8. Списки в Python. Ключевые операции, проводящие к изменению списка и порождающие измененные списки.

Что такое список и что он делает - билет 7. Списки в Python. Обращение к элементам списка и создание срезов. Обход списка и поиск элементов в списке.

Изменение списка с помощью его базовых методов
.append(<Объект>) - добавляет один объект в конец списка. Является наиболее эффективной операцией (с точки зрения затраты вычислительных ресурсов) изменения списков.
Про возможности используй его. Он твой БРО.

```
>>> s = [1,2,3,4]
>>> s.append("koshkas")
>>> s
[1, 2, 3, 4, 'koshkas']
>>>
```

.extend(<Последовательность>) - добавляет элементы последовательности в конец списка.

```
>>> s = [1,2,3,4]
>>> m = [6,4,2]
>>> s.extend(m)
>>> s
[1, 2, 3, 4, 6, 4, 2]
>>>
```

.insert (<Индекс>, <Объект>) - добавляет один объект в указанную позицию. Остальные элементы смещаются. Метод insert() позволяет добавить только один объект. Чтобы добавить несколько объектов, можно воспользоваться операцией присваивания значения срезу.

```
>>> l = [1,2,3,4]
>>> l.insert(1,"koshkas")
>>> l
[1, 'koshkas', 2, 3, 4]
>>>
```

.remove (<Значение>) - удаляет первый элемент, содержащий **указанное значение**. Если элемент не найден, возбуждается исключение ValueError. Метод изменяет текущий список и ничего не возвращает.

```
l = list(range(5))
l.reverse() #[4, 3, 2, 1, 0]
l.remove(3) #[4, 2, 1, 0]
```

Все методы выше не возвращают какие-либо значения, лишь только изменяют текущий список.

.pop([<Индекс>]) - удаляет элемент, расположенный по указанному индексу, и возвращает его.
— Если индекс не указан, то удаляет и возвращает последний элемент списка.
— Если элемента с указанным индексом нет или список пустой, то IndexError.

```
>>> l = [str(x) for x in range(1,5)]
>>> l
['1', '2', '3', '4']
>>> l.pop(2)
'3'
>>> l
['1', '2', '4']
>>>
```

Изменение списка с помощью срезов

С помощью среза можно изменить фрагмент списка.
Вставка нескольких элементов при помощи среза (без потери существующих элементов):

```
l = list(range(1, 5)) #[1, 2, 3, 4]
l[2:2] = ['a', 'b', 'c']
print(l) #[1, 2, 'a', 'b', 'c', 3, 4]
```

Можно вставлять несколько элементов при помощи среза, замещая уже существующие элементы:

```
l = list(range(1, 5)) #[1, 2, 3, 4]
l[1:3] = ['a', 'b', 'c']
print(l) #[1, 'a', 'b', 'c', 4]
```

Если срезу присвоить пустой список, то элементы, попавшие в срез, будут удалены:

```
l = list(range(1, 5)) #[1, 2, 3, 4]
print(l)
l[1:3] = []
print(l) #[1, 4]
```

Инструкция del

Может удалять из списка как единичные элементы, так и элементы, получаемые при помощи среза.
Срез:

```
l = [5,3,6,2,9]
del l[:1]
print(l) # [3, 6, 2, 9]
```

Единичный элемент:

```
l = [5,3,6,2,9]
del l[1]
print(l) # [5, 6, 2, 9]
```

Проверка на вхождение элемента в список при помощи in

Оператор in осуществляет проверку на вхождение элемента в список. Если элемент входит в список, то возвращается значение True, в противном случае - False.

Оператор in не дает никакой информации о местонахождении элемента внутри списка.

```
l = ["koshka"+str(e) for e in range(4,0,-1)]
#l = ['koshka4', 'koshka3', 'koshka2', 'koshka1']
if "koshka3" in l:
print("Элемент есть в списке")
```

Сортировка

Отсортировать список позволяет метод sort(). Метод изменяет текущий список и ничего не возвращает.

— key - используется лямбда-выражение вида `key=lambda x: (x[0])`

— reverse - использование обратной сортировки, значения True или False

```
from random import shuffle
l = [(e+1,"koshka"+str(e)) for e in range(1,5)]
shuffle(l) # [(3, 'koshka2'), (5, 'koshka4'), (2, 'koshka1'), (4, 'koshka3')]
l.sort(key=lambda e: e[0], reverse=True)
#[(5, 'koshka4'), (4, 'koshka3'), (3, 'koshka2'), (2, 'koshka1')]
```

В некоторых случаях необходимо получить отсортированный список, а текущий список оставить без изменений. Для этого следует воспользоваться функцией sorted().

— В первом параметре указывается список, который необходимо отсортировать.

— Остальные параметры эквивалентны параметрам метода sort().

```
from random import shuffle
l = [(e+1,"koshka"+str(e)) for e in range(1,5)]
shuffle(l) # [(3, 'koshka2'), (5, 'koshka4'), (2, 'koshka1'), (4, 'koshka3')]
new_l = sorted(l, key=lambda e: e[0], reverse=True)
print(l) # [(3, 'koshka2'), (5, 'koshka4'), (2, 'koshka1'), (4, 'koshka3')]
print(new_l) # [(5, 'koshka4'), (4, 'koshka3'), (3, 'koshka2'), (2, 'koshka1'
)]
```

Инвертирование списка

Операция переворачивает список изменяя исходный объект:

lst8.reverse()

```
>>> l
['1', '2', '4']
>>> l.reverse()
>>> l
['4', '2', '1']
>>>
```

Если необходимо изменить порядок следования и получить новый список, то следует воспользоваться функцией reversed(<Последовательность>). Функция возвращает итератор, который можно преобразовать в список с помощью функции list():

```
l = [1,2,3,4]
new_l = list(reversed(l))
print(l) #[1, 2, 3, 4]
print(new_l) #[4, 3, 2, 1]
```