Study Unit 1   
Introduction to Python Programming

## Learning Outcomes

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

1. Differentiate the various aspects of Python programming
2. Employ logic control flows in Python programmes

## Overview

This study unit introduces the Python programming environment and the writing of Python programs with some foundation elements. We will also learn how to create different types of variables and how to assign values to them for further operations. Since input and output belong to the core of any computer program, we will learn how to create user input and construct formatted strings for printing as well. Also, we will cover the construction of Boolean expressions as conditional statements to control the behaviour of the program. Eventually, we will find out how to create finite loops to repeat routine instructions iteratively.

## Chapter 1 Python Programming Environment

Lesson Recording - Introduction to Python Programming

### 1.1 Installation of Python and Atom

Visit the URL <https://www.python.org/downloads/> and click on “Download Python [Version]”, where [Version] is the latest version number of Python (Version 3.9.0 is the latest version when this Study Guide was being developed).

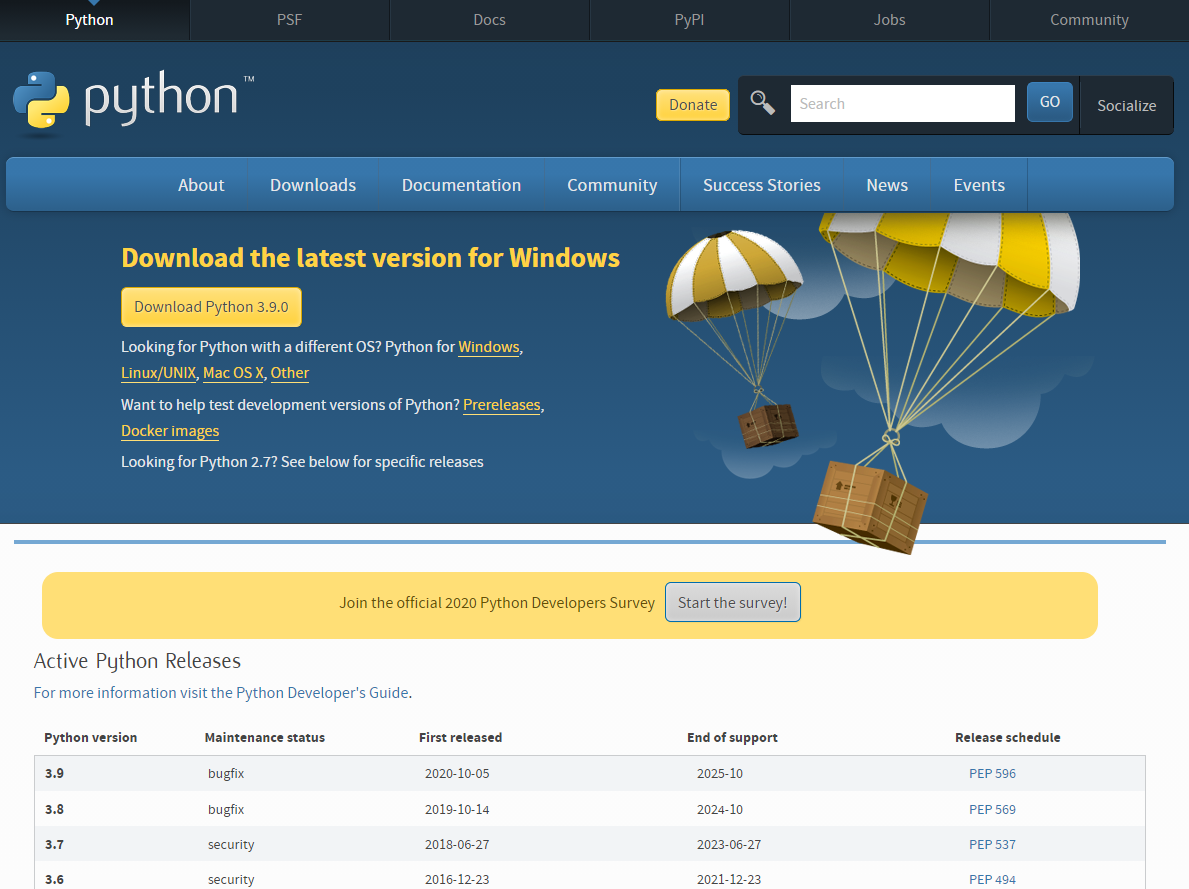


Figure 1.1 Python Website: Download the Latest Version

In this Study Guide, we will standardise the operating system to Windows 10. Users of Linux/UNIX, Mac OS X or other operating systems can find equivalent applications to execute the same steps. After downloading the installer and double-clicking on it, the following window will then appear:

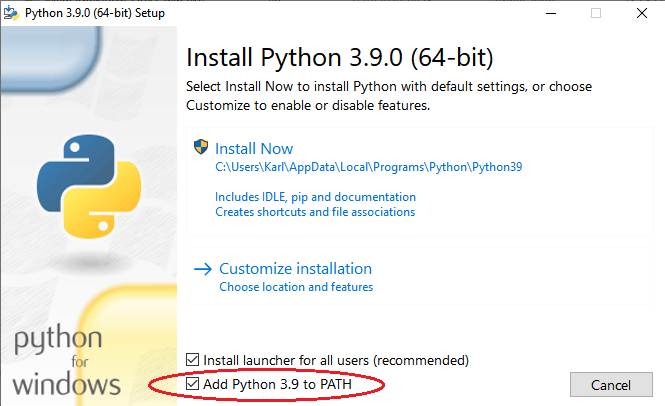


Figure 1.2 Python Installation on Windows: Check the Box to Add Python to PATH

Note that when installing Python on Windows, ensure that the box “Add Python [Version] to PATH” as in Figure 1.2 is checked.

Furthermore, we will need Atom as our editor for composing Python scripts. Though we can also execute Python codes without writing them in an editor, it is much more convenient to do so. We can download Atom text editor at <https://atom.io>:



Figure 1.3 Download Website of Atom for Windows

We can select the operating system that we prefer for the Atom installer. Press the “Download” button and install Atom by executing the installer after the download has completed.

**Read**

Read the following two sections of the textbook on installing Python 3 on mac OS or Windows:

Exercise 0. The Setup (Windows)

Exercise 0. The Setup (maxOS)

### Writing and Executing Python Programs

After the installations, we can start writing our Python program. One simple way is to write and run in Python directly, which we can find in the start menu:

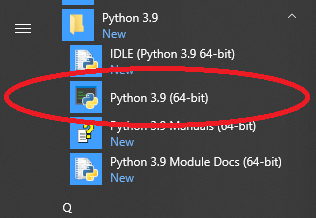


Figure 1.4 Finding Python in the Start Menu

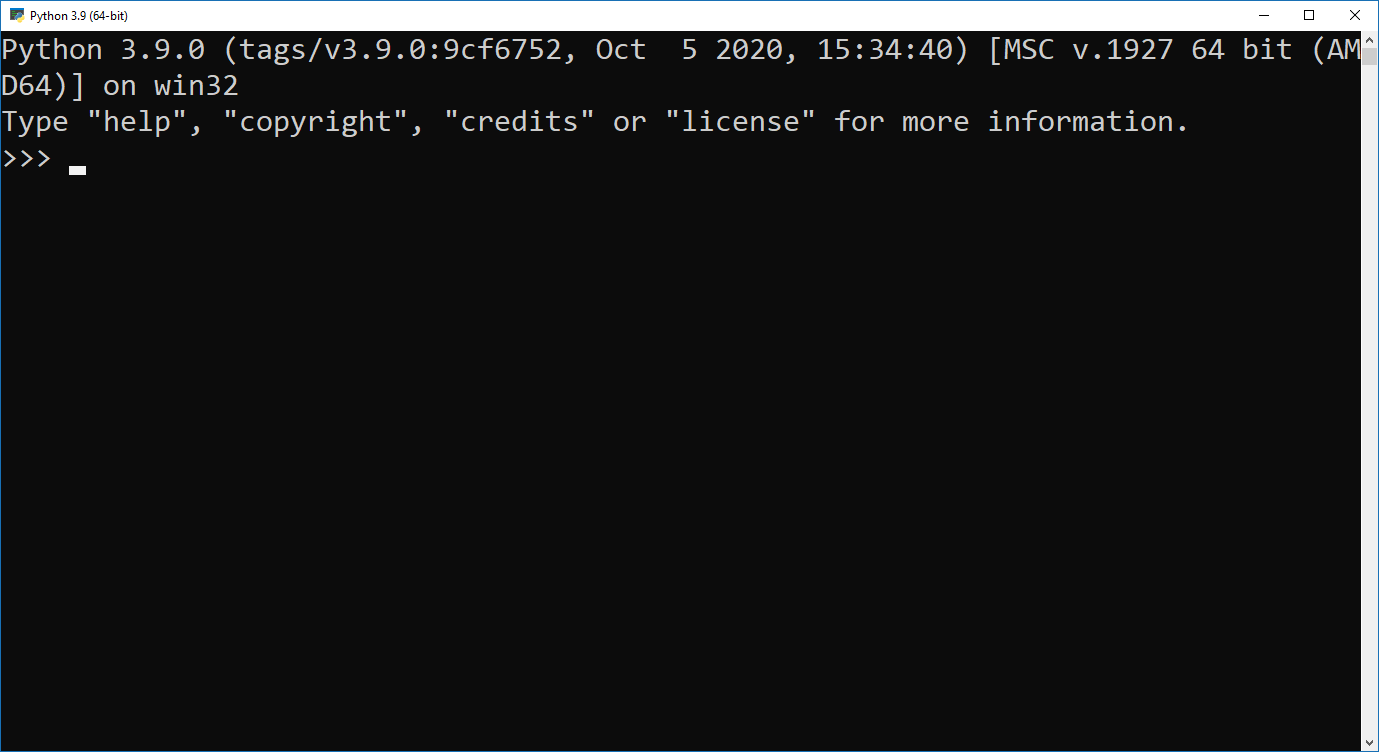


Figure 1.5 Python Interpreter

Once we see the “>>>” prompt, we can type in our Python command and let the Python interpreter execute it by pressing ENTER once the syntax is completed.

