Python Programming

for Engineers and Scientists

Irfan Turk

Edited by
Ibrahim Emre Celikkale

Printed in the United States of America.
ISBN-13: 978-1543173833
ISBN-10: 1543173837

PREFACE

Python is a high level programming language which has an increasing reputation during the last few decades. It was originally created by Guido van Rossum, and first released in February 1991. The language was developed under an open source license administered by the Python Software Foundation (PSF). It can be freely distributed and used by programmers under many operating systems.

The programming language has an easy to use syntax that makes the language very straightforward to learn and grasp. Unlike some other famous programming languages such as C, C++, or Java, there is no need to pre-define the data types of the variables. The variables are created as they are used in Python, which often requires less coding to carry out the tasks.

Python addresses the problems of many people from a wide range of areas including all fields of engineering branches, mathematicians, physicists, computer scientists, and even programmers from social sciences. It has applications regarding web and internet development, databases, software development, scientific and numerical computing, digital image processing, graphical user interfaces and some others that attract many users from different backgrounds and areas of study.

Although this book is aimed at teaching Python as a computer programming course for one semester in colleges and universities, it is also a nice resource and textbook for individual learners providing a complete collection of written and tested codes along with applications. If there arise any questions or recommendations, please feel free to send an e-mail to iturkbooks@gmail.com.

Teaching Python as a programming language is explained in the first part of the book. In the second part, object-oriented programming and working with databases are dealt with as part of an attempt for software engineering concentration. In the third and the last part, some of the selected topics that are essential for engineers and scientists are covered. The topics considered in the last part include Matrix Algebra, Plotting Graphics, Symbolic Calculations, Introduction to Statistics, Numerical Methods, Digital Image Processing, and Graphical User Interfaces.

The author of this book has been teaching mathematics and computer programming courses in secondary schools and universities for more than 10 years. In this regard, the

book is prepared considering the pedagogical approach for teaching a subject. The book provides 117 illustrative and instructive examples including the solutions along with the codes. The reader can easily modify and update the presented codes depending on their needs.

ABOUT THE AUTHOR

Dr. Irfan Turk is a mathematician who has worked as a math and computer programming teacher in middle schools, high schools, and universities for more than 10 years. Dr. Turk earned his B.S. degree from Fatih University, M.S. degree from The University of Texas at Arlington, and PhD. from Istanbul University all in mathematics. His research interests include but are not limited to numerical solutions of differential equations, scientific computing, mathematical modeling, and programming in Python and MATLAB.

ACKNOWLEDGMENTS

I would like to mention and thank a few people who helped me in the preparation of this book.

First of all, I would like to express my special appreciation and grateful thanks to my colleague and friend Ibrahim Emre Celikkale, who accepted to be the editor of this book. His recommendations regarding the language of the book, and its presentation were of supreme importance to me.

Next, I would like to mention a few names and give credits about their nice works from which I learned a lot about Python. I would like to thank Dr. Charles Severance from the University of Michigan for his instructive courses about Python on youtube.com and coursera.org. I would also like to thank Derek Banas, Bucky Roberts, and Mustafa Murat Coskun for their helpful and instructive videos about Python on youtube.com.

And finally, I would like to thank my dear wife, Maksude, for her unwavering support and understanding, and for always being there.

Irfan Turk, PhD

Contents

Part I

FUNDAMENTALS OF PYTHON

Chapter 1

INTRODUCTION TO PYTHON

Python 3 programming language will be explained in this work. Python version 3.5.2 is used to test the codes and scripts.

Python holds an open source license (administered by Python Software Foundation) which makes the language free to use and accessible to anyone. The software has a simple syntax that yields shorter programs compared to other popular languages such as C, C++, or Java. Python can be used in a wide range of applications due to its large standard library and remarkable number of modules. Therefore, it attracts many users and is one of the best choices for engineers and scientists in various scientific and computational applications.

In this chapter, installation of the Python software will be explained as an introduction. Next, we will take a look at the modules of the language. Then, the following topics will be handled: Using Python as a calculator, variables and expressions, overview of data types, operators, built-in functions, and formatting outputs.

1.1 Installing Python

In this section, we will show how to install Python on a PC. One can find the most recent version of the software at https://www.python.org/downloads/. As will be seen on the web page, it is available for various operating systems. After the version that fits your operating system is selected and downloaded, it can be installed on your computer.

While installation, it is highly recommended to click on "Add Python 3.x to PATH" check button. Here, 3.x is the version that you try to install. If this option is checked, one can run programs from the command window after the installation of Python (or Python interpreter). Once the command window is opened, type python, and then Python shell will be ready to start programming. You can type your commands next to "⋙" symbol, which is called the "prompt".

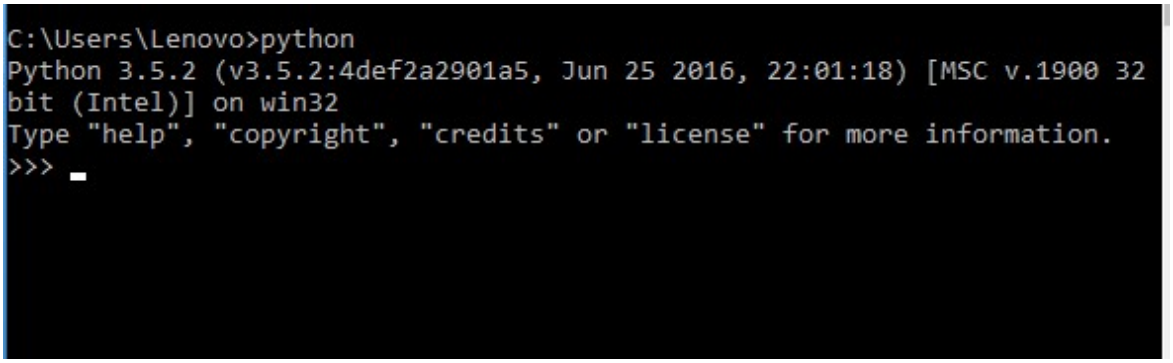

Figure 1.1: Running Python from Command window in Windows operating system.

After installing Python, one can also click on IDLE, which stands for **I**nteractive **D**eve**L**opment **E**nvironment, to run the codes.

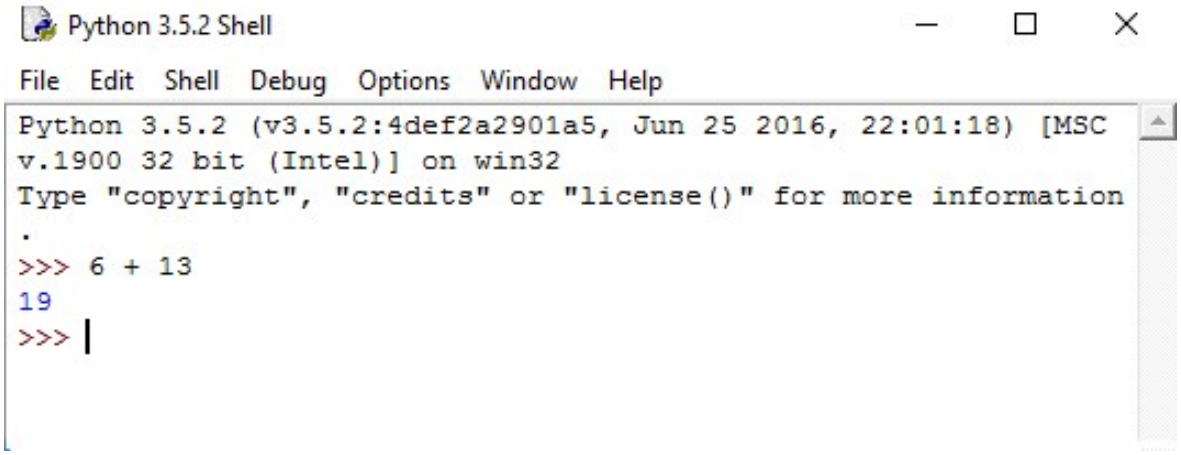

Figure 1.2: IDLE running on a Windows operating system

The IDLE environment having extra buttons on it is also called an interactive shell.

