**2023/11/29 00:00|Лекция. Списковые, словарные сборки.**

## Списковые, словарные сборки

На данном уроке мы с вами изучим списковые сборки и узнаем, как они связаны с функциональным программированием.

Для начала расскажем, что такое списковые сборки на небольшом примере.

Помните, у нас на прошлом уроке изучали функцию «map» и функцию «filter»? Две встроенные функции высшего порядка, которые достаточно часто используются в Python. На самом деле их не очень удобно использовать постоянно, особенно, чтобы генерировать списки. Мы все это вам наглядно покажем. У нас есть функция «def by\_3» и функция «def is\_odd». Первая функция у нас возвращает аргумент умноженный на 3, a вторая возвращает остаток деления на 2. Первая функция для фильтрации у нас будет использоваться, вторая функция будет использоваться для операций над элементом множества (Рис.1).

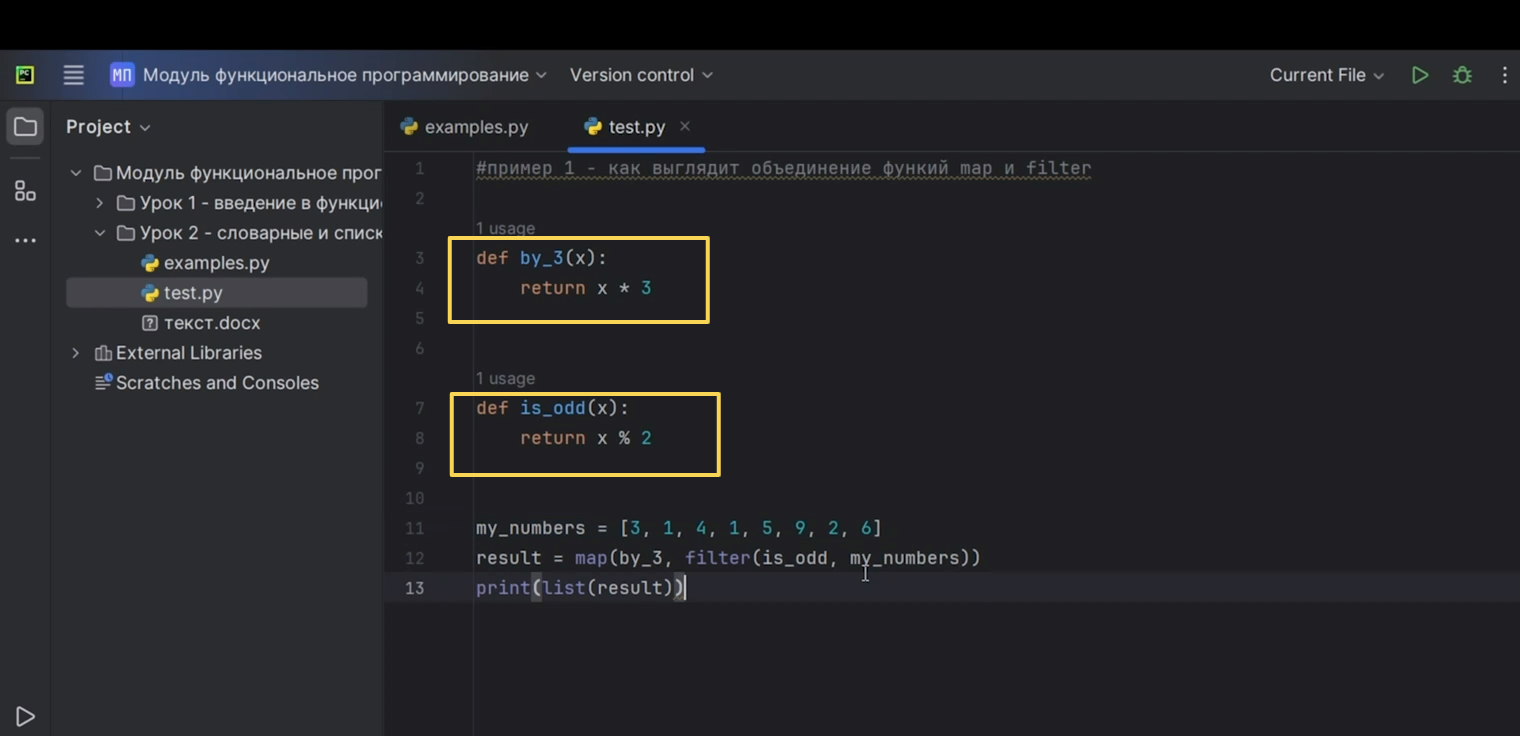


Рис.1

И точно также у нас есть список «my\_numbers», который мы помещаем в функцию «map». Но смотрите, каким необычным образом мы её сюда помещаем: в переменную «result» сохраняем результат функции «map», только вместо нашего списка «my\_numbers» мы передаём результат функции «filter», в которую мы передаём функцию «is\_odd». То есть нам будут доступны только те числа, которые у нас нечётные. И уже список, состоящий из нечётных чисел, мы будем умножаем на 3. И только после этого выводим наш результат (Рис.2). То есть в результате у нас только нечётные числа, при этом они еще и умноженные на 3.

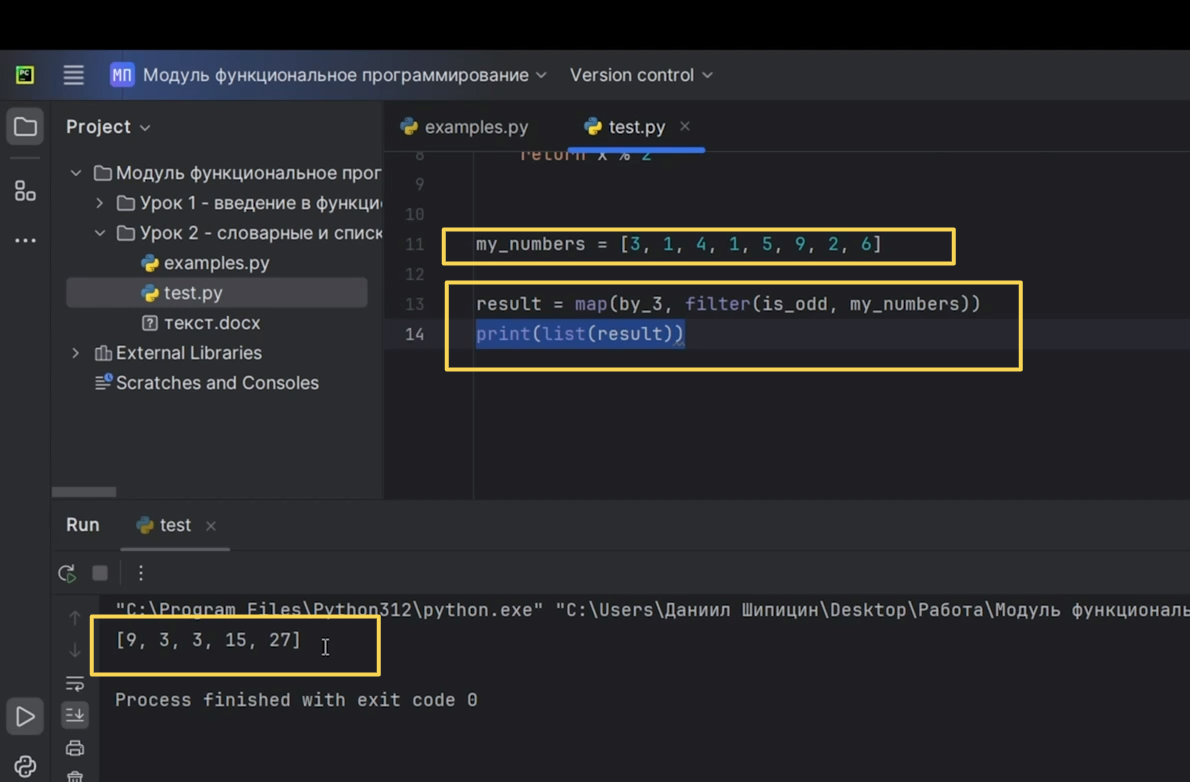


Рис.2

На самом деле это достаточно объёмная конструкция, то есть нам для того, чтобы получить результат, пришлось написать две функции. Мы можем использовать лямбда функции, сокращённые функции, помещать их прямо в переменную «result». Нам это никто не запрещает, это те же самые функции. Но из-за этого строчка становится достаточно громоздкой, нечитабельной и использовать это становится неудобно.

Программисты подумали, как исправить систему, и придумали такое решение как словарные и списковые сборки. На английском языке говорят «list comprehension». Рассмотрим все на примере.

**Синтаксис для списковых сборок. Что такое списковая сборка?**

Списковая сборка это, когда мы прямо внутрь списка, то есть в квадратные скобочки, помещаем некоторый результат, цикл «for», туда можно также помещать условия, и это все комбинируется между собой и создаётся, генерируется новый список.

В самом простом варианте мы просто можем использовать цикл «for». У нас написан синтаксис «list\_comp\_1», то есть первый список, который состоит из какого-то элемента «x» умноженного на 2. Мы можем вместо этого «х» добавлять просто элемент, либо изменённый элемент, то есть можем поменять тип данных, можем переприсваивать его, можем сделать с ним какие-то математические выражения, применить на нём какой-то метод и так далее. То есть «x\*2»-это у нас операция, это то, что будет добавляться у нас в наш новый список. Потом мы после того, как определили наш элемент и его операцию, пишем цикл «for» обязательно. Следовательно, сразу же после элемента пишем цикл «for» для этого же элемента. То есть мы добавляем список «x\*2» для таких-то «x in collection» (Рис.3).

