**Chapter 5 Data Structures**

This chapter will cover ***data structures***. Data structures, or ***structured variables***, are used to store more than one value. These are also called ***compound types***. We have already seen two types of data structures, **string**, which store multiple characters, and lists. Recall that strings and lists are examples of a particular type of data structure called a ***sequence***. With sequences, the values are put in an order and can be indexed using integer indices. Another simple type of sequence data structure, which is used very commonly in Python, is a ***tuple***. We will also cover more detail on the **range** function, as well as **enumerate**.

**5.1 Strings**

We have seen that strings in Python are text, and can be stored in either single quotes or double quotes. Strings are an example of a particular kind of data type called a ***sequence***. Sequences in general are collections of values (for strings, collections of characters) that can be indexed using integers. For strings, indexing returns the characters in the string.

We have seen indexing into a string starting at the beginning uses positive integer indices, beginning with 0. Negative indices can be used to index into a string from the end of a string. Using an index of -1 returns the last character in a string, regardless of how long it is.

*>>> myword = 'hello'*

*>>> myword[-1]*

'o'

For this string, with a length of 5, an index of -5 will return the first character.

*>>> myword = 'hello'*

*>>> myword[-5]*

'h'

So, note that with a string that has a length of 5, the positive indices go from 0 through 4, and the negative indices go from -1 through -5 (in reverse order).

Strings can also be ***sliced***. A slice of a string is a subset of the characters in the string. This is also sometimes called a ***substring***. The colon operator is used in a variety of formats to create string slices. The colon operator specifies the beginning index (which is included in the string that is returned), and the ending index (which is not included).

The following example returns the characters in indices 0 and 1:

*>>> myword = 'hello'*

*>>> myword[0:2]*

'he'

The following example only returns the character in index 2 (which is the second from the end):

*>>> myword = 'help'*

*>>> myword[-2:-1]*

'l'

The starting and/or ending indices can also be omitted. If the starting index is not specified, the default is 0 (the first index), and if the ending index is not specified, the default is the length of the string.

The following example is equivalent to *myword[3:5]* so it returns the characters in indices 3 and 4:

*>>> myword = 'hello'*

*>>> myword[3:]*

'lo'

The following example is equivalent to *myword[0:1]* so it returns the character in index 0:

*>>> myword = 'hello'*

*>>> myword[:1]*

'h'

This means that just using a colon, without specifying any indices, returns a slice consisting of the entire string.

*>>> myword = 'hello'*

*>>> myword[:]*

'hello'

When indexing into a string, a second colon can be used, and the third value indicates a step. For example, the following specifies a slice of a string beginning at index 3, through index 9 (since 10 is specified), in steps of 2, which means every other character, beginning with the fourth.

*>>> mystr = 'helloabcde'*

*>>> print(mystr[3:10:2])*

lace

Using a negative step means indexing into the string in reverse order, so the first index specified should be larger than the second.

*>>> mystr = 'helloabcde'*

*>>> print(mystr[7:2:-2])*

cal

Just specifying the step of 2 will return every other character, from the beginning:

*>>> mystr = 'helloabcde'*

*>>> mystr[::2]*

'hlobd'

Only specifying a step of -1 will reverse a string.

*>>> mystr = 'helloabcde'*

*>>> mystr[::-1]*

'edcbaolleh'

We have seen the concatenation operator +, which concatenates strings together. Notice that concatenating slices obtained by using the same number, first for the ending index (omitting the starting index), and then for the starting index (omitting the ending index), results in the entire string. The following concatenates ‘hel’ with ‘lo’:

*>>> myword = 'hello'*

*>>> myword[:3] + myword[3:]*

'hello'

To concatenate a string with itself n times, where n is an integer, the ‘\*’ operator is used.

*>>> news = 'cde'*

*>>> news\*3*

'cdecdecde'

Two functions, **ord** and **chr**, can be used to convert a character to its integer equivalent, and to convert an integer to its character equivalent, respectively. The function **ord** returns the Unicode code for a character, and **chr** returns a string consisting of one character:

*>>> ord('a')*

97

*>>> chr(97)*

'a'

*>>> ord('b')*

98

Although the letters of the alphabet are in sequence, there are other characters in the Unicode encoding sequence before the letters start. Also, the upper case letters are together in the encoding sequence, and the lower case letters are together, but there are other characters in between the upper and lower case letters.

*>>> ord('A')*

65

*>>> ord('B')*

66

*>>> ord('a') - ord('Z')*

7

We have seen that the **str** type has methods, including **index**.

