# Study Unit 2 Data Types and Functions

## Learning Outcomes

By the end of this unit, you should be able to:

1. Differentiate the various types of compound data structures in Python
2. Discuss how Python manages packages, modules, functions, etc.

## Overview

Python provides numerous compound data types to group a collection of values together. Compound data structures organise and store data in a way that they can be accessed and worked with efficiently. These structures also define the relationship between the data and the operations that can be performed on them. We will learn in this study unit how to create, when to use, and what operations to perform on the most common Python compound data structures: tuples, lists and dictionaries. Furthermore, we will also learn about what functions and methods are and how they can be integrated in our program. Lastly, we will also deal with Python packages and modules which are, together with functions and methods, very useful for reusing and extending our tools of performing specific tasks in Python programming.

## Chapter 1 Tuples, Lists, Dictionaries

### Tuples

Lesson Recording - Python Tuples

#### 1.1.1 Defining Tuples

In Study Unit 1, we have learned that we could store values in a variable for later operations in our code. The type of a variable depends on whether we want to store numeric values or character strings in it. More often, we work with multiple data points of the same nature such as the names of all students in a class. It would be inconvenient to create a new python variable for each data point like, e.g., student1, student2, etc. What we can do instead is to store all data points in some kind of compound data structure.

Python facilitates several types of compound data for our use. One of these data types is the tuple. A tuple is a collection of values written as comma-separated items between a pair of round brackets, similar like vectors in mathematics. Unlike vectors, tuples are not specifically designed for mathematical operations. The items, or elements in a tuple can be numeric, string, or a mixture of both.

|  |
| --- |
| tuple\_name = (element1, element2, …) |

Note that tuples are immutable, that is, we are not allowed to modify them once they are defined. Nevertheless, they are more efficient in terms of performance and memory use. Tuples are useful in situations where we want to share the data with other users without granting them the permission to edit the content.

|  |
| --- |
| **Example (Students’ score, cont’d):** We assign the names of two students to a tuple called names, the scores of the same students to a tuple scores and a mixture of the names and the scores to all\_data.      Figure 2.1 Definition of Tuples  We can assign elements to a tuple with or without writing them in parentheses in our code, as long as they are separated by commas. Nevertheless, when printing them on the screen, they will be wrapped by a pair of round brackets. Furthermore, the tuple all\_data, which consists of two strings and two integers, shows us that tuples can be indeed a mixture of strings, integers, and floats. |

#### 1.1.2 Subsetting Tuples and Indexing

To access one specific element in a tuple, we need to use the index operator []. In Python, the index of any compound data type, i.e., tuples, lists and dictionaries, begins with 0 and ends with the total number of elements minus 1. In other words, the index for the first element in a tuple is 0, the index for the second element is 1, and the third one is 2, and so on. If we want to subset more than a single element from a tuple, we can put the start and end indices in the index operator, connecting them with a colon.

|  |
| --- |
| tuple\_subset = tuple\_name[start:end] |

It is important to recall that the index end will *not* be included in the subsetting procedure.

|  |
| --- |
| **Example (Cont’d):** We would now like to extract the first element from the tuple names and store it in a variable called name1, the second element in name2, as well as the entire data of the first student including his name and score from the tuple all\_data in a new tuple called student1\_data.      Figure 2.2 Subsetting Tuples  If we subset a single element from a tuple, it will become a variable of the type that corresponds to the type of the extracted data. In this case, the variables name1 and name2 are strings since “Peter” and “Mary” are stored as string in the original tuple names. And if we subset multiple elements from a tuple, the result will be a tuple as well and the data type of each element will also be taken over from the original tuple.  In the above example, the tuple student1\_data should be a subset of the tuple all\_data with the element indexed 0:2, where the index 2 is excluded. In other word, it only contains the elements 0 and 1, which are the first and second elements: Peter and 72. |

We can also use negative indices to access the elements of a tuple starting from the last element. That is, the index -1 indicates the last element, -2 the second last, etc.

|  |
| --- |
| **Example (Cont’d):** We would now like to extract the last element from the tuple names and store it in a variable called name1, the second last element in name2, as well as the entire data of the second student including his name and score from the tuple all\_data in a new tuple called student2\_data.      Figure 2.3 Subsetting of Tuples with Negative Indices  Everything seems alright in the above output except that the last element of the tuple student2\_data, the value 86, is missing. The reason is that the last index is never included. In our example, the subsetting indices we wish to have are -2 and  -1. But since Python does not include -1, we only receive student2\_data[-2], which is “Mary” in this case. To overcome this dilemma, we need to leave the end index blank after the colon. Python will interpret it as “take all indices until the end”.      Figure 2.4 Subsetting Tuples with "Open End" Indexing  This is the correct output that we want to obtain originally. Note that the “open end” indexing also works for positive indices. |

#### 1.1.3 Concatenating Tuples

Though we can access the tuple elements by using the index operator, we are neither allowed to change the values of it nor to add a new value to an existing tuple.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to change the first element in the tuple names from “Peter” to “John”, we will receive an error message as result.      Figure 2.5 Erroneous Modification of Tuples  We will now try to add the name “John” to the tuple names. Since names has 2 elements, the index of the last element must be 1. Hence, we add the new element to names by referring to a new index, namely 2.      Figure 2.6 Erroneous Adding of Elements to Tuples  Unsurprisingly, we also receive an error message here. |

Nevertheless, we are allowed to concatenate two tuples into a single tuple by connecting them with a “+” sign.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to concatenate the two tuples names and scores and name the new one newtuple.      Figure 2.7 Concatenation of Two Tuples  The concatenation of tuples works in the same way as it works for strings. As a result, we can re-attempt to add a new name "John" to the tuple names. The only thing we need to change is to put "John" in a tuple first.      Figure 2.8 Adding Elements to Tuples by Concatenation  To put the new name “John” into a tuple is indeed tricky since Python would not recognise syntaxes such as ("John") or "John" as a tuple. The reason here is that all elements in a tuple must be separated by commas. As a result, we put a comma behind "John" and leave the next element blank so as to tell Python that this is a tuple of length 1. We can see that after concatenating newname with names, Python will ignore the blank element in newname and append "John" as the only element in newname to "Peter" and "Mary", the original elements in names. |

#### 1.1.4 Length of Tuples

When dealing with tuples, lists or dictionaries, the function len() can be useful since it will return the length, i.e., the number of elements of such an object.

|  |
| --- |
| tuple\_length = len(tuple\_name) |

The length is also often used as an index to subset a tuple or control the indexing in our code so that we cannot refer to indices that go beyond the largest index of a tuple.

|  |
| --- |
| **Example (Cont’d):** We use the len() function to determine the length of the tuple all\_data. We will then use it as an index to subset the last element of a tuple.      Figure 2.9 Length of Tuples and Index Out of Range  We receive an error message here since all\_data[data\_length] is equal to all\_data[4]. Recall that the last index of a tuple is the length of the tuple minus 1. As a result, we can reach our original goal by subtracting 1 from the data\_length, which is equal to 4, for indexing.      Figure 2.10 Subsetting the Last Element of Tuples  By using the index data\_length - 1, we are now accessing the element all\_data[3], which is the last element of the tuple all\_data and 86 in this case. |

#### 1.1.5 For-Loops and Tuples

One advantage of tuples, or other compound data types, is that we can access and extract their elements by using the for-loop iteratively.

In Study Unit 1, we learned how to use the range() function to generate a list for the for-loop to run over it. Generally, if there are tuples, lists or dictionaries already created and existing while the program is running, we can use the for-loop directly by putting the name of the tuple, list, or dictionary in the for-statement.

|  |
| --- |
| for counter in tuple\_name:  instructions |

In the above syntax, tuple\_name can certainly be replaced by any list\_name or dictionary\_name.

|  |
| --- |
| **Example (Cont’d):** Now we would like to print out each element of the tuple all\_data onto the screen. The name of the counter here is records.      Figure 2.11 Printing Tuple Elements by for-Loops |

### 1.2 Lists

Lesson Recording - Python Lists

#### 1.2.1 Creating Lists

Another type of compound data type in Python is the list. A list is just like a tuple but with two main differences:

1. The data are comma-separated items wrapped by a pair of square brackets.
2. The content of a list is modifiable.

We can construct a Python list in a similar fashion like a tuple.

|  |
| --- |
| list\_name = [element1, element2, …] |

As mentioned before, we must wrap the data in a list by square brackets. However, unlike a tuple which we may omit the round brackets when defining it in our code as long as the data are separated by commas, we must define a list with the square brackets in the program. If we omit the brackets, Python will interpret the data as a tuple.