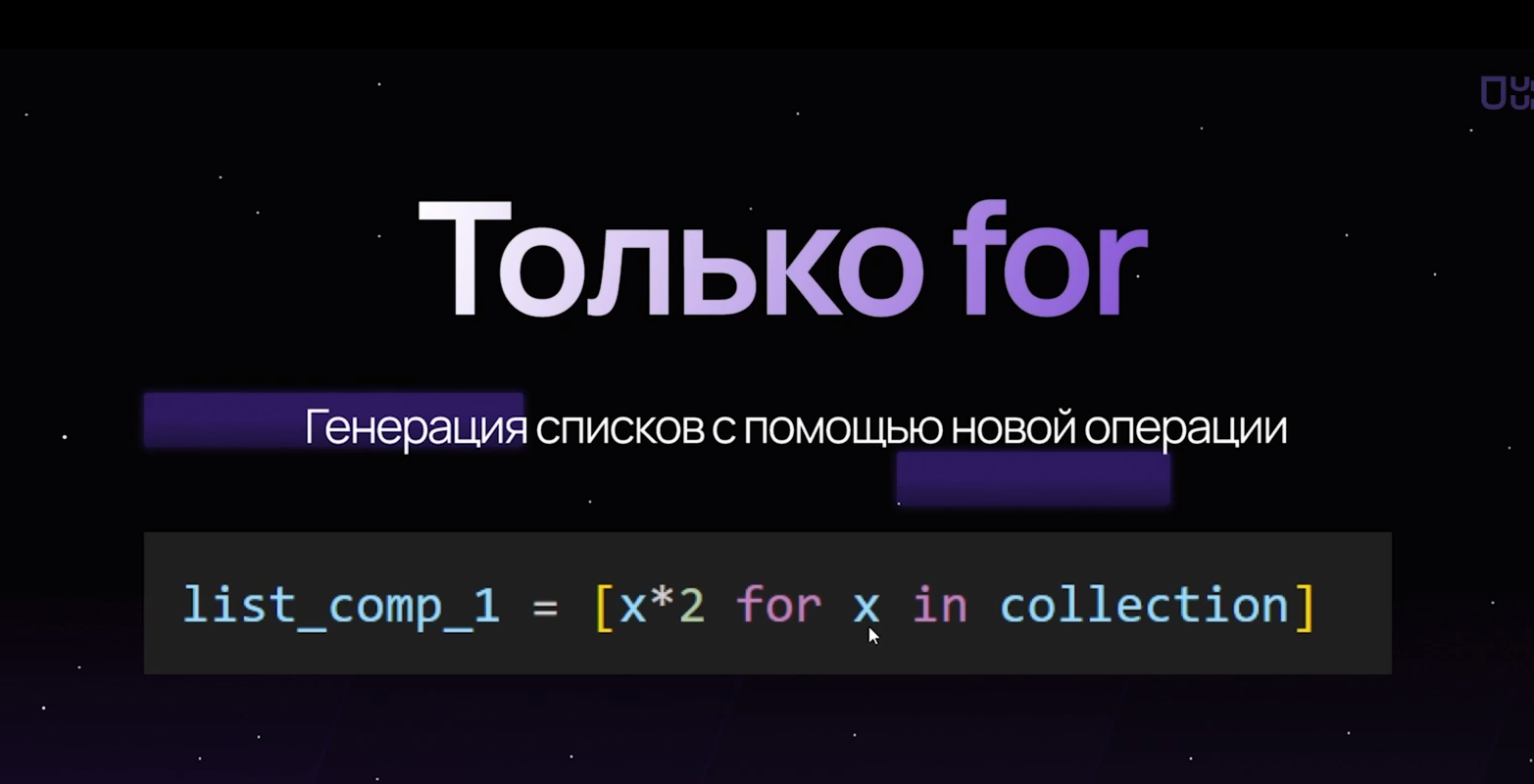


Рис.3

Мы можем использовать не только какую-то коллекцию данных, которая у нас уже до этого была, а можем, например, использовать функцию «range» точно так же. То есть вместо «collection» мы просто помещаем функцию «range», которая нам будет создавать диапазон чисел, который нам необходим. Допустим, если бы у нас был «list\_comp\_1=[x\*2 for x in range (1,5)]», тогда бы у нас добавились все числа от 1 до 5 не включительно, но они были бы у нас умножены на 2. Мы вам показали, конечно же, простой пример, но бывают очень длинные списковые сборки, и это действительно становится удобно.

У нас дан пример списковой сборки: есть просто обычный список «my\_numbers», и мы в переменную «result» сохраняем каждый «x», точнее каждый элемент, умноженный на 3 для таких-то «х» в нашей коллекции «my\_numbers». То есть все элементы из этой коллекции «my\_numbers» будут умножены на 3 (Рис4).

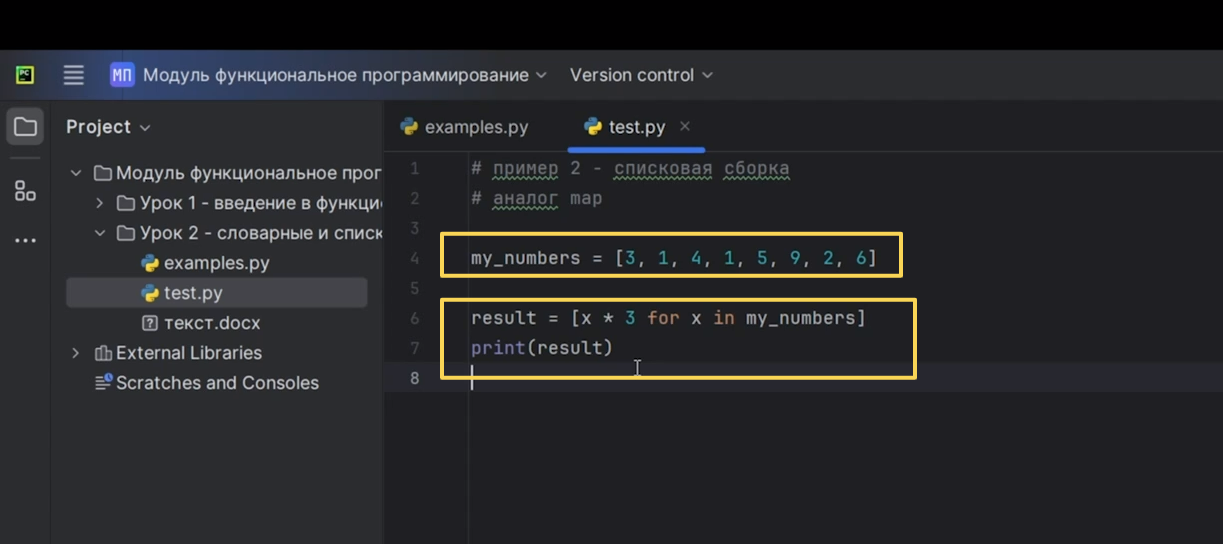


Рис.4

И, как вы можете заметить, это простой аналог «map». Это называется «синтаксический сахар», когда мы упрощаем то, что и так достаточно удобно и понятно читается. Но мы ещё это больше упрощаем, потому что вот этот код «x\*3 for x in my\_numbers» внутри списка читается намного легче, чем функция «map» (ещё непонятно, какая функция применяется и так далее). То есть именно эта операция «х\*3»-это и есть наша функция «map».

Если мы выведем в консоль, результат нашей этой списковой сборки выведется «9, 3, 12... и так далее», то есть каждый элемент нашего изначального списка, но умноженный на 3 (Рис.5). Функцию умножения мы написали прям внутри нашего списка.