In the Python IDLE , if you click on File ⇒ New File button, the following editor will be popped up.

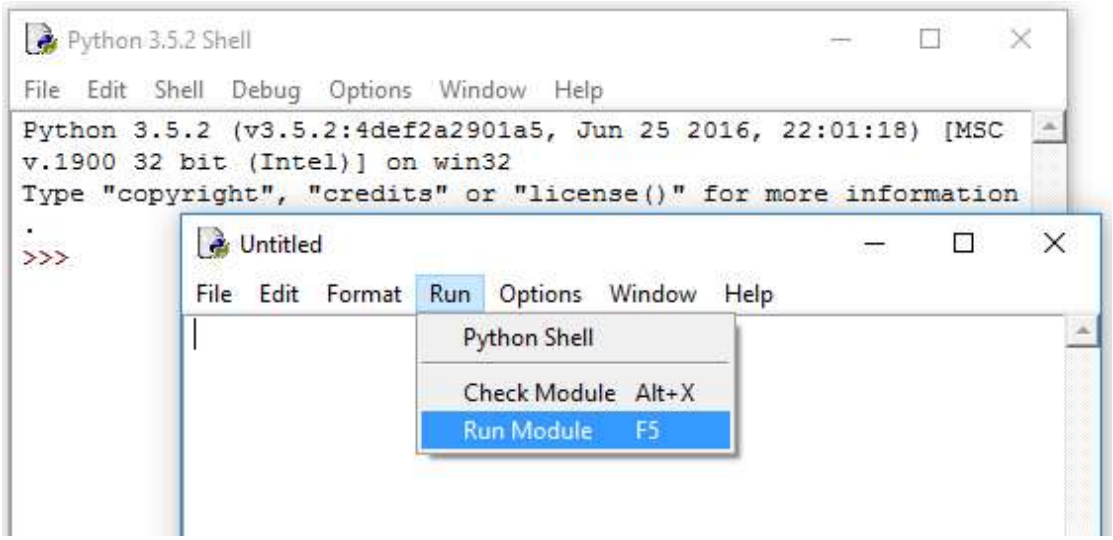

Figure 1.3: Python IDLE editor

Coding can be achieved using the editor, and then, one can save them by adding the extension .py (Name.py) making it a Python file. Then, click on Run ⇒ Run Module buttons to run the codes within the editor.

There are several nice editors available to edit and run new codes. If the keywords "Python Editor" is searched on Google, one can find plenty of them. Some of these editors have nice features such as coloring the scripts, assisting the user with completing the codes, etc. The reader is recommended to use PyCharm and Pydev as a Python editor.

1.2 Using Modules in Python

A module is a file composed of Python definitions and expressions. Modules are very useful packages that set the syntax of built-in functions available with that module.

Throughout this book, various modules will be used depending on the subject. Before using a module, it should be imported to the program.

After importing a module, i.e. math, one can find out some information regarding the module by using the **help()** function as below.

```
>>> import math
>>> help(math)
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.
...
...
```

To check the names that are defined by a module, **dir()** function may be used after importing the module as shown below.

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot',
'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin',
```

```
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>>
```

In order to use the functions that are defined within the module, it is necessary to write the module name followed by a dot, and the name of the function.

As an example, if the user wants to calculate the value of $\sin(\frac{\pi}{2})$, he/she may type

```
>>> import math
>>> math.sin(math.pi/2)
1.0
>>>
```

There is another way of adding $\sin(x)$ function and a constant π as follows:

```
>>> from math import sin
>>> from math import pi
>>> sin(pi/2)
1.0
>>>
```

In this case, it is required to use the words "from" and "import" to calculate the same value.

1.3 Using Python as a Calculator

Python can be used as a big calculator, as well. In Python interactive shell, if the user types $\ggg 2 + 3$ at the prompt and hit enter, the result is shown on the screen. Similar operations can be carried on in a similar manner. Several examples of these operations are listed below:

```
>>> 2 + 3
5
>>> 5 - 8
-3
```

```
>>> 7 * 4
28
>>> 4 / 5
0.8
>>> 2**3
8
>>> 24//5
4
>>>
```

In the above example, double star "**" represents the power operator. It corresponds to **pow()**, a built-in math function in Python.

Another operator is about rounding the number to the nearest smallest integer which can be applied by using double slashes. It corresponds to **floor()**, a built-in math function. If 24 is divided by 5, the result is 4.8. If 4.8 is rounded to the smallest integer, the result is obtained as 4.

To use such other operations as taking the square root of a number, or using exponential functions, one needs to import the math module.

Example 1.1 Find the result of $5 + \frac{7}{9} * \cos(\pi)$ with Python.

Solution. The following code may be used to find the result.

```
>>> import math
>>> 5 + (7 / 9) * math.cos(math.pi)
4.222222222222222
>>>
```

◀

Once a module is imported, it is not needed to import it again for additional usages or calculations during the same session.

Example 1.2 Find the result of $y = \frac{100}{e^4} + 4\sqrt{23}$.

Solution. The following code can be typed at prompt to find the result.

```
>>> (100 / (math.exp(4))) + math.sqrt(23)
6.627395412186138
>>>
```

◀

1.4 Variables and Expressions

After calculating a result, one may need to use the value of that result for subsequent calculations as well. In that case, it is a better idea to store these values under some structures called variables. We give names to functions, modules, or variables that certain values are assigned. In the literature, these names are called *identifiers*. Any value contained in variables or functions is called an expression. Something does not necessarily need to be evaluated in all expressions.

There are certain names, or identifiers that are reserved to the Python language. These keywords cannot be used as regular names. Since Python is a case sensitive language, these names (or identifiers) must be spelled correctly if they are to be used for other purposes. One can find out these keywords by using the **kwlist** function in **keyword** module as shown in the following code:

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
>>>
```

Example 1.3 Check whether any of the following names is a keyword or not. "Howareyou", "and"

Solution. By using **iskeyword** function from keyword **module**, we can check the first name "Howareyou" as following.

```
>>> import keyword
>>> keyword.iskeyword("Howareyou")
False
>>>
```

Note that the result is False, meaning that the entered name is not a keyword. Similarly, the word "and" can be checked as follows:

```
>>> from keyword import iskeyword as isk
>>> isk("and")
True
>>>
```

◀

In the previous program, the structure of the code should be kept as it is. The only term that can be altered is the "isk" term. Notice that the result is turned out to be "True", meaning that the word that we just checked is a keyword.
In some programming languages such as C, C++, or Java, the types of the variables should be defined. In Python, this is not the case. Variables can be immediately used as they are assigned corresponding values. This property often makes the code shorter compared to other languages.