Another way to run Python is to call it from a terminal app. For this, we will need Windows PowerShell or Command Prompt to open it. Type “PowerShell” or “Command Prompt” in the “Search Windows” box on the task bar:

|  |  |
| --- | --- |
|  |  |

Figure 1.6 Searching for Windows PowerShell and Command Prompt

For simplicity, we will use Windows PowerShell in the following. Type “python” in PowerShell and then press ENTER:

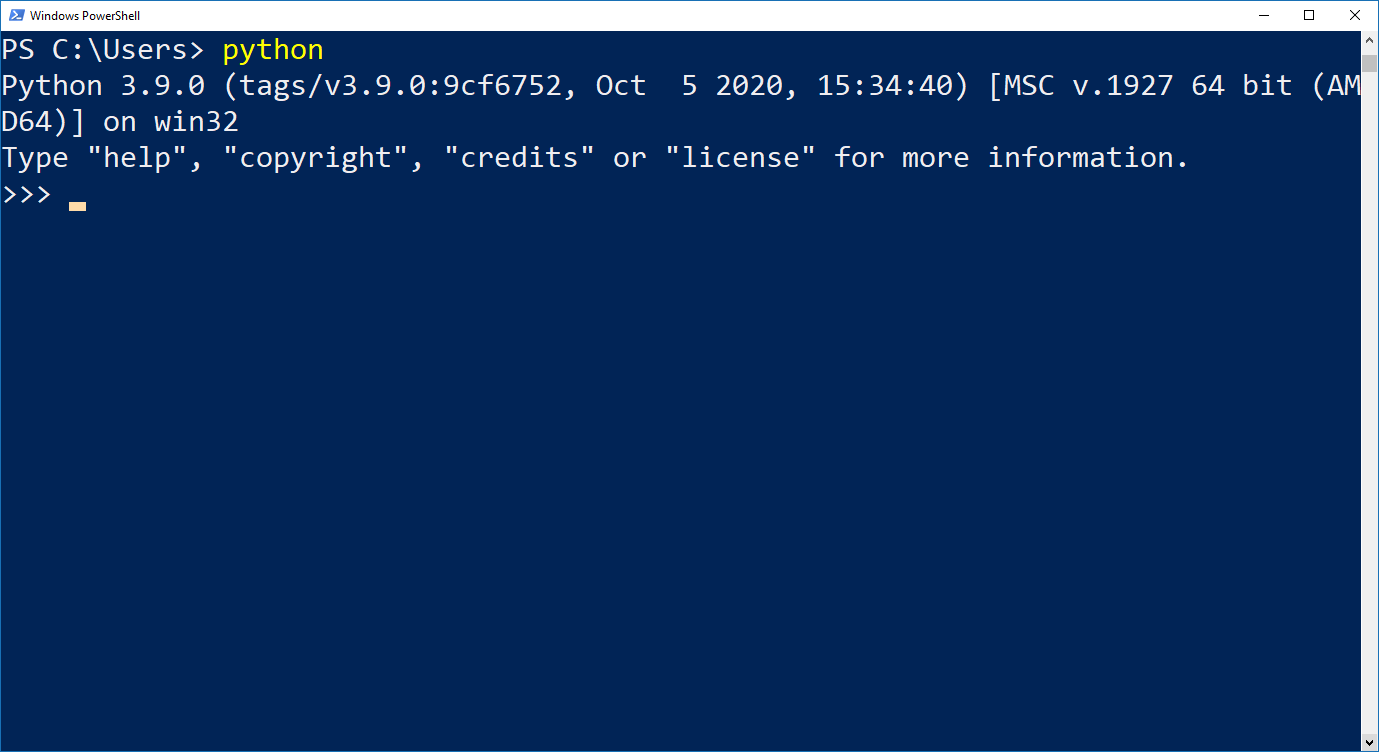


Figure 1.7 Launching Python in Windows PowerShell

Once the Python interpreter has been started, we can see a very similar screen layout as shown in Figure 1.5. Now we can type in our Python code and immediately see the output once we have pressed ENTER. For instance, if we would like to do a simple calculation such as 2 + 7, we simply type in this equation and press ENTER. Python will interpret what we have typed in and print the result in the next line.

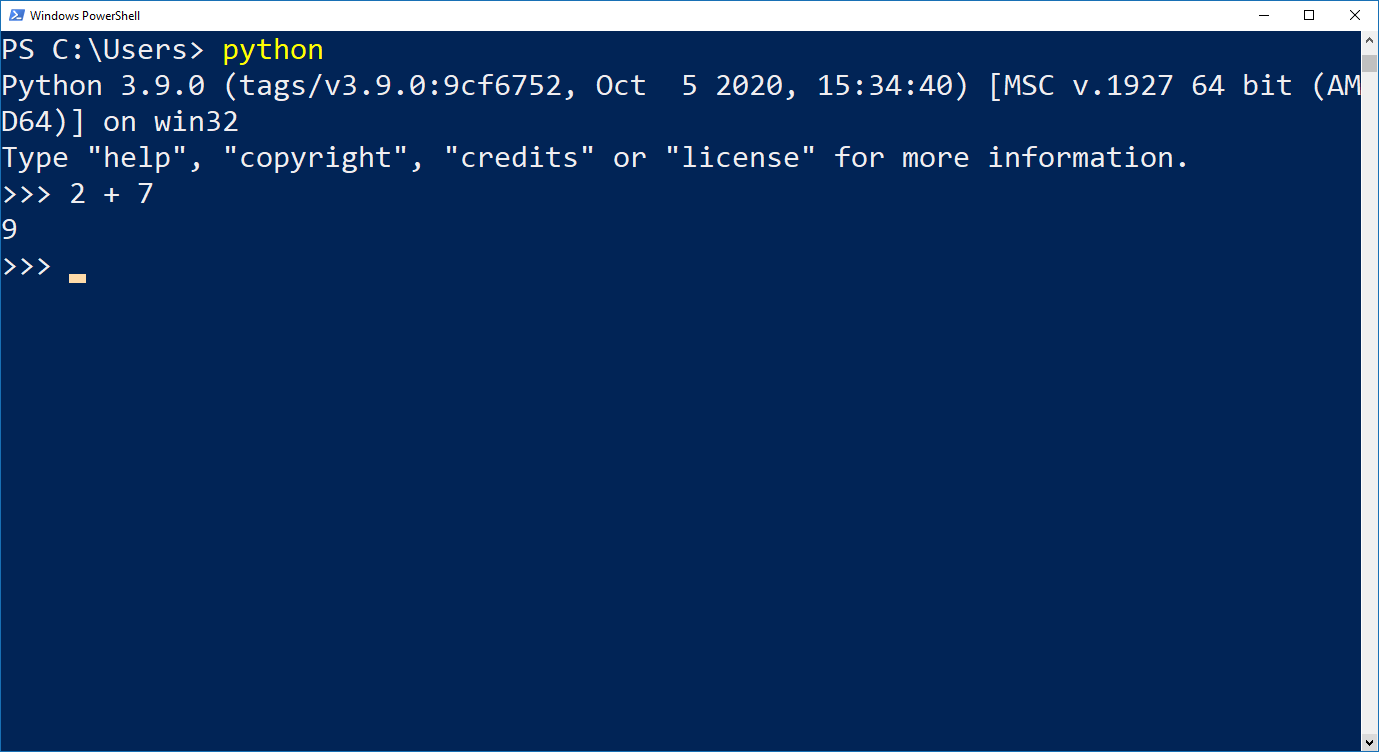


Figure 1.8 Execute a Simple Python Code

After we have finished executing our Python programs, we can quit Python by entering quit() and then press ENTER. We will return to the prompt of the operating system same as what we had seen before we started Python.

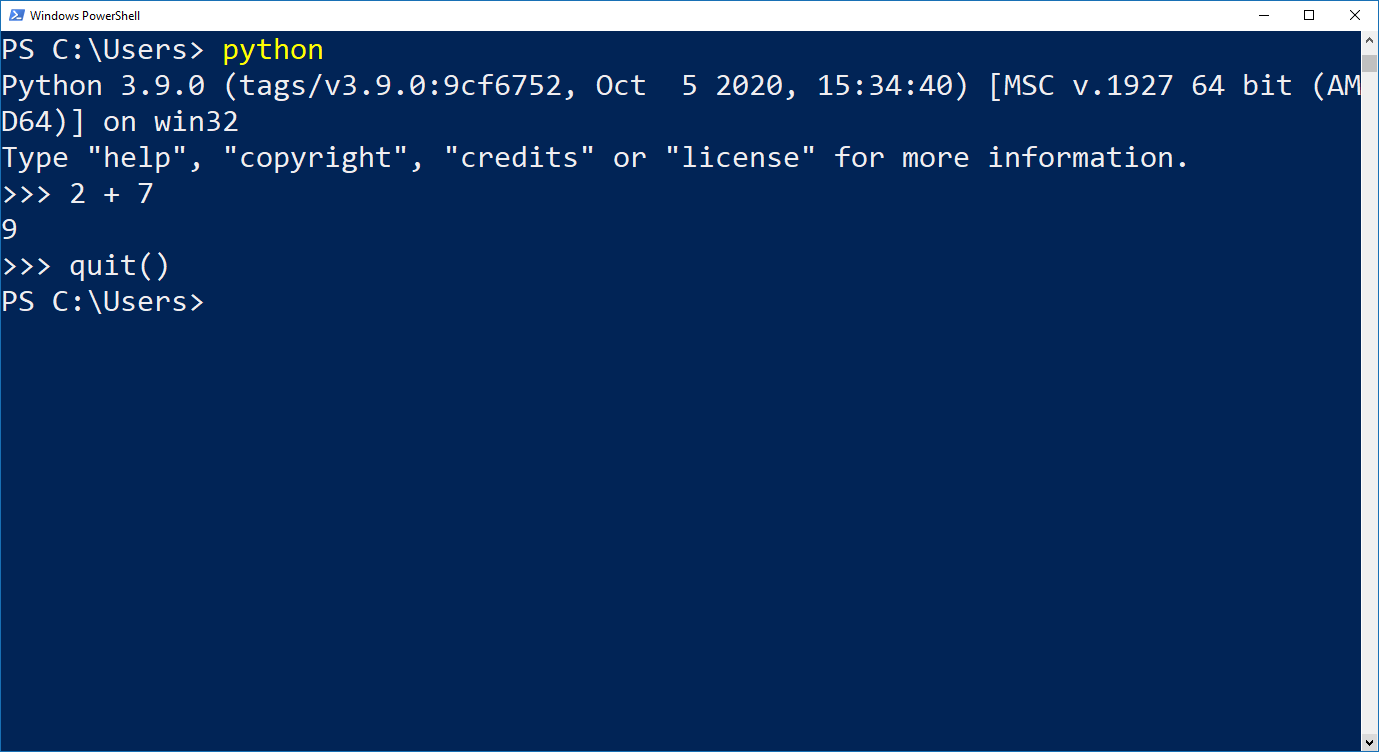


Figure 1.9 Quit the Python Interpreter

Apart from interactively working with the Python interpreter, we can also let Python run our own program scripts. These scripts are text files saved with the extension .py. In these script files, we put all the Python codes to be executed in a batch, instead of typing in and executing the syntax line by line like in the interpreter.

One obvious advantage of using a script file is that in most of the cases, we may intend to do a couple of calculations and data manipulation steps before asking Python to return the final output to us. Some of these executions could be quite inconvenient, or perhaps even impossible, if we must run them line by line.

We will use Atom as the editor to compose our Python scripts in the first part of this study guide, and then execute these scripts using the python command in PowerShell.

Now, we open Atom and write our first program.

