Срезы списков и сравнение списков

Мы продолжаем изучение списков языка Python. Это занятие начнем со срезов. О срезах мы с вами уже говорили, когда рассматривали строки и с их помощью выделяли наборы символов из строк. Со списками все работает аналогично: срезы позволяют выделять наборы элементов. Например, у нас есть список:

```
lst = ["Москва", "Уфа", "Тверь",
```

и мы хотим выбрать из него 2-й и 3-й элементы. Это можно сделать, так:

```
lst[1:3]
```

Давайте разберемся, как это работает. Синтаксис для срезов имеет вид:
список[start:end]

В данном случае, мы указываем стартовый индекс 1 (второй элемент) и конечный 3 (последний элемент, до которого выделяем срез, не включая его). В итоге, на выходе получаем список из двух элементов:

['Уфа', 'Тверь']

То есть, это новый список, состоящий из выделенных элементов. Мы в этом можем легко убедиться. Сохраним через переменную результат среза:

```
a = lst[1:3]
```

Затем, изменим его первый элемент: `a[0] = "Воронеж"`

и, как видим, это никак не повлияло на исходный список lst.

В срезах можно указывать только начальный индекс: `lst[2:]`

тогда все будет выбрано с 3-го элемента и до конца списка. Или только конечный индекс: `lst[:3]`

Обратите внимание, последний указанный индекс не включается в срез. Если же не указывать ни первый, ни последний индексы, то получим копию исходного списка: `cities = lst[:]`

Здесь создается именно копия списка. Если мы посмотрим на их id:

`d = lst` будет ли здесь создаваться копия списка? Вспоминая занятие по переменным, мы говорили, что переменные – это ссылки на объекты и оператор присваивания лишь копирует ссылку, но не сам объект. Поэтому переменные d и lst будут ссылаться на один и тот же список, их id:

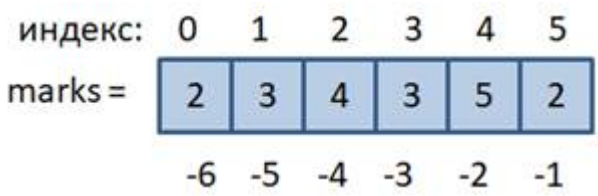
равные. Ну а наиболее сомневающимся предлагаю изменить значение списка через d:

```
d[0] = "Самара"
```

и посмотреть на список lst. В нем произойдет такое же изменение.

Далее, так как у списка имеются отрицательные индексы, то они также могут быть использованы при определении срезов. Например, создадим список оценок:

```
marks = [2, 3, 4, 3, 5, 2]
```



и запишем следующие срезы:

```
marks[2:-1]
marks[-3:-1]
```

Дополнительно в срезах можно указывать шаг перебора элементов, согласно синтаксису:

список[start:stop:step]

Например:

```
marks[1:5:2]
```

или без указания границ:

```
marks[:5:2]
marks[1::2]
marks[::2]
```

Если шагом является отрицательное число, то перебор элементов осуществляется с конца списка, например:

```
marks[::-1]
marks[::-2]
```

Вообще, механизм срезов работает абсолютно также как и со строками, только здесь они применяются к спискам.

Но, учитывая, что списки относятся к изменяемым типам данных, то со срезами можно выполнять одну дополнительную операцию –изменение группы элементов. Например, для списка marks мы хотим 3-ю и 4-ю оценки представить в виде строк. Это можно сделать, следующим образом:

```
marks[2:4] = ["хорошо", "удовлетв"]
```

И теперь коллекция содержит данные: [2, 3, 'хорошо', 'удовлетв.', 5, 2]

Видите, как легко и просто это можно сделать. Или, другой пример, перебрать все элементы через один и присвоить им 0:

```
marks[::2] = [0, 0, 0]
```

Правда, такая конструкция содержит потенциальную ошибку. Если увеличить размер списка: `marks += [3]`

и повторить операцию: `marks[::2] = [0, 0, 0]`

то получим ошибку, так как число замен здесь уже четыре, а не три. Поэтому, в таких присваиваниях лучше явно указывать границы срезов:

```
marks[:5:2] = [0, 0, 0]
```

Также, для группового присваивания можно использовать и такой синтаксис:

```
marks[2:4] = 10, 20
```

Сравнение списков

В заключение этого занятия рассмотрим возможности сравнения списков между собой с помощью операторов: >, <, ==, !=

```
[1, 2, 3] == [1, 2, 3]   # True
[1, 2, 3] != [1, 2, 3]   # False
[1, 2, 3] > [1, 2, 3]    # False
```

В последнем сравнении получим False, т.к. списки равны

то первый список будет больше второго. Здесь сравнение больше, меньше выполняется по тому же принципу, что и у строк: перебираются последовательно элементы, и если текущий элемент первого списка больше соответствующего элемента второго списка, то первый список больше второго. И аналогично, при сравнении меньше:

```
[10, 2, 3] < [1, 2, 3]  # False
```

Все эти сравнения работают с однотипными данными:

```
[1, 2, "abc"] > [1, 2, "abc"]  #
```

сработает корректно, а вот так: `[1, 2, 3] > [1, 2, "abc"]`

произойдет ошибка, т.к. число 3 не может быть сравнено со строкой «abc».