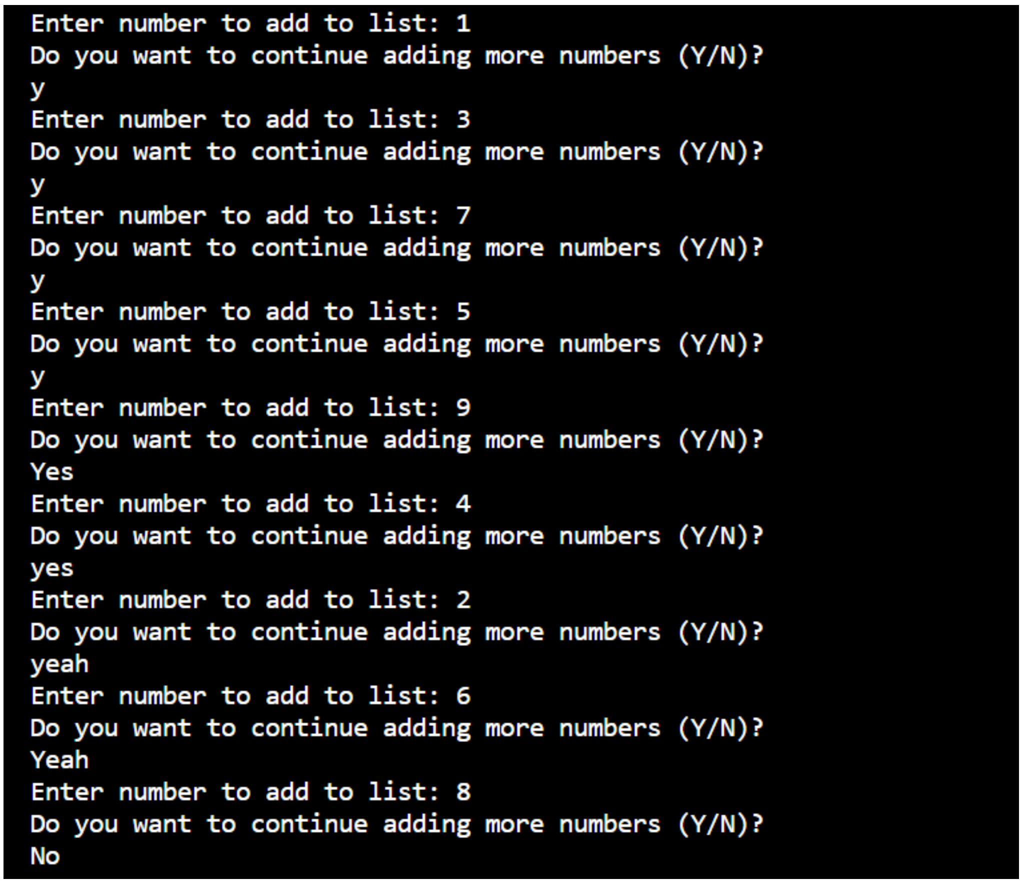
The program begins by declaring a list and initialising its value to an empty list. It then proceeds by asking the user to enter a number to be added to the list. The program checks whether the user input is an integer. If successful, the number is added to the array the user is then prompted to enter whether he/she wants to add any more numbers in the array. The program first checks whether the user enters something in the console before pressing Enter. If that is successful, it then checks whether the first character of the user input is either a ‘Y’ or an ‘N’. If the first character is a ‘Y’, the program redirects the user to enter another number. Otherwise, if it is an ‘N’, the iteration stops and proceeds with the rest of execution. If all checks are unsuccessful, the user is redirected accordingly.

Once this process is done successfully, the program then declares another list called pairs_list which is used to add in any distinct pairs of integers that are found. It then proceeds by finding any distinct pairs. This is done by using 3 nested for loops. If any pairs are found, each pair is set in its own variable and then added to pairs list. Once all pairs are found, if any, a message is then displayed on the console stating whether there had been distinct pairs found or not. If there were any pairs found, the program also prints them out.

❖ How the Program was Tested + Screen Dumps

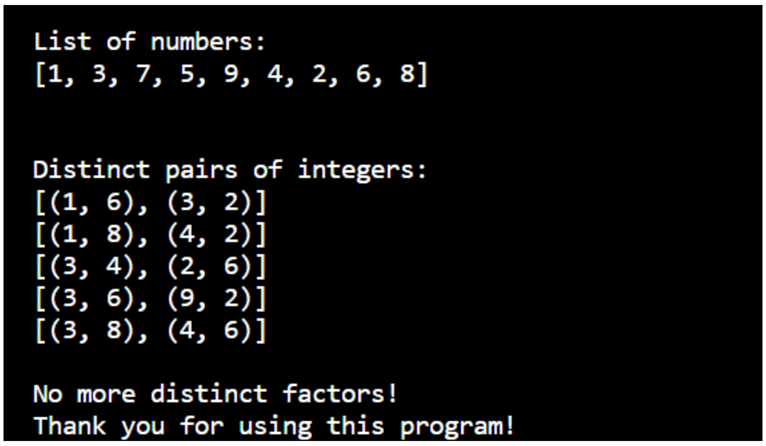
The program was first tested by entering a set of numbers such that there exist distinct pairs of integers that have the same product. The numbers entered were 1, 3, 7, 5, 9, 4, 2, 6 and 8 in that order. As expected, the program realised that there are distinct pairs, and those were printed out to the console. This can be seen below as follows:

Inputting numbers



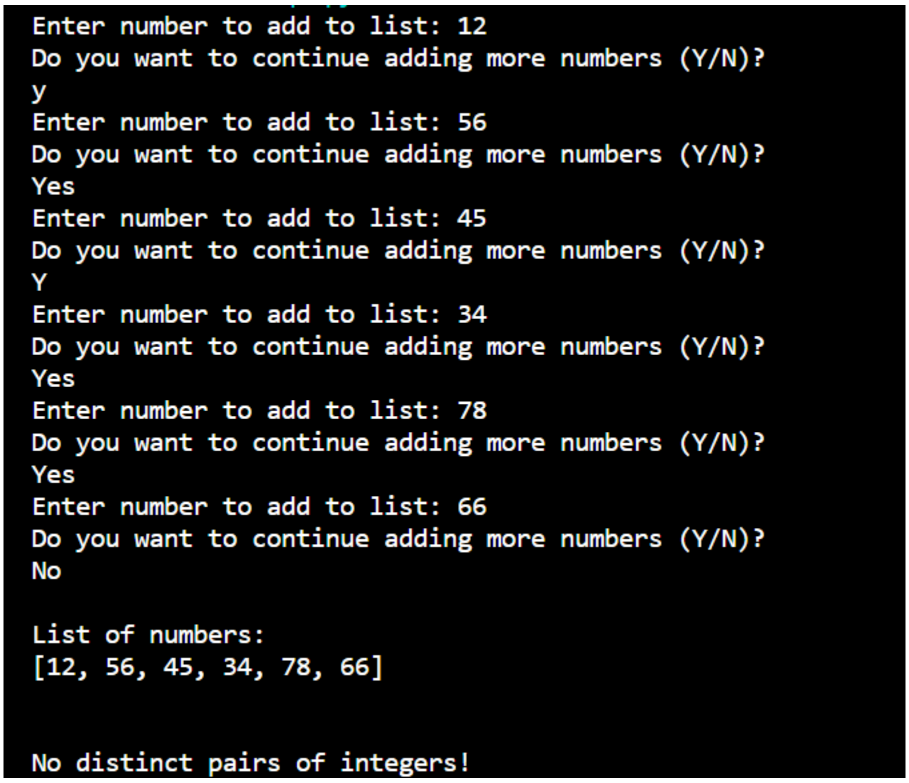
```
Enter number to add to list: 1
Do you want to continue adding more numbers (Y/N)?
y
Enter number to add to list: 3
Do you want to continue adding more numbers (Y/N)?
y
Enter number to add to list: 7
Do you want to continue adding more numbers (Y/N)?
y
Enter number to add to list: 5
Do you want to continue adding more numbers (Y/N)?
y
Enter number to add to list: 9
Do you want to continue adding more numbers (Y/N)?
Yes
Enter number to add to list: 4
Do you want to continue adding more numbers (Y/N)?
yes
Enter number to add to list: 2
Do you want to continue adding more numbers (Y/N)?
yeah
Enter number to add to list: 6
Do you want to continue adding more numbers (Y/N)?
Yeah
Enter number to add to list: 8
Do you want to continue adding more numbers (Y/N)?
No
```

Output



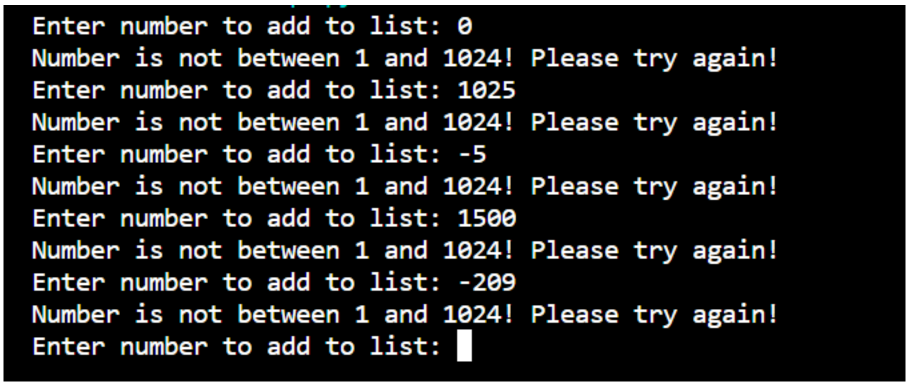
```
List of numbers:
[1, 3, 7, 5, 9, 4, 2, 6, 8]


Distinct pairs of integers:
[(1, 6), (3, 2)]
[(1, 8), (4, 2)]
[(3, 4), (2, 6)]
[(3, 6), (9, 2)]
[(3, 8), (4, 6)]

No more distinct factors!
Thank you for using this program!
```

The program was then tested by entering a set of numbers such that there are no distinct pairs of integers. As expected, the program detected that this was the case and thus, the result was that a message was printed stating that there were no distinct pairs of integers. This is shown below:



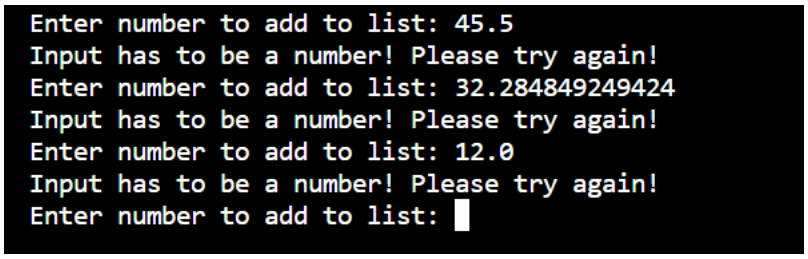
```
Enter number to add to list: 12
Do you want to continue adding more numbers (Y/N)?
y
Enter number to add to list: 56
Do you want to continue adding more numbers (Y/N)?
Yes
Enter number to add to list: 45
Do you want to continue adding more numbers (Y/N)?
Y
Enter number to add to list: 34
Do you want to continue adding more numbers (Y/N)?
Yes
Enter number to add to list: 78
Do you want to continue adding more numbers (Y/N)?
Yes
Enter number to add to list: 66
Do you want to continue adding more numbers (Y/N)?
No

List of numbers:
[12, 56, 45, 34, 78, 66]


No distinct pairs of integers!
```

Next, the program was tested by entering numbers that were not between 1 and 1024. The program detected that the numbers broke this condition and thus, a message was printed out prompting the user to enter a number between 1 and 1024. This can be seen below as follows:



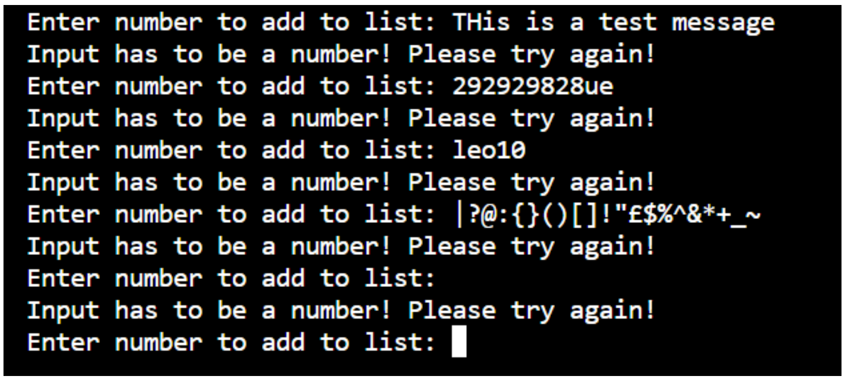
```
Enter number to add to list: 0
Number is not between 1 and 1024! Please try again!
Enter number to add to list: 1025
Number is not between 1 and 1024! Please try again!
Enter number to add to list: -5
Number is not between 1 and 1024! Please try again!
Enter number to add to list: 1500
Number is not between 1 and 1024! Please try again!
Enter number to add to list: -209
Number is not between 1 and 1024! Please try again!
Enter number to add to list: 
```

Afterwards, the program was tested by entering a set of float values. This was done to test the program's ability to distinguish a float from an integer. As expected, the program detected that such user input is invalid and thus, a message was displayed on the console telling the user that input has to be a number, before redirecting the user to enter a valid number. This can be seen below:



```
Enter number to add to list: 45.5
Input has to be a number! Please try again!
Enter number to add to list: 32.284849249424
Input has to be a number! Please try again!
Enter number to add to list: 12.0
Input has to be a number! Please try again!
Enter number to add to list: 
```

The program was tested for the final time by again restricting user input, this time entering string values only. As expected, the program detected that this user input is again invalid. Thus, as a result, a message was printed out on the console stating that the user input has to be a number, before redirecting the user to enter a number. This can be seen as follows:



```
Enter number to add to list: THis is a test message
Input has to be a number! Please try again!
Enter number to add to list: 292929828ue
Input has to be a number! Please try again!
Enter number to add to list: leo10
Input has to be a number! Please try again!
Enter number to add to list: |?@:{}()[]!"£$%^&*+_~
Input has to be a number! Please try again!
Enter number to add to list:
Input has to be a number! Please try again!
Enter number to add to list: 
```

The tests for the choice whenever the user is asked whether to add more numbers or not were done in a similar way as in the previous program, with the very same end results. For all these tests, it was assumed that the number that was to be added to the list was valid.

- Question 5
  - Source Code & Explanation

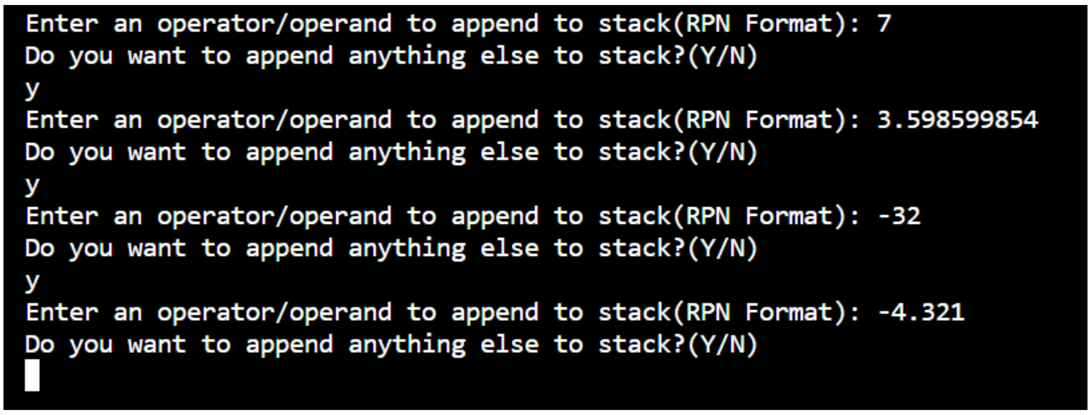
```
# Declaration of stack array used to evaluate arithmetic expressions in RPN
format
stack = []


# Keep on entering elements in stack until user enters 'N'
finished_input = False

while not finished_input:
    # Checks for valid user input for number or operator
    valid_option_1 = False

    while not valid_option_1:
        # Asks the user to enter an item (operator/operand)
        item = input("Enter an operator/operand to append to stack(RPN
Format): ")

        # Check if user did not enter a digit and instead entered a string of
length larger than 1 or did not enter any value
        if not item.lstrip('-+').replace(".","").isdigit() and len(item) != 1:
            valid_option_1 = False
            print("Please enter a valid number or any of the following
operators (+,-,/,*)!")
        else:
            # Check if user input is either a number or any of the operators:
+, -, *, /
            if item.lstrip('-+').replace(".","").isdigit() or ord(item) == 43
or ord(item) == 42 or ord(item) == 45 or ord(item) == 47:  # If True, then
item is valid, so exit loop
                valid_option_1 = True
            else:   # If False, item is invalid, so ask user to enter a valid
item
                valid_option_1 = False
                print("Wrong number or operator! Please re-enter item!")

    # Append item to stack
    stack.append(item)

    # Checks for user choice
    valid_option_2 = False

    while not valid_option_2:
        choice = input("Do you want to append anything else to stack?(Y/N)\n")
        # Check if user enters something before pressing Enter
        if len(choice) > 0:
            if choice[0].capitalize() == 'Y':   # valid option, proceeds with
entering next item
                valid_option_2 = True
                finished_input = False
            elif choice[0].capitalize() == 'N': # valid option, exits entire
iteration
                valid_option_2 = True
                finished_input = True
            else:   # invalid option, asks user to enter a valid option
                valid_option_2 = False
                print("Please enter Y or N (Yes or No)!")
        else:
            valid_option_2 = False
            print("Please enter a value!")

# Stack pointer
i = 0

# Used to track how many operations have been done until the end
counter = 0

while i in range(len(stack)):
    # Print contents of stack
    print("Operation "+str(counter)+":")
    print(stack)
    print('\n')

    counter += 1

    # Check if current item is a number and contains a decimal point (hence a
float)
    if stack[i].lstrip('-+').replace(".","").isdigit():  # if number, convert
it accordingly
        index = stack[i].find(".")
        if index == -1: # if integer, convert current item into int
            stack[i] = int(stack[i])
        else:   # if float, convert current item into float
            stack[i] = float(stack[i])
        i += 1
    else:   # if it is an operator, then perform operation
        if ord(stack[i]) == 43:     # addition
            num1 = stack[i-2]
            num2 = stack[i-1]
            j = i-1
            stack.pop(j)
            j -= 1
            stack.pop(j)
            stack[j] = num1+num2
            i -= 1
        elif ord(stack[i]) == 42:   # multiplication
            num1 = stack[i-2]
            num2 = stack[i-1]
            j = i-1
            stack.pop(j)
            j -= 1
            stack.pop(j)
            stack[j] = num1*num2
            i -= 1
        elif ord(stack[i]) == 47:   # division
            num1 = stack[i-2]
            num2 = stack[i-1]
            j = i-1
            stack.pop(j)
            j -= 1
            stack.pop(j)
            stack[j] = num1/num2
            i -= 1
        elif ord(stack[i]) == 45:   # subtraction
            num1 = stack[i-2]
            num2 = stack[i-1]
            j = i-1
            stack.pop(j)
            j -= 1
            stack.pop(j)
            stack[j] = num1-num2
            i -= 1

    # Prints out final stack before displaying result in the next line
    if len(stack) == 1:
        print("Final Stack:")
        print(stack)
        print('\n')

# Check whether size of stack at the end of execution is larger than 1
if len(stack) > 1:  # if True, then expression not fully solved - prints out
error
    print("Error! Expression not completely solved!")
else:   # if False, then print the result
    print("Final Result:",stack[0])
```

The program starts by declaring a variable which is used as the stack. It then asks the user to enter either a number or any valid operator as listed in the source code. Then, the program checks whether the item entered by the user is either a number (integer or float) or a valid number. If so, the item is appended to the list. If not, an error gets displayed on the console and the user is redirected to enter a valid item. Once this process is successful, the user is then prompted to enter whether he/she wants to add other items into the list. If the first character of the user input is a ‘Y’, the user is redirected to enter another item. If it is an ‘N’, the program exits iteration and proceeds with the rest of execution. Otherwise, if the user input is invalid, an error gets printed on the console and the user is prompted to enter a valid choice.

Once this whole process is completed, the program then evaluates the expression entered in RPN format. If the item being processed is a number, it is converted into the correct data type. Otherwise, if it is an operator, the program performs the correct operation on the 2 numbers preceding it. This is done by checking the ASCII decimal value of the operator being processed. The result of each operation is displayed to the user. After the expression is evaluated, if it finds a conflict on the way, a message gets displayed on the console stating that the program could not finish evaluating expression. Otherwise, it prints out the result from the stack.

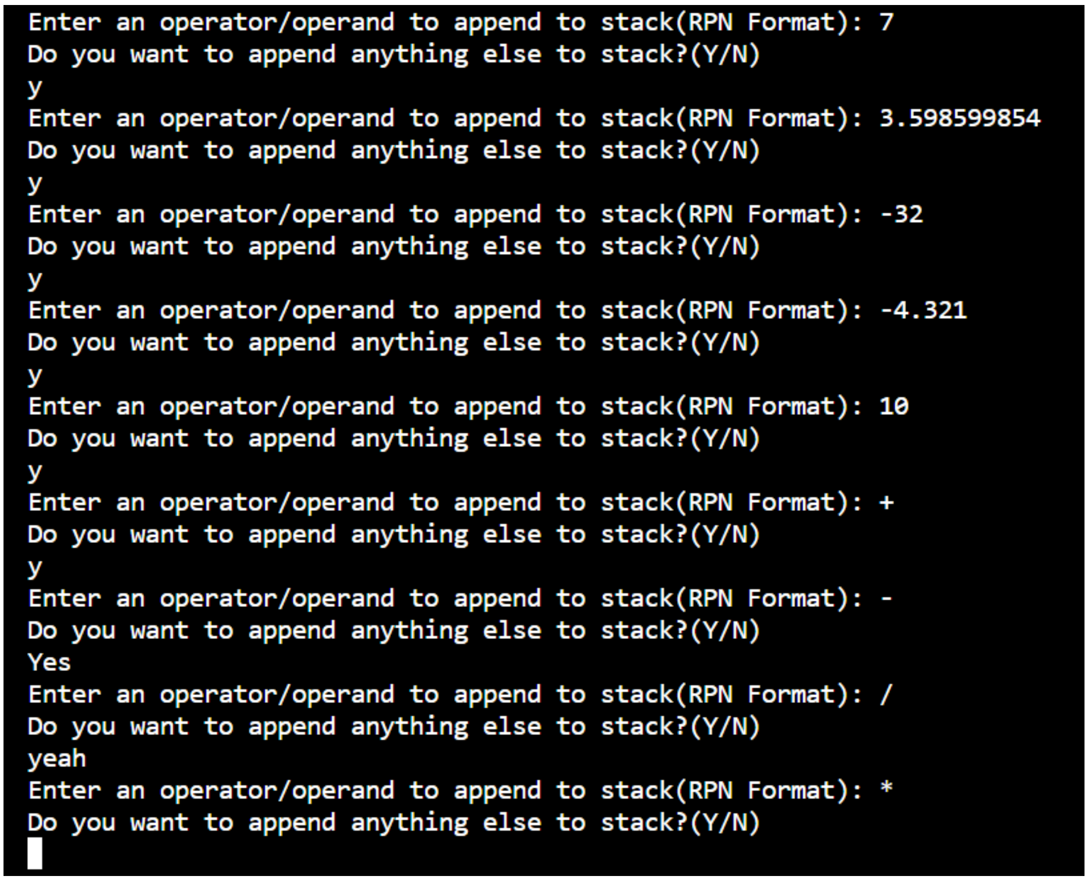
❖ How the Program was Tested + Screen Dumps

The program was first tested by inputting several numbers to stack. These include integers and floats. As expected, the program detected that this type of user input is valid, and proceeded by asking the user to choose whether to add more operands to stack. This can be seen below:



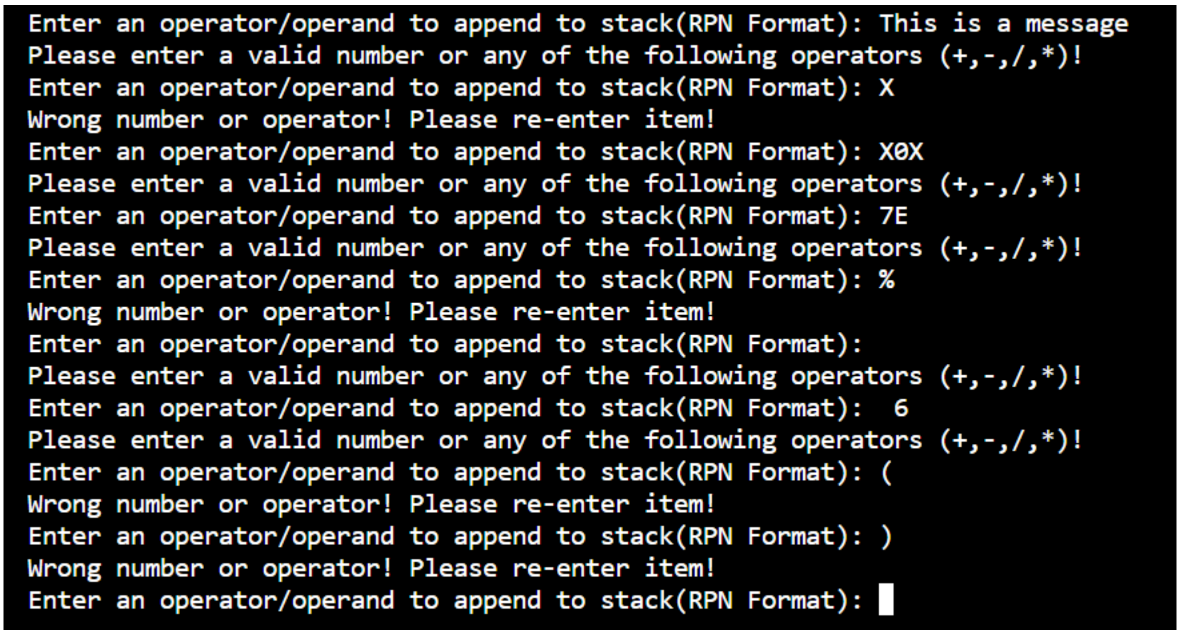
```
Enter an operator/operand to append to stack(RPN Format): 7
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): 3.598599854
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): -32
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): -4.321
Do you want to append anything else to stack?(Y/N)
█
```

