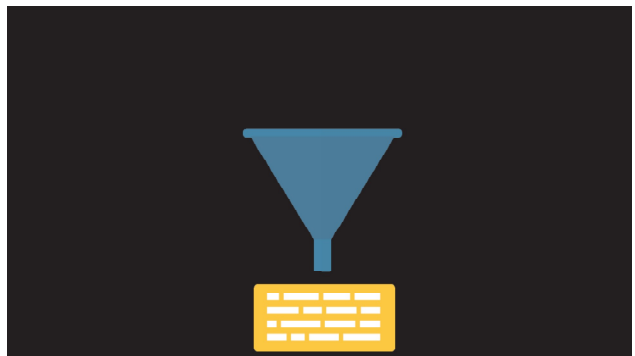




# UNIT 5

## Data Structures





## ● Overview

In this unit, you will learn about data structures. The types of data structures covered in this unit are lists and dictionaries. You will be able to demonstrate how to create, read and write data. You will also be able to perform various other operations on the data structures. The range of skills and programming techniques you learned in term 2 will be useful when you work with data structures. Some of these techniques include selection, repetition, and the use of variables.

## Key terms

Table 36

abstraction	removing unnecessary detail from a problem to generalise or create a model of a physical object or process
data type	a classification of data based on the values it can take
data structure	a collection of related data stored in an organised manner
array	a type of data structure for storing the same data type organised in rows and columns
list	a type of sequential data structure made up of an ordered collection of smaller data types; It can also be a collection of other lists
dictionary	a data structure organised in a key-value manner; the keys are associated with values and the keys are used to access the values
index	a way of accessing or identifying elements in an array or list
key	a way of accessing or identifying elements in a dictionary
value	a single unit of data in a dictionary

## Learning outcomes

After completing this unit, you will be able to:

- ① describe and use appropriate data types.
- ① describe data structures and their associated operations.
- ① test and evaluate computer programs.
- ① design computer programs using sequencing, selection and repetition statements.

## Data structures

In this section, you will learn about lists and dictionaries.

### Python list

Khalid, a grade 10 student in Ras al Khaimah, has designed a Python program for his computer science teacher. Khalid wants his teacher to use the program to record the marks for the class.

```
1 print("Welcome to the marks entry system")
2
3 gradeStudent1 = 15
4 gradeStudent2 = 80
5 gradeStudent3 = 9
```

Figure 119

Khalid has used several variables to store the marks of each student.

Khalid has a problem. There are 28 students in the class. He will need to create 28 different variables for each student such as:

```
gradeStudent4 = 57
```

```
gradeStudent5 = 78
```

```
gradeStudent6 = 63
```

Khalid's teacher would like to use the program for all his classes. He has 121 students in all his classes. Can you imagine the number of variables Khalid needs to add to his program!



Figure 120

This is where data structures are useful. We use data structures to store a collection of related data efficiently.

You can create a type of data structure called a *list*. A list will store all the grades for each student in the class one after the other in the same area in memory.

In Python, you can declare and assign values to a list as follows:

```
7
8 studentGrades = [15, 80, 9, 57, 78, 63, 73, 98, 91, 46, 98, 35]
9
```

Figure 121

Each value in the list has its own memory location. The locations are arranged one after the other.

Here is a representation of the first seven elements in the **studentGrades** list.

<b>element</b>	15	80	9	57	78	63	73
<b>index</b>	0	1	2	3	4	5	6

The first element has an index value of 0.

You can print the whole list using the code:

```
9
10 print(studentGrades)
11

>>>
>>> studentGrades=[15, 80, 9, 57, 78, 63, 73, 98, 91, 46, 98, 35]
>>> print(studentGrades)
[15, 80, 9, 57, 78, 63, 73, 98, 91, 46, 98, 35]
>>> |
```

Figure 122

You use the index to access any element in the list. To check the value of the first element in the list, you can print it using its index as follows:

```
11
12 print(studentGrades[0])
13
```

Figure 123

The results are shown below:

You can also use a similar statement to print the value of the sixth element in the list.

```
>>>
>>> print(studentGrades[0])
15
>>> |
```

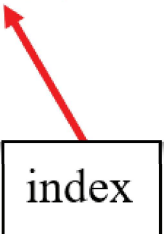


Figure 124

You can also print the sixth element using the statement below:

```
13
14 print(studentGrades[5])
15
16
```