There are certain rules that need to be considered when giving names to variables or identifiers.

The name should start with either a letter or an underscore ("_" character). The remainder of the name may be composed of letters, numbers, or underscores.

Example 1.4 Check whether it is permissible to use the following names as a variable name. "Hola", "15Good", "Alex+Huns", "Nice_Job", "WhatisWrong?", "Che_ese2017"

Solution. It can be checked whether a name is valid to be a variable or identifier name or not by using the **isidentifier()** function as follows:

```
>>> 'Hola'.isidentifier()
True
>>> "15Good".isidentifier()
```

```
False
>>> "Alex+Hunt".isidentifier()
False
>>> "Nice_Job".isidentifier()
True
>>> "WhatisWrong?".isidentifier()
False
>>> "Che_ese2017".isidentifier()
True
>>>
```

◀

As it can be seen above, the names that turn out **True** are permissible to be a variable name. We can place the words between quotation marks, or apostrophes. Both are acceptable in Python.

If we consider the structure of 'Hola'.isidentifier(), we see that there is a string, then a dot, and finally a function. We may think of dot "." as an operator. What it performs is that, it applies the function following the dot to the string preceding it. This structure will be frequently used throughout the book.

Example 1.5 In a given equation, $P * V = n * R * T$, some of the variables are as follows: $P = 10$, $n = 2$, $R = 7$, and $T = 4$. Write a code that calculates the unknown parameter V.

Solution. First, we may assign the given values to the variables. Then, the unknown V can be determined from $V = (n * R * T)/P$ as shown in the following program:

```
>>> P, n, R, T = 10, 2, 7, 4
>>> V = (n * R * T ) / P
>>> V
5.6
>>>
```

◀

Note that, the variables can be defined in one row by separating them with a comma to make the code more compact, as shown in the program above. The known variables can also be as-signed in a separate row for each one, as well. But, that makes the code longer. After calculating V, if you type V and hit enter, you can see the value of the variable V.

1.5 Overview of Core Data Types

There are different types of data in Python. These data types will be briefly introduced in this section in order to make the reader familiar with them. In Chapter 5, we will elaborate on the examination of these data types. There are six standard data types in Python. These include Numbers, Strings, Booleans, Lists, Tuples and Dictionaries.

1.5.1 Numbers

There are two types of numbers; namely *integers* and *floats*.

```
>>> x = 4
>>> type(x)
<class 'int'>
>>> y = 3.5
>>> type(y)
<class 'float'>
>>>
```

In order to find out the data type of a number, we use the **type()** function. As seen above, 4 is assigned to the variable x, and 3.5 is assigned to y. Python automatically concludes that x is an integer, and y is a float number. Once you check the classes of these numbers, you can see that they are marked as 'int' which stands for integer, and 'float' obviously meaning a float number.

1.5.2 Strings

A string is a series of characters saved to the memory in the form of text. In python, one can assign strings to variables by using quotation marks, or apostrophes, or by putting

the text inside the **str()** function as shown in the following.

```
>>> Str1 = "This is a string"
>>> Str2 = 'This is also a string'
>>> Str3 = str("2017 is another string")
>>> type(Str1)
<class 'str'>
>>> type(Str2)
<class 'str'>
>>> type(Str3)
<class 'str'>
>>>
```

The number of characters from which a data is composed of may be checked by using the **len()** function.

```
>>> len(Str1)
16
>>> len(Str3)
22
>>>
```

As can be seen above, Str1 possesses 16 characters. We can get inside of these strings by using square brackets.

```
>>> Str1[2]
'i'
>>> Str1[0:5]
'This '
>>> Str1[6:]
's a string'
>>> Str1[-5:]
'tring'
>>>
```

Indexing starts with zero (0) in Python. Therefore, Str1[2] means the 3^{rd} element of the string Str1. Similarly, Str1[0:5] yields all elements of Str1 from the 1^{st} element to the 5^{th}; Str1[6:] yields all elements of Str1 from the 7^{th} to the last element. We can reach the elements in the reverse order, as well. For example, Str1[-5:] yields all elements of Str1 from the last 5^{th} element up to the last element.

Strings may be concatenated by adding them up as follows:

```
>>> Str1 = "Hello"
>>> Str2 = "World"
>>> Str1+Str2
'HelloWorld'
>>>
```

As it is seen, by using the plus (+) sign, strings are concatenated without any space between them.

The function **print()** is another important built-in function to print data on the screen.

```
>>> print(Str1)
This is a string
>>>
```

By using the print() function, we can also concatenate strings as shown in the following.

```
>>> Str1="Alexander"
>>> Str2="Friend"
>>> print(Str2,Str1)
Friend Alexander
>>>
```

As shown above, by using the **print()** function, a space is inserted between the strings.

Example 1.6 Three strings are given as St1 = "My Big Brother", St2 = "Sports Club", St3 = "Barcelona Nice". By using these three strings given, obtain the new string "Big Club Barcelona".

Solution. We may write the following piece of code at prompt to get the required string.

```
>>> St1 = "My Big Brother"
>>> St2 = "Sports Club"
>>> St3 = "Barcelona Nice"
>>> Get_Str = str(St1[3:6]+" "+St2[7:]+" "+St3[0:9])
>>> print(Get_Str)
Big Club Barcelona
>>>
```

◀

1.5.3 Booleans

Boolean values are used to represent truth values as True, or False. Function **bool()** can be used to return a Boolean value.

```
>>> bool(3<5)
True
>>> bool(10>20)
False
>>>
```

We may want return Boolean values while checking a string for different purposes, as well.

```
>>> St1 = "This is NICE"
>>> St2 = "2017"
>>> St3 = "WOW"
>>> St4 = "Year 12345"
>>> St1.isalnum() #check if all characters of St1 are numbers
False
>>> St2.isdigit() # check if St2 has all digits
True
>>> St3.isupper() #check if characters of St3 are all upper case
True
```

```
>>> St4.endswith('5') # check if St4 ends with string 5
True
>>>
```

As it is seen above, the character "#" is used to leave a comment in Python. Whatever you write after the "#" symbol is ignored.

1.5.4 Lists

Lists are very important data types in Python. We can create arrays or matrices using lists. These topics will be covered in Chapter 5.

Lists are enclosed with square brackets [ ].

```
>>> MyList=[1,2,3,'Banana','Orange']
>>> type(MyList)
<class 'list'>
>>> MyList=([1,2,3,'Banana','Orange'])
>>> type(MyList)
<class 'list'>
>>>
```

As we can see above, we can use square brackets inside parentheses or square brackets alone to define a list.

To get data from a list, indexing may be used.

```
>>> MyList[2]
3
>>> MyList[3][1:5]
'anan'
>>>
```

We can delete any of the items of the list by using **del** function. We can delete the 4^{th} element of MyList as follows.

```
>>> del MyList[3]
>>> MyList
```

```
[1, 2, 3, 'Orange']
>>>
```

After the **del** command,we need to put a space, and then the item we want to remove from the list.

There are some useful functions available to work with the lists.