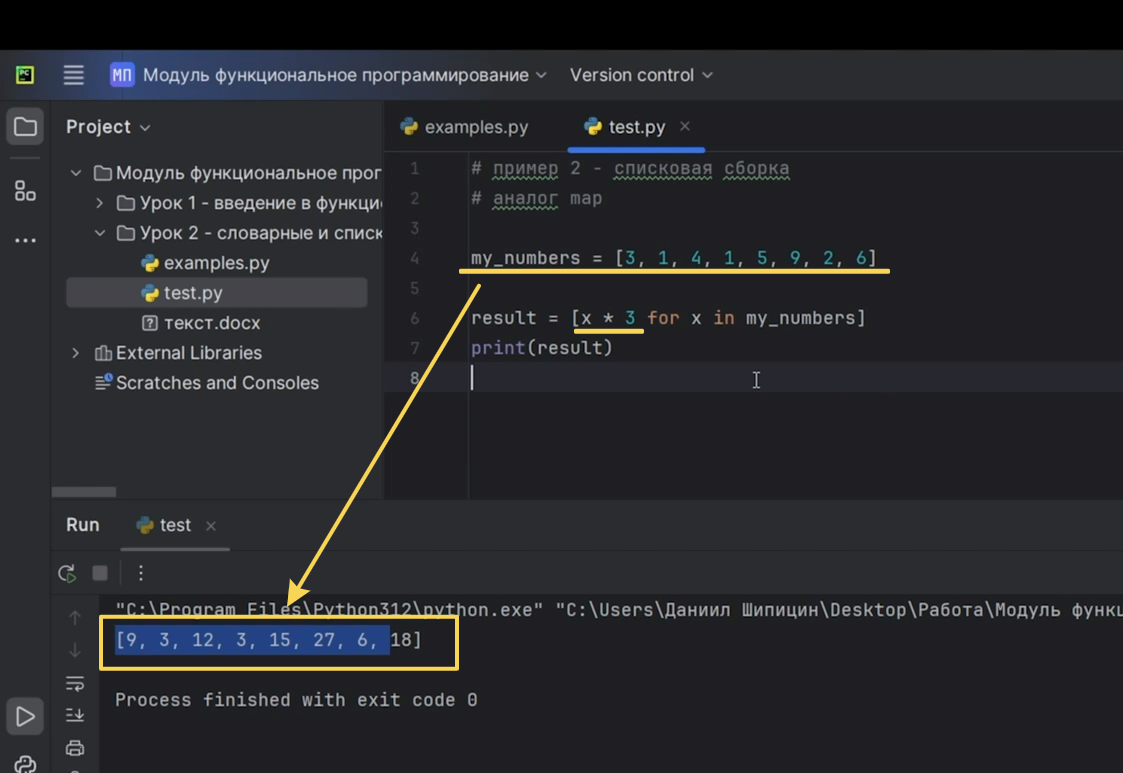


Рис.5

Помимо того, что мы можем проводить какие-то операции над элементами, мы можем их ещё точно также фильтровать. То есть это уже комбинация функции «map» и функции «filter». Куда у нас пишется условие в «list comprehension»? Мы пишем условия уже после нашего цикла «for». Сперва пишем элемент-операцию над элементом, потом пишем стандартную конструкцию цикла «for» и только в конце мы пишем одно условие «if x>5», или «if type x==чему-то», или «if x%5==0» (Рис.6). То есть условия могут быть совершенно разные. Иногда «list comprehension» растягивается до необъятных величин. Но зато это очень минималистично выглядит в плане количества строчек кода.

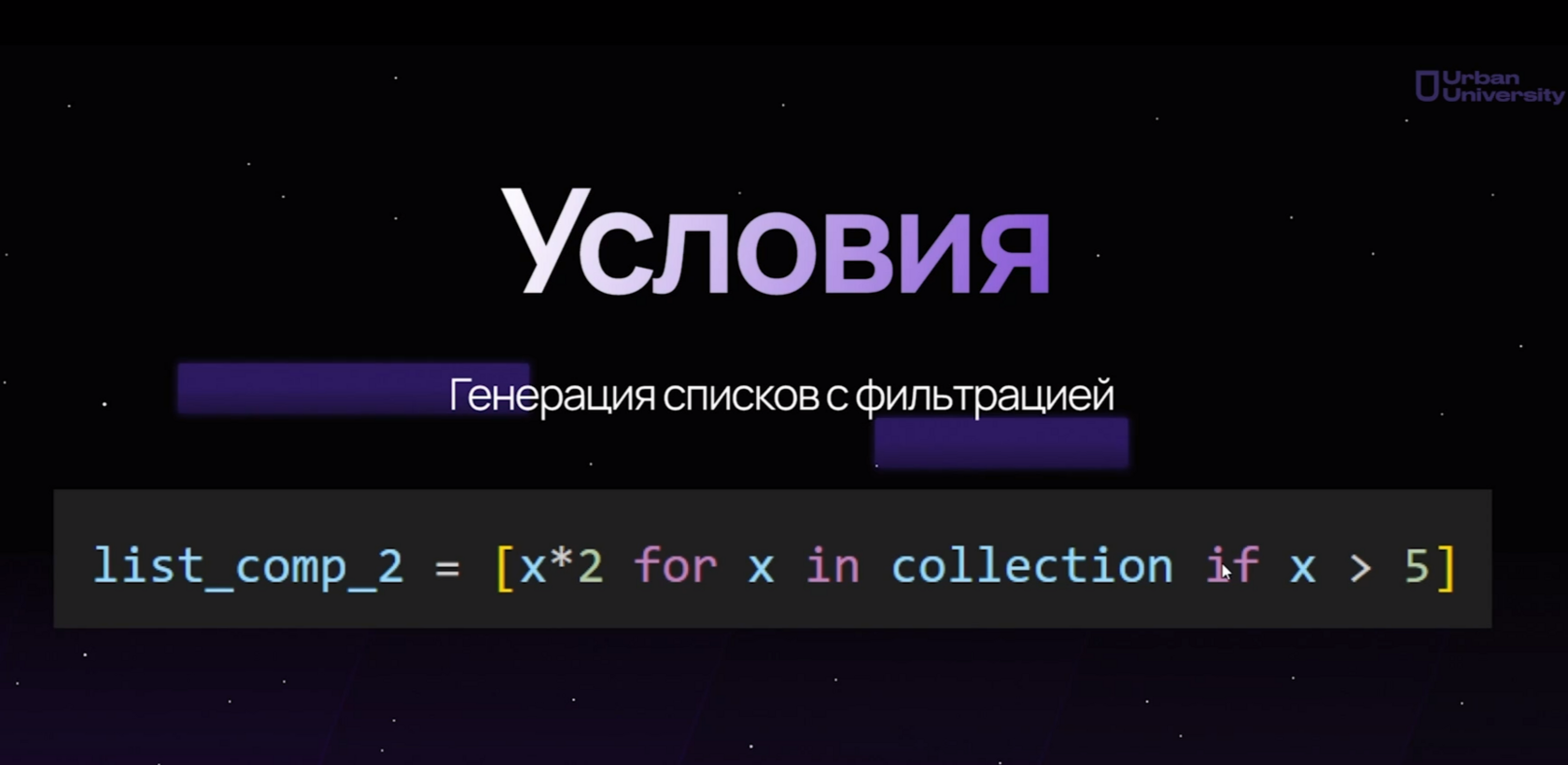


Рис.6

У нас может быть только один «if», то есть нет «elif», нет «else». С этим разберёмся попозже, просто поясняем вам сразу же заранее.

И теперь будет в наш список «list\_comp\_2» добавляться каждый элемент «x» из коллекции, только мы будем умножать его на 2, но если изначальный наш элемент из коллекции был больше 5. То есть мы пробегаемся циклом «for» по коллекции, проверяем, если «x» у нас больше 5, то мы умножаем его на 2 и добавляем в наш новый список. Сейчас на примере все это покажем.

Рассмотрим третий пример- списковая сборка с «if», аналог «filter». Теперь мы смогли объединить и «map», и «filter», которые мы вам показывали ещё в первом примере, но в более красивую конструкцию.

Нам точно также дан список чисел «my\_numbers». И теперь мы будем в результат добавлять каждый элемент из «my\_numbers» умноженный на 3, но только при том условии, если у нас остаток деления на 2 больше нуля, то есть число нечётное. То есть у нас добавятся только нечётные числа из списка «my\_numbers» в список «result» только нечётные числа, но при этом они ещё будут умножены на 3 (Рис.7).

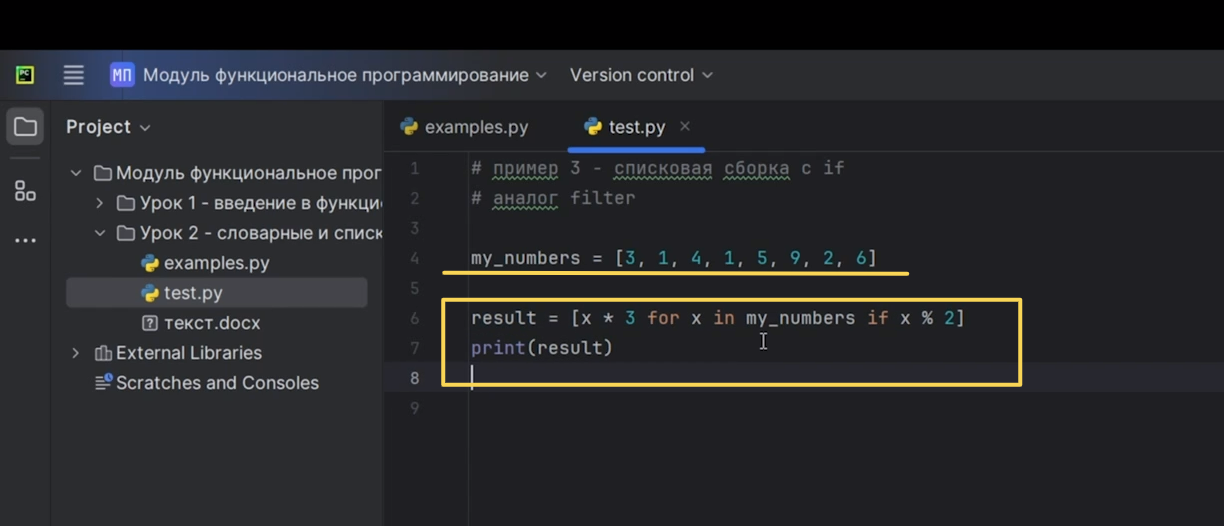


Рис.7

Получили результат «9, 3, 3, 15 и 27» (Рис.8).