Figure 125

Here are the results:

```
>>> print(studentGrades[5])
63
>>>
```

Figure 126

Use the '=' to set the values for specific elements in the list.

```
15
16 studentGrades[4] = 33
17
```

Figure 127

The results before and after changing the value are shown below.

```
>>> print(studentGrades[4])
78
>>>
>>> studentGrades[4] = 33
>>>
>>> print(studentGrades[4])
33
>>>
```

Figure 128

You can also assign values directly using the keyboard:

```
17
18 studentGrades[6] = input("Enter the grade:")
19
```

Figure 129

Here are the results:

```
>>> print(studentGrades[6])
73
>>> studentGrades[6] = input("Enter the grade:")
Enter the grade:91
>>>
>>> print(studentGrades[6])
91
>>>
```

Figure 130

Be careful about the index values when reading or setting values for elements. Your program will crash if you try to access a memory location outside the list. For example:

```
20
21 print(studentGrades[77])
22
```

Figure 131

This will give an error:

```
>>>
>>> print(studentGrades[77])
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    print(studentGrades[77])
IndexError: list index out of range
>>>
```

Figure 132



## Creating lists





## Reading and writing data to lists

You have already learned about the **for** and **while** loops. Loops are very useful when you need to set, change or read data in a list. The following two examples are brief reminders of the 2 types of loops you covered in term 2.

Below is an example of a **for** loop.

```
for x in range(0, 9, 1):
    square = x**2
    print(x, "squared is:", square)
```

Figure 133

Here are the results after running the code:

```
0 squared is: 0
1 squared is: 1
2 squared is: 4
3 squared is: 9
4 squared is: 16
5 squared is: 25
6 squared is: 36
7 squared is: 49
8 squared is: 64
>>>
```

Figure 134

The code block below is an example of a **while** loop. The code block will be executed until the user enters a temperature that is higher or lower than the limits.

```
carTemp = 23.5

while carTemp < 25.0 and carTemp > 21.4:
    print("Enter the temperature in the car")
    carTemp = input("#>")
    carTemp = float(carTemp)

print("Turn the heating or aircon on!")
```

Figure 135

The results are shown below:

```
Enter the temperature in the car
#>22
Enter the temperature in the car
#>24
Enter the temperature in the car
#>26
Turn the heating or aircon on!
>>>
```

Figure 136

Loops are well suited for the repetitive operations usually needed to set, change, or read values in data structures.

## Using a loop to set values

The example below shows how a loop is used to set the values in a list from 10, 20, 30, 40, up to 100.

```
1 #Setting values in a list using a for loop
2 percentNumbers = [0,0,0,0,0,0,0,0,0,0]
3 for num in range(0,10,1):
4     percentNumbers[num] = (num + 1) * 10
5
6 print(percentNumbers)
```

Figure 137

The variable **num** is used to store the values from the **range** function. On line 4, **num** is then used as an index in the code block to point to each element in the list.

**num** is also used in the calculation to set the value for each element. The index for the list starts from 0. You can see the results below.



Figure 138

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
>>>
```

Figure 139

## Using user input to set values

We use the **input()** function to set the values to what the user enters.

```
1 #Using user input to set values
2 percentNumbers = [0,0,0,0,0,0,0,0,0,0]
3
4 for num in range(0,10,1):
5     print("Enter value number", num + 1)
6     value = input()
7     value = int(value)
8     percentNumbers[num] = value
9
10 print(percentNumbers)
```

Figure 140

This code will produce the following output:

```
Enter value number 1
34
Enter value number 2
22
Enter value number 3
57
Enter value number 4
2
Enter value number 5
1
Enter value number 6
88
Enter value number 7
7
Enter value number 8
32
Enter value number 9
90
Enter value number 10
100
[34, 22, 57, 2, 1, 88, 7, 32, 90, 100]
>>>
```

Figure 141

## *Printing the values in a list*

So far, you have been printing entire lists at once. You can use a loop to print one value at a time. You can use the index to select and print only a few values from a list.

```
1 #Printing one value at a time
2 percentNumbers = [0,0,0,0,0,0,0,0,0,0]
3
4 for num in range(0,10,1):
5     percentNumbers[num] = (num + 1) * 10
6
7 for number in percentNumbers:
8     print(number)
9
```

Figure 142

The output is shown below.

```
10
20
30
40
50
60
70
80
90
100
>>>
```

Figure 143

## Creating an empty list

Sometimes you neither have the data nor the size of your list at the start of your program. In such cases, you can create an empty list first, and then add elements and data to the list later. You can use the **append()** function to add data to your list.

