**Chapter 4 Loops**

Loops are statements that repeat other statements (called the ***action***). **For** loops are generally used as ***counted loops***, in which the number of repetitions is specified ahead of time. In Python, **for** loops are frequently used to perform the same operation(s) on everything stored in a sequence. **While** loops are ***conditional loops***, which repeat statements as long as an expression is **True**, or until something happens. ***Nested loops*** will also be covered; with nested loops, one loop is in the action of another.

**4.1 For loops**

**4.1.1 Loops using range**

**For** loops are used to repeat other statements (called the ***action***) a specified number of times.

The easiest way to specify how many times is to use the **range** function. The **range** function generates a sequence of numbers. Called with an integer *n*, **range(n)** creates a sequence of *n* integers. By using an ***iterator variable*** and the **in** operator along with the **range** function, we can specify how many times to repeat an action. The general form to specify repeating an action *n* times is:

for itervar in range(n):

action

# rest of code

This says to repeat the action *n* times. The iterator variable iterates through all of the numbers generated by the **range** function, and for each value of the iterator variable, the action is executed. The action can be any number of valid statements. All of the statements in the action must be indented to the same level (typically 4 spaces). For historical reasons, the iterator variable (*itervar*) is frequently named *i*.

For example, the following **for** loop specifies repeating the action of printing a single ‘!’ 5 times.

| *for i in range(5):*  *print('!', end='') print('\nOK')* |
| --- |

This produces the output:

!!!!!

OK

The iterator variable *i* iterates through the 5 values produced by the **range** function, and for each of those five values it prints a single exclamation point, on the same line. Specifying end = ‘ ’ keeps all of the exclamation points on the same line, with nothing printed in between them. After the loop, printing the newline character moves down to the next line, where ‘OK’ is printed.

Note that in this example, the iterator variable *i* was not used in the action; it simply specified how many times to execute the action. In some cases, however, using the value of the iterator variable in the action is desired.

The sequence that is produced by **range** begins with 0, and ends at n-1. For example, range(4) creates a sequence of 4 integers: 0, 1, 2, 3.

The following loop prints the integers 0 to 3 in a column, and then ‘OK’ after the loop.

| *for i in range(4):*  *print(i)*  *print('OK')* |
| --- |

0

1

2

3

OK

In this example, the iterator variable *i* iterates through the values in the sequence produced by the **range** function. For every value of the iterator variable, the action is executed. The action in this case was to print the value of *i* (and the newline character, which print does by default). So, when *i* had the value 0, 0 was printed. Then, the iterator variable *i* got the value of 1, and 1

was printed. Then, *i* got the value of 2 and 2 was printed. Then, *i* got the value of 3 and 3 was printed. Once the iterator variable has iterated through all of the values in the sequence, the loop is over. Once the action was repeated 4 times, for every value in the sequence 0, 1, 2, 3, the next statement in the code after the loop was executed, which printed ‘OK’.

The following prints the integers 0 to 2, each of which is followed by a colon, space, and ‘!’.

| *for num in range(3): print(f'{num}: !')* |
| --- |

0: !

1: !

2: !

Although frequently named *i*, the iterator variable can have any name. In this case, the iterator variable *num* iterated through the values 0, 1, and 2. The action that was executed for each value of *num* was to print *num* in a formatted line including a colon and exclamation mark.

So, again, sometimes the iterator variable is just used to specify how many times to repeat an action, as in:

| for i in range(3): print('\*') |
| --- |

\*

\*

\*

Sometimes the value of the iterator variable is used in the action, as in:

| for i in range(3):  print(i) |
| --- |

0

1

2

**4.1.2 Looping through sequences**

A **for** loop can be used in Python to iterate through the items in any sequence, and perform the same action for each one. The general form is

for itervar in sequence:

action

# rest of code

There is a variable, *itervar*, that iterates through all of the items in the sequence in order. For each item, the action is executed.