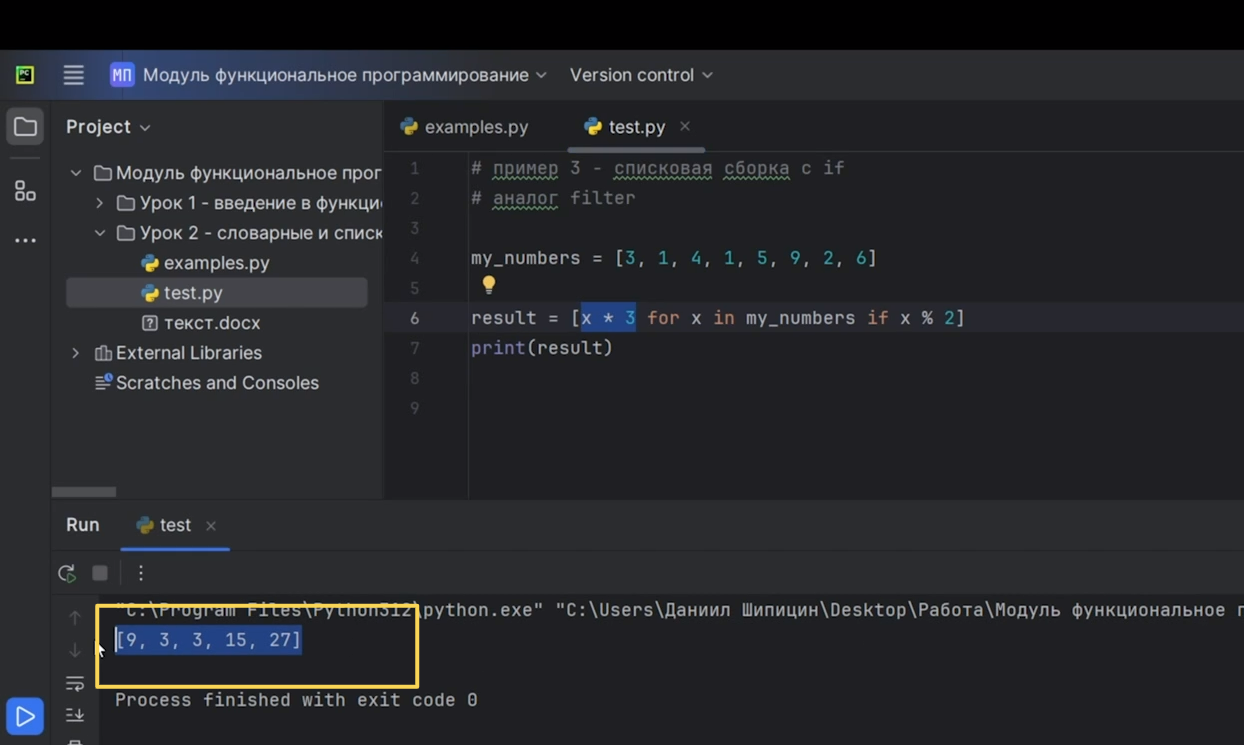


Рис.8

Но мы вам говорили, что можно добавить только 1 «if» и его нужно добавлять всегда после цикл «for». Но мы можем немножечко поменять наш порядок элементов. Теперь мы можем добавить «if» сразу же после того, как мы объявили какую операцию и какой элемент мы добавляем в «list comprehension» (Рис.9). Но зачем нам это надо?

Во-первых, мы теперь можем дописать «else».

Во-вторых, это у нас будет не фильтрация наших элементов в изначальном списке, то есть не в списке «collection». Это будет проверка условия, которое выполняется именно с этим элементом и выполнение разных операций над этим же элементом.

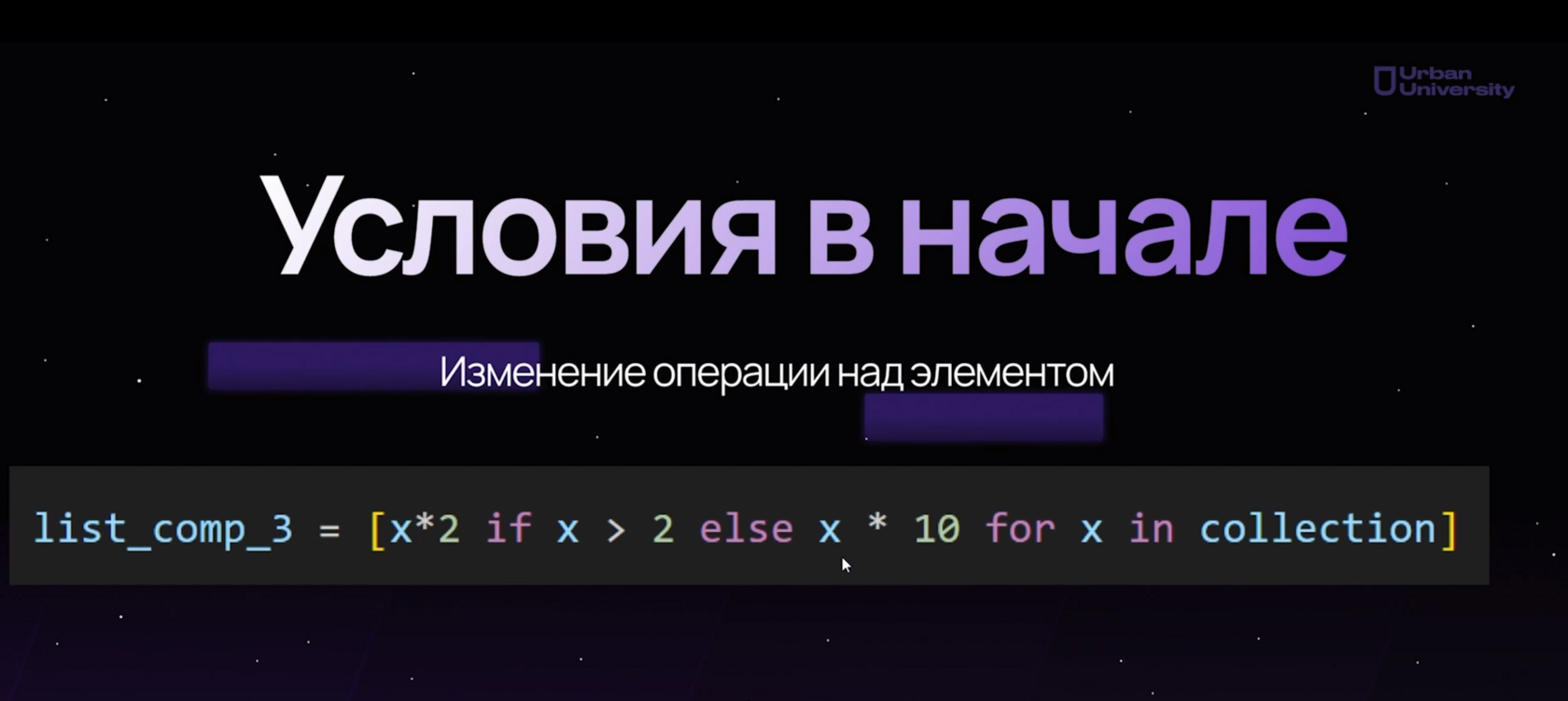


Рис.9

То есть, смотрите, мы будем добавлять «х\*2», только если «x>2». То есть мы будем 100% добавлять «х\*2» только если «x» больше 2. Иначе, у нас тут появляется ещё один блок «else», мы добавляем «х\*10». Как это работает? У нас теперь есть два пути:

1. Если у нас «x>2», тогда мы добавляем «х\*2»;

2. Если у нас «х<2», то тогда мы добавляем «х\*10».

То есть, если мы ставим конструкцию «if…else» в начале перед циклом «for» сразу же после элемента, тогда это уже будет изменение операции над элементом, а не его фильтрация.

Можно не добавлять блок «else» можно добавить только блок «if». Тогда это будет, по сути, та же самая запись, что если бы мы добавляли блок «if» после коллекции. В данном случае у нас фильтровались бы наши данные, добавлялись бы только «х\*2», если у нас «x>2». То есть мы могли бы этот «if» записать в конце, но если помимо того, что мы должны добавлять «х\*2», если у нас «x>2», нам ещё что-то нужно делать, то тогда мы дописываем блок «else», в котором, если не выполняется условие «if», будет выполняться блок «else». И тогда мы будем добавлять каждый элемент умноженный на 10. То есть мы добавим теперь все элементы списка, потому что у нас либо «x» может быть больше 2, либо «x» не больше 2, то есть у нас только два пути. Мы 100% все элементы добавим, но по разным путям: если у нас «x» больше 2 тогда добавляем «х\*2», если у нас «x» не больше 2, тогда добавляем «х\*10» «for x in collection».

На самом деле данная конструкция с «if…else» перед циклом «for», то есть такая уже длинная списковая сборка, это редкость. Это применяется больше для алгоритмов, то есть для решения алгоритмов. Если мы действительно хорошо разбираемся в этой теме, такую запись можно очень быстро напечатать, и мы сэкономим для себя кучу времени, нежели будем придумывать разные функции и использовать «map» и «filter». Но мы не говорим вам о том, что нельзя использовать «map» и «filter». Их можно использовать в других ситуациях, а мы говорим именно про генерацию списков.

Рассмотрим четвертый пример-условия перед циклом для того, чтобы не отфильтровывать данные наши в изначальной какой-то коллекции или в функции «range», а поменять операцию над этими данными. У нас дана коллекция «my\_numbers», но помимо того, что в ней есть числа, есть ещё и строчки (буквы). Но нам это не важно. У нас есть в данной коллекции числа, есть ещё тип данных «str» (Рис.10).