Table 1.1: Some of the available functions used with the lists

Function	**Explanation**
ListName.append(x)	adds the item x to the end of the list called ListName
ListName.count(x)	returns how many times the item x appears from the list called ListName
ListName.index(x)	returns the index of the first appeared item x from the list called ListName
ListName.insert(i, x)	inserts the item x after the ith position of the list called ListName
ListName.remove(x)	removes the first item x from the list called ListName
ListName.reserve(x)	reverses the elements of the list called ListName
ListName.sort()	sorts the orderable (only numbers or strings) items of the list called ListName

Example 1.7 Consider the list given as List1 = [2, 5, 'Onions', 'Truck', 50, 5, -3].
a) Remove the item named 'Truck' from the list.
b) Insert number 1234 after the 2^{nd} item in the list.
c) Put the final list in the reversed order.

Solution. a) We can use **remove()** function to remove the first appeared specific value as follows:

```
>>> List1 = [2, 5, 'Onions', 'Truck', 50, 5, -3]
>>> List1.remove('Truck')
>>> print(List1)
[2, 5, 'Onions', 50, 5, -3]
>>>
```

b) We can use the **insert()** function to insert the number 1234 after the 2^{nd} item as follows.

```
>>> List1.insert(2, 1234)
>>> print(List1)
[2, 5, 1234, 'Onions', 50, 5, -3]
>>>
```

c) Finally, we can use the **reverse()** function to put the items in reversed order as follows:

```
>>> List1.reverse()
>>> print(List1)
[-3, 5, 50, 'Onions', 1234, 5, 2]
>>>
```

◀

For sorting the items, **sort()** function can be used. But in using this function, the items should be all numbers, may be composed of mixed integers and float numbers, or strings.

1.5.5 Tuples

Tuples are yet another sequence of data types in Python. They are enclosed with parentheses () instead of square brackets. Tuples have structures. Their items cannot be changed once they are created.

```
>>> Tup = (1, 2, 5, 'Of', 'FC')
>>> type(Tup)
<class 'tuple'>
>>> len(Tup)
```

```
5
>>> Tup[3]
'Of'
>>>
```

1.5.6 Dictionaries

Dictionaries are another data type of pairs that holds keys corresponding to certain values. They are defined with curly brackets. Dictionaries appear like lists. However, lists are ordered sets of objects while dictionaries are unordered. Another important difference is that in dictionaries, items are reached by using their keys, not via their positions.

```
>>> Diction= {"Cars":"Cool","Doctor":"Antonio","City":"Dallas"}
>>> type(Diction)
<class 'dict'>
>>> Diction['Doctor']
'Antonio'
>>> Diction["City"] = "Austin"
>>> Diction["City"]
'Austin'
>>>
```

As can be above, each key corresponds to a value. Values can be changed through their keys.

Table 1.2: Some available functions used with the dictionaries

Function	**Explanation**
DictName.clear()	removes all the elements of dictionary named DictName
dict.fromkeys(seq,val)	creates a new dictionary with keys from seq and values from val
DictName.get(x)	finds the corresponding value of key x from dictionary named DictName
DictName.items()	returns a list of (key, value) pairs from dictionary named DictName
DictName.update(dictB)	adds key-value pairs from dictionary DictB to dictionary named DictName

Example 1.8 Consider the dictionary given as Dict1 = "Math":"Calculus", "Biology":"Cell", "Year":2017 . Write a code to update this dictionary with "Age":21

Solution. We can update the dictionary by using **update()** function as shown in the following.

```
>>> Dict1 = {"Math":"Calculus", "Biology":"Cell", "Year":2017}
>>> DictB = {"Age":21}
>>> Dict1.update(DictB)
>>> Dict1
{'Biology': 'Cell', 'Math': 'Calculus', 'Age': 21, 'Year': 2017}
>>>
```

◀

As it is seen above, after updating the dictionary, the items do not appear in order.

Example 1.9 A sequence is given as Seq1 = ('first_name', 'last_name', 'Number'). Write a code to create a dictionary by using Seq1 as keys where 50 is the assigned value to all keys in the dictionary.

Solution. We can create the dictionary as follows.

```
>>> Seq1 = ('first_name', 'last_name', 'Number')
>>> NewDict = dict.fromkeys(Seq1,50)
>>> NewDict
{'last_name': 50, 'Number': 50, 'first_name': 50}
>>>
```

◀

1.6 Operators

Operators are the symbols or expressions that makes Python interpreter to infer results. There are arithmetic operators, relational or comparative operators, logical operators, assignment operators, bitwise operators and some other special operators used in Python.

1.6.1 Arithmetic Operators

Arithmetic operators are used to carry on mathematical calculations.

Table 1.3: Arithmetic Operators used in Python

Operator	**Description**	**Example**
+	Addition	6 + 3 = 9
-	Subtraction	6 - 3 = 3
*	Multiplication	6 * 3 = 18
/	Division	6 / 3 = 2
%	Modulo	6 % 3 = 0
**	Exponent	6 ** 3 = 216
//	Floor Division	6 // 3 = 2

In the table above, the Floor Division operator is a rounding operator. What it does is, after dividing, it rounds the result to the closest small integer. As an example, if you type ⋙ 24//5 and hit enter at the prompt, the result will be 4. When 24 is divided by 5, the result is 4.8; and if 4.8 is rounded to the closest small integer value, it equals 4.

1.6.2 Relational Operators

Relational operators are called comparative operators as well.2 operands are compared with the symbols given in the following table.

Table 1.4: Relational Operators used in Python

Operator	**Description**	**Example**
>	Greater than	6 >5
>=	Greater than or equal to	5 >= 5
<	Less that	2 <3
<=	Less than or equal to	3 <=5
==	Equal to	4 ==4
!=	Not equal to	1 !=5

If the case is correct, then True is returned from this comparison.

1.6.3 Logical Operators

There are three logical operators being used in Python.

Table 1.5: Logical Operators used in Python

Operator	Description	Example
and	True if both operands are true	A and B
or	True if one operand is true	A or B
not	True if operand is false	not A

1.6.4 Bitwise Operators

Bitwise operators result differently comparing to arithmetic operators. The operands behave like binary digits.

Table 1.6: Bitwise Operators used in Python

Operator	Description	Example
&	AND	12 & 7 =4
|	OR	12 |7 = 15
$\sim$	NOT	$\sim$ 12 = -13
$\wedge$	XOR	12$\wedge$7 =11
$\ll$	Right Shift	12 $\ll$ 7 = 1536
$\gg$	Left Shift	12 $\gg$ 7 = 0

In the figure above, 12 is equal to '1100' and 7 is equal to '0111' in binary format. If you apply & (and) operator to 1100 and 0111, the result is 0100 which is equal to 4 in decimal format. All other examples are calculated in similar ways.

1.6.5 Assignment Operators

Assignment operators are used to assign values to the variables.

Table 1.7: Assignment Operators used in Python

Operator	**Sample Usage**	**Equivalent Usage**
=	x = 9	x = 9
+=	x + = 9	x = x + 9
-=	x - = 9	x = x - 9
*=	x * = 9	x = x * 9
/=	x / = 9	x = x / 9
%=	x % = 9	x = x % 9
//=	x // = 9	x = x // 9
**=	x ** = 9	x = x ** 9
&=	x & = 9	x = x & 9
|=	x |= 9	x = x |9
∧ =	x ∧ = 9	x = x ∧ 9
≪ =	x ≪ =	x = x ≪ 9
≫ =	x ≫ = 9	x = x ≫ 9

1.6.6 Special Operators

There are some special operators being used in Python. These operators can be grouped as identity operators and membership operators. Identity operators are used to check whether two identity values are the same or not by using “is” and “is not”. On the other hand, membership operators are used to find out whether or not a variable exists in a list, or in a group of items by using “in” and “not in”.