The second test was done by inputting any of the following operators: +, -, /, *. This was done using the very same numbers as above, with an additional number included to make room for an additional operator. As expected, the program detected these operators as valid and, once again, proceeded by asking the user to choose whether to add more operands to stack. This can be seen below:



```
Enter an operator/operand to append to stack(RPN Format): 7
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): 3.598599854
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): -32
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): -4.321
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): 10
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): +
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): -
Do you want to append anything else to stack?(Y/N)
Yes
Enter an operator/operand to append to stack(RPN Format): /
Do you want to append anything else to stack?(Y/N)
yeah
Enter an operator/operand to append to stack(RPN Format): *
Do you want to append anything else to stack?(Y/N)
█
```

The last test that was done for stack input was done by inputting any invalid operands and operators. This includes strings, individual characters and any operator that is considered invalid, such as % which is the modulus operator. As expected, the program saw this type of user input as invalid. This can be seen below:

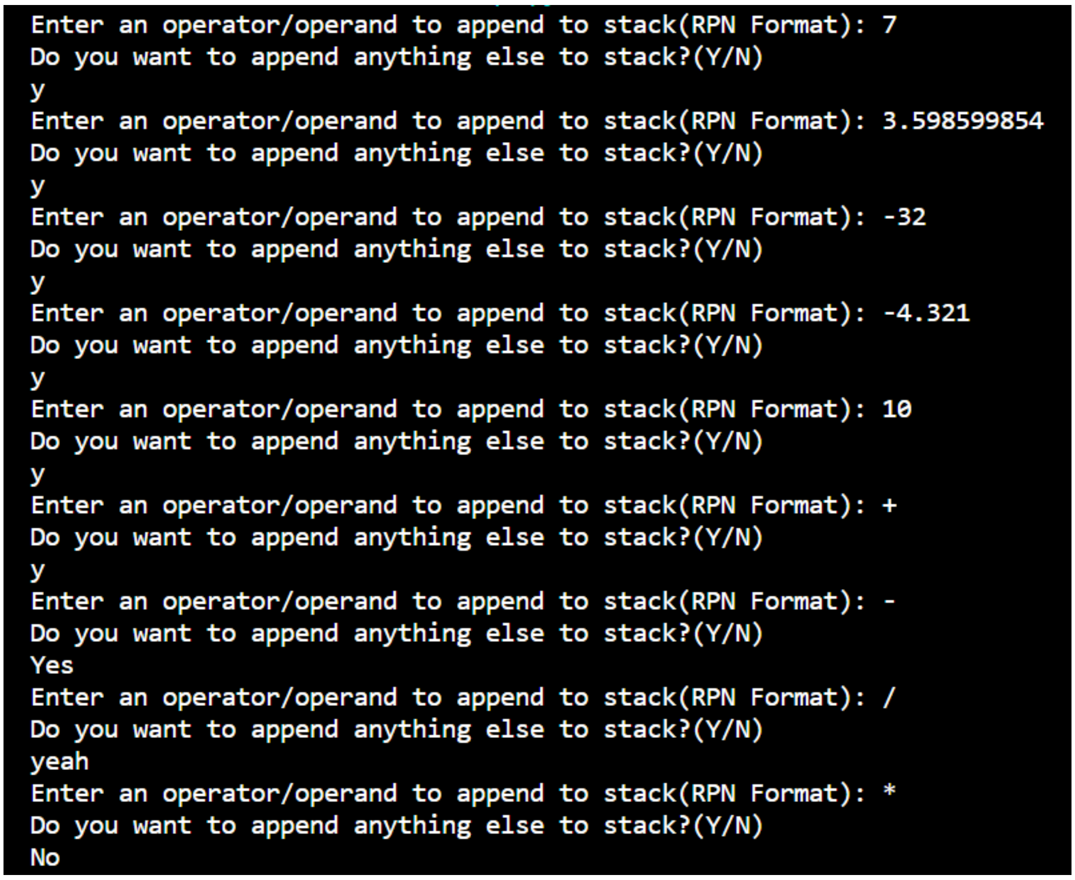


```
Enter an operator/operand to append to stack(RPN Format): This is a message
Please enter a valid number or any of the following operators (+,-,/,*)!
Enter an operator/operand to append to stack(RPN Format): X
Wrong number or operator! Please re-enter item!
Enter an operator/operand to append to stack(RPN Format): X0X
Please enter a valid number or any of the following operators (+,-,/,*)!
Enter an operator/operand to append to stack(RPN Format): 7E
Please enter a valid number or any of the following operators (+,-,/,*)!
Enter an operator/operand to append to stack(RPN Format): %
Wrong number or operator! Please re-enter item!
Enter an operator/operand to append to stack(RPN Format):
Please enter a valid number or any of the following operators (+,-,/,*)!
Enter an operator/operand to append to stack(RPN Format):  6
Please enter a valid number or any of the following operators (+,-,/,*)!
Enter an operator/operand to append to stack(RPN Format): (
Wrong number or operator! Please re-enter item!
Enter an operator/operand to append to stack(RPN Format): )
Wrong number or operator! Please re-enter item!
Enter an operator/operand to append to stack(RPN Format):
```

The next few tests were done to test the user input whenever the user is asked whether to input any more items to stack. These tests were also used to test the program's functionality when evaluating the stack when the user finishes inputting all necessary items. The tests whenever the user types a string with the first character as a 'Y' was already tested previously.

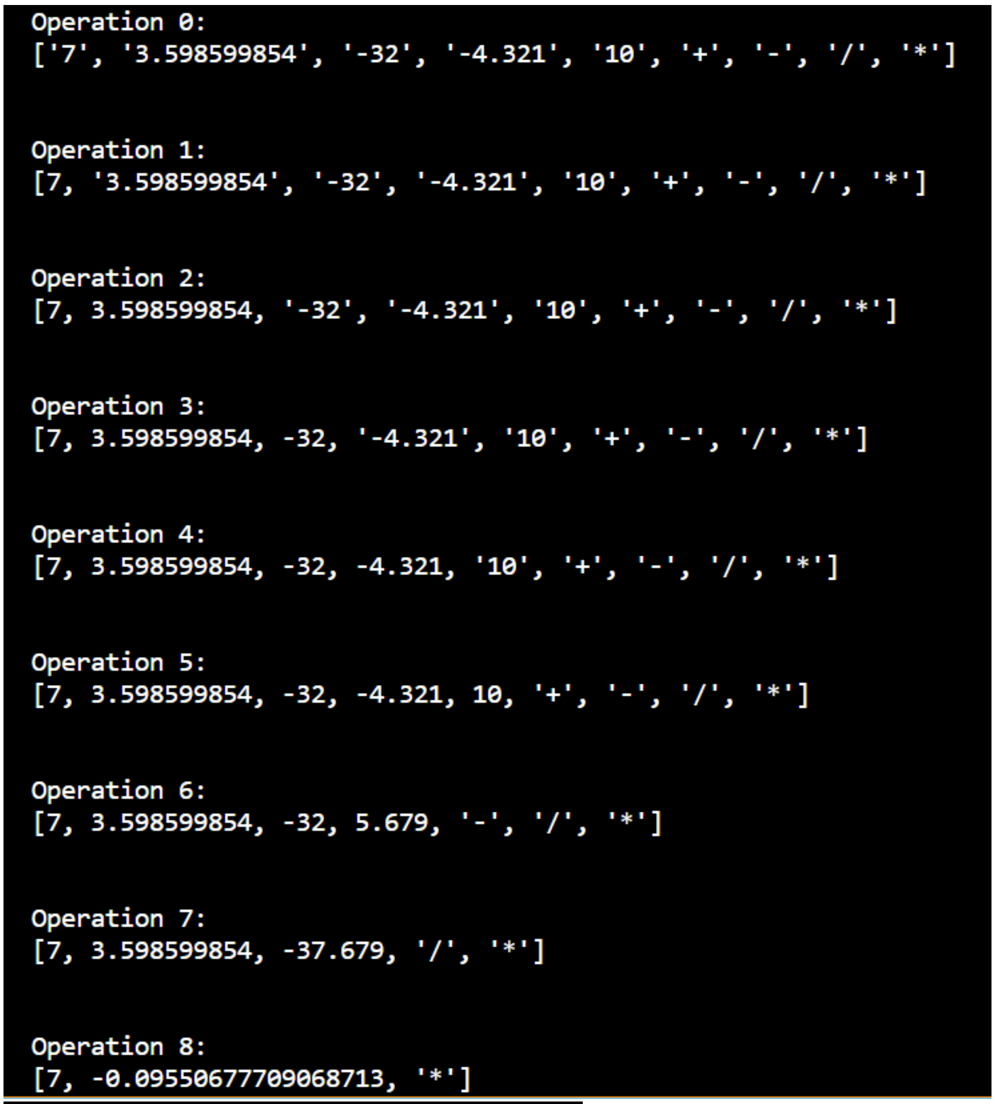
Apart from that, the first test was done to test whenever the user inputs any string with the first character 'N'. As expected, the program stopped asking the user to enter a new item and proceeded by evaluating the stack, if possible. The output generated was, after each operation, the contents of the stack were displayed and then, once all the stack is evaluated, the final answer was outputted individually. The items used for this test were taken from the valid items test above. This can be seen below in the next few pages:

Inputting values

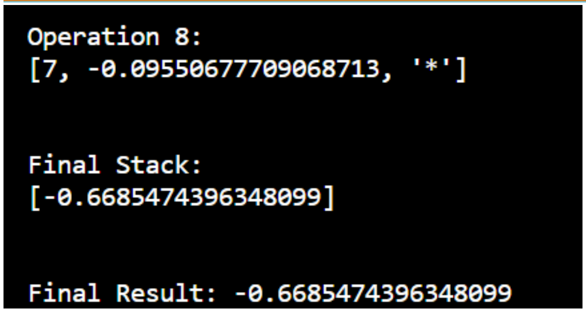


```
Enter an operator/operand to append to stack(RPN Format): 7
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): 3.598599854
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): -32
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): -4.321
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): 10
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): +
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): -
Do you want to append anything else to stack?(Y/N)
Yes
Enter an operator/operand to append to stack(RPN Format): /
Do you want to append anything else to stack?(Y/N)
yeah
Enter an operator/operand to append to stack(RPN Format): *
Do you want to append anything else to stack?(Y/N)
No
```

Operation Evaluation and Answer Extraction:



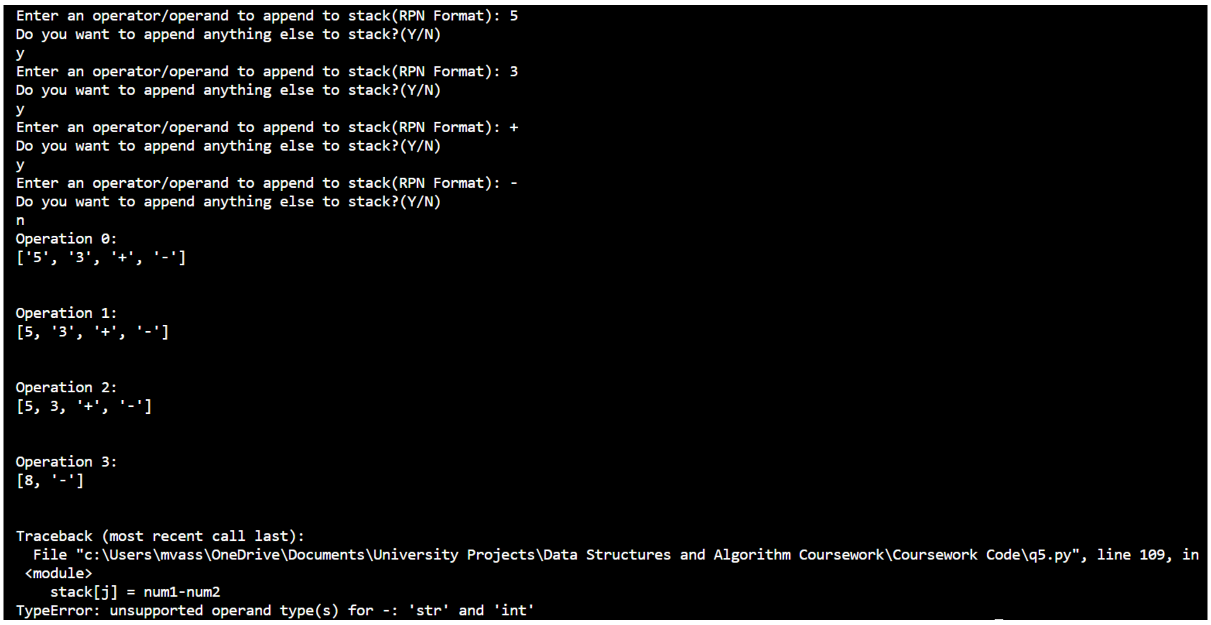
```
Operation 0:
['7', '3.598599854', '-32', '-4.321', '10', '+', '-', '/', '*']


Operation 1:
[7, '3.598599854', '-32', '-4.321', '10', '+', '-', '/', '*']


Operation 2:
[7, 3.598599854, '-32', '-4.321', '10', '+', '-', '/', '*']


Operation 3:
[7, 3.598599854, -32, '-4.321', '10', '+', '-', '/', '*']


Operation 4:
[7, 3.598599854, -32, -4.321, '10', '+', '-', '/', '*']


Operation 5:
[7, 3.598599854, -32, -4.321, 10, '+', '-', '/', '*']


Operation 6:
[7, 3.598599854, -32, 5.679, '-', '/', '*']


Operation 7:
[7, 3.598599854, -37.679, '/', '*']


Operation 8:
[7, -0.09550677709068713, '*']
```



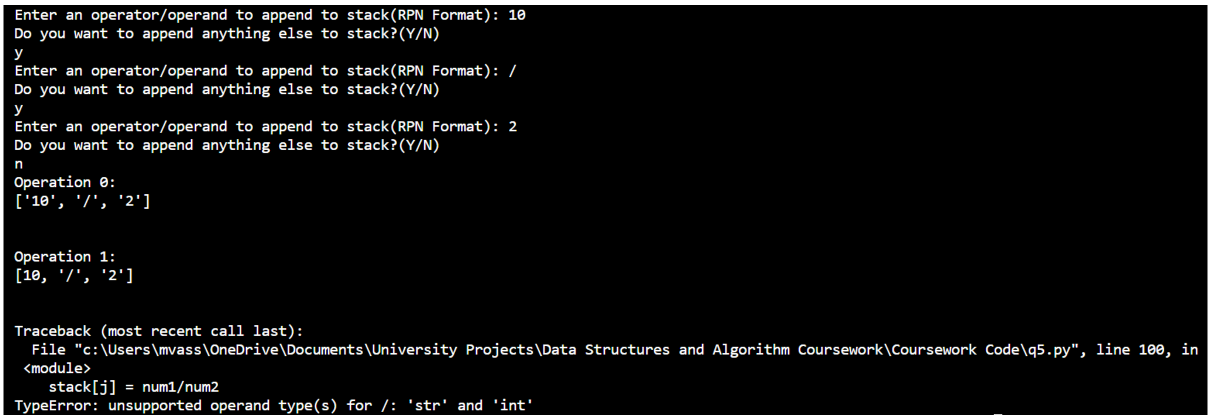
```
Operation 8:
[7, -0.09550677709068713, '*']
```

```
Final Stack:
[-0.6685474396348099]


Final Result: -0.6685474396348099
```

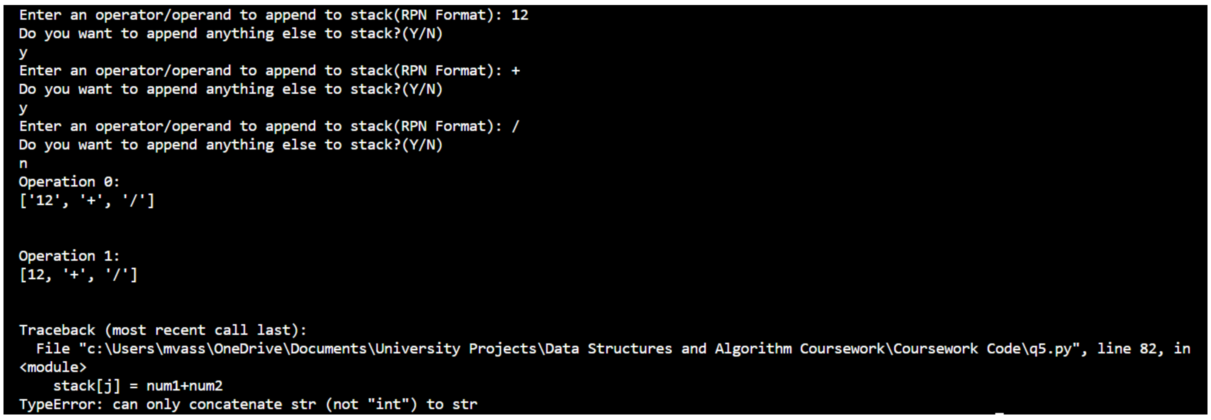
The next test was done in the case where the stack could not be fully evaluated. This was tested by having more or equal number of operators as operands and having an operand as the last item in stack. Unfortunately, what happens is, whenever the stack pointer reaches an operator and it detects only one operand before it, an error gets generated and the program suddenly terminates. This can be seen below:



```
Enter an operator/operand to append to stack(RPN Format): 5
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): 3
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): +
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): -
Do you want to append anything else to stack?(Y/N)
n
Operation 0:
['5', '3', '+', '-']


Operation 1:
[5, '3', '+', '-']


Operation 2:
[5, 3, '+', '-']


Operation 3:
[8, '-']


Traceback (most recent call last):
  File "c:\Users\mvass\OneDrive\Documents\University Projects\Data Structures and Algorithm Coursework\Coursework Code\q5.py", line 109, in
 <module>
    stack[j] = num1-num2
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

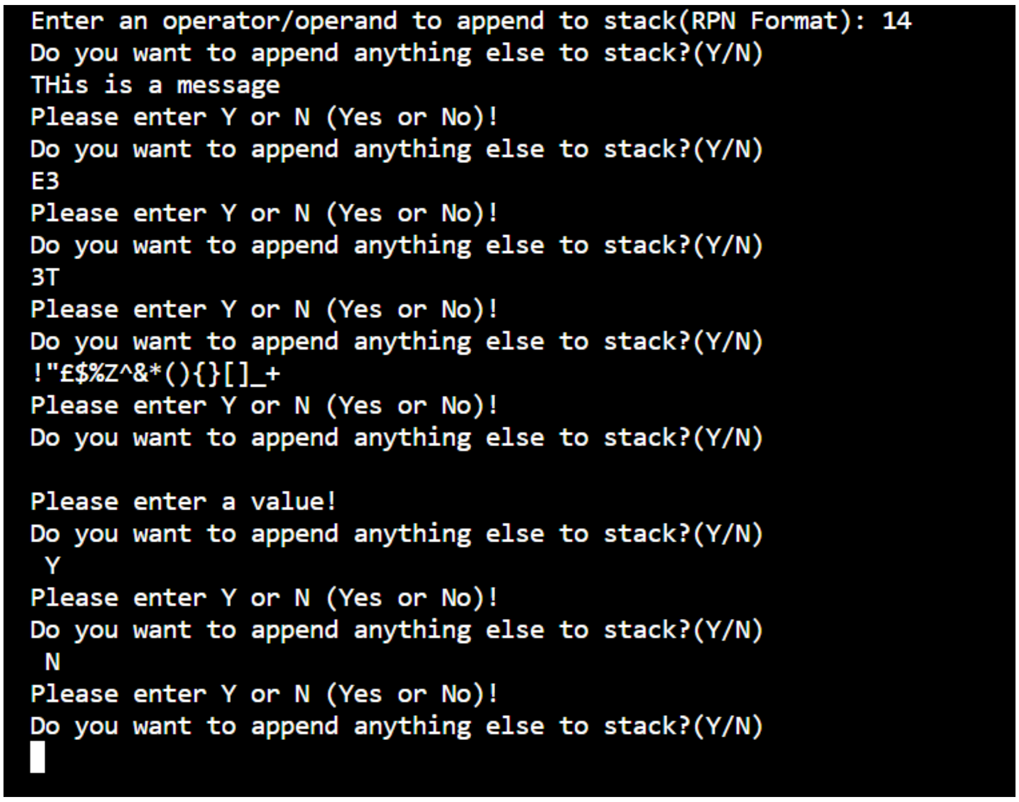


```
Enter an operator/operand to append to stack(RPN Format): 10
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): /
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): 2
Do you want to append anything else to stack?(Y/N)
n
Operation 0:
['10', '/', '2']


Operation 1:
[10, '/', '2']


Traceback (most recent call last):
  File "c:\Users\mvass\OneDrive\Documents\University Projects\Data Structures and Algorithm Coursework\Coursework Code\q5.py", line 100, in
 <module>
    stack[j] = num1/num2
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```



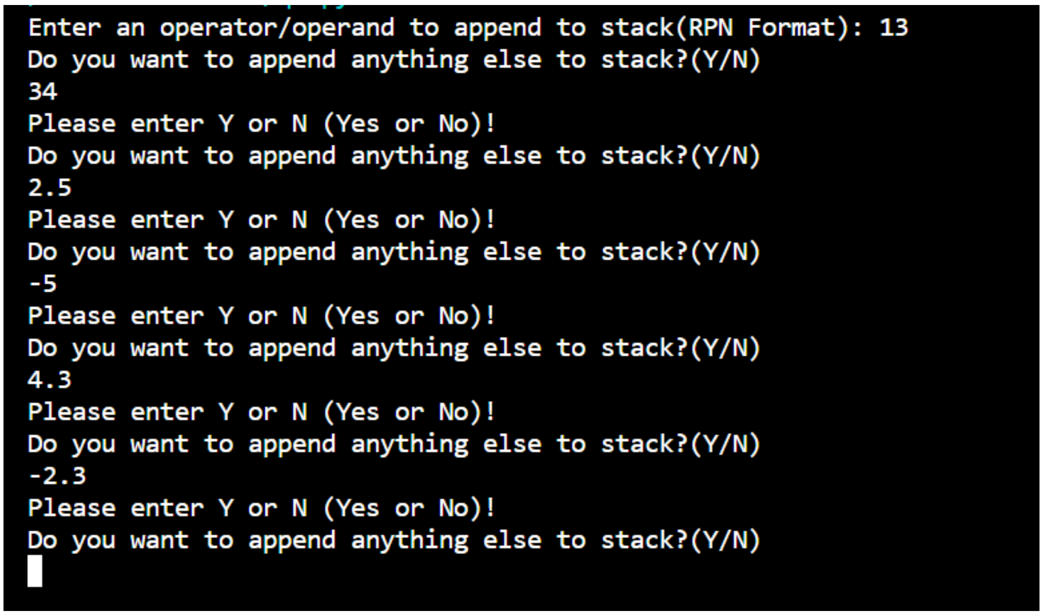
```
Enter an operator/operand to append to stack(RPN Format): 12
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): +
Do you want to append anything else to stack?(Y/N)
y
Enter an operator/operand to append to stack(RPN Format): /
Do you want to append anything else to stack?(Y/N)
n
Operation 0:
['12', '+', '/']


Operation 1:
[12, '+', '/']


Traceback (most recent call last):
  File "c:\Users\mvass\OneDrive\Documents\University Projects\Data Structures and Algorithm Coursework\Coursework Code\q5.py", line 82, in
<module>
    stack[j] = num1+num2
TypeError: can only concatenate str (not "int") to str
```

The next test was done specifically to test the choice of the user whenever an invalid option is entered. This was done by inputting several strings which do not have the first characters as ‘Y’ or ‘N’. As expected, the program saw such user input as invalid and a message is displayed on the console prompting the user to enter a valid option. For the last two string values that were tested, a space was inputted before the option. This can be seen below as follows:



```
Enter an operator/operand to append to stack(RPN Format): 14
Do you want to append anything else to stack?(Y/N)
THis is a message
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
E3
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
3T
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
!"£$%Z^&*(){}[]_+
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)

Please enter a value!
Do you want to append anything else to stack?(Y/N)
 Y
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
 N
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
```

The last test on the program was also done to test the choice of the user whenever an invalid option is entered. This time, several numerical values were inputted, which included both integer and float values. As expected, the program once again saw such user input as invalid and a message is displayed on the console prompting the user to enter a valid option. For the last two string values that were tested, a space was inputted before the option. This can be seen below as follows:



```
Enter an operator/operand to append to stack(RPN Format): 13
Do you want to append anything else to stack?(Y/N)
34
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
2.5
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
-5
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
4.3
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
-2.3
Please enter Y or N (Yes or No)!
Do you want to append anything else to stack?(Y/N)
```

- Question 6
  - ❖ Source Code & Explanation

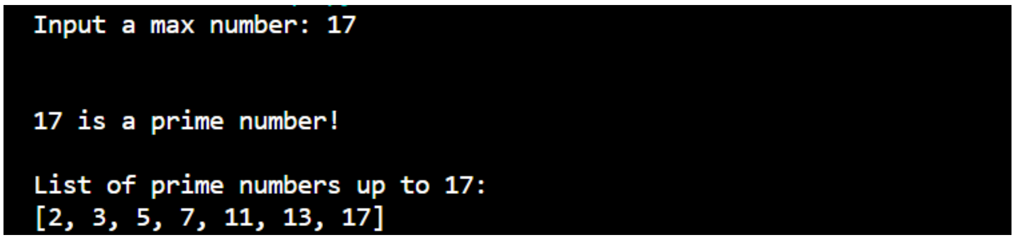
```
# Import math package - used to calculate square root of num (line 50)
import math

def main():
    # Check if user input is a numerical value
    is_num = False

    while not is_num:
        # Ask user to input number to act as end of range of numbers to check
which numbers are prime
        num = input("Input a max number: ")

        # Check if user input is an integer
        if not num.lstrip('+-').isdigit():   # If False, then prompt user to
enter a number
            is_num = False
            print("Please enter a number, preferably an integer!")
        else:   # If True, then exit while loop
            num = int(num)
            if num <= 0:
                is_num = False
                print("Any number less than or equal to 0 is never a prime
number! Please enter a positive number!")
            else:
                is_num = True

    print('\n')

    # List which is used to store every possible prime number
    numbers_list = []

    # Add every number from 2 to that entered by the user (1 is not prime)
    for i in range(2, num+1):
        numbers_list.append(i)

    numbers_list = checkPrimeNumbers(numbers_list, num)

    if num in numbers_list:
        print(str(num) + " is a prime number!\n")
    else:
        print(str(num) + " is not a prime number!\n")

    # Print out list which contains prime numbers
    print("List of prime numbers up to "+str(num)+":")
    print(numbers_list)


# Function which checks which numbers in list are prime numbers and returns
list of prime numbers
# Implements the Sieve of Eratosthenes algorithm
def checkPrimeNumbers(list, num):
    i = 2

    while not i > int(math.sqrt(num)):

        for j in range(2, num):
            element_to_remove = i*j
            if element_to_remove in list:
                list.remove(element_to_remove)
                #print(element_to_remove) - TESTING

        i += 1

        while i not in list:
            i += 1


    return list


# Call main() function to start execution
main()
```

The program begins execution by asking the user to enter a number which is then used by the program to check if that number is a prime number. It even looks for any prime numbers up to that number. After the user inputs a value, the program then checks if the value is an integer. If it not an integer, the user gets an error message displayed on the console and gets redirected to enter a valid number. Otherwise, the program then checks to see if it is less than or equal to 0. If it is, then an error is displayed to the user and is asked to enter a positive number. If it is not, the program exists iteration and carries with the execution.

