This chapter introduces a new concept: data structures. A data structure is a collection of data elements (such as numbers or characters, or even other data structures) that is structured in some way, such as by numbering the elements. The most basic data structure in Python is the sequence. Each element of a sequence is assigned a number—its position, or index. The first index is zero, the second index is one, and so forth.

**Note**  When you count or number things in your daily life, you probably start counting from 1. The numbering scheme used in Python may seem odd, but it is actually quite natural. One of the reasons for this, as you see later in the chapter, is that you can also count from the end: the last item of a sequence is numbered –1, the next-to-last –2, and so forth. That means you can count forward or backward from the first element, which lies at the beginning, or 0. Trust me, you get used to it.

This chapter begins with an overview of sequences, and then covers some operations that are common to all sequences, including lists and tuples. These operations will also work with strings, which will be used in some of the examples, although for a full treatment of string operations, you have to wait until the next chapter.

After dealing with these basics, we start working with lists and see what’s special about them. After lists, we come to tuples, which are very similar to lists, except that you can’t change them.

### Sequence Overview

Python has six built-in types of sequences. This chapter concentrates on two of the most common ones: lists and tuples. The other built-in sequence types are strings (which I revisit in the next chapter), Unicode strings, buffer objects, and xrange objects.

The main difference between lists and tuples is that you can change a list, but you can’t change a tuple. This means a list might be useful if you need to add elements as you go along, while a tuple can be useful if, for some reason, you can’t allow the sequence to change. Reasons for the latter are usually rather technical, having to do with how things work internally in Python. That’s why you may see built-in functions returning tuples. For your own programs, chances are you can use lists instead of tuples in almost all circumstances. (One notable exception, as described in Chapter 4, is using tuples as dictionary keys. There lists aren’t allowed, because you aren’t allowed to modify keys.)
Sequences are useful when you want to work with a collection of values. You might have a sequence representing a person in a database, with the first element being their name, and the second their age. Written as a list (the items of a list are separated by commas and enclosed in square brackets), that would look like this:

```python
>>> edward = ['Edward Gumby', 42]
```

But sequences can contain other sequences, too, so you could make a list of such persons, which would be your database:

```python
>>> edward = ['Edward Gumby', 42]
>>> john = ['John Smith', 50]
>>> database = [edward, john]
>>> database
[['Edward Gumby', 42], ['John Smith', 50]]
```

---

**Note** Python has a basic notion of a kind of data structure called a *container*, which is basically any object that can contain other objects. The two main kinds of containers are sequences (such as lists and tuples) and mappings (such as dictionaries). While the elements of a sequence are numbered, each element in a mapping has a name (also called a key). You learn more about mappings in Chapter 4. For an example of a container type that is neither a sequence nor a mapping, see the discussion of sets in Chapter 10.

---

**Common Sequence Operations**

There are certain things you can do with all sequence types. These operations include *indexing*, *slicing*, *adding*, *multiplying*, and checking for *membership*. In addition, Python has built-in functions for finding the length of a sequence, and for finding its largest and smallest elements.

---

**Note** One important operation not covered here is *iteration*. To iterate over a sequence means to perform certain actions repeatedly, once per element in the sequence. To learn more about this, see the section “Loops” in Chapter 5.

---

**Indexing**

All elements in a sequence are numbered—from zero and upwards. You can access them individually with a number, like this:

```python
>>> greeting = 'Hello'
>>> greeting[0]
'H'
```