Video 24 : Iterables

In this video I'll cover iterables and show how you can add iterable behaviors to your classes using magic methods.

An iterable is a stored sequence of values (list) or, as we will see when we cover generators, an object that produces one value at a time.

Iterables differ from iterators in that an iterable is an object with an __iter__ method which returns an iterator. An iterator is an object with a __next__ method which retrieves the next value from sequence of values. Let's see an example.

CODE

```
# Define a string and convert it into an iterator
samp_str = iter("Sample")

print("Char :", next(samp_str))
print("Char :", next(samp_str))
```

Custom Iterable

Now I'll show how you can add iterator behavior to your custom classes.

CODE

```
class Alphabet:

    def __init__(self):
        self.letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        self.index = -1

    def __iter__(self):
        return self

    def __next__(self):
        if self.index >= len(self.letters) - 1:
            raise StopIteration
        self.index += 1
        return self.letters[self.index]

alpha = Alphabet()

for letter in alpha:
    print(letter)

# Iterate through a dictionary because it is an iterable
derek = {"fName": "Derek", "lName": "Banas"}

for key in derek:
    print(key, derek[key])
```

Python Problem for you to Solve

It's time for another problem. Create a class that returns values from the Fibonacci sequence each time next is called.

Sample Output

```
Fib : 1
Fib : 2
Fib : 3
Fib : 5
```

Solution

```
class FibGenerator:

    def __init__(self):
        self.first = 0
        self.second = 1

    def __iter__(self):
        return self

    def __next__(self):
        fib_num = self.first + self.second
        self.first = self.second
        self.second = fib_num
        return fib_num

fib_seq = FibGenerator()

for i in range(10):
    print("Fib :", next(fib_seq))
```

That's all for now. In the next video I'll cover list comprehensions, generator functions, and generator expressions.