**2023/11/30 00:00|Лекция. Генераторные сборки**

## Генераторные сборки

Наш сегодняшний урок посвящён генераторным сборкам. Он отчасти связан с прошлым уроком, посвящённым списковым сборкам, потому что точно также генерирует какие-то значения для коллекции данных, но генераторная сборка, она работает совершенно по другому. Конкретно на этом уроке мы не будем говорить полностью о генераторах, лишь частично их затронем, а поподробнее мы эту тему ещё раскроем. Сейчас мы вам расскажем некоторые базовые моменты о генераторах. Но для этого нам нужно понять, что такое ленивые вычисления. Потому что именно генераторные сборки используются для ленивых вычислений. **Ленивые вычисления**-это те вычисления, которые выполняются только тогда, когда они нам необходимы. То есть, если мы создадим какой-нибудь список «а», который хранит в себе списковую сборку, где каждый элемент у нас будет возводиться в сотую степень в каком-то списке (Рис.1). И у нас сразу же в памяти будет храниться этот список «а», который хранит в себе много, много чисел, возведённых в сотую степень. Это будет огромный пласт памяти. Но, конечно, не такой большой. Однако, если мы говорим о действительно серьёзных программах, это будет очень ресурсозатратно.

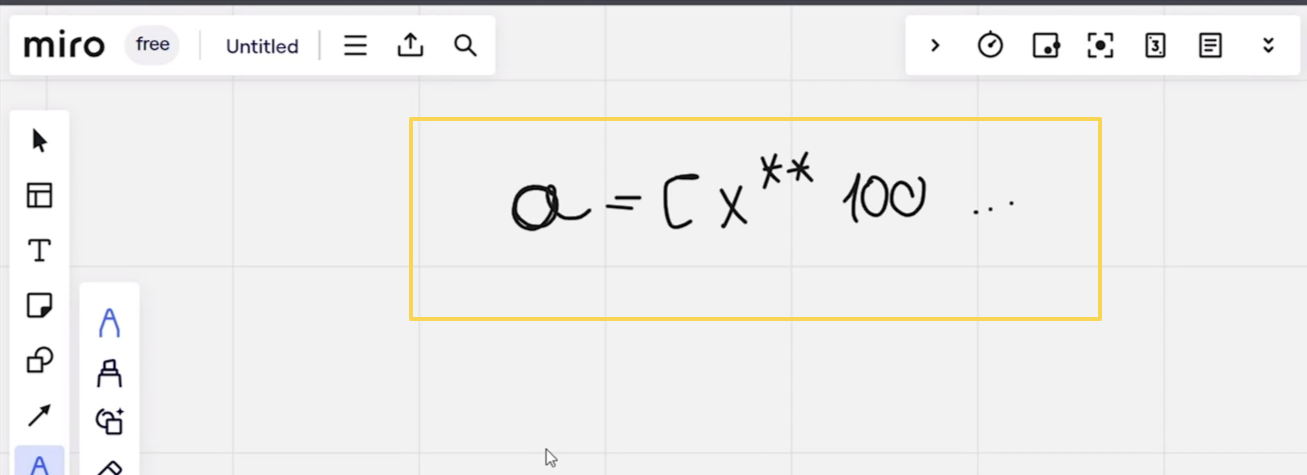


Рис.1

Ленивые вычисления-это те, которые производятся только тогда, когда нам надо. Мы сделали этот список «а», он хранится у нас в памяти. Если мы хотим пройтись циклом «for» по этому «а» и напечатать каждый элемент, то мы просто будем пробегаться по этому списку «а». У нас есть уже каждый вычисленный элемент, который всегда хранится в списке «а», поэтому мы его просто выводим (Рис.2).

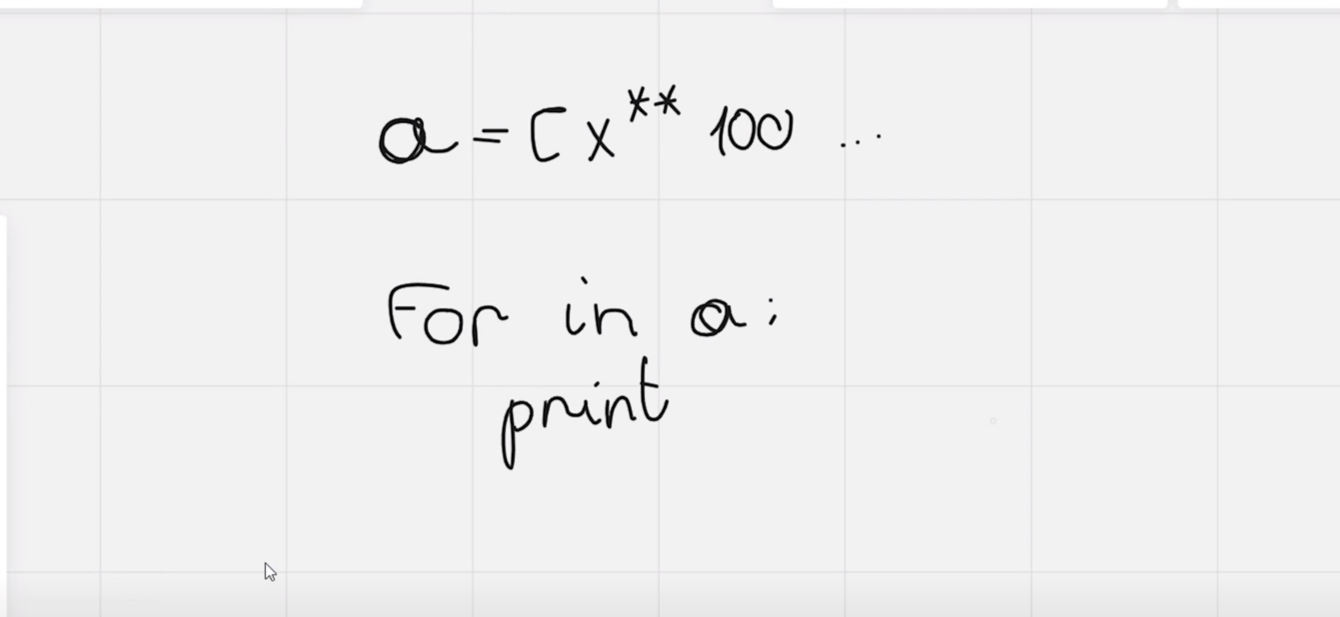


Рис.2

Ленивые вычисления работают по другому. Генераторные сборки всегда записываются в круглых скобочках как картежи. Поэтому у нас есть только списковые сборки, словарные сборки, множественные сборки, но картёжных сборок нет, потому что в круглых скобочках записывается у нас именно генераторные сборки. Мы создадим похожий список «а» и запишем допустим, выражение «x» в сотой степени в таком-то диапазоне (Рис.3).

