Python provides a rich set of built-in methods and functions to work with lists, which are essential data structures that hold an ordered collection of elements. These allow you to perform operations such as adding, removing, modifying, sorting, and querying list contents. Below is a comprehensive list of important list methods and functions in Python, accompanied by real-life examples:

1. **Built-in List Methods**

`append(x)`

- **Purpose**: Adds an item at the end of the list.
- **Syntax**: `list.append(item)`
- **Example**:

```
items = ['apple', 'banana']
items.append('orange')
print(items)  # Output: ['apple', 'banana', 'orange']
```

`extend(iterable)`

- **Purpose**: Adds all elements of an iterable (like list, tuple, or set) to the end of the list.
- **Syntax**: `list.extend(sequence)`
- **Example**:

```
cities = ['Tokyo', 'New York']
more_cities = ['London', 'Paris']
cities.extend(more_cities)
print(cities)  # Output: ['Tokyo', 'New York', 'London',
'Paris']
```

`insert(i, x)`

- **Purpose**: Inserts an item at a given position in the list.
- **Syntax**: `list.insert(index, item)`
- **Example**:

```
numbers = [10, 20]
numbers.insert(1, 15)
```

```
print(numbers)  # Output: [10, 15, 20]
```

`remove(x)`

- **Purpose**: Removes the first occurrence of a value from the list.
- **Syntax**: `list.remove(item)`
- **Example**:

```
courses = ['Math', 'Physics', 'Chemistry']
courses.remove('Physics')
print(courses)  # Output: ['Math', 'Chemistry']
```

`pop([index])`

- **Purpose**: Removes and returns the element at the specified index or, if no index is provided, removes and returns the last item in the list.
- **Syntax**: `list.pop(index)`
- **Example**:

```
data = ['apple', 'banana', 'cherry']
popped_item = data.pop(1)  # Returns 'banana'
print(data)  # Output: ['apple', 'cherry']
```

`index(x[, start [, end]])`

- **Purpose**: Finds the index of the first occurrence of an element in the list.
- **Syntax**: `list.index(item, [start], [end])`
- **Example**:

```
fruits = ['apple', 'banana', 'cherry']
print(fruits.index('banana'))  # Output: 1
```

`count(x)`

- **Purpose**: Counts the number of times an item appears in a list.
- **Syntax**: `list.count(item)`
- **Example**:

```
items = ['apple', 'banana', 'apple']
print(items.count('apple'))  # Output: 2
```

`sort([key], [reverse])`

- **Purpose**: Sorts the elements of a list in place.
- **Syntax**: `list.sort(key=None, reverse=False)`
- **Example**:

```
scores = [100, 95, 85]
scores.sort()
print(scores)  # Output: [85, 95, 100] (Ascending order)
```

`reverse()`

- **Purpose**: Reverses the elements of a list in place.
- **Syntax**: `list.reverse()`
- **Example**:

```
names = ['Alice', 'Bob']
names.reverse()
print(names)  # Output: ['Bob', 'Alice'] (Reversed order)
```

`copy()`

- **Purpose**: Creates a shallow copy of the list.
- **Syntax**: `list.copy()`
- **Example**:

```
original = [1, 2, 3]
copied = original.copy()
print(copied)  # Output: [1, 2, 3] (Independent list)
```

2. **Built-in List Functions**

`len(x)`

- **Purpose**: Returns the number of items in a list.
- **Syntax**: `len(list)`
- **Example**:

```
print(len([1, 2, 3]))  # Output: 3
```

`max(x)`

- **Purpose**: Finds the largest item in an iterable or the largest of two or more arguments.
- **Syntax**: `max(iterable)` or `max(args)`
- **Example**:

```
print(max([10, 20, 30]))  # Output: 30 (Largest number)
```

`min(x)`

- **Purpose**: Finds the smallest item in an iterable or the smallest of two or more arguments.
- **Syntax**: `min(iterable)` or `min(args)`
- **Example**:

```
print(min([10, 20, 30]))  # Output: 10 (Smallest number)
```

`sum(x[, start])`

- **Purpose**: Sums up all the elements in a list.
- **Syntax**: `sum(iterable[, start])`
- **Example**:

```
print(sum([1, 2, 3, 4], 0))  # Output: 10 (Summing with
initial value)
```

`enumerate(x)`

- **Purpose**: Takes a collection (tuple, list, string) and returns an iterator that produces pairs of values and their indices.
- **Syntax**: `list(enumerate(sequence), start=0)`
- **Example**:

```
for index, name in enumerate(['Alice', 'Bob']):
  print(f"{index}: {name}")
# Output:
# "0: Alice"
# "1: Bob"
```

`any(x)`

- **Purpose**: Returns True if any element of an iterable is true.
- **Syntax**: `any(iterable)`
- **Example**:

```
print(any([False, False, True]))  # Output: True (At least
one True value)
```

`all(x)`

- **Purpose**: Returns True if all elements of an iterable are true.
- **Syntax**: `all(iterable)`
- **Example**:

```
print(all([True, True, True]))  # Output: True (All values
are True)
```

Real-Life Examples

Let's take a real-life scenario involving managing a list of students' grades in a class. Assume we have the following lists:

```
students = ['Alice', 'Bob', 'Charlie']
grades = [92, 85, 76]
```

1. **Grade Distribution**: We can use `len()` to find out how many students are there:

```
print(len(students))  # Output: 3
```

2. **Sorting Grades**: Sorting the list of grades in ascending order using `.sort()`.

```
grades.sort()
print(grades)  # Output: [76, 85, 92]
```

3. **Finding Highest Grade**: Using `max()` to find the highest grade.

```
print(max(grades))  # Output: 92 (Highest grade)
```

4. **Calculating Average Grade**: Summing up all grades and dividing by total students using `sum()` and `len()`.

```
average_grade = sum(grades) / len(students)
print(average_grade)  # Output: ~85.33 (Average grade)
```

These examples demonstrate how the list methods and functions can be applied in practical scenarios to manage, manipulate, and analyze data effectively.

This covers a wide range of operations that are commonly used when working with lists in Python, providing you with powerful tools for data manipulation tasks.