```
1 #Adding data to an empty list
2 percentNumbers = []
3 print("Enter the marks. Type any letter to exit.")
4 num = "0"
5
6 while num.isdigit():
7     num = input("Enter the mark:")
8     if num.isdigit():
9         mark = float(num)
10        percentNumbers.append(mark)
11
12 print(percentNumbers)
```

append()  
used to  
add data

Figure 144

Here is the output:

```
Enter the marks . Type any letter to exit .
Enter the mark:98
Enter the mark:45
Enter the mark:33
Enter the mark:y
[98.0, 45.0, 33.0]
>>>
```

Figure 145



## Using loops to set and read values in lists



## Creating lists from other lists

So far, we have been able to create lists within a program and by using data entered by a user. There are other ways to make lists. Let us look at some of these methods.

### Copying a list

You can make a copy of a list if you need to keep the original list the same and modify the copy.

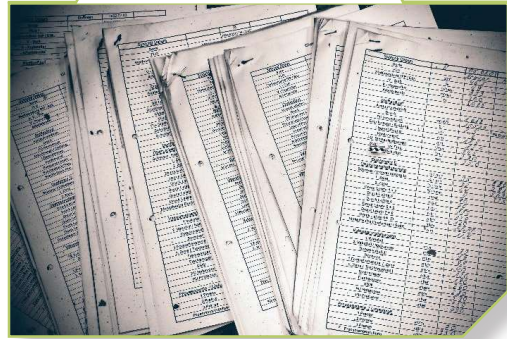


Figure 146

1. Use the **copy ()** function.

```
1 #Copying a list - 1
2 marks1 = [73, 61, 22, 89, 45, 70, 37, 64, 70, 61, 45]
3 marks2 = marks1.copy()
4 print(marks2)
5
6 marks1[3] = 2048
7
8 print(marks1)
9 print(marks2)
10
```

Figure 147

Here are the results. You can see the fourth element in the first list has changed but is still the same in the second one.

```
[73, 61, 22, 89, 45, 70, 37, 64, 70, 61, 45]
[73, 61, 22, 2048, 45, 70, 37, 64, 70, 61, 45]
[73, 61, 22, 89, 45, 70, 37, 64, 70, 61, 45]
```

Figure 148

## Creating sublists

You can create a smaller list known as a sublist from longer ones.

You must specify a range as shown in the following code to create a sublist with only a few elements from the long list.

```
1 #Creating a sublist
2 marks = [73, 61, 22, 89, 45, 70, 37, 64, 70, 61, 45]
3 marks3 = marks[1:5]
4 print(marks3)
```

Figure 149

The statement will create the sublist shown below:

```
▶ ↑ [61, 22, 89, 45]
```

Figure 150

## Joining lists

You can also create longer lists from shorter lists using the following statements.

1. To repeat the same elements, use the '\*' operator:

```
1 #Repeating elements
2 alphabet = ["a", "b", "c", "d"]
3 repeatAlphabet = alphabet * 3
4
5 print(alphabet)
6 print(repeatAlphabet)
```

Figure 151

This will create a longer list with repeated values from the original list.

```
['a', 'b', 'c', 'd']
['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']
```

Figure 152



2. You can also use the '+' to merge or link lists:

```
1 #Joining 2 lists
2 animals1 = ["rabbit","cow","buffalo"]
3 animals2=["tiger","wolf","lion","leopard","fox","cat"]
4 animals3 = animals1 + animals2
5 print(animals3)
```

Figure 153

You can see the results below.

```
['rabbit', 'cow', 'buffalo', 'tiger', 'wolf', 'lion', 'leopard', 'fox', 'cat']
```

Figure 154



## Creating lists from other lists



## Useful list operations

Below are other functions and operations you will find useful when working with lists.

### *Deleting an element from a list*

You use the **del ()** function to remove elements from a list. Use the statement below to delete 'lion' from the list.

Remember that the index starts at 0. You must use a valid index value before you attempt to delete an element.

```
1: #Deleting an element
2: cats=["tiger", "cheetah", "lion", "jaguar", "lynx", "puma"]
3: del(cats[2])
4: print(cats)
5:
```

Figure 155

Can you predict the animal that will be deleted from the list?

## The length of a list

You use the `len()` function to get the length of a list.

```
10 #Deleting an element
11 cats=["tiger","cheetah","lion","jaguar","lynx","puma"]
12 del(cats[2])
13 print(cats)
```

Figure 156

The results are shown below.