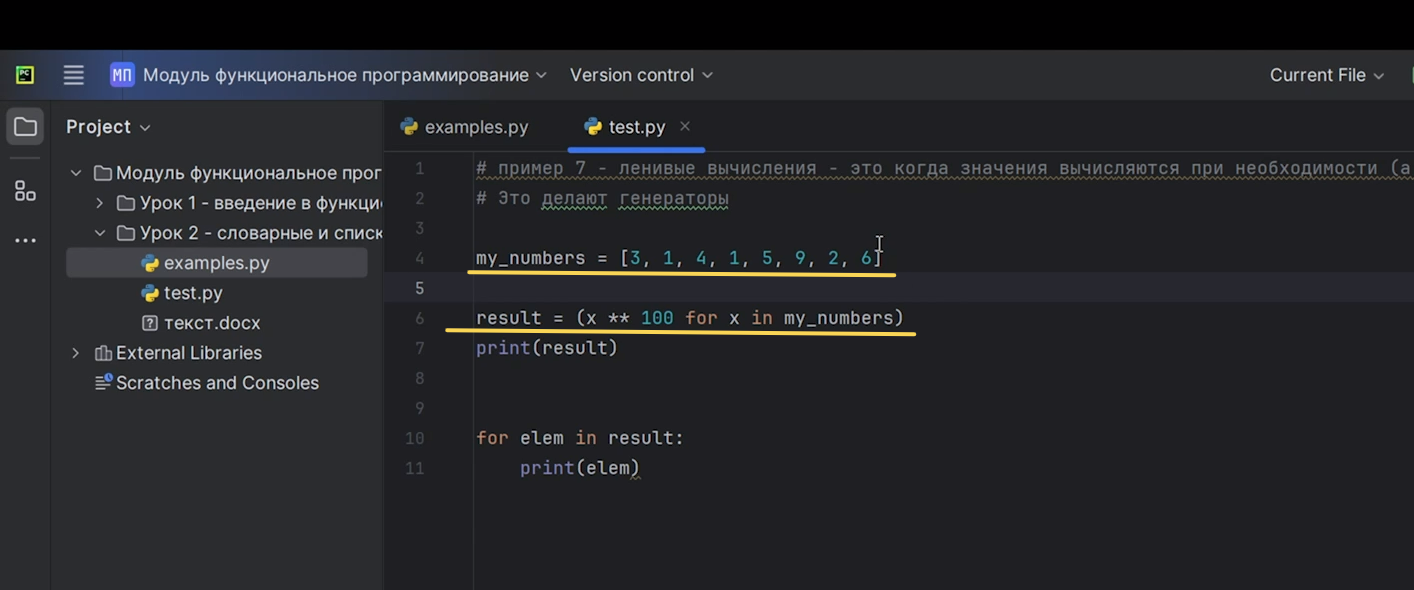


Рис.3

«А» хранит в себе только данную сборку и все. То есть сейчас в переменной, «а» не хранится огромный объём данных, который мы раньше создавали в списке. Они не хранятся там, потому что даже не вычислены. Они будут вычисляться только тогда, когда нам будут нужны. Например, когда мы пройдёмся по ним циклом «for». Когда мы пройдёмся циклом «for» по «а», тогда у нас выведутся все вот эти вот вычисления по порядку. То есть сперва нам дастся первый элемент, мы его выведем, и генератор навсегда забудет про первый элемент. Он его вывел или куда-нибудь сохранил и все, дальше он берет следующий элемент. Это уже идут свойства генератора. Сейчас мы вам о них тоже расскажем.

**Особенности генераторных сборок:**

1. они выдают результат только тогда, когда надо, то есть вычисляются при необходимости;

2. могут выполниться только один раз. Генераторы могут выполниться только один раз, то есть могут провести свои вычисления только один раз;

3. они занимают мало места в памяти. Так как они вычисляют только тогда, когда нам необходимо и при этом забывают прошлый вычисленный результат, то они в памяти хранят очень мало места. При этом ещё и выполняется гораздо эффективное вычисление. Все это рассмотрим на примерах.

Первый пример на этом уроке-ленивые вычисления. У нас есть список «my\_numbers» и в переменную «result» мы сохраняем нашу генераторную сборку: открываем круглые скобки и пишем в них «x\*\*100 for x in my\_numbers» (Рис.4).

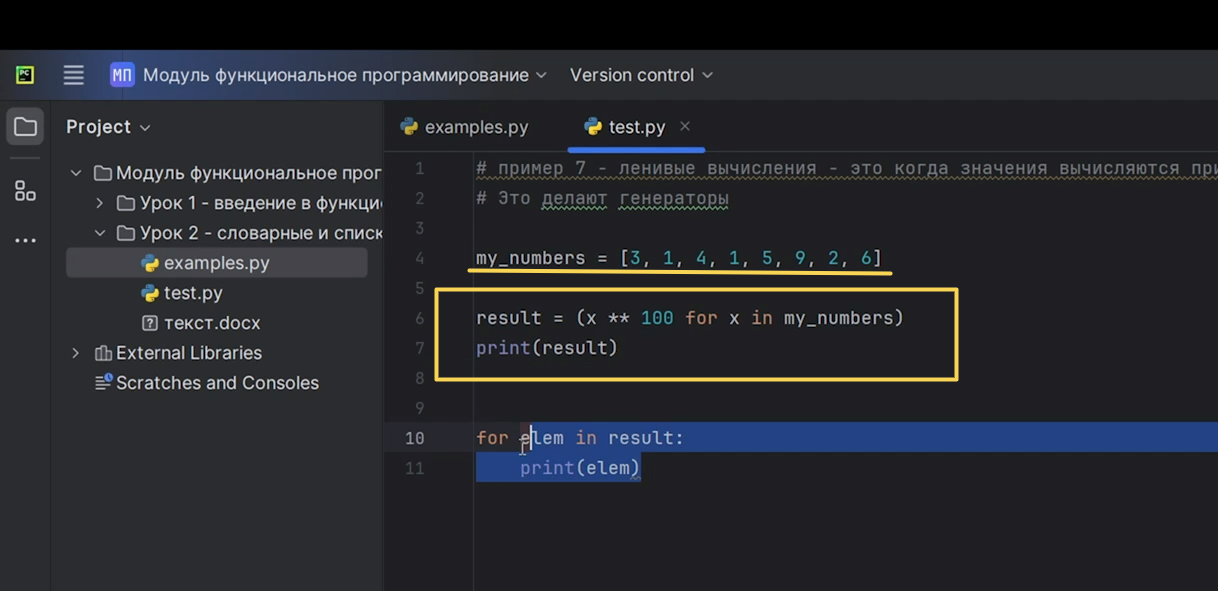


Рис.4

Понятное дело, сейчас вообще без разницы хранить это в списке или хранить это все в генераторной сборке, потому что чисел очень мало, данных очень мало и возводим их всего лишь в сотую степень. Это не так уж и много. Просто хотим вам показать, что если мы вдруг сейчас захотим просто вывести наш «result», то есть результат нашей функции генераторной сборки, просто выведется «generator object» (Рис.5). Потому что в данный момент написанные вычисления даже не производились.