Same as tuples, lists may contain any type of values: floats, integer, Booleans, strings, or more advanced Python types like lists. The last one is indeed a very interesting property of the Python compound data type since we can namely put a list within a list, or a tuple in a dictionary, and so on.

|  |
| --- |
| **Example (Cont’d):** We define two lists, names and scores, to store the data of the students in the class.      Figure 2.12 Creating Lists in Python |

#### 1.2.2 Subsetting Lists

We can also access elements of a list by the index operator. All the indexing and subsetting techniques introduced in Chapter 1.1.2 for tuples are applicable to lists.

|  |
| --- |
| **Example (Cont’d):** Here we will subset the lists that we have defined above into four new lists and variables, respectively.      Figure 2.13 Subsetting Lists  In the first line (line 1 and 2 are not counted here), name1 = names[0], we simply extract the first element of names and store it in name1. Since we have only extracted one element, it will not be stored as a new list. Instead, it will become a variable, which we can see from the screen output that the value "Peter" is not wrapped by a pair of square brackets. And the type of the new variable will be the same as the type of the extracted element, which is a string variable in this case.  In the second line, name2 = names[1:], we extract multiple elements from the original list and save it as a new list called name2. Recall that if we use open end indexing, that is, we leave the value behind the colon blank, Python will take all elements from the original list starting from the starting value until the end. Here, it indicates that Python should extract the elements with the indices 1 and 2 from names.  In the third line, score1 = scores[0:2], the first and second elements from scores will be extracted and saved into a new list score1. We should always be aware that the last index is never included in indexing. As a result, only the elements 72 and 86, and not 35 are taken over in score1.  In the fourth line, score2 = scores[-2], we use negative indexing to extract element starting from the last element of scores. The value -2 indicates that we would like to extract the second last element, which is 86 in this case. The resulting object here is indeed a numeric variable since we only extract one numeric value from the original list.  In the fifth line, score3 = scores[-2:-1], we extract the element with the indices -2:-1 from scores. Since the last index is not included in the subsetting, the only relevant index here is -2. As a result, we should obtain the same result as the fourth line. However, there is one significant difference between this line and the fourth line: the new object here is still a list instead of a variable. In other words, if we want to create a new list with a single element from another list, we will have to use multiple indexing. |

#### 1.2.3 Modifying Lists

Different from tuples, we can change the content of a list or add elements to it. To edit specific items of a list, we simply assign new values to them after subsetting them.

|  |
| --- |
| list\_name[index] = new value |

We can also use multiple indexing to edit multiple items.

|  |
| --- |
| list\_name[start index:end index] = [list with new values] |

If we only intend to edit the items in the list, then the length of the indices on the left-hand side, which is the number of items to be modified, must be the same as the length of the list with the new values. If the list of new values is longer than the indices, items will be added to the list on the left-hand side. On the other hand, if the list of indices is shorter, items will be removed from the original list.

|  |
| --- |
| **Example (Cont’d):** Suppose the name of the third student is not John, but Jon, and we would like to change it now. Once again, the third item has the index 2.      Figure 2.14 Editing Single Item in a List  Figure 2.14 shows how the third element has been replaced by a new value. Now, suppose that the lecturer has to deduct two marks from the students who scored 72 and 86 after rechecking their exam papers. In other words, we need to modify the first two elements of the list scores.      Figure 2.15 Editing Multiple Items in a List  In the second line of the output, the first value has been replaced by 70 and the second one by 84. However, if our list [70, 84], does not only contain two values but three, the output will become:      Figure 2.16 Replacing More Values than the Specified Indices  Python will not treat it as a syntax error. Instead, it will replace the values of the indices 0 and 1 by the first two new values, 70 and 84, and add a new value to it, namely 60, before returning to the rest of the original list. As a result, the list scores has now four instead of three elements.  Let us turn to the opposite situation and assume that the indices for modification are longer than the list of new values.      Same as before, Python will not treat the code here as a syntax error as well. Instead, it will replace the values of the indices 0 and 1 by the only new value, 70, and then return to the rest of the original list, which is 35, the last value. As a result, the list scores has now two instead of three elements. |

In the above examples, we can see that lists can be extended or shrunk with the replacement of certain items in the original list by a longer or shorter list of new values.

#### 1.2.4 Concatenating & Merging Lists

Same as tuples, we can concatenate multiple lists into one by “adding” them together.

|  |
| --- |
| combined\_list = list1 + list2 + … |

Basically, Python uses the addition operation “+” to concatenate objects such as strings, tuples, or lists, rather than to carry out arithmetic addition except for numeric values. Concatenating lists is a recommended step in Python programming if the nature, i.e., content and type, of the lists is identical.

|  |
| --- |
| **Example (Cont’d):** Suppose we have two classes for the same course. The student names of these classes are stored in the lists class1 and class2, respectively. Equivalently, the exam scores are stored in the lists scores1 and scores2. Now, we would like to concatenate class1 and class2 into a new list called names and scores1 and scores2 into a new list called scores.      Figure 2.17 Concatenating Lists |

Even if the nature of the two lists is not identical, it is sometimes still quite convenient to store their data in one list for later use.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to concatenate the two lists names and scores into a new list called combined\_list in order to store the data of the entire class in a single compound data source.      Figure 2.18 Concatenating Lists with Different Data Types  So, the new list consists of the ten elements combined from the two lists names and scores. Apparently, concatenating lists with different types of data, such as strings and integers in this example, is allowed in Python. |

Figure 2.18 shows a straightforward concatenation of two lists in Python. However, such a combination of the two lists will lead to difficulty to distinguish the original nature such as meanings and types of the elements. For instance, in the above example, suppose the first item in the list scores is the exam score of the first student in names, that is, Peter’s exam score is 72, we will not be able to assign the score to the corresponding name unless we know that each score always belongs to the name five positions before. If we keep in mind that the length and contents of the concatenated list may change every now and then, the meaning and source of each element will become more and more untraceable with time.

Another way to solve this problem is to merge two lists into a new list without combining the elements together. Instead, each element of the new list is a list and not a single value.

To merge two lists into a new list while keeping them as “list elements”, we cannot use the addition operator as introduced before. Instead, we define the new list by putting the list names instead of some values as the elements.

|  |
| --- |
| list\_name = [list1, list2, …] |

The advantage of this merging technique is that we will always know that index 0 of the new list refers to the list names and index 1 to the list scores if the new list is a combination of two known lists in a fixed sequence. As a result, we will be able to trace back the origin and meaning of the data in the new list.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to put the two lists names and scores into a new list called class\_data and keep them as lists in the new list.      Figure 2.19 Merging Two Lists by Keeping Them as Lists in the New One  Different from the concatenation technique where all elements of the new list are just combined straightforwardly, we can see from Figure 2.19 that the elements of the list names are wrapped by each one pair of outer *and* inner square brackets to indicate that they belong to a list that in turn is an element of the new list class\_data. The same can be observed for the elements of scores too. |

Note that this merging technique is not limited to merging lists. We can also merge two tuples into a list and keep their types as tuple in the new list as well.

To access a single element in the merged list, we need to use the double index operator [] since the single index operator would return one of the original lists to us.

|  |
| --- |
| **Example (Cont’d):** Now we extract the first element as well as the first name and score from the merged list class\_data and print them to the screen.      Figure 2.20 Extracting Elements from a Merged List  In line 10, the object that we put in the print() function is class\_data[0], which is the first element of class\_data. And this element happens to be the original list names. In line 11, we try to extract the first element from the first list within class\_data by using the double index operator class\_data[0][0]. As a result, Python extracts the first element in class\_data first, which corresponds to the list with the names elements. From there, Python extracts the element with the index 0, which is “Peter” in this case. The same has also been carried out with class\_data[1][0], which is the first element of the second list. The resulting element is 72 here. |

It is noteworthy that the indexing technique introduced in the above example also works for multiple indexing.

#### 1.2.5 Printing Lists

In the previous study unit, we learned to use loops to carry out iterative tasks. In fact, loops can be very useful when working with lists. The reason is obvious: since the items of a list can be accessed by their indices, we can easily use loops to subset, print, and/or modify them.

Furthermore, we learned how to generate a list of integers to serve as sort of a counter for the iterations of the for-loops. Now, after being familiarised with the concept of lists, we do not always need the range() function to create these integers for us. Instead, we can simply use any available list as our counter. Nevertheless, the range() function can still be very useful in some situations.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to print all the student names of the two classes to the screen.      Figure 2.21 Printing List Elements Line by Line to the Screen  In this example, we print the elements of the list names line by line to the screen. In each iteration, Python will assign the value of the current list element to the counter variable i, and the print() function will print the value stored in i to the screen.  Suppose we would like to modify the screen output and use string formatting to print the score and the student name in the same line by linking them up using more natural language.      Figure 2.22 Printing Elements of Multiple Lists Using Formatted String  Since we want to run through two lists, names and scores, and print out their corresponding elements, we cannot use one of the lists as our counter in the for-loop. Thus, we need to use the range() function to generate a list of integers as our counter. Most conveniently, the integers should be exactly corresponding to the indices so that we can subset our lists within the for-loop directly. The obvious start index is 0, and the end index would be the number of items in our lists minus one. Since the range() function does not include the end value of the range in the result, we can therefore simply take the length of our list for this purpose. Here, we can apply the len() function, introduced in Chapter 1.1.4, to determine the length of names, and then store the result in the variable listlen, which will in turn be taken as the end of our integer list for the range() function.  Within the for-loop, we will have to instruct Python to print the element of names and scores with the index i for each iteration. And we can subset them by names[i] and scores[i], respectively. |

