1. Using `enumerate()` and `sort()`

`enumerate()` adds a counter to an iterable (like a list or a tuple), which you can use in a `for` loop or for sorting purposes. You can then sort the data based on the enumerated index or another criterion.

Example 1: Sorting with `enumerate()`

Let's say you have a list of names and want to sort it, but keep track of the original indices.

python Copy code

```python
names = ['Julia', 'Alex', 'Chris', 'Bob'] # Enumerate the list enumerated_names =
list(enumerate(names)) # Sort by the names (which are the second element in the tuple)
enumerated_names.sort(key=lambda x: x[1]) # Result after sorting for index, name in
enumerated_names: print(f'Original index: {index}, Name: {name}')
```

Output:

yaml Copy code

```yaml
Original index: 3, Name: Bob Original index: 1, Name: Alex Original index: 2, Name: Chris
Original index: 0, Name: Julia
```

2. Using `zip()` and `sort()`

`zip()` is used to combine two or more iterables (like lists or tuples) element-wise into tuples. You can then sort the combined data based on one of the elements.

Example 2: Sorting with `zip()`

Let's say you have two lists, one for names and another for scores, and you want to sort the names based on scores.

python Copy code

```python
names = ['Julia', 'Alex', 'Chris', 'Bob'] scores = [85, 92, 78, 88] # Combine names and scores
using zip combined = list(zip(names, scores)) # Sort by the second element (the scores)
combined.sort(key=lambda x: x[1]) # Result after sorting for name, score in combined:
print(f'Name: {name}, Score: {score}')
```

Output:

yaml Copy code

```yaml
Name: Chris, Score: 78 Name: Julia, Score: 85 Name: Bob, Score: 88 Name: Alex, Score: 92
```