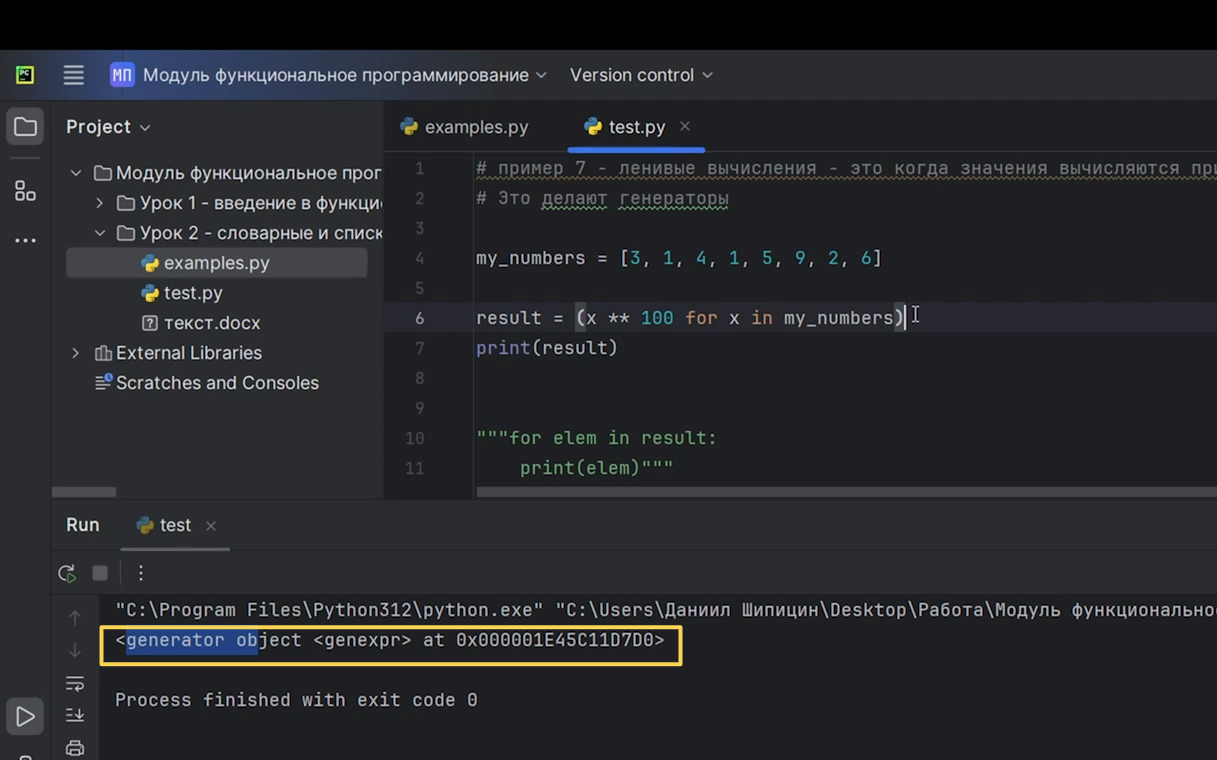


Рис.5

Только тогда, когда мы, например, введем элементы в цикл «for» и решим вывести каждый элемент «in result», выведутся все наши элементы (Рис.6).

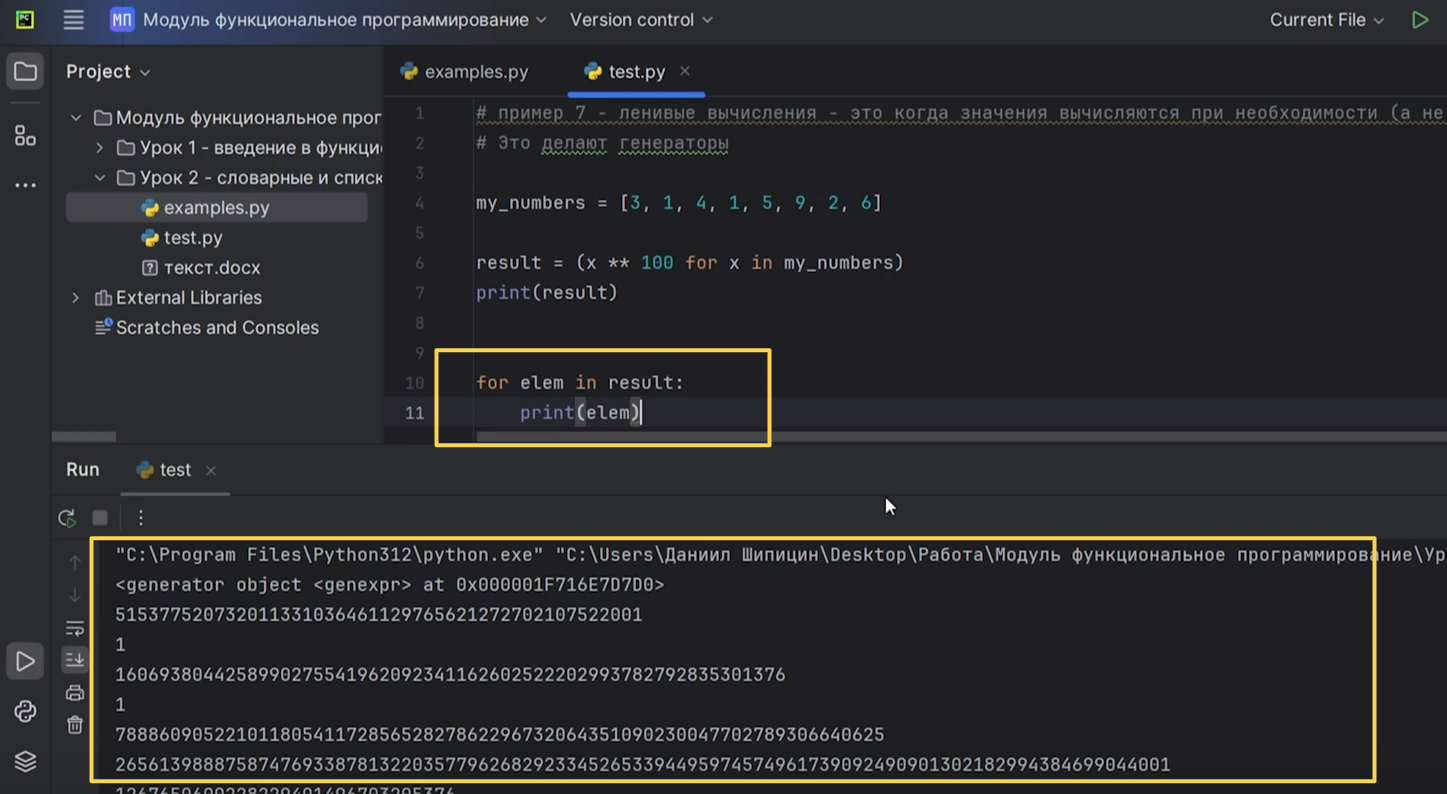


Рис.6

Теперь покажем то, что генераторную сборку можно использовать только один раз. Смотрите, снова у нас есть список чисел «my\_numbers». В переменную «result» сохраняем ту же самую генераторную сборку. Потом пытаемся первый раз пройти циклом «for» и выводим элементы. Потом ещё раз пытаемся по этой же генераторной сборке пройтись и вывести эти элементы (Рис.7).

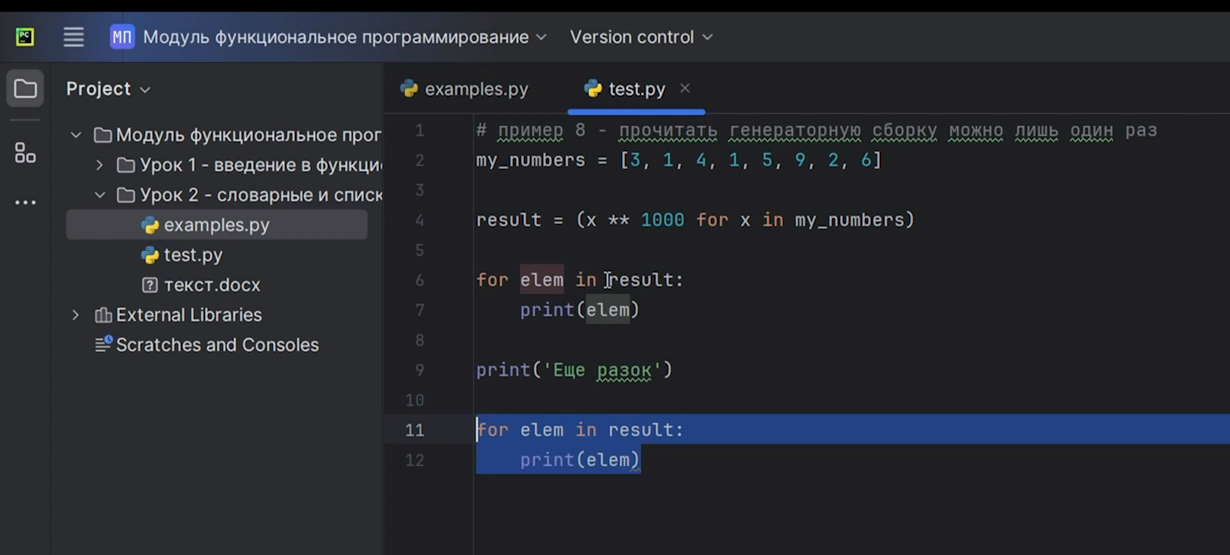


Рис.7

Но ничего не получилось, потому что у нас вывелись все элементы, возведённые в тысячную степень, далее написано «Ещё разок» и все (Рис.8). Результат второй части кода не напечатался, он не выполнилась, потому что генераторная сборка выполняется только один раз.