Table 1.8: Special Operators used in Python

Operator	Sample Usage	Equivalent Usage
is	True if the values are same	x is y
is not	True if the values are not same	x is not y
in	True if value is in the list	2 in List
not in	True if value is not in the list	2 not in List

Example 1.10 Consider the variables given as Str1 = "Hello World", Num1 = 5, and Num2 = 5.
a) Write a code to check if "Hello" exists in the string Str1 by using special operator "in".
b) Write a code to check if Num1 and Num2 are equal by using special operator "is".

Solution. a) The following code may be entered at prompt to accomplish the given task.

```
>>> Str1 = "Hello World"
>>> print('Hello' in Str1)
True
>>>
```

b) The following code may be entered at prompt to accomplish the task.

```
>>> Num1 = 5
>>> Num2 = 5
>>> print(Num1 is Num2)
True
>>>
```

◀

1.7 Built-in Functions

Similar to other programming languages, there are ready to use functions called built-in functions in Python, too. These functions make the programmer's job much easier.

There are also functions defined by the user called user-defined functions. Construction of user-defined functions will be presented in Chapter 6.

Table 1.9: Built-in functions available in Python

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

Any of these functions can be examined by using the help function at prompt. As an example, if you wonder what **bin()** function does, you can type ⋙ help(bin) and hit enter at prompt to see the following information.

```
>>> help(bin)
Help on built-in function bin in module builtins:
```

```
bin(number, /)
    Return the binary representation of an integer.

    >>> bin(2796202)
    '0b1010101010101010101010'

>>>
```

1.8 Formatting Output

We can set the format of the output while printing data on the screen. By using the "modulo" operator, an output can be shaped in the following manner.

```
>>> import math
>>> print("Pi is %10.4f" % math.pi)
Pi is     3.1416
>>> print("Pi is %2.7f" % math.pi)
Pi is 3.1415927
>>>
```

Above, the numbers after the modulo symbol set the spaces coming before the dot and digits after the dot as shown in π number above.

By using **format()** function, the same outputs can be printed on the screen in the following way.

```
>>> print("Pi is {:10.4f}".format(math.pi))
Pi is     3.1416
>>> print("Pi is {:2.7f}".format(math.pi))
Pi is 3.1415927
>>>
```

In the program above, curly brackets tell the Python interpreter that a data will be printed out in that position. We can use the **format()** function to print multiple data on the screen as follows, too.

```
>>> St1="Hello my friend"
>>> "{:s} : {:2.5f}".format(St1, math.pi)
'Hello my friend : 3.14159'
>>>
```

Table 1.10: Conversions used with modulo operator

Used Letter	**Meaning**
d	Signed integer in decimal format
i	Signed integer in decimal format
e	Floating point in exponential format
E	Signed integer in decimal format
f	Floating point in decimal format
F	Floating point in decimal format
c	Single character
s	String

1.9 PROBLEMS

1.1 Write a code to calculate the value of $y = sin(\frac{\pi}{3}) + \sqrt{5} + e^3$.
1.2 What are the differences between the functions *iskeyword()* and *isidentifier()*?
1.3 Which of the following or followings can be a variable name?
a) My_journey
b) What_wrong?_John
c) The yearis2018
d) 2000Yes
1.4 Given the formula F = m * a, write a script to calculate the value of a when m = 5 and F = 24.
1.5 Two strings are given as St1 = "Paris is in Europe" and S2 = "This Nice Continent". Write a script to obtain "Europe Continent is Nice" by using S1 and S2.
1.6 If "Logical = bool(5¡4) or bool(10¿12)" is typed at prompt, what value will be assigned to the variable "Logical"?
1.7 A list given as List1 = [17, 61, 'Table']. Write a script that adds the number "1900" as a third item to List1.
1.8 A dictionary is given as Dict = "Name":"Alex", "Lesson":"PE", "Age":40. Write a script that finds the corresponding value of the key "Age".
1.9 If you type "(2**5)//6" at the prompt and hit the enter, what will the result be?
1.10 If you type "10%5" at the prompt and hit the enter, what will the result be?
1.11 After typing ⋙import math, and ⋙":0.5f : :0.10f".format (math.pi, math.pi) at the prompt, what will the output be?

Chapter 2

FOUNDATIONS OF PROGRAMMING

In this chapter, we will present the basics of programming with Python.

The topics such as algorithms, flowcharts, pseudocodes, scripts and Py files will be introduced and handled.

2.1 Introduction

A computer program is basically a set of instructions that tells computer what to do. In Python, for larger programs or codes, editor is preferred rather than typing everything at the prompt. The information we have learned so far was easy to execute and accomplish. Solving problems by just using the prompt, was enough for the examples we have done up to this point. We could use Python's editor to solve the same questions as well. However, for larger and more complex problems, it is recommended to keep your codes together organized in an editor file. Therefore, we will attempt to use the editor to type our codes for the rest of the problems throughout this book, as much as possible. As mentioned in the previous chapter, there are other Python editors that assist you in writing your codes, as well.

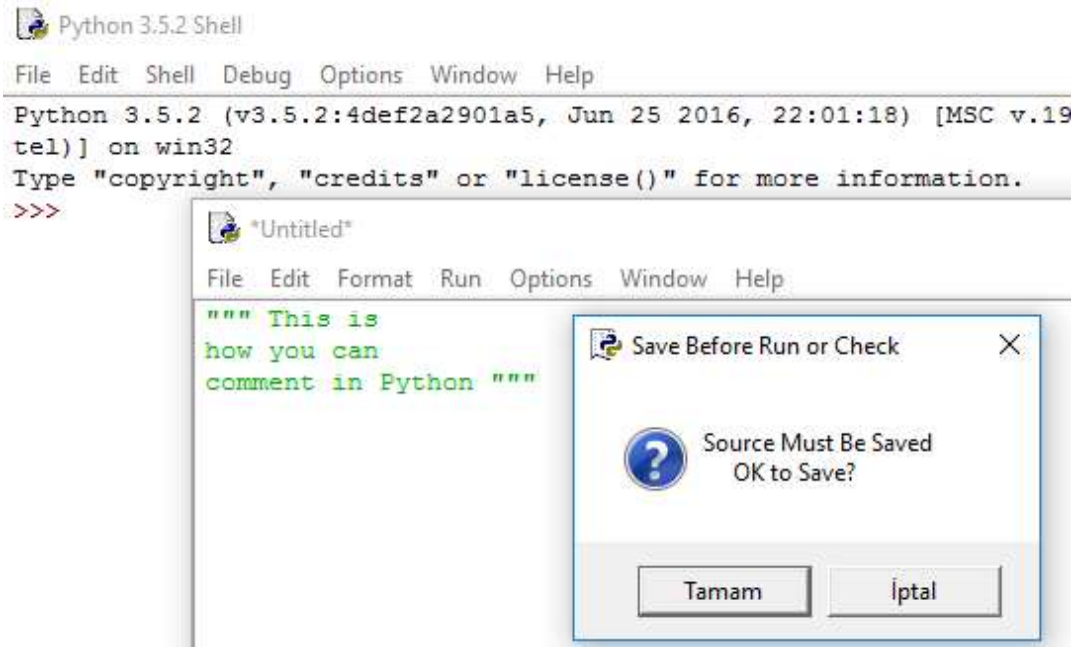

Figure 2.1: An outlook of a Warning pop-up window

If you type your codes in the editor and try to run it before saving, Python gives a warning to save it first. After saving your code, you can hit F5 over the editor, or follow the path Run → Run Modulo from the editor's menu.