The program continues by declaring a new variable called numbers_list and adding every number up to the number entered by the user. Then, the program calls the checkPrimeNumbers() function which checks for prime numbers up to a specific number using the Sieve of Eratosthenes algorithm. After the function finishes execution, the list is checked to look for the number entered by the user. A message is then displayed on the console stating that it is either a prime number or not. Finally, the contents of the list are then printed to the user.

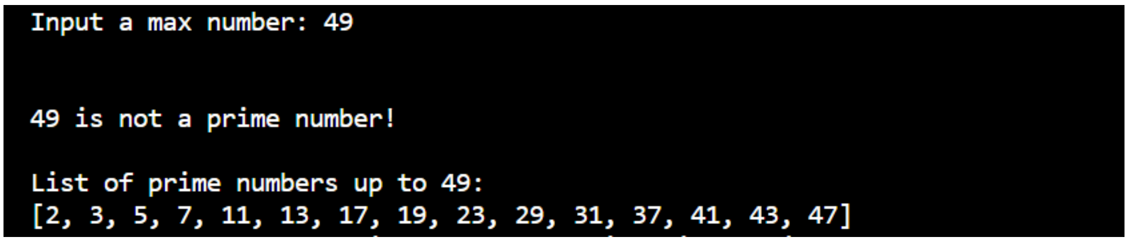
❖ How the Program was Tested + Screen Dumps

The program was first tested by letting the user enter a number which is considered a prime number (e.g. 17). The console then displayed a message stating that the number is indeed a prime number and a list of all the prime numbers up to, and including, that number. This can be seen as follows:



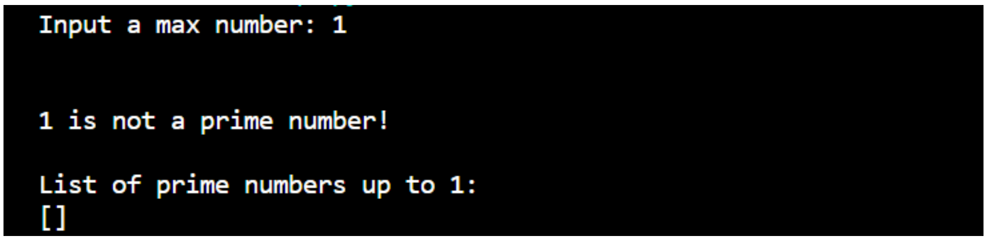
```
Input a max number: 17


17 is a prime number!

List of prime numbers up to 17:
[2, 3, 5, 7, 11, 13, 17]
```

The second test that was done was by entering a positive number which is not considered a prime number (e.g. 49). The console printed out a message stating that the number is not a prime number and then displayed a list of all the prime number up to, but not including, that number. This is shown below:



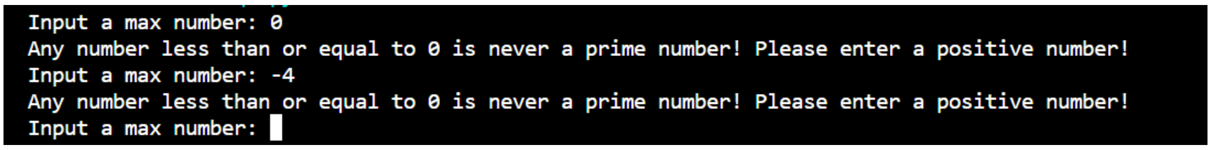
```
Input a max number: 49


49 is not a prime number!

List of prime numbers up to 49:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

The program was then tested for a third time using the number 1. The output displayed on the console stated that the number 1 is indeed not a prime number, but also included an empty list, showing that there are no prime numbers up to the number 1. This can be seen below:



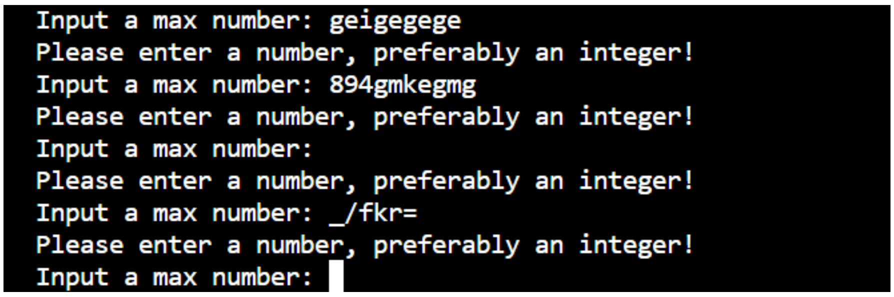
```
Input a max number: 1


1 is not a prime number!

List of prime numbers up to 1:
[]
```

Afterwards, the program was tested by inputting both the number 0 and a negative number (e.g. -4). The program then displayed a message on the console stating that any number which is less than or equal to 0 can never be a prime number. However, the program also prompted the user to enter a positive number on both occasions. This is shown below as follows:



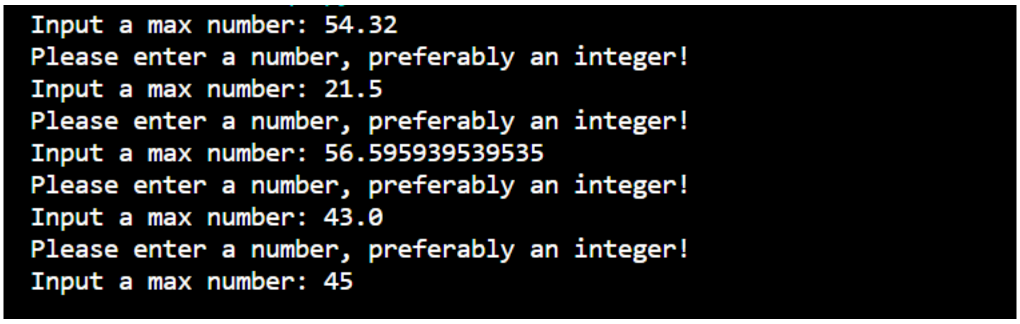
```
Input a max number: 0
Any number less than or equal to 0 is never a prime number! Please enter a positive number!
Input a max number: -4
Any number less than or equal to 0 is never a prime number! Please enter a positive number!
Input a max number: 
```

Afterwards, the program was tested by inputting several strings. This was done to test the data type check for the user input. The tested strings were divided in four parts, whole text, text with numbers included, empty values, and special characters. For all these instances, the console displayed a message telling the user to input an integer. This can be seen below:



```
Input a max number: geigegege
Please enter a number, preferably an integer!
Input a max number: 894gmkegmg
Please enter a number, preferably an integer!
Input a max number:
Please enter a number, preferably an integer!
Input a max number: _/fkr=
Please enter a number, preferably an integer!
Input a max number: 
```

Finally, the program was tested by inputting several numbers of type float. The tested values included numbers with a lot of decimal places and those ending with .0. For all these instances, the console displayed a message telling the user to input an integer, just like the previous test. The results of this test can be seen below:



```
Input a max number: 54.32
Please enter a number, preferably an integer!
Input a max number: 21.5
Please enter a number, preferably an integer!
Input a max number: 56.595939539535
Please enter a number, preferably an integer!
Input a max number: 43.0
Please enter a number, preferably an integer!
Input a max number: 45
```

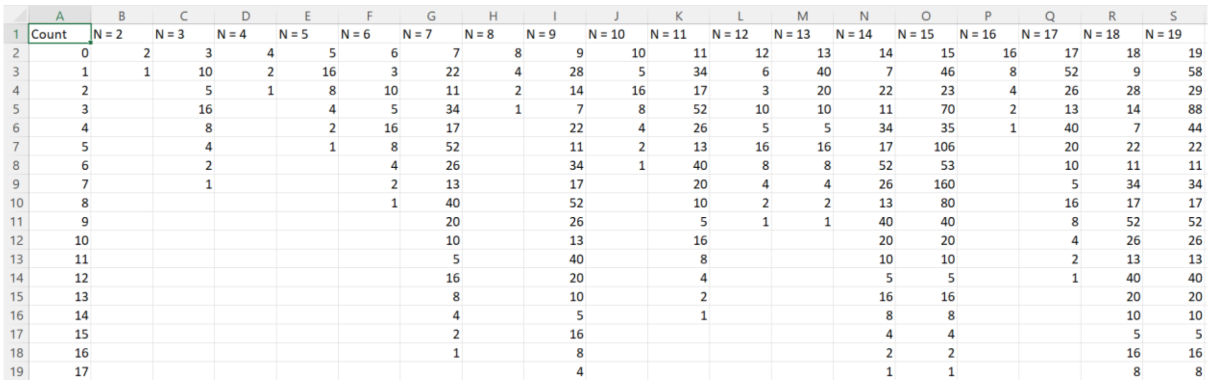
- Question 7
  - Source Code & Explnation

```
# Importation of csv package - used to enter results in a csv file
import csv

def main():
    # Write headers to csv file
    write_header_to_table()

    # Array used to store all generated number from the Collatz sequence
    num_list = [num for num in range(2,513)]

    # Perform Collatz sequence on all numbers in the array
    collatzSequence(num_list)


def collatzSequence(list):
    count = 0
    count_list = []

    # Write original data to csv file
    write_data_to_table(list,0)

    # Keep doing the Collatz sequence until n becomes 1
    size = len(list)

    # Variable used to check if all elements in the array are 1 during
calculation
    all_are_1 = False

    while not all_are_1:
        # Check that every element in the array is equal to 1
        for element in list:
            if (element == 1) or (element == ""):   # if True, keep checking
every element
                all_are_1 = True
            else:   # if False, exit for loop and perform Collatz sequence on
the array
                all_are_1 = False
                count += 1
                count_list.append(count)
                break

        if all_are_1:
            break
        else:
            for i in range(size):
                # Set variable currentElement to be equal to the value of the
array pointed to by index i
                currentElement = list[i]
                if list[i] == 1 or list[i] == "":   # if currentElement is
already equal to 1, then set value as empty since there is no need to print it
again
                    list[i] = ""
                else:   # otherwise, perform Collatz on the current element
                    if currentElement%2==0:  # If n is even, divide by 2
                        currentElement //= 2
                    else:   # If n is odd, perform 3n+1
                        currentElement = (3*currentElement)+1

                    # Assign the value of list[i] to currentElement
                    list[i] = currentElement

            # Once all elements have been dealt with, write the contents of
the list to csv file
            write_data_to_table(list,count)

    print(count_list)

    # Print out contents of array - testing
    print(list)


# Function which writes table headers to csv file
def write_header_to_table():
    header = ['Count']

    # Append another header stating which number is being dealt with in which
column
    for i in range(2,513):
        header.append('N = '+str(i))

    # Write contents of header list to csv file
    with open('data/collatz.csv', 'w', encoding='UTF8', newline='') as f:
        writer = csv.writer(f)

        # write the header
        writer.writerow(header)


def write_data_to_table(list_num,count):
    # Array used to store data of list_num and count
    data = []
    # Declare variable called size and assign its value to the length of
list_num
    size = len(list_num)

    # Add count and every element in the list to the data array
    data.append(count)
    for index in range(size):
        data.append(list_num[index])

    # Write contents of data array to csv file
    with open('data/collatz.csv', 'a', encoding='UTF8', newline='') as f:
        writer = csv.writer(f)

        # write the data
        writer.writerow(data)


# Call main() function to start execution
main()
```

The program starts by writing the header in a file called collatz.csv by calling the write_header_to_file() function. This includes every number between 2 and 512. The program then declares a variable called num_list and sets it to include every number from 2 to 512. It then proceeds by calling the collatzSequence() function which performs the Collatz Sequence on the previously declared list. After every iteration, the results are written into the dataset by calling the write_data_to_file() function. If during operation, the result from any number n becomes 1, in the next iteration, the next value becomes blank to signal that the sequence for n is completed. Execution of this program stops when all final results for each number n have become 1.

❖ How the Program was Tested + Screen Dumps

For this particular program, there is nothing important that was displayed on the console. Instead, all the output is displayed in a csv file called collatz.csv, which evaluated the Collatz sequence of every number from 2 to 512. There were several tests done to check and modify the header for every column, as well as to replace all unnecessary values after 1 with empty strings, which shows that the Collatz sequence for that number is complete. A sample screenshot of the first 18 numbers and 19 rows, including the headers, is shown below:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Count	N = 2	N = 3	N = 4	N = 5	N = 6	N = 7	N = 8	N = 9	N = 10	N = 11	N = 12	N = 13	N = 14	N = 15	N = 16	N = 17	N = 18	N = 19
2	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
3	1	1	10	2	16	3	22	4	28	5	34	6	40	7	46	8	52	9	58
4	2		5	1	8	10	11	2	14	16	17	3	20	22	23	4	26	28	29
5	3		16		4	5	34	1	7	8	52	10	10	11	70	2	13	14	88
6	4		8		2	16	17		22	4	26	5	5	34	35	1	40	7	44
7	5		4		1	8	52		11	2	13	16	16	17	106		20	22	22
8	6		2			4	26		34	1	40	8	8	52	53		10	11	11
9	7		1			2	13		17		20	4	4	26	160		5	34	34
10	8					1	40		52		10	2	2	13	80		16	17	17
11	9						20		26		5	1	1	40	40		8	52	52
12	10						10		13		16			20	20		4	26	26
13	11						5		40		8			10	10		2	13	13
14	12						16		20		4			5	5		1	40	40
15	13						8		10		2			16	16			20	20
16	14						4		5		1			8	8			10	10
17	15						2		16					4	4			5	5
18	16						1		8					2	2			16	16
19	17								4					1	1			8	8

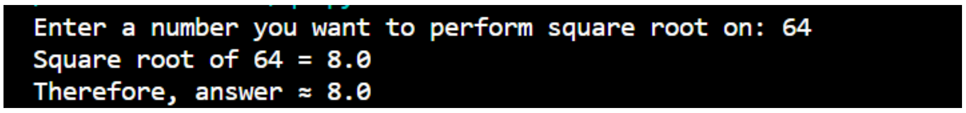
- Question 8
    - ❖ Source Code & Explanation

```
def main():
    # Check that user input is an integer
    is_valid = False

    while not is_valid:
        # Asking user for the number to perform the square root on
        y = input("Enter a number you want to perform square root on: ")

        if y.isdigit() and len(y) > 0:
            y = int(y)
            if y > 0:
                is_valid = True
                y = int(y)
            else:
                is_valid = False
                print("Please enter a positive integer!")
        else:
            is_valid = False
            print("Please enter a number!")

    # Calling square_root_approximation() method and assigning final value in
variable called ans
    ans = square_root_approximation(y)

    # Printing the answer after function execution
    print("Square root of", y, "=", ans)
    print("Therefore, answer ≈", round(ans, 2))


# Function which calculates approximation to the square root of the parameter
num
# using the Newton-Raphson Method
def square_root_approximation(num):
    # Set x0 to num at start of execution
    x0 = num
    count = 0

    # Set nextX to 0 by default before start of while loop
    nextX = 0

    # Calculates 5 approximations of the square root of num
    while count < 10:
        # Checks if first approximation has already been found and sets
        # x0 to nextX if condition returns True
        if count >= 1:
            x0 = nextX

        nextX = x0 - (((x0*x0)-num)/(2*x0))
        count += 1

    return nextX


# Call main() function to start execution
main()
```

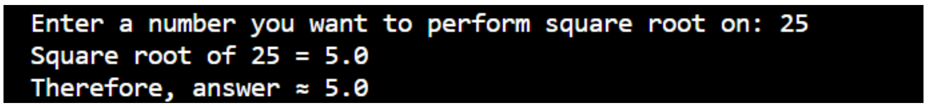
The program begins execution by asking the user to enter a number which is used to calculate its square root. It then checks to see if it is a positive integer. If it is, then proceed with calculating square root. Otherwise, an error message is displayed, and the user is prompted to enter a valid number. Once completed, the square_root_approximation() function is called and the final answer is stored in a variable called ans. The function calculates the square root of the number entered by the user using the Newton-Raphson method. The answer is then printed out to the user.

❖ How the Program was Tested + Screen Dumps

The program was first tested by entering a number which is the square of another number (e.g. 64, 25). The console then printed the answer of the square root. As expected, when a square number is entered, the answer returned was a number ending in .0. This can be seen as follows:

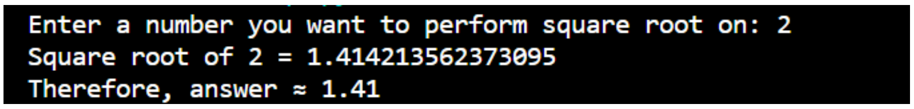


```
Enter a number you want to perform square root on: 64
Square root of 64 = 8.0
Therefore, answer ≈ 8.0
```

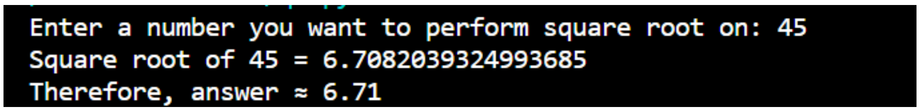


```
Enter a number you want to perform square root on: 25
Square root of 25 = 5.0
Therefore, answer ≈ 5.0
```

The second test was done by entering a number which is not a perfect square of another number (e.g. 2, 45). The console then printed the answer of the square root, both with a lot of decimal places and rounded to 2 decimal places. As expected, when a square number is entered, the answer returned was not a number ending in .0. This can be seen as follows:

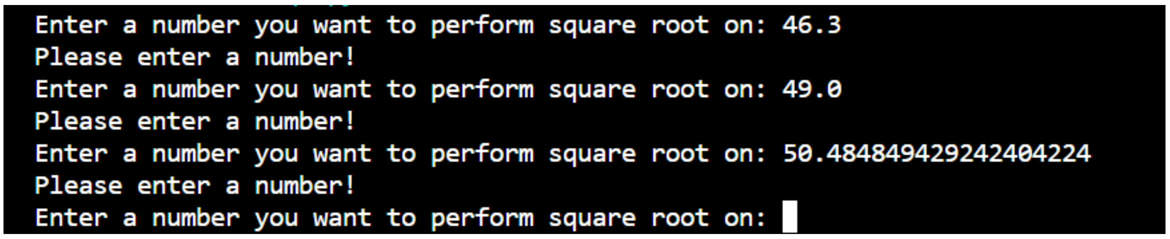


```
Enter a number you want to perform square root on: 2
Square root of 2 = 1.4142135623730951
Therefore, answer ≈ 1.41
```



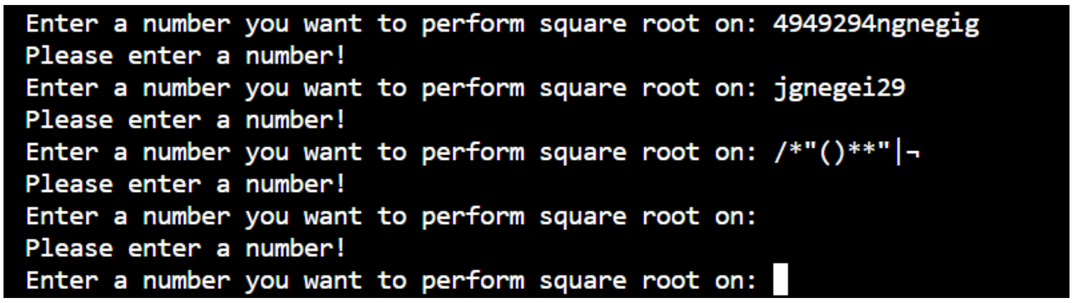
```
Enter a number you want to perform square root on: 45
Square root of 45 = 6.708203932499369
Therefore, answer ≈ 6.71
```

The program was tested for a third time by entering numbers of type float (e.g. 46.3, 49.0). As expected, the program saw this type of user input as invalid and prompted the user to enter an integer. This is shown below.



