Iterator and Iterable

- An object is **iterable** if it can be iterated over it, such as a list or an ArrayList. We can use a for loop to iterate an iterable object.
- Iterable objects has built-in iterator. An **iterator** is like a one-time clockwork toy that cannot be re-winded, it can iterate an object once; it knows what the current item is, and it can go to the next item, it can stop after visiting the last item.
  - To see an object is iterable, we can use dir() to check whether is contains the "dunder" method __iter__().
  - A **dunder** (which mean double under) method is used to extend the meaning of operator. For example, in a user developed class, if we want to use the operator "+", we need to implement the dunder method __add__(); if we want to use a for loop to print out items inside of an object that is of a user developed class, we need to implement __iter__() method in that class.

- As a class, an iterator contains the following attributes:
  - The iterable object it needs to iterate.
  - The starting and ending spots.
  - A pointer that shows the current spot (it points at the starting spot at first).

  And an iterator has at least these two methods:
  - __iter__(): return the iterator of itself.
  - __next__(): return the next item (where the pointer points at), then moves the pointer to the next spot.

Generator

- A **generator** provides similar service as an iterator, it can create each item in a sequence one by one in order.
- Both list and generator can create a sequence of items. Comparing to a list, a generator doesn't create all items at once and store them in the memory; instead, it only creates the generator itself, and it creates the next item in the sequence whenever it is needed.

- In a generator, we don't use key word "return"; instead, we use keyword "**yield**". The keyword yield converts a Python method into a generator object.
  - Yield gives an intermediate result to the caller, and code written after yield statement execute in next call.
- We can use a generator as an iterator in a class that is iterable.