```
 1: def shell_sort(collection):
 2:     gaps = [701, 301, 132, 57, 23, 10, 4, 1]
 3:     for gap in gaps:
 4:         i = gap
 5:         while i < len(collection):
 6:             temp = collection[i]
 7:             j = i
 8:             while j >= gap and collection[j - gap] > temp:
 9:                 collection[j] = collection[j - gap]
10:                 j -= gap
11:             collection[j] = temp
12:             i += 1
13:     return collection
14:
15:
16: def set_x():
17:     global x
18:     x = 2
19:     shell_sort([1,2,3,4])
20:
21:
22: # Global from ... import ... statement for commonly used functions
23: from random import randint as global_randint, choice
24:
25: # Global variable definition
26: global_counter = 0
27:
28: # Define a class with a method that includes a nested function
29: class Counter:
30:     def __init__(self):
31:         # Instance variable to track counts
32:         self.count = 0
33:
34:     def update_counter(self):
35:         # Local import within the function for demonstration
36:         import time
37:
38:         # Method variable
39:         method_counter = 0
40:
41:         def increment():
42:             # Use nonlocal to modify method_counter
43:             nonlocal method_counter
44:             # Use global to modify the global_counter
45:             global global_counter
46:
47:             try:
48:                 # Use the globally imported global_randint function directly
```

```
49:                 increment = global_randint(1, 10)
50:                 # Simulate a random error
51:                 if choice([True, False]):
52:                     raise ValueError("Simulated error")
53:
54:                 method_counter += increment
55:                 global_counter += increment
56:                 self.count += increment
57:
58:                 # Demonstrating the use of the locally imported time module
59:                 time.sleep(1)  # Sleep for 1 second to simulate a delay
60:
61:                 print(f"Method counter incremented by {increment}, total method counter: {method_counter}")
62:                 print(f"Global counter updated to {global_counter}, instance counter: {self.count}")
63:
64:             except ValueError as e:
65:                 print(f"Exception caught: {e}")
66:
67:         increment()
68:
69: # Instantiate the class and call the method
70: counter_instance = Counter()
```