It is not difficult to write complex, unorganized codes. After writing a program, if you need to look back to your code for some reason such as upgrading or rearranging the code, it might be difficult to understand what your code does in each line. Therefore, one of the primary concerns while creating a code is making them clear and easy to understand. To accomplish this, you can put some comments in the code as reminders after # symbol in a line. Comments are informative expressions to make the codes easy to understand. Python ignores whatever is written after the # symbol in a line. Putting a comment block is also possible by putting three quotation marks """ to start a comment. It is necessary to put another set of three quotation marks to terminate the comment block as shown in the editor in Figure 2.1.

Another important factor for writing better codes is to use meaningful names for variables, or structures. That helps us remember our scripts easily for later times.

There exists other useful information related to writing better codes which is out of

the scope of this book. For a basic understanding of the subject, we will keep it simple here.

2.2 Algorithms

Algorithm is a computational procedure which shows all the steps involved in solving a problem. Generally, an input (or a set of inputs) is taken and an output (or some outputs) is (are) produced. To solve a problem, the problem is separated into parts to examine and accomplish the pieces separately. In this way, the problem can be analyzed in a detailed manner. This method is often called a **top-down** approach. Sometimes, pieces are combined to get information about the entire system which yields more complex systems. We can build the system by analyzing and combining the pieces that we have gathered information about. This method is called **bottom-up** processing.

In either way, the purpose of the algorithms is to define a solution to a problem in some ways such as writing the solution with simple words, or showing the solution by using flowcharts.

2.3 Flowcharts and Pseudocodes

Both Pseudocodes and flowcharts are used to construct an algorithm of a computer program. While Pseudocodes are generally composed of words, flowcharts are represented by some simple shapes.

Pseudocodes are informal ways of describing the steps of the program to be written. It helps the programmers to plan and develop the ultimate algorithm for the computer program.

Flowchart is a graphical representation of the steps towards writing the program. Shapes along with diagrams are being used to explain the algorithm of the computer program.

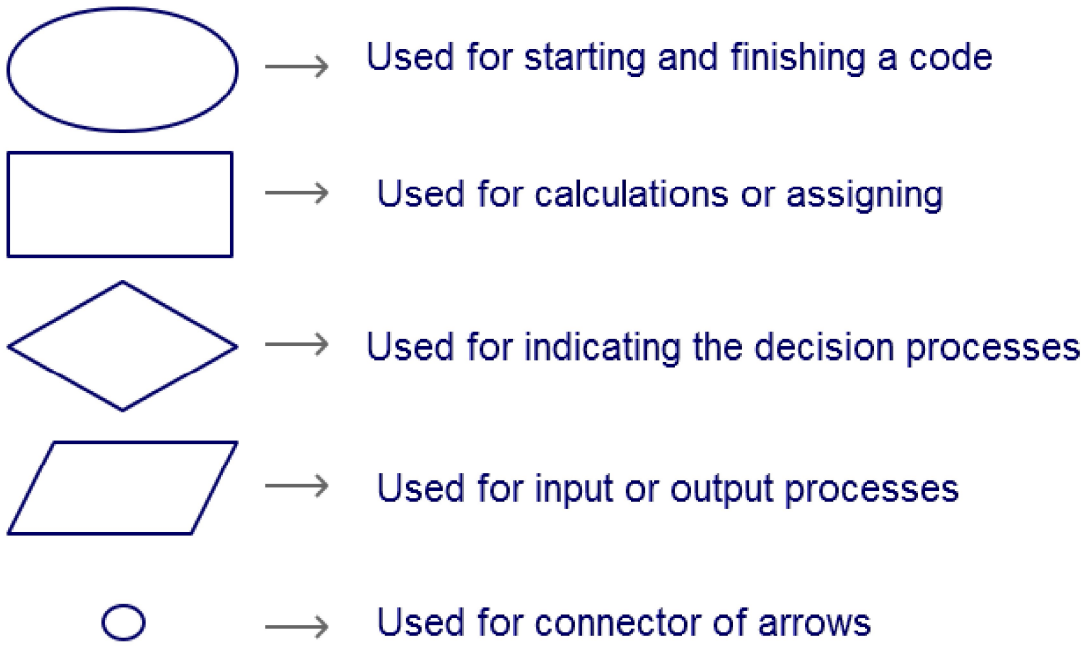

Figure 2.2: Commonly used flowchart symbols

Example 2.1 Construct a pseudocode and flowchart for an algorithm that calculates the area of a square. The length of one side of the square should be externally entered by the user.

Solution. For the pseudocode, we can write the steps of the algorithm as below.

1. Enter the side of the square, called B

2. Calculate the formula of the area of a square $Area = B^2$

3. Display the result, we got from the calculation, $Area$

Per the flowchart, we can represent the necessary steps as follows:

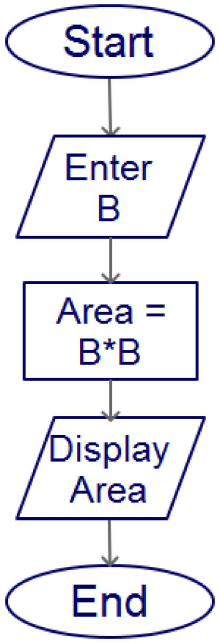

Figure 2.3: Flowchart of Example 2.1

◀

2.4 Scripts and Py files

After an algorithm is determined to solve the problem, the code must be written to accomplish the given task. These codes can be written in Python editor and saved as a *.py file where * is substituted by the name you give your program. Thus, the codes written in Python are called Py files and these codes are called scripts or programs.

Example 2.2 Write the code of Example 2.1 to calculate the area of a square.

Solution. In this solution, we will use an important function, called **input()**. A data is passed to the computer by using input command. The default data type of input() function is the string. Therefore, if an integer or a float type is expected, then the data should be converted to int or float.

```
1 #Example 2.2
2 #This example calculates the area of a square
```

```
3 SideB = int(input("Please enter a side of the square :"))
4 Area = (SideB)**2
5 print("The Area of the square is :",Area)
```

Listing 2.1: Example2p2.py

Once you run the code, the following output will be obtained:

```
Please enter a side of the square :12
The Area of the square is : 144
```

◀

Above, the computer expects you to enter a number. Once the number is entered, it is convert-ed to an integer format and assigned to variable named *SideB*. Then the square of *SideB* is calculated and assigned to the variable named *Area*. Finally, the words between quotation marks are printed on first, and next to it, the value of Area is printed on the screen.

Example 2.3 Draw a flow chart of an algorithm and write the code which calculates the area and perimeter of a circle where the radius is entered by user.

Solution. The flowchart of the algorithm can be drawn as below.