In the above example, we need to extract the information from two separate lists. In Chapter 1.2.4, we learned how to merge two lists into one single list where the original lists are kept as list elements in the new one, which makes the coding easier since we only need to work with one list. Instead of using the syntax we learned there, we can also use loops to carry out simple or complicated merging of lists.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to extract the first student’s name and score from the two lists and put these data into a small list, then do the same for the second student, etc. Eventually, all these small lists will be merged into a single list.      Figure 2.23 Rearranging Data Storage in Lists  In line 12, we create a temporary list to store the pair of values that we have extracted from names and scores. This temporary list templist will then be appended to finallist, our target list, in line 13. This line is essential for the whole process as we would like to merge all the information into a single list eventually.  However, it is important that we define finallist as an empty list before the for-loop starts. To define an empty list, we simply assign a pair of square brackets [] with no content to our target list. If we did not define our target list first, line 13 would produce a syntax error as Python would not know what finallist is.  Furthermore, the instructions inside a loop are identical in every iteration unless we add an if-condition in it. If we defined finallist as an empty list inside the for-loop instead, the list would be re-initialised and become empty in each iteration. And if we simply have finallist = templist in line13, the complete content of finallist would be replaced in each iteration. The final output would then only consist of the data of the last student. It is therefore important to accumulate data in every iteration by appending templist to finallist.  The square brackets wrapping templist in line 13 is important since we want to keep the data of each student as a small “sublist” within finallist. If the square brackets are omitted, each element in the small list will be appended as an individual element to finallist. Hence, it is important to be familiarised with how brackets are placed in a syntax appropriately. The output can be quite different.  This arrangement of finallist is perhaps the most natural way to store the data. In each small list, we have the data of each student. In other words, if we extract an element of the list finalist, we will have all the data of the *one* corresponding student which can be quite convenient when dealing with these data further.      Figure 2.24 Printing Data of a List Using Formatted String  After creating finallist, we can use a for-loop to run through it. Within the for-loop, Python will extract one element from finallist in each iteration and store it in the variable i. Note that i is not a counter variable here. It is more like a temporary storage for the current element of the list that is being run through. Since each element of finallist is a list itself with 2 elements, we can subset them by i[0] and i[1], respectively. From the construction concept of each of these small lists we know that the first element is the student’s name, and the second element is the corresponding exam score. As a result, all we need to do is to put the subsets at the right place of the formatted string for screen output. |

#### 1.2.6 Entering Data to Lists

So far, we assign pre-defined values to the lists in our code directly. But we can also let the user enter his/her own data and store them into a list by the input() function. If we do not limit the number of entries like in Study Unit 1, a while-loop with an appropriate exit condition will be the right approach here.

|  |
| --- |
| **Example (Cont’d):** The user is now asked to enter the student’s name and score, and he can stop entering by pressing ENTER for either the name or the score. After the entering process, the list will be printed to the screen for checking.        Figure 2.25 User’s Input to a List  In this example, we use several elements that we have learned in this and the previous study units to provide a clear input environment to the user and the suitable functionality for data storage in a list.   * The program begins with the definition of an empty list called finallist that will be used to store the data eventually. This is the same step as in line 9 of Figure 2.23. * We initiate two variables before the while-loop starts: the counter variable i which is set to 1 initially, and the Boolean variable proceed which controls whether the while-loop should continue with the next iteration or not. * The while-loop will continue to run as long as proceed is True. Its value will only change from True to False if the user’s input of either the name or the score is an empty string "", i.e., ENTER. * We print an instruction to tell the user what to do before the loop starts. This instruction can also be put within the while-loop. In that case, the instruction will be printed to the screen in every iteration. * We start the printing string in line 4 with an escape sequence “\n” to create a blank line between the system prompt and our instruction. It is visually more comfortable when the texts are not put too close together. We have built in several escape sequences for line break in the program to create blank lines for the same purpose. * In line 6, we include the value of the counter variable in the formatted string to show the number of the student whose data the user is about to enter. * The if-condition in line 7 checks whether the user’s input for the student’s name is an ENTER key or not. If it is the case, the program will switch proceed to False and break from the while-loop by the command break, which we learned in Chapter 5.3 of Study Unit 1. For the mechanism of breaking from the while-loop, the line proceed = False is insufficient since the remaining instructions within the while-loop will still be carried out although the user intents to stop the entering process immediately. As a result, we need to add the break-command to it. Nevertheless, for the sake of programming “cleanliness”, we also add the line to change the value of proceed to False. * Starting from line 10, we implement a while-loop controlled by a Boolean variable called valid\_input to check whether the user’s input of the score is numeric or not. This part of the code is mostly taken over from Chapter 5.4 in Study Unit 1. The main difference between the code there and our current one is that the user is allowed to press ENTER, i.e., an empty string to quit the entering process instead of -1 as in Study Unit 1. As a result, we cannot convert the user’s input to float or integer in the same line as the input() function, or else the if-condition in line 13 could become invalid. * The exit mechanism for the input of the student’s score is almost the same as the one for the student’s name. The only difference here is that both syntaxes proceed = False and break are not redundant. The break command is used here to break out from the while-loop that controls the numeric input, and not the outer while-loop for entering data of multiple observations. The change of value for proceed from True to False prevents the merge of list in lines 22-25. Hence, the program will jump to the end of the while-loop and break from it due to proceed = False. * The conversion of the variable std\_score to a numeric value will be put in the try-block to prevent Python from stopping the program due to the occurrence of an error. If the input of std\_score is indeed a number (but stored as string temporarily), the control variable valid\_input will turn to True and the while-loop will stop iterating. However, if the input is a character string (with exception of an empty string), Python will print the warning message that is written in the except-block, and the while-loop will start a new iteration. * In the formatted string of the input() function to enquire for the exam score in line 12, we embed the student’s name that has just been entered by the user in line 6. This can prevent users from entering the score of another student due to the visual confusion caused by the mass of information on the screen. * The same mechanism to store the data in templist and then to append them to finallist as in Figure 2.23 is employed here in line 23 and line 24. But the whole procedure will only be carried out if proceed = True. That is, if the inputs of both the name *and* the score are not empty strings. * We increase our counter variable by 1 after all the data for the ith student have been typed in and confirmed by the user. * Subsequent to the entire entering process, the program will print the entire list to the screen for checking purpose. However, this will only be executed if the user has at least entered the entire set of data for *one* student. This control is implemented in case the user has already quitted the entering process during the first iteration and prevents us from printing an empty list. We have also built in several escape sequences for line breaks here to enhance the visual comfort of the output. |

**Read**

Read the following official Python documentation: (<https://docs.python.org/3/library/‌stdtypes.html#sequence-types-list-tuple-range>) for details and examples on lists and on operations applicable to lists.

**Read**

Read the following section of the textbook on looping over elements of lists:

Exercise 32 Loops and Lists

**Read**

Read the following section of the textbook on accessing, adding, removing and joining elements of lists:

Exercise 34 Accessing Elements of Lists

Exercise 38 Doing Things to Lists

### 1.3 Dictionaries

Lesson Recording - Python Dictionaries

#### 1.3.1 Defining & Extracting Dictionaries Values

Another useful built-in data type is the dictionary. It is best to think of a dictionary as an unordered set of key:value pairs, with the requirement that the keys are unique (within one dictionary). The key:value pairs are separated by commas and wrapped in a pair of braces.

|  |
| --- |
| dictionary\_name = {"key1":value1, "key2":value2, …} |

Unlike lists and tuples, which are indexed by a range of numbers, dictionaries are indexed by keys, which are usually strings or numbers. As a result, we use keys to subset a dictionary instead of indices.

|  |
| --- |
| value = dictionary\_name["key"] |

Hence, we need to use the keys to extract values from the dictionary.

|  |
| --- |
| **Example (Cont’d):** Previously, we store the name and score of a student in a small list and merge all these “sub-lists” into a big list subsequently. Now, we save our data of class1 in a dictionary instead. In the following, we will use the student names as the key and their scores as the value.      Figure 2.26 Defining a Dictionary  To extract the score of Peter, we can put the key, "Peter", in the index operator [].      Figure 2.27 Extracting Value from a Dictionary  Note that if the key that we would like to refer to is a string, we will have to put it within a pair of quotation marks. |

The advantage of dictionary is that we can choose a much suitable description for the value and use it as the key. The programmers or users do not need to rely on their memory or understanding of the code to trace back the nature and type of the values in a tuple or a list like our examples in Chapter 1.2.4 and Chapter 1.2.5.

#### 1.3.2 Printing Dictionaries

We can use for-loops to print out the items from a dictionary line by line. However, the syntax is quite different from tuples and lists since each entry in a dictionary contains two elements: key and value. On the one hand, we can use the key to extract the corresponding value as demonstrated in Chapter 1.3.1, on the other hand, we cannot simply use the index operator [] to refer to the keys. One possibility to get the keys of a dictionary is to use the .keys() method:

|  |
| --- |
| dictionary\_name.keys() |

Equivalently, the .values() method can extract all values from a dictionary.

|  |
| --- |
| dictionary\_name.values() |

Method is like a function to carry out certain actions on the object before the dot (.). We will have detailed discussion on methods and functions in later chapters. Here, the object is a dictionary, and the methods are .keys() and .values(). Applying these syntaxes on any dictionary, Python can extract all the keys and values from it.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to obtain all the keys and values separately from the dictionary class1 defined in Figure 2.26.      Figure 2.28 Extracting Keys and Values from a Dictionary  The method .keys() returns an object called dict\_keys() with the keys in our dictionary, and the method .values() returns an object called dict\_values() with the values. However, we cannot work with these keys and values yet since we cannot extract them from the dict\_keys() and dict\_values() objects directly. We need to transform them to list by a function called list() first.      Figure 2.29 Store Keys and Values from a Dictionary in Lists  The keys and values are now stored in their corresponding lists, recognisable by the square brackets around them. From the techniques introduced in Chapter 1.2, we can print the keys line by line now.      Figure 2.30 Print All the Keys from a Dictionary Line by Line  We can apply the same technique to print the values line by line too. Nevertheless, we can also use the index operator [] on the original list class1 to print the keys and the corresponding values. To extract the keys individually, we need to initiate a for-loop that iterates through the list containing the dictionary keys, which is class\_keys here. In each iteration within the for-loop, one key of the list will be stored in the variable i. The value can then be extracted by class1[i].      Figure 2.31 Print All the Keys and Values from a Dictionary Line by Line  We put two objects separated by comma in the print() function. This is a command to tell Python to print these objects in the same line to the screen. And the values assigned in these objects are separated by an empty space between them. |

