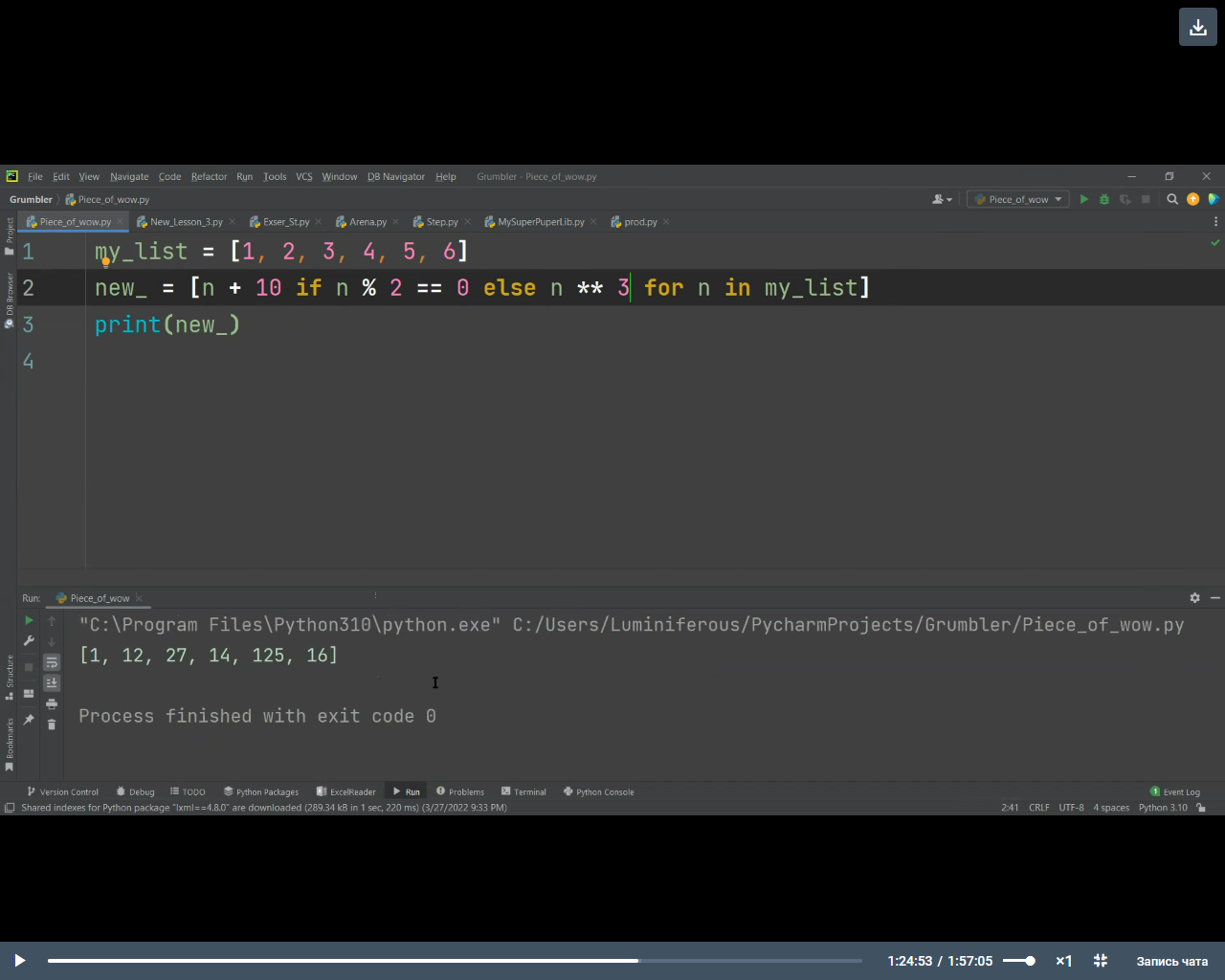
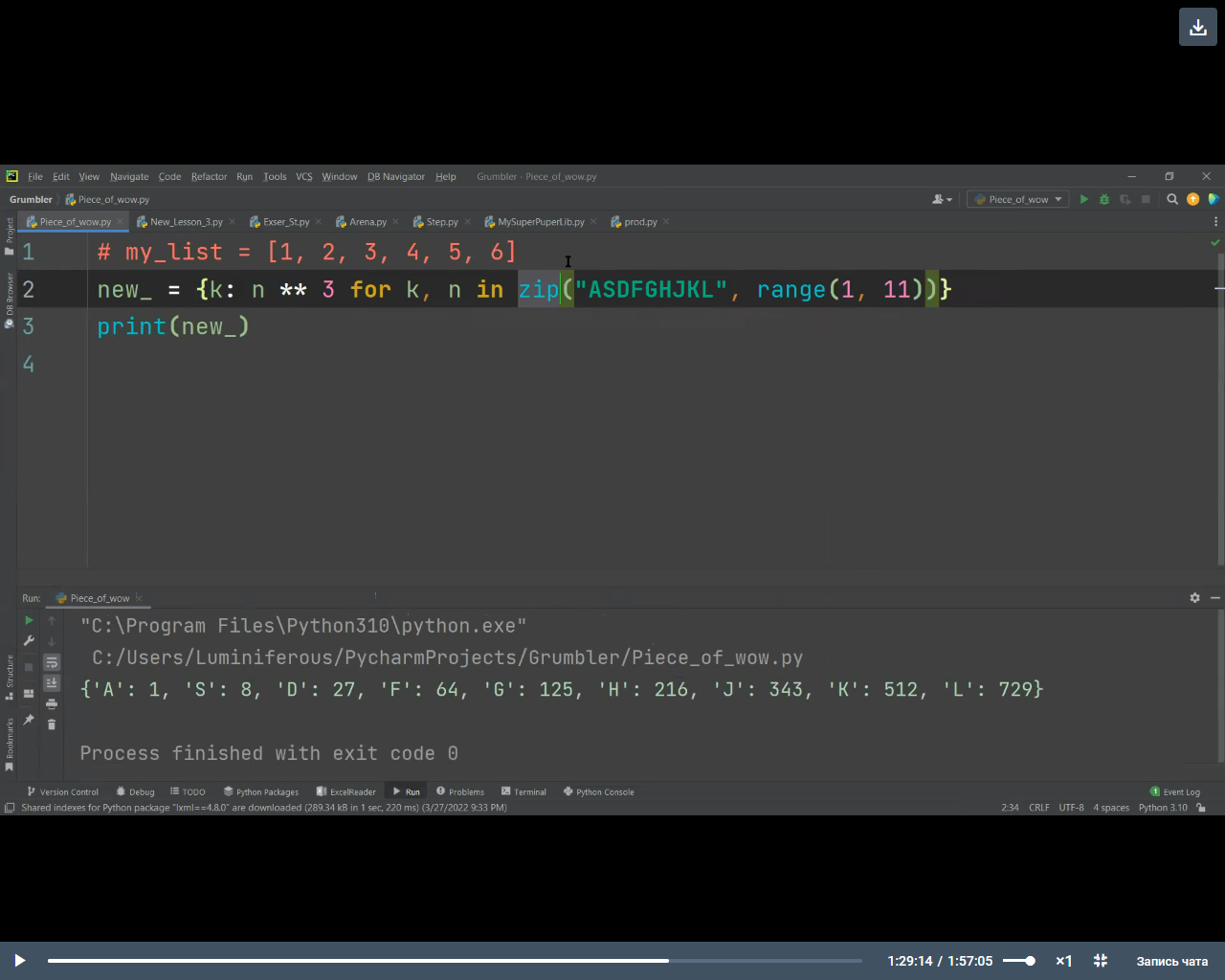