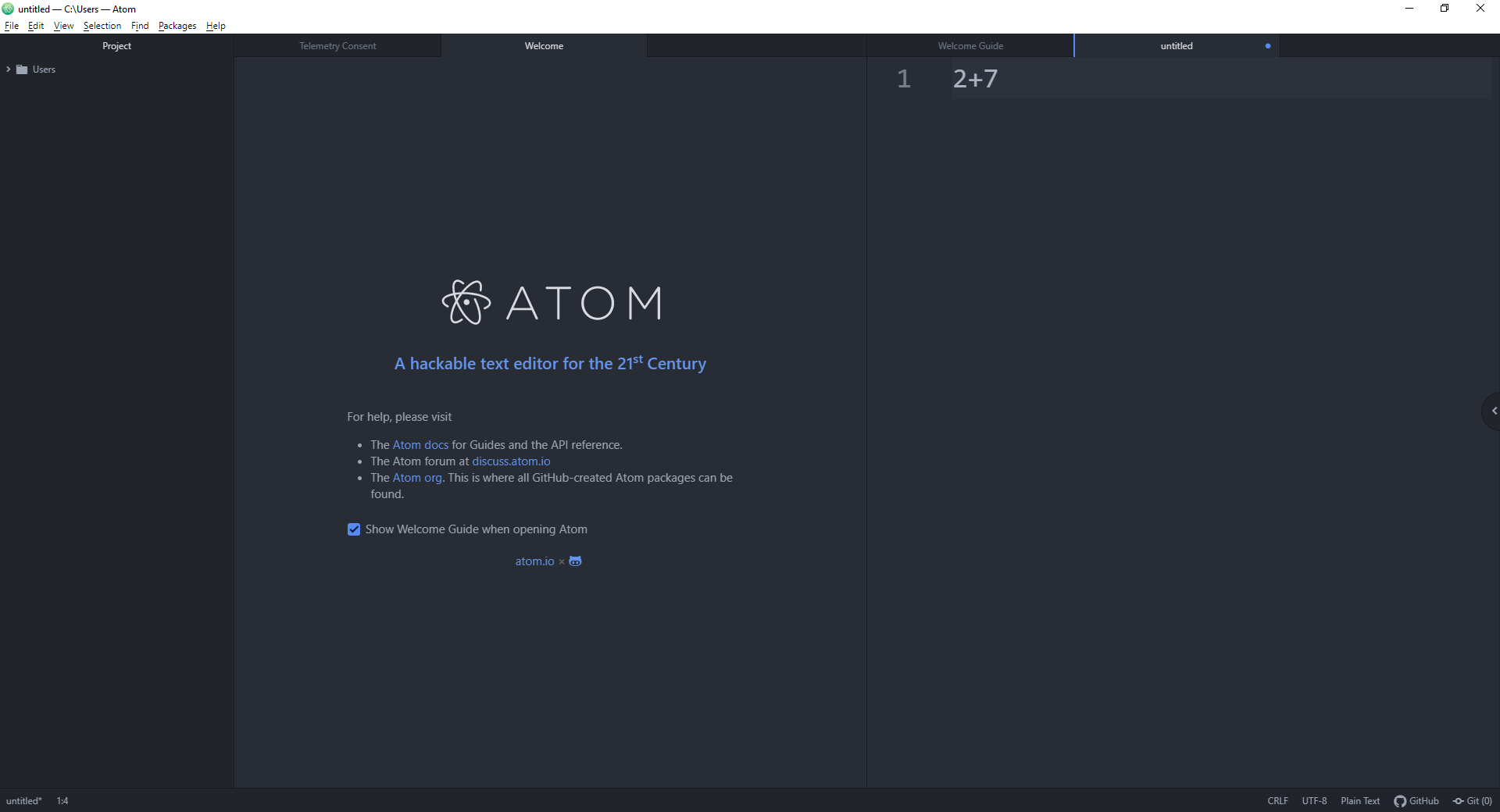


Figure 1.10 Writing Python Code with Atom

Another advantage of writing Python code in a script is that we can add comments to it. Comments are lines in a script that will not be executed by Python. We can use comments to explain the procedures that the Python code is executing. Including comments is important since they will make it more readable and understandable for future editing or debugging, and simplify the overall maintenance of the program.

Comments in Python scripts start with a hash (#). After the hash, we can type in our explanations or descriptions of the referred syntaxes. Comments can be placed as a complete single line or after a line of syntax. If we need multiple lines for our comments, we will have to start every comment line with a hash.

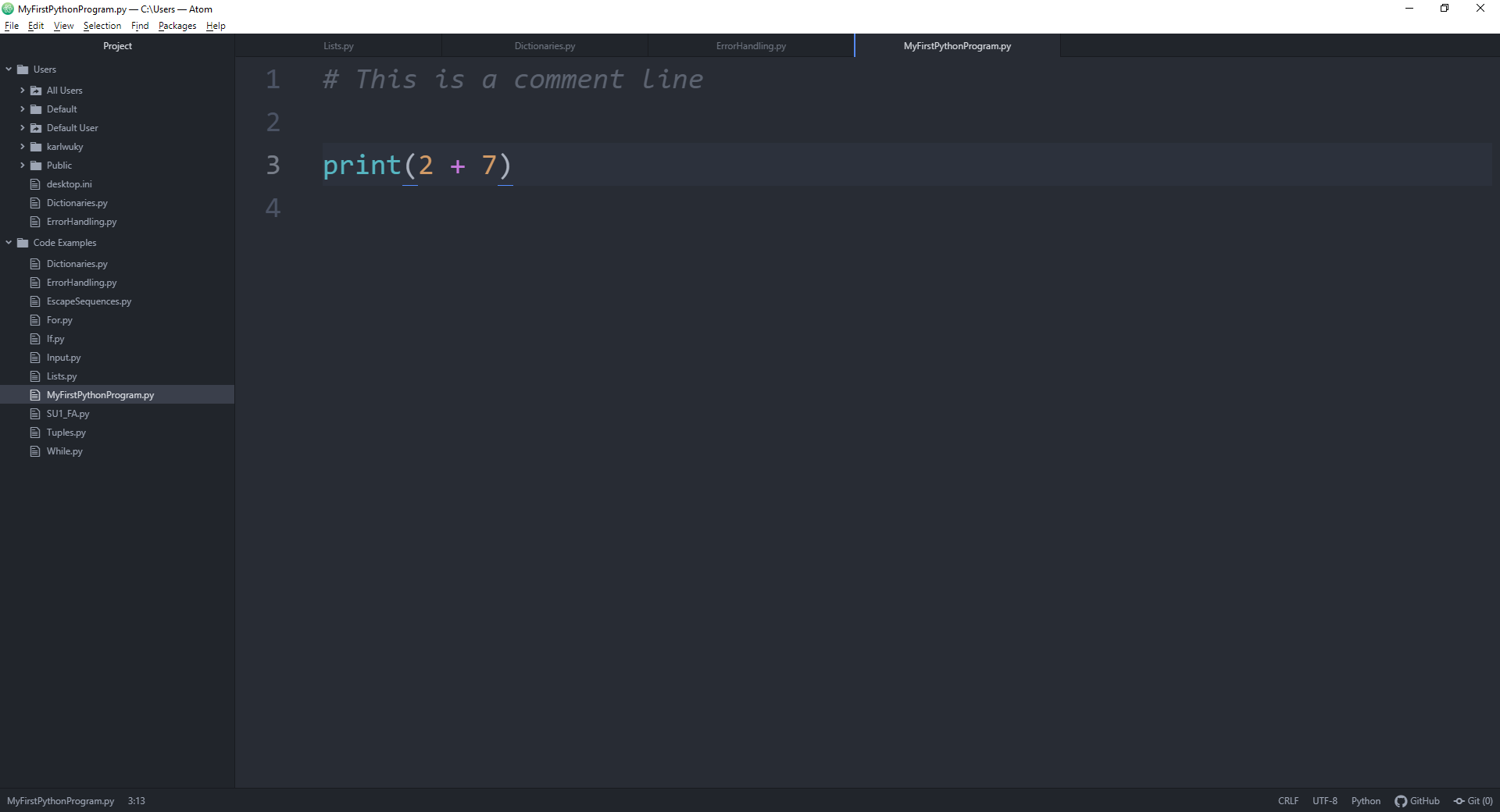


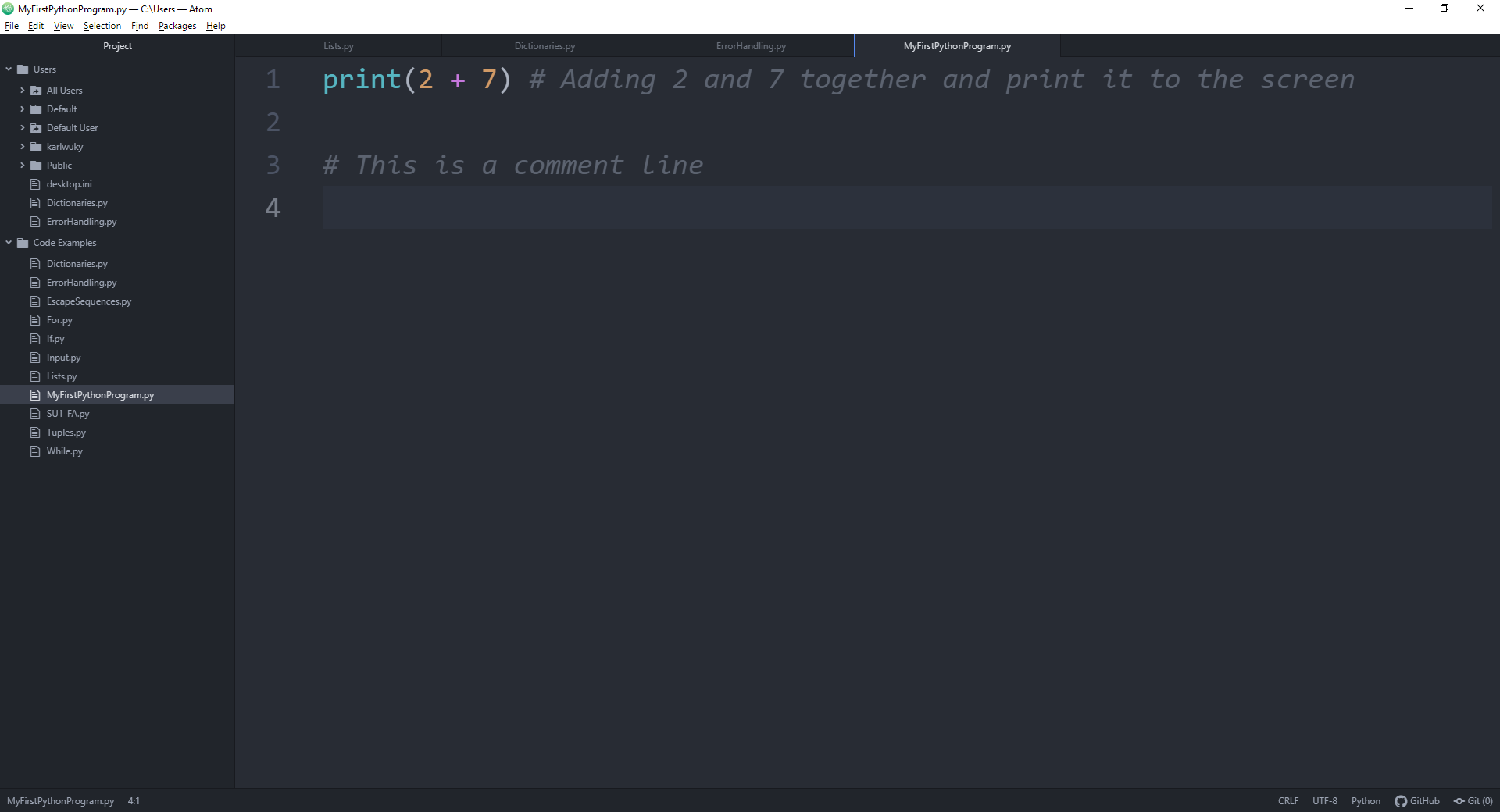
Figure 1.11 A Comment Line in Python

After editing the Python commands in a script using Atom, we can save the script as a .py text file. Different from the Python interpreter, we need to use the print() function explicitly to generate an output to the screen while Python is executing the script.