If we want to extract all keys and values from a dictionary in the same step, we can apply the .items() method on a dictionary.

|  |
| --- |
| dictionary\_name.items() |

Same as the .keys() and .values() methods, the result returned by Python from the .items() method is not an object that can be accessed directly. Nevertheless, it can be converted to a list of keys and values by the list() function.

|  |
| --- |
| **Example (Cont’d):** We apply the .items() method on class1 to obtain all the keys and values of the dictionary.      Figure 2.32 Extracting Items from a Dictionary  The first print() function prints the dict\_items() object returned from the .items() method to the screen. This object contains every pair of keys and values from the dictionary stored in a tuple. In the second print() function, we convert the dict\_items() object to a list. And the printed object is a list with each pair of keys and values stored in a tuple as its elements. |

The list created from a dict\_items() object contains tuples that have two elements in each of them: the keys and the values. We can easily use a for-loop and the index operator to print the contents. Here, we introduce an extension of the for-loop so that we can omit the indices when referring to the keys and values.

|  |
| --- |
| for element1, element2 in list(dictionary\_name.items()):  instructions |

The syntax list(dictionary\_name.items()) in the above for-loop can certainly be replaced by any defined list created from a dictionary. The main difference between the usual for-loop we know and the one we introduce here is that we use two temporary storage variables instead of one in it. The mechanism is rather simple. In each iteration, Python will extract one tuple from the list, and each element of the tuple will then be stored in one of these storage variables. Since the keys are stored first in the tuples, element1 will contain the key, and element2 will contain the value. If we want to use these two values in our instructions, we can simply refer to these variables without using the index operator.

|  |
| --- |
| **Example (Cont’d):** Now we use the double storage variables in a for-loop to print out the keys and values of our dict\_items() object after being converted to a list.      Figure 2.33 Print Items from a Dictionary after Converting it to a List  In line 2, we extract the dict\_items() object and convert it directly to a list named class\_items. The two storage variables of the for-loops are key and val here. And in the print() function, we use a formatted string to print out the two values in a normal sentence for the user to read. |

#### 1.3.3 Editing Dictionaries

We can change a value of a dictionary by assigning a new value to a certain key.

|  |
| --- |
| dictionary\_name["key"] = value |

The value can be a numeric value, a character string, a tuple, or a list.

|  |
| --- |
| **Example (Cont’d):** Suppose the score of Peter was 70 instead of 72. We can simply change it by assigning a new value to the key “Peter”.      Figure 2.34 Change One Value in a Dictionary  There was then a second exam and each of our three students has a new score in addition to the scores that are already stored in the dictionary. Below is one rather trivial approach to carry out this task.      Figure 2.35 Add Items to Every Dictionary Key and Convert Values to Lists  The above approach replaces the original values in the dictionary by some lists. Each list contains two scores of each student. Therefore, we must include the first score of each student in the list as well, which is not quite elegant in terms of programming conciseness. Another possibility is to use a for-loop to convert the value of each key and append the new score to it.      Figure 2.36 Print Items from a Dictionary after Converting to Lists Using for-loops  In this approach, we defined a new dictionary named exam2\_score to store the scores of the second exam. We also use the students’ names as our dictionary key here. Hence, class1 and exam2\_score share the same keys which makes the mechanism within the for-loops much easier to handle. The list that is used for the for-loop to iterate is the key list of the dictionary class1. In each iteration, the variable i will store one key from the list. Since both dictionaries have identical keys, we can use the dictionary subsetting technique on both dictionaries by putting i in the index operator. Subsequently, we merge the extracted values from both dictionaries to one list, and then assign it to the corresponding key of class1. Eventually, we print out the edited dictionary for checking purpose. |

While assigning new values to a key in a dictionary is rather straightforward, editing a key in a dictionary is not a simple task in Python. Basically, the keys of a dictionary are immutable and cannot be changed directly. But we can create a new key in a dictionary, take over the values from the old key, and then delete the old key in the last step. To delete a key in a dictionary, we can use the syntax del.

|  |
| --- |
| del object |

Note that the object in this syntax does not only refer to a key in a dictionary. It can be any Python object such as a variable, a list, a tuple, a dictionary, etc. Once an object is deleted, it will no longer be available in the program.

|  |
| --- |
| **Example (Cont’d):** Suppose the key “John” was a typo when entering the data. It should be “Jon” instead.      Figure 2.37 Changing a key of a Dictionary  First, we assign the value of “John” in the dictionary class1 to a new key called “Jon” of the same dictionary. Subsequently, we delete the key “John” and its value as an entire object from the running program. |

On the other hand, if we want to add a new key to a dictionary, we can simply assign a value to a new dictionary key.

|  |
| --- |
| dictionary["new key"] = value |

Basically, the syntax to add a new key is just the same as the syntax to edit an existing key. The only difference is that the key put in the index operator must be a new one if we wish to add a new item.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to add a new student, Michael, who scores 60 in the exam, to our dictionary.      Figure 2.38 Adding a New Key to a Dictionary |

However, if we want to merge two dictionaries, Python offers two options.

|  |
| --- |
| new\_dictionary = dictionary1 | dictionary2 *#Version 3.9+* |
| new\_dictionary = {\*\*dictionary1, \*\*dictionary2} *#Version 3.5+* |

The first syntax is a new option available from Python version 3.9 onwards (Recall from Study Unit 1 that this study guide is written based on Python version 3.9). It uses the “Bitwise Or” operator, which is a vertical line “|”, to merge two dictionaries. The second syntax is an option available from Python version 3.5 onwards.

|  |
| --- |
| **Example (Cont’d):** Suppose we have a second dictionary with the exam results of a second class, and we would like to merge these two dictionaries together.      Figure 2.39 Merging Two Dictionaries Using the v3.5+ Option      Figure 2.40 Merging Two Dictionaries Using the v3.9+ Option  As we can see, the two methods deliver the same result. |

**Read**

Read the following section of the textbook on creating and manipulating dictionaries: Exercise 39 Dictionaries, Oh Lovely Dictionaries

**Read**

Read the following official Python documentation for more details and examples on dictionaries:

<https://docs.python.org/3/library/stdtypes.html#typesmapping>

## Chapter 2 Integrated Methods and Functions

Lesson Recording - Integrated Methods and Functions in Python

### 2.1 Built-In Functions in Python

In the previous chapters, we have come across various built-in functions of Python, such as print(), input(), int(), etc. These functions are always available in the Python environment. A function is a routine program that processes values which are passed on to it as an argument, or a parameter, and returns some results to the user eventually. For example, a float value is passed on to the int() function as an argument, Python then removes all the decimal digits and returns an integer as result.

The following table contains all the built-in functions in alphabetical order.

Table 2.1 Built-in Functions of Python

|  |  |  |  |
| --- | --- | --- | --- |
| abs() | all() | any() | ascii() |
| bin() | bool() | breakpoint() | bytearray() |
| bytes() | callable() | chr() | classmethod() |
| compile() | complex() | delattr() | dict() |
| dir() | divmod() | enumerate() | eval() |
| exec() | filter() | float() | format() |
| frozenset() | getattr() | globals() | hasattr() |
| hash() | help() | hex() | id() |
| input() | int() | isinstance() | issubclass() |
| iter() | len() | list() | locals() |
| map() | max() | memoryview() | min() |
| next() | object() | oct() | open() |
| ord() | pow() | print() | property() |
| range() | repr() | reversed() | round() |
| set() | setattr() | slice() | sorted() |
| staticmethod() | str() | sum() | super() |
| tuple() | type() | vars() | zip() |
| \_\_import\_\_() |  |  |  |

(Source: <https://docs.python.org/3/library/functions.html>)

Some of the listed functions are rather straightforward such as abs(), sum(), round() , etc. There are also some that are quite unclear in terms of their functionality or area of use such as frozenset() or staticmethod() just by looking at their names. You can visit the website <https://docs.python.org/3/library/functions.html> to get detailed explanation on how to integrate and apply all these functions in Python programs.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to summarise the combined exam results of class1 and class2 by calculating their mean, maximum and minimum.      Figure 2.41 Computing Statistics of a Dictionary’s Values  In the first three lines, we repeat the syntaxes from Chapter 1.3.2 to merge two dictionaries into a new one called all\_classes. Subsequently, all the exam scores are extracted by the .values() method from all\_classes. In line 5, we use the sum() function to add up all the numbers stored in scores and divide the result by the number of elements in scores, determined by the len() function, to obtain the mean exam scores. The mean will then be rounded to two decimal places by the round() function when embedding it in a formatted string for printing purpose. In the last two lines, we determine the highest and lowest exam scores by the max() and min() functions and include them in the formatted strings for the print() function to print them onto the screen. |

You may wonder why some rather basic functions such as a function for the calculation of the mean is missing in the above list. Some of those functions may be included in some common packages that will be introduced in Chapter 4. Some of them could be built-in methods instead, which will be introduced in the next section.

