Lists: The Shopping Cart

Depict lists as a line of shopping carts at a supermarket:

Each cart (element) can hold different items

Carts can be rearranged (mutable)

Each cart has a number (index) showing its position in line

Can add/remove carts or swap their positions

Elaboration:

Example, carts or boxes labeled with numbers (0, 1, 2...)

Interactive Activity:

Place different items in each cart

Call out operations: "Add a new cart at position 2!" or "Remove the last cart!"

Rearrange the carts to show list operations

Examples:

```
shopping_list = ["milk", "eggs", "bread"]

shopping_list.append("cheese")      # Adding a new cart at the end

shopping_list.insert(1, "butter")    # Adding a cart in position 1

shopping_list.pop()                  # Removing the last cart

shopping_list[0] = "almond milk"     # Changing what's in the first cart
```

Discussion Points:

"Why might we want to change our shopping list as we shop?" (Connecting to real-world mutability)

"What happens if we try to grab the item from cart #10 when we only have 3 carts?" (IndexError)

Tuples: The Museum Display Case

Each case contains various items (elements)

Cases are locked with "Do Not Touch" signs (immutable)

Each item has a position number (index)

You can try but, you will get an error when add something to a Tuple

Interactive Activity:

Decide what items to place in each "display case" before sealing

Once sealed, it can't be opened!

Compare with the shopping carts that could be modified

```
# Show the difference between lists and tuples
days_list = ["Monday", "Tuesday", "Wednesday"]
days_tuple = ("Monday", "Tuesday", "Wednesday")

# This works
days_list[0] = "Sunday"

# This causes an error
try:
    days_tuple[0] = "Sunday"  # TypeError: 'tuple' object does not support item assignment
except TypeError as e:
    print("Can't modify a sealed display case!")
```

Discussion Points:

"When would we want data that can't be changed?" (Constants, days of week, coordinates)

"Why might a museum put items in sealed cases?" (Protection, preservation - like protecting data integrity)

Dictionaries: The Locker

Place different items in each labeled locker

You can retrieve items by calling out the label, not position

Can add new lockers or change contents without affecting others

```
 student_lockers = {
    "Alex": ["textbook", "lunch", "gym clothes"],
    "Taylor": ["notebook", "calculator", "water bottle"],
    "Jordan": ["laptop", "headphones"]
}


# Finding items by name, not position
print(f"What's in Alex's locker? {student_lockers['Alex']}")


# Adding a new locker
student_lockers["Morgan"] = ["art supplies", "snack"]


# Changing what's in a locker
student_lockers["Taylor"] = ["project", "lunch"]
```

"Why is it easier to find something by name rather than remembering its position?"

"What happens if two lockers have the same name?" (Keys must be unique)