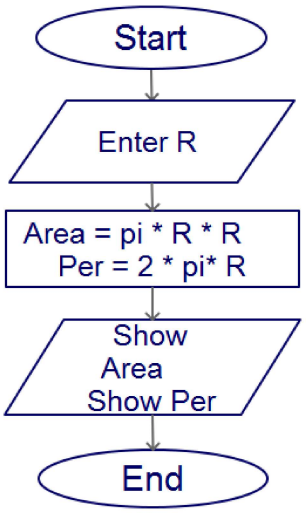

Figure 2.4: Flowchart of Example 2.3

The code can be written as follows:

```
#Example 2.3
#This example calculates the area and perimeter of a circle
import math
Ra = float(input("Please enter the Radius of the circle :\n"))
Area = math.pi * Ra**2
Per = 2*math.pi*Ra
print("The Area of the circle is :",Area)
print("The Perimeter of the circle is :",Per)
```

Listing 2.2: Example2p3.py

Once you run the code, the following output will be obtained:

```
Please enter the Radius of the circle :
6.5
The Area of the circle is : 132.73228961416876
The Perimeter of the circle is : 40.840704496667314
```

◀

As it is seen from the code, we imported math module in order to use the value of π. There is "\n" is used in 4^{th} line. This is one of the escape sequences used in Python. What it does is, after printing whatever comes before it, and then the cursor goes to the next line to continue from there.
That is the reason that, number 65 is entered in the next line at the output.

Table 2.1: Some of the used Escape Sequences in Python

Escape Sequence	**Meaning**
\‘	Single quote
\‘‘	Double quote
\\	Backslash
\n	New line
\t	Tab
\f	ASCII Form feed

Example 2.4 Write a flowchart of an algorithm that compares 20 with a number entered by the user. If the number that is entered is greater than 20, then the program must show “Number is greater than 20”, otherwise, it should print “The entered number is less than or equal to 20” (Note: We will present how to write codes using if, if-else structures in the next chapter).

Solution. The flowchart of the algorithm can be drawn as below.

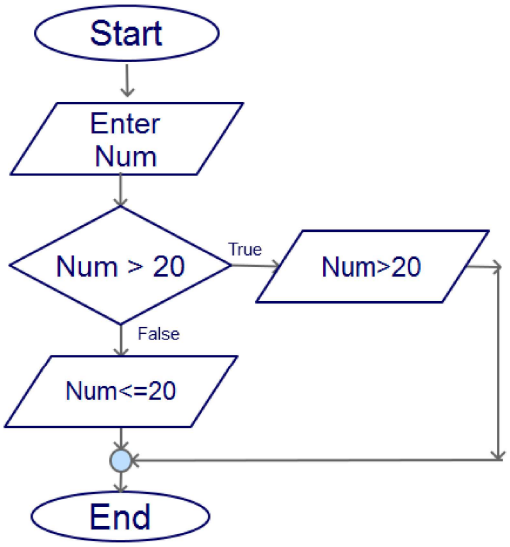

Figure 2.5: Flowchart of Example 2.4

◀

2.5 PROBLEMS

2.1 Write the code and construct the flowchart of a program that takes 2 inputs as the shorter and longer sides of a rectangle, and print the area of the rectangle on the screen.
2.2 Create the flowchart of an algorithm that, for an entered number greater than 18, it prints "Yes you can take a driver's license", and otherwise "No driver license".
2.3 Think of an air pumping device that inflates tires. When the barometer of the pump reads 32, the device stops working. If it if less than 32, it keeps working and pumping air into the tire. Construct a flowchart for the algorithm of this device's operation.
2.4 Express the output of the following flowchart. What is the value printed on the screen?

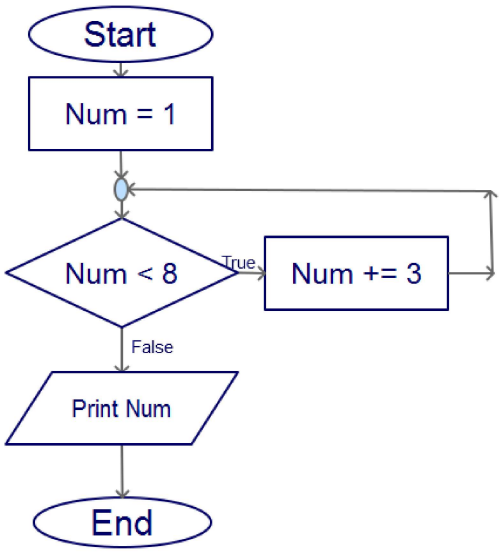

Figure 2.6: Flowchart of chapter problem 2.4

2.5 Express the output of the following flowchart. What is the value printed on the screen?

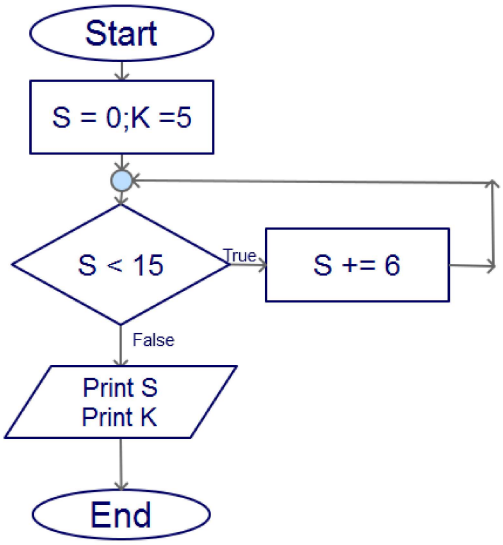

Figure 2.7: Flowchart of chapter problem 2.5

Chapter 3

LOGICAL FUNCTIONS AND SELECTION STRUCTURES

In this chapter we will present how to use **if**, **elif**, and **else** statements along with handling some error scenarios by using **try-except-finally** statements. In some cases, we need to select an option among a set of possible candidates. This selection may be made by using if-elif-else statements in Python.

Before continuing, we need to understand a rule related to indention.

Unlike some of other programming languages, in Python, when some keywords are used such as if, for, while, etc. neither the body part of these keywords are closed with the word "end" nor curly brackets are used to single out these blocks. Python interpreter needs to understand how to group these blocks, where they start and where they end. Therefore, when such statements are used including the "if" statement, indentation needs to be applied. Default indention amount is four spaces. Within our codes, we will use 4 spaces for all flow control blocks, too.

3.1 Single if-else Statement

If statements are always followed by a logical expression. However, in **else** statements, it is optional depending on the problem.

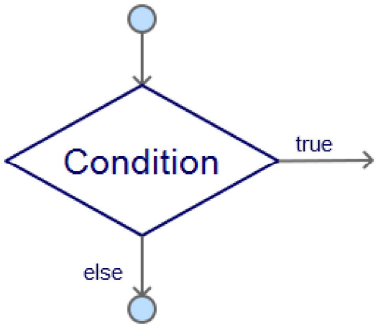

Figure 3.1: Flowchart of a single if-else statement

The usage of an if-else structure is given below.

```
if (Condition) :
        statement
else:
        statement
```

Example 3.1 Write a program which picks a pair of dice. If the sum of the rolled numbers are 10, then the program should print " Yes, you WON" on the screen. For all other cases, the computer should print "Try again" on the screen.