In the PowerShell, we need to change to the directory where we have saved the Python script and then run the script by executing the following command:

|  |
| --- |
| python filename.py |

Note that filename is the file name of our Python script.



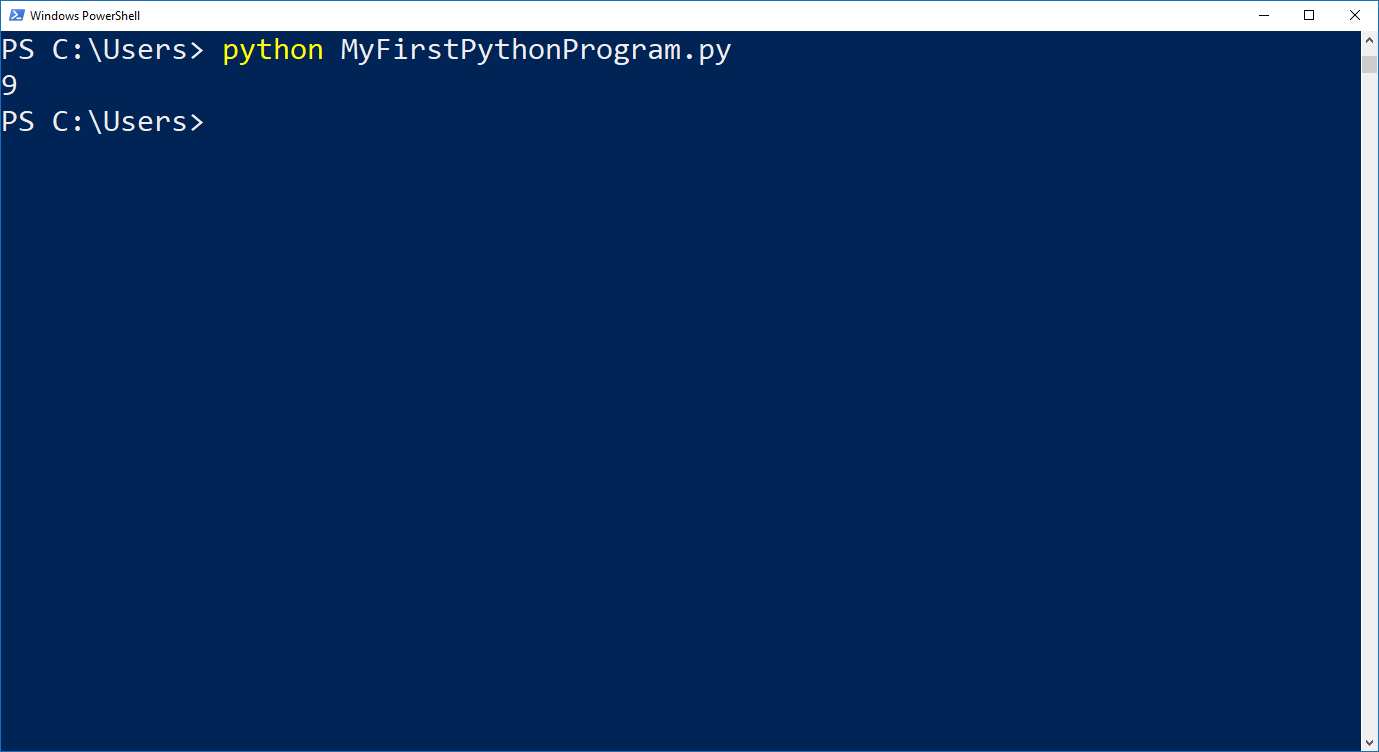


Figure 1.12 Executing a Python Scripts written in Atom Using PowerShell

Figure 1.12 shows us how PowerShell presents the output of a Python script. Since our program asks Python to print out the result of the addition 2 + 7, the Python interpreter will execute the arithmetic operation in the background and return the value to the function print() for output. After Python has executed the whole script, it will return to the prompt of the operating system.

**Read**

Read the following section of the textbook on examples of composing and executing Python scripts:

Exercise 1 A Good First Program (Windows)

Exercise 1 A Good First Program (macOS)

## Chapter 2 Basic Arithmetic and Variables

Lesson Recording - Basic Arithmetic and Variables in Python

### 2.1 Arithmetic Operators

Before we start writing more sophisticated programs, we shall first go one step backwards and familiarise ourselves with the most origin function of a computer: calculation. Python can be powerful in many ways, but we can also use it for very trivial tasks such as adding two numbers together. In Figure 1.8 and Figure 1.12, we instruct Python to carry out a simple addition 2 + 7 for us. Similarly, we can also command Python to do other basic arithmetic operations.



Figure 1.13 Simple Calculations with Python

The following Python arithmetic operators are available for mathematical calculations:

Table 1.1 Python Arithmetic Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Function** | **Example** |
| + Addition | Adds values on either side of the operator | 10 + 20 = 30 |
| - Subtraction | Subtracts right-hand operand from left-hand operand | 10 – 20 = -10 |
| \* Multiplication | Multiplies values on either side of the operator | 10 \* 20 = 200 |
| / Division | Divides left-hand operand by right-hand operand | 20 / 10 = 2 |
| % Modulus | Divides left-hand operand by right-hand operand and returns remainder | 20 % 10 = 0 |
| \*\* Exponent | Performs exponential (power) calculation on operators | 10 \*\* 20 = 10,000,000,000,000,000,000 |
| // Floor Division | The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) | 9 // 2 = 4  9.0 // 2.0 = 4.0  -11 // 3 = -4  -11.0 // 3 = -4.0 |

(Source: <https://www.tutorialspoint.com/python/python_basic_operators.htm>)

Same as normal mathematics, the exponent operator has higher priority than the operators of multiplication or division, which in turn will be calculated prior to the addition or subtraction operators. Furthermore, we can add parentheses to our equation to indicate that those terms within a parenthesis should have the highest priority in the calculation.

Note that mathematical functions such as the square root, the logarithm, the exponential, or the trigonometrical functions are not included in basic Python. If we want to include these functions in our calculation, we will need to import packages such as “math” or “NumPy” in our code. We will discuss how to import and call functions from external libraries or packages in Study Unit 2.

And then there are other operators in Basic Python such as relational operators, logical operators, etc. We will also discuss them at a later stage of this study unit.

**Read**

Read <https://www.tutorialspoint.com/python/python_basic_operators.htm> for more about basic operators in Python.

### 2.2 Variables

In most of the situations, we wish to write programs that help us automate routine operations without adjusting our programs according to the actual needs. For instance, we may not always want to add 2 and 7 together. Instead, we would prefer to let the computer add up any pair of arbitrary numbers for us, and we can choose these numbers depending on the situation. As a result, we would like to keep our program as general as possible by using variables instead.

In python, we define a variable by its name which is an arbitrary combination of characters (A-Z, a-z), underscores (\_) and numbers (0-9). Subsequently, we assign a value to the variable and let Python operate with it. And we can change the value of the defined variable at any stage of our program.

To assign a value to a variable, all we need to do is:

|  |
| --- |
| variable = value |

Remember that it is important to put the variable left of the equal sign (=) and the value right of it. If we switched their positions, it would be equivalent to assigning a name to a number. This would result in a syntax error, and Python will stop executing the rest of the program at once.

In Python, the name of a variable can be short (e.g., x, y, z) or more descriptive (e.g., age, carname, total\_volume). But there are certain rules which we must follow when we create our variable names.

* A variable name must start with a letter or an underscore (\_).
* A variable name cannot start with a number.
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_).
* Variable names are case-sensitive (age, Age and AGE are three different variables).

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

Here are some examples of valid variable names:

|  |
| --- |
| myvar = 10  my\_var = 10  my\_var = 10  myVar = 10  MYVAR = 10  \_myvar2 = 10 |

Here are some examples of invalid variable names:

|  |
| --- |
| 2myvar = 10  my-var = 10  my var = 10 |

Once values are assigned to variables, we can use them for any arithmetic operations as introduced in Chapter 2.1 for numeric values.

|  |
| --- |
| **Example (Students’ score):** Suppose we have the exam scores of two students, 30 and 65, and we would like to store them in two variables, score1 and score2, for some mathematical operations. Subsequently, we can conduct arithmetic operations with these variables.    Figure 1.14 Assigning Values to Variables for Operations |

**Read**

Read the following section of the textbook on examples of creating and using variables in Python:

Exercise 4 Variables and Names

### 2.3 Types of Variable and Expressions

In Python, there are different types of variable that we can work with. In the previous section, we assign numeric values to variables which makes them numeric variables. Nevertheless, there are also different types of numeric variable. Here are some main types of variable available in Python:

Table 1.2 Types of Variable in Python

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Example** |
| Integer | The value of an integer variable must be an integer, a value without decimal point. It can be both positive and negative. | a = 5 |
| Float | The value of a float variable can be an arbitrary numeric value with a floating point. | b = 10.5 |
| String | The value of a string variable can contain any letters in both cases, special characters as well as numbers. Note that if numbers are assigned to a string variable, no mathematical operations can be applied on it.  To assign a value to a string variable, the value must be written between a pair of quotation marks (it can be single or double quotation mark, but it must be consistent for the same value). | c = "John"  d = "Tan" |
|  | Furthermore, two strings can be concatenated by being “added up”. | c + d = "JohnTan" |
| Boolean (Bool) | The value of a Boolean variable can be either True or False. | e = True |

In the following, we will use the general term *expression* for variables or when they are linked with operators. For instance, a + b is an expression and not a variable, unless we define c = a + b in our program where c is then a new variable. However, we would rather refer to expressions in our program directly since we do not always define new variables for calculation steps in between.

To check the type of a variable, we can use the type() function on any variable in our program.

|  |
| --- |
| type(variable\_name) |

Python will then print the variable type such as “int” (for integer), “float” (for float), or “str” (for character string) to the screen.

|  |
| --- |
| **Example (Cont’d):** In Figure 1.14, we assign 30 and 65 to the two variables, score1 and score2, respectively. We can use the type() function to check their variable type.    Figure 1.15 Checking Variable Type  We can see that Python returns <class 'int'> as the screen output. The information we are enquiring is put in the single quotation marks like int in this case. As a result, we can see that both score1 and score2 are integer variables. |

**Read**

Read the Python documentation (<https://docs.python.org/3/library/stdtypes.html#‌numeric-types-int-float-complex>) for more about the different operations on numeric data types.

## Chapter 3 Print and Input

Lesson Recording - Print and Input in Python

### 3.1 Printing

Writing programs is not only to automate routine operations by the computer. It is also of interest to show the results, information, or messages to the user while the Python program is running. We can use the print() function to generate screen output for the user to read.

|  |
| --- |
| print("My String") |

All we need to do here is to put the text within a pair of quotation marks and pack everything inside the print() function. While the program is being executed, Python will then extract the content within the quotation marks and print it onto the screen.

