Iterators

This lesson will discuss built-in iterators in Python, and teach you to build your own custom iterator class.

WE'LL COVER THE FOLLOWING

Iterators

As we saw previously, in Python we use the “for” loop and “while” to iterate over the contents of objects:

```
for value in [0, 1, 2, 3, 4, 5]:
    print(value)
```

Objects that can be used with a for loop are called iterators. An iterator is, therefore, an object that follows the iteration protocol.

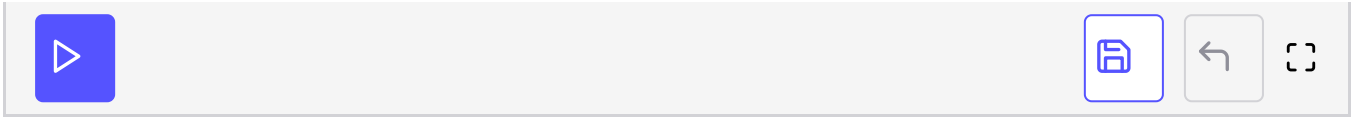
The built-in `iter` method can be used to build iterator objects, while the `next` method can be used to gradually iterate over their content:

```
my_iter = iter([1, 2, 3])
print (next(my_iter))
print (next(my_iter))
print (next(my_iter))
```

If there are no more elements, the iterator raises a “StopIteration” exception.

```
my_iter = iter([1, 2, 3])
next(my_iter)

next(my_iter)
next(my_iter)
```



Iterator Classes

Iterators can be implemented as classes; you just need to implement the `__next__` and `__iter__` methods. Here's an example of a class that mimics the `range` function, returning all values from `a` to `b`:

```
class MyRange:

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __iter__(self): # returns the iterator object itself
        return self

    def __next__(self): # returns the next item in the sequence
        if self.a < self.b:
            value = self.a
            self.a += 1
            return value
        else:
            raise StopIteration
```

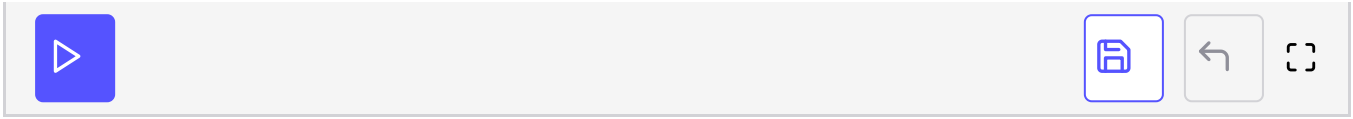
Basically, on every call to `next`, it moves forward the internal variable `a` and returns its value. When it reaches `b`, it raises the StopIteration exception. You can observe this behavior by uncommenting the last line.

```
class MyRange:
  def __init__(self, a, b):
    self.a = a
    self.b = b

  def __iter__(self):# returns the iterator object itself
    return self

  def next(self):
    if self.a < self.b:# returns the next item in the sequence
      value = self.a
      self.a += 1
      return value
    else:
      raise StopIteration
```

```
myrange = MyRange(1, 4)
print (myrange.next())
print (myrange.next())
print (myrange.next())
##print (myrange.next())
```

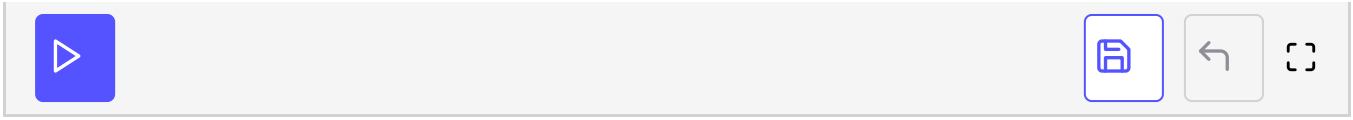


But most importantly, you can use the iterator class in a “for” loop:

```
class MyRange:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __iter__(self): # returns the iterator object itself
        return self

    def __next__(self): # returns the next item in the sequence
        if self.a < self.b:
            value = self.a
            self.a += 1
            return value
        else:
            raise StopIteration
for value in MyRange(1, 4):
     print(value)
```



Now that you know the basics, let’s solve some challenges on iterators.