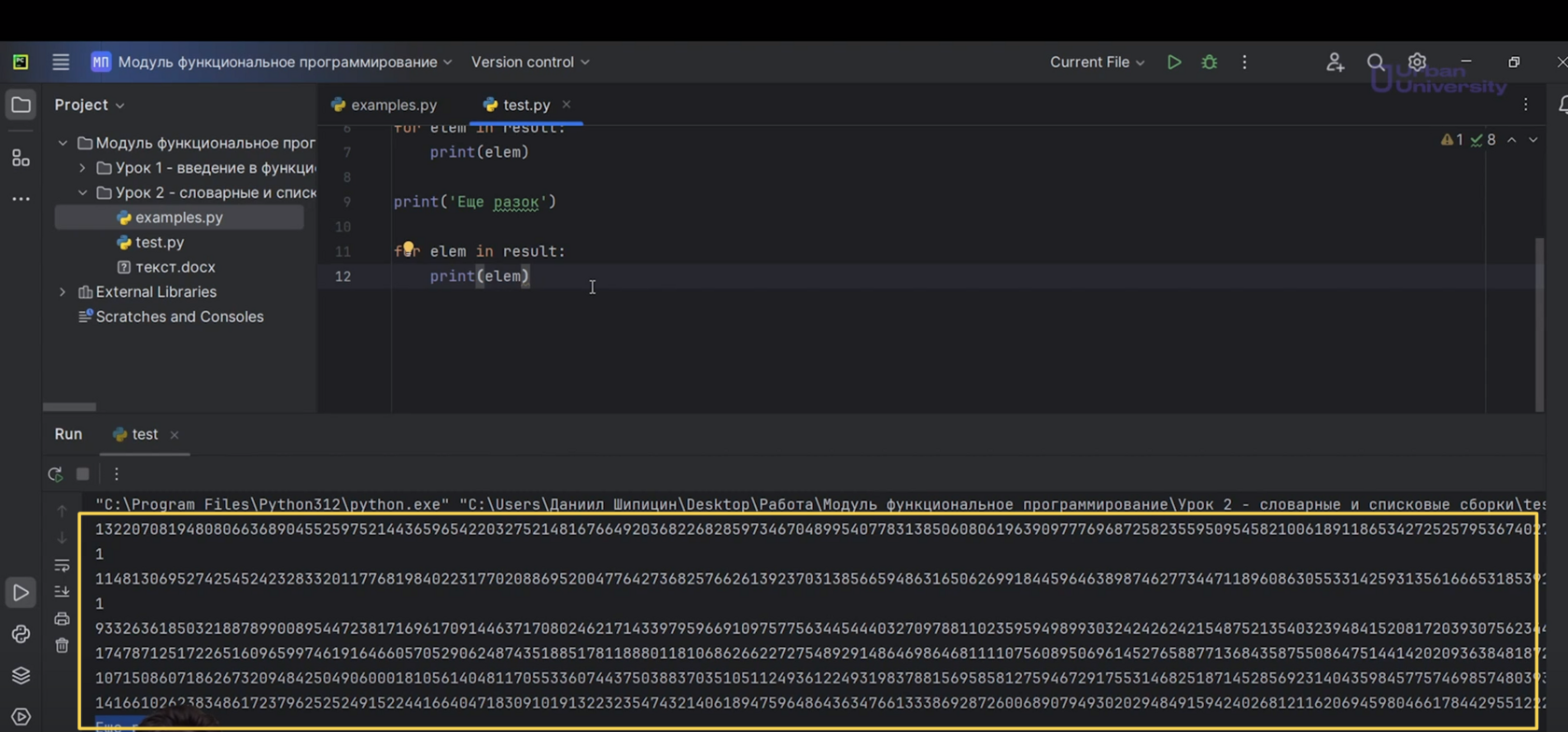


Рис.8

Она очень мало места в памяти сохраняет и выполняется только тогда, когда это нам необходимо, то есть такой очень лёгкий алгоритм. Это и есть наши ленивые вычисления. То есть они выполняются один раз и только тогда, когда нам необходимо. Это, знаете, как ленивый студент, который делает свою домашнюю работу прямо в день её сдачи.

Но это действительно полезно. Сейчас мы вам покажем почему на последнем завершающем на сегодня примере. Генератор используется там, где надо производить очень затратные операции. Мы в этом примере ещё дополнительно импортировали модуль «time» и зафиксировали время начала и конца нашей программы. И потом в конце вывели ответ: разницу времени конца и начала, их разницу умножить на 1000. То есть ответ будет в миллисекундах (Рис.9).



Рис.9

В списке «my\_numbers» всего лишь 8 чисел. Это достаточно маленький объем данных. Каждое из этих чисел возведём в трехтысячную степень. Это действительно очень маленькие операции для компьютера. И все это будем сохранять в списковую сборку и список. То есть в переменной «result» будут храниться все числа. И потом, мы пытаемся, во-первых, вывести этот «result» и у нас выведутся все эти числа. А во-вторых, попытаемся ещё и циклом «for» пройтись и вывести каждый элемент. И только потом у нас напечатается время, затраченное на эту всю программу (Рис.10).

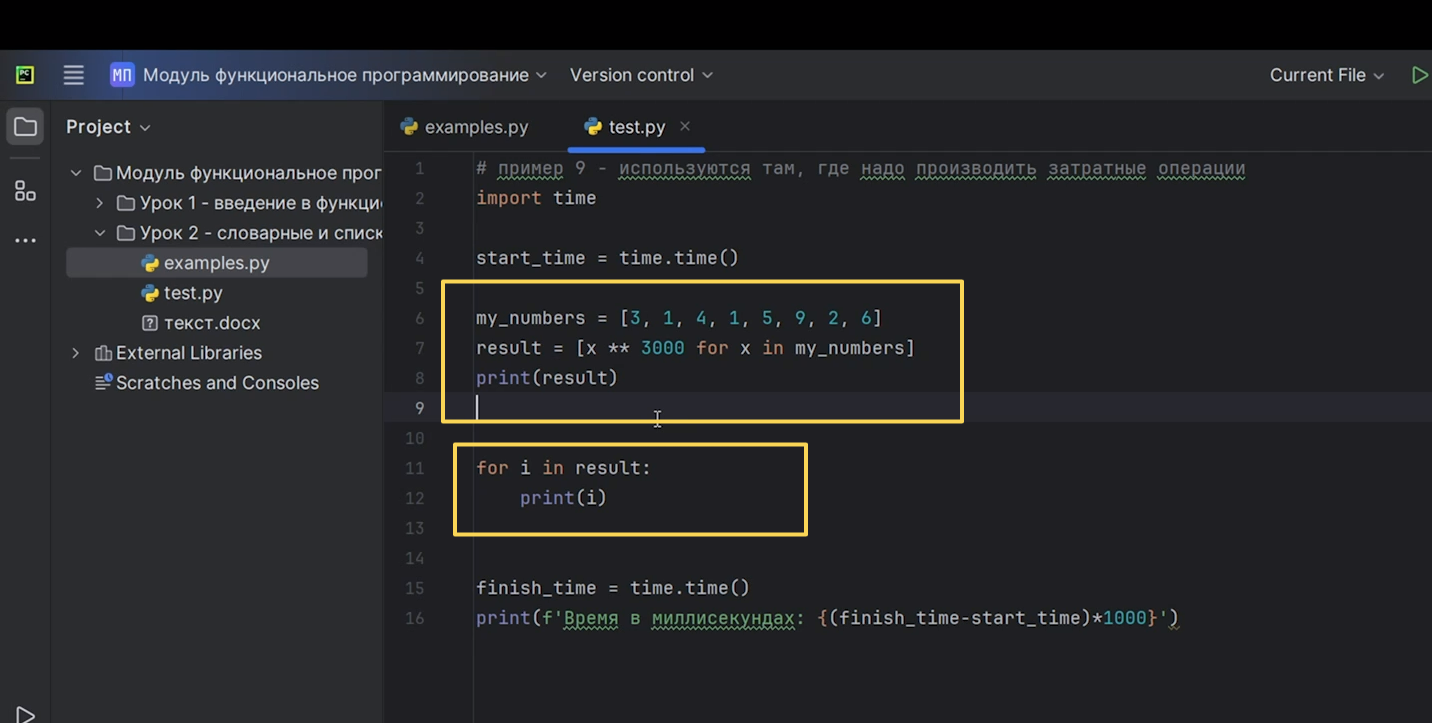


Рис.10

Выведем результат (Рис.11). У нас вывелся весь список, потом отдельно точно также элементы из этого списка. И в конце вывелось время, затраченное на программу около 9 миллисекунд. Достаточно быстро.