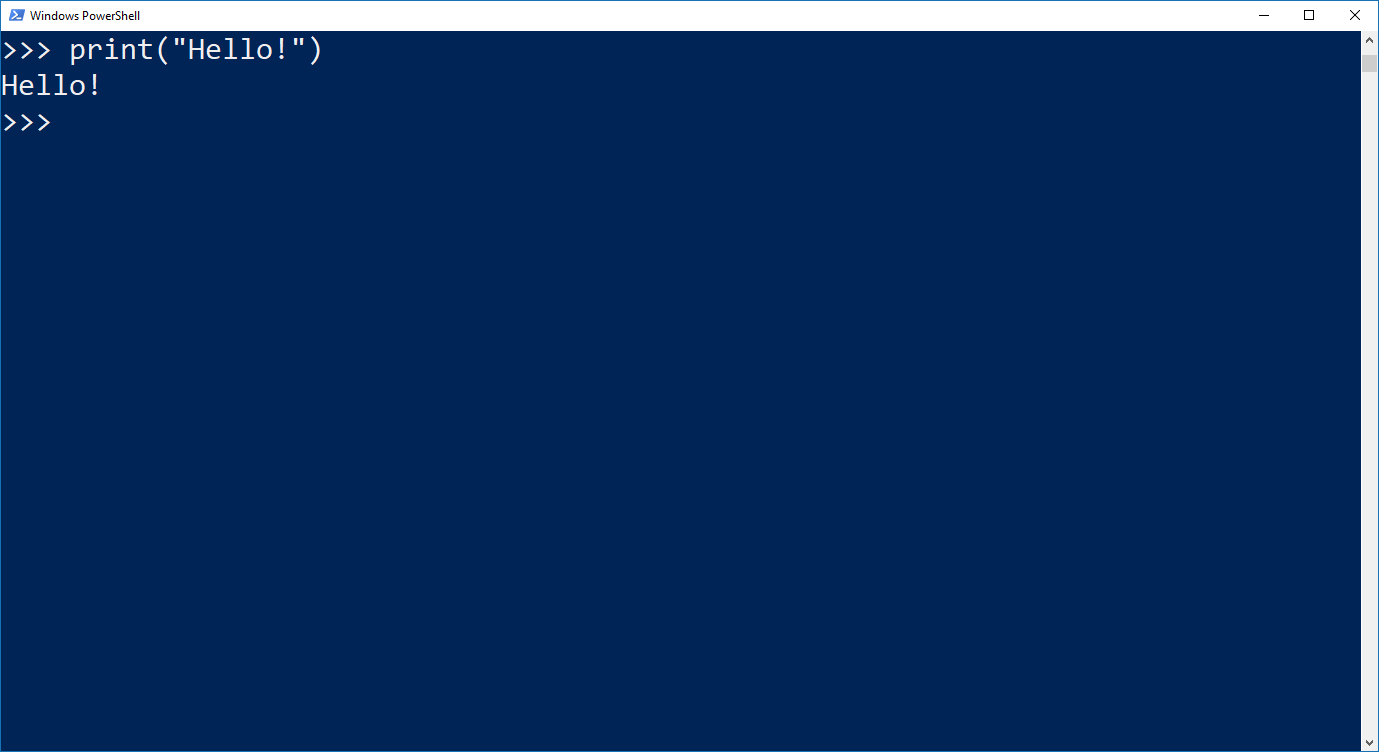


Figure 1.16 Simple Use of the print() Function

The print function is not only limited to print pre-defined strings. We often wish to print out the result of a calculation, as shown in Figure 1.8 and Figure 1.12, or the value of a variable, or the result of a calculation based on variables as well.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like Python to print the exam score of student1, score1, and the sum of the two scores, score1 + score2, onto the screen.    Figure 1.17 Printing Variables and Their Calculations |

### 3.2 String Formatting

In the last command of Figure 1.17, we ask Python to first execute the mathematical operation score1 + score2, and then print it out to the user. Basically, there is nothing wrong with this screen output, it only looks a bit “cold” and rather not interactive. To make such an output to look more like a statement addressing to the user, we can mix a statement with variables or expressions in our printing string to become a formatted string for printing.

|  |
| --- |
| print(f"My String {expression1} {expression2} …") |

The print-command is almost identical to the one for normal printing. The only difference here is to put an “f” before the open quotation mark. Subsequently, we can place the variables or expressions that we would like to print anywhere within the text and wrap it within a pair of curly brackets {}.

|  |
| --- |
| **Example (Cont’d):** Suppose we would like to print the scores of the two students as well as their sum in a sentence such as “Our scores are 30 and 65. The total score is 95”. We can use the following code to create this screen output.    Figure 1.18 Printing Formatted String |

Unlike the entire argument within the quotation marks, every expression written inside the curly brackets of a formatted string will be evaluated before being printed onto the screen. In Figure 1.18, the expressions score1, score2 and score1 + score2 will be evaluated first. That is, Python will execute the print() function with the value assigned to these expressions and not with the expressions as part of the string.

|  |
| --- |
| **Example (Cont’d):** If the variables or expressions were not put inside some curly brackets as a formatted string, the expressions score1, score2, and score1 + score2 would be treated as ordinary strings and printed just as how they were written. And if the “f” were forgotten at the beginning of the argument in the print() function, Python would interpret the missing of the “f” as an instruction to print the entire text within the quotation mark without evaluating the expressions in the curly brackets first. Since the curly brackets are part of the string in the print() function, they will be printed as well.    Figure 1.19 Output of Incomplete String Formatting Syntax |

We can also use the .format() method for string formatting. Note that the .format() method only takes one expression in its argument.

|  |
| --- |
| print("My String {}".format(expression)) |

For the .format() method, we need to place the curly brackets at the position within the string where we would like to print our expression.

|  |
| --- |
| **Example (Cont’d):** We can print the total score of the two students at the end of a statement such as “The total score is 95.” by using the .format() method.    Figure 1.20 Simple Usage of the .format() Method  Figure 1.21 shows us how the printing string can be extended in order to get the same output as in Figure 1.18. But the syntax is much longer here.    Figure 1.21 Multiple Usage of .format() Method  Nevertheless, the .format() method can also be useful if we have one variable to be printed at the end of our statement. |

**Read**

Read the following section of the textbook on printing formatted strings:

Exercise 5 More Variables and Printing

**Read**

Read the following two sections of the textbook on printing formatted strings using the .format() method:

Exercise 6 Strings and Text

Exercise 7 More Printing

### 3.3 Escape Sequences

Escape sequences are used to print special characters that are invisible such as ENTER, or characters that can cause syntax error such as single (') or double quotation marks ("). Suppose we would like to include a quote within a string for the screen output.

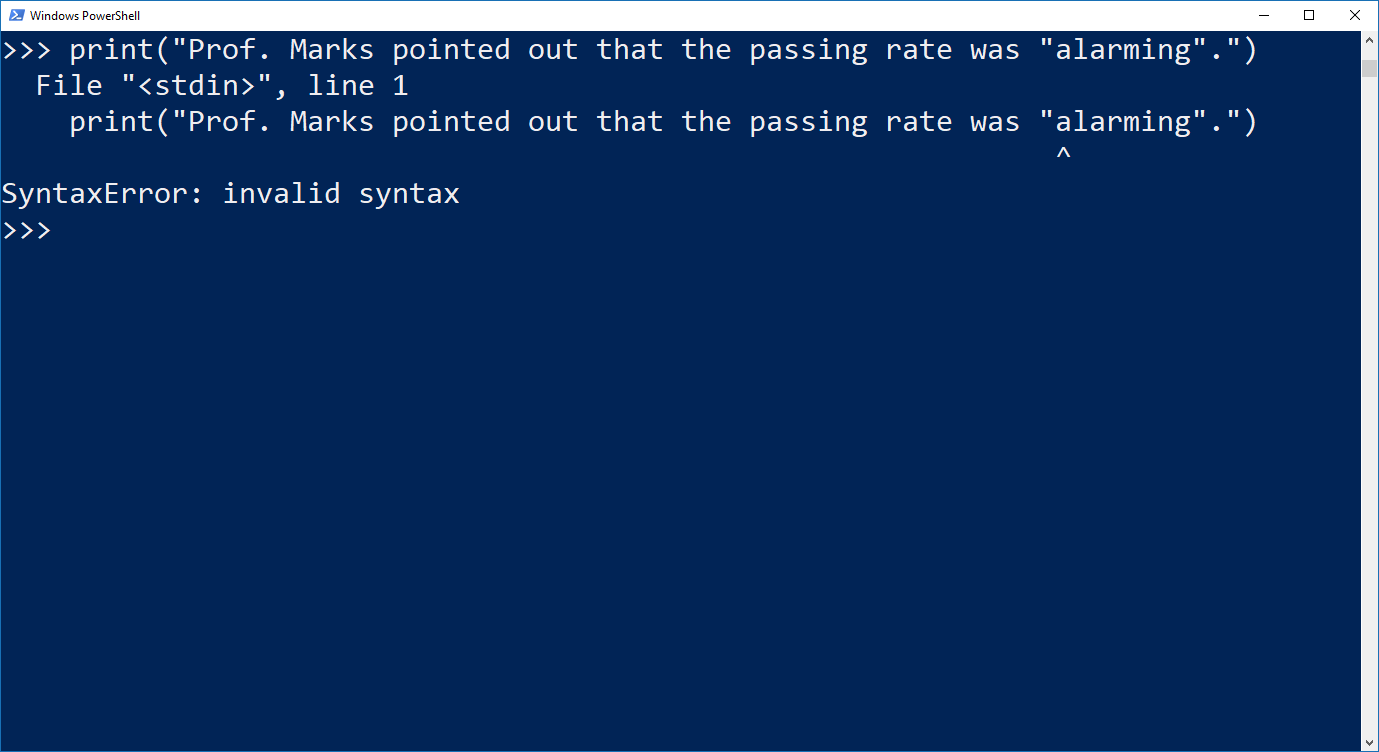


Figure 1.22 Syntax Error Caused by Quotation Marks within a string

In Figure 1.22, the string in the print() function ends with the second quotation mark. Everything subsequent to it will be interpreted as part of the code. Since the word “alarming” is neither a Python command nor a variable, Python simply interprets it as an erroneous syntax. One way to avoid this error is to use single quotation marks for either the citation quote or the string definition.

