**Introduction to generator expressions**

Now that you're getting a bit more comfortable with comprehensions, we're going to check out generators, which are related to comprehensions in way that will soon become evident.

**Generator expressions**

Recall that this list comprehension will create a list of the first 10 even numbers. Now lets replace the square brackets with round parentheses and voila! Something called a generator object has been created.

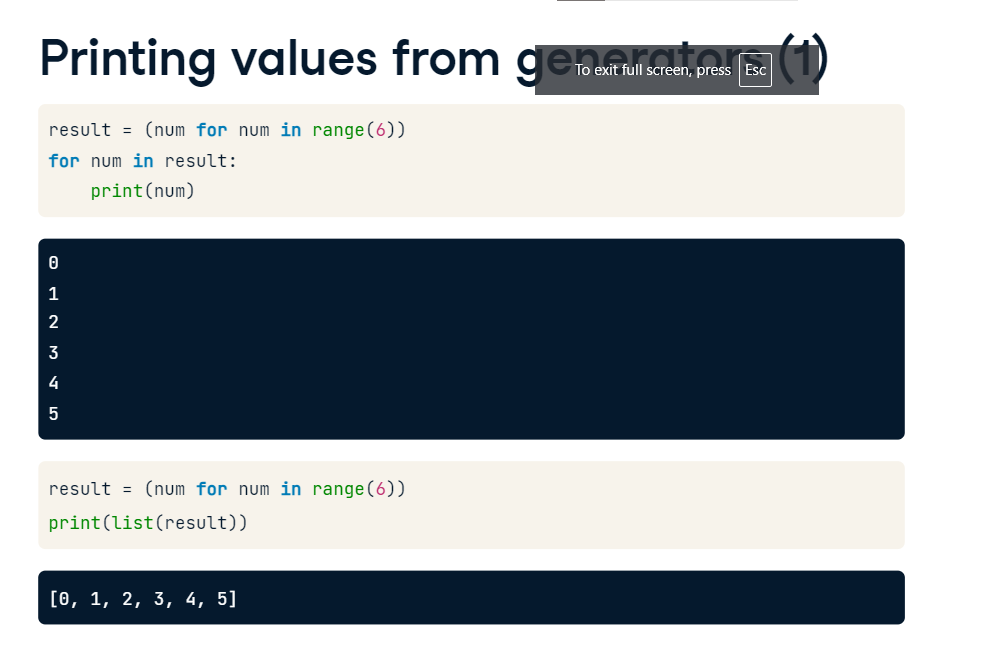
**List comprehensions vs. generators**

Now the question on everybody's lips is, "What is this generator object?" Well, a generator is like a list comprehension except it does not store the list in memory: it does not construct the list, but is an object we can iterate over to produce elements of the list as required.



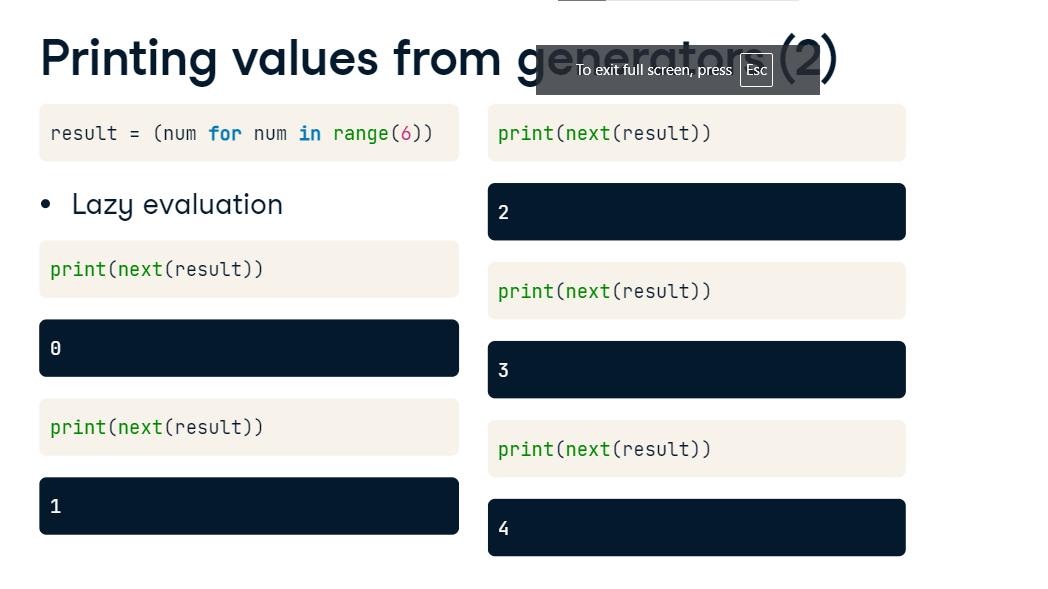
**Printing values from generators (1)**

Here we can see that looping over a generator expression produces the elements of the analogous list. We can also pass a generator to the function list to create the list. Moreover,



**Printing values from generators (2)**

like any other iterator, we can pass a generator to the function next in order to iterate through its elements. For the geeks like me, this is an example of something called lazy evaluation, whereby the evaluation of the expression is delayed until its value is needed. This can help a great deal when working with extremely large sequences as you don't want to store the entire list in memory, which is what comprehensions would do; you want to generate elements of the sequence on the fly.