**Read**

Read the following official Python documentation for more details and examples on Python functions:

<https://docs.python.org/3/library/functions.html>

### 2.2 Built-In Methods in Python

Some routines in Python are not supposed to be applied as functions in the Python environment; instead, they are methods that can be applied to the objects they are attached to. In Study Unit 1, we learned the method .format() for format printing; and in this study unit, we come across the .keys(), .values() and .items() methods that can be applied on dictionaries to extract their keys and values. Same as functions, there are built-in methods that are always available in the Python environment. These methods will return results to the program once they are applied on defined objects during runtime. However, each method can only be applied to a certain object type. Below is a list of selected built-in methods of Python.

Table 2.2 Built-in Methods of Python

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Applicable Object Type** |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list | List |
| [capitalize()](https://www.w3schools.com/python/ref_string_capitalize.asp) | Converts the first character to upper case | String |
| [casefold()](https://www.w3schools.com/python/ref_string_casefold.asp) | Converts string into lower case | String |
| [center()](https://www.w3schools.com/python/ref_string_center.asp) | Returns a centred string | String |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements | List, Dictionary |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy | List, Dictionary |
| [count()](https://www.w3schools.com/python/ref_string_count.asp) | Returns the number of times a specified value occurs in a string | String |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value | List, Tuple |
| [endswith()](https://www.w3schools.com/python/ref_string_endswith.asp) | Returns True if the string ends with the specified value | String |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list | List |
| [find()](https://www.w3schools.com/python/ref_string_find.asp) | Searches the string for a specified value and returns the position of where it was found | String |
| [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and value | Dictionary |
| [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key | Dictionary |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the position of the first element with the specified value | List, Tuple |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position | List |
| [isalnum()](https://www.w3schools.com/python/ref_string_isalnum.asp) | Returns True if all characters in the string are alphanumeric | String |
| [isalpha()](https://www.w3schools.com/python/ref_string_isalpha.asp) | Returns True if all characters in the string are in the alphabet | String |
| [islower()](https://www.w3schools.com/python/ref_string_islower.asp) | Returns True if all characters in the string are lower case | String |
| [isnumeric()](https://www.w3schools.com/python/ref_string_isnumeric.asp) | Returns True if all characters in the string are numeric | String |
| [isspace()](https://www.w3schools.com/python/ref_string_isspace.asp) | Returns True if all characters in the string are whitespaces | String |
| [istitle()](https://www.w3schools.com/python/ref_string_istitle.asp) | Returns True if the string follows the rules of a title | String |
| [isupper()](https://www.w3schools.com/python/ref_string_isupper.asp) | Returns True if all characters in the string are upper case | String |
| [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair | Dictionary |
| [join()](https://www.w3schools.com/python/ref_string_join.asp) | Joins the elements of an iterable to the end of the string | String |
| [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys | Dictionary |
| [lower()](https://www.w3schools.com/python/ref_string_lower.asp) | Converts a string into lower case | String |
| [lstrip()](https://www.w3schools.com/python/ref_string_lstrip.asp) | Returns a left trim version of the string | String |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position | List, Dictionary |
| [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair | Dictionary |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the first item with the specified value | List |
| [replace()](https://www.w3schools.com/python/ref_string_replace.asp) | Returns a string where a specified value is replaced with a specified value | String |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list | List |
| [rfind()](https://www.w3schools.com/python/ref_string_rfind.asp) | Searches the string for a specified value and returns the last position of where it was found | String |
| [rstrip()](https://www.w3schools.com/python/ref_string_rstrip.asp) | Returns a right trim version of the string | String |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list | List |
| [split()](https://www.w3schools.com/python/ref_string_split.asp) | Splits the string at the specified separator, and returns a list | String |
| [splitlines()](https://www.w3schools.com/python/ref_string_splitlines.asp) | Splits the string at line breaks and returns a list | String |
| [startswith()](https://www.w3schools.com/python/ref_string_startswith.asp) | Returns true if the string starts with the specified value | String |
| [strip()](https://www.w3schools.com/python/ref_string_strip.asp) | Returns a trimmed version of the string | String |
| [swapcase()](https://www.w3schools.com/python/ref_string_swapcase.asp) | Swaps cases, lower case becomes upper case and vice versa | String |
| [title()](https://www.w3schools.com/python/ref_string_title.asp) | Converts the first character of each word to upper case | String |
| [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs | Dictionary |
| [upper()](https://www.w3schools.com/python/ref_string_upper.asp) | Converts a string into upper case | String |
| [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary | Dictionary |
| [zfill()](https://www.w3schools.com/python/ref_string_zfill.asp) | Fills the string with a specified number of 0 values at the beginning | String |

(Source: <https://www.w3schools.com/python/default.asp>)

The complete list of methods can be found on [https://www.w3schools.com/‌python/python\_ref\_tuple.asp](https://www.w3schools.com/python/python_ref_tuple.asp), <https://www.w3schools.com/python/python_ref_list.‌asp>, <https://www.w3schools.com/python/python_ref_dictionary.asp>, and <https://‌www.w3schools.com/python/python_ref_string.asp>.

|  |
| --- |
| **Example (Cont’d):** Now we also have each student’s surname in our dictionary, and we want to sort the data according to it. In the first step, we would like to erase all the double spacing between the first and surname in the dictionary keys.      Figure 2.42 Replacing Double Spacing by Single Spacing  In the first three lines, we again repeat the syntaxes from Chapter 1.3.2 to merge two dictionaries into a new one called all\_classes. We implement a for-loop to iterate through all the dictionary keys and search for one with double spacing by the .find() method. If a key has double spacing, it will be replaced by single spacing. The .replace(original\_string, replacing\_string) method will take over this substitution process. Subsequently, Python assigns the original value to the new key and delete the old key from the dictionary.  In the next step, we would like to capitalise each word in the dictionary keys. It is important to mention that the .capitalize() method is not suitable for our task since it will only convert the first character in each key to upper case. Indeed, we need the .title() method to make sure that each word in a key starts with a capital letter after the entire process.      Figure 2.43 Capitalise Each Word in Strings  In the above program, we do not search and replace double spacing first and capitalise each word in the subsequent step. Instead, we initiate a new variable newkey which is the result of a chain execution of two methods in the same line. First, we convert the first letter of each word in the keys to upper case by the .title() method and then replace the double spacing by a single spacebar with the .replace() method. If the original key was correctly capitalised and spaced, the value stored in newkey must be the same as the original one. In this case, Python would do nothing and jump to the next key. Otherwise, Python will create a new key with the capitalisation and spacing correction and assign the value of the original key to it, and then delete the original key from the dictionary.  In the third step, we need to switch the format of the keys from “First name Last name” to “Last name, First name” for the subsequent sorting process (for simplicity, we assume that the students’ names are all in the “First name Last name” format).      Figure 2.44 Switching Surname and First name in Strings  In Figure 2.44, we add two lines to format the dictionary keys according to our needs. First, we use the .split() method to separate each student’s name into two parts: first name and last name, and the spacebar is used as the separation character of the string here. The result will be then stored as a list called name\_parts with the structure [“First name”, “Last name”]. In the next line, we swap the appearance order of first and last names by concatenating “Last name” and “First name” from the list and add a “, “ between them to become a new string that is used as our final key. The replacement and delete process is identical to the previous one, and it should only be carried out if the original key and the new key are not identical.  Now, we can sort the dictionary by its keys in the ascending alphabetical order.      Figure 2.45 Sorting Dictionary by Its Keys  In line 11, the extracted dictionary keys are stored in a list called classkeys and this list is then sorted in the ascending order by the .sort() method in line 12. In line 13, we indicate that the contents of the dictionary all\_classes will be “reassigned” by wrapping the expression on the right-hand side with a pair of braces. In the braces, we initiate a variable k that will store the sorted keys in classkeys once the for-loop starts to iterate. Behind the variable k, we use a colon to indicate that the expression following it is the value that belongs to the key k in all\_classes. This syntax will then be followed by the for-loop mentioned before. It is important to use the same key variable, k in this case, in this for-loop.  It is noteworthy that no colon behind the for-statement will be needed if a for-loop is initiated this way. The instruction to be carried out in each iteration should be placed before the for-loop. By doing this, we can simplify our program by writing two commands in the same line instead of writing some lengthy syntaxes to initiate a for-loop to run through all the dictionary keys one by one in the traditional way.  Note that the sorting step can also be achieved with the same efficiency by the sorted() function. Since we only focus on the functionality and application of methods in this section, we try to use methods merely in our demonstration of the examples. |

**Read**

Read the following website for more details and examples on Python methods for strings:

<https://www.w3schools.com/python/python_ref_string.asp>

**Read**

Read the following website for more details and examples on Python methods for lists:

<https://www.w3schools.com/python/python_ref_list.asp>

**Read**

Read the following website for more details and examples on Python methods for dictionaries:

<https://www.w3schools.com/python/python_ref_dictionary.asp>

**Read**

Read the following website for more details and examples on Python methods for tuples:

<https://www.w3schools.com/python/python_ref_tuple.asp>

## Chapter 3 User-defined Functions

Lesson Recording - User-defined Functions in Python

In Chapter 2, we are introduced to some built-in functions and methods that are already included in the Python programming environment. Functions and methods help us to carry out routine tasks which would require us to write very lengthy code to achieve the same functionality if we were to create the program by ourselves. Nevertheless, sometimes we can also write our own functions that suit our own needs.

