For Each Loop

There's one more way to give looping instructions to a computer. We define a sequence of values and tell the computer to repeat the instructions for each item in the sequence.

```
foreach (type element in sequence)
{
  statement;
}
```

The `foreach` loop is used to iterate over collections, such as an array.

In our video game, we want to play a melody. We can do that by iterating through a list of individual notes, playing one after the other. Here's an example array of notes:

```
string[] melody = { "a", "b", "c", "c", "b" };
```

and the loop would look like:

```
foreach (string note in melody)
{
  PlayMusic(note);
}
```

The sequence we used was an array, but we can use other similar data structures. The umbrella term for those is collection, so we can also call `foreach` loops *collection loops*.

Use this loop when you need to perform a task for every item in a list, or when the order of things must be maintained. In this case, both are important. A note must be placed for each item in the list and the order of them is essential to the musical pattern.

☑Instructions

1.

Now you want to create a To Do list to keep track of your tasks. Write an empty loop that will iterate through each item in your `todo` array.

Hint

Here's an example of an empty `foreach` loop that can iterate through an item in a collection:

```
string[] melody = { "a", "b", "c", "c", "b" };

foreach (string note in melody)
{

}
```

2.
Inside of the loop, call the `CreateTodoItem()` method.

Hint

Calling a method inside of a `foreach` loop will call the method for each item in the collection:

```
string[] melody = { "a", "b", "c", "c", "b" };

foreach (string note in melody)
{
  PlayMusic(note);
}
```