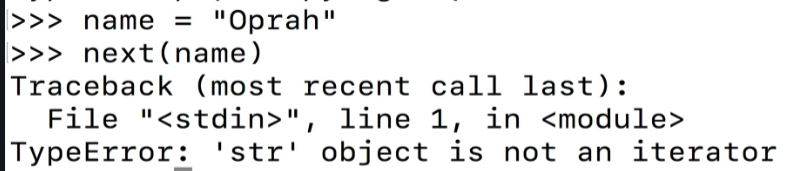
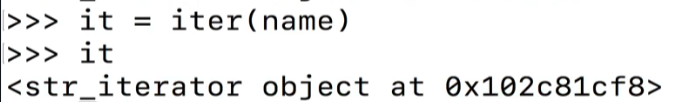
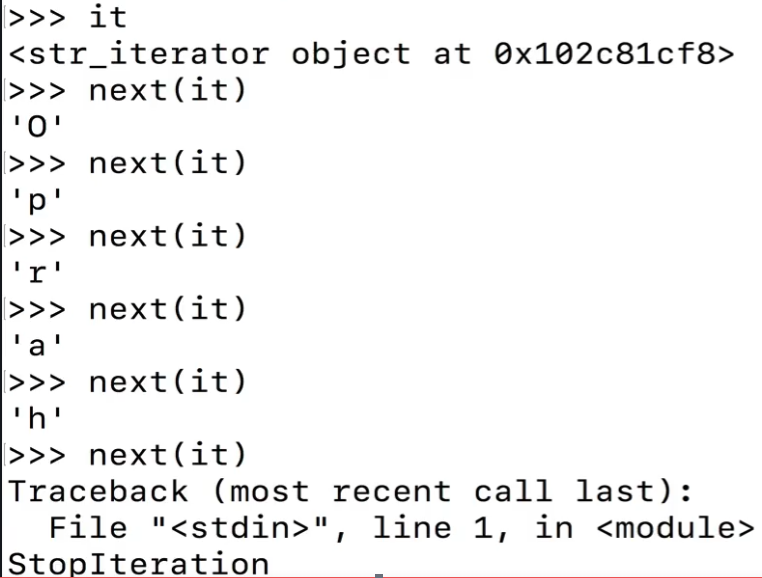
* Iterators and Iterable are NOT the same things
  + An **iterator** is an object that can be iterated *upon*. More specifically, it is an object that returns data, one element at a time when next() is called on it
  + An **iterable** is an object which will return an iterator when iter() is called on it.
    - Basically, you’ll need to convert an iterable into an iterator, then iterate upon that iterator
  + Let’s look at an example!
    - The string “HELLO”, or any string, is NOT an iterator. However it is an iterable, because we can call iter() on it, and an iterator is returned
    - The iterable is never directly “looped over”. Instead, a for loop (for example) will call the iter() function, pass in the iterable, which returns an iterator that the loop will call next() on over and over
* Code example of iterator
  + If you attempt to call next() on an iterable, it will not work because next() can only be called on an iterator



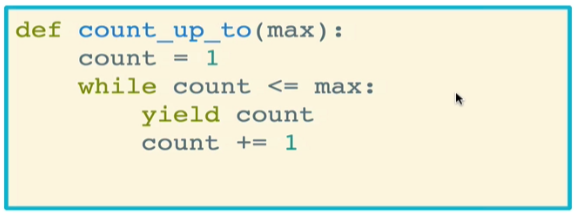
* + Calling iter() on an iterable, however, will return a str\_iterator object



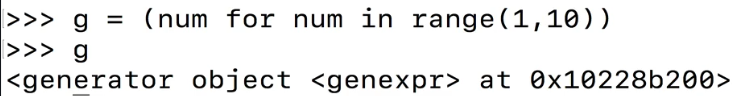
* + After the str\_iterator object is returned, you can call the next() function on it
* What is the **next()** function?
  + When next() is called on an iterator, the iterator returns the next item in the iterator. It keeps doing so, one item at a time each time it is called, until it raises a **StopIteration error** at the end of the object
  + So let’s now try calling next() on our “it” object that we defined above (the string “oprah”)

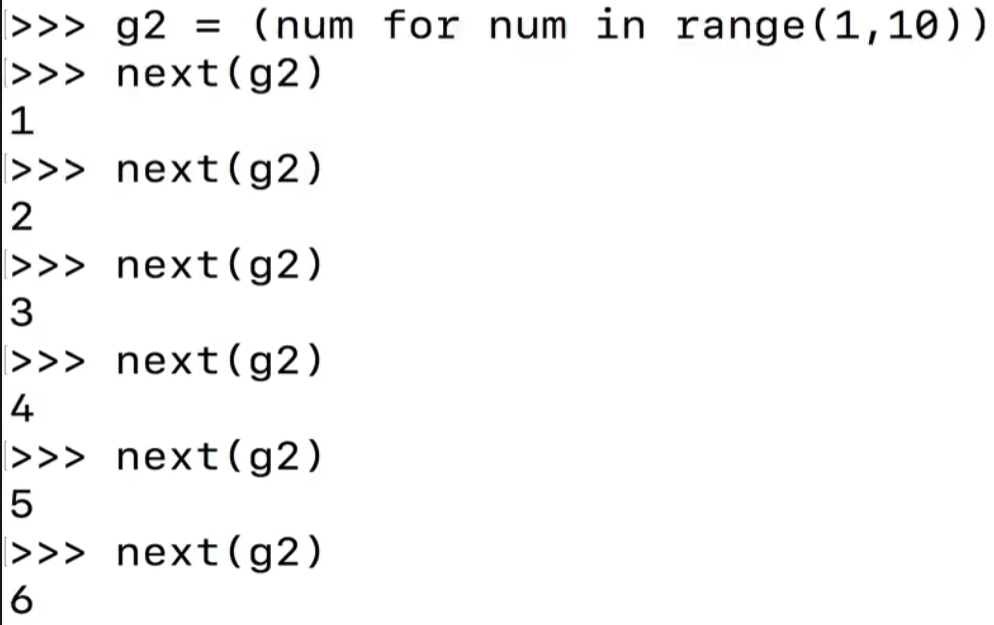


* What is a **generator?**
  + Generators are a subclass of iterators. Every generator is an iterator. NOT every iterator is a generator
  + They can be thought of as an easy quick way to create iterators
  + They can be created in two ways
    - Generator functions using the **yield** keyword
    - Using **generator expressions**
* **Generator functions** are functions that use **yield**instead of **return**
  + They also can yield more than once, as opposed to return which returns only once
  + When invoked, generator functions return a *generator*, as opposed to regular function which return a *value*
  + Example: This generator function returns a generator. As we run through the while loop, if the count is less than or equal to the max value provided, the function will return (yield) the count (as a generator object), and then PAUSE until next() is called on count\_up\_to. Once next() is called, count is incremented by 1 and the loop circles around again
    - All generator objects have a next() function associated with it
    - Note that it is NOT a class, but rather a function that we’re calling next() on



* **Generator expressions** are to generators what list comprehension is to lists
  + They even look a lot like list comprehensions
  + They use parens () instead of brackets []
  + Example: Below we create a simple generator expression. Note that the object itself is a generator object that indicates that it was created by a generator expression <genexpr>





* + We can also do things like sum()