```
Enter a number you want to perform square root on: 46.3
Please enter a number!
Enter a number you want to perform square root on: 49.0
Please enter a number!
Enter a number you want to perform square root on: 50.484849429242404224
Please enter a number!
Enter a number you want to perform square root on: 
```

Finally, the program was tested by entering several string values. The tested strings were divided in four parts, whole text, text with numbers included, empty values, and special characters. For all these instances, the console displayed a message telling the user to input an integer, as the program saw these user inputs as invalid, as expected. This can be seen below:



```
Enter a number you want to perform square root on: 4949294ngnegig
Please enter a number!
Enter a number you want to perform square root on: jgnegei29
Please enter a number!
Enter a number you want to perform square root on: /*"()**"|¬
Please enter a number!
Enter a number you want to perform square root on:
Please enter a number!
Enter a number you want to perform square root on:
```

- Question 9
    - ❖ Source Code & Explanation

```python
# An array containing list of integers
integers_list = []

# Add user input to list until user chooses to quit
finished_input = False

while not finished_input:
    # Ask the user to enter an item to append to list
    item = input("Enter an item to add to list: ")

    # Check if item to be added is a digit (integer)
    if item.isdigit():  # if True, then append item to list
        item = int(item)
        integers_list.append(item)

        # Check whether user enters a valid option when deciding to continue
        valid_option = False

        while not valid_option:
            # Asks user if he/she wants to continue adding numbers to list
            option = input("Do you want to add more to list? (1 for Yes or 2
for No)\n")

            # Check if option entered by user is a number
            if option.isdigit():    # If True, then proceed with checking of
value
                option = int(option)
                if option == 1: # if Yes, then exit current while loop but not
whole iteration
                    valid_option = True
                    finished_input = False
                elif option == 2:   # if No, then exit from all while loops
                    valid_option = True
                    finished_input = True
                else:   # Otherwise, invalid option - stay on current loop
until user inputs correct option
                    valid_option = False
                    print("Invalid option!")
            else:   # If False, then print error to user
                valid_option = False
                print('Please enter a number, not a string!')
    else:   # if False, print error to user
        print("Please enter a whole number!")
```

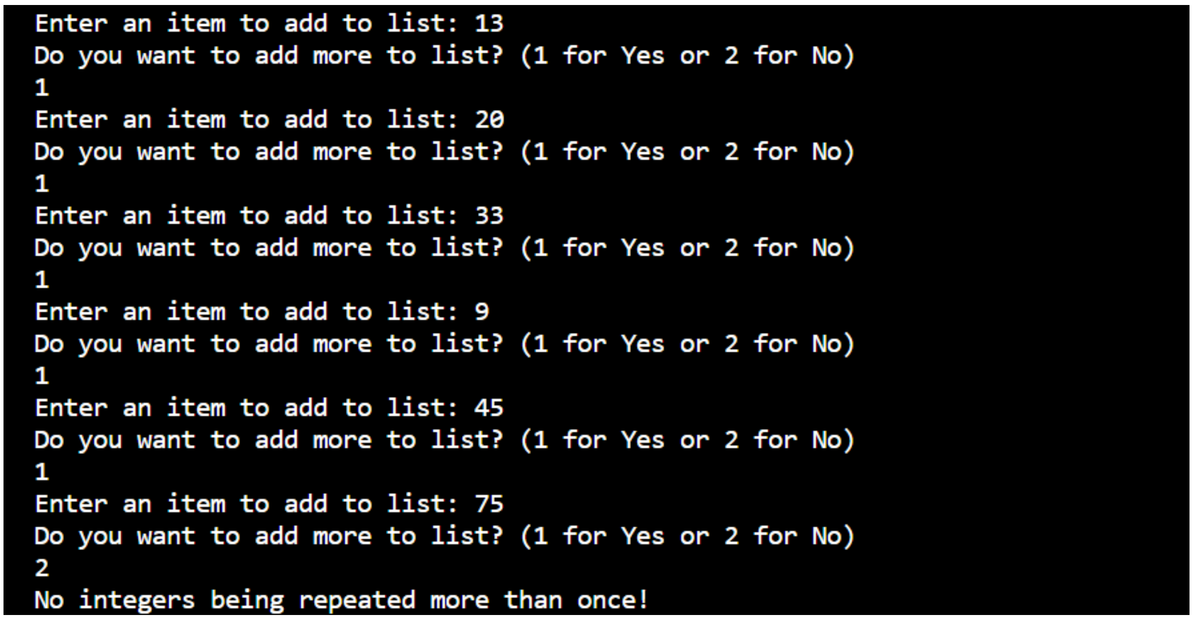
```
# An empty array used to add any integers that are repeated more than once
repeated_integers_list = []

# Nested for loop which compares the current element with the other elements
in the list
# and add element to repeated_integers_list if it exists more than once
for index1 in range(len(integers_list)):
    for index2 in range(index1+1, len(integers_list)):
        # Set currentElement to be the element currently being processed by
the loop
        currentElement = integers_list[index1]

        # Checks if the current element exists later on in the list and
        # if it does not already exist in repeated_integers_list
        if currentElement == integers_list[index2] and currentElement not in
repeated_integers_list:
            # Add element to list
            repeated_integers_list.append(currentElement)

# Checks if there are any elements in repeated_integer_list after loop
execution
# and prints out necessary information to the user
if not repeated_integers_list:
    print("No integers being repeated more than once!")
else:
    print(repeated_integers_list)
```

The program starts execution by declaring a variable called integers_list and its value is set to empty list. Then, it asks the user to enter a number to be added to the list. If the number is an integer, it is then added to the list. The user is then asked to choose whether he/she wants to add in more items to the list. If the choice entered by the user is an integer, it then checks to see whether it is valid. If it is '1', then the user is redirected to enter another number. Otherwise, if it is '2', the program exits iteration and proceeds with execution. If during this process, the user input was invalid, then the user gets an error message on the console and is redirected accordingly without the program terminating.

The program then declares another list variable called repeated_integers_list, which is used to store any repeated integers from the first list. It then traverses integers_list to check for any repeated integers. If it finds any, that number is added to repeated_integers_list. After execution, if the program did not detect any repeated integers, a message gets printed stating that no integers were repeated. Otherwise, the numbers are displayed to the user.

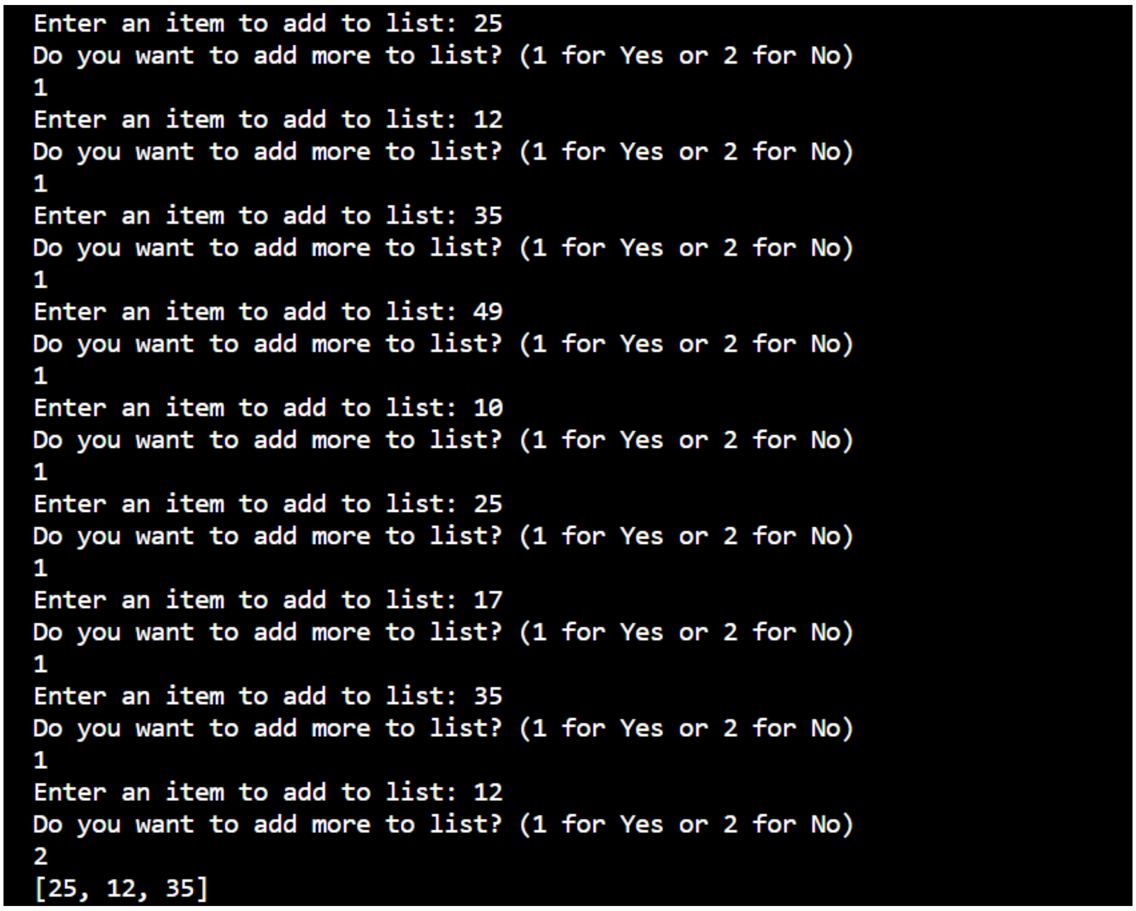
❖ How the Program was Tested + Screen Dumps

The program was first tested by having the user input different integer values. The numbers entered were 13, 20, 33, 9, 45 and 75. Also, the user input when asked to add more items to the list was valid. As expected, the program did not detect that any of the integers were repeated. Thus, the result was that a message was displayed on the console stating that no integers were repeated. This can be seen below:



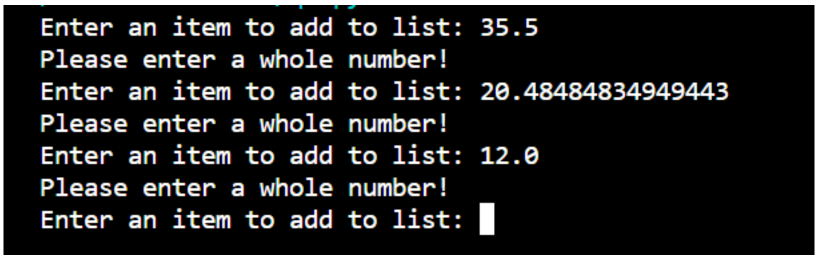
```
Enter an item to add to list: 13
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 20
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 33
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 9
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 45
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 75
Do you want to add more to list? (1 for Yes or 2 for No)
2
No integers being repeated more than once!
```

The program was then tested by inputting a list of integers which contain more than 1 integer which exists at least twice in the list. The numbers inputted were 25, 12, 35, 49, 10, 25, 17, 35 and 12 in that order. Also, the user input when asked to add more items to the list was valid. As expected, the program detected that there were several integers repeated. Thus, the program returned a list containing the repeated integers only, which was displayed on the console. This is shown below:



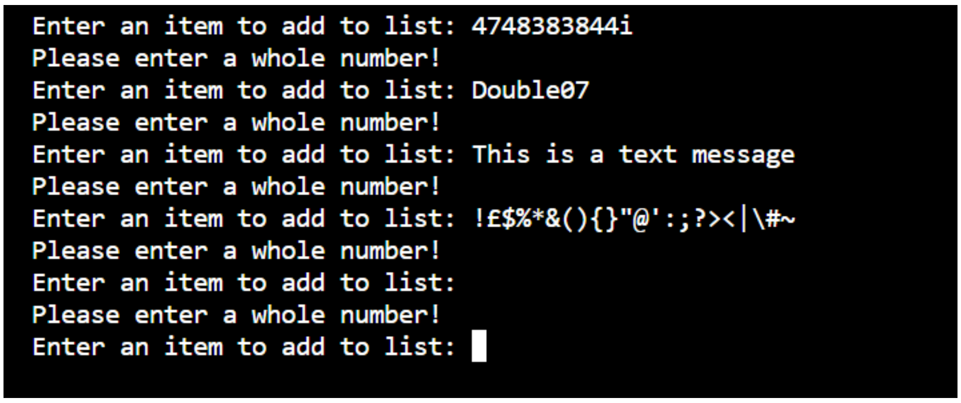
```
Enter an item to add to list: 25
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 12
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 35
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 49
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 10
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 25
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 17
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 35
Do you want to add more to list? (1 for Yes or 2 for No)
1
Enter an item to add to list: 12
Do you want to add more to list? (1 for Yes or 2 for No)
2
[25, 12, 35]
```

The program was tested for a third time by entering a set of float numbers. This was done to test the program's ability to distinguish a float from an integer. As expected, the program realised that such input was indeed invalid and thus a message was displayed prompting the user to enter a whole number, without the program asking the user whether to add more items to the list. This is shown below:



```
Enter an item to add to list: 35.5
Please enter a whole number!
Enter an item to add to list: 20.48484834949443
Please enter a whole number!
Enter an item to add to list: 12.0
Please enter a whole number!
Enter an item to add to list:
```

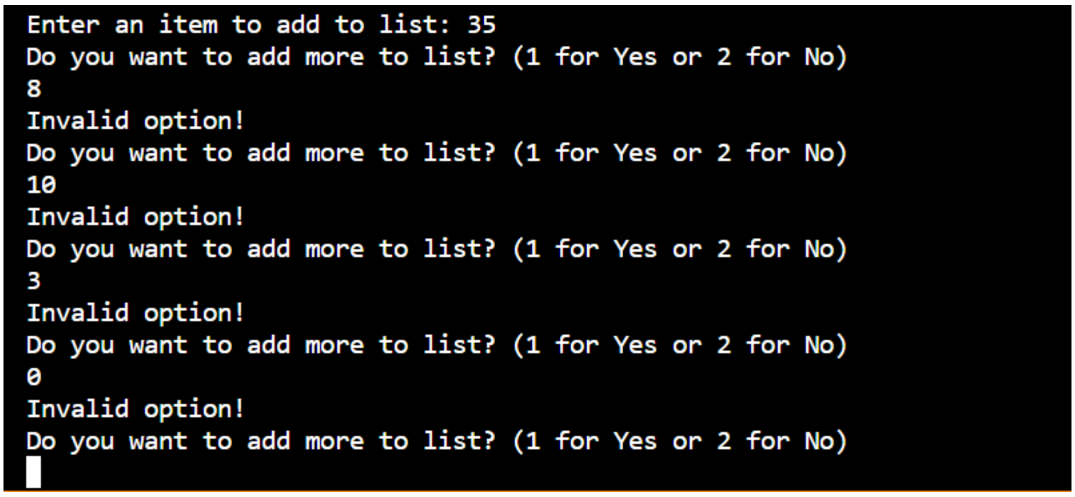
Afterwards, the program was tested by again restricting the user input, this time only for string values. The tested strings were divided in four parts, whole text, text with numbers included, empty values, and special characters. For all these instances, the console displayed a message telling the user to input an integer, as the program saw these user inputs as invalid, without the program asking the user whether to add more items to the list, as expected. This can be seen below:



```
Enter an item to add to list: 4748383844i
Please enter a whole number!
Enter an item to add to list: Double07
Please enter a whole number!
Enter an item to add to list: This is a text message
Please enter a whole number!
Enter an item to add to list: !£$%*&(){}"@':;?><|\#~
Please enter a whole number!
Enter an item to add to list:
Please enter a whole number!
Enter an item to add to list:
```

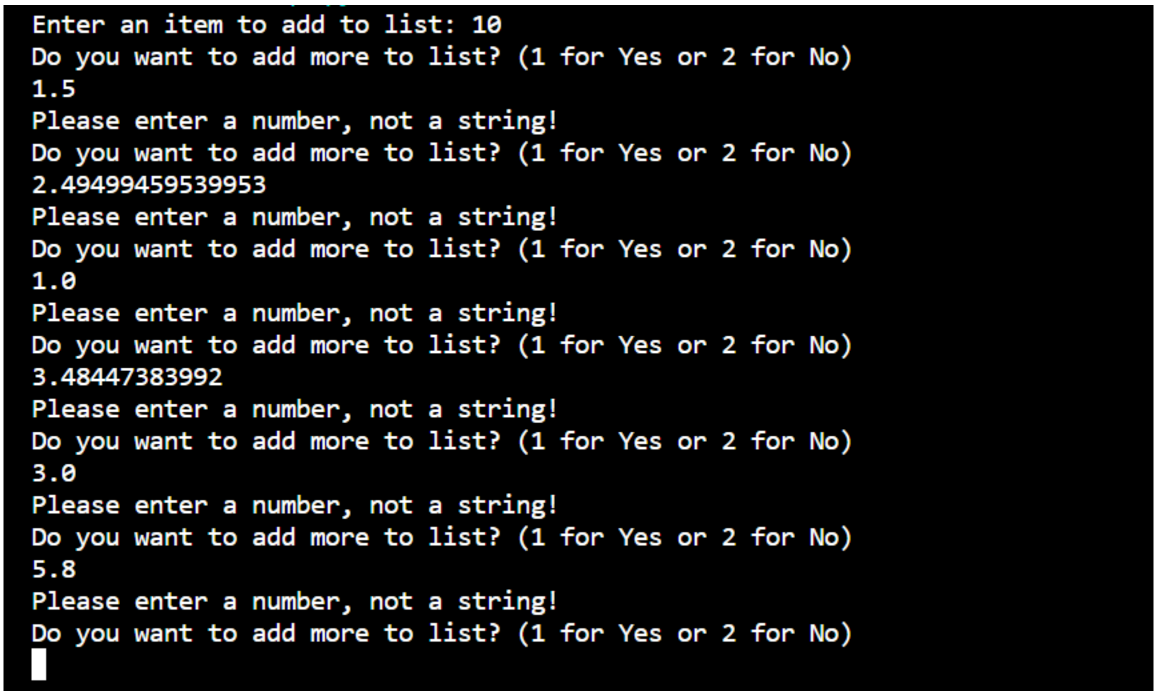
Once those were completed, the next few tests were on the user input when the user is asked whether to add more numbers to the list. These were only restricted to invalid values, as the valid values were tested previously. For all these instances, the number that was to be added to the list was valid.

The first test was done by letting the user enter any whole number other than 1 or 2, which are the only accepted numbers for the choice of user. As expected, the user detected that the value was not equal to either 1 or 2 and thus a message was printed out stating that the option entered was invalid. Afterwards, it prompts the user to enter a valid choice. This can be seen below:



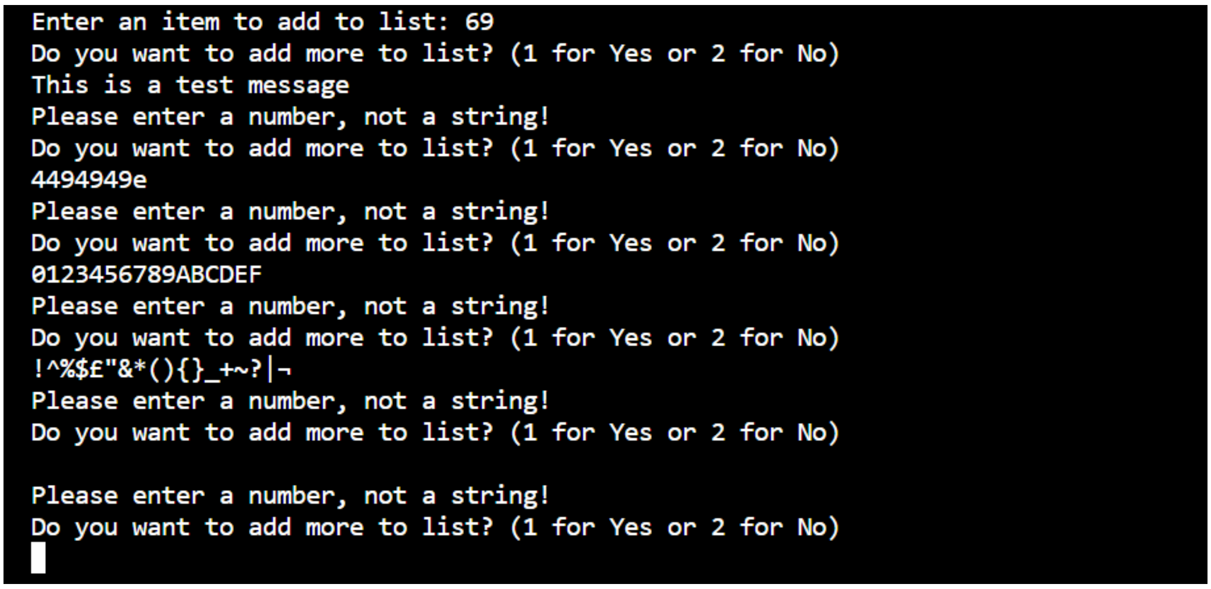
```
Enter an item to add to list: 35
Do you want to add more to list? (1 for Yes or 2 for No)
8
Invalid option!
Do you want to add more to list? (1 for Yes or 2 for No)
10
Invalid option!
Do you want to add more to list? (1 for Yes or 2 for No)
3
Invalid option!
Do you want to add more to list? (1 for Yes or 2 for No)
0
Invalid option!
Do you want to add more to list? (1 for Yes or 2 for No)
```

Afterwards, the user choice was tested by inputting float numbers only. This was done to test the program's ability to distinguish a float from an integer. As expected, the program realised that such input was indeed invalid and thus a message was displayed prompting the user to enter a whole number. This is shown below:



```
Enter an item to add to list: 10
Do you want to add more to list? (1 for Yes or 2 for No)
1.5
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
2.49499459539953
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
1.0
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
3.48447383992
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
3.0
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
5.8
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
```

The program was tested for a final time by entering a set of string values. The tested strings were divided in four parts, whole text, text with numbers included, empty values, and special characters. For all these instances, the console displayed a message telling the user to input a number, as the program saw these user inputs as invalid, without the program asking the user whether to add more items to the list, as expected. This can be seen below:



```
Enter an item to add to list: 69
Do you want to add more to list? (1 for Yes or 2 for No)
This is a test message
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
4494949e
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
0123456789ABCDEF
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
!^%$£"&*(){}_+~?|¬
Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)

Please enter a number, not a string!
Do you want to add more to list? (1 for Yes or 2 for No)
```

- Question 10
    - Source Code & Explanation

```
# Import sys package - used when calling findLargestNum() from main method (to
retrieve smallest integer number available)
# This helps deal with negative numbers
import sys

def main():
    # An array containing a list of numbers
    numbers_list = []

    has_stopped = False

    while not has_stopped:
        # Ask user to input an integer
        num = input("Enter an integer: ")

        # Check if user input is an integer
        if num.lstrip('-+').isnumeric():    # If True, add number to list
            numbers_list.append(int(num))

            # Check for valid option
            valid_choice = False

            while not valid_choice:
                # Ask user to decide whether to continue
                choice = input("Do you want to continue adding more numbers
(Y/N)?\n")

                if len(choice) > 0:
                    if choice[0].capitalize() == 'Y':   # valid option,
proceeds with entering next number
                        has_stopped = False
                        valid_choice = True
                    elif choice[0].capitalize() == 'N': # valid option, exits
entire iteration
                        has_stopped = True
                        valid_choice = True
                    else:   # invalid option, asks user to enter a valid
option
                        valid_choice = False
                        print("Please enter Y or N (Yes or No)!")
                else:
                    valid_choice = False
                    print("Please enter a value!")
        else:   # Number not an integer!
```

```
            print('Invalid input! Please enter an integer!')

    # Find largest number and store in variable called largestNum
    largestNum = findLargestNum(numbers_list, -sys.maxsize-1)

    # Printing the answer after function execution
    print("Largest Number =", largestNum)


# Function which finds the largest number from the given list in a recursive
manner
def findLargestNum(array, ans):
    # Checks if list is empty
    if not array:   # if True, return ans
        return ans
    else:   # if False, check if current element is larger than the number set
in ans
        if array[0] > ans:
            ans = array[0]

        # Remove first element from list (with index 0)
        array.pop(0)

        # Call findLargestNum() function - recursion
        return findLargestNum(array, ans)



# Call main() function to start execution
main()
```

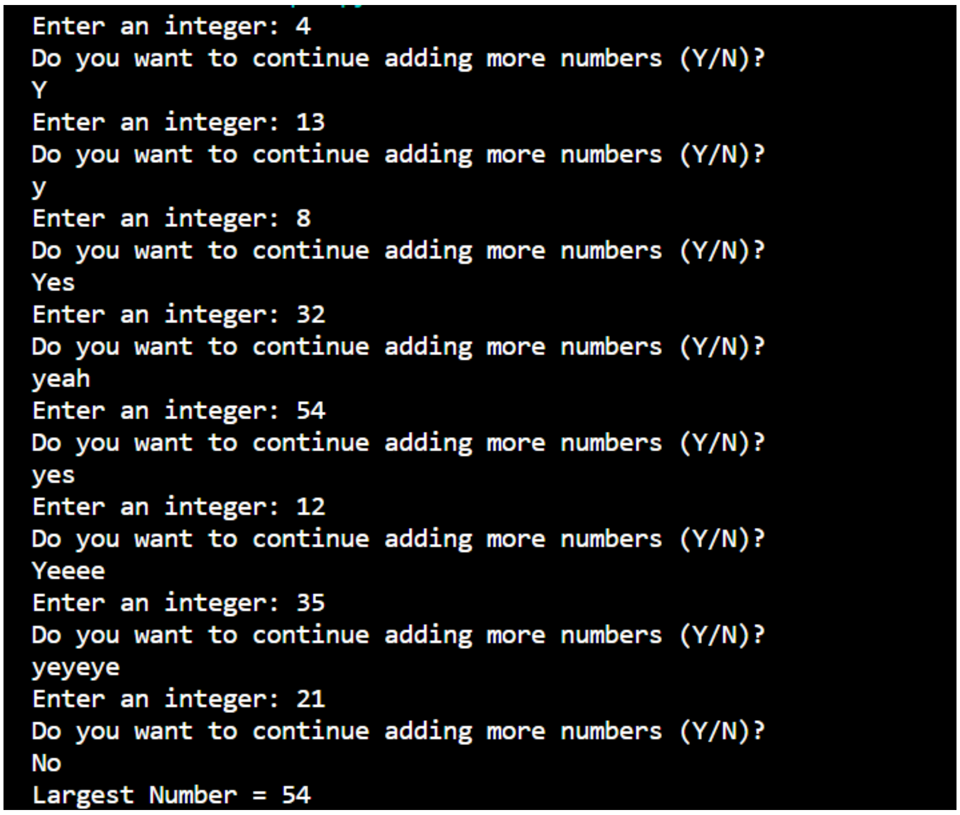
The program starts execution by declaring a variable called numbers_list and its value is set to empty list. Then, it asks the user to enter a number to be added to the list. If the number is an integer, it is then added to the list. The user is then asked to choose whether he/she wants to add in more items to the list. If the choice entered by the user is valid, it then checks to see whether it is a ‘Y’ or an ‘N’. If it is a ‘Y’, then the user is redirected to enter another number. Otherwise, if it is an ‘N’, the program exits iteration and proceeds with execution. If during this process, the user input was invalid, then the user gets an error message on the console and is redirected accordingly.

Once completed, the findLargestNum() function is called and the final answer is stored in a variable called largestNum. The second argument passed on when the function is called is set to be the smallest integer available in order to deal with negative numbers. The function finds the largest number in the list in a recursive manner. This is done by checking whether the array is empty. If it is not empty, the first element in the list is compared with the current largest number stored in ans to determine whether the element is larger than the value in ans, and then removing the element from the list. The answer is then printed out to the user.

❖ How the Program was Tested + Screen Dumps

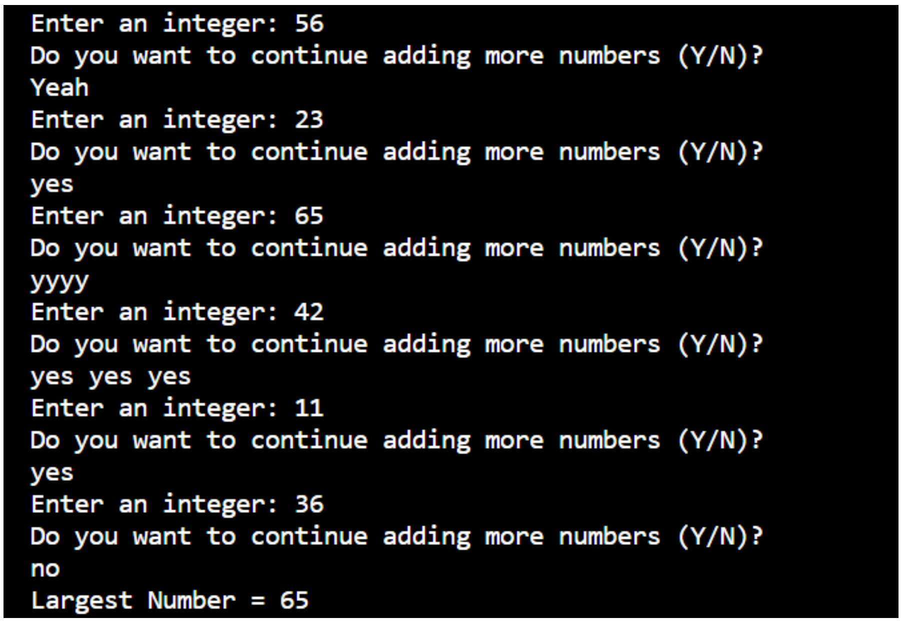
The first test done on the program was made by entering two sets of positive numbers. The two lists of numbers that were tested were [4, 13, 8, 32, 54, 12, 35, 21] and [56, 23, 65, 42, 11, 36]. As expected, the largest numbers from both lists, which were 54 and 65 respectively, were retrieved and displayed to the user. This can be seen as follows:

List 1: Inputting values and Answer Output



