Самое важное

Списковые сборки (list comprehensions)
По сути, это цикл, заполняющий список элементами, то есть можно сказать, что списковая сборка аналогична по результату циклу подобного вида:
x = [ ] — создаём новый список,
for i in range(10): — запускаем простой цикл с 10 итерациями,
    x.append(i) — добавляем каждый элемент цикла в список.

Такое же создание и наполнение списка мы можем записать проще: x = [i for i in range(10)].

При этом мы можем как-либо усложнять генерацию элементов:
x = [(i**3 + 2**i) for i in range(10)].
x → [1, 3, 12, 35, 80, 157, 280, 471, 768, 1241]
Таким образом можно визуально разделить сборку на две части:
[(i**3 + 2**i) for i in range(10)].
Зелёная — описание элемента, который будет добавляться в список.
Жёлтая — создание цикла с переменной цикла, который в первую очередь определяет, сколько будет элементов в списке, а во вторую — может задать изменение элементов (в нашем случае он изменяет ‘i’, из которого генерируются элементы списка от 0 до 9 включительно).

Так, например, мы можем вовсе не использовать переменную цикла в сборке:
[5 for i in range(10)] → [5, 5, 5, 5, 5, 5, 5, 5, 5, 5].

Но мы можем добавлять насколько угодно сложную обработку для переменных цикла:

```
def func(x):
    return min((x ** 3 + 2 * x), (x * x + 2 ** x + 92))

numbers = [func(i) for i in range(20)]
```

[0, 3, 12, 33, 72, 135, 192, 269, 412, 685, 1020, 1353, 1752, 2223, 2772, 3405, 4128, 4947, 5868, 6897]

Условия в списковых сборках
В области описания элемента (зелёной зоне) мы можем также использовать условные операторы: numbers = [func(i) if i > 10 else 0 for i in range(20)].

Такое условие заставит Python выбирать один из вариантов элемента для списка, то есть теперь у него два варианта: либо мы добавим в список элемент ‘i’, который прошёл через функцию func, либо добавим просто значение 0. Сам выбор зависит от условия i > 10.

Если оно выполняется, мы добавляем в список ‘i’, прошедшее через func. Иначе — добавляем 0. Получаем в итоге подобный список: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1353, 1752, 2223, 2772, 3405, 4128, 4947, 5868, 6897].

Модуль random

В первую очередь стоит заглянуть в документацию. Сам по себе этот модуль позволяет нам добавить фактор случайности при работе с числами. Так, например, функция randint из этого модуля позволяет выбрать одно из случайных целых чисел в указанном диапазоне:

- import random — импортирование модуля, чтобы его функции были доступны в нашей программе;
- random.randint(0, 100) — такая операция выдаст нам случайное целое число от 0 до 100;
- random.random() — вернёт случайное вещественное число от 0 до 1.

Если же вам нужно будет указать диапазон для вещественных чисел, то нужна будет функция random.uniform(a, b), где при помощи параметров a и b можно будет задать начало и конец диапазона для генерации случайного вещественного числа.

Срезы списков

Срезы — инструмент, который позволяет обратиться к какой-либо части вашего списка, указав при помощи трёх переменных индексы, которые должны входить в новый список.

Эти три переменные напоминают аналогичные параметры функции range(start, stop, step).

Мы должны указать начальный индекс, конечный индекс (элемент с последним индексом не будет включен в срез, так же как и в range) и шаг, с которым мы будем брать элементы.

Например:
peoples = [19, 42, 20, 32, 65]
peoples_all = peoples[:] — подобная хитрость, когда мы не указываем начало и конец, позволяет нам сделать независимую копию списка, изменение которой не приведёт к изменению оригинала.

peoples_half = peoples[2:5] — так мы получим список [20, 32, 65] с частью элементов, начиная со 2-го индекса и заканчивая 4-м (5-й не включается).

Кстати, если мы укажем индекс, которого нет, то ошибки не будет:
peoples_half = peoples[2:10] — Python дойдёт до последнего доступного индекса и остановится — [20, 32, 65].

peoples_odd = peoples[0:5:2] — указав, помимо старта и конца, шаг, мы сможем получить часть элементов. В нашем случае это будут все нечётные элементы:

[19, 20, 65] — 1-й, 3-й и 5-й элементы списка.
Индексы этих элементов при этом будут чётными (0, 2, 4).

peoples_reverse = peoples[::-1] — в качестве шага мы также можем указывать отрицательный шаг. Подобная запись очень распространена, так как позволяет быстро получить копию списка с элементами, добавленными в обратном порядке:
[19, 42, 20, 32, 65] → [65, 32, 20, 42, 19]

Срезы и индексы в строках
Сам механизм работы с индексами и срезами строк мало отличается от работы со списками. Мы также можем указывать начальный и конечный индексы и шаг, с которым мы будем проходить от начального индекса к конечному:
hello = ‘Приветствие!’
hello[1::2] — так мы пройдём от 1-го индекса до последнего с шагом 2.
“рвтти!” — результат.
hello[::-1] — мы также можем использовать обратный срез.
“!еивтстевирП” — получим перевёрнутую строку.

Различие: строка, в отличие от списка, является неизменяемым типом данных. Поэтому мы не можем заменить какой-либо элемент строки по индексу, хотя можем заменить элемент списка:
peoples[0] = 20 — после этой операции наш список будет выглядеть так:
[20, 42, 20, 32, 65]
hello[-1] = ‘?’
А после этой операции мы получим ошибку:
TypeError: 'str' object does not support item assignment

Кстати, индекс -1 не будет причиной ошибки — это просто ещё один способ использования отрицательных значений. Индекс -1 указывает на первый элемент с конца, в нашем случае это ‘!’.

Не допускай следующих ошибок!

При работе со срезами важно понимать, что Python будет «проходить» по заданной последовательности от начала, которое вы укажете, до конца с заданным шагом.

Если же, например, вы укажете начало в индексе 10, а конец — в индексе 5, то с обычным шагом, равным 1, Python не сможет пройти от 10 до 5.
10 + 1 = 11 — то есть Python с шагом 1 будет идти вперёд, к индексам, которые больше 10. Если вам нужно пройти в обратном направлении, нужно указывать отрицательный шаг: hello[11:1:-1] → “!еивтстеви”.