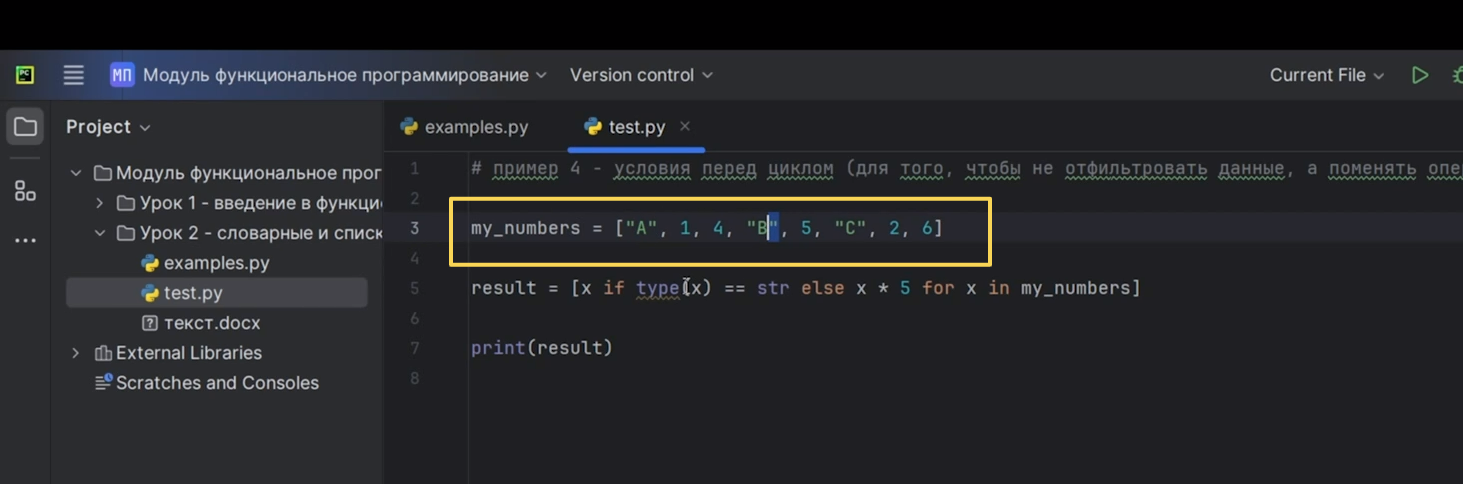


Рис.10

Мы хотим в переменную «result» добавить все наши буквы, то есть они должны сохранить свой изначальный вид, а также добавить все цифры, но они должны быть умножены на 5 (Рис.11). Мы могли бы пройтись циклом «for» и проверить каждый тип данных, но это была бы длинная большая запись, поэтому будем использовать списковые сборки. Мы будем добавлять просто «x», то есть мы будем просто добавлять наш элемент, если тип «х» равен строчке. То есть, если выполнится условие: если «type(x)==str», тогда мы просто добавим «x», то есть это идёт в совокупности. Иначе, мы добавляем «х» умноженный на 5. То есть, если у нас тип данных не строчка, тогда я просто добавляю наш элемент, но умножаем его на 5. Как мы можем заметить, это будут только числа. И потом, естественно, дописываем цикл «for», чтобы понять, откуда мы вообще будем брать наши данные «for x in my\_numbers».

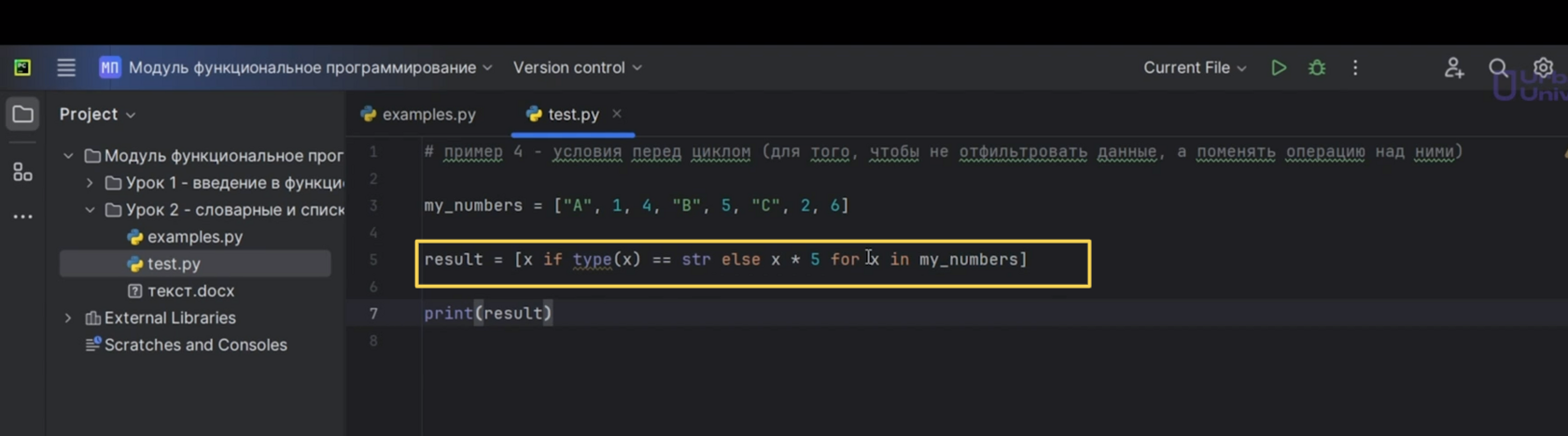


Рис.11

Выведем результат (Рис.12). Как вы видите, буквы сохранились в том же порядке, в котором они были изначально, но каждая каждое число у нас было умножено на 5.

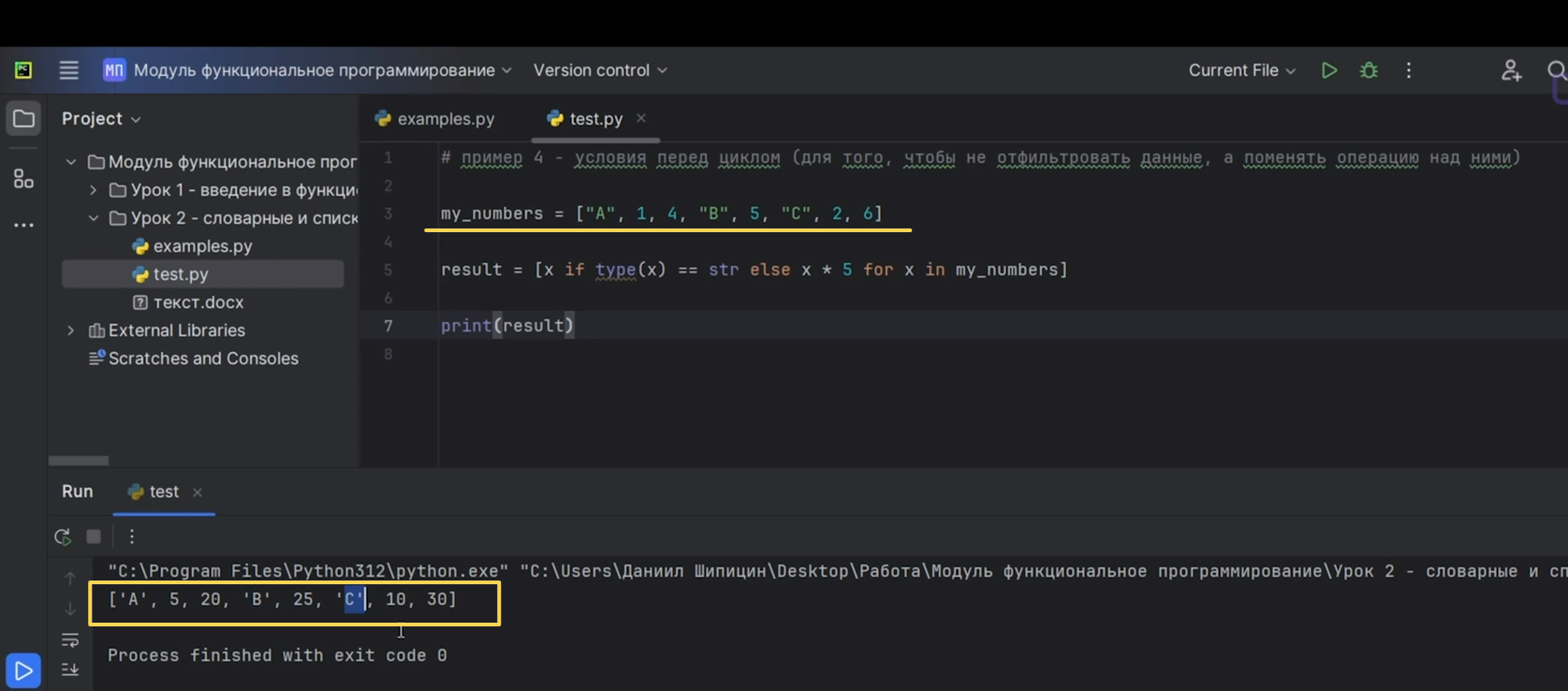


Рис.12

Мы можем использовать с вами вложенные списки для генерации для двух элементов. То есть мы теперь можем брать не только один список, не только один цикл «for x in collection». Помимо этого, можно брать другой список «for у in collection\_2» (Рис.13).