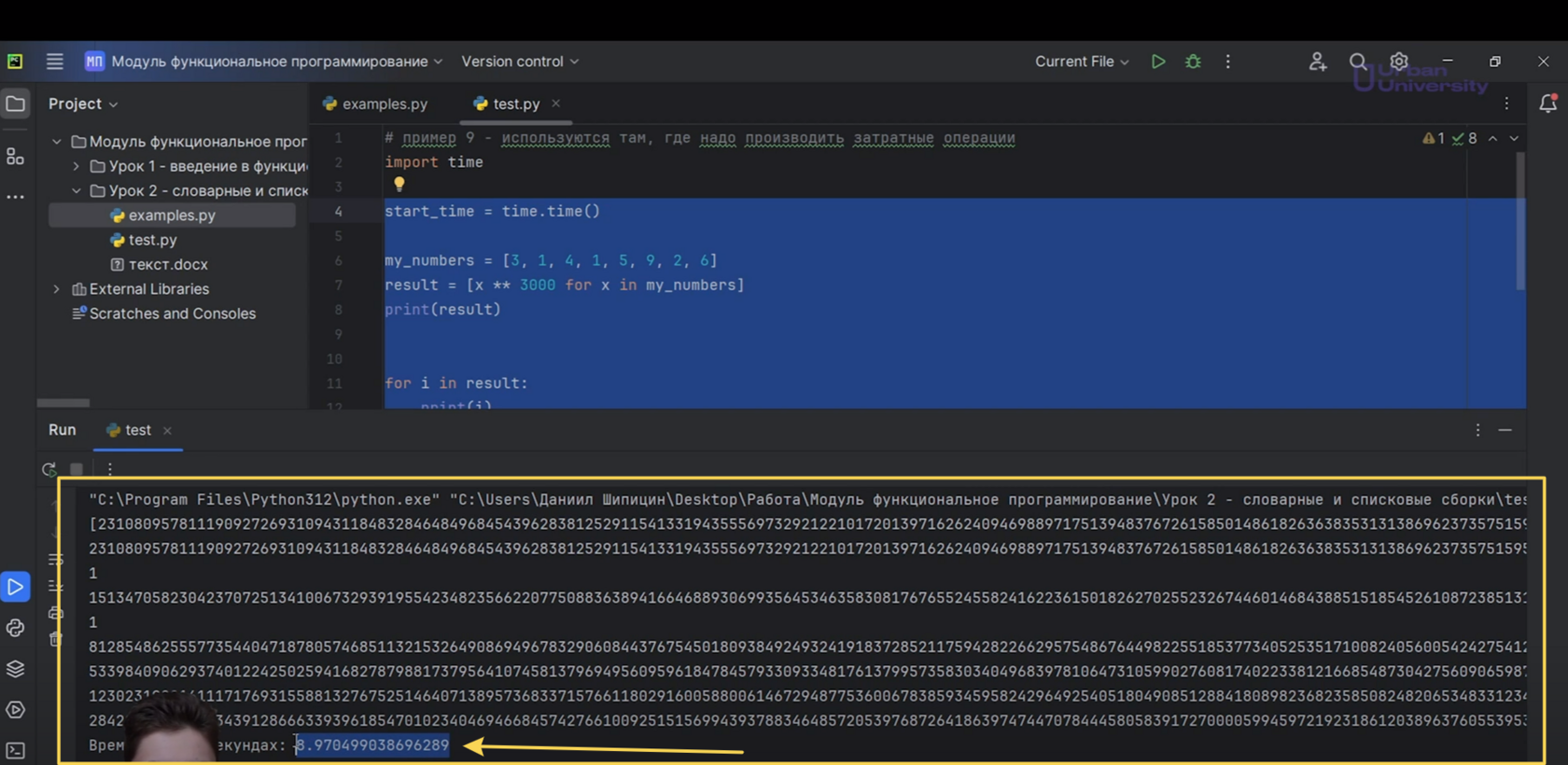


Рис.11

А теперь мы превратим наш списковую сборку в генераторную сборку (поменяем квадратные скобки на круглые). Попытаемся, во-первых, вывести«result». Но просто так у нас ничего не выводится, выводится просто «generator object» (Рис.12).

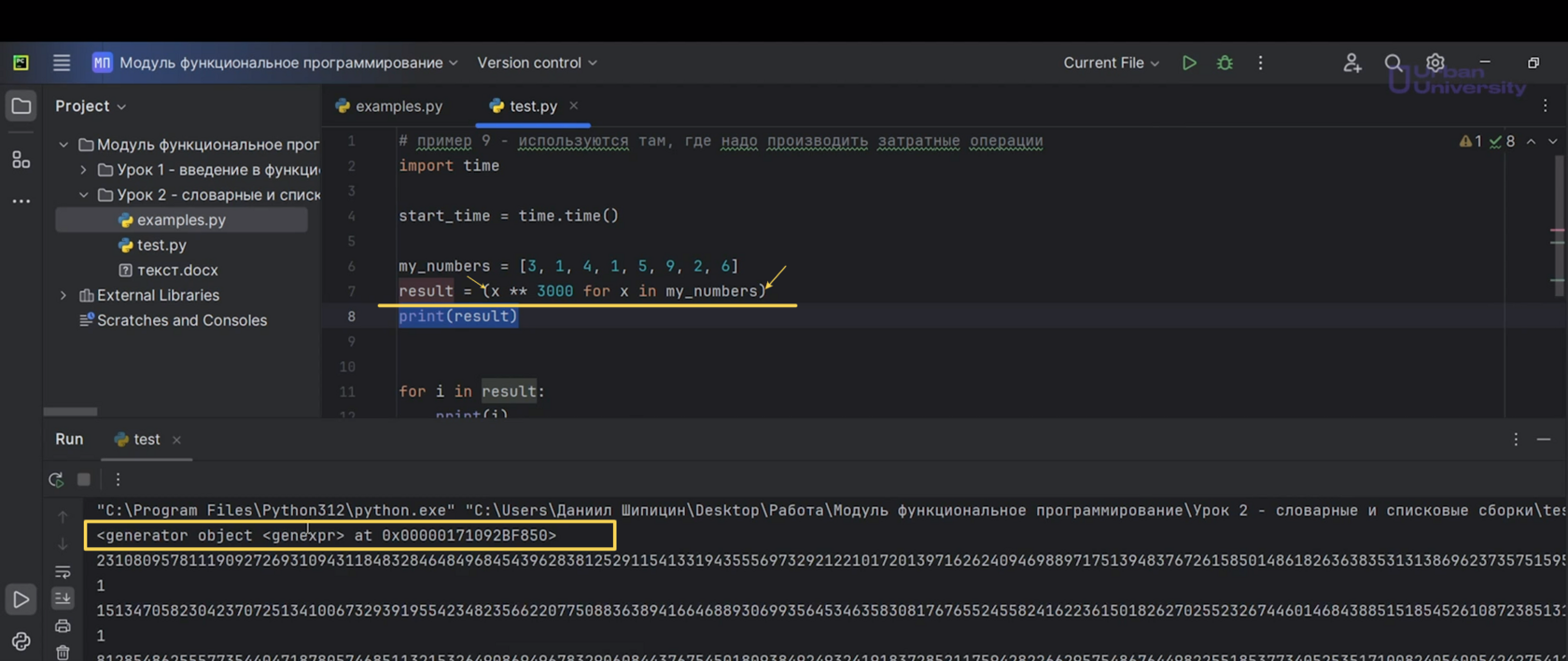


Рис.12

Потом с помощью цикла «for» мы пробегаемся, выводим каждый отдельный элемент. Причём вычисляем его только в данный момент, когда он появляется. То есть при необходимости и забывается этот объект (Рис.13).

