Complex Data Types in Python

Lists, Dictionaries, and Nested Structures

- Python supports complex data types like lists and dictionaries.
- Lists: ordered collections of items (e.g., `[1, 2, 3]`)
- Dictionaries (mappings): key-value pairs

```
{
    "description": "A wooden torch. It might be useful in the dark.",
    "type": "light"
}
```

- You can nest these structures for more complex data.

Why Use Nested Structures?

- They let us represent complex information in a way that's easy for both humans and programs to understand.
- Example: A room with items and exits, each described in detail.

Example: `advent/game.json`

- The game data is stored as a nested structure:
  - The top level is a dictionary with keys like `rooms`, and `start_room`.
  - `rooms` is a dictionary where each value is another dictionary describing a room.
  - Each room dictionary can contain lists (like `item_names`) and dictionaries (like `exits`).

Accessing and Modifying Lists and Dictionaries in Python

- **Adding to a list:**

```
self.inventory.append(item)  # Add an item to the player's inventory
```

- **Removing from a list:**

```
room["item_names"].remove(item)  # Remove an item from the room's item_names list
```

- **Dictionary lookup with [square brackets]:**

```
room = self.rooms[self.current_room]  # Get the current room dictionary
desc = room["description"]            # Get the description string
```

- **Dictionary lookup with .get() and a default value if key is not found:**

```
item_names = room.get("item_names", [])  # Get "item_names", or empty list if not present
exits = room.get("exits", {})            # Get "exits", or empty dictionary if not present
```

Common operation: iteration

- Do the same operation on each item in a list

```
for item_name in item_names:
    output += f"\n  - {item_name}"
```

- Do the same operation on each key in a dictionary

```
for direction in exits.keys():
    output += f" {direction}"
```

Discussion

- What are the advantages and challenges of using nested data structures?
- How do type annotations help us work with these structures?