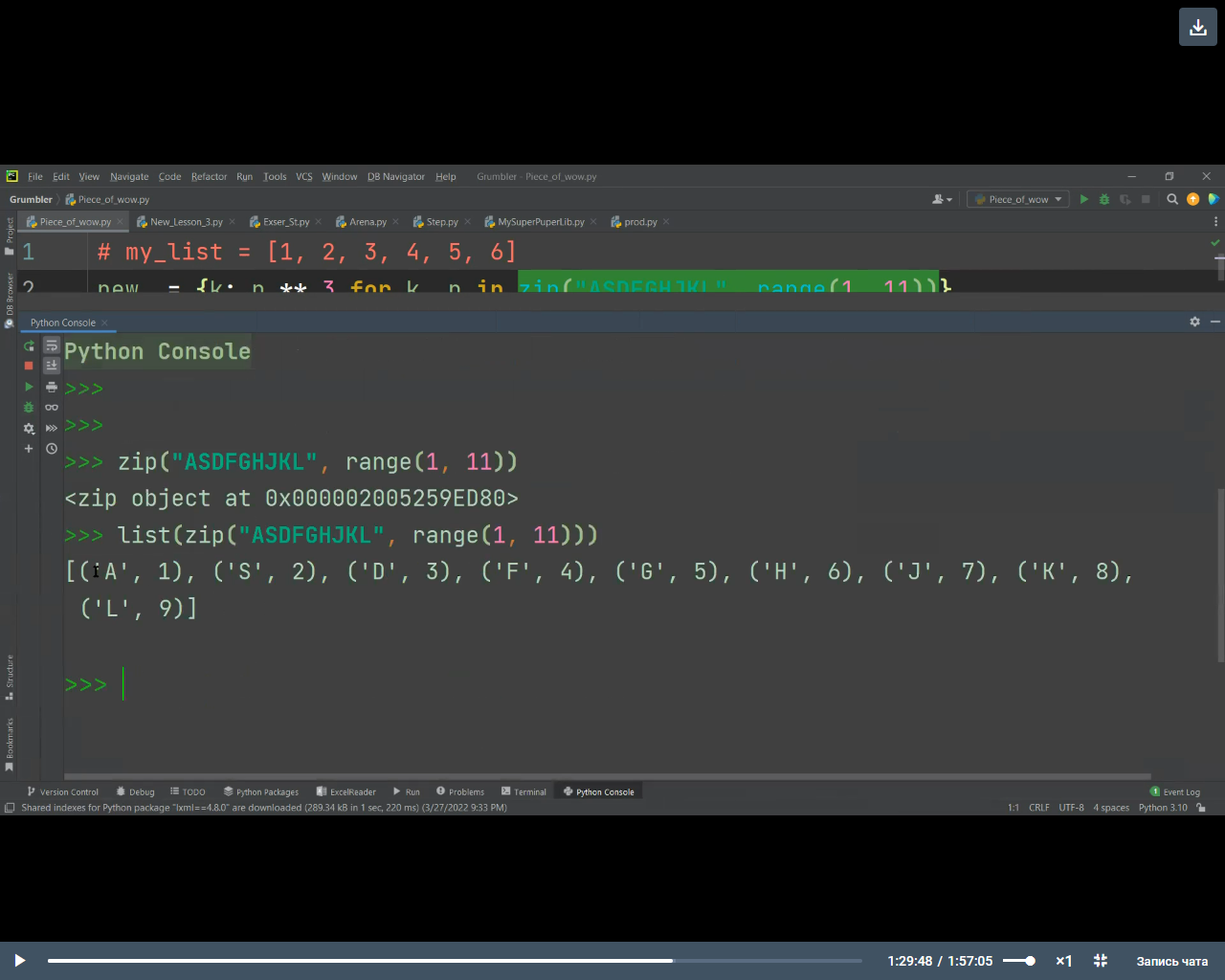
При использовании тернарного оператора, он должен располагаться вначале. Все остальные условия пишутся вконце.



Генератор словарей. Пример генератора по ключу и по значению.

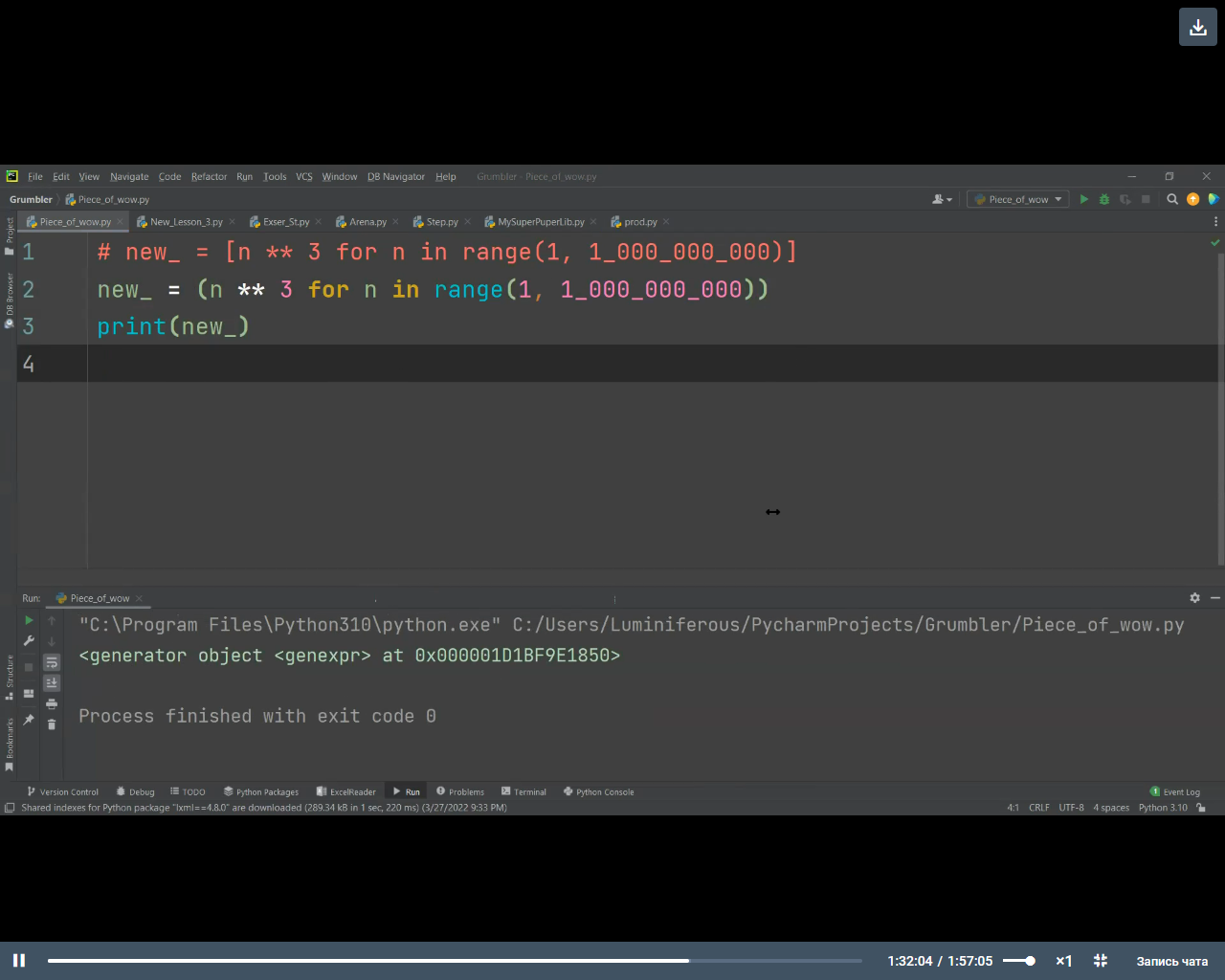


Конструкция zip. Обычно ее оборачивают в список. Получается список кортежей. Он берет по элементно, пока есть пара.



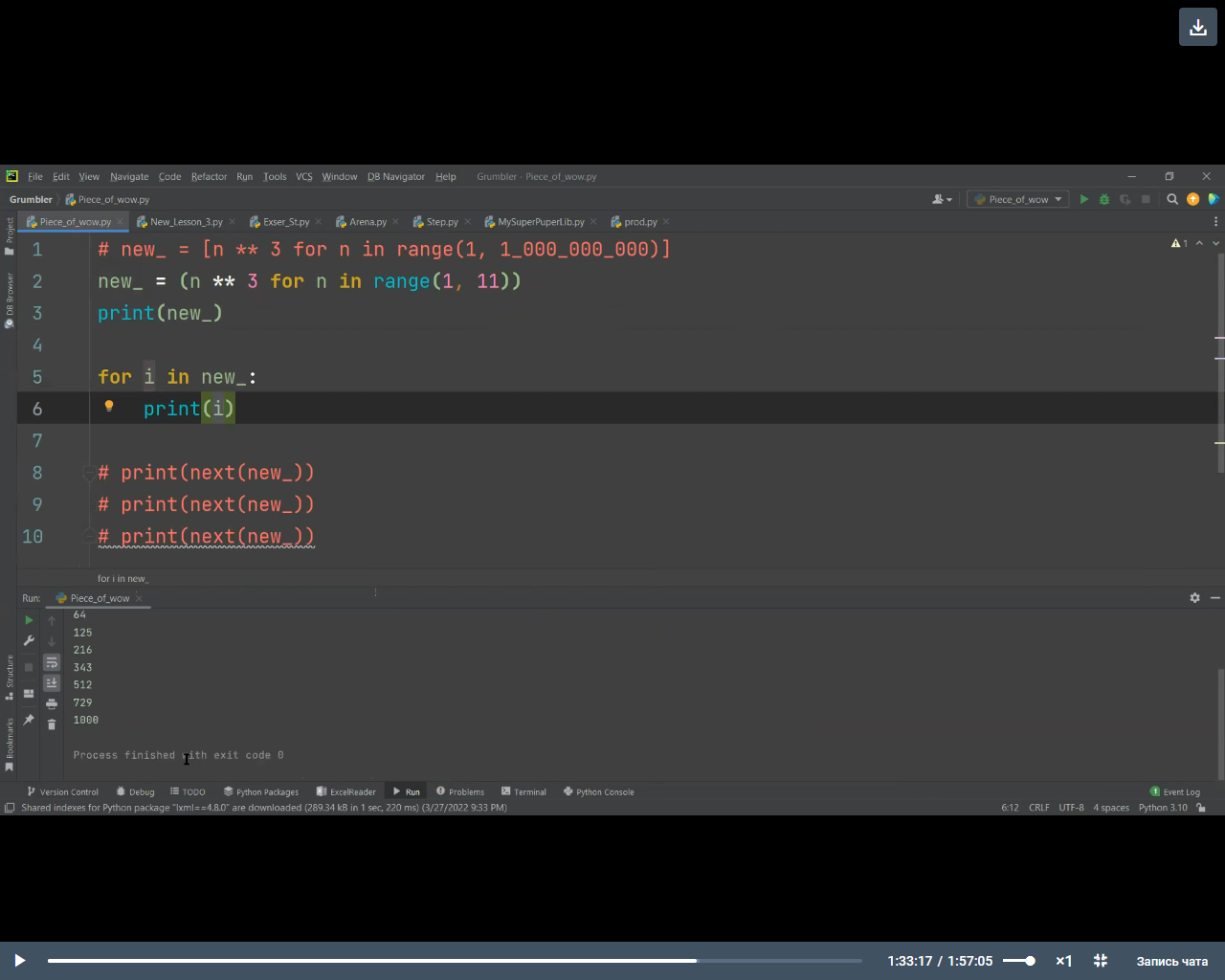
При генераторе списка может зависнуть программа из-за большого количества данных.

