A dictionary is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type. You can think of a dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps to a value. The association of a key and a value is called a key-value pair or sometimes an item. As an example, we’ll build a dictionary that maps from English to Spanish words, so the keys and the values are all strings. The function dict creates a new dictionary with no items. Because dict is the name of a built-in function, you should avoid using it as a variable name.

The curly brackets, {}, represent an empty dictionary. To add items to the dictionary, you can use square brackets:

>>> eng2sp['one'] = 'uno'

Suppose you are given a string and you want to count how many times each letter appears. There are several ways you could do it: 1. You could create 26 variables, one for each letter of the alphabet. Then you could traverse the string and, for each character, increment the corresponding counter, probably using a chained conditional. 2. You could create a list with 26 elements. Then you could convert each character to a number (using the built-in function ord), use the number as an index into the list, and increment the appropriate counter. 3. You could create a dictionary with characters as keys and counters as the corresponding values. The first time you see a character, you would add an item to the dictionary. After that you would increment the value of an existing item. Each of these options performs the same computation, but each of them implements that computation in a different way. An implementation is a way of performing a computation; some implementations are better than others. For example, an advantage of the dictionary implementation is that we don’t have to know ahead of time which letters appear in the string and we only have to make room for the letters that do appear.

One of the common uses of a dictionary is to count the occurrence of words in a file with some written text. Let’s start with a very simple file of words taken from the text of Romeo and Juliet. For the first set of examples, we will use a shortened and simplified version of the text with no punctuation. Later we will work with the text of the scene with punctuation included. But soft what light through yonder window breaks It is the east and Juliet is the sun Arise fair sun and kill the envious moon Who is already sick and pale with grief We will write a Python program to read through the lines of the file, break each line into a list of words, and then loop through each of the words in the line and count each word using a dictionary. You will see that we have two for loops. The outer loop is reading the lines of the file and the inner loop is iterating through each of the words on that particular line. This is an example of a pattern called nested loops because one of the loops is the outer loop and the other loop is the inner loop. Because the inner loop executes all of its iterations each time the outer loop makes a single iteration, we think of the inner loop as iterating “more quickly” and the outer loop as iterating more slowly. The combination of the two nested loops ensures that we will count every word on every line of the input file.

dictionary A mapping from a set of keys to their corresponding values. hashtable The algorithm used to implement Python dictionaries. hash function A function used by a hashtable to compute the location for a key. histogram A set of counters. implementation A way of performing a computation. item Another name for a key-value pair. key An object that appears in a dictionary as the first part of a key-value pair. key-value pair The representation of the mapping from a key to a value. lookup A dictionary operation that takes a key and finds the corresponding value. nested loops When there are one or more loops “inside” of another loop. The inner loop runs to completion each time the outer loop runs once. value An object that appears in a dictionary as the second part of a key-value pair. This is more specific than our previous use of the word “value”.