Тема урока: методы списков

1. Метод `insert()`
2. Метод `index()`
3. Метод `remove()`
4. Метод `pop()`
5. Метод `reverse()`
6. Метод `count()`
7. Метод `clear()`
8. Метод `copy()`
9. Метод `sort()`

Аннотация. Другие методы списков.

Мы уже познакомились с двумя списочными методами `append()` и `extend()`. Первый добавляет в конец списка один новый элемент, а второй расширяет список другим списком. К спискам в Python применимы и другие удобные методы, с которыми мы познакомимся в этом уроке.

Метод insert()

Метод `insert()` позволяет вставлять значение в список в заданной позиции. В него передается два аргумента:

1. `index` : индекс, задающий место вставки значения;
2. `value` : значение, которое требуется вставить.

Когда значение вставляется в список, список расширяется в размере, чтобы разместить новое значение. Значение, которое ранее находилось в заданной индексной позиции, и все элементы после него сдвигаются на одну позицию к концу списка.

Приведённый ниже код:

```
names = ['Gvido', 'Roman', 'Timur']
print(names)
names.insert(0, 'Anders')
print(names)
names.insert(3, 'Josef')
print(names)
```

выводит:

```
['Gvido', 'Roman', 'Timur']
['Anders', 'Gvido', 'Roman', 'Timur']
['Anders', 'Gvido', 'Roman', 'Josef', 'Timur']
```

Если указан недопустимый индекс, то во время выполнения программы не происходит ошибки. Если задан индекс за пределами конца списка, то значение будет добавлено в конец списка. Если применен отрицательный индекс, который указывает на недопустимую позицию, то значение будет вставлено в начало списка.

Метод index()

Метод `index()` возвращает индекс первого элемента, значение которого равняется переданному в метод значению. Таким образом, в метод передается один параметр:

1. `value` : значение, индекс которого требуется найти.

Если элемент в списке не найден, то во время выполнения происходит ошибка.

Приведённый ниже код:

```
names = ['Gvido', 'Roman', 'Timur']
position = names.index('Timur')
print(position)
```

выводит:

```
2
```

Приведённый ниже код:

```
names = ['Gvido', 'Roman', 'Timur']
position = names.index('Anders')
print(position)
```

приводит к возникновению ошибки:

```
ValueError: 'Anders' is not in list
```

Чтобы избежать таких ошибок, можно использовать метод `index()` вместе с оператором принадлежности `in` :

```
names = ['Gvido', 'Roman', 'Timur']
if 'Anders' in names:
    position = names.index('Anders')
    print(position)
else:
    print('Такого значения нет в списке')
```

Метод remove()

Метод `remove()` удаляет первый элемент, значение которого равняется переданному в метод значению. В метод передается один параметр:

1. `value` : значение, которое требуется удалить.

Метод уменьшает размер списка на один элемент. Все элементы после удаленного элемента смещаются на одну позицию к началу списка. Если элемент в списке не найден, то во время выполнения происходит ошибка.

Приведённый ниже код:

```
food = ['Рис', 'Курица', 'Рыба', 'Брокколи', 'Рис']
print(food)
food.remove('Рис')
print(food)
```

выводит:

```
['Рис', 'Курица', 'Рыба', 'Брокколи', 'Рис']
['Курица', 'Рыба', 'Брокколи', 'Рис']
```

Важно: метод `remove()` удаляет только первый элемент с указанным значением. Все последующие его вхождения остаются в списке. Чтобы удалить все вхождения, нужно использовать цикл `while` в связке с оператором принадлежности `in` и методом `remove` .

Метод pop()

Метод `pop()` удаляет элемент по указанному индексу и возвращает его. В метод `pop()` передается один **необязательный** аргумент:

1. `index` : индекс элемента, который требуется удалить.

Если индекс не указан, то метод удаляет и возвращает последний элемент списка. Если список пуст или указан индекс за пределами диапазона, то во время выполнения происходит ошибка.

Приведённый ниже код:

```
names = ['Gvido', 'Roman', 'Timur']
item = names.pop(1)
print(item)
print(names)
```

выводит:

```
Roman
['Gvido', 'Timur']
```

Метод count()

Метод `count()` возвращает количество элементов в списке, значения которых равны переданному в метод значению.

Таким образом, в метод передается один параметр:

1. `value` : значение, количество элементов, равных которому, нужно посчитать.

Если значение в списке не найдено, то метод возвращает 0.

Приведённый ниже код:

```
names = ['Timur', 'Gvido', 'Roman', 'Timur', 'Anders', 'Timur']
cnt1 = names.count('Timur')
cnt2 = names.count('Gvido')
cnt3 = names.count('Josef')

print(cnt1)
print(cnt2)
print(cnt3)
```

выводит:

```
3
1
0
```

Метод reverse()

Метод `reverse()` инвертирует порядок следования значений в списке, то есть меняет его на противоположный.

Приведённый ниже код:

```
names = ['Gvido', 'Roman', 'Timur']
names.reverse()
print(names)
```

выводит:

```
['Timur', 'Roman', 'Gvido']
```

Существует большая разница между вызовом метода `names.reverse()` и использованием среза `names[::-1]` . Метод `reverse()` меняет порядок элементов на обратный **в текущем списке**, а срез создаёт копию списка, в котором элементы следуют в обратном порядке.

Метод clear()

Метод `clear()` удаляет все элементы из списка.

Приведённый ниже код:

```
names = ['Gvido', 'Roman', 'Timur']
names.clear()
print(names)
```

выводит:

```
[]
```

Метод copy()

Метод `copy()` создаёт поверхностную копию списка.

Приведённый ниже код:

```
names = ['Gvido', 'Roman', 'Timur']
names_copy = names.copy()              # создаем поверхностную копию списка names

print(names)
print(names_copy)
```

выводит:

```
['Gvido', 'Roman', 'Timur']
['Gvido', 'Roman', 'Timur']
```

Аналогичного результата можно достичь с помощью срезов или функции `list()` :

```
names = ['Gvido', 'Roman', 'Timur']
names_copy1 = list(names)              # создаем поверхностную копию с помощью функции list()
names_copy2 = names[:]                 # создаем поверхностную копию с помощью среза от начала до конца
```

Примечания

Примечание. Существует большая разница в работе строковых и списочных методов. Строковые методы не изменяют содержимого объекта, к которому они применяются, а возвращают новое значение. Списочные методы, напротив, меняют содержимое объекта, к которому применяются.

❤ Happy Pythoning!