```
Enter an integer: 4
Do you want to continue adding more numbers (Y/N)?
Y
Enter an integer: 13
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: 8
Do you want to continue adding more numbers (Y/N)?
Yes
Enter an integer: 32
Do you want to continue adding more numbers (Y/N)?
yeah
Enter an integer: 54
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: 12
Do you want to continue adding more numbers (Y/N)?
Yeeee
Enter an integer: 35
Do you want to continue adding more numbers (Y/N)?
yeyeye
Enter an integer: 21
Do you want to continue adding more numbers (Y/N)?
No
Largest Number = 54
```

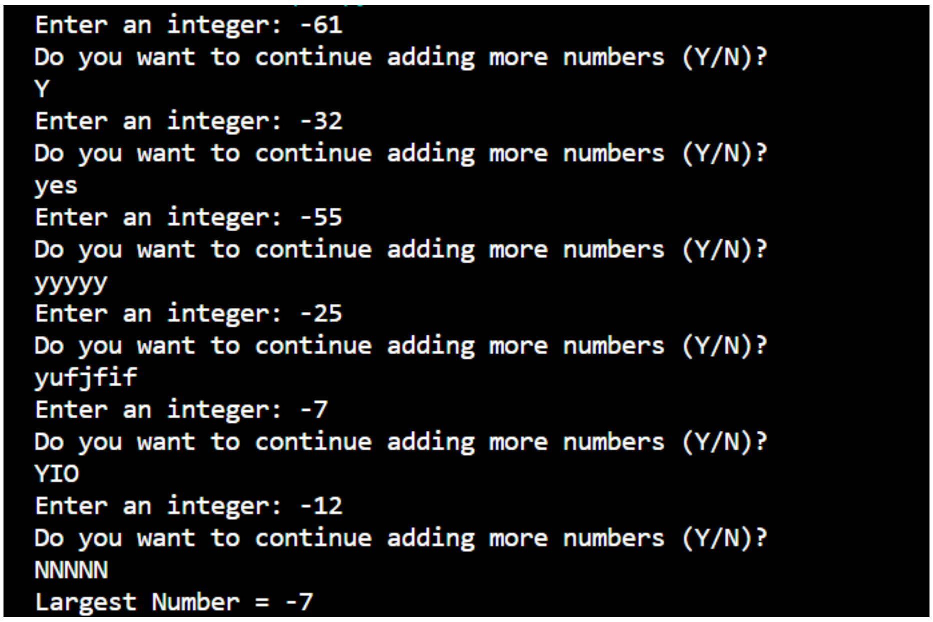
List 2: Inputting values and Answer Output



```
Enter an integer: 56
Do you want to continue adding more numbers (Y/N)?
Yeah
Enter an integer: 23
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: 65
Do you want to continue adding more numbers (Y/N)?
yyyy
Enter an integer: 42
Do you want to continue adding more numbers (Y/N)?
yes yes yes
Enter an integer: 11
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: 36
Do you want to continue adding more numbers (Y/N)?
no
Largest Number = 65
```

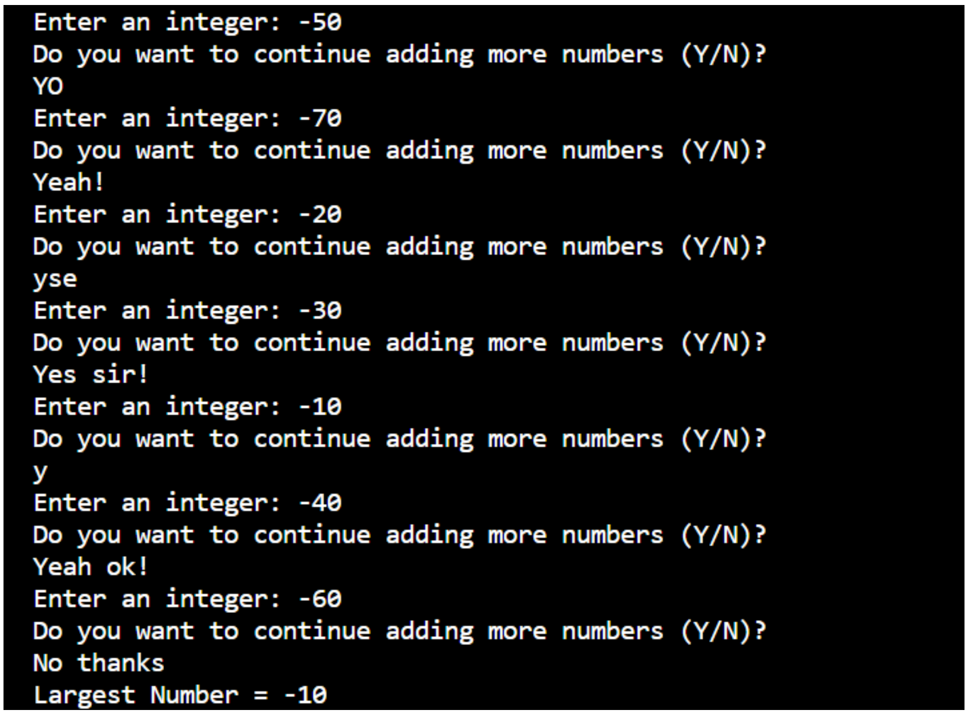
The program was then tested by entering two sets of negative numbers. This was done to test the program's ability to deal with negative numbers in the same way as positive numbers. The two lists of numbers that were tested were [-61, -32, -55, -25, -7, -12] and [-50, -70, -20, -30, -10, -40, -60]. As expected, the largest numbers from both lists, which in this case were -7 and -10 respectively, were retrieved and displayed to the user. This can be seen as follows:

List 1: Inputting values and Answer Output



```
Enter an integer: -61
Do you want to continue adding more numbers (Y/N)?
Y
Enter an integer: -32
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: -55
Do you want to continue adding more numbers (Y/N)?
yyyyy
Enter an integer: -25
Do you want to continue adding more numbers (Y/N)?
yufjfif
Enter an integer: -7
Do you want to continue adding more numbers (Y/N)?
YIO
Enter an integer: -12
Do you want to continue adding more numbers (Y/N)?
NNNNN
Largest Number = -7
```

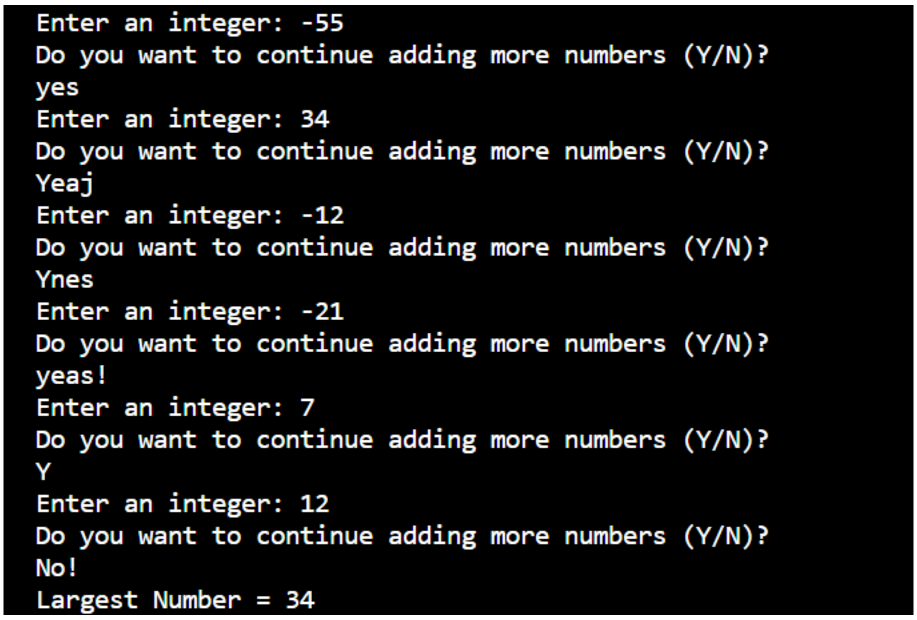
List 2: Inputting values and Answer Output



```
Enter an integer: -50
Do you want to continue adding more numbers (Y/N)?
YO
Enter an integer: -70
Do you want to continue adding more numbers (Y/N)?
Yeah!
Enter an integer: -20
Do you want to continue adding more numbers (Y/N)?
yse
Enter an integer: -30
Do you want to continue adding more numbers (Y/N)?
Yes sir!
Enter an integer: -10
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: -40
Do you want to continue adding more numbers (Y/N)?
Yeah ok!
Enter an integer: -60
Do you want to continue adding more numbers (Y/N)?
No thanks
Largest Number = -10
```

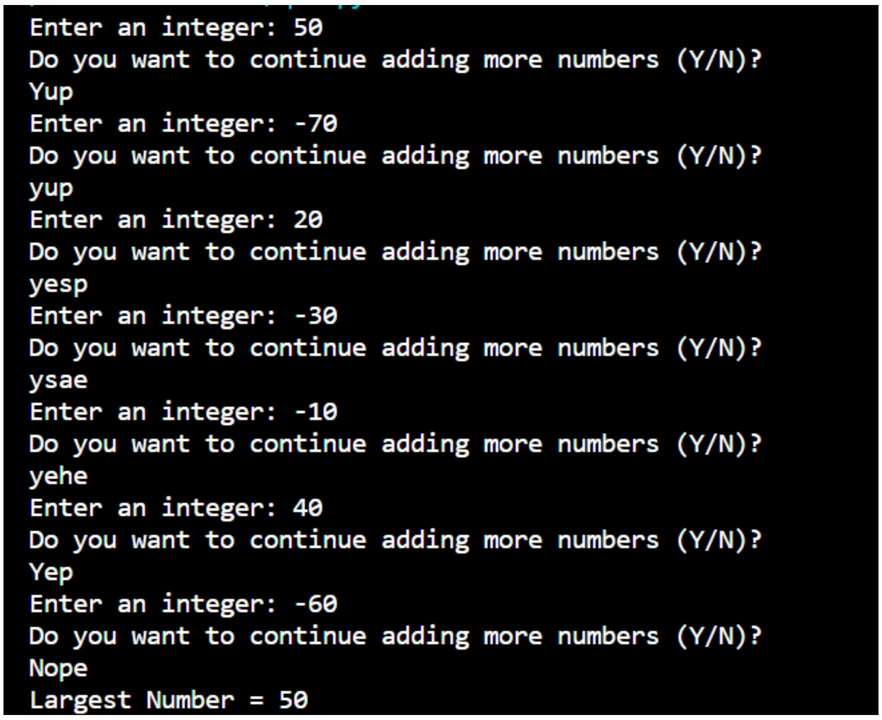
The program was then tested for a third time by entering two sets of numbers such that there is a mix between negative and positive numbers. This was done to test the program's ability to distinguish between the two types of numbers. The two lists of numbers that were tested were [-55, 34, -12, -21, 7, 12] and [50, -70, 20, -30, -10, 40, -60]. As expected, the largest numbers from both lists, which in this case were 34 and 50 respectively, were retrieved and displayed to the user. This can be seen as follows:

List 1: Inputting values and Answer Output



```
Enter an integer: -55
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: 34
Do you want to continue adding more numbers (Y/N)?
Yeaj
Enter an integer: -12
Do you want to continue adding more numbers (Y/N)?
Ynes
Enter an integer: -21
Do you want to continue adding more numbers (Y/N)?
yeas!
Enter an integer: 7
Do you want to continue adding more numbers (Y/N)?
Y
Enter an integer: 12
Do you want to continue adding more numbers (Y/N)?
No!
Largest Number = 34
```

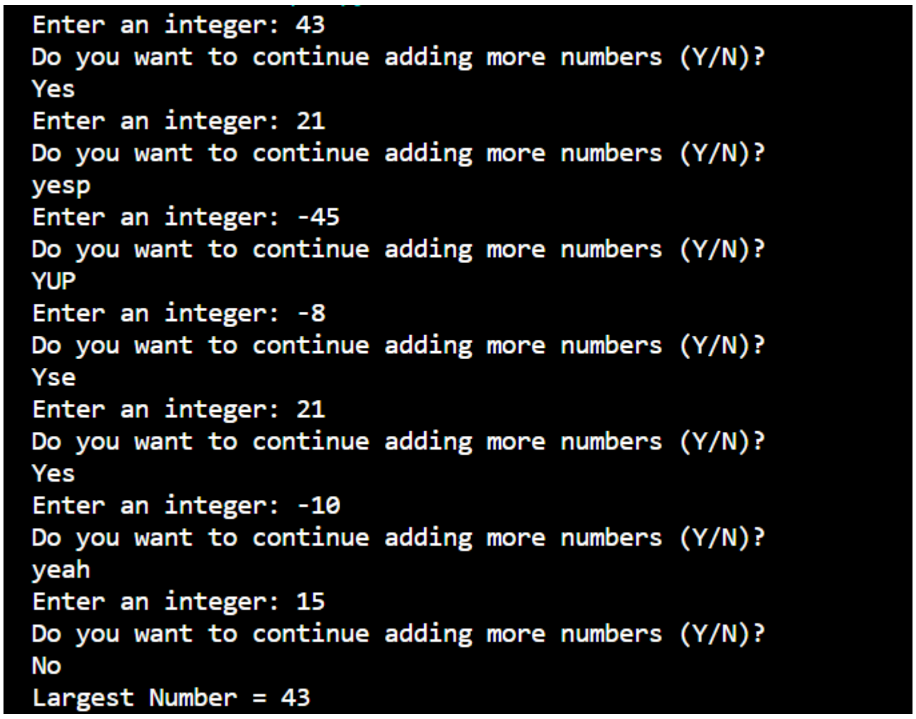
List 2: Inputting values and Answer Output



```
Enter an integer: 50
Do you want to continue adding more numbers (Y/N)?
Yup
Enter an integer: -70
Do you want to continue adding more numbers (Y/N)?
yup
Enter an integer: 20
Do you want to continue adding more numbers (Y/N)?
yesp
Enter an integer: -30
Do you want to continue adding more numbers (Y/N)?
ysae
Enter an integer: -10
Do you want to continue adding more numbers (Y/N)?
yehe
Enter an integer: 40
Do you want to continue adding more numbers (Y/N)?
Yep
Enter an integer: -60
Do you want to continue adding more numbers (Y/N)?
Nope
Largest Number = 50
```

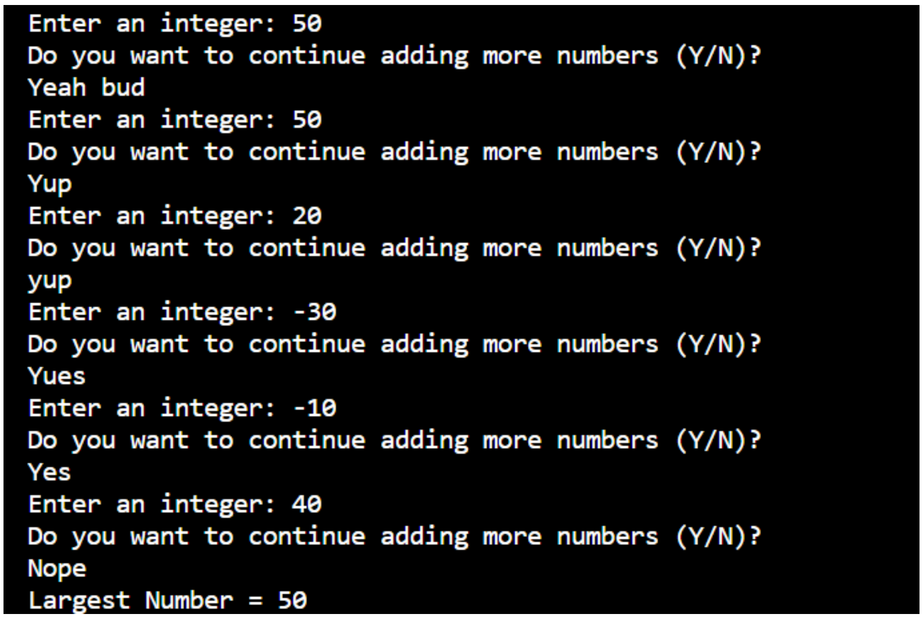
The program was then tested for a third time by entering three sets of numbers such that there are integers that are repeated. This was done to check for any effects caused by repeated integers. For all lists, both positive and negative numbers were entered. The four lists of numbers that were tested were [43, 21, -45, -8, 21, -10, 15], [50, 50, 20, -30, -10, 40], [-32, -45, -24, -75, -32, -9] and [-34, -10, -10, -29, -50, -21]. The result was that, again, the largest numbers from both lists, which in this case were 43, 50, -9 and -10 respectively, were retrieved and displayed to the user. Thus, there were no effects caused by repeated integers. This can be seen as follows:

List 1: Inputting values and Answer Output



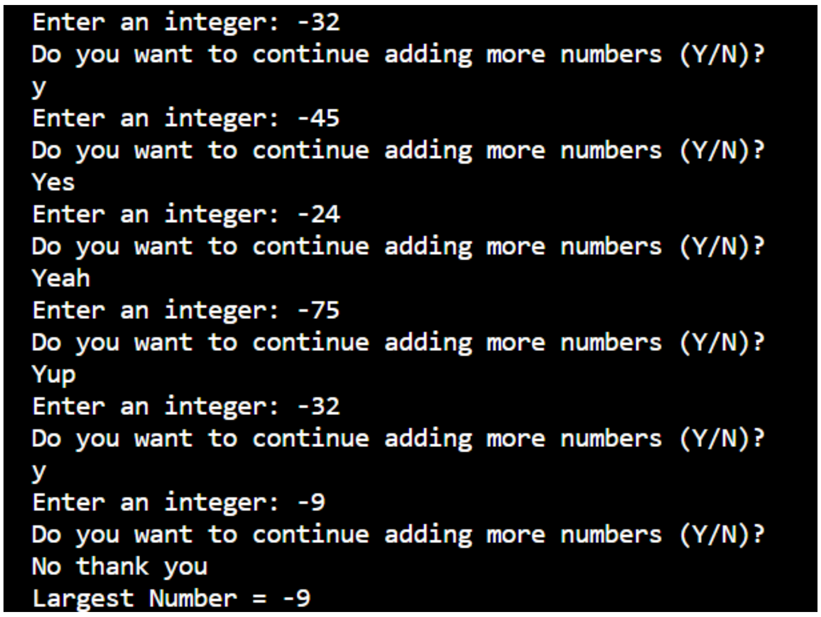
```
Enter an integer: 43
Do you want to continue adding more numbers (Y/N)?
Yes
Enter an integer: 21
Do you want to continue adding more numbers (Y/N)?
yesp
Enter an integer: -45
Do you want to continue adding more numbers (Y/N)?
YUP
Enter an integer: -8
Do you want to continue adding more numbers (Y/N)?
Yse
Enter an integer: 21
Do you want to continue adding more numbers (Y/N)?
Yes
Enter an integer: -10
Do you want to continue adding more numbers (Y/N)?
yeah
Enter an integer: 15
Do you want to continue adding more numbers (Y/N)?
No
Largest Number = 43
```

List 2: Inputting values and Answer Output



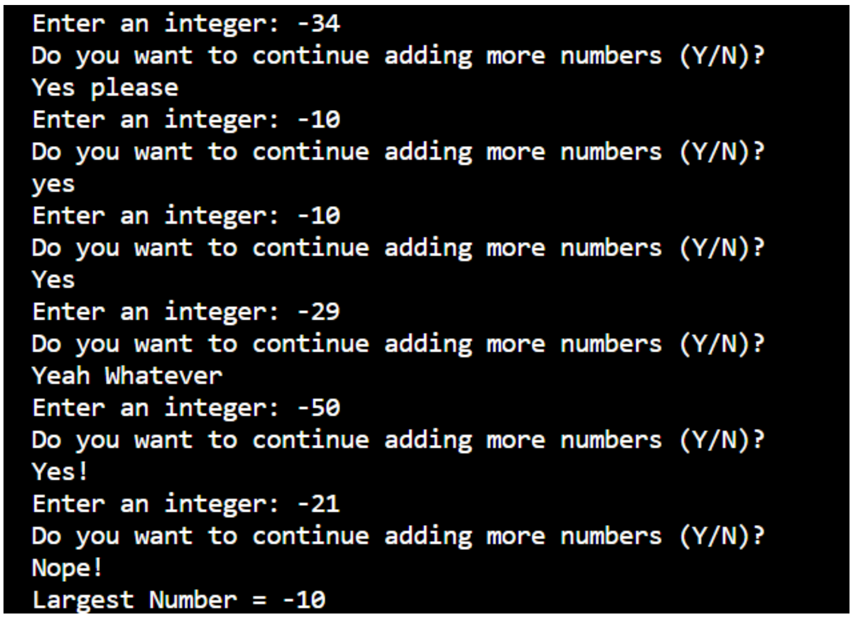
```
Enter an integer: 50
Do you want to continue adding more numbers (Y/N)?
Yeah bud
Enter an integer: 50
Do you want to continue adding more numbers (Y/N)?
Yup
Enter an integer: 20
Do you want to continue adding more numbers (Y/N)?
yup
Enter an integer: -30
Do you want to continue adding more numbers (Y/N)?
Yues
Enter an integer: -10
Do you want to continue adding more numbers (Y/N)?
Yes
Enter an integer: 40
Do you want to continue adding more numbers (Y/N)?
Nope
Largest Number = 50
```

List 3: Inputting values and Answer Output



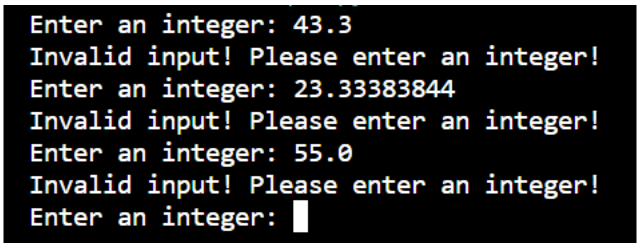
```
Enter an integer: -32
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: -45
Do you want to continue adding more numbers (Y/N)?
Yes
Enter an integer: -24
Do you want to continue adding more numbers (Y/N)?
Yeah
Enter an integer: -75
Do you want to continue adding more numbers (Y/N)?
Yup
Enter an integer: -32
Do you want to continue adding more numbers (Y/N)?
y
Enter an integer: -9
Do you want to continue adding more numbers (Y/N)?
No thank you
Largest Number = -9
```

List 4: Inputting values and Answer Output



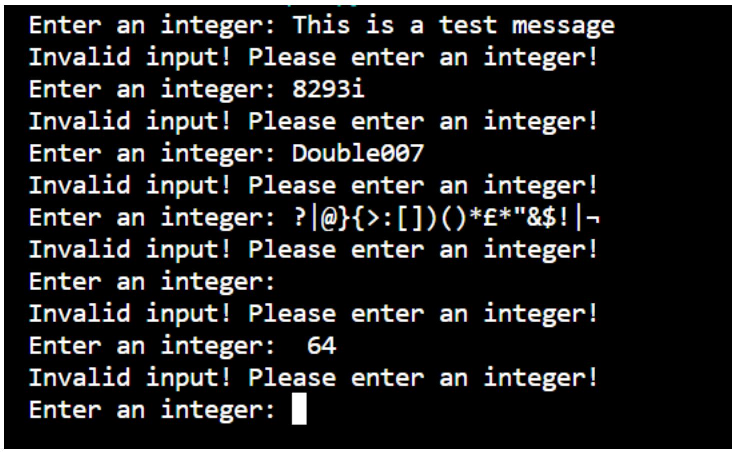
```
Enter an integer: -34
Do you want to continue adding more numbers (Y/N)?
Yes please
Enter an integer: -10
Do you want to continue adding more numbers (Y/N)?
yes
Enter an integer: -10
Do you want to continue adding more numbers (Y/N)?
Yes
Enter an integer: -29
Do you want to continue adding more numbers (Y/N)?
Yeah Whatever
Enter an integer: -50
Do you want to continue adding more numbers (Y/N)?
Yes!
Enter an integer: -21
Do you want to continue adding more numbers (Y/N)?
Nope!
Largest Number = -10
```

The program was then tested by trying to add float values to the list. This was done to test the program's ability to distinguish floats from integers. As expected, the program saw this type of input as invalid and the user was prompted to enter an integer. This can be seen below:



```
Enter an integer: 43.3
Invalid input! Please enter an integer!
Enter an integer: 23.33383844
Invalid input! Please enter an integer!
Enter an integer: 55.0
Invalid input! Please enter an integer!
Enter an integer: 
```

Afterwards, the program was tested by trying to add string values to the list. For the last test, a space was added in before the number was inputted. Once again, the program saw this input as invalid and again prompts the user to enter an integer. This is shown below as follows:



```
Enter an integer: This is a test message
Invalid input! Please enter an integer!
Enter an integer: 8293i
Invalid input! Please enter an integer!
Enter an integer: Double007
Invalid input! Please enter an integer!
Enter an integer: ?|@}{>:[])()*£*"&$!|¬
Invalid input! Please enter an integer!
Enter an integer:
Invalid input! Please enter an integer!
Enter an integer:  64
Invalid input! Please enter an integer!
Enter an integer: 
```

The next few tests were done specifically to test invalid user input whenever the user is asked to choose whether he would like to add new numbers to the list. These tests were done in a similar manner to previous programs, ending with the very same results as before.

- Question 11
    - ❖ Source Code & Explanation

```python
# Importing math package - used to calculate xRadians and for Maclaurin's
series
import math


def main():
    xRadians = 0
    n = 0

    # Repeat until user chooses to exit the program
    option = 0
    while(option != 3):
        # Boolean variable to check for numerical overflow
        valid = False

        # Print menu to user and ask user to enter an option
        option = input("1. Compute sine\n2. Compute cosine\n3. Exit\nChoose an
option: ")
        print("\n")

        # Check if option is an integer
        if option.isdigit():
            option = int(option)
            # If user does not exit, ask the user to enter angle and number of
iterations
            if (option == 1) or (option == 2):
                # Check if number of iterations entered by user is valid
                while not valid:
                    x = input("Enter number for x (in degrees): ")

                    # Check if x is an integer
                    if x.lstrip('+-').replace(".","").isdigit():
                        index = x.find(".")
                        if index == -1: # if integer, convert current item
into int
                            x = int(x)
                        else:   # if float, convert current item into float
                            x = float(x)

                        # Converting user input to radians since Maclaurin's
expansion works with radians
                        xRadians = (x*math.pi)/180

                        # Ask user to input number of iterations
                        n = input("Enter number for n: ")

                        # Check if n is an integer
                        if n.lstrip('+-').isdigit():
                            n = int(n)
                            if option == 1:     # sin
                                if n >= 86: # numerical overflow
                                    print("Error! Entering such a high number 
can induce errors in calculation!\n")
                                    print("Please enter a value less than 
86!")
                                elif n <= 0:    # negative number or 0
                                    print("Error! Cannot perform calculation 
for negative number of iterations!\n")
                                    print("Please enter a value greater than 
0!")
                                else:   # valid
                                    valid = True
                            elif option == 2:   # cos
                                if n >= 87: # numerical overflow
                                    print("Error! Entering such a high number 
can induce errors in calculation!\n")
                                    print("Please enter a value less than 
87!")
                                elif n <= 0:    # negative number or 0
                                    print("Error! Cannot perform calculation 
for negative number of iterations!\n")
                                    print("Please enter a value greater than 
0!")
                                else:   # valid
                                    valid = True
                        else:
                            print("Please enter a number!")
                    else:
                        print("Please enter a number!")

                print("\n")

                # Call computeMaclaurins() function to compute sin() or cos()
                computeMacluarins(n, xRadians, option, x)

                print("\n")

            elif option == 3:   # Exit program
                print("Thanks for using this program!")
            else:   # Invalid option
                print("Invalid option!\n")
        else:
            print("Please enter a number!\n")


# Function which computes cosine or sine using Maclaurin's series expansion
def computeMacluarins(n, xRad, option, xDeg):
        ans = 0

        # Perform operation based on user's choice of option
        if option == 1:     # Calculate approximation of sin(x) using
Maclaurin's series
            for r in range(0,n):
                ans += ((-1)**r)*((xRad**((2*r)+1))/math.factorial(((2*r)+1)))
            print("sin("+str(xDeg)+") = ",round(ans,2))
        elif option == 2:   # Calculate approximation of cos(x) using
Maclaurin's series
            for r in range(0,n):
                ans += ((-1)**r)*((xRad**(2*r))/math.factorial(2*r))
            print("cos("+str(xDeg)+") = ",round(ans,2))


# Call main() function to start execution
main()
```

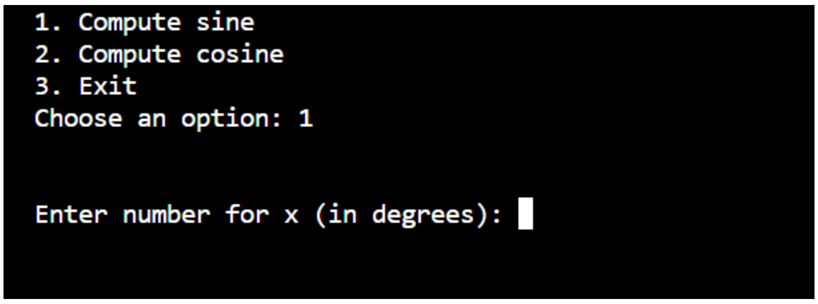
The program starts by printing out a menu in the console and asking the user to choose an option from the menu. If the option is valid, it then checks whether it is any one of the listed options. If the option is ‘1’ or ‘2’, the user is then prompted to enter the angle in degrees x and the number of iterations n to calculate sine or cosine respectively. The angles are then converted into radians by using math.pi, which is taken by using the imported package called math. If, when inputting the value for n, a situation arises when there might be a possible case of numerical overflow or when the value entered is 0 or less, the error is shown to the user and is asked to enter a new value for n. If the option is ‘3’, the program stops execution and a thank you message is shown to the user. If, during this whole operation, any user input is invalid, an error gets displayed on the console and the user is redirected accordingly.

Once everything is inputted correctly by the user, the program then calculates the value of sine or cosine for the angle set in x using Maclaurin’s series, depending on if the option was either ‘1’ or ‘2’ respectively. This is done by calling the function computeMaclaurins(). The answer is then displayed on the console.

❖ <u>How the Program was Tested + Screen Dumps</u>

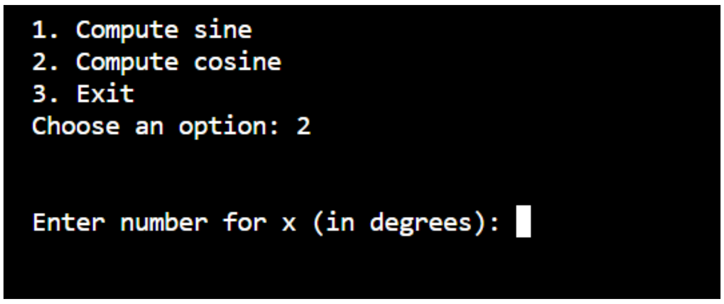
The first few tests that were done on the program were to test the option that is entered when the main menu is displayed. To start, the valid options were tested to make sure that they work correctly. As expected, whenever the option '1' or '2' is entered, the program prompts the user to enter a number x. Otherwise, if it was a '3', then the program terminates. This can be seen below:

<u>Option 1:</u>



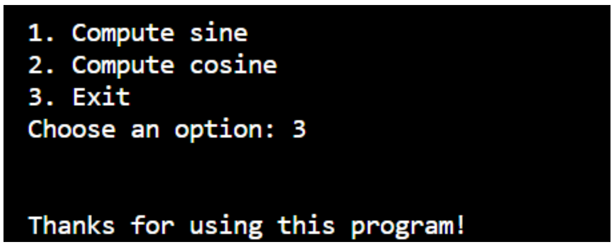
```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 1


Enter number for x (in degrees):
```

<u>Option 2:</u>



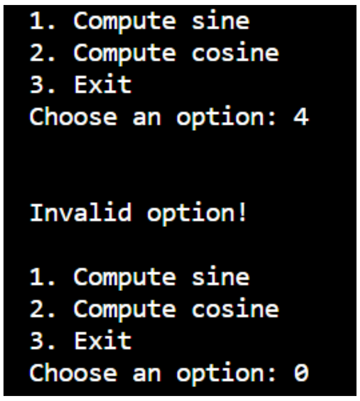
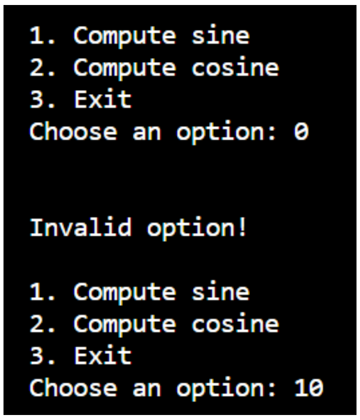
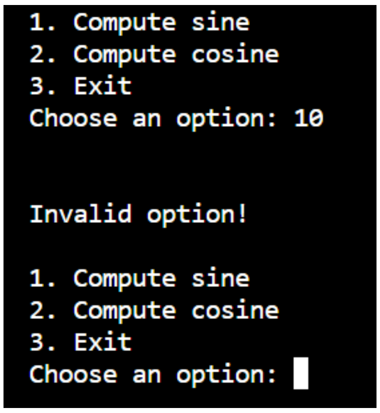
```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 2


Enter number for x (in degrees):
```

<u>Option 3:</u>



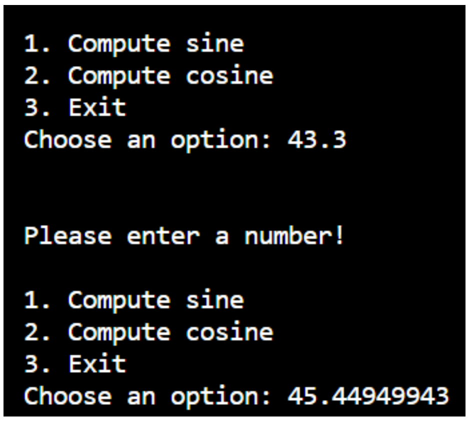
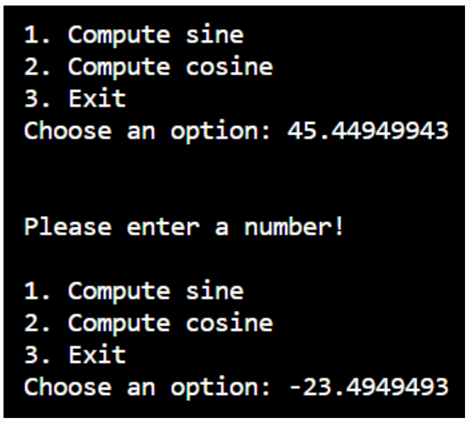
```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 3


Thanks for using this program!
```

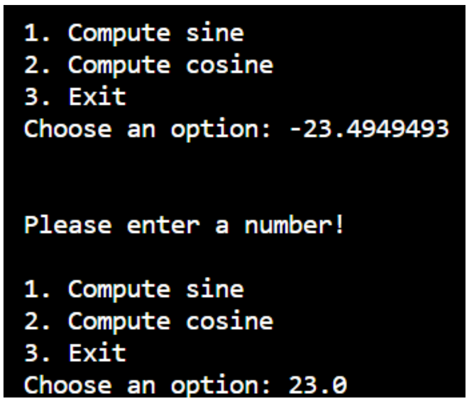
This value was tested again, this time by inputting several integers which are outside of the available options listed in the main menu. As expected, the program detected that the numbers were not any one of the listed options, and thus a message was displayed stating that the option was invalid, before redirecting the user back to the main menu. This can be seen below:

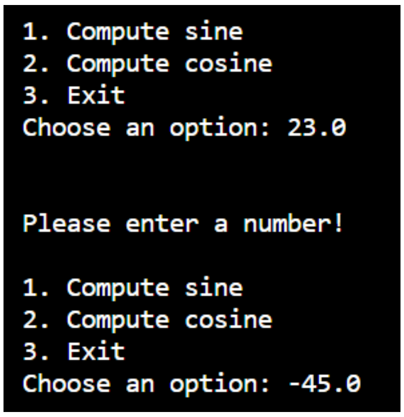

```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 4


Invalid option!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 0
```

```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 0


Invalid option!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 10
```

```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 10


Invalid option!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option:
```

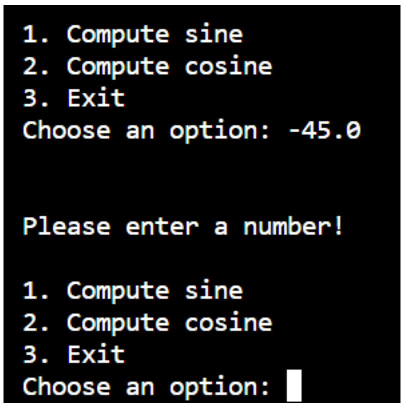
Finally, the value was tested by inputting several float and string values. This was done to test the program's ability to distinguish floats and strings from integers. As expected, the program detected such input as invalid, and thus a message was displayed on the console, prompting the user to enter a valid number. For the last string value, a space was added before the option was inputted. This can be seen below:

Float values

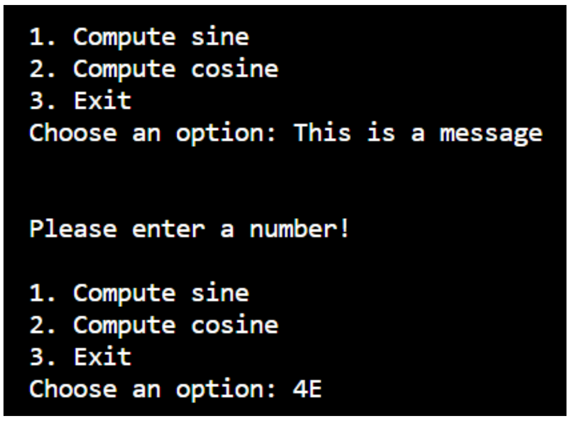

```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 43.3


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 45.44949943
```

```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 45.44949943


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option: -23.4949493
```

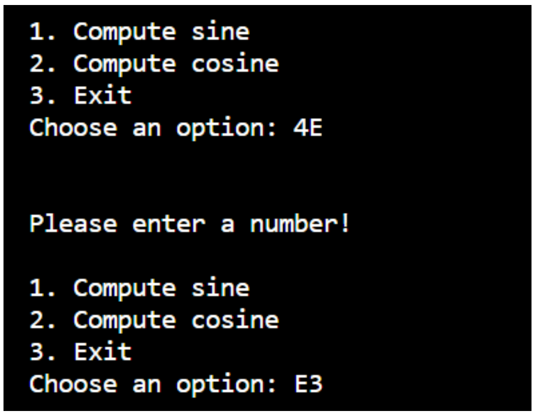

```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: -23.4949493


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 23.0
```

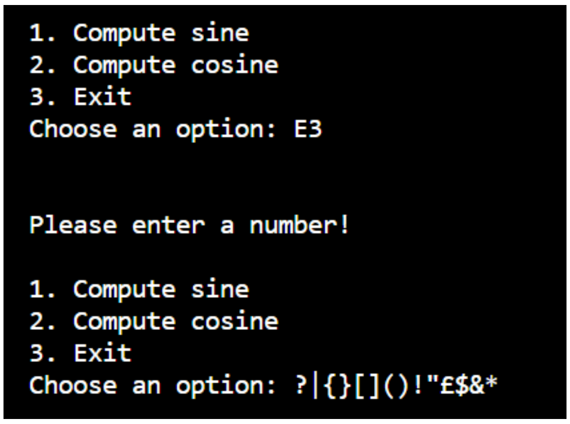

```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 23.0


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option: -45.0
```


```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: -45.0


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option:
```

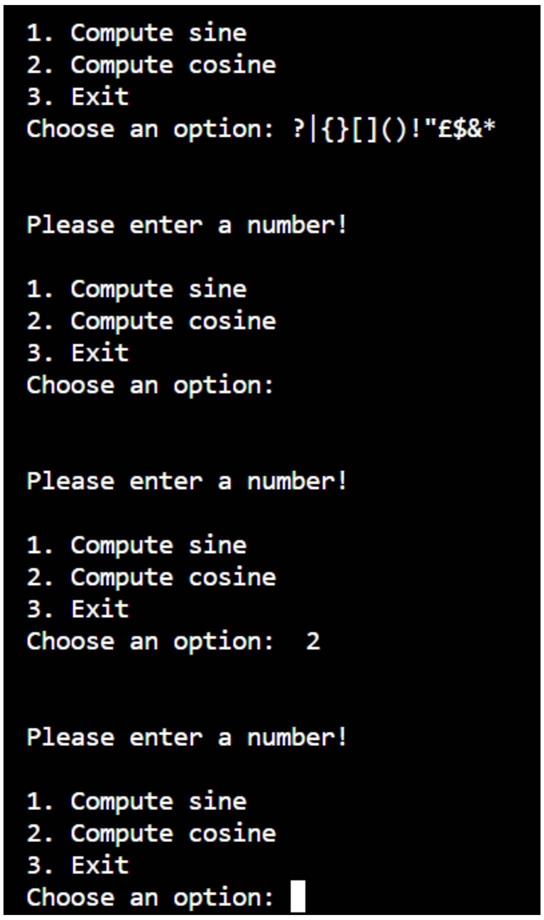
String values


```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: This is a message


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 4E
```


```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: 4E


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option: E3
```

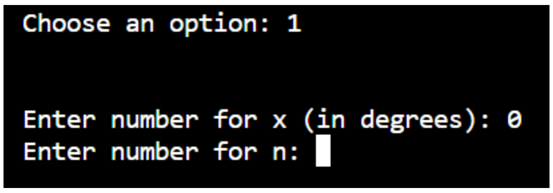


```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: E3


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option: ?|{}[]()!"£$&*
```

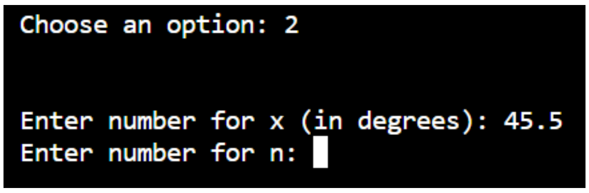


```
1. Compute sine
2. Compute cosine
3. Exit
Choose an option: ?|{}[]()!"£$&*


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option:


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option:   2


Please enter a number!

1. Compute sine
2. Compute cosine
3. Exit
Choose an option:
```

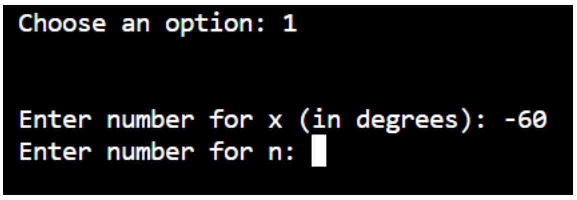
The next couple of tests that were done were made to test the input value for the number x. First, numerical user input was tested, which includes integers and floats. The result was that, as expected, the program proceeded by prompting the user to type a number n. This can be seen below:



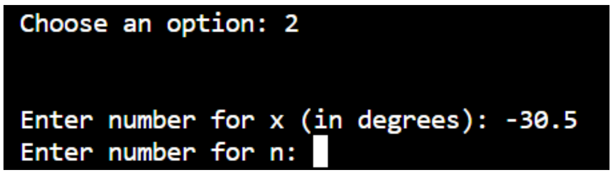
```
Choose an option: 1


Enter number for x (in degrees): 0
Enter number for n:
```



```
Choose an option: 2


Enter number for x (in degrees): 45.5
Enter number for n:
```

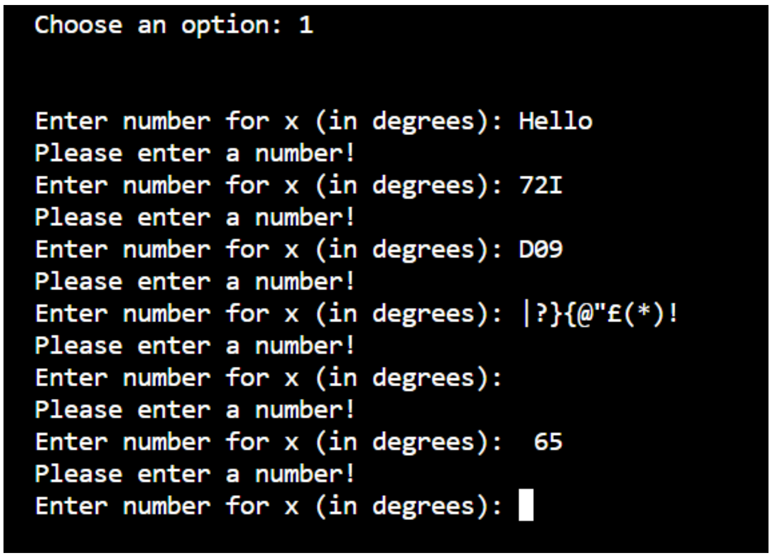


```
Choose an option: 1


Enter number for x (in degrees): -60
Enter number for n:
```

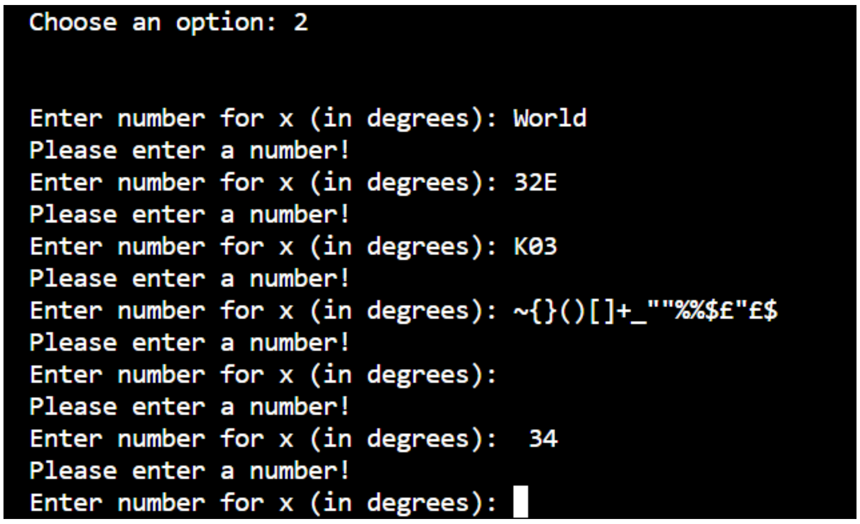


```
Choose an option: 2


Enter number for x (in degrees): -30.5
Enter number for n:
```

Afterwards, the number x was tested again, this time by trying to input a string value. This was done to test the program's ability to identify and distinguish characters from numbers. As expected, the program saw this type of user input as invalid and the user is redirected to enter a new value for x. For the last test for both options '1' and '2', a space was added before the number was entered. This can be seen below:

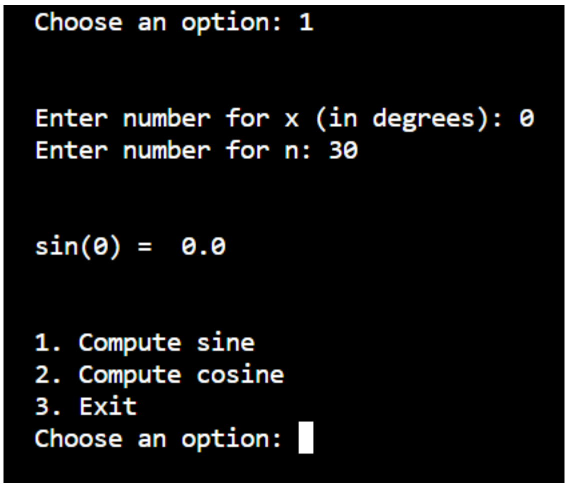


```
Choose an option: 1


Enter number for x (in degrees): Hello
Please enter a number!
Enter number for x (in degrees): 72I
Please enter a number!
Enter number for x (in degrees): D09
Please enter a number!
Enter number for x (in degrees): |?}{@"£(*)!
Please enter a number!
Enter number for x (in degrees):
Please enter a number!
Enter number for x (in degrees):  65
Please enter a number!
Enter number for x (in degrees):
```

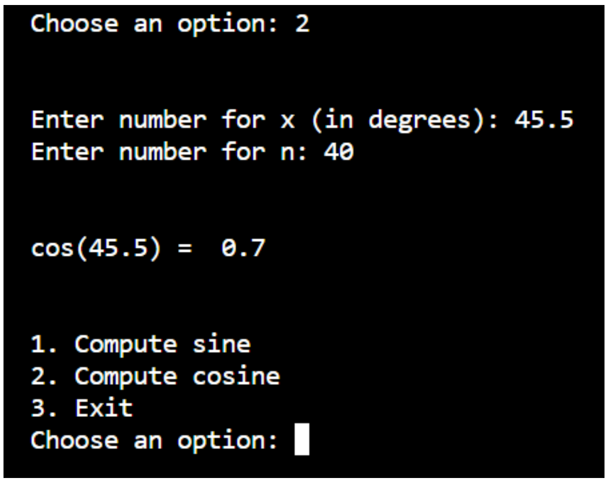