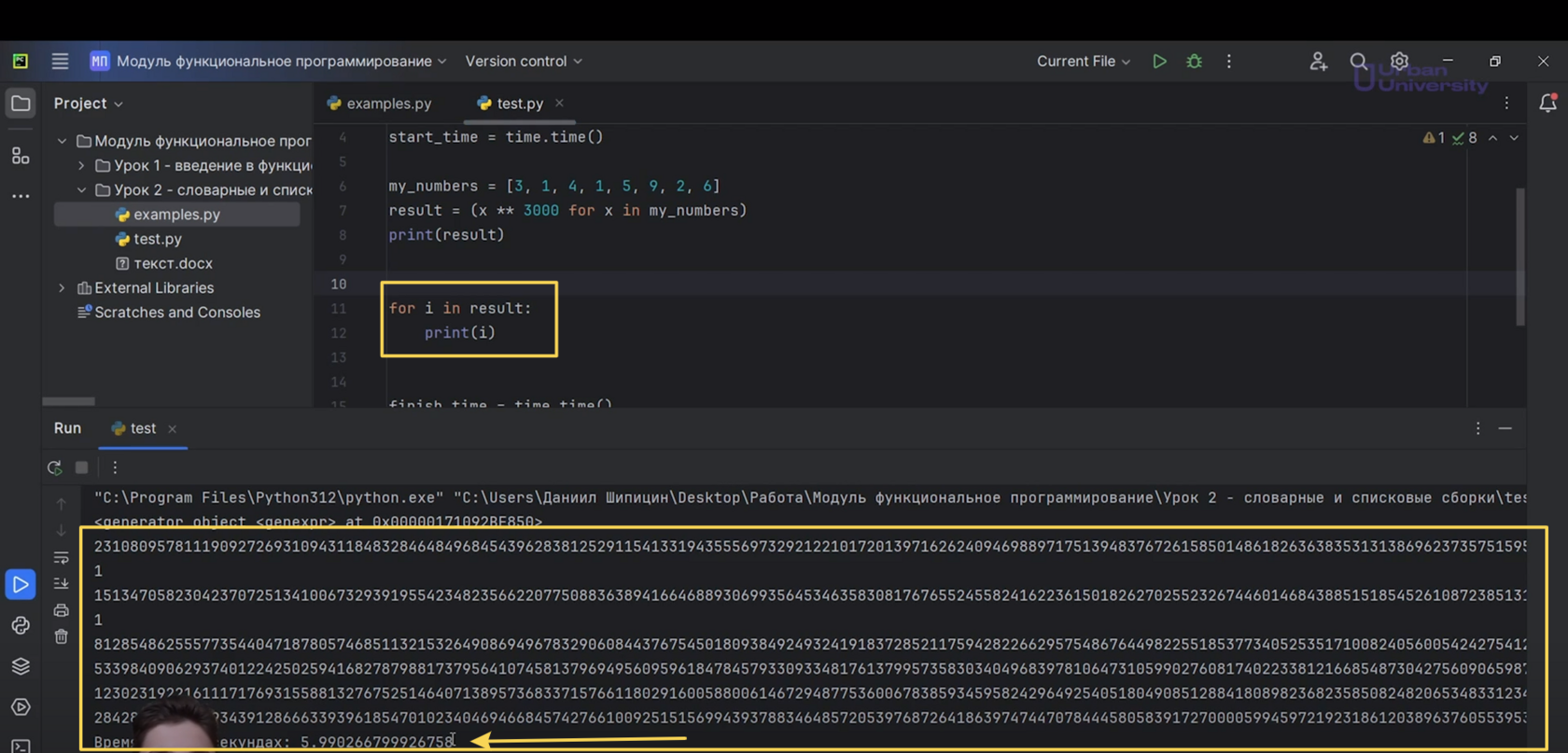


Рис.13

Во-первых, у нас программа намного быстрее выполнилась за 6 миллисекунд. То есть в прошлый раз было 9, а тут 6. Даже на таких маленьких данных программа уже быстрее выполняется.

А вы представьте себе огромный сервер, который каждую миллисекунду вычисляет объём данных намного больше этого. Поэтому на самом деле генераторные сборки они нужны, но это всего лишь часть тех возможностей, которые дают нам генераторы.

Тема с генераторами это тоже функциональное программирование. Поэтому мы ещё попозже с вами это все рассмотрим.

И в конце мы хотим вам рассказать, г**де у нас ещё применяются ленивые вычисления (Рис.14).** А точнее, где мы уже их применяли, но даже не подозревали об этом. Ленивые вычисления есть во встроенных функциях, таких как «range», «zip», «open», «map». Все эти функции, которые мы с вами до этого прошли, это функции, которые используют ленивые вычисления. Почему? Потому что если мы просто напечатаем результат этой функции, просто, например, сохраним их в переменную и напечатаем, у нас ничего не произойдёт, потому что вычисления не производились. Они будут производиться только тогда, когда они действительно нам будут нужны.

Также ленивые вычисления-это кастомизированные функции или классы. Это то, что мы с вами сегодня делали, то есть мы сами создавали генераторные сборки. Это точно также ленивое вычисление.