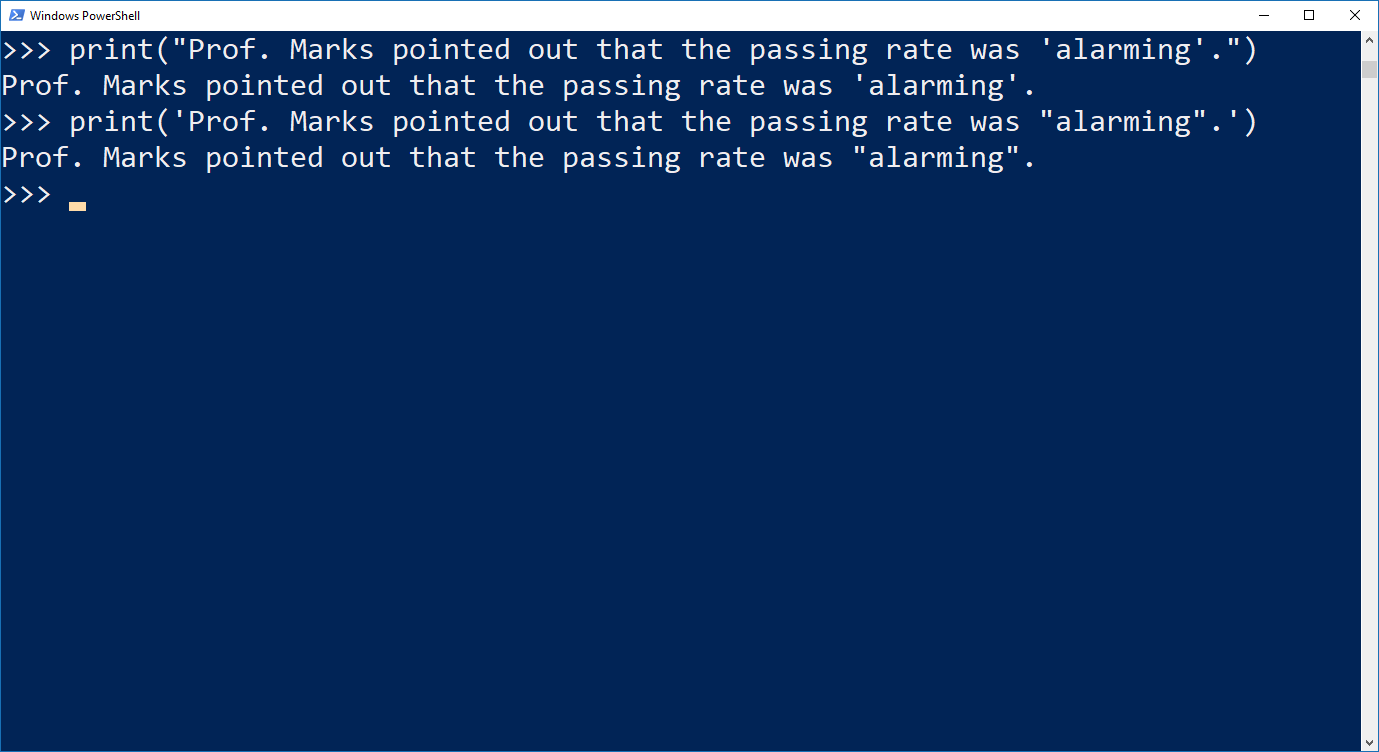


Figure 1.23 Printing Quotation Marks within a String

Another way is to use the escape sequence \" within the string instead of switching between single and double quotation marks.

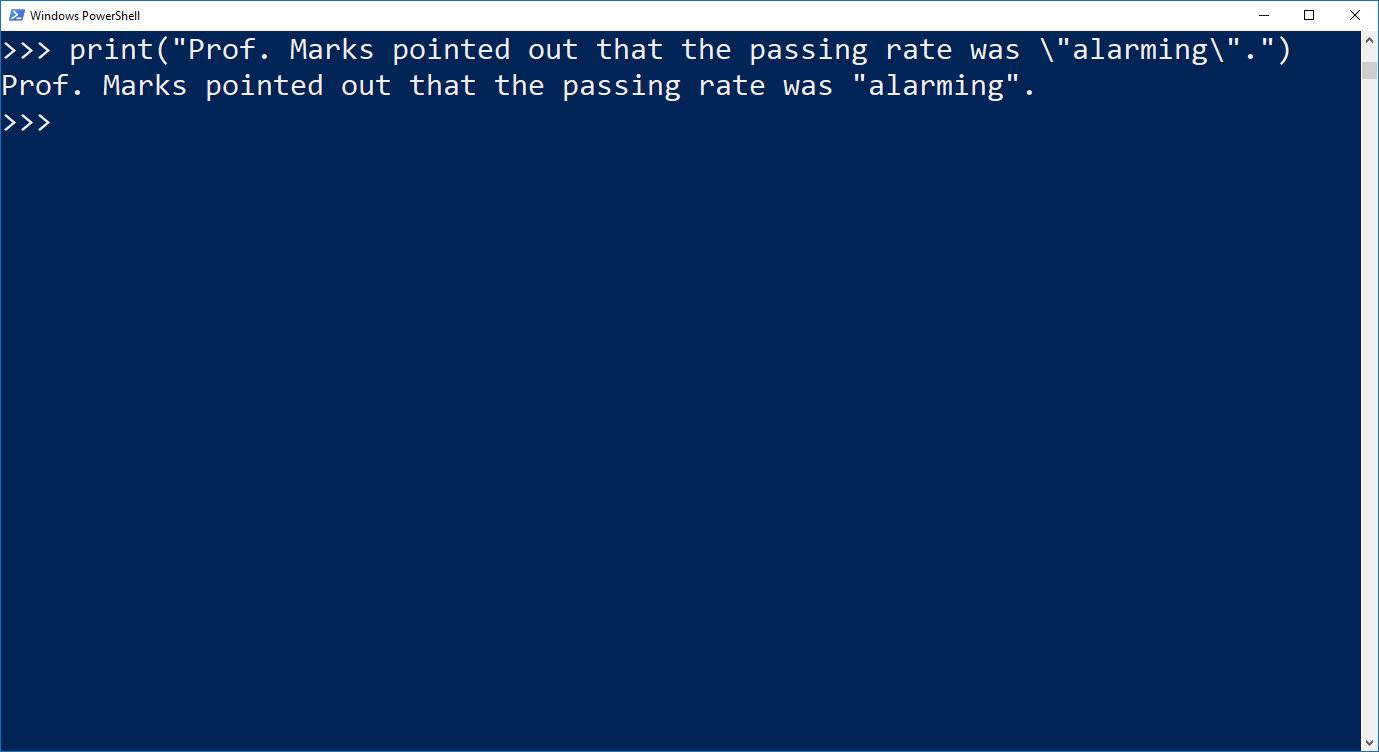


Figure 1.24 Printing Quotation Marks using Escape Sequence

Escape sequences are also useful when line breaks should be inserted within a string. By adding the escape sequence “\n” at the position within the string, the subsequent part of the string will be printed in the next line of the output screen.

|  |
| --- |
| **Example (Cont’d):** Now we would like to print the first and the second sentences in Figure 1.18 in two separate lines. However, we would create a syntax error if we just placed a line break in our Python script.      Figure 1.25 Erroneous Line Breaks within a String  Figure 1.25 illustrates that Python treats such a line break within a string as a syntax error. The reason is that the string in the print() function must be closed by a quotation mark in the same line. Instead of closing the first line directly and start a new print() function in the second line to solve this problem clumsily, we can add an escape sequence \n into the string:      Figure 1.26 Line Breaks Created by Escape Sequence \n |

The following list contains some useful escape sequences available in Python:

Table 1.3 List of Some Escape Sequences Available in Python

|  |  |  |
| --- | --- | --- |
| **Escape Sequences** | **Description** | **Example** |
| \newline | Backslash and newline ignored | >>> print("line1 \  Line2")  line1 line2 |
| \\ | Backslash (\) | >>> print("\\")  \ |
| \' | Single quote (') | >>> print("\'")  ' |
| \" | Double quote (") | >>> print("\"")  " |
| \n | ENTER or line break | >>> print("line1 \n line2")  line1  line2 |
| \b | Backspace (BS) | >>> print("line1 \b line2")  line1 line2 |
| \t | Horizontal Tab (TAB) | >>> print("line1 \t line2")  line1 line2 |
| \v | Vertical Tab (VT) | >>> print("line1 \vline2")  line1  line2 |

(Source: <https://www.python-ds.com/python-3-escape-sequences>)

**Read**

Read the following two sections of the textbook on using escape sequence in printing formatted strings as in Figure 1.8:

Exercise 9 Printing, Printing, Printing

Exercise 10 What Was That?

### 3.4 Input

While a program script is being executed, it requires values to be assigned to the variables in order to proceed in its instructions. So far, we have discussed the possibility to assign values to the variables in the script. That means, the values are fixed when the program started to run. However, in most of the cases, those values are unknown and can only be assigned while the program is running, mostly based on the input of the user. In Python, we can use the input() function to ask the user to enter the value for a variable.

|  |
| --- |
| variable = input("My String") |

The whole syntax will be put on the right-hand side of an equal sign so that Python can assign the user input to the variable that is defined on the left-hand side of the same equal sign.

Unlike the print() function, Python requires the user to type in something and then press ENTER to complete the execution of the input() function. Same as the print() function, we can instruct Python to print a string to the screen within the input() function. Usually, this string should be a question and/or some instructions to inform the user what they shall input here. Furthermore, we are also allowed to mix the assigned values of some variables with the instruction text to become a formatted string that will be printed on the screen for the subsequent input.

|  |
| --- |
| **Example (Cont’d):** Instead of pre-assigning values to the variables, we will ask the user to enter his/her name and his/her score. Subsequently, we will print out his/her score by addressing his/her name and embed his/her score in a sentence such as “Your score is …”.      Figure 1.27 Example of Using input() Function |

In Python, the value assigned by the user within an input() function will be stored as string. If the input should be an integer or a number with a floating point, we can convert the input using:

|  |
| --- |
| variable = int(input("My String")) |
| variable = float(input("My String")) |

The functions int() and float() are used to convert a string variable to become an integer or a float variable. (Conversely, there is the str() function to convert an integer or a float variable into a string variable.)

|  |
| --- |
| **Example (Cont’d):** Since the score of a student must be an integer within 0 and 100, we can convert it to an integer by embedding the input() function within an int() function. At the same time, we add a new question to ask the student for his/her CGPA and convert it to a float variable.      Figure 1.28 Convert Input Value to Integer and Float |

The syntax introduced above is to put the input() function inside the int() and float() functions and construct the instruction within a single line. Nevertheless, we can also separate these commands into two lines without changing the behaviour of the program:

|  |
| --- |
| variable = input("My String")  variable = int(variable) |

These lines are certainly applicable to the float() function as well. It is noteworthy that if the user enters a value that is not a number, the int() or float() functions will interpret it as a syntax error and stop executing the code immediately. It is therefore a good programming habit to build in certain control mechanism for any user input command in our code. We will discuss the construction of such control mechanism in Chapter 5 of this study unit.

**Read**

Read the following two sections of the textbook on getting the user input:

Exercise 11 Asking Questions

Exercise 12 Prompting People

## Chapter 4 If-elif-else-Conditions

Lesson Recording - If-elif-else-Conditions in Python

### 4.1 Boolean Expressions

To automate a routine by a computer program, we usually need to let the program “decide” what to execute in the next step based on some conditions. For instance, the user can choose to stay or quit the program after certain operations have been completed.

Before introducing the if-elif-else-conditions that Python uses to decide how the program should behave after a certain stage of the code, we need to get ourselves familiarised with the Boolean expressions first. As introduced in Chapter 2.3, Boolean variables have only two possible values: True and False. So, the basic concept of the conditional control flow is to evaluate whether a Boolean expression is True or not first, and then carry out either set of instructions depending on the evaluation.