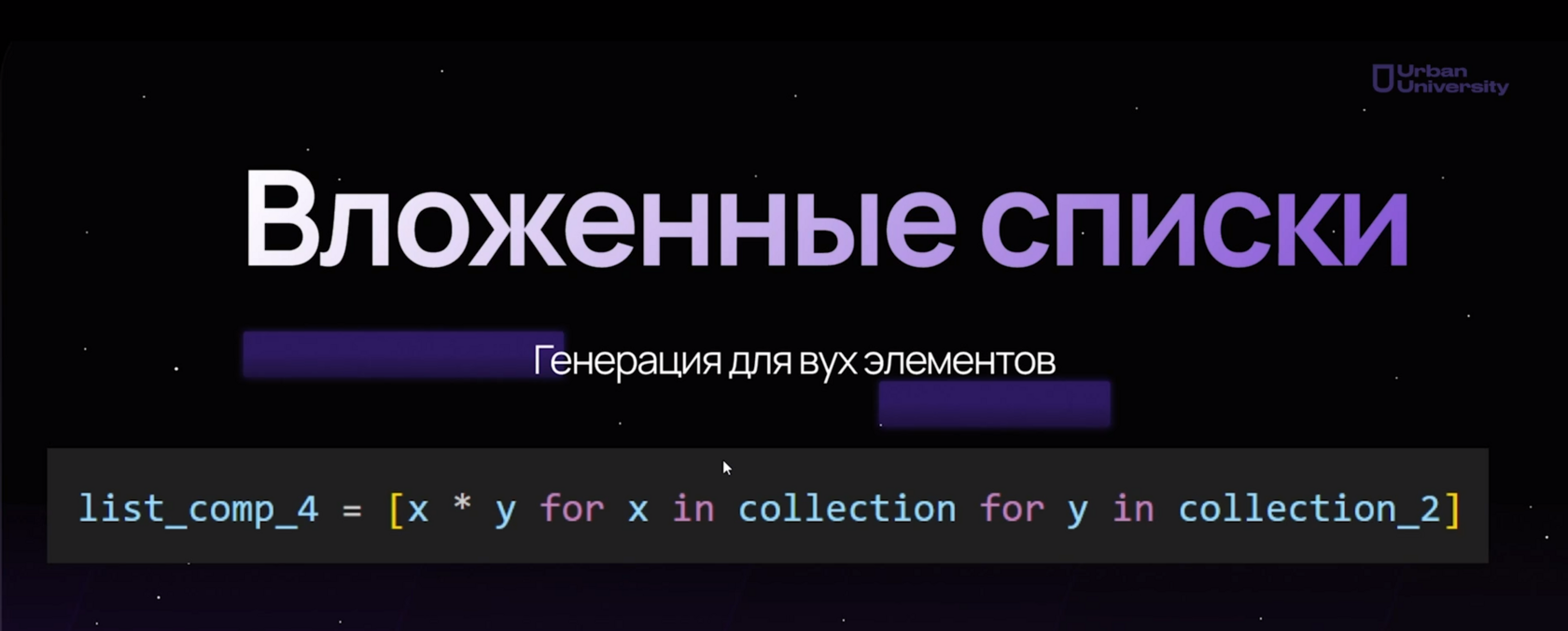


Рис.13

Теперь мы можем добавлять данные, мы их можем комбинировать, то есть у нас есть разные операции. Мы можем добавлять, например, только «x\*y». Мы можем добавлять «х» и «у», которые каждый отдельно умножается на какие-то разные числа. И теперь у нас будет продвигаться «х» по каждому элементу, и каждый элемент «x» у нас будет умножаться на каждый элемент «у». Давайте посмотрим на примере №5.

У нас есть список «my\_numbers» и есть список «theу\_numbers»- это два разных списка (Рис.14).

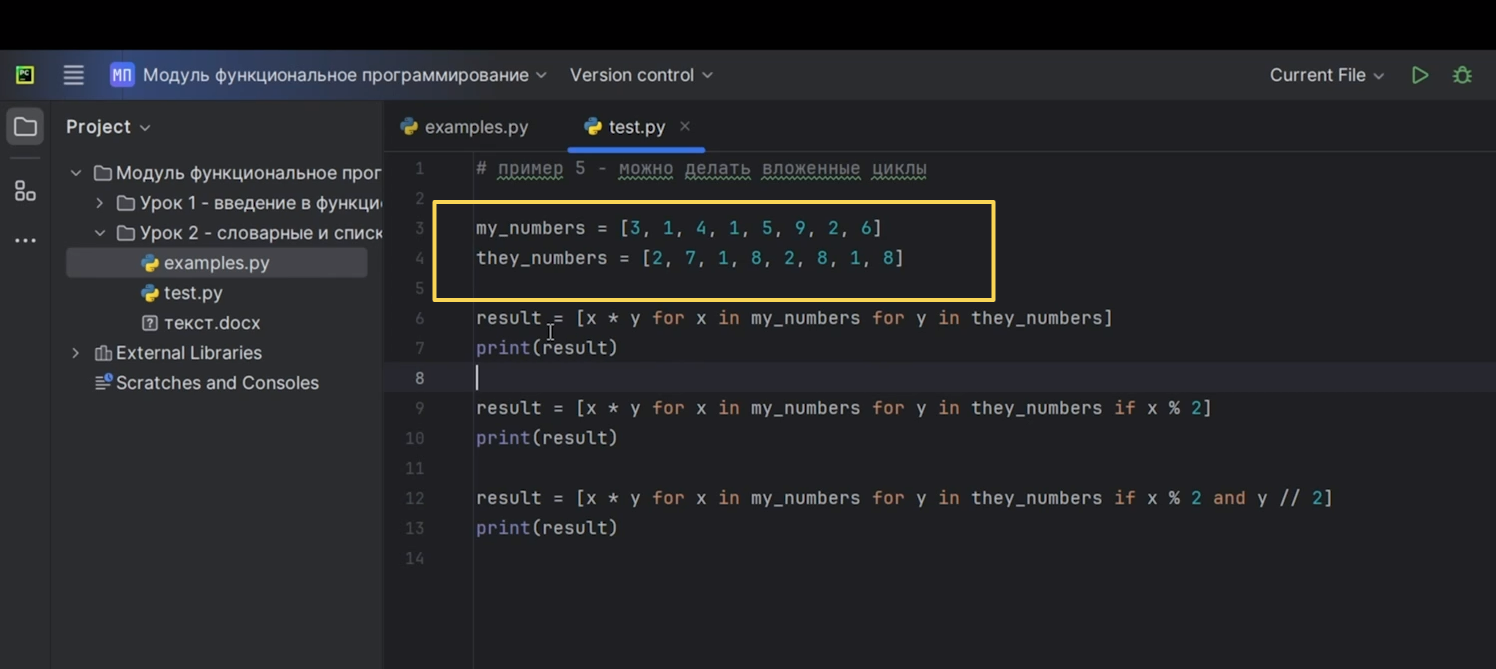


Рис.14

Первая наша списковая сборка это мы «x\*у» для каждого из «x» и для каждого из «у», точнее для каждого «x» из «my\_numbers» и для каждого «у» из «theу\_numbers» (Рис.15).

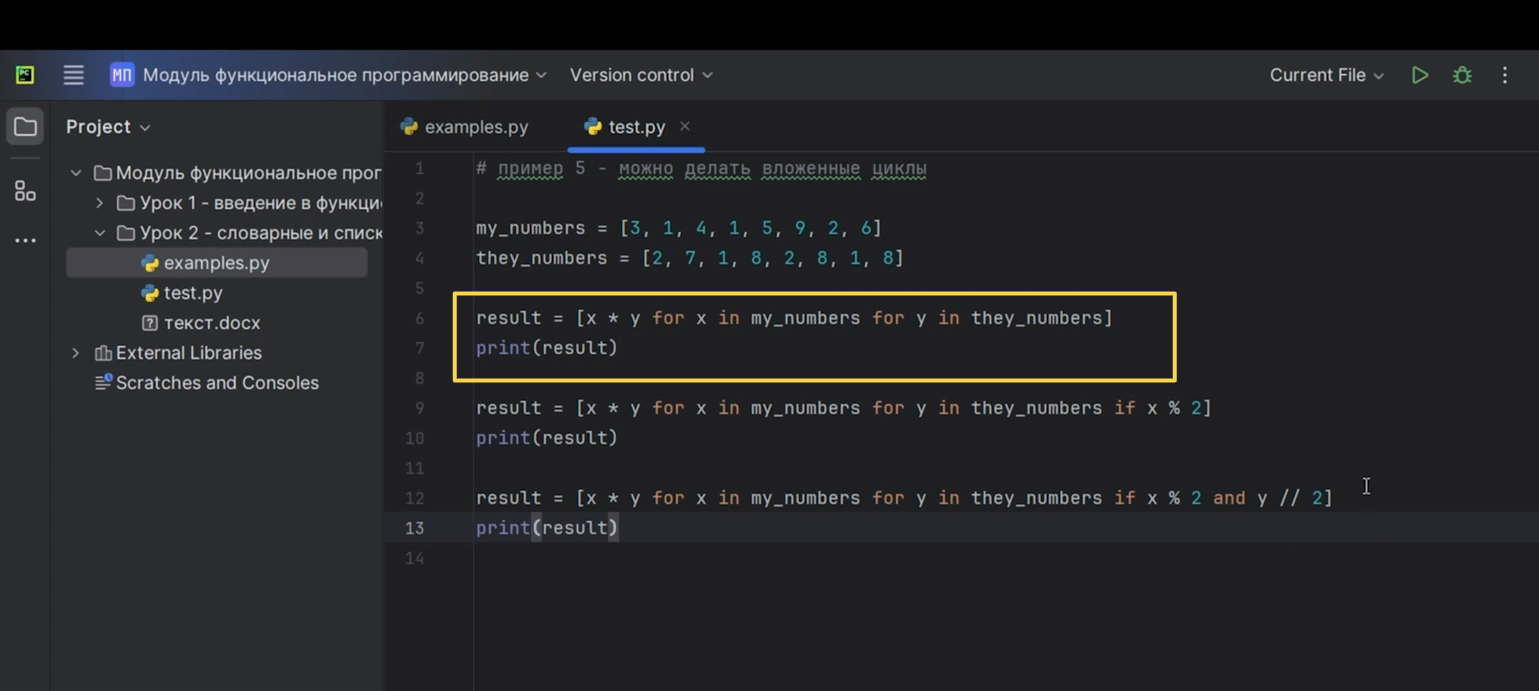


Рис.15

Дальше мы ещё хотим добавить условия. Мы будем добавлять «х\*у» для каждого «x in my\_numbers», для каждого «у in theу\_numbers», только если у нас остаток деления на 2 будет больше нуля (Рис.16).