Solution. We will use random module to write the code.

```
#Example 3.1
#This example finds the sum of a pair of rolled dice
import random
Num1 = random.randrange(1,7)
Num2 = random.randint(1,6)
Sum = Num1 + Num2
if Sum == 10:
  print("Yes you WON")
else:
  print("Try again")
```

Listing 3.1: Example3p1.py

Once you run the code, the following output will be obtained.

```
Yes you WON
```

◀

In the above program, the functions **randrange()** and **randint()** are used via the random module. The difference between these functions is that the randrange() function picks numbers up to 7, excluding 7, while randint() function picks numbers up to 6, including 6.

Another point that needs to be considered is that you should put a colon at the end of each line of the "if" and "else" statements.

3.2 If-elif-else Statements

In this type, more than 1 option is checked. The flowchart of one **elif** with **if-else** statement can be drawn as following.

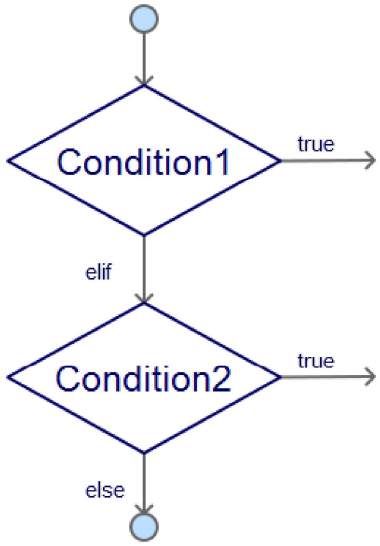

Figure 3.2: Flowchart of if-elif-else statement

As can be seen in the flowchart, if Condition1 is not satisfied, then Condition2 is checked. We may think of the statement elif as else if. If more than one elif statements

are used, then every other condition is checked one by one with the other given conditions in the order.

Example 3.2 Write a program that requests the age of the user. If the entered age is less than 6, the code should print "No school yet" on the screen. If the age is between 6 and 14, it should print "Middle school". If the age is between 15 and 18, it should print "High school", and finally, if the age is between 19 and 24, it should print "University time". Otherwise, the program should print "You are ready for life" on the screen.

Solution. The following code may be used to accomplish the task.

```
#Example 3.2
#This example use if-elif-else statements
Age = int(input("Please enter your age :"))
if Age < 6:
    print("No school yet")
elif Age < 15:
    print("Middle school")
elif Age < 19:
    print("High school")
elif Age < 25:
    print("University time")
else:
    print("You are ready for the life")
```

Listing 3.2: Example3p2.py

Once you run the code, the computer expects a number. The number that is entered will be converted to integer format and assigned to the variable "age". The variable "age" is compared to every condition. If one of the conditions is satisfied before the statement "else", then the body part of that condition is executed. After running the code, we obtain the following output.

```
Please enter your age :25
You are ready for the life
```

◀

3.3 Logical Operators with if-else Statements

We might encounter some problems that we need to use logical operators with if statements.

Example 3.3 Write a program that requests the name and password of the user. If the name is "Alex" and the password is "2017 is nice", then the computer prints "Access is Granted". Otherwise, the computer should print "You are not the ONE" on the screen.

Solution. The following code may be used to accomplish the given task.

```
#Example 3.3
#This example uses logical and
UserName = 'Alex'
UserPassword = "2017 is nice"
User  = input("Your User Name :\n")
Password = input("Enter Password :\n")

if((UserName == User) and (UserPassword == Password)):
    print("Access is Granted")
else:
    print("You are not the ONE")
```

Listing 3.3: Example3p3.py

The following output will be obtained as you run the code.

```
Your User Name :
Alex
Enter Password :
2017 is nice
Access is Granted
```

◀

3.4 try-except-finally Statements

Up to this point, we have used nice inputs to run our codes. As an example, while it is expected to enter an integer value by the user, a string may be entered, and in such

a case, an error may occur. Or some other types of error might occur due to some other reasons. To overcome such problems, **try-except** format is used. The main part that represents the aim is written after the "try" statement, and then the part that handles possible errors is written after the "except" statement. The word **finally** is used to execute the code written after that point for all possibilities.

Example 3.4 Write a program that requests the age of the user. Then, the age is printed on the screen. If an integer is not entered, then the code should handle the error with **except** statement.

Solution. The following code may be used to accomplish the given task.

```
#Example 3.4
#This example uses try-except
try:
    Age = int(input("Enter your Age :"))
    print("Your age is ", Age)

except ValueError:
    print("You should enter an integer")
```

Listing 3.4: Example3p4.py

In the code, a value error may occur for entering a string. And this situation can be handled by using "except ValueError" pair. Following is the output of the code above.

```
Enter your Age :Hello
You should enter an integer
```

◀

Sometimes, errors may occur while performing the calculations. Dividing a value by zero is such a kind.

Example 3.5 Write a program that requests your favorite number. The computer then prints the division of 100 by that favorite number. Use finally statement to print "Do not worry".

Solution. The following code may be used to accomplish the task.

```
#Example 3.5
#This example uses try-except-finally
try:
    FavNumber = int(input("Whats your favorite number: "))
    Divide = 100/ FavNumber
    print("100 / your number is : ", Divide)
except ZeroDivisionError:
    print("Do not pick ZERO")
finally:
        print("Do not worry")
```

Listing 3.5: Example3p5.py

The following output will result after running the code.

```
Whats your favorite number: 0
Do not pick ZERO
Do not worry
```

◀

Example 3.6 Write a program which calculates the roots of a quadratic equation of the form $ax^2 + bx + c = 0$. The coefficients a,b, and c should be entered by the user. If the roots are not real, then the code should give a value error.

Solution. The following code may be used to accomplish the task.

```
#Example 3.6
#This example find the roots of quadratic equation
import math
try:
    print("Please enter (a,b,c) the coefficients of quadratic equation")
    A = int(input("a :"))
    B = int(input("b :"))
    C = int(input("c :"))
    Delta = B**2-4*A*C
    x1 = (-B - math.sqrt(Delta))/(2*A)
    x2 = (-B + math.sqrt(Delta))/(2*A)
    print("The roots are {} and {}".format(x1,x2))
except ValueError:
```

```
14     print("A value error occured")
```

Listing 3.6: Example3p6.py

The following output will be obtained after running the code.

```
Please enter (a,b,c) the coefficients of quadratic equation
a :12
b :5
c :3
A value error occured
```

◀

3.5 PROBLEMS

3.1 Write a code that rolls a pair of dice. If one of the numbers is 6 and the total sum of these two numbers is 10, then the computer prints "You are lucky" on the screen.
3.2 Write a program that requests the first and the last names of the user. If the first name is "Alex", or the last name is "Oliver", then the computer should print "You are my friend" on the screen. For all other options, it should print "Keep trying" on the screen.
3.3 Write a program that requests to enter a float number. If the number that is entered by the user is a string, then the code should request to enter it again by using the **except** statement.
3.4 Write a program that requests the user to enter a number. Then the program should calculate the absolute value of this number. If the data entered by the user is not a number, then the code should print a warning on the screen by using the "except ValueError" statement.
3.5 Write a program that asks your age. If the age is less than 18, the computer should print " No driver license". If the age is between 18 and 60, the program should print "You can apply for a driver license". If the age entered by the user is greater than 60, then the computer should print "You have a lot of experience" on the screen.
3.6 Draw the flowchart for the previous scenario in Problem 3.5.