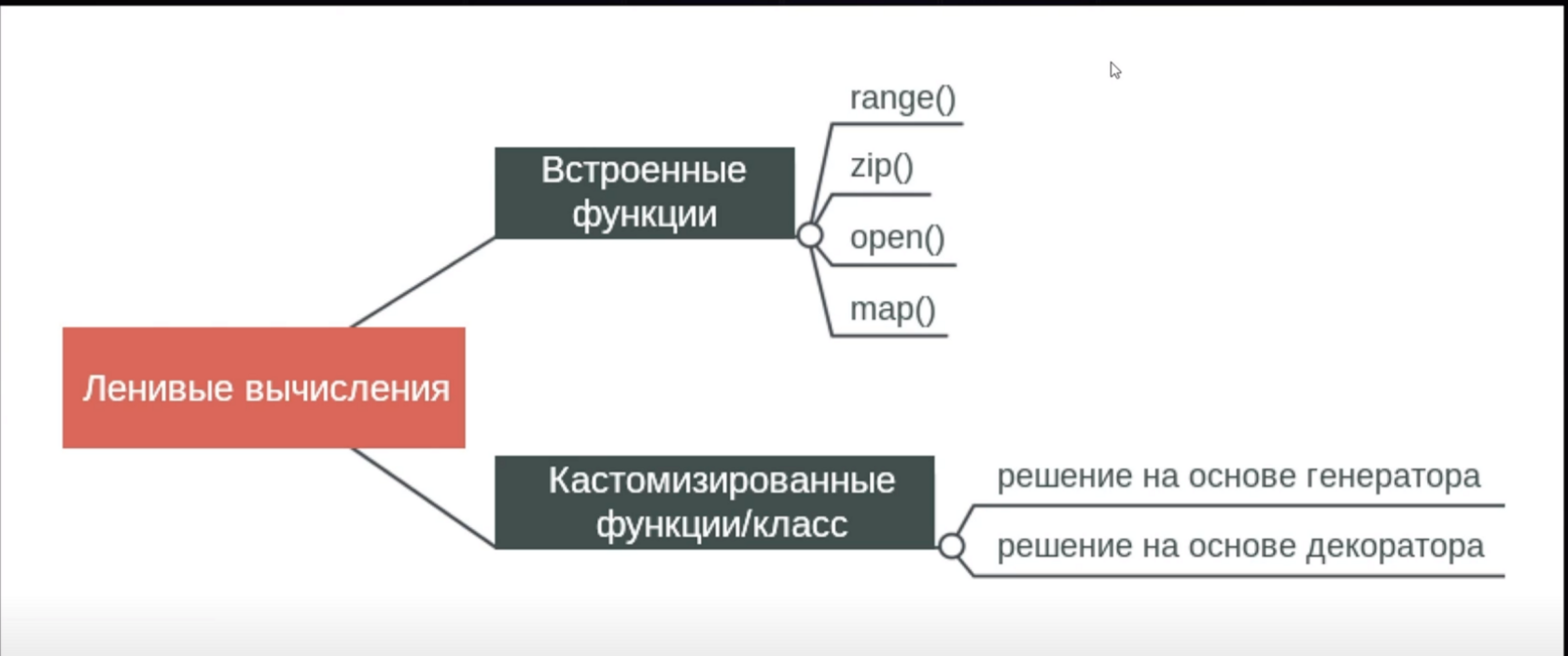


Рис.14

Рассмотрим пример: у нас дано два списка «list\_1» и «list\_2». Это просто списки для демонстрации функции «zip» и функции «map». А также у нас даны три переменные: «ran» «zp» «mp», в которых мы сохраняем результаты наших функций с ленивыми вычислениями, то есть функцию «range», функцию «zip», которая объединяет два наших списка, точнее, попарно объединяет элементы из двух наших списков и создаёт из них картежи, и функцию «map», с которой мы уже с вами прекрасно знакомы (Рис.15).

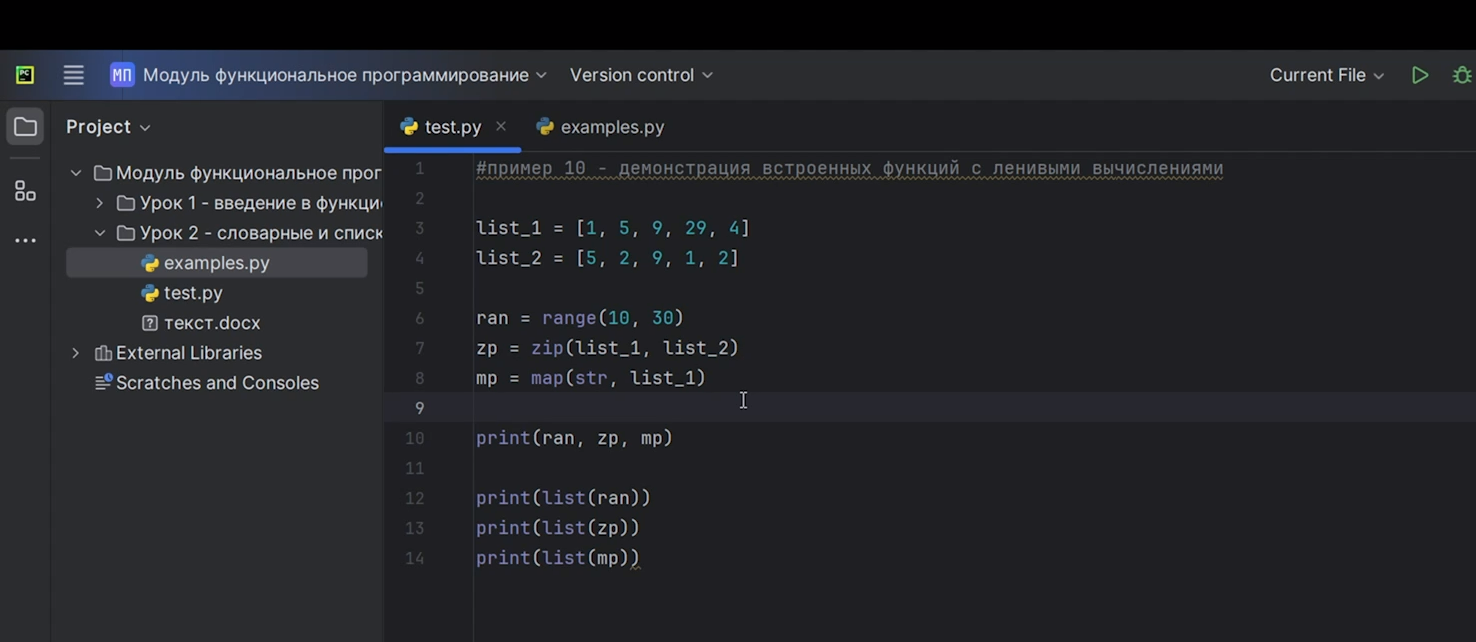


Рис.15

Если мы просто попытаемся вывести эти три переменные, то есть напечатать их, то у нас напечатается сама функция «range», то есть просто, как будто строчка была, «zip object» и «map object» (Рис.16).

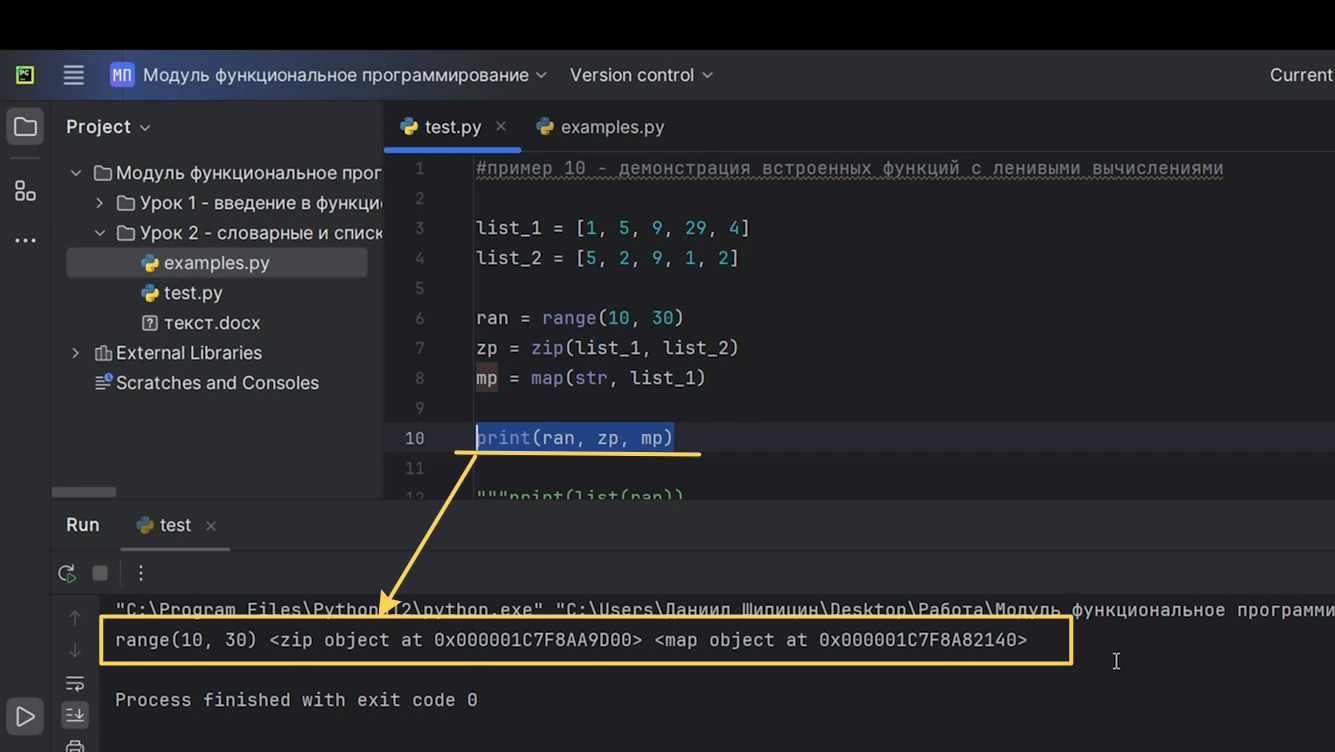


Рис.16

Значит вычисления у нас не производились: у нас не был дан большой список от 10 до 30, у нас не объединились два списка, и не поменялся наш список по операции «map», где мы пытались все наши числа перевести в строчки. Ничего из этого не произошло. Просто потому что эти функции сейчас не вычисляли никакой результат. Когда он действительно нам нужен будет, когда нам нужно будет знать какой-то определённый элемент из вот этих функций или все элементы из этих функций, тогда они действительно вычислятся. Если мы решим перевести каждую из этих ленивых функций в список, то есть мне нужно сделать так, чтобы все данные сохранялись где-то, то у нас все получится (Рис.17).