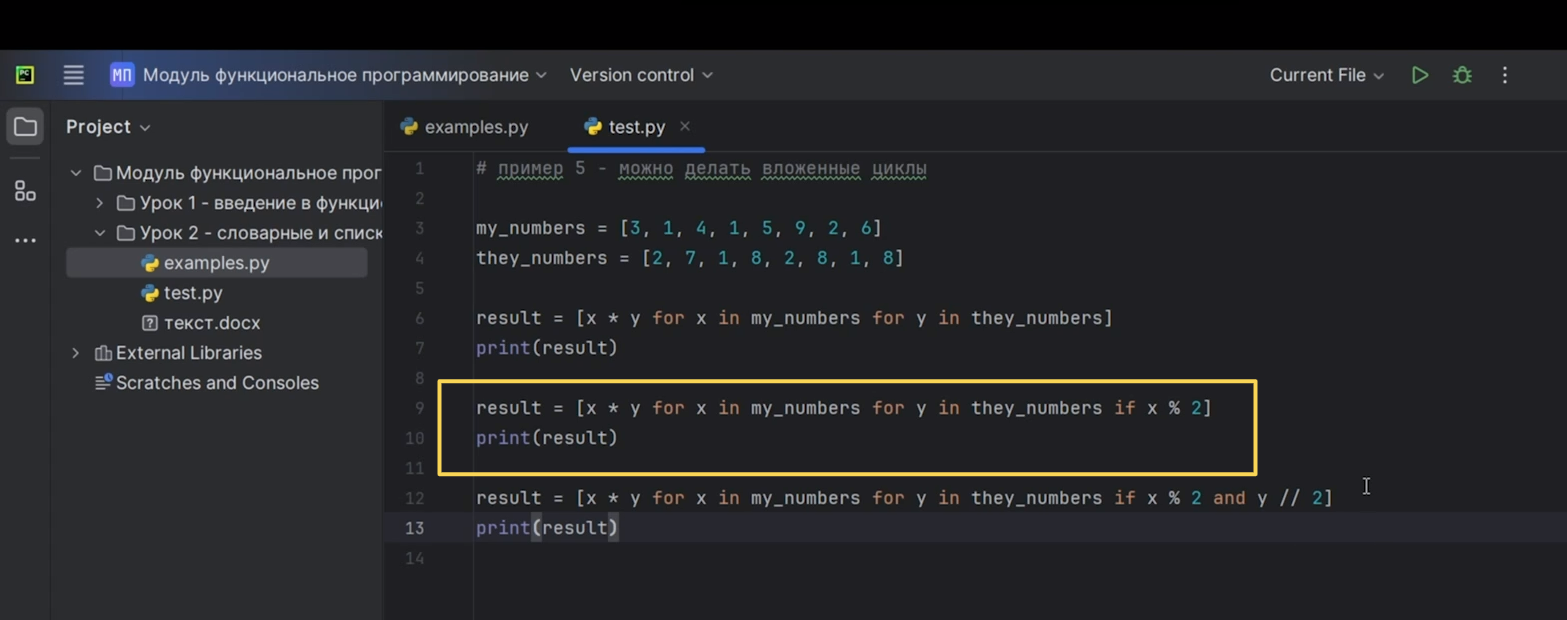


Рис.16

Потом мы ещё добавляем дополнительное условие. Конечно же можно в одном «if» записывать большое условие, то есть не генерировать несколько «if», а просто в одной цепочке проверять с помощью логических операторов «and», «or» и «not» точно также. Поэтому мы и говорим, что списковые сборки могут быть действительно очень большими по своей длине.

Теперь мы будем добавлять каждый «х\*у» для каждого «x in my\_numbers», для каждого «у in theу\_numbers», если у нас «x%2» будет больше нуля, а также «у» целочисленно делится на 2 (Рис.17).

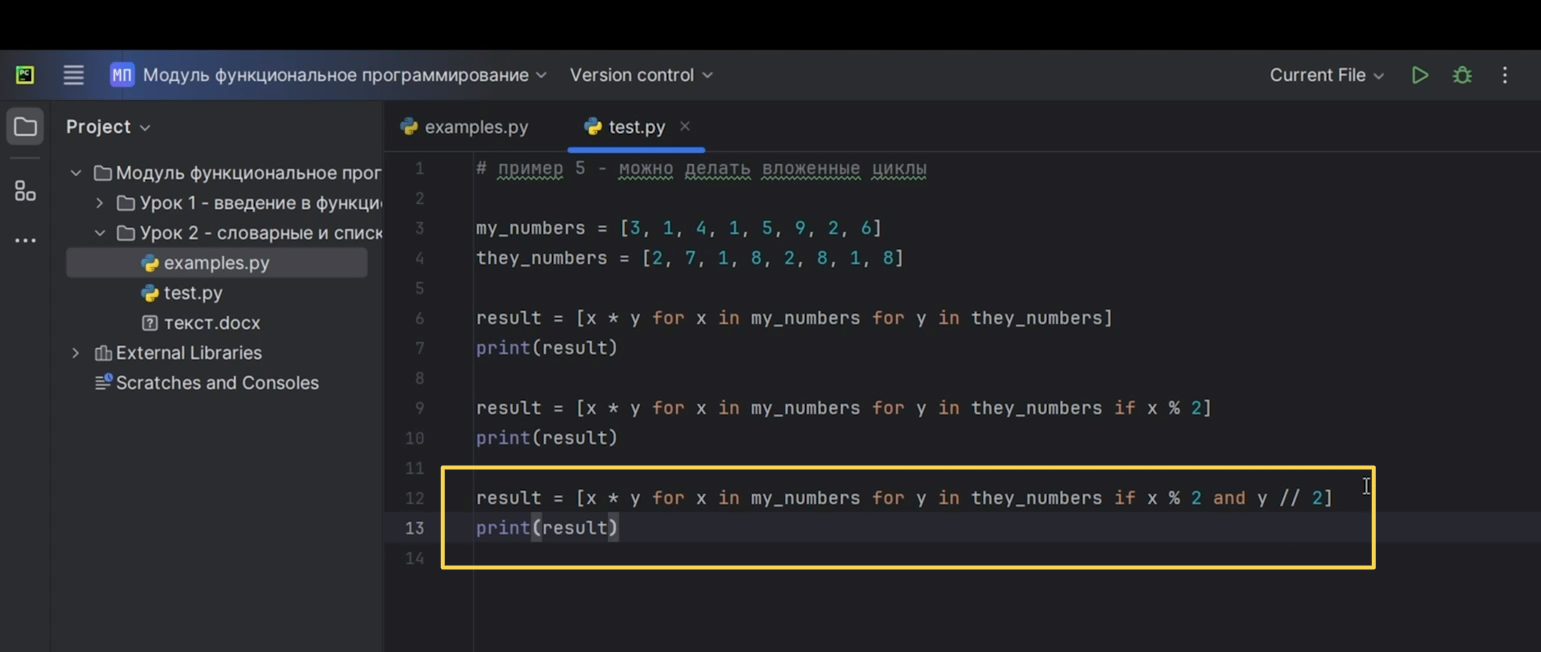


Рис.17

Выведем результат (Рис.18). У нас выходят три списка. Первый из них большой, потому что мы берём и каждый элемент из «x» умножаем на каждый элемент из «у». Второй у нас поменьше, потому что мы уже фильтруем некоторые данные. А третий ещё меньше, потому что у нас в добавок к тому что мы фильтруем наш «х», мы ещё будем фильтровать наш «у».