**Generators vs. list comprehensions**

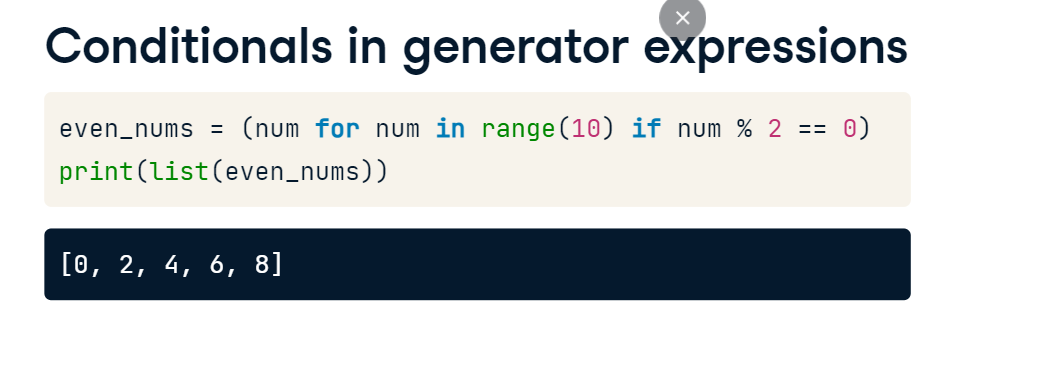
Let's say that we wanted to iterate over a very large sequence of numbers, such as from 0 up to 10 to the power of a million, or at least wanted to do so until another condition was satisfied. Look what happens when I try to build such an iterable list using a comprehension on DataCamp's servers.



**Generators vs. list comprehensions**

My colleagues disconnect me because the list I'm trying to create can't even be stored in memory! Be warned though, don't try this at home, on our servers or yours!

Check this out, however: I can easily create the analogous generator object because it does not yet create the entire list.

****

**Conditionals in generator expressions**

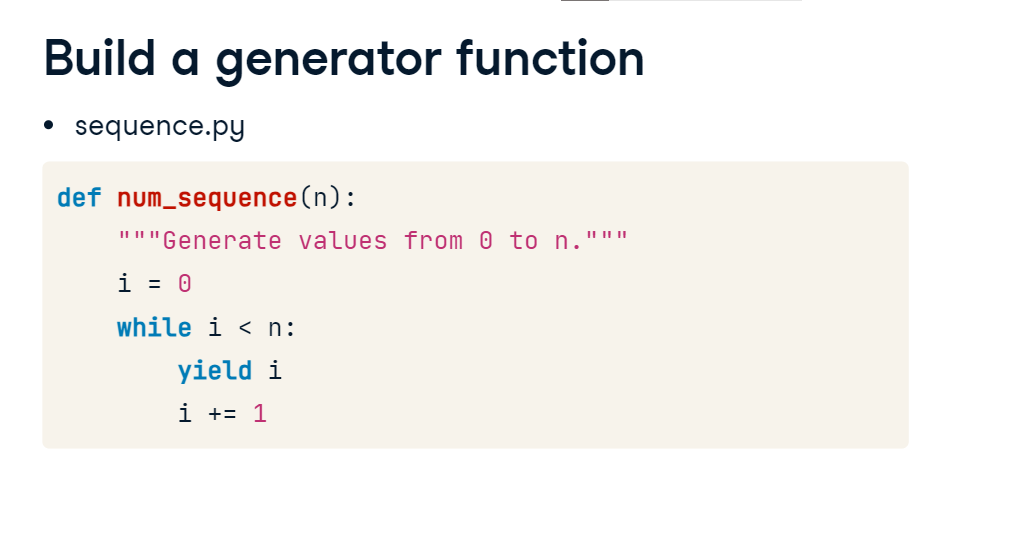
What's really cool is that anything we can do in a list comprehension such as filtering and applying conditionals, we can also do in a generator expression, such as you see here. You'll get a whole bunch of practice with this in the upcoming exercises.

**Generator functions**

The last thing to discuss before you get coding is the ability to write generator functions. Generator functions are functions that, when called, produce generator objects. Generator functions are written with the syntax of any other user-defined function, however instead of returning values using the keyword return, they yield sequences of values using the keyword yield.

