Iterators

An iterator is an object that contains a countable number of values.

An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.

Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of methods `__iter__()` and `__next__()`.

Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable *containers* which you can get an iterator from.

All these objects have a `iter()` method which is used to get an iterator:

Example

Return an iterator from a tuple, and print each value:

```
mytuple = ("apple","cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
========================
apple
cherry
```

Even strings are iterable objects, and can return an iterator:

Example

Strings are also iterable objects, containing a sequence of characters:

```
mystr = "banana"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

Looping Through an Iterator

We can also use a **for** loop to iterate through an iterable object:

Example

Iterate the values of a tuple:

```
mytuple = ("apple","cherry")

for x in mytuple:
  print(x)
 ===================
apple
cherry
```

Example

Iterate the characters of a string:

```
mystr = "banana"

for x in mystr:
  print(x)
==================
b
a
n
a
n
a
```