Используются круглые скобки ( ) для генератора, а потом значения достаются по одному через функцию next.

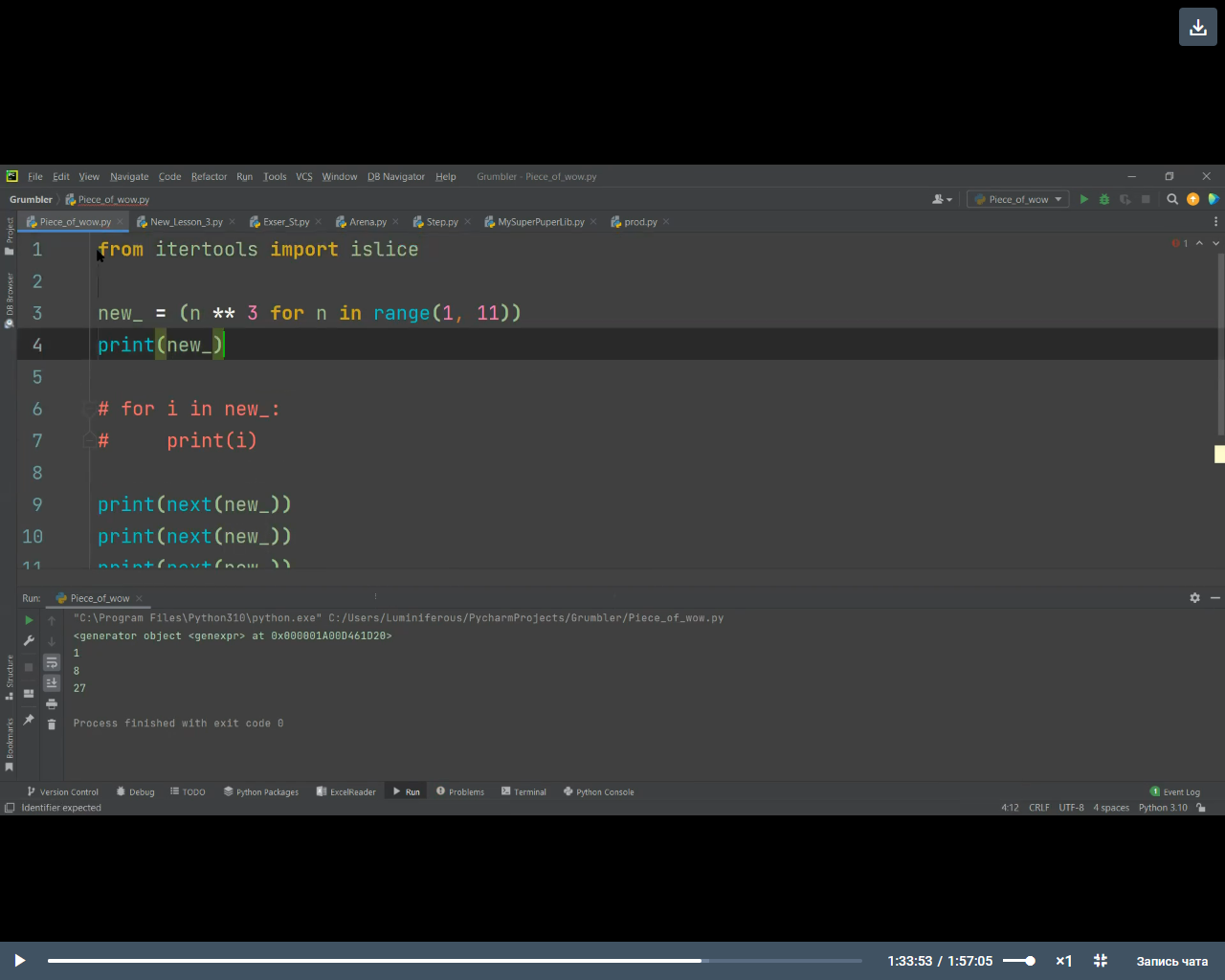


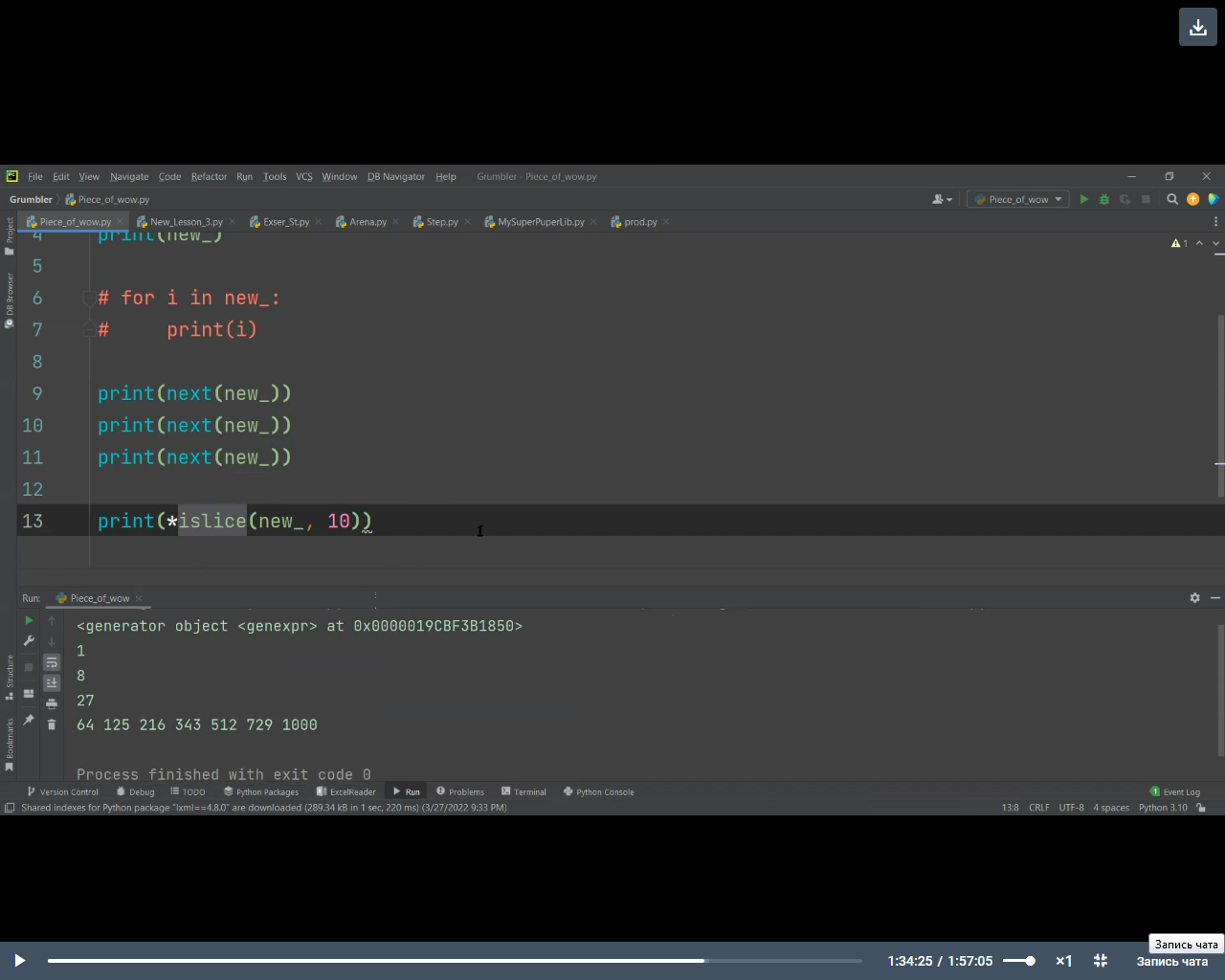


Но чаще используется цикл



