Debugging the ChatGPT solution to the Halloween candy problem

input	4	3	1	5	
	9	15	2	7	
	2	5	6	17	
	11	13	4	8	
max from	38	37	35	37	
each house	27	34	27	32	
	15	18	19	25	
	11	13	4	8	copy this row from input table

```
def max_candy_collection(grid):
    m = len(grid) # number of rows
    n = len(grid[0]) # number of columns

    # Initialize the memo table with the same values as the bottom row of the grid
    memo = [list(row) for row in grid[-1:]]

    # Fill in the memo table by working our way up from the second-to-last row
    for i in range(m-2, -1, -1):
        for j in range(n):
            # Calculate the maximum number of candy pieces that can be collected
            # from the next house by taking the maximum of the three possible
            # moves from the next row's corresponding houses, plus the value of
            # the current house.
            candy = grid[i][j] + max(
                memo[-1][j],
                memo[-1][max(j-1, 0)],
                memo[-1][min(j+1, n-1)]
            )

            # Store the maximum number of candy pieces in the table for this house
            memo.insert(0, [candy])

    # Find the max collection by checking the values in the top row of the table
    max_candy = max(memo[0])

    return max_candy
```

Comparing Code Styles for the Original Flavor-Selection Problem

```
# the OO solution that we wrote by hand
class PickSweets: # the OO solution that we wrote by hand

   def __init__(self, sweets_lst : "list[tuple[str, int]]"):
       self.sweets_list = sweets_lst

   def rating(self, for_index : int) -> int:
       return self.sweets_list[for_index][1]

   def max_sweets_rating(self, start_ind: int) -> int:
       if start_ind == len(self.sweets_list) - 1: # is the last sweet
           return self.rating(start_ind)
       elif start_ind == len(self.sweets_list) - 2: # is the next to last sweet
           return max(self.rating(start_ind), self.rating(start_ind + 1))
       else:
           pick_this_sweet = self.rating(start_ind) + self.max_sweets_rating(start_ind + 2)
           skip_this_sweet = self.max_sweets_rating(start_ind + 1)
           return max(pick_this_sweet, skip_this_sweet)

   def pick_sweets(self) -> int:
       return self.max_sweets_rating(0)
```

```
# the non-OO solution from ChatGPT
def max_sweets(n, tasty):
  dp = [0] * n          # create an array of length n to hold results
  dp[n-1] = tasty[n-1][1]
  dp[n-2] = max(tasty[n-2][1], tasty[n-1][1])
  for i in range(n-3, -1, -1):
      dp[i] = max(dp[i+1], dp[i+2] + tasty[i][1])
  return dp[0]
```

```
# A list of sweets and their ratings
our_sweets_lst = [("choc", 3), ("straw", 10), ("vanilla", 12), ("pistachio", 16), ("rasp", 4)]

p = PickSweets(our_sweets_lst)
p.pick_sweets()

max_sweets(our_sweets_list)
```