```
>>>
===== RESTART: C:/Python pro:
6
>>>
>>>
```

Figure 157

## Checking if a value is in a list

To find out if an element is in a list, you can use the `in` keyword as follows:

```
12 #Checking if a value is in the list
13 cats=["tiger","cheetah","lion","jaguar","lynx","puma"]
14 if "jaguar" in cats:
15     print("Yes, the jaguar is in there!")
16 else:
17     print("Oh no, the jaguar is missing!")
```

Figure 158

Here are the results:

```
Yes, the jaguar it is in there !
>>>
```

Figure 159

Change the code to search for a leopard. What is the result?

## *Finding the maximum and minimum values in a list*

To find the elements in a list with the largest and smallest values, use the **max()** and **min()** functions.

```
1 #Finding the largest and smallest values
2 marks = [73, 61, 22, 89, 45, 70, 37, 64, 70, 61, 45]
3 largest = max(marks)
4 smallest = min(marks)
5 print("The highest marks was:", largest)
6 print("The lowest marks was:", smallest)
7
```

Figure 160

This code produces the following results:

```
The highest marks was : 89
The lowest marks was : 22
>>>
```

Figure 161

In the marks list, change 73 to -11 and 64 to 203. Run the code and state your results.

## Python dictionary

A dictionary is a data structure that lets you store key-value pairs. We use a key to access the value rather than the index. It is like a language dictionary. You use the word (= key) to find the meaning (= value).

In the Python dictionary, you will use a key to find the value.

From the list we made in the first chapter, we only had the grades. We can add names to this data to make it more meaningful. Here is how it looks:

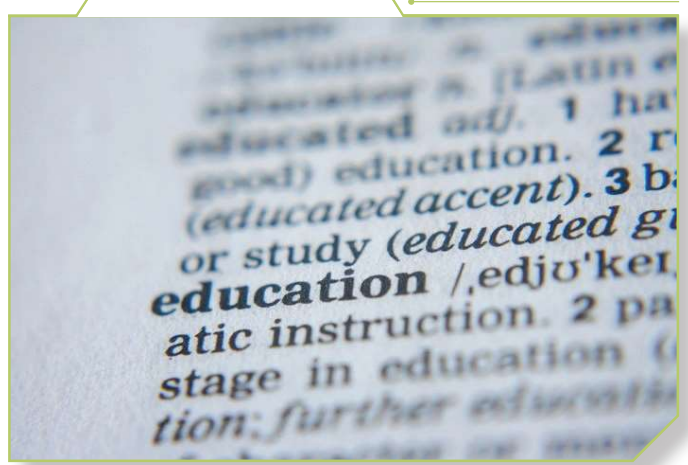


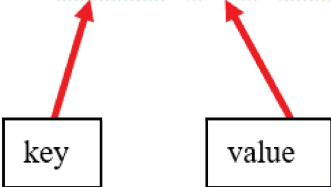
Figure 162

Table 42

Key	Value
Mane	15
Rashid	80
Saaed	9
Eman	57
Razan	78
Mariam	63
Haitham	73
Hamid	98
Esraa	91

This is how you define a dictionary in Python:

```
#Defining a dictionary  
grades = {"Mane":15, "Rashid":80, "Saaed":9, "Eman":57, "Razan":78}  
print(grades)
```



The diagram shows two boxes labeled 'key' and 'value'. A red arrow points from the 'key' box to the string 'Rashid' in the dictionary definition. Another red arrow points from the 'value' box to the integer '80' in the dictionary definition.

Figure 163

The results of running this code are shown below.

```
{'Mane': 15, 'Rashid': 80, 'Saaed': 9, 'Eman': 57, 'Razan': 78}  
  
Process finished with exit code 0
```

Figure 164

## Reading and writing data in a dictionary

You used the index values to access the data in lists. You will use keys to access the values in a dictionary. In the **grades** dictionary above, the names of the students are the keys.

### 1. Reading values from a dictionary

Use the key to read the value linked with any key in the dictionary. The example below shows how to read the data.

```
#Reading data from a dictionary  
grades = {"Mane":15, "Rashid":80, "Saaed":9, "Eman":57, "Razan":78}  
  
print(grades["Rashid"])
```

Figure 165

Here are the results.

```
80  
  
Process finished with exit code 0
```

Figure 166

How would you print Razan's grade?



## 2. Setting values in a dictionary

Again, you need to use the key to set or change values in a dictionary. In the example below, the value for the key **Saaed** is changed from 9 to 93.