```
Choose an option: 2


Enter number for x (in degrees): World
Please enter a number!
Enter number for x (in degrees): 32E
Please enter a number!
Enter number for x (in degrees): K03
Please enter a number!
Enter number for x (in degrees): ~{}()[]+_""%%$£"£$
Please enter a number!
Enter number for x (in degrees):
Please enter a number!
Enter number for x (in degrees):  34
Please enter a number!
Enter number for x (in degrees):
```

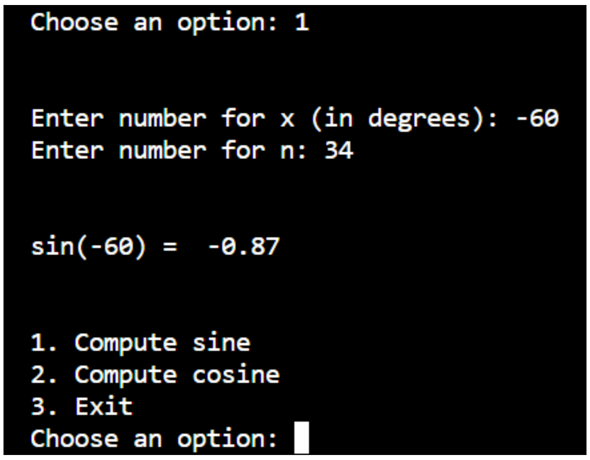
Afterwards, the very same tests were done, this time to test the input value for the number n. First, positive integer values were tested. For this set of tests, the valid numbers that were inputted when x was tested were used. As expected, the program computed the proper Maclaurin's series expansion given by x and n, and the answer was displayed on the console. Afterwards, the user is redirected to the main menu to choose another option of his choice. This can be seen as follows:



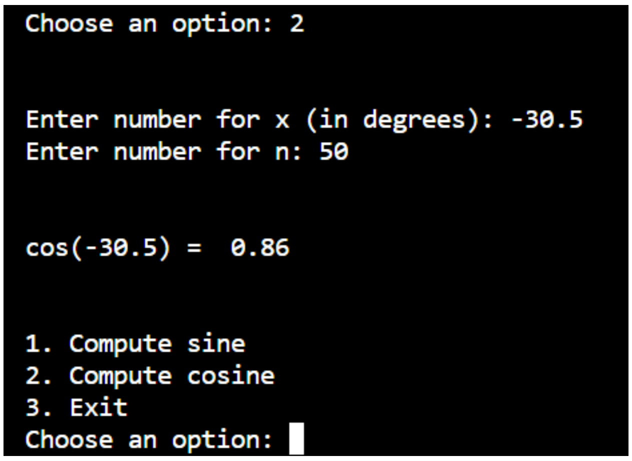
```
Choose an option: 1


Enter number for x (in degrees): 0
Enter number for n: 30


sin(0) =  0.0


1. Compute sine
2. Compute cosine
3. Exit
Choose an option:
```



```
Choose an option: 2


Enter number for x (in degrees): 45.5
Enter number for n: 40


cos(45.5) =  0.7


1. Compute sine
2. Compute cosine
3. Exit
Choose an option:
```

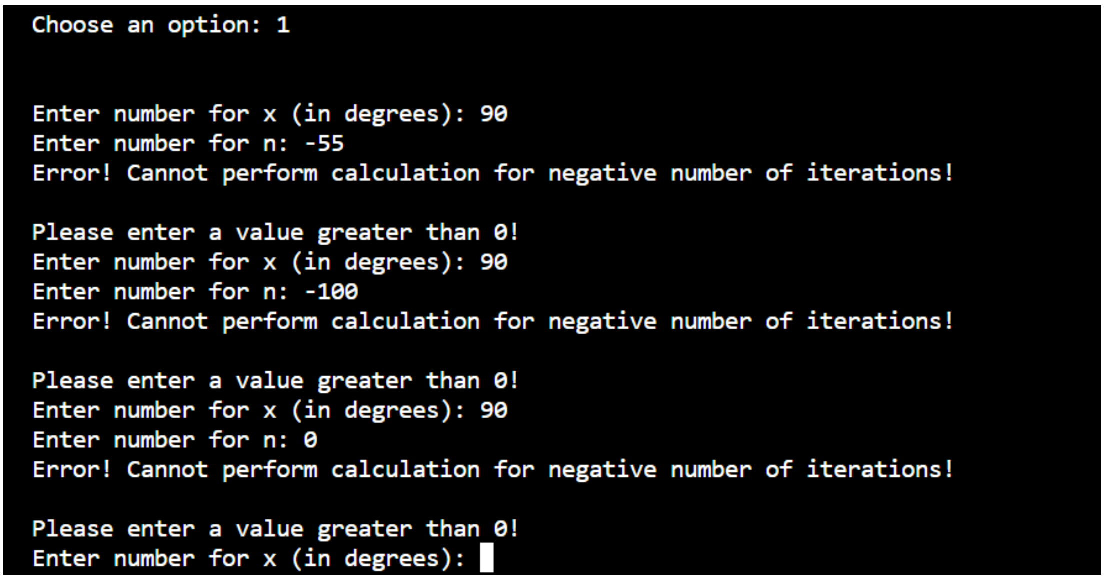


```
Choose an option: 1


Enter number for x (in degrees): -60
Enter number for n: 34


sin(-60) =  -0.87


1. Compute sine
2. Compute cosine
3. Exit
Choose an option:
```

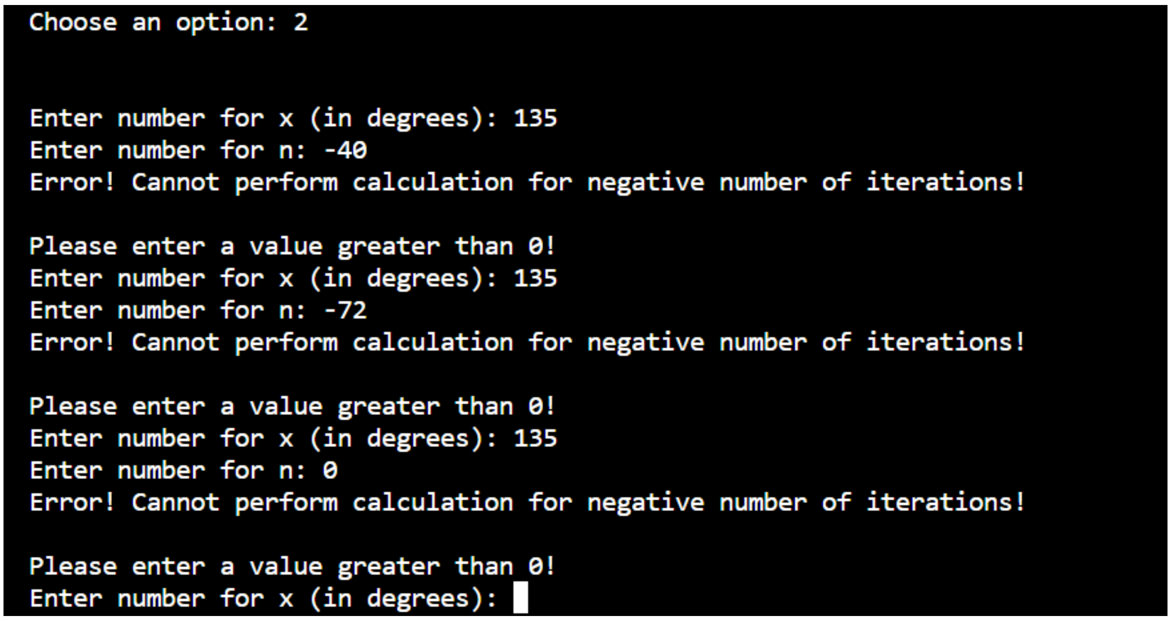


```
Choose an option: 2


Enter number for x (in degrees): -30.5
Enter number for n: 50


cos(-30.5) =  0.86


1. Compute sine
2. Compute cosine
3. Exit
Choose an option:
```

The number n was then tested by inputting a negative value or a 0. As expected, the program detected that it could not compute Maclaurin's when n is negative and thus, a message was displayed on the console stating that this was the case, and it prompts the user to enter a number which is greater than 0. However, the user is redirected to enter a new value for x, instead of doing so for n. This can be seen below:

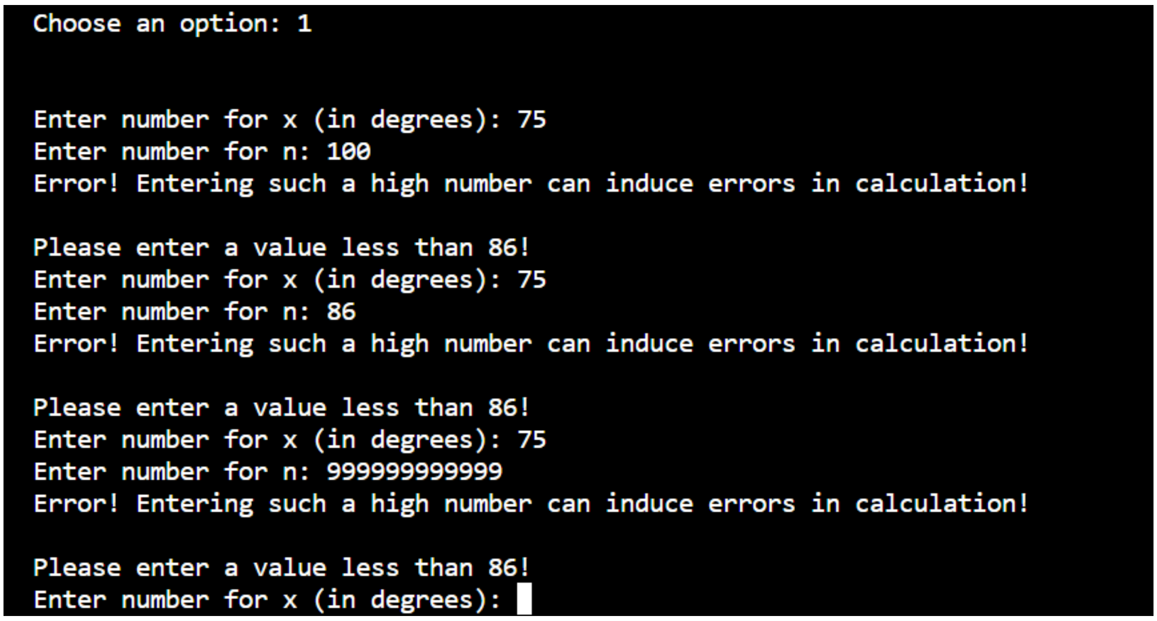


```
Choose an option: 1


Enter number for x (in degrees): 90
Enter number for n: -55
Error! Cannot perform calculation for negative number of iterations!

Please enter a value greater than 0!
Enter number for x (in degrees): 90
Enter number for n: -100
Error! Cannot perform calculation for negative number of iterations!

Please enter a value greater than 0!
Enter number for x (in degrees): 90
Enter number for n: 0
Error! Cannot perform calculation for negative number of iterations!

Please enter a value greater than 0!
Enter number for x (in degrees):
```

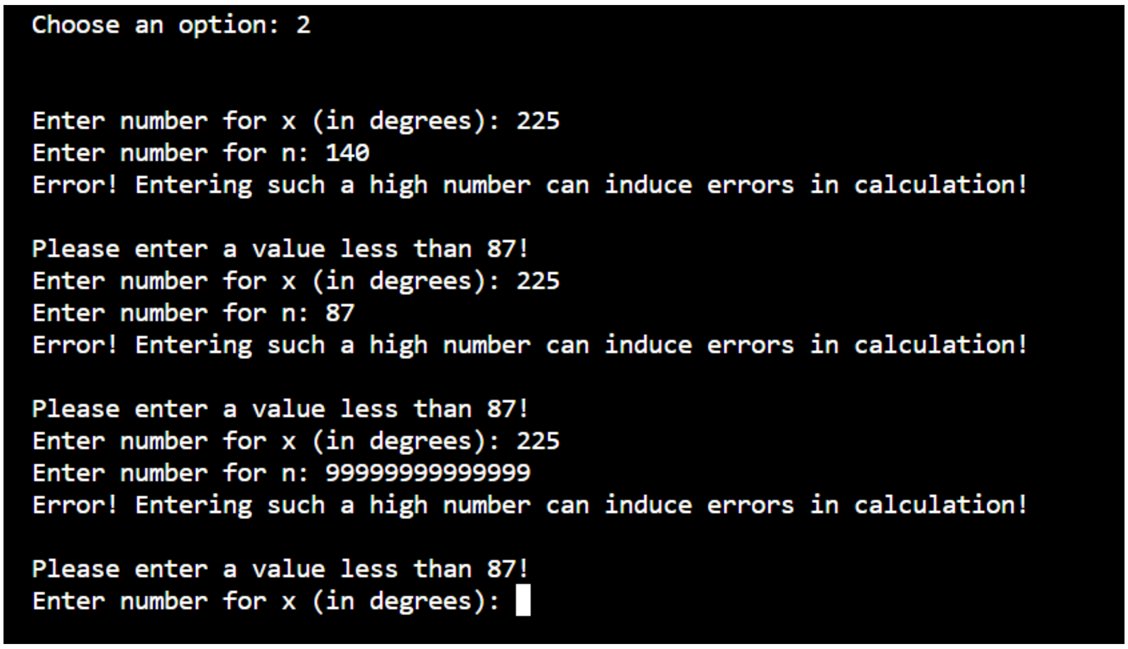


```
Choose an option: 2


Enter number for x (in degrees): 135
Enter number for n: -40
Error! Cannot perform calculation for negative number of iterations!

Please enter a value greater than 0!
Enter number for x (in degrees): 135
Enter number for n: -72
Error! Cannot perform calculation for negative number of iterations!

Please enter a value greater than 0!
Enter number for x (in degrees): 135
Enter number for n: 0
Error! Cannot perform calculation for negative number of iterations!

Please enter a value greater than 0!
Enter number for x (in degrees):
```

The value for n was tested for a third time by inputting a very large number, such that it is a valid integer. However, as expected, the program realised that it could lead to errors in computing Maclaurin's. Thus, a message was displayed, prompting the user to enter a smaller value, before redirecting him to enter a new value for x. This can be seen below as follows:



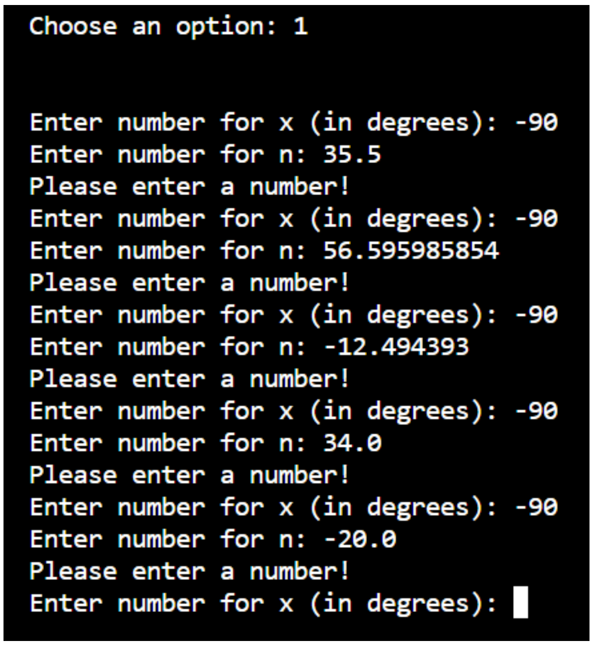
```
Choose an option: 1


Enter number for x (in degrees): 75
Enter number for n: 100
Error! Entering such a high number can induce errors in calculation!

Please enter a value less than 86!
Enter number for x (in degrees): 75
Enter number for n: 86
Error! Entering such a high number can induce errors in calculation!

Please enter a value less than 86!
Enter number for x (in degrees): 75
Enter number for n: 999999999999
Error! Entering such a high number can induce errors in calculation!

Please enter a value less than 86!
Enter number for x (in degrees):
```



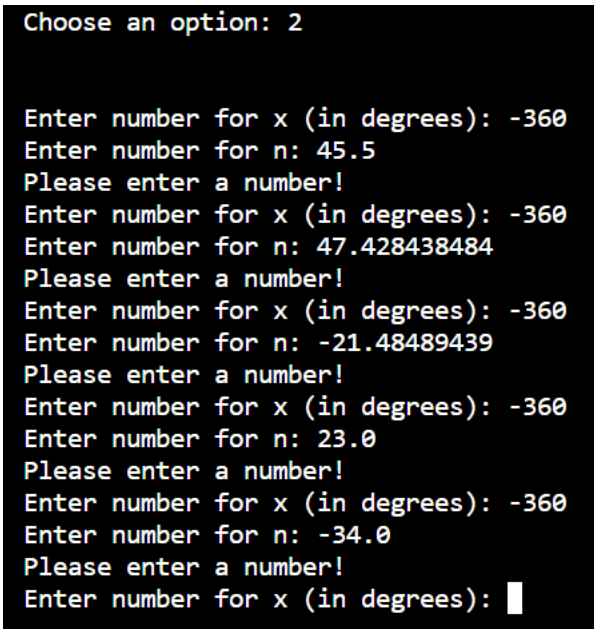
```
Choose an option: 2


Enter number for x (in degrees): 225
Enter number for n: 140
Error! Entering such a high number can induce errors in calculation!

Please enter a value less than 87!
Enter number for x (in degrees): 225
Enter number for n: 87
Error! Entering such a high number can induce errors in calculation!

Please enter a value less than 87!
Enter number for x (in degrees): 225
Enter number for n: 9999999999999
Error! Entering such a high number can induce errors in calculation!

Please enter a value less than 87!
Enter number for x (in degrees):
```

Lastly, the value for n was tested by inputting several float and string values. This was done to test the program's ability to distinguish both floats and strings from integers. As expected, the program detected that such input was invalid and thus, a message was printed out prompting the user to enter a number, before redirecting the user to enter a new value for x. For the last string value, a space was added before the number was inputted. This can be seen below:

Float values:

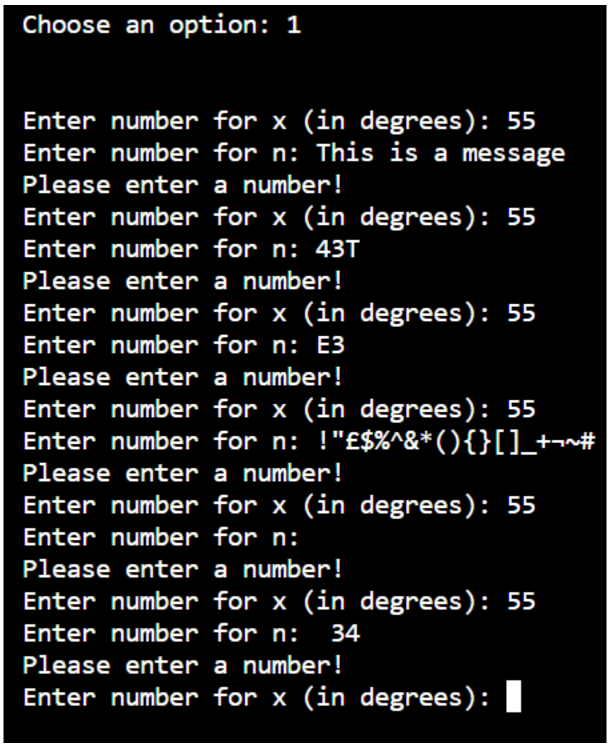


```
Choose an option: 1


Enter number for x (in degrees): -90
Enter number for n: 35.5
Please enter a number!
Enter number for x (in degrees): -90
Enter number for n: 56.595985854
Please enter a number!
Enter number for x (in degrees): -90
Enter number for n: -12.494393
Please enter a number!
Enter number for x (in degrees): -90
Enter number for n: 34.0
Please enter a number!
Enter number for x (in degrees): -90
Enter number for n: -20.0
Please enter a number!
Enter number for x (in degrees):
```

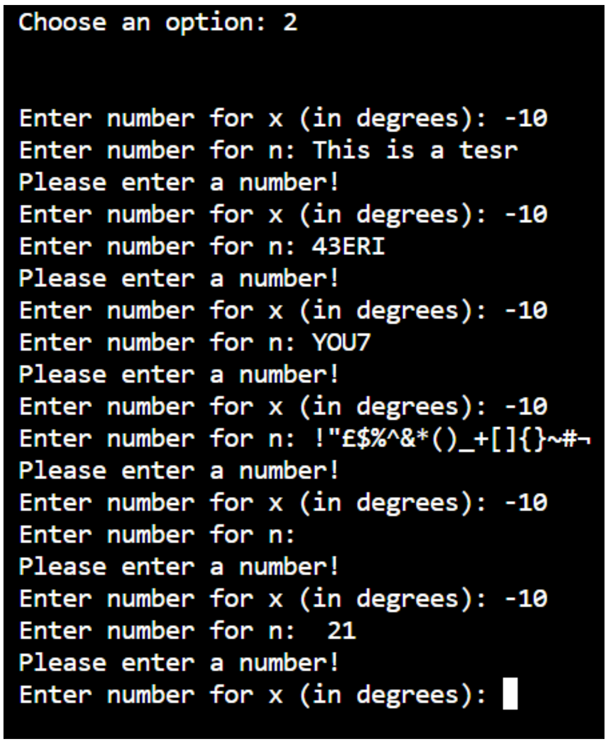


```
Choose an option: 2


Enter number for x (in degrees): -360
Enter number for n: 45.5
Please enter a number!
Enter number for x (in degrees): -360
Enter number for n: 47.428438484
Please enter a number!
Enter number for x (in degrees): -360
Enter number for n: -21.48489439
Please enter a number!
Enter number for x (in degrees): -360
Enter number for n: 23.0
Please enter a number!
Enter number for x (in degrees): -360
Enter number for n: -34.0
Please enter a number!
Enter number for x (in degrees):
```

String values:



```
Choose an option: 1


Enter number for x (in degrees): 55
Enter number for n: This is a message
Please enter a number!
Enter number for x (in degrees): 55
Enter number for n: 43T
Please enter a number!
Enter number for x (in degrees): 55
Enter number for n: E3
Please enter a number!
Enter number for x (in degrees): 55
Enter number for n: !"£$%^&*(){}[]_+¬~#
Please enter a number!
Enter number for x (in degrees): 55
Enter number for n:
Please enter a number!
Enter number for x (in degrees): 55
Enter number for n:  34
Please enter a number!
Enter number for x (in degrees):
```



```
Choose an option: 2


Enter number for x (in degrees): -10
Enter number for n: This is a tesr
Please enter a number!
Enter number for x (in degrees): -10
Enter number for n: 43ERI
Please enter a number!
Enter number for x (in degrees): -10
Enter number for n: YOU7
Please enter a number!
Enter number for x (in degrees): -10
Enter number for n: !"£$%^&*()_+[]{}~#¬
Please enter a number!
Enter number for x (in degrees): -10
Enter number for n:
Please enter a number!
Enter number for x (in degrees): -10
Enter number for n:  21
Please enter a number!
Enter number for x (in degrees):
```

- Question 12
  - Source Code & Explanation

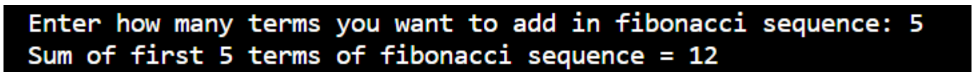
```
def main():
    answer = 0

    # Check whether input entered by user is valid
    valid_input = False
    while not valid_input:
        # Asking user to input number of terms to be calculated and added
together in fibonacci
        n = input("Enter how many terms you want to add in fibonacci sequence:
")

        # Check if user has entered value for n and that value is an integer
        if len(n) > 0 and n.lstrip('+-').isdigit():
            n = int(n)
            # Check if n is less than or equal to 0
            if n <= 0:  # If True, tell user to input another number
                valid_input = False
                print("Number must be at least 1 or greater than 1!")
            elif n > 20575:
                valid_input = False
                print("Numerical overflow might be possible! Please enter a
smaller number!")
            else:   # If False, proceed with calculating answer
                valid_input = True
                answer = sum_fibonacci(n)
        else:
            valid_input = False
            print("Please enter an integer!")