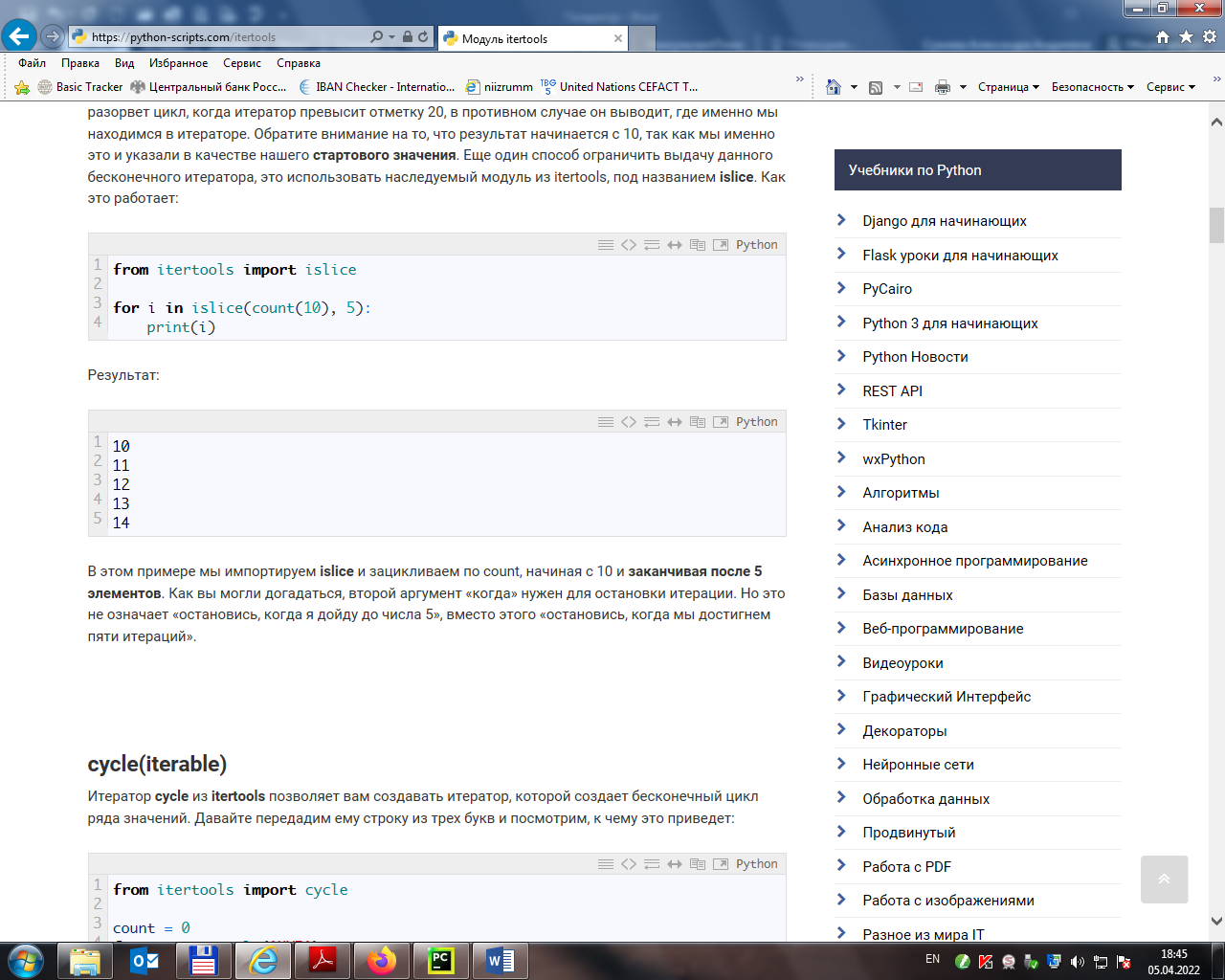
Третий способ. Islice



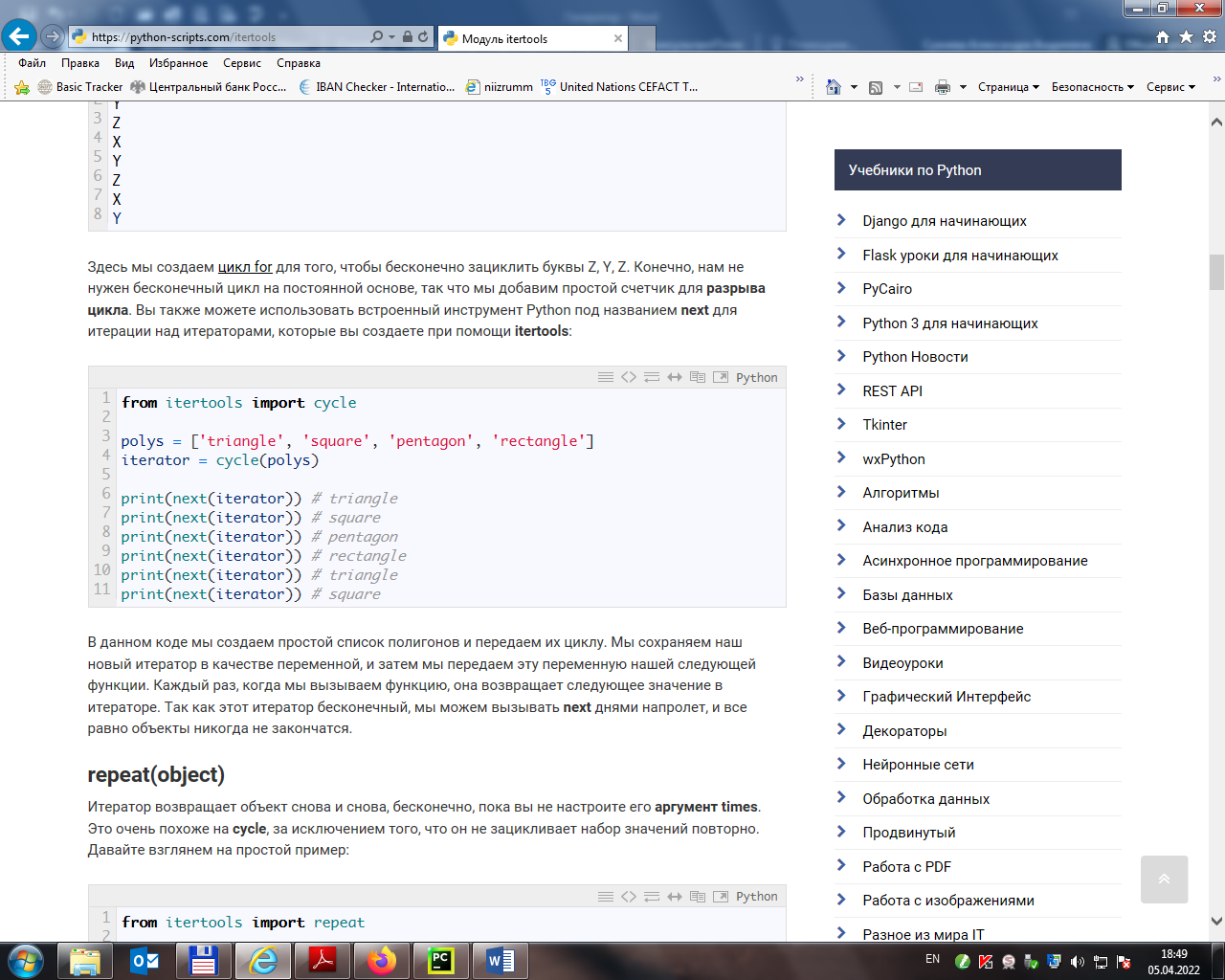


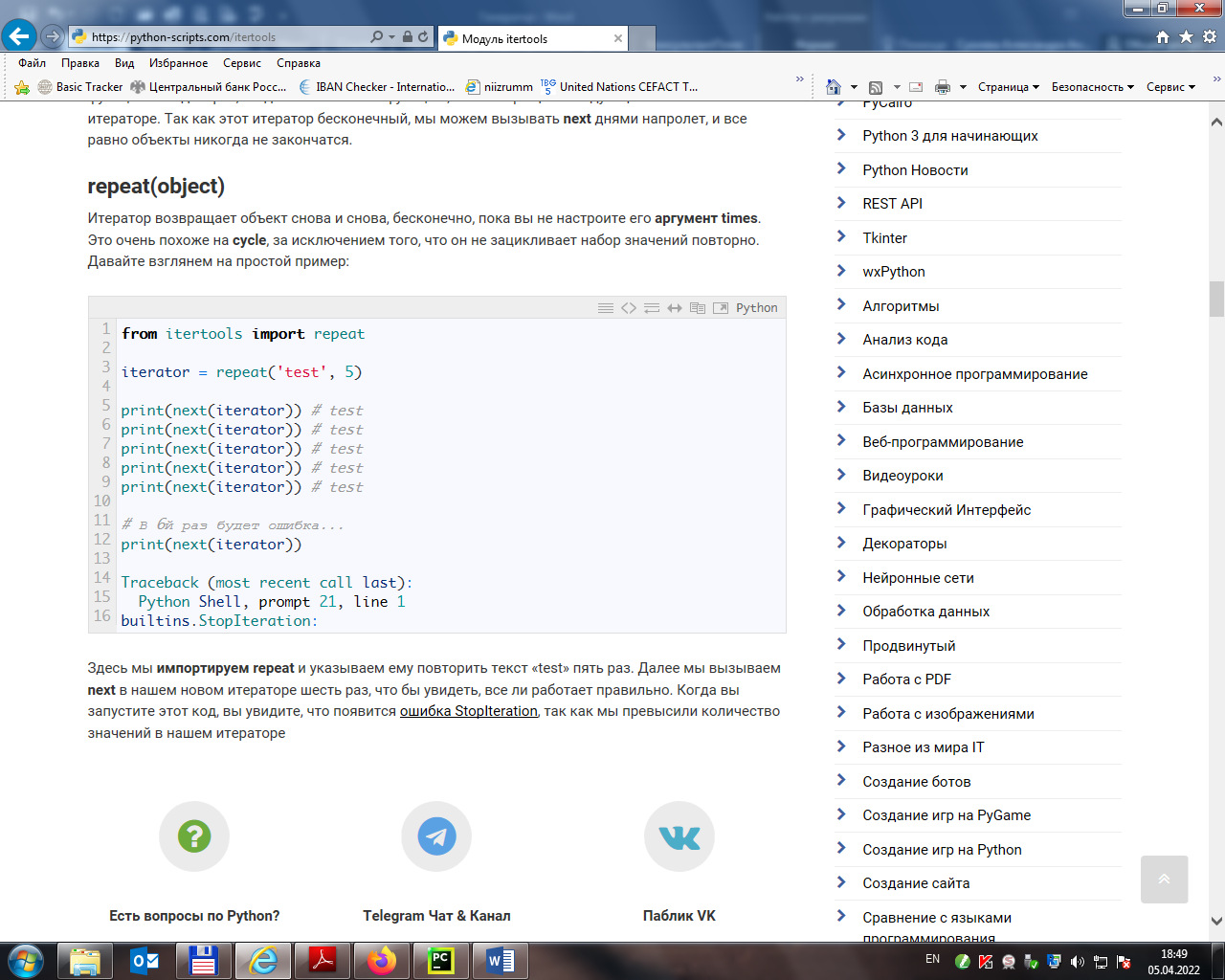
Особенность генератора в том, что он движется вперед. То, что достали, уже нельзя достать заново.