A user-defined function (we will call it function in the following due to simplicity) can be viewed as a separate part of the code that will not be interpreted by Python until it is called from the main program. Usually, a function consists of four parts:

1. the function name
2. some arguments, i.e., values or parameters the function needs for its processing. This part is optional. If the function has no arguments, it will simply process all the instructions without any input from the main program.
3. the instructions of how and what to process within the function
4. an object (or a value) that should be returned to the main program at the end of the function. This is also optional. If no return object is specified, the main program will proceed without any output from the function.

In Python, the def-syntax indicates the definition of a function:

|  |
| --- |
| def function\_name(argument1, argument2, …):  instructions  return object |

Same as for-loops or if-conditions, the def-statement must end with a colon, and all the follow-up instructions and codes that belong to the function must be indented. Subsequently, the function can be called in the main program by integrating the function\_name at an appropriate place.

|  |
| --- |
| *… (Main program)*  y = function\_name(argument1 = object1, argument2 = object2, …)  print(y)  *… (Continue with main program)* |

We use a function to carry out a certain process. The objects object1 and object2 are the corresponding input for argument1 and argument2 to the function. It is essential for Python that object1 and object2 are already defined somewhere in the previous part of the main program. The output object (or value) from the function will then be assigned to the variable y.

**Caution!** It is tempting for beginners to “outsource” some parts of the main program and make them separate functions for the sake of “program cleanliness”. Though the code may look more structured at the first sight, the debugging process can be quite challenging if the whole program is jumping between the main program and the functions. The rules of thumb for using user-defined functions appropriately are:

1. if the same routine, probably with different arguments, appears more than once in the main program
2. if several functions should be combined into one which will then be used in the main program on multiple occasions
3. if a function can really increase the efficiency of the main program

In these cases, function can really simplify the program code and the debugging of it since there are less chances for syntax or logical errors.

|  |
| --- |
| **Example (Cont’d):** We repeat the same task from Chapter 2.2 by implementing two user-defined functions: list\_dictkeys(dicts) and sort\_dictkeys(dicts).    Figure 2.46 Defining Functions  The first function list\_dictkeys(dicts) is used to extract the keys from a dictionary and convert them to a list in the same step. It has an argument dicts, which is a dictionary variable to store the dictionary passed on by the main program to the function for the extraction and conversion process. This process is written in the return syntax directly since we target on returning the list as the output object of our function anyway. An additional step to save the list in an extra variable first and then to return the extra variable is therefore not necessary here.  The second function sort\_dictkeys(dicts) sorts the dictionary keys directly. The code is basically identical to the lines 11, 12 and 13 in Figure 2.45. The only difference here is the dictionary object is not a specific dictionary from the main program, but a variable called dicts that is used as the argument in the definition of the function and is also only used within the function. The main program will then pass on a dictionary to sort\_dictkeys and store the dictionary in dicts once it starts to process the instructions inside the function. It is noteworthy to mention that we can call another function within a function like in line 5, where we call list\_dictkeys(dicts) to generate the list of dictionary keys for further process.  The main program does not become significantly shorter, but simpler in terms of the number of functions and methods involved.      Figure 2.47 Sorting Dictionary Using User-Defined Functions  In line 12, the program commands the list that should be run through in the for-loop is the output object returned from the list\_dictkeys(dicts) function. In the bracket following the function name list\_dictkeys, we specify the dictionary all\_classes which should be stored in the variable dicts when passing on to list\_dictkeys(dicts). The same happened in line 19 where we use the sort\_dictkeys(dicts) function to sort the dictionary all\_classes. Note that the “dicts =” part is not required as long as the function only contains one argument, or when the objects to be passed on to the function are listed in the same sequence as the argument list written in the function code. It is therefore important to know either the sequence of the argument list of each function or to type the “argument =” part when calling a function.  Certainly, more from the main program can be “outsourced” to a function if those routines are called more frequently in the program. For instance, the process of formatting the students’ name can be written as a user-defined function as well. Since defining such a function will not simplify our program here, we keep this part of the program as in Figure 2.45 due to the aforementioned guidelines of using user-defined functions appropriately. |

**Read**

Read the following three exercises of the textbook for more details and examples on user-defined functions:

Exercise 18: Names, Variables, Code, Functions

Exercise 19: Functions and Variables

Exercise 21: Functions Can Return Something

## Chapter 4 Modules, Packages and Libraries

Lesson Recording - Modules, Packages and Libraries in Python

### 4.1 Import a Standard Package

Beside build-in functions and methods, as well as user-defined functions, Python also provides packages which we can think of as a directory of Python scripts, the so-called modules. These modules specify new functions, methods, and object types for solving particular tasks. Packages are organised hierarchically; that means they may contain sub-packages, as well as regular modules themselves.

The packages of the standard library are already installed in the Python environment. A library is a collection of codes for us to perform specific tasks without writing our own code. But before we can use the modules in our program, we need to import the package or a specific module of the package first.

|  |
| --- |
| import package\_name as package\_alias  from package\_name import module\_name as module\_alias |

In the first syntax, we import the whole package into our program. The alias is a name that is used to refer to that particular package from thereon in our program. It is advantageous to use a package alias if it has a very long name. Note that the “as package\_alias” part is optional in the import syntax. If the original package name is preferred, this part can be omitted.

The second syntax imports a particular module from a package. The alias here is the referral name of the module that we will use in our program, and not the package. Once again, the alias part is optional and can be omitted.

If the whole package is imported and we want to call a certain module from it, we will need to use the package name as the prefix and then indicate the module after a dot (.). The syntax should be something like:

|  |
| --- |
| y = package\_name.module\_name(argument1, argument2, …)  y = package\_alias.module\_name(argument1, argument2, …) |

If we have used an alias for the package name in the import process, we will have to use the second syntax instead of the first one.

But if we have only imported a single module from a package, we could call it directly by its name without referring to the package:

|  |
| --- |
| y = module\_name(argument1, argument2, …)  y = module\_alias(argument1, argument2, …) |

If we have only imported a single module from a package with an alias, we will have to use the alias instead of the original module name.

|  |
| --- |
| **Example (Cont’d):** If we need to calculate the mean and the standard deviation of the scores of the two classes, we can import the package statistics for this task.      Figure 2.48 Integrate Entire Package in a Program  We first create a function list\_dictvalues(dicts) to extract the values from a dictionary and convert them to list in the same step. Subsequently, we import the package statistics and use stat as our alias to refer to it in our program. Since we have imported the whole package, we need to use the prefix stat. whenever we call a module from it, which then happens in line 10 and line 11. Line 10 contains the calculation of the mean by using the stat.mean() function, and in line 11, we instructed Python to compute the standard deviation by the stat.stdev() function.  If we just want to calculate the mean, we can then choose to import the mean() function from the statistics package instead.      Figure 2.49 Integrate One Module from a Package  In line 4, we modify the syntax to the from … import … version. Since we feel that the module name mean does not need an alias, we keep the name as it is and call it in line 10 for the mean calculation. |

### 4.2 Managing Packages with pip/pip3

There are many Python packages available from the internet but not yet installed in the Python environment. To use those Python packages, we will first have to install them on our system. Then we can import them into our program same as the standard library. The simplest way to install such packages is to use PowerShell or Command Prompt as well (or similar terminal apps from other operating systems). Once we are prompted in the terminal window to give instructions to the operation system to carry out, type in one of the following commands and then press ENTER:

|  |
| --- |
| > pip install package\_name  > pip3 install package\_name |

“pip” or “pip3” refers to the installer program that Python uses for installing external packages. Basically, “pip3” is a newer version of “pip”. In most of the cases, we can use either one for our installation.

For instance, the package “numpy” will be needed in the next study units, and we would like to install it for our class preparation:

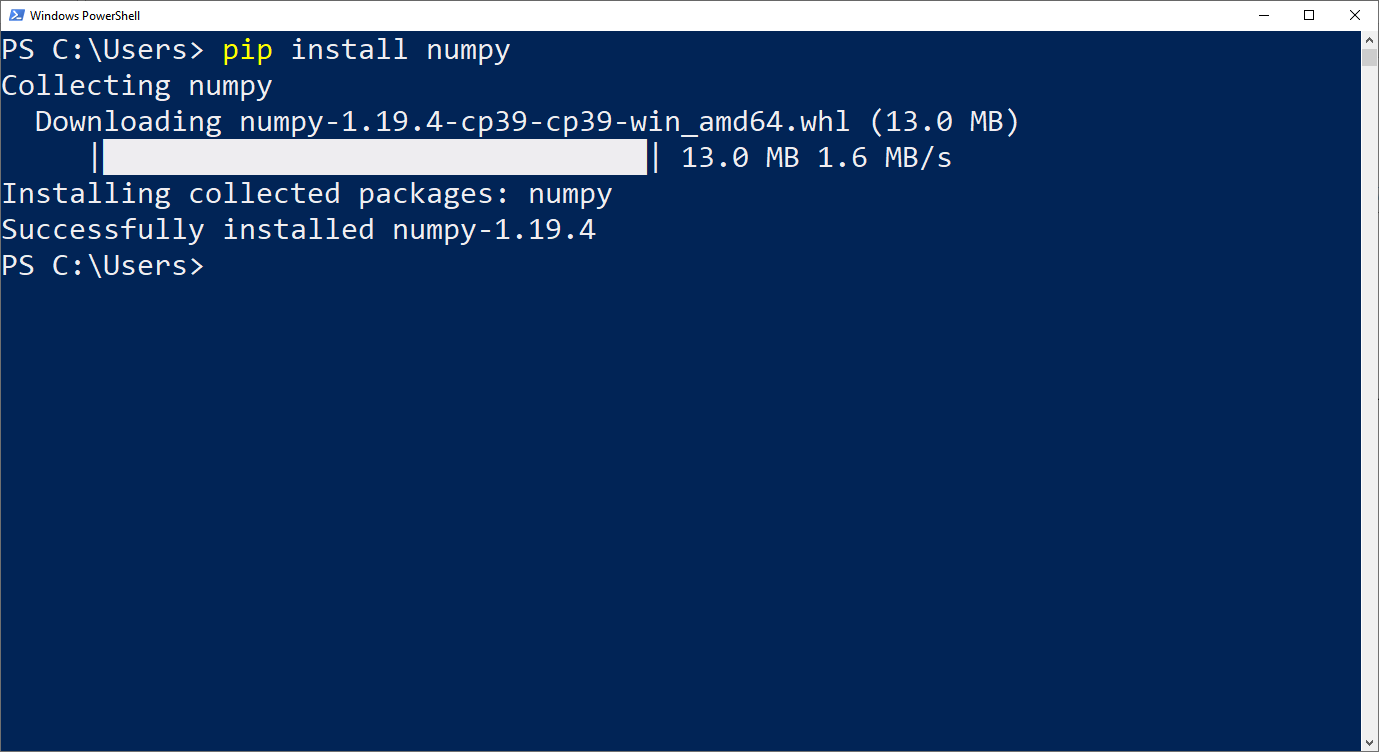


Figure 2.50 Installing Package “numpy” with pip

Once ENTER is pressed, Python will download the package installation file and install it subsequently. The message “Successfully installed xxx” will appear on the screen once the installation has been completed.

Another package that we will need in the next study units is “matplotlib”. This time we install it with pip3.

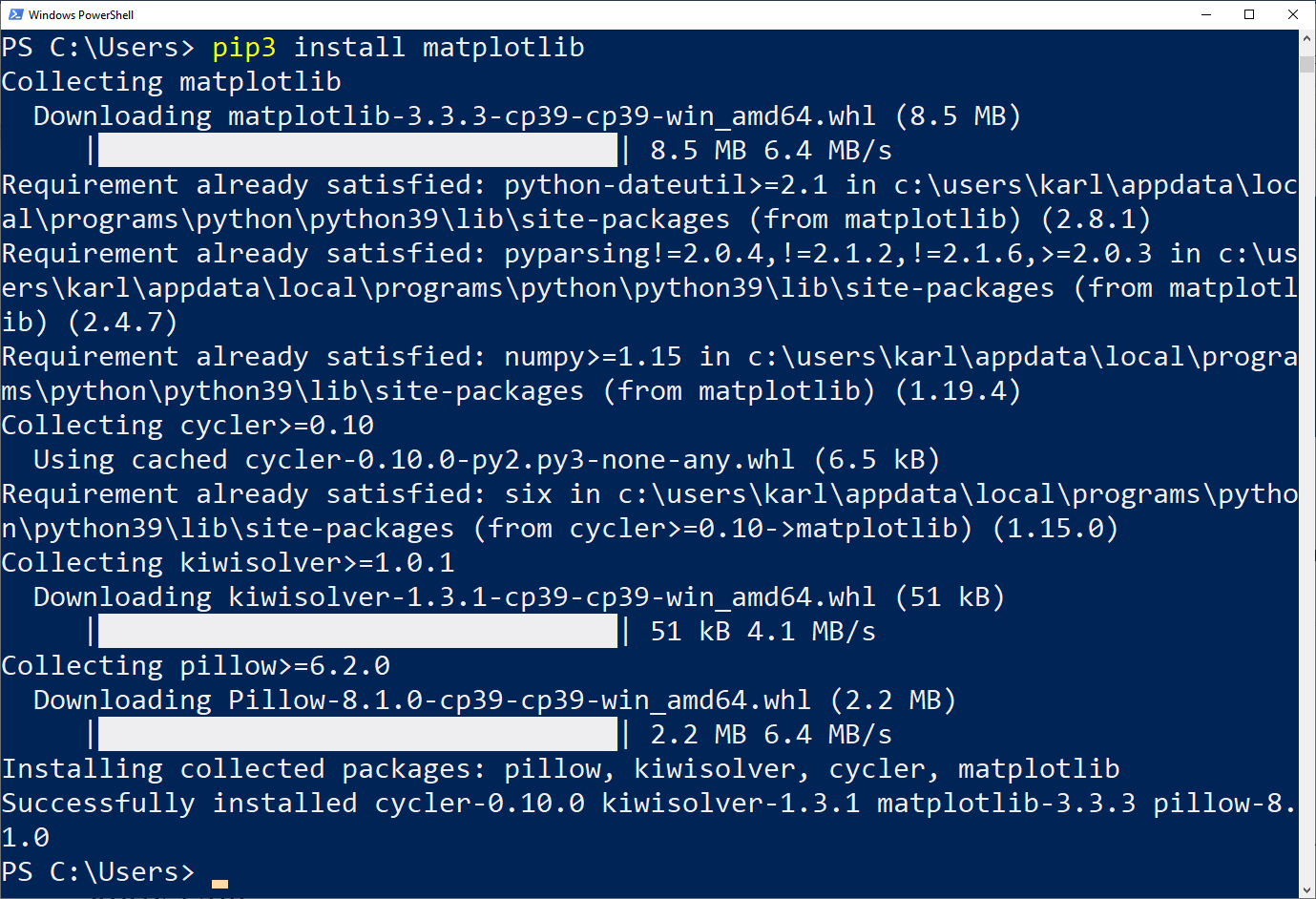


Figure 2.51 Installing Package “matplotlib” with pip3

In the same installation process, some other packages are downloaded and installed as well. This is because “matplotlib” needs some modules of these packages so that it can work.

If we wish to update/upgrade a package, the command that we need to type in our terminal app will be:

|  |
| --- |
| > pip install package\_name --upgrade  > pip3 install package\_name --upgrade |

And if we want to uninstall a package, the command will become:

|  |
| --- |
| > pip uninstall package\_name  > pip3 uninstall package\_name |

In Python, there are some packages that are used quite commonly for data analytics, and a couple of them will also be covered in this study guide.

Table 2.3 Some Common Packages for Data Analytics in Python

|  |  |  |
| --- | --- | --- |
| **Package** | **Description** | **In this Study Guide (Y/N)?** |
| [matplotlib](https://www.w3schools.com/python/ref_string_casefold.asp) | Creates data visualisation | Y |
| [n](https://www.w3schools.com/python/ref_list_append.asp)umpy | Manages multi-dimensional arrays | Y |
| [pandas](https://www.w3schools.com/python/ref_string_capitalize.asp) | Handles two-dimensional data tables | Y |
| pendulum | Provides complex coding for dates and times | N |
| requests | Sends HTTP requests from Python code | N |
| [sciki](https://www.w3schools.com/python/ref_string_center.asp)t-learn | Provides tools of data analytics | Y |
| [scipy](https://www.w3schools.com/python/ref_list_clear.asp) | Carries out scientific and technical computations | N |
| [sqlite](https://www.w3schools.com/python/ref_list_copy.asp)3 | Manages SQL database in Python | Y |

## Summary

In this unit, we have learned three types of built-in compound data structures of Python: tuples, lists and dictionaries. We discussed the major differences among these objects and the typical applications of them. Their creation and modification were also explained and demonstrated in detail. The most important issue here is the technique for subsetting and indexing the elements in these objects. We have also been introduced to functions, methods, and packages in Python. While some of them are built-in, i.e., they are already included in the Python environment, others can be user-defined or installed from external sources. Based on examples, we have been shown how built-in functions and methods, user-defined functions, as well as standard or external libraries can be applied to our Python programs.

## References

Python.org (n.d.). *Built-in Types*. Python Software Foundation. <https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range>

Python.org (n.d.). *Built-in Exceptions*. Python Software Foundation. <https://docs.python.org/3/library/stdtypes.html#typesmapping>

Python.org (n.d.). *Built-in Functions*. Python Software Foundation. <https://docs.python.org/3/library/functions.html>

Python.org (n.d.). *Python String Methods*. Python Software Foundation. <https://www.w3schools.com/python/python_ref_string.asp>

Python.org (n.d.). *Python List Methods*. Python Software Foundation. [https://www.w3schools.com/python/ python\_ref\_list.asp](https://www.w3schools.com/python/%20python_ref_list.asp)

Python.org (n.d.). *Python Dictionary Methods*. Python Software Foundation. <https://www.w3schools.com/python/python_ref_dictionary.asp>

Python.org (n.d.). *Python Tuple Methods*. Python Software Foundation. <https://www.w3schools.com/python/python_ref_tuple.asp>

Shaw, Zed. A. (2017). *Learn Python 3 the Hard Way.* Addison-Wesley Professional.

