Урок №10. Срезы. Равные и совпадающие объекты.

План урока

1. Срезы.
2. Переменная как указатель на объект.
3. Оператор is.

Аннотация

В уроке рассматриваются всевозможные виды срезов (`slices`). Поскольку полный срез — естественный способ скопировать список, далее рассматривается тот нетривиальный факт, что в Питоне переменная служит лишь указателем на объект. Обсуждается разница между равными и совпадающими объектами и последствия этой разницы для объектов-списков.

Как мы узнали на прошлом занятии, цикл `for` позволяет пройти непосредственно по элементам списка (или иного контейнера). На первый взгляд перебор индексов с помощью `for i in range(...)` кажется более гибким: можно перебрать не все индексы, можно идти с шагом, скажем, 2 или даже -1, то есть в обратном порядке. Но существует способ без всякого цикла преобразовать список нужным образом: взять отдельный его кусок и т. д. Этот способ — **срез** (slice).

В самом простом варианте срез списка — это его кусок от одного индекса включительно до другого — не включительно (как для `range`). То есть это новый, более короткий список. Срез записывается с помощью квадратных скобок, в которых указывается начальный и конечный индекс, разделённые двоеточием:

```
months = ['январь', 'февраль', 'март', 'апрель', 'май', 'июнь', 'июль', 'август',
          'сентябрь', 'октябрь', 'ноябрь', 'декабрь']
spring = months[2:5]  # spring == ['март', 'апрель', 'май']
for month in spring:
    print(month)
```

Разрешены отрицательные индексы для отсчёта с конца списка. Если не указан начальный индекс, срез берётся от начала (от 0). Если не указан конечный индекс, срез берётся до конца. В следующем примере список `winter` составлен из двух срезов: от последнего элемента до конца (в таком срезе всего один элемент) и от начала до элемента с индексом 2 не включительно. Заметьте, что когда конечный индекс среза совпадает с начальным индексом другого среза, эти срезы содержат соседние части списка, но не пересекаются.

```
months = ['январь', 'февраль', 'март', 'апрель', 'май', 'июнь', 'июль', 'август',
          'сентябрь', 'октябрь', 'ноябрь', 'декабрь']
winter, spring, summer, fall = months[-1:] + \
    months[:2], months[2:5], months[5:8], months[8:11]
```

В частности, `a[:]` — это срез, который содержит все элементы списка a, т. е. полная его копия. Позже мы поговорим, зачем может потребоваться создать копию списка.

Срезы строк и кортежей делаются точно так же, как срезы списков: `'Не ходите в Африку гулять!'[3:]` даст строку `'ходите в Африку гулять!'`

Как и для `range`, в параметры среза можно добавить третье число — шаг обхода. Оно записывается через второе двоеточие, необязательное для среза.

```
months = ['январь', 'февраль', 'март', 'апрель', 'май', 'июнь',
          'июль', 'август', 'сентябрь', 'октябрь', 'ноябрь', 'декабрь']
print('Каждый второй месяц, начиная с января:', months[::2])
print('Каждый второй месяц, начиная с февраля:', months[1::2])
print(months[2:9:3])  # months[2], months[5], months[8] - март, июнь, сентябрь
```

Шаг может быть и отрицательным — для прохода по списку в обратном порядке. Если в этом случае не указать начальный и конечный индекс среза, ими станут последний и первый индексы списка, соответственно (а не наоборот, как при положительном шаге):

```
months = ['январь', 'февраль', 'март', 'апрель', 'май', 'июнь',
          'июль', 'август', 'сентябрь', 'октябрь', 'ноябрь', 'декабрь']
print('В обратном порядке:')
for month in months[::-1]:
    print(month)
```

Задачи:

- **Фильтр (https://lms.yandexlyceum.ru/task/view/982)** ,
- **От и до (https://lms.yandexlyceum.ru/task/view/983) +** ,
- **Анонс новости (https://lms.yandexlyceum.ru/task/view/984)** ,
- **Найди кота — 5 (https://lms.yandexlyceum.ru/task/view/985)**

Необязательные задачи:

- **Масштабирование (https://lms.yandexlyceum.ru/task/view/988)** ,
- **Вредные советы (https://lms.yandexlyceum.ru/task/view/989)** ,
- **Буквоедство (https://lms.yandexlyceum.ru/task/view/990)** ,
- **Крупные буквы — 2 (https://lms.yandexlyceum.ru/task/view/991).**