Using **index**, and slicing, we can break a string into pieces. The following indexes into the name to find the location of the space between the first and last names, and then slices up to that location to get just the first name.

*>>> urname = 'monty python'*

*>>> wherespace = urname.index(' ')*

*>>> firstname = urname[:wherespace]*

*>>> firstname*

'monty'

Since the **input** function returns a string, text processing must be used to split the string into multiple values if multiple inputs are desired. For example, the following prompts the user for the length and width of a rectangle. It finds the location of the space in between them, and separates the string into the two values. It then converts both to numbers and calculates and prints the area of the rectangle.

*print('When prompted, enter the length and width of a rectangle')*

*lenwid = input('Separate with a space: ')*

*wherespace = lenwid.index(' ')*

*len = float(lenwid[:wherespace])*

*wid = float(lenwid[wherespace+1:])*

*print("Area = ", round(len \* wid, 2))*

When prompted, enter the length and width of a rectangle

Separate with a space: 3.1 4.4

Area = 13.64

**5.2 Lists**

Lists are sequences of values. Simple lists are created by putting values in square brackets, separated by commas. As we have seen, the values in lists can be different types, although it is more common for them to be the same type.

The **len** function returns the number of elements in a list..

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> len(numlist)*

5

An ***empty list*** can be created using square brackets with nothing inside.

*>>> el = []*

*>>> len(el)*

0

Since lists are a sequence type, they can be indexed using integers and they can be sliced.

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> numlist[0]*

4

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> numlist[-2:]*

[11, -3]

The **del** statement can delete item(s) from a list. This will actually modify the list variable.

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> del numlist[1]*

*>>> numlist*

[4, 33, 11, -3]

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> del numlist[1:3]*

*>>> numlist*

[4, 11, -3]

Assigning the empty list to item(s) is another way to delete items (besides using **del**).

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> numlist[0:2] = []*

*>>> numlist*

[33, 11, -3]

Since lists are mutable, items can be modified by either indexing or slicing and assigning new value(s).

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> numlist[1] = 99*

*>>> print(numlist)*

[4, 99, 33, 11, -3]

*>>> strlist = ['hi', 'hello', 'ciao']*

*>>> strlist[0] = 'howdy'*

*>>> print(strlist)*

['howdy', 'hello', 'ciao']

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> numlist[3:] = [7, 8, 9]*

*>>> print(numlist)*

[4, 52, 33, 7, 8, 9]

Notice that the number of items being replaced does not have to be the same as the number of items replacing them.

An entire list can be obtained using the slice operator, as in:

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> numlist[:]*

[4, 52, 33, 11, -3]

However, caution should be used with this because in some contexts, this is different from just referring to the list variable. We will see more on this in Section 6.1.

Lists of characters can be created from strings using the **list** function.

*>>> list("monty")*

['m', 'o', 'n', 't', 'y']

The **choice** function, which is in the **random** module, can be used to choose a random item from a list.

*>>> from random import choice*

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> choice(numlist)*

33

The **shuffle** function randomly organizes the items in a sequence. For example, the shuffle function is used to modify the order of a list in the following.

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> from random import shuffle*

*>>> shuffle(numlist)*

*>>> numlist*

[33, -3, 52, 11, 4]

Operators and functions can be used on lists. For example, lists can be concatenated using the + operator.

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> newnums = numlist + [25]*

*>>> print(newnums)*

[4, 52, 33, 11, -3, 25]

The concatenation operator can only concatenate one list to another (not an individual value to a list), which is why the 25 is in square brackets.

The **in** operator will determine whether or not a value is in a list, and return **True** if it is or **False** if it is not.

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> 33 in numlist*

True

Statistical functions can be called to perform operations on lists such as **min** for the minimum value in a list and **max** for the maximum value**.**

*>>> min(numlist)*

-3

*>>> max(numlist)*

52

There is also a built-in function **sum** that will sum the numbers in a list of numbers:

*>>> numbers = [5, 11, 4]*

*>>> sum(numbers)*

20

An error will occur if the list contains values, for example strings, that cannot be added using **sum**.

*>>> slist = ['hi', 'hello', 'ciao']*

*>>> sum(slist)*

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Like strings, lists have an **index** method that will return the index of a specified value. For example, to find the location in the list of the largest value:

*>>> numlist = [4, 52, 33, 11, -3]*

*>>> numlist.index(max(numlist))*

1