A Boolean expression can be the result of a single relational operation or a combination of multiple relational operations linked by logical operators. Here are some relational operation examples:

Table 1.4 Examples of Relational Operation

|  |  |
| --- | --- |
| **Relational Operation** | **Result** |
| 1 == 1 | True |
| 3 > 2 | True |
| 0 <= -5 | False |
| a + b < 10 | False if a + b >= 10 |

In the above examples, the first two operations are obviously True since they correspond to the mathematical relationship between the left-hand and the right-hand sides of the equations. Note that if we want to check whether two expressions are identical, we will have to use the double equal sign (==) instead of the ordinary equal sign (=) since the single equal sign is used to assign a value to a variable. So, if we wrote 1 = 1 instead of 1 == 1, a syntax error would return since Python would interpret our intention to be assigning a value to a number, which we know is not allowed from Chapter 2.2.

Below is a list of relational operators that we can use in Python.

Table 1.5 List of Relational Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | True if the values of two operands are equal. | (10 == 20) is False. |
| != | True if values of two operands are not equal. | (10 != 20) is True. |
| <> | True if values of two operands are not equal (Similar to the != operator). | (10 <> 20) is True. |
| > | True if the value of left operand is greater than the value of right operand. | (10 > 20) is False. |
| < | True if the value of left operand is less than the value of right operand. | (10 < 20) is True. |
| >= | True if the value of left operand is greater than or equal to the value of right operand. | (10 >= 20) is False. |
| <= | True if the value of left operand is less than or equal to the value of right operand. | (10 <= 20) is True. |

(Source: <https://www.tutorialspoint.com/python/python_basic_operators.htm>)

A Boolean expression can also be a combination of multiple relational operations, connected by the logical operators. Below is a list of logical operators in Python.

Table 1.6 List of Logical Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and | If both the operands are True, then the condition becomes True. | (10 > 0 and 20 > 0) is True. |
| or | If any of the two operands are non-zero, then the condition becomes True. | (10 > 0 or 20 > 0) is True. |
| not | Used to reverse the logical state of its operand. | not(10 > 0 or 20 > 0) is False. |

(Source: <https://www.tutorialspoint.com/python/python_basic_operators.htm>)

The use of a single logical operation is usually quite straightforward as the operators are designed in a way that it just simply matches our spoken language. However, it could become quite confusing if we combine these operators in a Boolean expression. For instance, the following Boolean expressions are equivalent:

|  |  |  |
| --- | --- | --- |
| (a or b) and c | <==> | (a and c) or (b and c) |
| (a and b) or c | <==> | (a or c) and (b or c) |
| not(a or b) | <==> | (not a) and (not b) |
| not(a and b) | <==> | (not a) or (not b) |

These are just a few examples and can be extended endlessly. It is utmost important to get familiarised on how to create Boolean expressions using relational and logical operators. Any failure in combining these operators could lead to unexpected behaviour of our program. The only, and most effective way here is to practise them with Python since we can check on the result directly.

**Read**

Read the following two sections of the textbook on Boolean expressions:

Exercise 27 Memorising Logic

Exercise 28 Boolean Practice

### 4.2 Conditional Statements

The result of a Boolean expression can serve as the condition that changes the behaviour of a program dynamically by embedding it in an if-conditional statement:

|  |
| --- |
| if condition:  instructions |

In the if-condition, Python will execute the syntaxes in the instructions if the condition is True. However, if the condition is False, Python will simply skip these lines and proceed with the subsequent code lines. Note that it is mandatory to put the colon directly behind the condition, and the instructions must be indented so that Python can interpret them as part of the if-block.

|  |
| --- |
| **Example (Cont’d):** If the score of a student is below 40, we will show a message on the screen to tell him/her that he/she failed in the exam.      Figure 1.29 if-Statement Example with True Condition  Figure 1.29 illustrates what Python does if the condition is True. On the other hand, if a student scores more than 40, nothing will be printed based on the Python script.    Figure 1.30 if-Statement Example with False Condition  Figure 1.30 shows how Python skips all the instructions in the if-block since the condition is False. |

If we intend to let Python execute another set of instructions if the condition is False, and not just skip the if-block, we can add an else-statement to the if-block:

|  |
| --- |
| if condition:  instructions 1  else:  instructions 2 |

Same as the if-condition, we must add a colon to the else-statement and the instructions following it must be indented as well.

|  |
| --- |
| **Example (Cont’d):** If the score of a student is below 40, we will show a message on the screen to tell him/her that he/she failed in the exam. Otherwise, we will show a message to tell him/her that he/she passed.      Figure 1.31 if-else-Statement Example  Figure 1.31 shows that if the condition is False, Python will execute those instructions following the else-statement. |

If the construction of the condition allows more than two outcomes, we may need a third or fourth if-blocks, etc. In this case, we can use the if-elif-else-block:

|  |
| --- |
| if condition 1:  instructions 1  elif condition 2:  instructions 2  else:  instructions 3 |

Note that an if-elif-else-block does not necessarily need an else-statement. But we should ensure that the conditions being checked by the if-statement and the elif-statements must cover all possible outcomes, unless we are certain that only those possibilities are being uncovered which do not need any instructions to follow up.

In the example in Figure 1.30, the program is only constructed to separate students into two categories: Pass and fail. It will then print the statement to the user accordingly. Suppose we also give grades to evaluate the performance of the students, we can categorise the scores using if-conditions.

If we construct an if-block to categorise a numeric variable, we should ensure that the Boolean expressions do not overlap. For instance, if grade A is assigned when the score is between 80 and 100, then 80 should not be included in the condition for getting grade B. The logical operator and should be used to indicate the interval for each category since both conditions, namely that the value of the numeric variable must be larger than the lower bound, as well as smaller than the upper bound of the interval, must be fulfilled simultaneously.

In the example shown in Figure 1.32, the else-statement has also been omitted since all possible outcomes of the variable score have been covered by the if-block. Nevertheless, we can also use the else-statement instead of the whole elif-condition for grade A if we are confident to do so. Just be cautious that in this case, if certain possibilities were not covered, no instructions would be carried out from the if-block, and the behaviour of the subsequent part of the program may be affected.

|  |
| --- |
| **Example (Cont’d):** We will print the grade to the student according to his/her exam score. If a student scores between 80 and 100, his/her grade will be A; and if his/her score is between 80 and 60, he/she will get a B; a score between 50 and 60 is equivalent to grade C; grade D will be given if a student scores between 40 and 50 and any score below 40 belongs to grade F.      Figure 1.32 Example of if-elif-else-Statement  Lastly, the print() function is not indented here. As a result, Python interprets it as the part of the code that should be executed after the entire if-block and not as part of the instructions following the last condition (elif score1 >= 80 and score1 <= 100:). |

**Read**

Read the following three sections of the textbook on conditional statements for control flow:

Exercise 29 What If

Exercise 30 Else and If

## Chapter 5 Loops

Lesson Recording - Loops in Python

### 5.1 While-Loops

In the student score example, we construct a program in which the name and the mark of one student can be entered. In the early stage of our example, we had scores of two students. If we had to enter their names and then assign a grade to each of them, we would have to repeat the codes in Chapters 2, 3 and 4 twice.

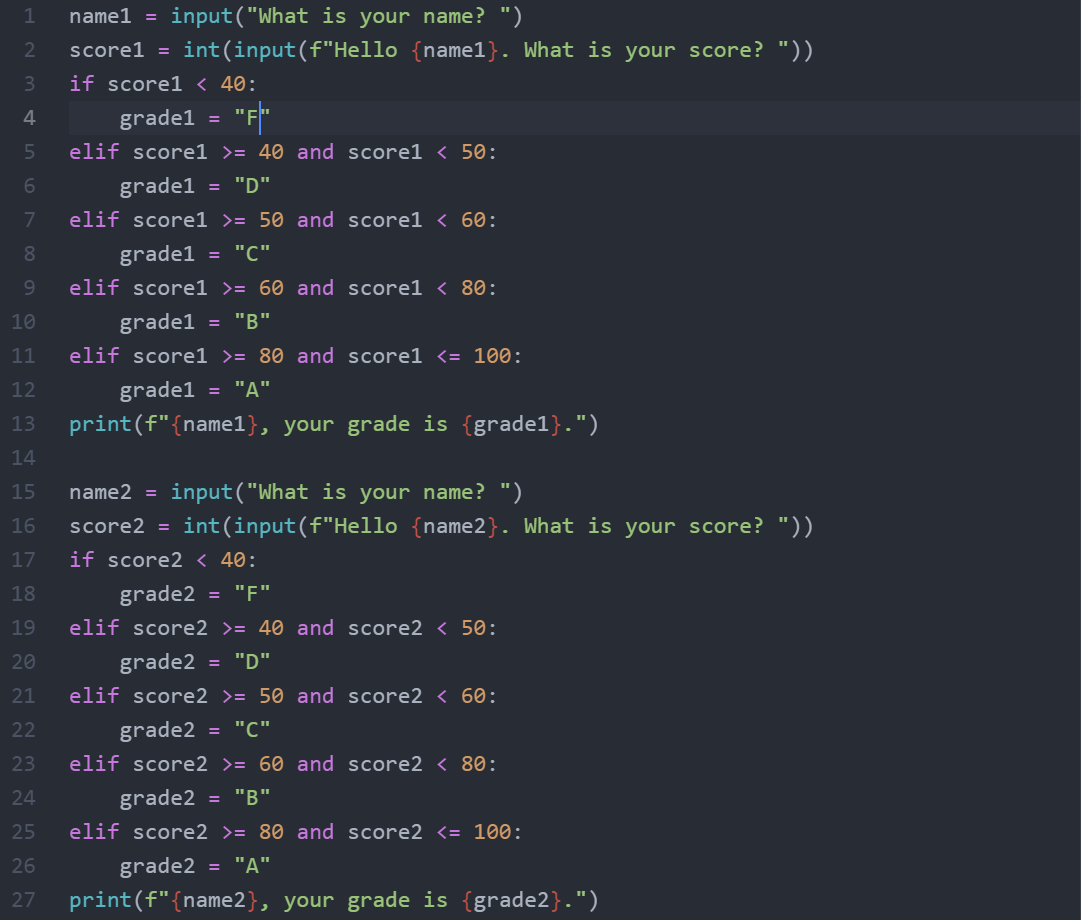


Figure 1.33 Repeating the Same Code for Two Individuals to Enter

Usually, we have more than two students in one class. In order not to expand our codes endlessly and make it clumsy and unreadable, we can construct a loop in our program to repeat the instructions that will be applied for many times. The first type of loops that we will introduce here is the while-loop.

|  |
| --- |
| while conditions:  instructions |

The condition is a Boolean expression which controls whether the loop will continue to run for a new iteration or not. If the condition is True, Python will go on to execute the instructions that are written with indentation and after the colon behind the conditions. The number of loops can be infinite and will be repeated as long as the condition is True. As a result, it is extremely important to ensure that a while-loop will be terminated at some stage by fulfilling the exit condition.