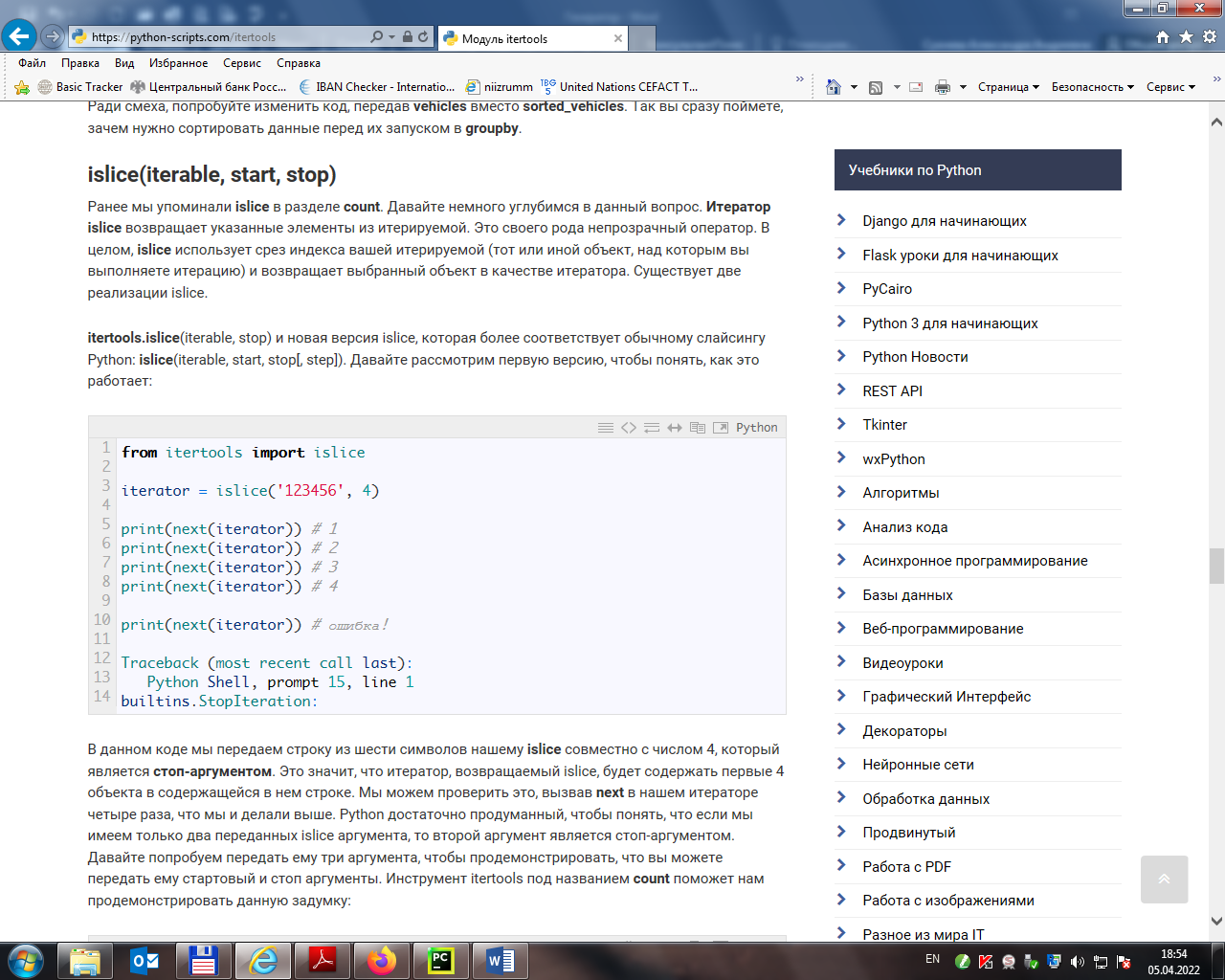
Генератор – это формула, list comprehensions – это готовая структура.