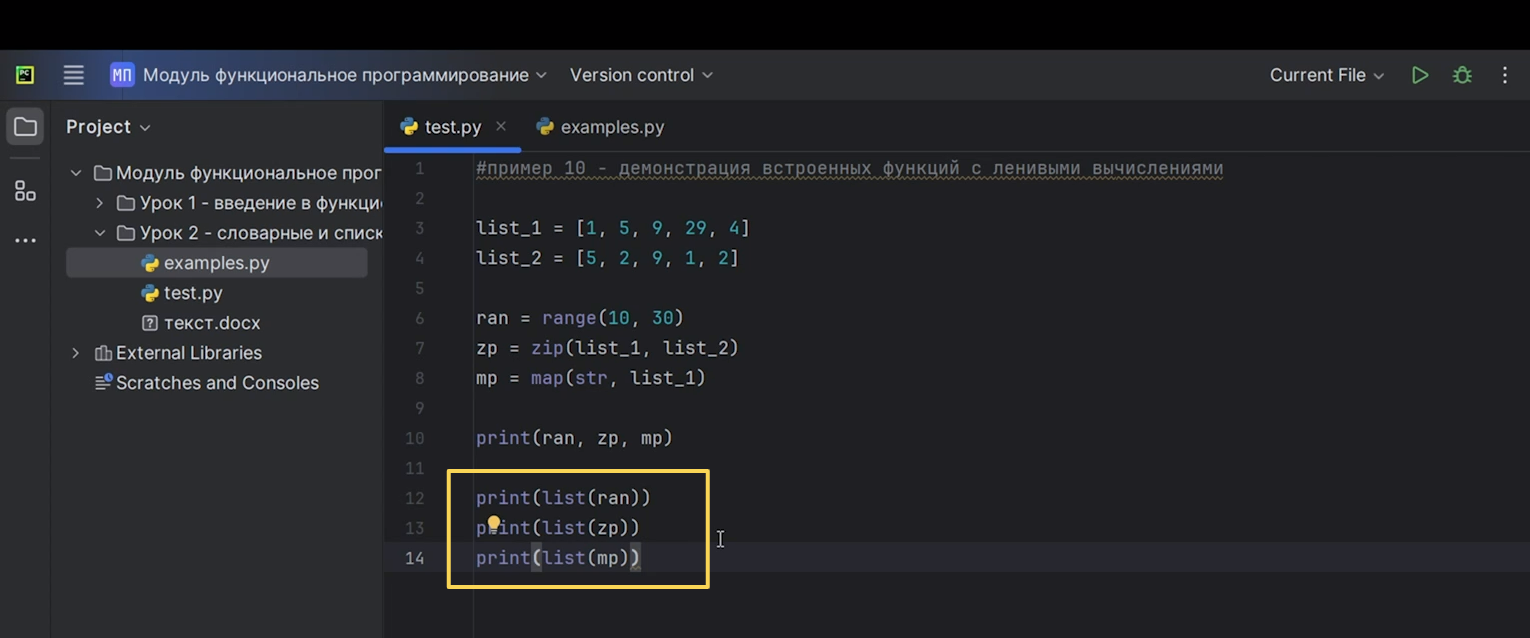


Рис.17

Как вы видите, у нас создался список от 10 до 30 не включительно. У нас объединились два списка: по парно у них элементы объединились и создали между собой картежи. А также у нас все элементы нашего списка перевелись строчки (Рис.18).

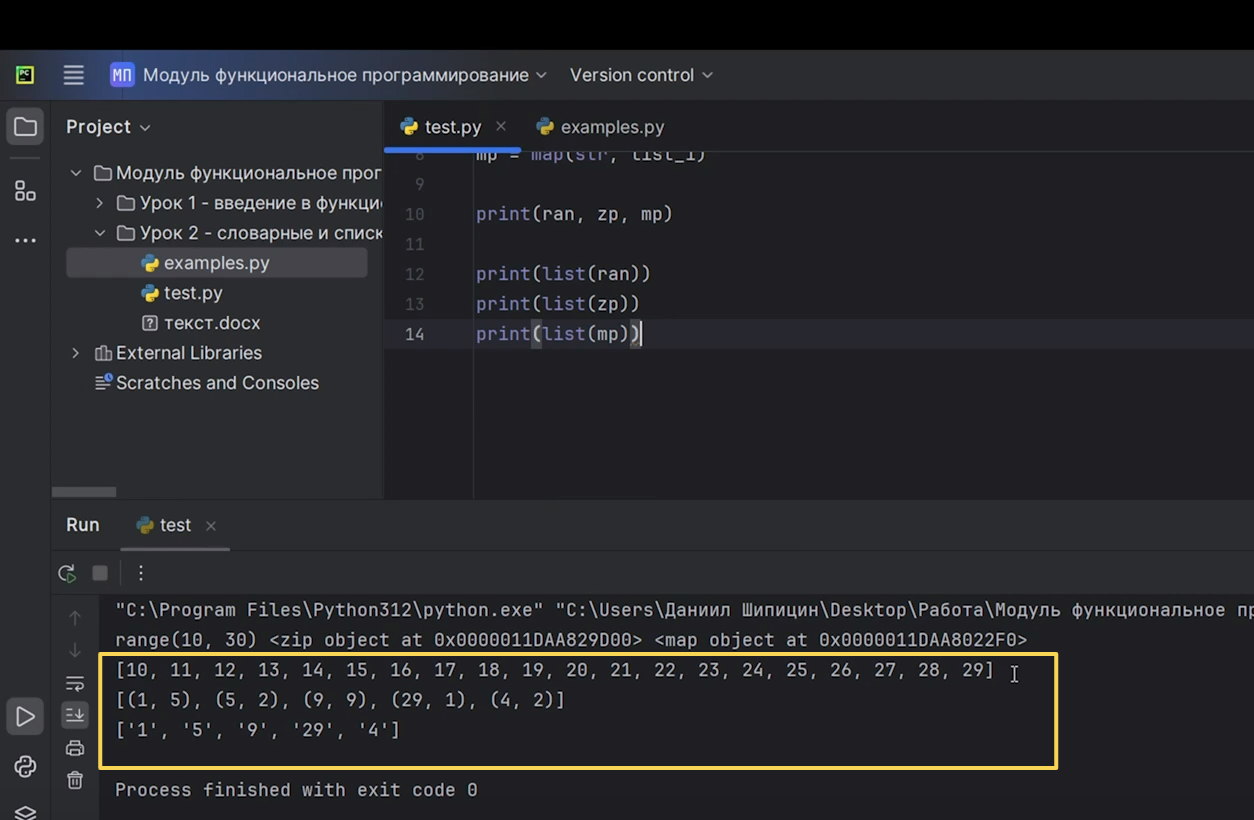


Рис.18

Мы уже с вами почти всегда использовали как минимум «range», и это была функция с ленивыми вычислениями. Очень мощный инструмент. Действительно стоит понимать, как это, когда работает. И это можно очень круто применять на практике. Когда мы с вами поближе познакомимся с генераторами, точнее, с тем функционалом, который они нам дают, тогда вы действительно поймёте, как это нужно использовать.