    # Printing the answer after function execution
    print("Sum of first", n, "terms of fibonacci sequence =", answer)


# Function which returns the sum of the first n fibonacci terms as stated by
the user input
def sum_fibonacci(counter):
    # Check if counter is greater than 1
    if counter == 1:    # If yes, there is only one number, so return 1
        return 1
    else: # Otherwise, perform Fibonacci sequence
        # Set term1 and term2 to be 1
        term1 = term2 = 1
        # Set sum to be equal to the sum of term1 and term2 before start of
loop
        sum = term1+term2

        # Calculate nextTerm and add it to sum variable until first n terms
are found
        for i in range(2, counter):
            nextTerm = term1 + term2
            term1 = term2
            term2 = nextTerm
            sum += nextTerm
            #print(nextTerm)  -  TESTING PURPOSES

        return sum


# Call main() function to start execution
main()
```

The program begins execution by asking the user to enter the number of terms to calculate the Fibonacci sequence. It then checks to see if it is a positive integer. If it is, then proceed with calculating the Fibonacci sequence. Otherwise, an appropriate error message is displayed, and the user is prompted to enter a valid number. Once completed, the sum_fibonacci() function is called and the final answer is stored in a variable called answer. The function calculates the sum for the number of terms entered by the user after calculating the Fibonacci sequence. The answer is then printed out to the user.

❖ <u>How the Program was Tested + Screen Dumps</u>

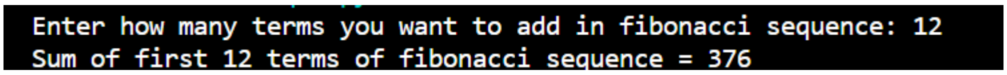
The program was first tested by entering a few numbers such that they are valid. The numbers that were inputted were 5, 12, 30 and 75. As expected, the sum of the Fibonacci sequence up to the numbers inputted was displayed on the console. This can be seen below as follows:

<u>Num 1: Inputting value and Answer Output</u>



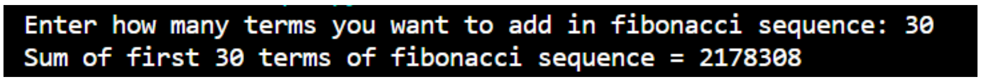
```
Enter how many terms you want to add in fibonacci sequence: 5
Sum of first 5 terms of fibonacci sequence = 12
```

<u>Num 2: Inputting value and Answer Output</u>



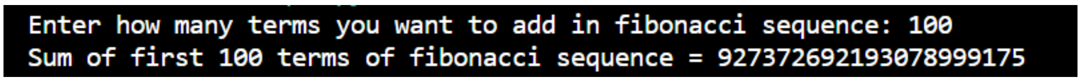
```
Enter how many terms you want to add in fibonacci sequence: 12
Sum of first 12 terms of fibonacci sequence = 376
```

<u>Num 3: Inputting value and Answer Output</u>



```
Enter how many terms you want to add in fibonacci sequence: 30
Sum of first 30 terms of fibonacci sequence = 2178308
```

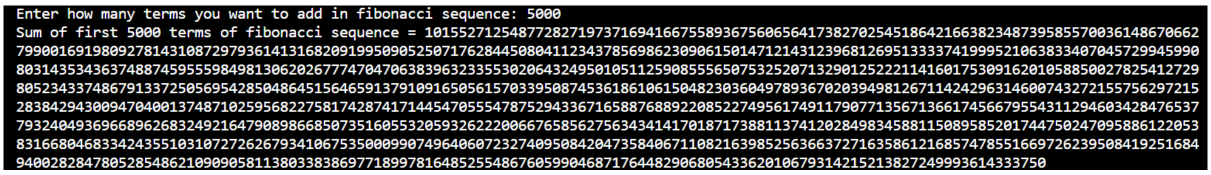
<u>Num 4: Inputting value and Answer Output</u>



```
Enter how many terms you want to add in fibonacci sequence: 100
Sum of first 100 terms of fibonacci sequence = 9273726921930789999175
```

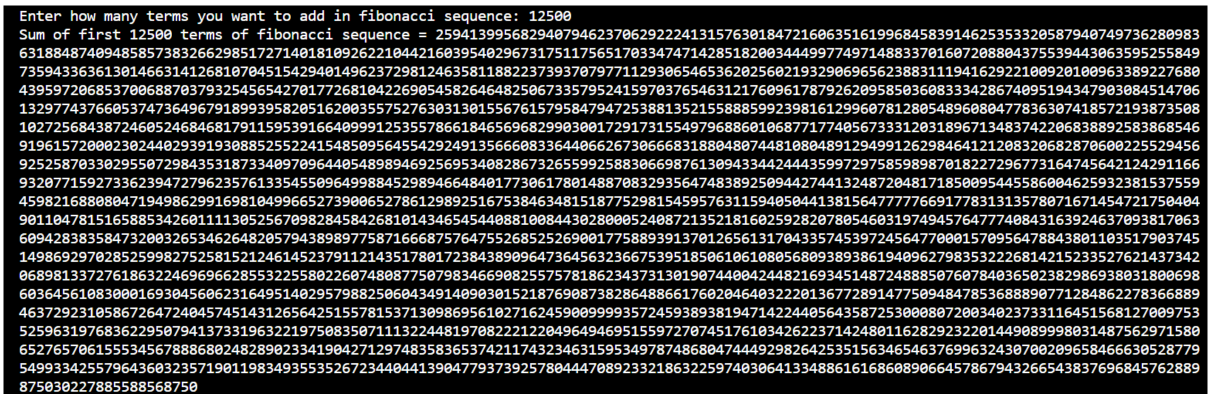
The program was then tested by entering a few large numbers. This was done to check if there was a possible chance that there would be a numerical overflow. The numbers that were inputted were 5000, 12500, 20000 and 21500. The result was that, for the first three numbers, the sum of the Fibonacci sequence up to those numbers was displayed on the console. However, for the last number, the program detected that it might cause a numerical overflow. Thus, a message was printed out stating that numerical overflow might be possible, before redirecting the user to input another value. This can be seen below as follows:

<u>Num 1: Inputting value and Answer Output</u>



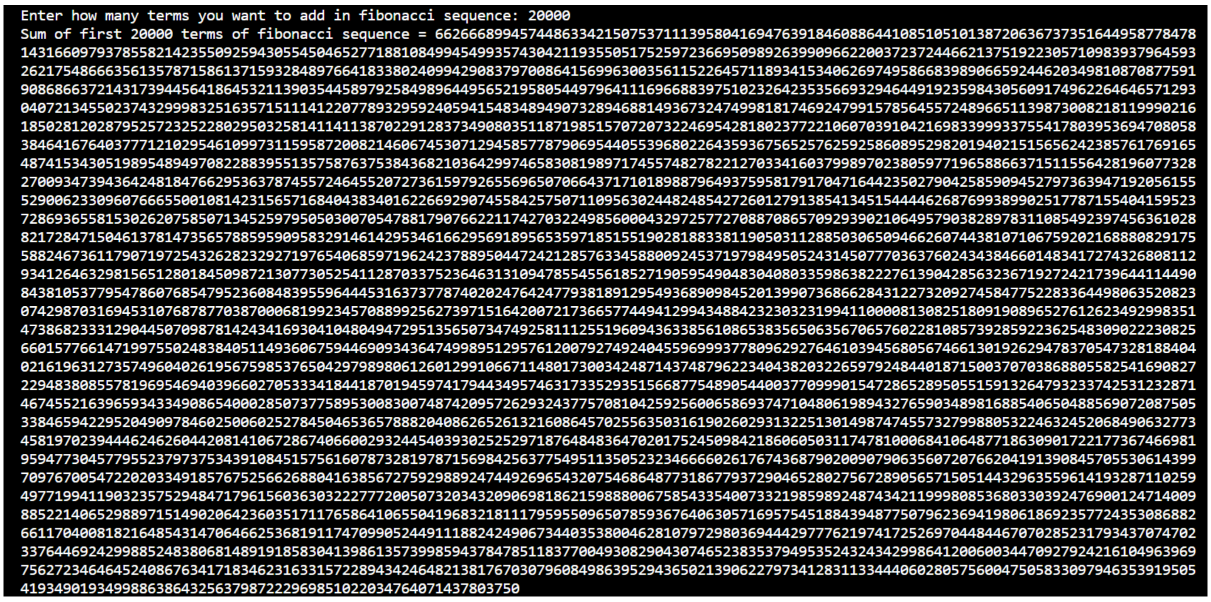
```
Enter how many terms you want to add in fibonacci sequence: 5000
Sum of first 5000 terms of fibonacci sequence = 1015527125487728271973716941667558936756065641738270254518642166382348739585570036148670662
7990016919809278143108729793614131682091995090525071762844508041123437856986230906150147121431239681269513333741999521063833407045729945990
8031435343637488745955598498130620267774704706383963233553020643249501051125908555650753252071329012522211416017530916201058850027825412729
8052343374867913372505695428504864515646591379109165056157033950874536186106150482303604978936702039498126711424296314600743272155756297215
2838429430094704001374871025956822758174287417144547055547875294336716588768892208522749561749117907713567136617456679554311294603428476537
7932404936966896268324921647908986685073516055320593262220066765856275634341417018717388113741202849834588115089585201744750247095886122053
8316680468334243551031072726267934106753500099074964060723274095084204735840671108216398525636637271635861216857478551669726239508419251684
9400282847805285486210909058113803383869771899781648525548676059904687176448290680543362010679314215213827249993614333750
```

Num 2: Inputting value and Answer Output



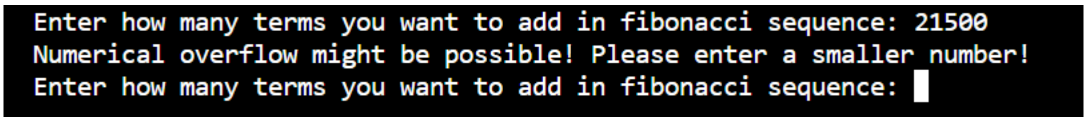
```
Enter how many terms you want to add in fibonacci sequence: 12500
Sum of first 12500 terms of fibonacci sequence = 2594139956829407946237062922241315763018472160635161996845839146253533205879407497362809836318848740948585738326629851727140181092622104421603954029673175117565170334747142851820034449977497148833701607208804375539443063595255849
7359433636130146631412681070451542940149623729812463581188223739370797711293065465362025602193290696562388311194162922100920100963389227680
4395972068537006887037932545654270177268104226905458264648250673357952415970376546312176096178792620958503608333428674095194347903084514706
1329774376605374736496791899395820516200355752763031301556761579584794725388135215588859923981612996078128054896080477836307418572193873508
1027256843872460524684681791159539166409991253557866184656968299030017291731554979688601068771774056733312031896713483742206838892583868546
9196157200023024402939193088525522415485095645542924913566608336440662673066683188048074481080489129499126298464121208320682870600225529456
9252587033029550729843531873340970964405489894692569534082867326559925883066987613094334424443599729758598987018227296773164745642124291166
9320771592733623947279623576133545509649988452989466484017730617801488708329356474838925094427441324872048171850095445586004625932381537559
4598216880804719498629916981049966527390065278612989251675384634815187752981545957631159405044138156477777669177831313578071671454721750404
9011047815165885342601111305256709828458426810143465454408810084430280005240872135218160259282078054603197494576477740843163924637093817063
6094283835847320032653462648205794389897758716668757647552685252690017758893913701265613170433574539724564770001570956478843801103517903745
1498692970285259982752581521246145237911214351780172384389096473645632366753951850610610805680938938619409627983532226814215233527621437342
0689813372761863224696966285532255802260748087750798346690825575781862343731301907440042448216934514872488850760784036502382986938031800698
6036456108300016930456062316495140295798825060434914090301521876908738286488661760204640322201367728914775094847853688890771284862278366889
4637292310586726472404574514312656425155781537130986956102716245900999935724593893819471422440564358725300080720034023733116451568127009753
5259631976836229507941373319632219750835071113224481970822212204964946951559727074517610342622371424801162829232201449089998031487562971580
6527657061555345678886802482890233419042712974835836537421174323463159534978748680474449298264253515634654637699632430700209658466630528779
5499334255796436032357190119834935535267234404413904779373925780444708923321863225974030641334886161686089066457867943266543837696845762889
875030227885588568750
```

Num 3: Inputting value and Answer Output



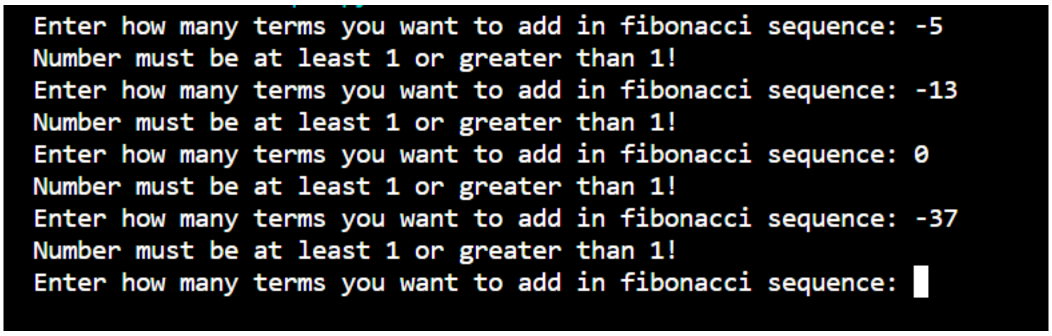
```
Enter how many terms you want to add in fibonacci sequence: 20000
Sum of first 20000 terms of fibonacci sequence = 6626668994574486334215075371113958041694763918460886441085105101387206367373516449587784781431660979378558214235509259430554504652771881084994549935743042119355051752597236695098926399096622003723724466213751922305710983937964593
2621754866635613578715861371593284897664183380240994290837970086415699630035611522645711893415340626974958668398906659244620349810870877591
9086866372143173944564186453211390354458979258498964495652195805449796411169668839751023264235356693294644919235984305609174962264646571293
0407213455023743299983251635715111412207789329592405941548348949073289468814936732474998181746924799157856455724896651139873008218119990216
1850281202879525723252280295032581411411387022912837349080351187198515707207322469542818023772210607039104216983399933755417803953694708058
3846416764037771210295461099731159587200821460674530712945857787906954405539680226435936756525762592586089529820194021515656242385761769165
4874153430519895489497082288395513575876375384368210364299746583081989717455748278221270334160379989702380597719658866371511556428196077328
2700934739436424818476629536378745572464552072736159792655696507066437171018988796493759581791704716442350279042585909452797363947192056155
5290062330960766655001081423156571684043834016226692907455842575071109563024482485427260127913854134515444462687699389902517787155404159523
7286936558153026207585071345259795050300705478817907662211742703224985600043297257727088708657092939021064957903828978311085492397456361028
8217284715046137814735657885959095832914614295346166295691895653597185155190281883381190503112885030650946626074438107106759202168880829175
5882467361179071972543262823292719765406859719624237889504472421285763345880092453719798495052431450777036376024343846601483417274326808112
9341264632981565128018450987213077305254112870337523646313109478554556185271905954904830408033598638222761390428563236719272421739644114490
8438105377954786076854795236084839559644453163737787402024764247793818912954936890984520139907368662843122732092745847752283364498063520823
0742987031694531076878770387000681992345708899256273971516420072173665774494129943488423230323199411000081308251809190896527612623492998351
4738682333129044507098781424341693041048049472951356507347492581112551960943633856108653835650635670657602281085739285922362548309022230825
6601577661471997550248384051149360675944690934364749989512957612007927492404559699937780962927646103945680567466130192629478370547328188404
0216196312735749604026195675985376504297989806126012991066711480173003424871437487962234043820322659792484401871500370703868805582541690827
2294838085578196954694039660270533341844187019459741794434957463173352935156687754890544003770999015472865289505515913264793233742531232871
4674552163965934334908654000285073775895300830074874209572629324377570810425925600658693747104806198943276590348981688540650488569072087505
3384659422952049097846025006025278450465365788820408626526132160864570255635031619026029313225130149874745573279988053224632452068490632773
4581970239444624626044208141067286740660029324454039302525297187648483647020175245098421860605031174781000684106487718630901722177367466981
9594773045779552379737534391084515756160787328197871569842563775495113505232346666026176743687902009079063560720766204191390845705530614399
7097670054722020334918576752566268804163856727592988924744926965432075468648773186779372904652802756728905657150514432963559614193287110259
4977199411903235752948471796156036303222777200507320343209069818621598880067585433540073321985989248743421199980853680330392476900124714009
8852214065298897151490206423603517117658641065504196832181117959550965078593676406305716957545188439487750796236941980618692357724353086882
6611704008182164854314706466253681911747099052449111882424906734403538004628107972980369444297776219741725269704484467070285231793437074702
3376446924299885248380681489191858304139861357399859437847851183770049308290430746523835379495352432434299864120060034470927924216104963969
7562723464645240867634171834623163315722894342464821381767030796084986395294365021390622797341283113344406028057560047505833097946353919505
41934901934998863864325637987222969851022034764071437803750
```

Num 4: Inputting value and Message Output



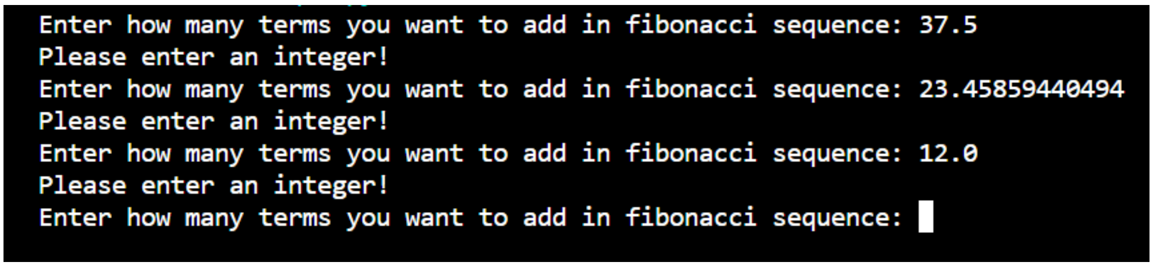
```
Enter how many terms you want to add in fibonacci sequence: 21500
Numerical overflow might be possible! Please enter a smaller number!
Enter how many terms you want to add in fibonacci sequence: █
```

Afterwards, the program was tested by inputting several numbers which are not positive. The numbers inputted were -5, -13, 0, -37. As expected, the program detected that trying to make a Fibonacci sequence with such numbers is not possible. Thus, a message was printed out stating that the number must be at least 1 or greater than 1, before redirecting the user to enter another value. This can be seen below:



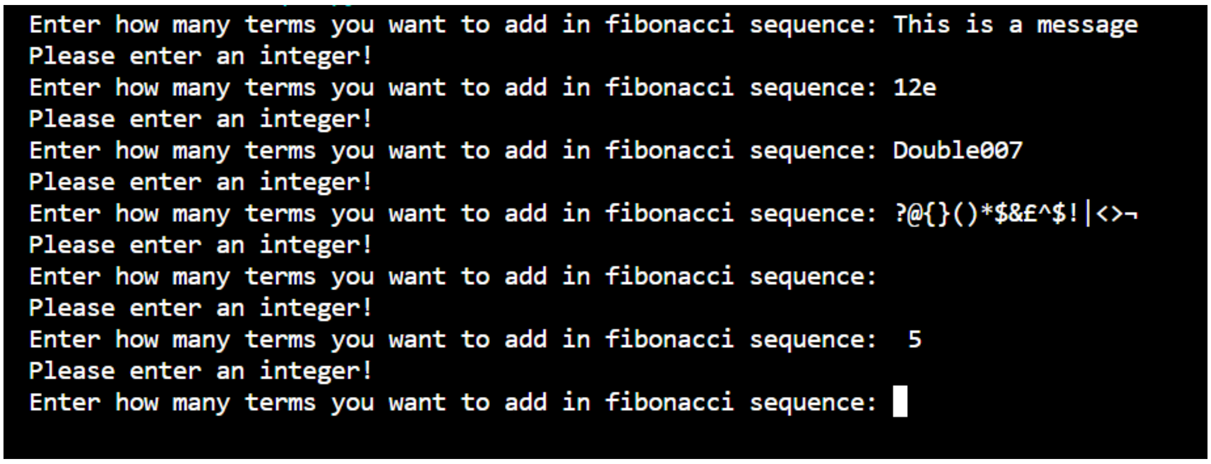
```
Enter how many terms you want to add in fibonacci sequence: -5
Number must be at least 1 or greater than 1!
Enter how many terms you want to add in fibonacci sequence: -13
Number must be at least 1 or greater than 1!
Enter how many terms you want to add in fibonacci sequence: 0
Number must be at least 1 or greater than 1!
Enter how many terms you want to add in fibonacci sequence: -37
Number must be at least 1 or greater than 1!
Enter how many terms you want to add in fibonacci sequence: 
```

The program was then tested by entering several float values. This was done to test the program ability to distinguish a float from an integer. As expected, the program saw thistype of user input as invalid, and the user was prompted to enter an integer. This can be seen below:



```
Enter how many terms you want to add in fibonacci sequence: 37.5
Please enter an integer!
Enter how many terms you want to add in fibonacci sequence: 23.45859440494
Please enter an integer!
Enter how many terms you want to add in fibonacci sequence: 12.0
Please enter an integer!
Enter how many terms you want to add in fibonacci sequence: 
```

Finally, the program was tested by entering several string values. The tested strings were divided in four parts, whole text, text with numbers included, empty values, and special characters. For the last test, a space was added before the number was inputted. Once again, the program saw this type of input as invalid and the user is prompted to enter an integer. This can be seen below:



```
Enter how many terms you want to add in fibonacci sequence: This is a message
Please enter an integer!
Enter how many terms you want to add in fibonacci sequence: 12e
Please enter an integer!
Enter how many terms you want to add in fibonacci sequence: Double007
Please enter an integer!
Enter how many terms you want to add in fibonacci sequence: ?@{}()*$&£^$!|<>¬
Please enter an integer!
Enter how many terms you want to add in fibonacci sequence:
Please enter an integer!
Enter how many terms you want to add in fibonacci sequence:  5
Please enter an integer!
Enter how many terms you want to add in fibonacci sequence:
```

Programming Choices Made

I chose to complete this coursework using the Python programming language because, not only was I more comfortable with Python due to its easy-to-use syntax, but it also has a lot more features that allowed me to develop certain parts of the coursework.

This was especially the case in q7.py, where I had to store the Collatz sequence for a set of numbers in a csv file. This is because Python has various functions that utilize csv files that are much easier to use than in other programming languages such as Java.

Also, whenever I had to check whether the user input value was of type integer, regardless of the number being positive of negative, Python allowed me to do this in a simple manner by using the lstrip() function. This was because, had this not been done, the program would have displayed an invalid input message since the input would be of string if the number was negative. Other programming languages might utilize this feature in a more complex manner.

Problems Encountered

Throughout the development of all the programs, there were several bugs and errors that were encountered along the way. However, almost all of them were resolved after a lot of testing.

One of the problems that I faced was that, during user input, whenever a variable tried to store a value whose datatype is not equal to the one expected by that variable, for example an integer, a ValueError gets brought up on the console and the program would suddenly terminate. This was because, at first, I was directly casting the user input into an integer without checking if the input value was of type integer. Below is an example of a line of code from q3.py that was used during development that would bring up this error:

```
num = int(input("Enter an integer: "))
```

I easily managed to solve this by removing the int() from the assignment statement and checking, using an if-else statement, whether the value stored in the variable is of type integer. This allowed for better error capturing and allowed each program that had this issue to continue executing without any errors generating. Below is a snippet of code from q3.py that better explains how the bug was solved:

```
# Ask user to input an integer
        num = input("Enter an integer: ")

        # Check if user input is an integer
        if num.lstrip('-+').isnumeric():    # If True, add number to list
            array.append(int(num))

            # Check for valid option
            valid_choice = False

            while not valid_choice:
                # Ask user to decide whether to continue
                choice = input("Do you want to continue adding more numbers
(Y/N)?\n")

                # Check if user entered a value before pressing Enter
                if len(choice) > 0:
                    if choice[0].capitalize() == 'Y':   # valid option,
proceeds with entering next number
                        has_stopped = False
                        valid_choice = True
                    elif choice[0].capitalize() == 'N': # valid option, exits
entire iteration
                        has_stopped = True
                        valid_choice = True
                    else:   # invalid option, asks user to enter a valid
option
```

```
                        valid_choice = False
                        print("Please enter Y or N (Yes or No)!")
                else:
                    valid_choice = False
                    print("Please enter a value!")
        else:   # Number not an integer!
            print('Invalid input! Please enter an integer!')
```

Another bug that was encountered happened in q12.py. Whenever a very large number was inputted to calculate the sum of the Fibonacci sequence up to that number, a ValueError got displayed on the console and the program would terminate. This was because the Python interpreter has a limit on how large the number must be in order to convert it from a string to an integer.

This issue was fixed by simply writing an if statement that occurs if the number inputted was larger than a certain range, and then displaying a message on the console. This again allowed better error capturing and the program could continue execution without any errors. This is a snippet of code from q12.py that shows how this issue was dealt with:

```
# Asking user to input number of terms to be calculated and added together in
fibonacci
        n = input("Enter how many terms you want to add in fibonacci sequence:
")

        # Check if user has entered value for n and that value is an integer
        if len(n) > 0 and n.lstrip('+-').isdigit():
            n = int(n)
            # Check if n is less than or equal to 0
            if n <= 0:  # If True, tell user to input another number
                valid_input = False
                print("Number must be at least 1 or greater than 1!")
            elif n > 20575:
                valid_input = False
                print("Numerical overflow might be possible! Please enter a
smaller number!")
            else:   # If False, proceed with calculating answer
                valid_input = True
                answer = sum_fibonacci(n)
        else:
            valid_input = False
            print("Please enter an integer!")
```

The last issue that was encountered took place in q5.py. This occurred when, during stack evaluation, whenever the stack pointer reached an operator and only detected one operand before it, a TypeError got brought up and the program would terminate. This was because the stack was programmed, during evaluation, to check if there are 2 operands, especially since the stack is in RPN format. Below is a snippet of code from q5.py that better explains this issue:

```
if ord(stack[i]) == 43:     # addition
            num1 = stack[i-2]
            num2 = stack[i-1]
            j = i-1
            stack.pop(j)
            j -= 1
            stack.pop(j)
            stack[j] = num1+num2
            i -= 1
```

Unfortunately, unlike the previous two bugs, this one could not be fixed. Thus, this is the only major bug that still exists in this coursework.