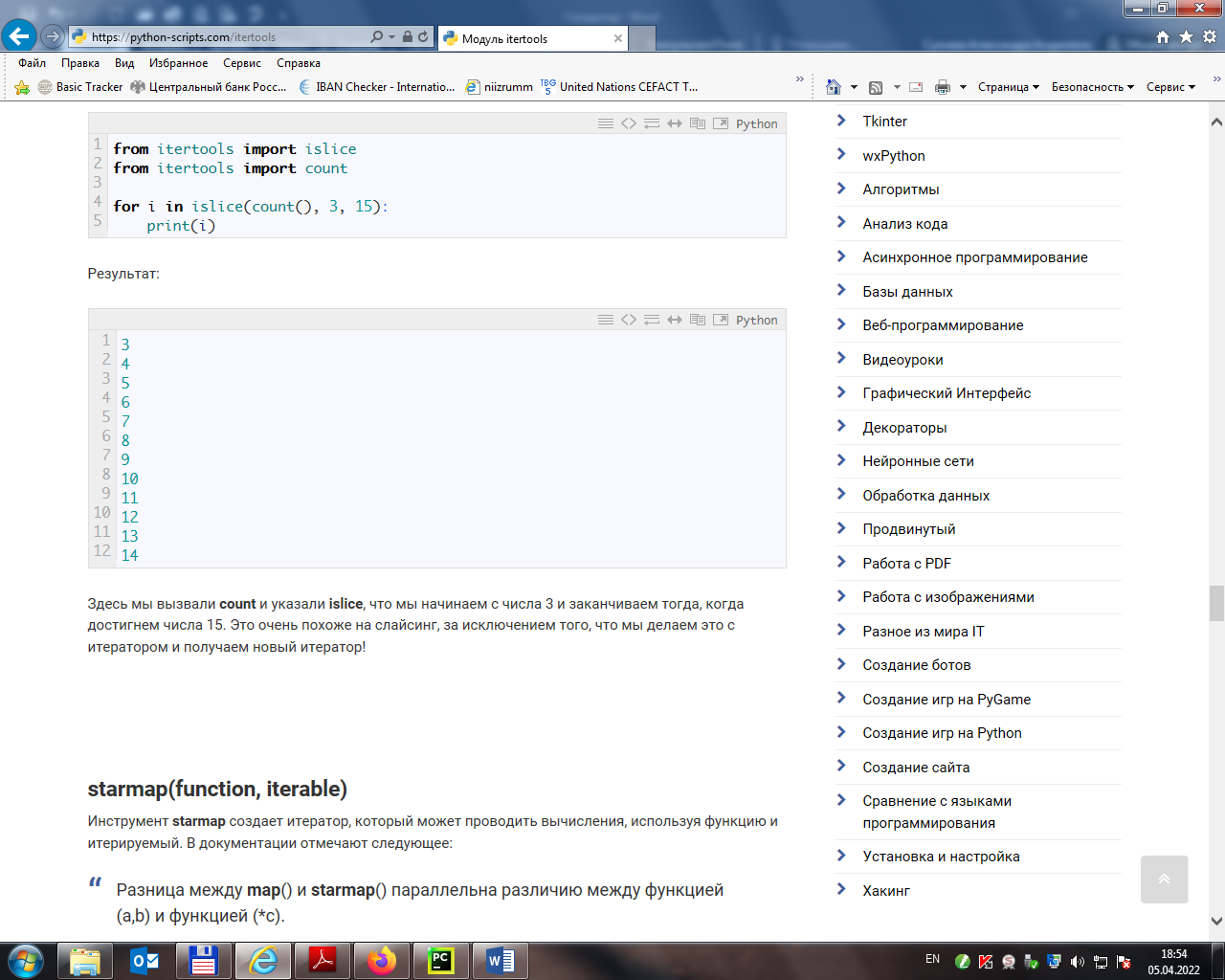


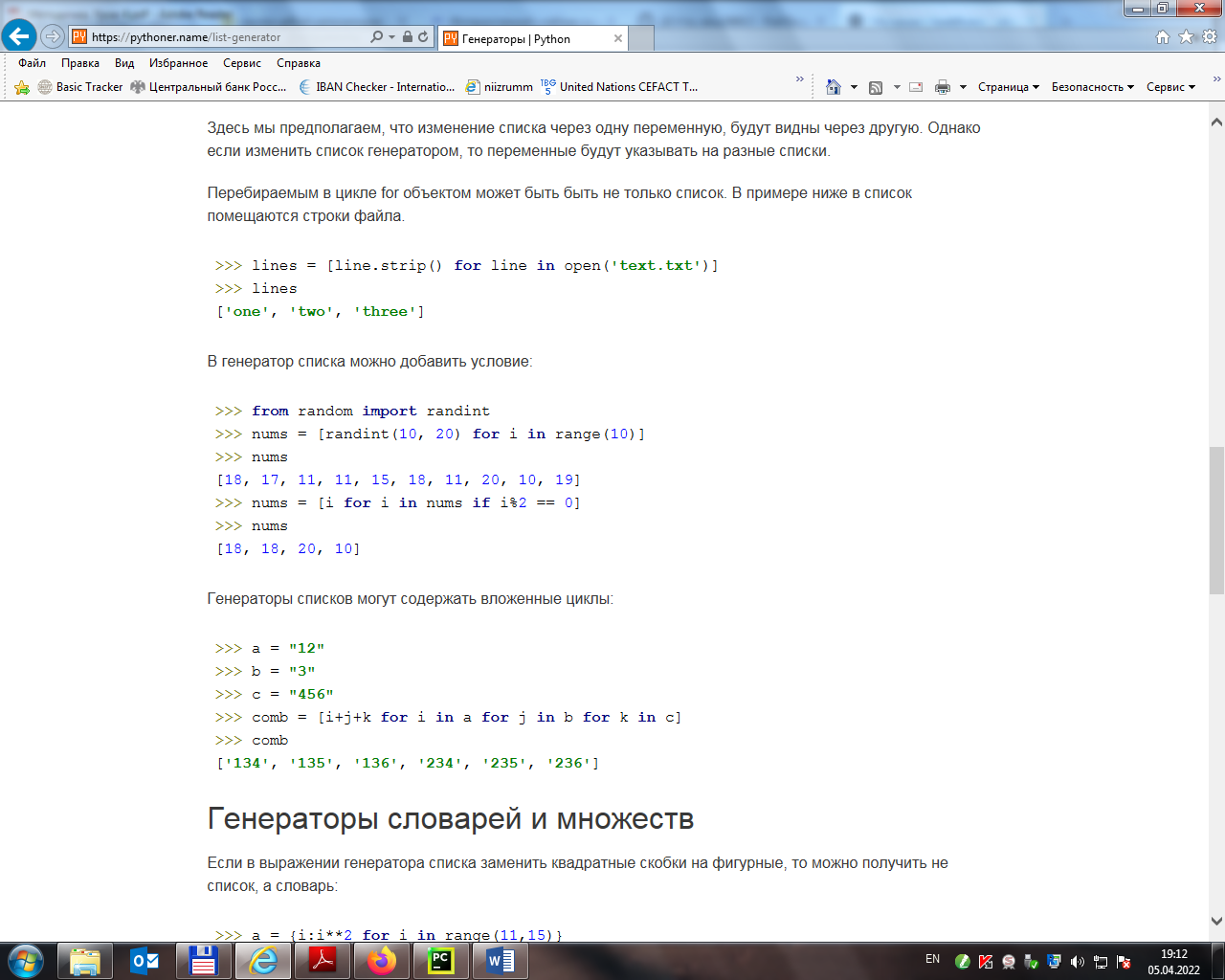












**Генераторы**

Выражения, создающие объекты-генераторы, похожи на выражения, генерирующие списки, словари и множества за одним исключением. Чтобы создать генераторный объект, надо использовать круглые скобки:

>>> a = (i for i in range(2, 8))

>>> a

<generator object <genexpr> at 0x7efc88787910>

>>> for i in a:

... print(i)

...

2

3

4

5

6

7

Второй раз перебрать генератор в цикле for не получится, так как объект-генератор уже сгенерировал все значения по заложенной в него "формуле". Поэтому генераторы обычно используются, когда надо единожды пройтись по итерируемому объекту.

Кроме того, генераторы экономят память, так как в ней хранятся не все значения, скажем, большого списка, а только предыдущий элемент, предел и формула, по которой вычисляется следующий элемент.

Выражение, создающее генератор, это сокращенная запись следующего:

>>> def func(start, finish):

... while start < finish:

... yield start \* 0.33

... start += 1

...

>>> a = func(1, 4)

>>> a

<generator object func at 0x7efc88787a50>

>>> for i in a:

... print(i)

...

0.33

0.66

0.99

Функция, содержащая yield, возвращает объект-генератор, а не выполняет свой код сразу. Тело функции исполняется при каждом вызове метода \_\_next\_\_(). В цикле for это делается автоматически. При этом функция сохраняет значения переменных от предыдущего вызова.

Если нет необходимости использовать функцию многократно, проще использовать выражение:

>>> b = (i\*0.33 for i in range(1,4))

>>> b

<generator object <genexpr> at 0x7efc88787960>

>>> for i in b:

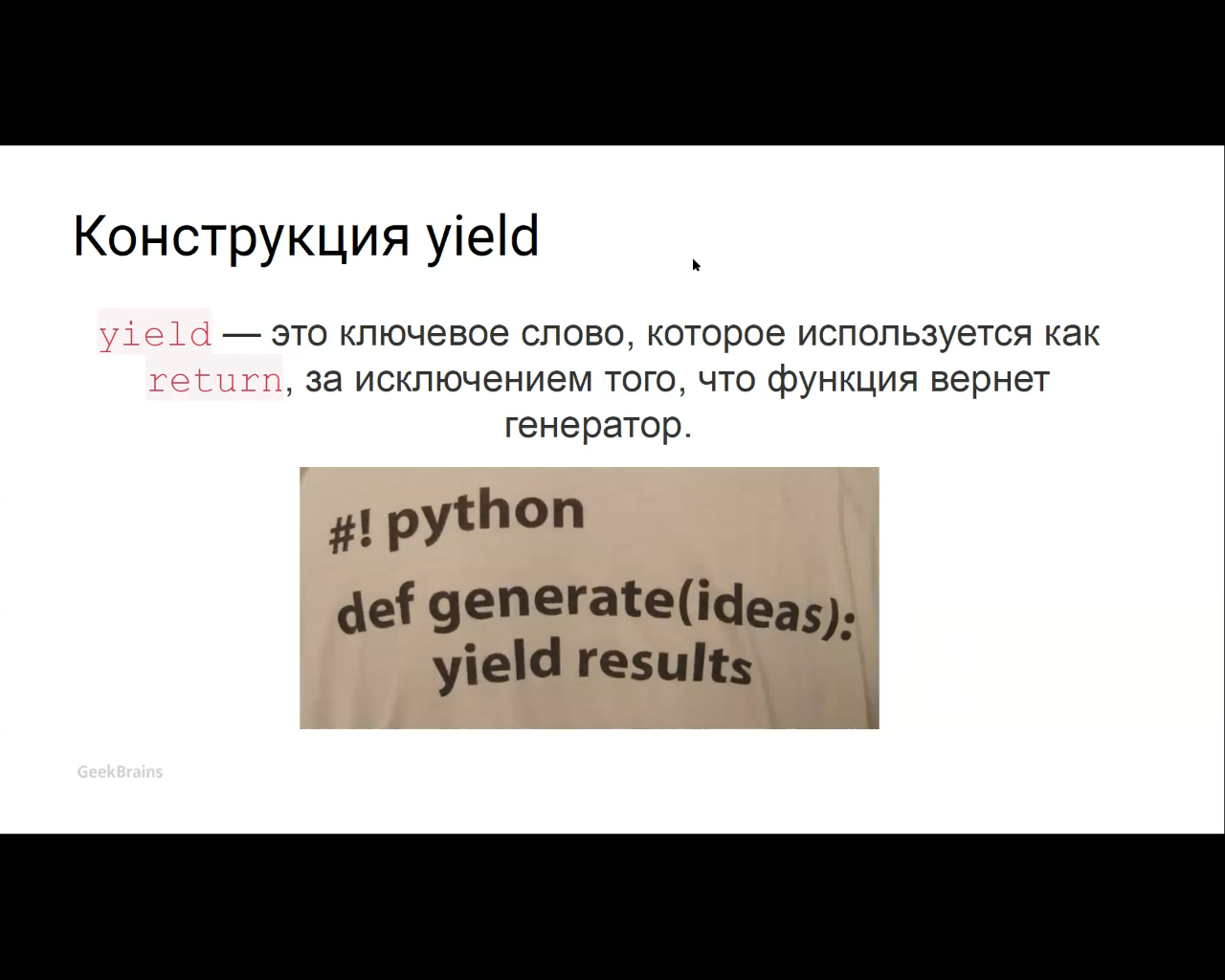
... print(i)