The following loops through all of the items in the list *somenums* and prints each in a sentence.

| *somenums = [4, 33, 11]*  *for n in somenums:*  *print('The number is', n)* |
| --- |

The number is 4

The number is 33

The number is 11

At the beginning of the loop, the iterator variable *n* gets the value of the first number in the list, which is 4, so the action prints ‘The number is 4’. Then, after the action has been executed in its entirety (in this case, it is just one statement), the iterator variable gets the next number in the list, 33. This is printed, and then *n* gets the value 11, which is printed. Once the action has been executed for all of the numbers in the list, the loop ends.

After the loop, the iterator variable *n* stores the last item from the list.

| *somenums = [4, 33, 11]*  *for n in somenums:*  *print('The number is', n) print('n is ', n)* |
| --- |

The number is 4

The number is 33

The number is 11

n is 11

The list does not need to be stored in a variable. The following will produce identical results.

| *for n in [4, 33, 11]:*  *print('The number is', n) print('n is ', n)* |
| --- |

The number is 4

The number is 33

The number is 11

n is 11

Another example illustrates looping through a list that stores different types of items, and displaying the type of each:

| *mixedlist = [33, 'hi', False] for m in mixedlist:*  *print(type(m))* |
| --- |

<class 'int'>

<class 'str'>

<class 'bool'>

Since strings are sequences, **for** loops can iterate through the characters in a string. For example, the following prints each character in a string followed by a space:

| *myword = 'hello'*  *for c in myword:*  *print(c, end = ' ') print()* |
| --- |

h e l l o

The last **print()** statement is used to move down to the next line (to print a newline).

Using the **choice** function, the following code uses a **for** loop to repeat the process of choosing a random item from a list 8 times, and prints each one.

| *from random import choice numlist = [4, 52, 33, 11, -3] for n in range(8):*  *print(choice(numlist))* |
| --- |

4

11

52

33

52

-3

33

11

To print 5 random integers in the range from 0 to 99 inclusive, we can first create a list and then as above choose random items from the list.

| *from random import choice nums = list(range(100)) for n in range(5):*  *print(choice(nums))* |
| --- |

10

42

60

54

12

This also works with just the **range** function; the list is not necessary.

| *from random import choice nums = range(100)*  *for n in range(5):*  *print(choice(nums))* |
| --- |

0

41

1

13

71

**4.1.3 Calculating Running Sums**

A useful application of a loop is to calculate a ***running sum***. A running sum typically starts at 0, and then numbers are added to the sum one at a time. For example, the sum 0+1+2+3+4+5 would start at 0, then 0+1 which is 1, then 1+2 which is 3, then 3 + 3 which is 6, then 6 + 4 which is 10 and finally 10 + 5 which is 15.

The following code accomplishes this, using a running sum variable *runsum*, and then printing the overall sum.

| *runsum = 0*  *for i in range(6):*  *runsum = runsum + i print('The sum is', runsum)* |
| --- |

The sum is 15

As the iterator variable *i* iterates through the values 0 through 5, each is added to the result that has already been stored in *runsum*. The action of the loop is simply to add to the running sum. Printing is only done after the loop, when the overall sum has been calculated.

In another example, the user is prompted for 4 numbers. Each of the numbers that the user enters is added to a running sum.

| *runsum = 0*  *for i in range(4):*  *num = float(input('Enter a number: ')) runsum = runsum + num*  *print('The sum is', runsum)* |
| --- |

Enter a number: 33

Enter a number: 4.5

Enter a number: -5

Enter a number: 2.8

The sum is 35.3

In this case, instead of adding the iterator variable *i* to the running sum, the numbers that the user enters into the variable *num* are added to the running sum.

A running sum can also be calculated from the numbers in a list.

| *runsum = 0*  *for n in [4, 33, 11]:*  *runsum = runsum + n*  *print('The sum is', runsum)* |
| --- |

The sum is 48