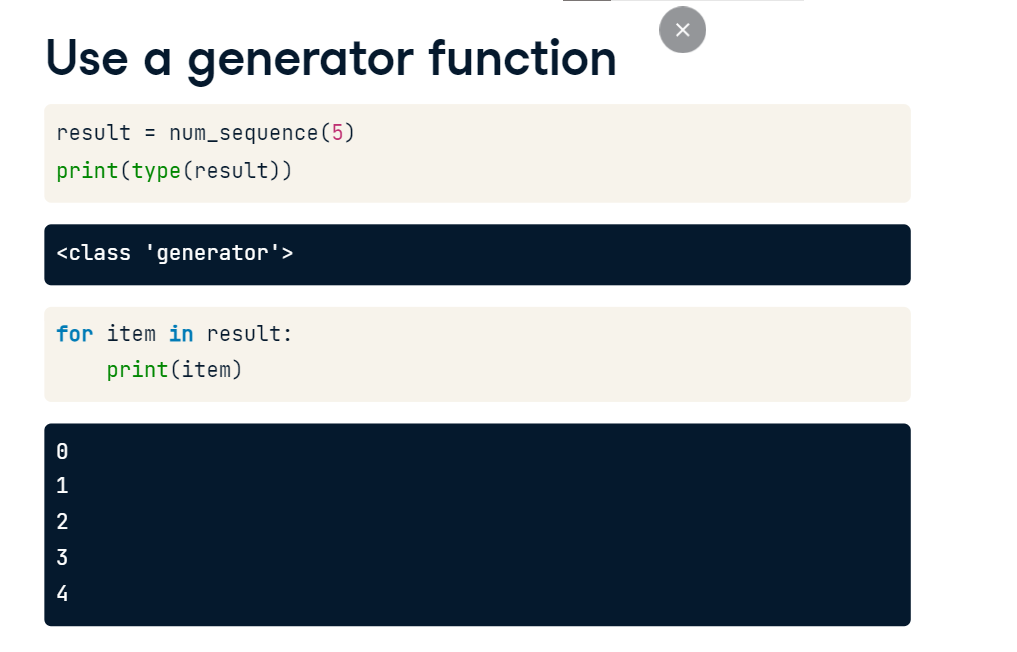
**Build a generator function**

Here I have defined a generator function that, when called with a number n, produces a generator object that generates integers 0 though n. We can see within the function definition that i is initialized to 0 and that the first time the generator object is called, it yields i equal to 0. It then adds one to i and will then yield one on the next iteration and so on. The while loop is true until i equals equals n and then the generator ceases to yield values.



**Use a generator function**

This generator function can be called as you do any other function. Here I call the generator function with the argument, 5. We see that it produces a generator object and that we can iterate over this generator object with a for loop to print the values it yields. Generator functions are a powerful and customizable way to create generators.

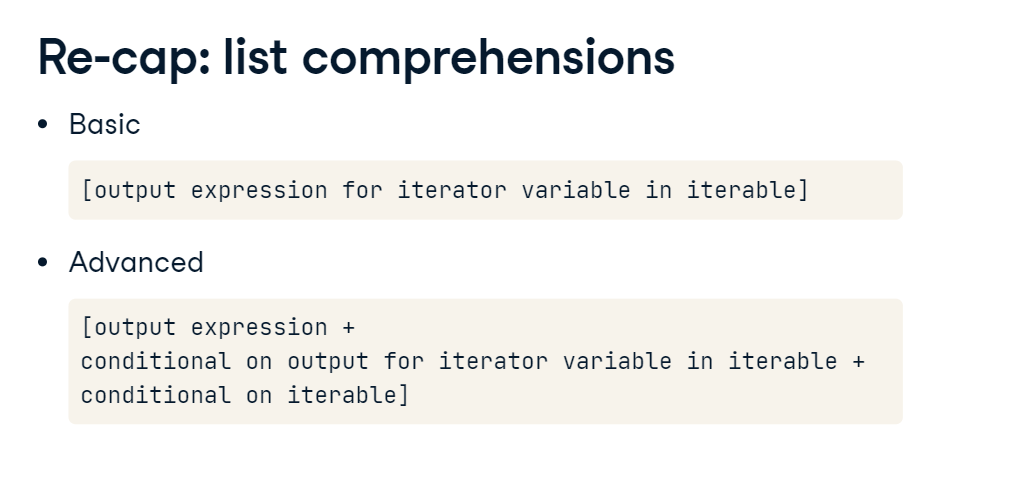


## Wrapping up comprehensions and generators.

It's now time to apply your new and hard-won skills. In the following exercises, you'll use list comprehensions to wrangle important information from the time-stamped data of real tweets. I want to remind you of the structure and syntax of a list comprehension:

**Re-cap: list comprehensions**

in its most basic form, it is enclosed in square brackets and is structured as output expression for iterator variable in iterable. More advanced comprehensions can include conditionals on the output expression and/or conditionals on the iterable.



**Let's practice!**

After you complete the upcoming exercises, I'll see you in the next Chapter. Therein, we will not teach any new material. We'll be consolidating everything that we've learned so far. This chapter will be dedicated to using the skills you're developing to extract meaningful information from a real-world dataset, the World Bank World Development Indicators dataset. Happy hacking, and see you there!