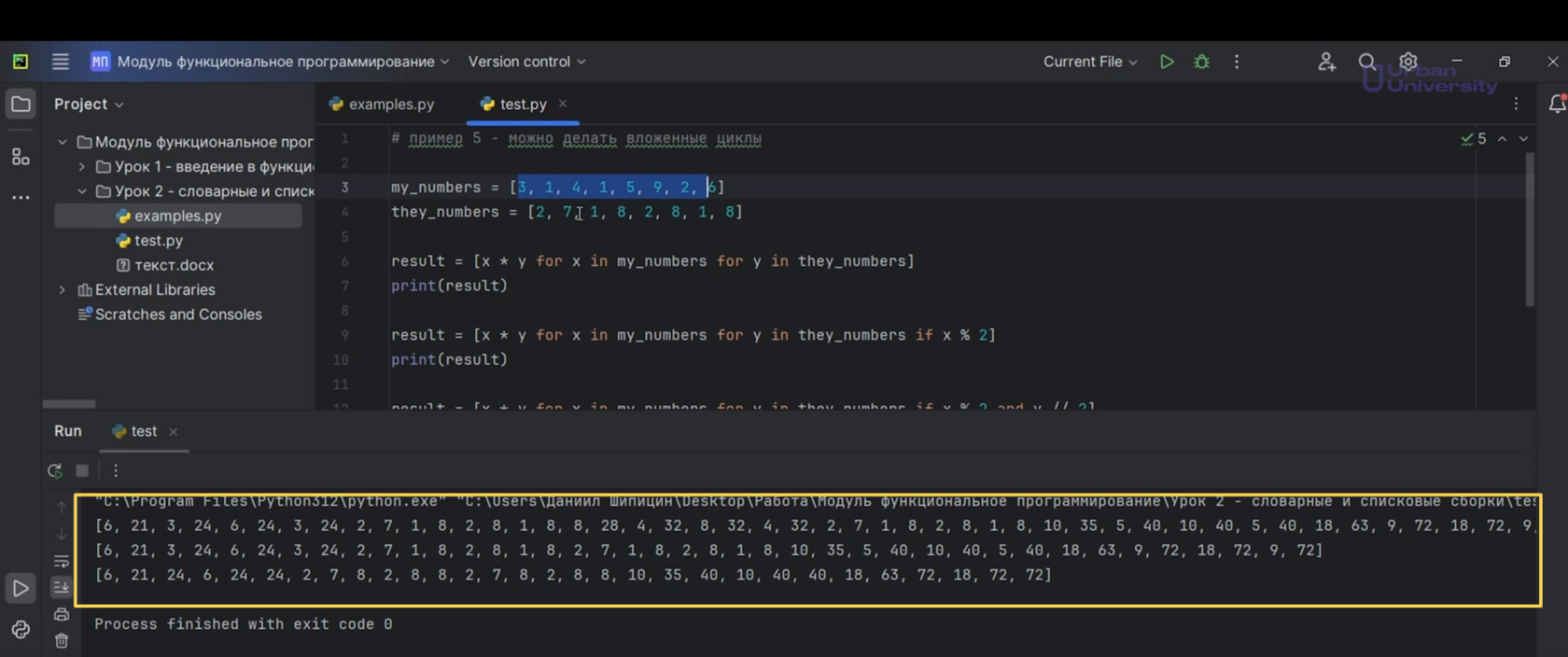


Рис.18

Такой вот мощный инструмент для вычислений списковые сборки.

Но на этом мы ещё с вами не останавливаемся, потому что сейчас будем с вами знакомиться с генераторами. Что это такое? И почему они вообще входят в списковые сборки? Но и не только списковые сборки, они могут использоваться отдельно. Поэтому давайте дальше изучать.

Перед изучением генераторов, хотели вам показать на примере, что точно так же можно генерировать множество на лету и словари. То есть, если мы хотим сгенерировать множество, мы просто пишем фигурные наши скобочки (Рис.19). Напоминаю, множество-это упорядоченная коллекция данных, в котором все данные, все наши элементы уникальны, поэтому у нас будет добавляться каждый «x». Но они, во-первых, добавятся у нас потом упорядочено, то есть по возрастанию. А во-вторых, все не уникальные элементы, они удалятся.

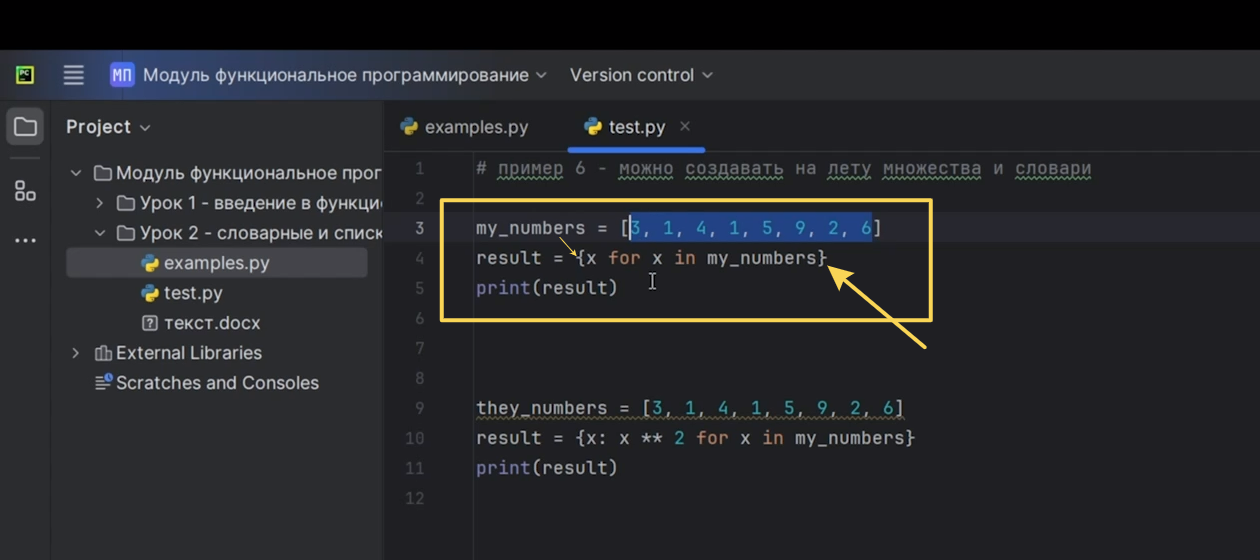


Рис.19

Точно также мы можем генерировать с вами словари. Напоминаем, это тоже у нас фигурные скобочки, просто мы пишем конструкцию, как у словарей. То есть мы пишем ключ, через двоеточие мы пишем значение (Рис.20). Естественно, можно использовать вложенные списки, например, ключом у нас будет «x», а значение будет «у». Мы можем прогнать их по циклам, можем накинуть какие-то условия и так далее.



Рис.20

Выведем результат (Рис.21). На первой строчке у нас вывелось наше множество. Оно упорядочено, каждый элемент уникален. А на второй строчке у нас вывелся наш вот этот список чисел «mу\_numbers». Только теперь у нас тут пары «ключ:значение». У нас идет ключ 3 и потом его квадрат. То есть у нас тут каждый ключ-это число, каждое значение у этого ключа-это квадрат этого числа.

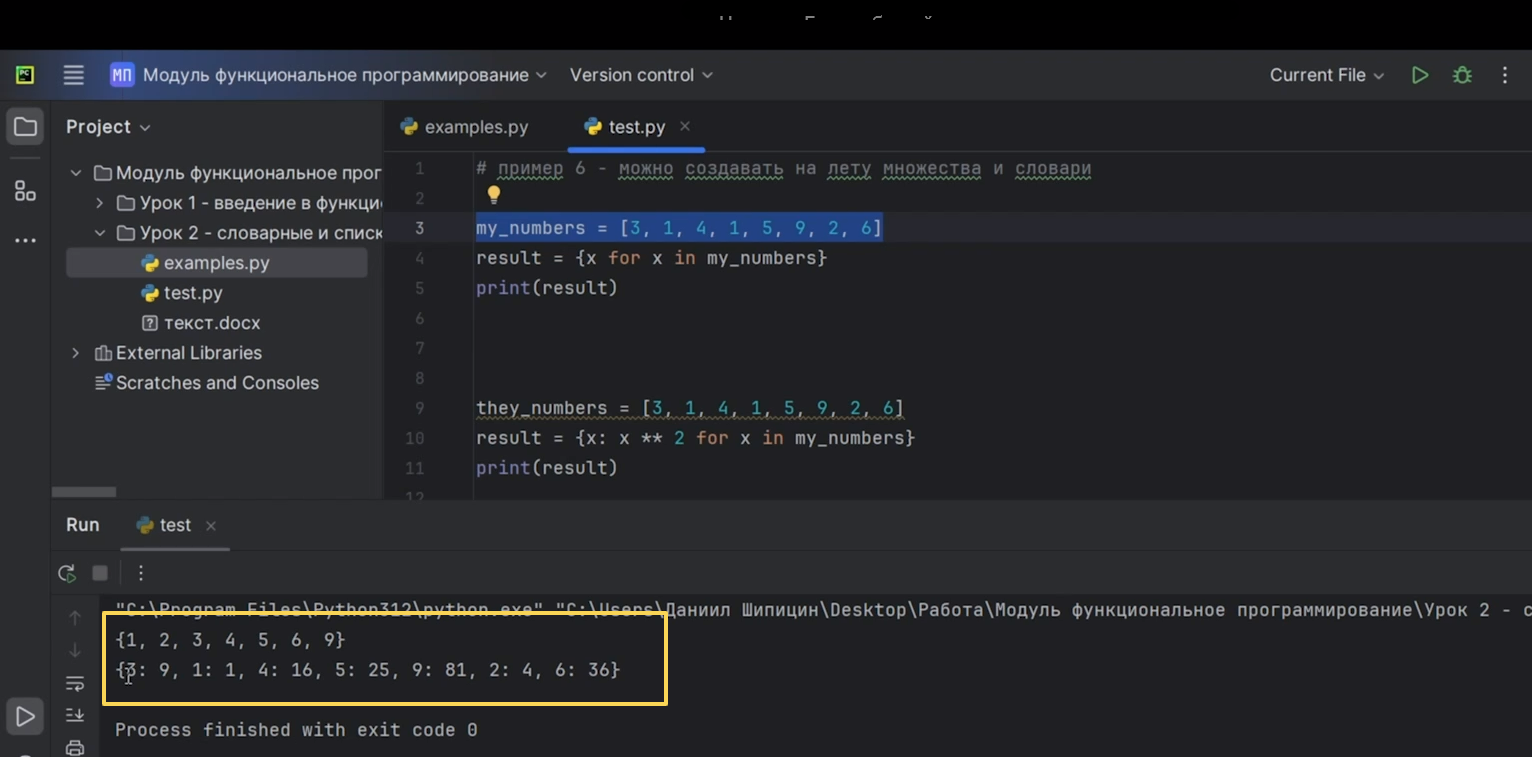


Рис.21

Словарная сборка, сборка словарей, такая генерация действительно могут пригодиться.