Самое важное

[Документация по спискам]

Метод insert
Метод позволяет **вставить** указанный объект на указанное место в списке.
Синтаксис
<список>.insert(<индекс>,<элемент>)

Пример
apples = [3, 4, 2]
apples.insert(1, 9)
apples → [3, 9, 4, 2]

Обратите внимание: элемент встал не после индекса 1, а на место под индексом 1, при этом элемент, который находился там ранее, сдвинулся на +1 индекс. Это, в свою очередь, привело к сдвигу и всех остальных элементов после него.

Метод remove
Позволяет **найти** и **удалить** элемент, который **первым** был найден по указанному значению.

Синтаксис
<список>.remove(<элемент>)

Пример
apples = [3, 4, 2, 4]
apples.remove(4)
apples → [3, 2, 4]

Обратите внимание, что удалена была только одна четвёрка, та, которая встречается раньше, если начинать поиск с начала.

Метод index
Позволяет **найти** указанный объект в списке. Этот метод возвращает **индекс первого** найденного объекта по указанному значению.

Синтаксис
<список>.index(<элемент>)

Пример
apples = [3, 4, 2, 4]
print(apples.index(4)) → 1
Возвращён один индекс, хотя элементов в списке два, при этом возвращён опять был индекс первого найденного элемента.

При желании мы можем уточнить, в какой части списка надо производить поиск: sequence.index(x[, i[, j]]) — так выглядит описание параметров в документации.

apples = [3, 4, 2, 4]
print(apples.index(4, 2, 4)) — так мы можем указать границы,
2 — индекс, с которого надо начать поиск,
4 — индекс, на котором надо закончить поиск.
В этом случае ответ будет уже не 1, а 3, так как мы указали, что поиск надо вести, начиная со второго индекса, и метод не проходил по первым двум числам с индексами 0 и 1.

Метод extend
С его помощью можно добавить элементы последовательности к текущему списку.

Синтаксис
<список>.extend(<элемент>)

Пример
apples = [3, 4, 2, 4]
bananas = [5, 2]
apples.extend(bananas)
apples → [3, 4, 2, 4, 5, 2]

Важно: этот метод изменяет текущий список, не создавая при этом новый.
Аналог такой операции — apples_and_bananas = apples + bananas. Но различие в том, что суммирование будет создавать третий, новый, список с элементами двух списков. В некоторых случаях это необходимо, но помните: новый список будет занимать дополнительное место, и, если вам не нужны старые списки по отдельности, эффективнее будет использовать extend().

Метод count
Возвращает количество элементов внутри вашего списка.

Синтаксис
<список>.count(<элемент>)

Пример
apples = [3, 4, 2, 4]
print(apples.count(4)) → результатом будет число 2.

Вложенные списки
Элементом списка может быть и другой список, что приведёт к усложнению структуры, но позволит применять списки для решения большего количества задач.

Пример

```
apples = [3, 4, 2, 4]
bananas = [9, 0]
apples.append(bananas)
print(apples) → [3, 4, 2, 4, [9, 0]]
```

Помните: хотя структура и усложнилась, правила работы со списком остались всё теми же.
Шаги, позволяющие получить доступ к элементам вложенного списка:

1) надо обратиться к самому вложенному списку: apples[4] → [9, 0];
2) apples[4] уже будет ссылкой на простой список, а значит, к apples[4] мы можем обращаться как к обычному списку.

Если добавляем ещё один индекс apples[4][0], то получаем элемент 9 из вложенного списка. Мы также можем использовать методы списка:
apples[4].append(3)
[3, 4, 2, 4, [9, 0, 3]] — получим такой список.

Не допускай следующих ошибок!

Не забывайте, что append добавляет указанный элемент в конец списка в том виде, в котором он находится на момент добавления:

```
apples = [3, 4, 2, 4]
bananas = [9, 0]
apples.append(bananas)
```

В результате такой операции мы получим [3, 4, 2, 4, [9, 0]], мы добавим вложенный список, но не соединим элементы двух списков. Для объединения элементов нам нужен метод extend(): apples.extend(bananas).

Задумывайтесь над сложностью операций:

- count() — будет проходить по всем элементам списка и проверять их, то есть если у вас будет список с сотней элементов, count сделает как минимум 100 операций.
- insert() — не только вставляет элемент внутрь списка, но ещё и сдвигает все элементы после вставленного на +1. Если у вас будет 200 элементов и вы через insert решите вставить элемент в начало, то Python придётся выполнять 200 операций сдвига.

Помня об этом, старайтесь экономно использовать подобные операции.