MODULES

ITERATORS IN PYTHON

In Python, Iterators are objects that let you access elements in Collection one at a time, in sequential order. They are like special faucets that dispense items from a Container, but instead of pouring everything out at once they just give you the next element when request it.

Key Concepts:

- **Iterables:** There are objects that you can iterate over, meaning you can use them in a for-loop to access their elements. Common iterables include lists, tuples, strings, dictionaries (for keys) and sets.
- **Iterator Protocol:** This is a Convension that defines how iterators work in Python. It Consists how of two special methods:

1) `__iter__()`: This method returns the iterator object itself. It's called behind the scenes when you use an iterable in a for loop
2) `__next__()`: This method is called each time you need the next element from the iterator. It should return the next item in the sequance or raise a stopIteration exception if there are no more elements left.

1

HOW ITERATORS WORK WITH FOR FOR LOOPS

When you use a ~~for~~ loop with an iterable, python does the following under the hood.

1) Calls `__iter__()` on the iterable. this creates an iterable object.

2) Enters the loop.

Inside the loop, Python Calls `__next__()` on the iterator to get the next element.

3) If `__next__()`:

returns an element, python assigns it to the loop variable you defined (eg. item in for item in my list.

The loop body executes using the current element.

Python goes back to step 2.b until `__next__()` raises a `StopIteration` exception. indicating the end of the iteration.

Example

```
my_list = [1, 2, 3, 4]
for item in my_list
    print(item)
```

Hard, the loop iterates over my_list iterator, printing each element (1, 2, 3, 4) one at a time.

Python Code

```
Class NumberGenerator:
    def __init__(self, start, end):
        self.start = start
        self.current = start - 1  # initialize current one before start.
    def __iter__(self):
        return self
    def __next__(self):
        self.current += 1
        if self.current > self.end:
            raise StopIteration.
        return self.current.

# Usage:
generator = NumberGenerator(1, 5)
for num in generator
    print(num)
# prints 1, 2, 3, 4, 5.
```