## Formative Assessment

1. What is the output of the following program?

a = 1, 4, 9, 16

a[2] = 3

print(a)

1. 1, 3, 9, 16
2. 1, 4, 3, 16
3. 1, 4, [3, 9], 16
4. Syntax Error
5. What is the output of the following program?

a = []

for i in range(0, 5):

a = a + [i \*\* 2]

print(a)

* 1. [0, 1, 2, 3, 4]
  2. [0, 1, 4, 9, 16, 25]
  3. [0, 1, 4, 9, 16]
  4. [0, 1, 2, 3, 4, 5]

1. Which of the following syntaxes will have “north” as output?
   1. d = {“north”: 2, “south”, -2}

dkeys = list(d.keys())

print(dkeys[0])

* 1. d = {“north”: 2, “south”, -2}

dkeys = list(d.keys())

print(d[0])

* 1. d = {“north”: 2, “south”, -2}

dkeys = list(d.items())

print(dkeys[0])

* 1. d = {“north”: 2, “south”, -2}

dkeys = list(d.values())

print(dkeys[0])

1. Which of the following statements is correct regarding the properties of dictionaries?
   1. The values of a dictionary can only be integers, floats, and strings.
   2. The keys of a dictionary cannot be modified.
   3. The curly brackets around a dictionary can be omitted.
   4. The elements of a dictionary must be separated by semi-colons.
2. What is a Python function?
   1. A stand-alone Python program
   2. A dictionary with the module names of the standard library as keys
   3. An object specifically designed for machine learning operations
   4. A Python routine code that is reusable for a particular task
3. Which of the following methods does not apply to string variables?
   1. .get()
   2. .lower()
   3. .replace()
   4. .strip()
4. Which component is not optional in a user-defined function?
   1. Function name
   2. Arguments
   3. Loops
   4. Return value
5. What is not a good habit when implementing user-defined functions in a program?
   1. We should only implement user-defined functions whenever it is sensible to combine multiple functions into one.
   2. We should only implement user-defined functions when we need to carry out the same routine repeatedly in our program.
   3. We should only implement user-defined functions when we can reduce our main program to some syntaxes merely for calling the functions.
   4. We should only implement user-defined functions when we have recurrent tasks that create certain output objects needed for the further parts of the main program. And the creation of such objects requires some input arguments from the previous parts of the main program.
6. Which of the following statements is correct when using alias for importing package/module?
   1. We can use both the original name and alias to refer to the package/module in our program.
   2. An alias is optional and can be omitted if we are comfortable to work with the original package/module name.
   3. An alias must be shorter than the original package/module name.
   4. If only a single module from a package is imported, the alias refers to the package and not the module.
7. Which command is used to install a new Python package?
   1. > pip3 setup package\_name
   2. > pip3 install package\_name --upgrade
   3. > pip3 install package\_name
   4. > pip3 import package\_name

## Suggested Solutions

1. What is the output of the following program?

a = 1, 4, 9, 16

a[2] = 3

print(a)

1. 1, 3, 9, 16

*Incorrect. Since a is a tuple, it is immutable, and we cannot assign a new value to one of the elements in it.*

1. 1, 4, 3, 16

*Incorrect. Since a is a tuple, it is immutable, and we cannot assign a new value to one of the elements in it.*

1. 1, 4, [3, 9], 16

*Incorrect. Since a is a tuple, it is immutable, and we cannot assign a new value or change the type of the elements in it.*

1. **Syntax Error**

**Correct. Since** a **is a tuple, it is immutable. If we try to reassign a value to one of the elements in it, we will get an error message from Python.**

1. What is the output of the following program?

a = []

for i in range(0, 5):

a = a + [i \*\* 2]

print(a)

* 1. [0, 1, 2, 3, 4]

*Incorrect. In each iteration, i square will be put into a list and then appended to a. And the values of i are 0, 1, 2, 3, 4.*

* 1. [0, 1, 4, 9, 16, 25]

*Incorrect. Though i square is taken here, the last value is out of range since the for-loop will stop running at i = 4.*

* 1. **[0, 1, 4, 9, 16]**

**Correct. The for-loop stops running at i = 4 and the values in a are** i **square.**

* 1. [0, 1, 2, 3, 4, 5]

*Incorrect. The last value is out of range, and the values in a are i and not i square.*

1. Which of the following syntaxes will have “north” as output?
   1. **d = {“north”: 2, “south”, -2}**

**dkeys = list(d.keys())**

**print(dkeys[0])**

**Correct. The .keys() method is used to extract the keys of a dictionary. After converting the object to a list, we subset its item with the index 0, which is the first key in this case: “north”.**

* 1. d = {“north”: 2, “south”, -2}

dkeys = list(d.keys())

print(d[0])

*Incorrect. In the print() function, it refers to the object d and not dkeys. And since d is dictionary, we can only access its element by the keys and not the indices.*

* 1. d = {“north”: 2, “south”, -2}

dkeys = list(d.items())

print(dkeys[0])

*Incorrect. The .items() method is used to extract the keys and values of a dictionary and store each pair of them in a tuple. dkeys[0] will return (“north”, 2) as its result.*

* 1. d = {“north”: 2, “south”, -2}

dkeys = list(d.values())

print(dkeys[0])

*Incorrect. The .values() method is used to extract the values of a dictionary. After converting the object to a list, we subset its item with the index 0, which is the first value 2 in this case, and not the key “north”.*

1. Which of the following statements is correct regarding the properties of dictionaries?
   1. The values of a dictionary can only be integers, floats, and strings.

*Incorrect. The value of a dictionary can also be tuples, lists or other object types.*

* 1. **The keys of a dictionary cannot be modified.**

**Correct. The keys of a dictionary cannot be modified. We can only modify a specific key indirectly by adding a new pair of key and value to the dictionary and delete the old pair from it.**

* 1. The curly brackets around a dictionary can be omitted.

*Incorrect. A Python dictionary must be wrapped by a pair of curly brackets.*

* 1. The elements of a dictionary must be separated by semi-colons.

*Incorrect. The elements of a dictionary must be separated by commas.*

1. What is a Python function?
   1. A stand-alone Python program

*Incorrect. A function cannot be a stand-alone program. We must have a main program to call a function for a specific task.*

* 1. A dictionary with the module names of the standard library as keys

*Incorrect. A function is a routine program and not a dictionary.*

* 1. An object specifically designed for machine learning operations

*Incorrect. A function is not an object, it is a routine program.*

* 1. **A Python routine code that is reusable for a particular task**

**Correct. A function is a chunk of code that performs a particular task and can be called endlessly by the main program.**

1. Which of the following methods does not apply to string variables?
   1. **.get()**

**Correct. It is a method for dictionary. It extracts the value of a particular dictionary key.**

* 1. .lower()

*Incorrect. It is a method for string variables. It converts a string to lower case.*

* 1. .replace()

*Incorrect. It is also a method for string variables. It replaces part of a string by another string.*

* 1. .strip()

*Incorrect. It is a method for string variables too. It removes all empty spaces at the beginning and at the end of a string.*

1. Which component is not optional in a user-defined function?
   1. **Function name**

**Correct. A user-defined function must have a function name.**

* 1. Arguments

*Incorrect. Arguments are input from the main program to the function and they are optional. A function can still be carried out without arguments provided.*

* 1. Loops

*Incorrect. A function does not need loops for its functionality. Its instructions can contain loops, but it would still work without them.*

* 1. Return value

*Incorrect. A function can also be carried out even if it does not return any value to the main program.*

1. What is not a good habit when implementing user-defined functions in a program?
   1. We should only implement user-defined functions whenever it is sensible to combine multiple functions into one.

*Incorrect. It is a good habit indeed. If multiple functions are applied to a single object step-by-step, we can create a function and give a sensible name to it to simplify our main program.*

* 1. We should only implement user-defined functions when we need to carry out the same routine repeatedly in our program.

*Incorrect. It is recommended to use user-defined functions to replace recurrent routines in the main program.*

* 1. **We should only implement user-defined functions when we can reduce our main program to some syntaxes merely for calling the functions.**

**Correct. It is not a good habit to “outsource” chunks of code to various functions just for the sake of simplifying the main program. It usually does not simplify the whole program at all and makes the debugging more difficult since we must jump between the functions on checking the source of error.**

* 1. We should only implement user-defined functions when we have recurrent tasks that create certain output objects needed for the further parts of the main program. And the creation of such objects requires some input arguments from the previous parts of the main program.

*Incorrect. It is a good habit to implement functions when they can really carry out recurrent tasks and return some objects/values to the main program based on some input arguments originated from the previous parts of the main program.*

1. Which of the following statements is correct when using alias for importing package/module?
   1. We can use both the original name and alias to refer to the package/module in our program.

*Incorrect. Once we have imported a package/module with an alias, we must use the alias to refer to it in our program.*

* 1. **An alias is optional and can be omitted if we are comfortable to work with the original package/module name.**

**Correct. The alias part is not compulsory. If we have a concise original package/module name, there is no reason to use an alias at all.**

* 1. An alias must be shorter than the original package/module name.

*Incorrect. There is no guideline for the length of the alias. It can even be longer than the original package/module name.*

* 1. If only a single module from a package is imported, the alias refers to the package and not the module.

*Incorrect. The alias refers to the module and not the package in the “from … import … as …” syntax.*

1. Which command is used to install a new Python package?
   1. > pip3 setup package\_name

*Incorrect. It should be “install” and not “setup” following “pip3”.*

* 1. > pip3 install package\_name --upgrade

*Incorrect. The “--upgrade”-option is only used for upgrading/updating existing packages.*

* 1. **> pip3 install package\_name**

**Correct. “install” should be following “pip3” for installation.**

* 1. > pip3 import package\_name

*Incorrect. It should be “install” and not “import” following “pip3”.*