В простейшем случае переменную можно представить себе как ящик с табличкой-именем, в который положили (как правило, при помощи оператора присваивания) то или иное значение. Однако в Питоне такое представление не совсем верно. Более точная метафора может быть такой: память компьютера — банковское хранилище. Данные лежат в разных ячейках этого хранилища, а переменная — это ключ от одной из ячеек, на который повесили уникальный брелок с именем переменной. Отличие в том, что если рядом стоят две коробки, то в них не может лежать один и тот же предмет (но могут лежать одинаковые), а вот у одной банковской ячейки могут быть два ключа. Иными словами, две переменные с разными именами могут быть связаны с одними и теми же данными или, точнее, **указывать** на один и тот же **объект**.

Что это меняет? Если этот объект — число, строка или кортеж, то практически ничего. Но в случае списка это важно, потому что список можно изменить.

Рассмотрим, например, числа. Оператор `x = x * 3 + 1` не меняет число, которое было значением переменной x: он вычисляет новое значение, помещает его в новую «банковскую ячейку» (область памяти) и делает так, чтобы переменная x была связана с этой ячейкой — вешает «брелок» x на новый «ключ». Старое значение при этом продолжает существовать, просто переменная x с ним больше не связана. Если значение больше не нужно, то область памяти, которую оно занимает, будет автоматически помечена как свободная. Программисту о таких объектах заботиться не нужно, за них отвечает специальный компонент интерпретатора — **сборщик мусора**. Содержимое соответствующей «банковской ячейки» — число, строка или кортеж — никогда не меняется. Оно может только отправиться в мусор.

Однако оператор `a.append(1)` меняет именно список, с которым связано имя a, то есть само содержимое «банковской ячейки».

Рассмотрим пример:

```
a1 = [1, 2, 3]
a2 = [1, 2, 3]
a3 = a1
a4 = a1[:]  # срез, содержащий все элементы a1 -- копия a1
a3.append(100)
print(a1, a2, a3, a4)
```

Такая программа выведет `[1, 2, 3, 100] [1, 2, 3] [1, 2, 3, 100] [1, 2, 3]`, потому что переменные `a1` и `a3` связаны с одним и тем же списком, а переменные `a2` и `a4` — с другими списками, хотя все они и были равны друг другу до изменения списка `a3` (он же `a1`).

Вот почему иногда нужно делать копию списка при помощи полного среза `[:]`.

Есть и другие методы, изменяющие списки. Мы рассмотрим их на следующем занятии. Кроме списков бывают и другие изменяемые объекты, которые мы тоже изучим в своё время.

Проверить, связаны ли две переменные с одним и тем же объектом, можно с помощью оператора `is`. Так, для предыдущего примера проверка `a1 is a2`, как и `a2 is a1 or a1 is a4 or a2 is a4`, выдаст `False`, а проверка `a1 is a3` выдаст `True`. При этом `a1 == a2 == a3 == a4` до изменения `a3`, безусловно, выдало бы `True`.

Заметьте, что в рамках метафоры с ключами и банковскими ячейками список — это лишь сложенный в ячейку набор ключей от других, отдельных ячеек со значениями элементов списка (в ячейку не запихивают значения, соответствующие всем возможным индексам списка). Когда в цикле `for` счётчик проходит по значениям из списка, «брелок» с таким именем последовательно цепляется на «ключи от ячеек» с элементами списка — иными словами, переменная-итератор в цикле `for` по списку указывает на сам элемент списка, а не на его копию:

```
table = [[1, 2], [3, 4]]
for row in table:
    row[0] = 0
    row.append(5)
print(table)  # выведет [[0, 2, 5], [0, 4, 5]]
```

Также отметим, что в случае списков оператор `a1 += a2` ведёт себя **не совсем** как a1 = a1 + a2. В первом случае изменяется сам список a1 (к его концу дописываются все элементы списка a2), во втором — создаётся новый. В случае чисел, строк и кортежей, которые изменяться не могут, две эти формы записи полностью эквивалентны.

Различие между переменными, связанными с одним и тем же объектом, и переменными, связанными с равными объектами, позже понадобится нам, когда мы будем писать собственные функции.