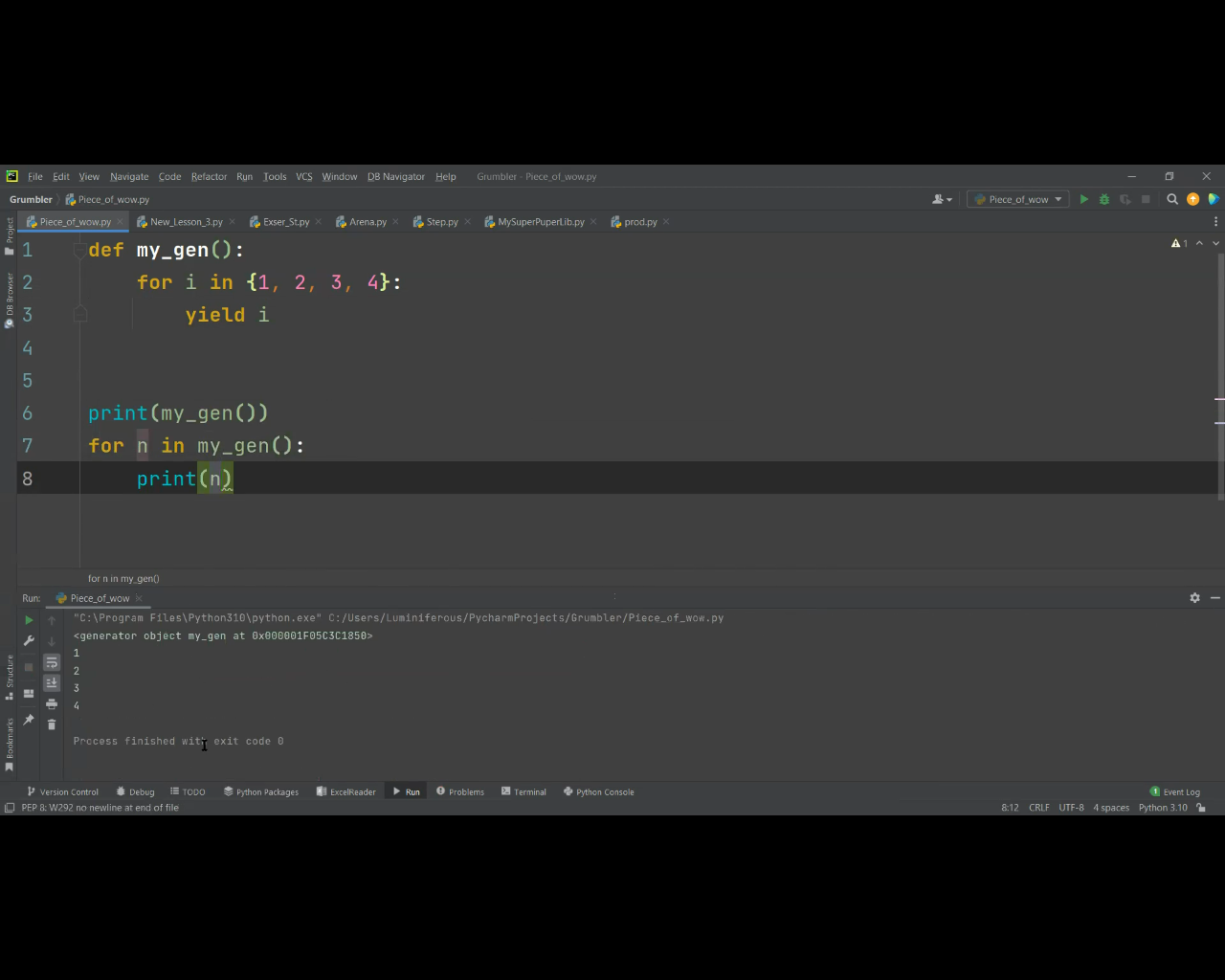
...