```
10: #Changing values in a dictionary
11: grad={"Mane":15,"Rashid":80,"Saaed":9,"Hadif":57,"Razan":78}
12: print(grad)
13: grad["Saaed"] = 93
14: print(grad)
```

Figure 167

You can see the results below.

```
9
93

Process finished with exit code 0
```

Figure 168



## Creating, reading, and writing data in dictionaries



## Useful operations for dictionaries

In addition to creating, reading and setting values in a dictionary, this section introduces more operations which you will find useful.

### 1. Deleting dictionary entries

You can delete any key and its value using the **del ()** function:

```
1 #Deleting dictionary entries
2 emp={"name":"Mariam","position":"trainee","age":19,"car":"VW"}
3 print(emp)
4 del(emp["car"])
5 print(emp)
```

Figure 169

Below, you can see the results before and after deleting the **car** key-value pair from the **emp** dictionary.

```
>>>
>>> emp={"name":"Mariam","position":"trainee","age":19,"car":"VW"}
>>> print(emp)
{'name': 'Mariam', 'position': 'trainee', 'age': 19, 'car': 'VW'}
>>> del(emp["car"])
>>>
>>> print(emp)
{'name': 'Mariam', 'position': 'trainee', 'age': 19}
>>>
```

Figure 170

## 2. Adding a new entry

Use the **update ()** function to add a new key and value to a dictionary:

```
1 #Updating dictionary entries
2 emp={"name":"Samira","position":"trainee","age":19}
3 print(emp)
4 emp.update({"Salary":47,"Gender":"F"})
5 print(emp)
```

Figure 171

Here are the results:

```
>>> print (emp)
{'name': 'Samira', 'position': 'trainee', 'age': 19}
>>>
>>>
>>>
>>> emp.update ({"Salary":47,"Gender":"F"})
>>> print (emp)
{'name': 'Samira', 'position': 'trainee', 'age': 19, 'Salary': 47, 'Gender': 'F'}
>>>
```

Figure 172

### 3. Listing all the keys in a dictionary

Use the **keys ()** function to get a list of all the keys in a dictionary.

```
1 #Listing all the keys
2 emp={"Name":"Hessa","Title":"CEO","age":19, "Town":"Al Hilo"}
3 print(emp)
4 print(emp.keys())
5
```

Figure 173

This code will produce a list of keys as follows:

```
>>>
>>> emp= {"Name": "Hessa", "Title": "CEO", "age": 19, "Town": "Al Hilo"}
>>> print(emp)
{'Name': 'Hessa', 'Title': 'CEO', 'age': 19, 'Town': 'Al Hilo'}
>>>
>>> print(emp.keys())
dict_keys(['Name', 'Title', 'age', 'Town'])
>>>
```

Figure 174

### 4. Listing the values in a dictionary

Use the **values ()** function to list all the values in a dictionary.

```
10 #Listing all the keys
11 emp={"Name":"Hessa","Title":"CEO","age":19, "Town":"Al Hilo"}
12 print(emp)
13 print(emp.keys())
14
```

Figure 175

All the values in the dictionary have been listed as shown in the results below.

```
>>>
>>> emp2={"Name": "Ahmed", "Title": "CIO", "Age": 92, "Town": "Sharjah"}
>>> print(emp2)
{'Name': 'Ahmed', 'Title': 'CIO', 'Age': 92, 'Town': 'Sharjah'}
>>>
>>>
>>> print(emp2.values())
dict_values(['Ahmed', 'CIO', 92, 'Sharjah'])
>>>
```

Figure 176

You can only use a key in the same dictionary once.

Your program will crash if you try to get values using a key that does not exist in the dictionary.

```
>>>
>>> print(emp2)
{'Name': 'Ahmed', 'Title': 'CIO', 'Age': 92, 'Town': 'Sharjah'}
>>>
>>> print(emp2["Town"])
Sharjah
>>>
>>> print(emp2["Salary"])
Traceback (most recent call last):
  File "<pyshell#173>", line 1, in <module>
    print(emp2["Salary"])
KeyError: 'Salary'
>>>
```

Figure 177



## Dictionary operations



## Completed unit objectives

You have completed the introduction to computer programming. You should now be able to do the following:

- ❖ Use appropriate data types.
- ❖ Describe data structures.





End-of-unit activities



Khalifa Fund for Enterprise Development – Data structures I



Khalifa Fund for Enterprise Development – Data structures II



Student reflection