Appendix G

Lists in Brief

G.1 About Lists

In the book's early chapters, each value and variable represented a single data item, for example a single number or a single string. In most programs, there is a need to keep track of related items together. In this appendix we'll look briefly at Python lists, which is a way to keep track of an ordered set of items. Chapter 6 explores lists in more detail.

Lists have many features. In this appendix we'll focus on how to create a list in a statement, and how to access one item of a list. We'll also see how you can ask certain questions of a list, for example how many items a list has, or whether a certain value is contained in a list. Another powerful feature of Python lists is the ability to get subsets using ***slicing***.

G.2 Creating Lists

There are two primary ways to create a list. You can define a list and its items in one statement - we'll cover this approach in this appendix. An alternative approach is to define an empty list and then add items to it – this is covered in section 6.2.

Square brackets are used in several ways to work with lists. For example, to define a list of the primary colors of light, you can use the following statement:

```
colors = ['red', 'green', 'blue']
```

This defines the variable `colors`, which points to the ordered set of values `'red'`, `'green'`, and `'blue'`.

Many list features can be done in a single statement. This lends itself to trying them interactively in the Python shell. For example, you can enter the above statement, and then enter the variable `colors` to see the list with its values.

```
>>> colors = ['red', 'green', 'blue']
>>> colors
['red', 'green', 'blue']
```

Python list items can be strings, numbers, or in fact any Python type. For example, the following statement defines a list of numbers.

```
>>> squares = [1, 4, 9, 16]
```


Python is very flexible with types. For example, a list could contain different types of items:

```
>>> mixedList = ['hello', 123, 45.6, True]
```

The above list is a heterogeneous list. For this appendix, we'll use homogenous lists that contain a common type – this is perhaps the most common use of lists.

In order to refer to one item of a list, you also use square brackets. Specifically, the combination of the name of the list, square brackets, and an integer value, variable, or expression inside the square brackets refers to a particular item of the list. Further, the first item in the list is identified as item 0.

```
>>> colors = ['red', 'green', 'blue']
>>> colors[0]
'red'
```

The numbered location of a list item is called its ***index***. In the above example, `colors[0]` refers to the item in the colors list at index 0. The first index of all Python lists is 0. Figure G-1 shows a diagram view of the colors list. Python also allows for negative indexes, which counts from the end of the list. For example, `colors[-1]` refers to `'blue'`.

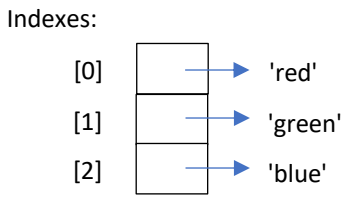

Figure G-1: List of colors

G.3 Getting part of a list with slicing

In this section we look at how to generate a list with some items from a source list. Python provides a very concise and powerful way to do this called ***slicing***.

Slicing involves using a list with square brackets `[]` as well as one or two colons. In one variation you specify the index location of the initial item of interest, a colon, and a second location, as in:

```
alist[start:end]
```

For example,


```
>>> alist = [1, 4, 9, 16, 25]
>>> alist[0:3]
[1, 4, 9]
```

Note that slicing returns a list starting at the first item, and going up to *but not including* the index of the last item. In the above statements, `alist[0:3]` returns a list with the items at index locations 0, 1 and 2.

Slicing allows for an optional third argument, indicating how to skip items. Continuing the statements above,

```
>>> alist[0:4:2]
[1, 9]
```

You can omit the start argument for the slice, in which case the slice starts at item 0.

```
>>> alist[:2]
[1, 4]
```

Similarly, you can omit the end argument for the slice - in this case the result is as if you specified the length of the list as the end argument.

```
>>> alist[3:]
[16, 25]
```

G.4 Asking questions of a list

Certain built-in functions can be used to ask questions about lists. For example, the `len()` function determines the number of items in a list. Other functions useful for lists include `min()`, which returns the smallest value in the list, and `max()` which returns the largest value. Note that `min()` and `max()` will give an error if the argument is an empty list.

Python provides the `in` keyword, which can determine if a value is in a list.
Note that `in` can be used together with the `not` keyword, to check if a value isn't in a list.

G.5 About tuples – unchangeable lists

We close out this appendix with this brief section about ***tuples***. Tuples are like lists except their items can't be changed once the tuple is created. This can help make a program more efficient in case you have an unchangeable ordered set of items. Tuples can be created using parentheses (as opposed to square brackets for lists). For example, the following statement:

```
>>> colors = ('Maroon', 'Purple', 'Red', 'Orange', 'Yellow', 'Green')
```

creates a tuple of 6 strings.

You can verify the unchangeable nature of tuples. If you interactively enter the above line, then try to change an item using:

```
>>> colors[0] = 'Brown'
```

you will get:

```
TypeError: 'tuple' object does not support item assignment
```