0.33

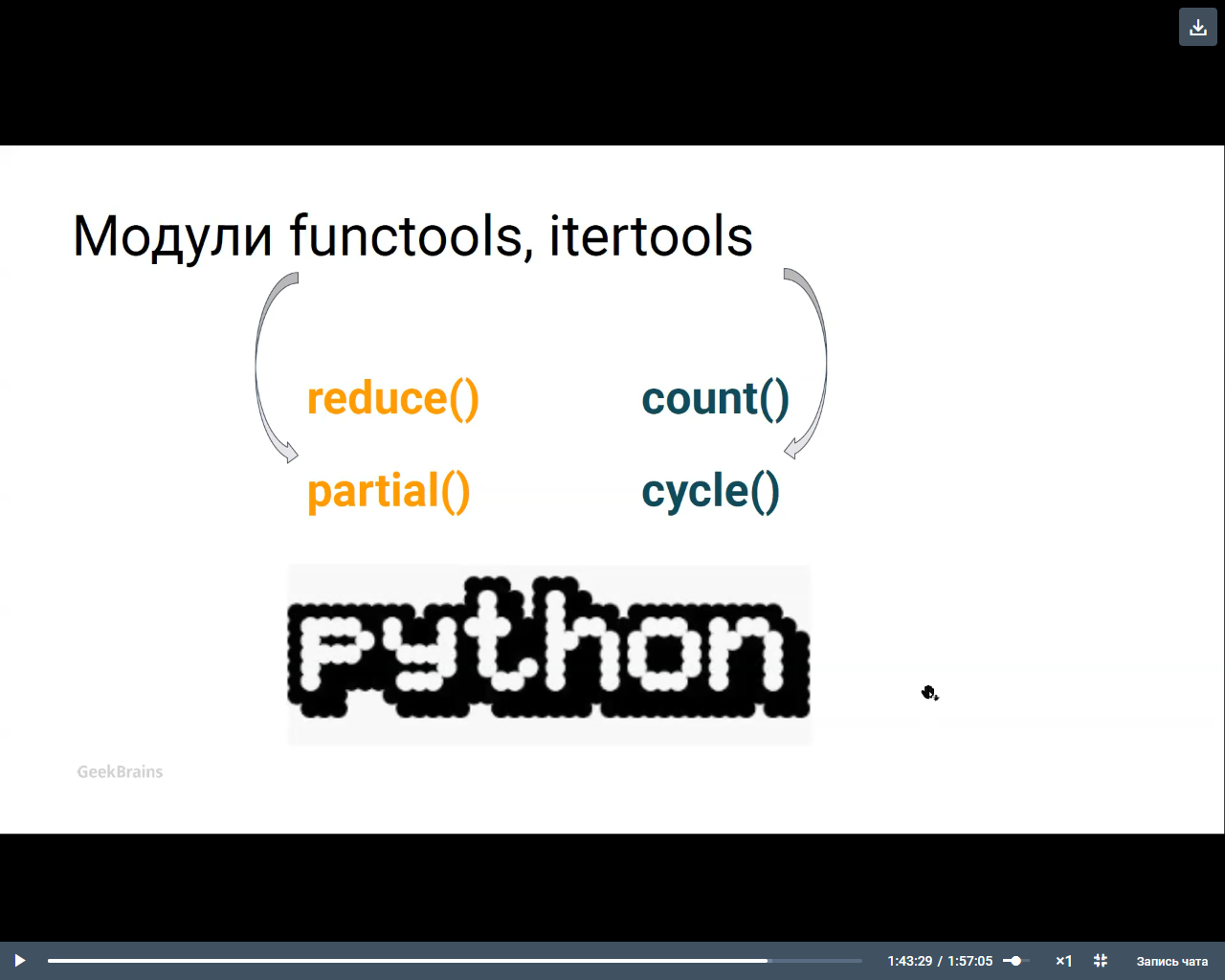
0.66

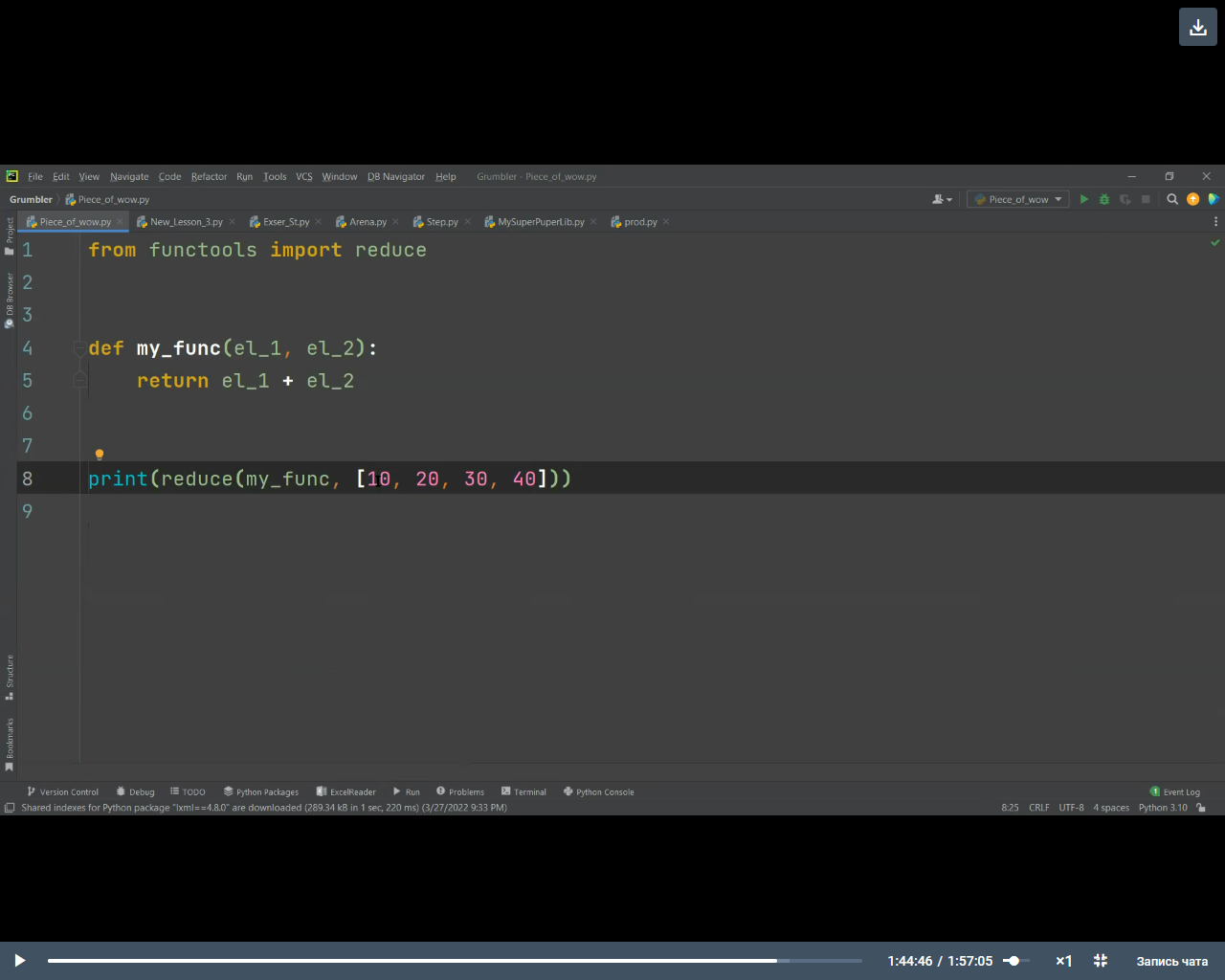
0.99



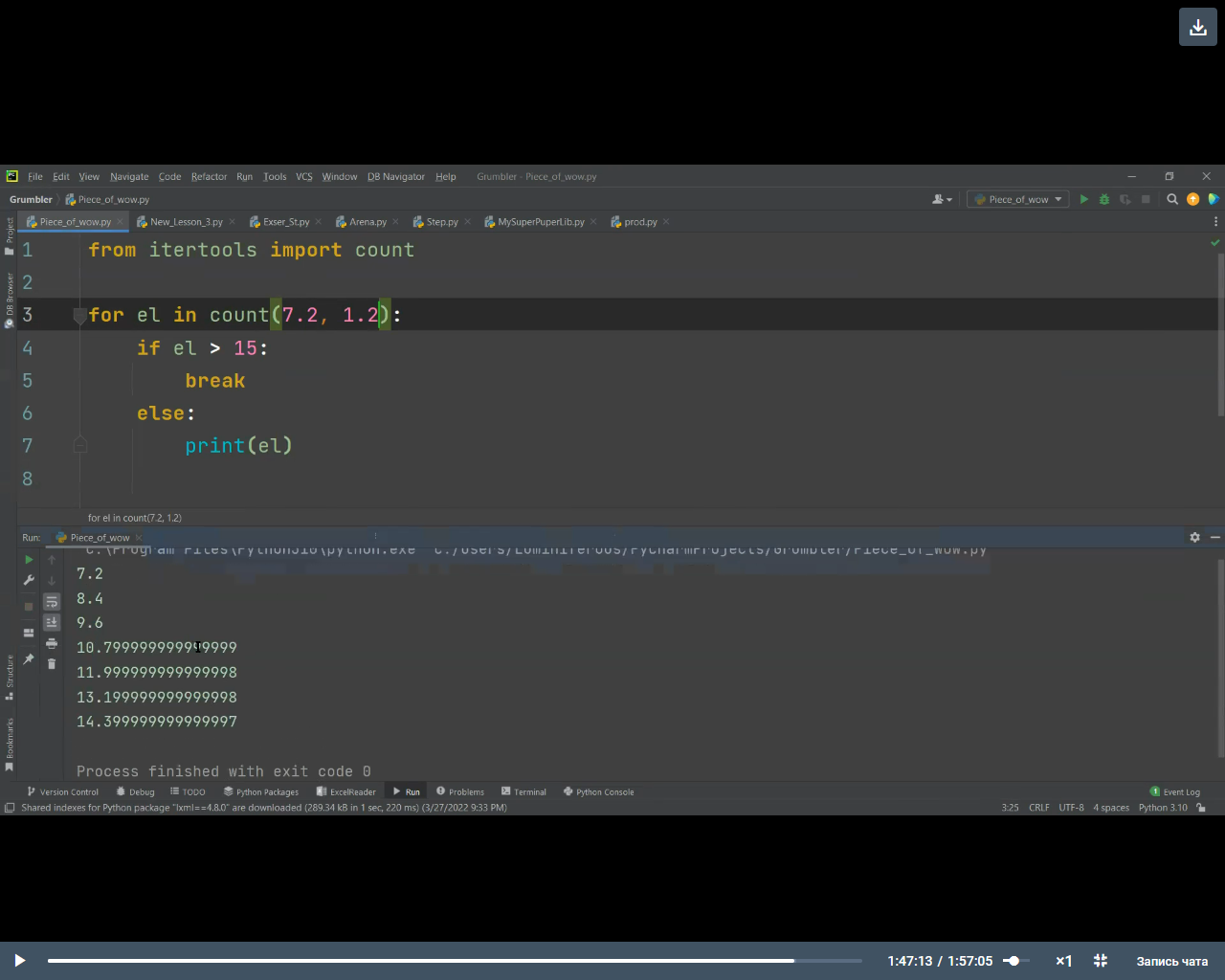


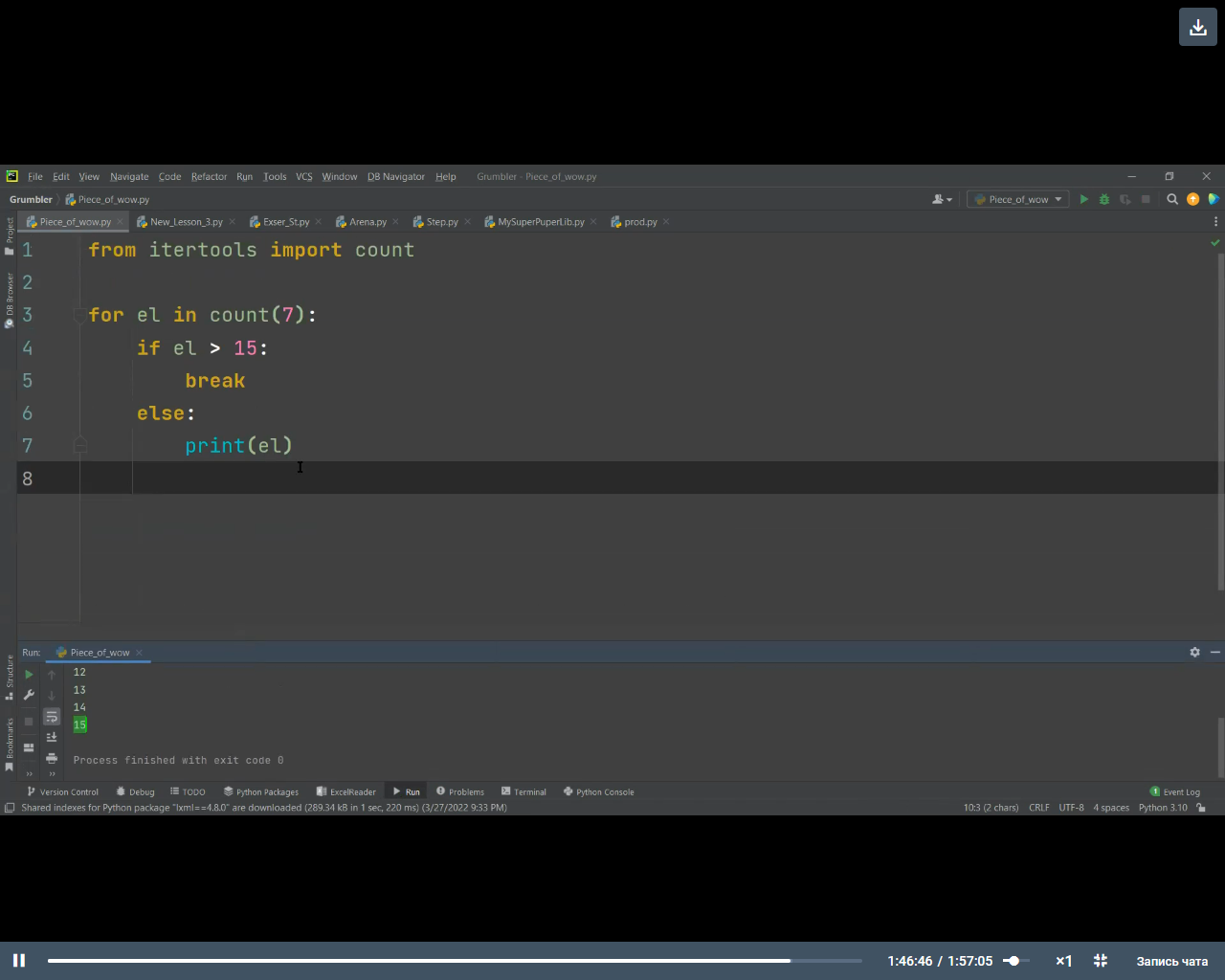
Используется для генератора, который находится в функции. Return вернет только первую итерацию и на этом все закончится. Yield проходит по всему генератору до конца. После него могут быть другие команды.





В итоге получается одно число (100) в скобках.

Count работает с числами (в т.ч. с дробными, может иметь шаг).



Cycle работает со строками. Будет перебирать содержимое списка n раз.