|  |
| --- |
| **Example (Cont’d):** We will repeat the previous task for 3 students.      Figure 1.34 Entering Names and Scores Repeatedly by while-Loops  Before the loop starts, we initiate a counter variable called i with the value 0. This counter will increase by 1 in each iteration. The while-loop is set to continue as long as i has not reached 3 yet. At the beginning of the last iteration, i should be 2 and will become 3 when Python executes the instructions within the while-loop. At the end of this iteration, Python will go back to the first line of the while-loop, and since the condition i < 3 is no longer True, the program will exit the loop as designed. Therefore, the most crucial command in the whole loop is i = i + 1. Without this line, i can never reach 3, which is the exit condition, after three iterations. Instead, it will stay at 0, the initial value defined before the loop has started, forever. |

**Read**

Read the following section of the textbook on while loops:

Exercise 33 While Loops

### 5.2 For-Loops

Another type of loops is the for-loop, which are constructed differently. While we need a True-condition for a while-loop to continue to iterate, we need a list for the running of the for-loops. We will discuss the construction and properties of a list in detail in Study Unit 2. Here, we will first learn to generate a simple list of consecutive integers by the range() function.

|  |
| --- |
| variable = range(start, end) |

The start value can be any integer as long as it is smaller than the end value. Note that the end value is not included in the list. In other words, the list will end at end – 1. The generated list of numbers will be assigned to the variable defined on the left-hand side of the syntax. For the for-loops, we do not need to store the generated list in a variable first. Instead, we can use the range() function in the for-statement directly.

|  |
| --- |
| for counter in range(start, end):  instructions |

The for-command must end with a colon, followed by the instructions that should be carried out in each iteration. These instructions must be written with indentation. The counter variable will do the counting for the iterations, starting from the start value in the first iteration and running through the entire integer list. Once the counter reaches end – 1, Python will execute the instructions for the last time and then exit the loop.

|  |
| --- |
| **Example (Cont’d):** We will carry out the previous task using the for-loop.      Figure 1.35 Entering Names and Scores Repeatedly by for-Loops  The range() function generates a list containing the values 0, 1, and 2, since 3 will not be included in the list by definition. The code in Figure 1.35 also shows that the counter variable i is already integrated in the for-statement, and an explicit instruction to increase it by one in each iteration is not required at all. |

**Read**

Read the following section of the textbook on while-loops:

Exercise 32 Loops and Lists

### 5.3 Breaking from Loops

Though we usually have a clear exit condition or a finite list to guarantee a loop to end at a certain point of the program, we may still be confronted with situations where we would like to interrupt the loop and continue with the subsequent program.

In our previous example, suppose we would like to quit the entire program after entering the first student’s data due to some reasons, although the program allows us to enter the data for up to three students. It would be reasonable to have a syntax that allows us to break from the loop in a “clean” manner instead of shutting down the computer entirely. The command for such situation is break.

|  |
| --- |
| for counter in range(start, end):  …  if conditions:  break |

Equivalently, the break command also works within a while-loop.

|  |
| --- |
| while conditions:  …  if conditions:  break |

Usually, break is used together with an if-condition since we would only want to break from a loop under some circumstances, and not in general.

|  |
| --- |
| **Example (Cont’d):** Suppose we let the user to quit the program by entering -1 for his/her score now. All we need to do is to add an if-condition after the input() statement where he/she can enter the score.      Figure 1.36 Breaking from Loops Using break  The break command is built after the user is asked to enter his/her exam score, but before the grade is being assigned. Basically, the program will still work normally if the break command is put after the whole if-elif-block. By breaking from the loop before a chunk of codes that will have no further influence on the execution of the program can shorten the running time and make it faster. |

Note that we can also apply the break command on our example in Figure 1.34 where we use the while-loops for the same task instead.

### 5.4 Error Handling in Input

Another common use of loops is to control the user input following an input() statement. For instance, it could happen that the user types in a letter instead of a number for the exam score by accident. In this case, we would like the user to redo the input until it is a number. As a result, we can put the input() statement within a while-loop and only break from it when the input of the user is valid.

Before we start to construct while-loops for user inputs, we have to learn how Python handles errors. In Chapter 3, we mention that if we apply the int() function to convert a string variable that contains a non-numeric value to an integer, the program will be interrupted due to value error. And the user will have to restart the program in PowerShell. This can be very annoying if the user only makes a small mistake in one of the input fields, but needs to re-type all the inputs because of the program interruption.

To avoid Python from stopping to execute the program by force, we can use the try: syntax.

|  |
| --- |
| try: #The try-block lets us test a block of code for errors.  instructions  except exception: #The except-block lets us handle the error.  instructions  else: #The else-block carries out instructions if no error occurs (optional).  instructions  finally: #The finally-block executes instructions regardless of the result of the try- and except blocks (optional).  instructions |

The try-except-block is an important instrument in Python to handle errors. Basically, we can put any syntax in the try-block if we think error can occur in those syntaxes. The except-block is to tell Python to continue with the program except for the occurrence of an error, or the occurrence of a specific error that we declare under exception. If error indeed occurs, Python will carry out the instructions written after the colon behind the except-statement, instead of stopping the program entirely. The else-block and finally-block are optional and can be used if we want certain instructions to be carried out if no error occurs or for finalising a try-block.

In Python, there are many built-in exceptions. Table 1.7 provides some common ones.

Table 1.7 List of Some Common Exceptions

|  |  |
| --- | --- |
| **Exception** | **Description** |
| NameError | Raised when a local or global name is not found. This applies only to unqualified names. The associated value is an error message that includes the name that could not be found. |
| TypeError | Raised when an operation or function is applied to an object of inappropriate type. The associated value is a string giving details about the type mismatch. |
| ValueError | Raised when an operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception. |

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

If the user is asked to enter a numeric value such as an exam score, but enter a string instead, we can use ValueError as our exception.

|  |
| --- |
| **Example (Cont’d):** We implement a try-block to test the validity of score input by the user. If it is a string, the program will warn the user of an erroneous input.      Figure 1.37 Using Exception for Error Handling  In the first run, we type in a string “test” where we should actually enter a numeric value for the score. Python detects an error since the syntax tries to use the int() function to convert character strings to integers which would usually cause the program to stop running abruptly. From the output, we can see that Python prints the warning message we put in the print() function to the screen and ends the program “properly” as if no error has occurred. In the second run, we enter a numeric value as required. Python detects no error and simply skips the except-block. |

After defining the try-block to instruct Python on how to handle errors, we can construct a while-loop around it. As condition for the loop to continue iterating is when a Boolean variable that indicates a valid input does not change from False to True. Hence, if this Boolean variable is True, the program will break from the loop.

|  |
| --- |
| **Example (Cont’d):** Now we put the entire try-block within a while-loop. The while-loop will stop iterating once the input for score is numeric. If it is non-numeric, the user will see a warning message and he will also be asked to re-enter his exam score. The whole procedure will last until the input is numeric.      Figure 1.38 Using while-loop for User Input If Error Occurs  We initiate a Boolean variable called valid\_input before the while-loop starts. The initial value of this variable is False which we also use as the condition for the while-loop to continue to iterate. In the try-block, we add an else-statement for the case that the input is correct, and the follow-up instruction here is to change the value of valid\_input to True so that the while-loop stops. Note that we can use the break command here as well. In the output screen, we can see that if the input is non-numeric, the program will print a warning message in a new line and then ask the user to re-enter the score until the input is valid. |

It is possible or even desirable to set a maximum number of input trials in order not to have the program running endlessly. A counter variable can be added to the loop, and the program will exit the loop after the maximum allowed number of iterations has been reached.

|  |
| --- |
| **Example (Cont’d):** The complete program of this study unit containing all the techniques we have learned is given in the following figure.      Figure 1.39 The Complete Example Program of Study Unit 1 |

## Summary

We have learned the basics of writing and executing Python programs. We have also been introduced to the various variable types and some operators that can be applied to them. We have then discussed how to generate screen output and how to let the user enter answers and assign them as values to the variables. Furthermore, we have covered the construction of conditional statements to dynamically change the program behaviour if necessary. Finally, we have also come to know the use of loops to repeat routine tasks for an endless number of times within a program.

## References

Atom.io (n.d.). *A hackable text editor for the 21st Century*. GitHub.   
<https://atom.io>

Learn Python (n.d.). *Python – Basic Operators*. tutorialspoint. <https://www.tutorialspoint.com/python/python_basic_operators.htm>

Python Tutorial (n.d.). *Python Variables*. w3schools.com. <https://www.w3schools.com/python/python_variables.asp>

Python Tutorial (n.d.). *Python 3 Escape Sequences*. Python-ds.com. <https://www.python-ds.com/python-3-escape-sequences>

Python.org (n.d.). *Download Python*. Python Software Foundation. <https://www.python.org/downloads/>

Python.org (n.d.). *Built-in Types*. Python Software Foundation. <https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex>

Python.org (n.d.). *Built-in Exceptions*. Python Software Foundation. [https://docs.python.org/3/library/exceptions.html](https://docs.python.org/3/library/exceptions.html# numeric-types-int-float-complex)

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

## Formative Assessment

1. From which directory (folder) can you run your Python script that is saved with the .py extension?
2. From the folder where Python is installed
3. From the folder where the operating system is installed
4. From the folder where I have saved my Python script
5. From any arbitrary folder
6. Which line of code is not a valid Python syntax?
   1. print("This is {a} wrong syntax!")
   2. 0 = 0
   3. 0 == 1
   4. y = a / int(b)
7. Which of the following is a valid Python variable name?
   1. iamavariablelol
   2. :)iamavariable
   3. 007imavariable
   4. i-am-a-variable-lol
8. What is the value of int(-0.5) in Python?
   1. -1
   2. 0.5
   3. 0
   4. 1
9. What does the .format() method do?
   1. It formats the font of the string in a print() function.
   2. It replaces the expression in a curly bracket within a string by its value.
   3. It replaces the curly bracket by a round bracket in a string.
   4. It replaces the curly bracket within a string by the value of the expression in the .format() method.
10. Which of the following Boolean expression is false?
    1. -5 <= 0
    2. 10 \*\* 2 == 100
    3. int(-0.5) < int(-0.2)
    4. 18 / 3 > 1 + 4
11. Which Boolean expression is equivalent to not(a and b)?
    1. not a and not b
    2. not a or not b
    3. not a and b
    4. a or b
12. Which statement is correct regarding the if-elif-else statement block?
    1. The if-elif-else statement must end with an end-statement.
    2. There must be an else-statement in every if-elif-else statement block.
    3. If-elif-else statement block can also start with an elif- or else-statement.
    4. Behind each if-, elif-, and else-statement must be a colon before the instruction block starts.
13. Which values will be printed on the screen given the following Python code?

counter = 0

while counter <= 3:

print(counter)

counter = counter + 1

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

1. Which of the following is correct about for-loops in Python?
   1. We need to initiate a counter before the loop starts.
   2. We must write a line to increase the counter by one within the loop.
   3. We need an exit condition for the loop.
   4. We can use the range() function to generate a list of number as the index of the for-loop iterations.

## Suggested Solutions

1. From which directory (folder) can you run your Python script that is saved with the .py extension?
2. From the folder where Python is installed

*Incorrect. Python would not be able to find your Python script there unless you have saved it in the Python program folder, which is rather unlikely.*

1. From the folder where the operating system is installed

*Incorrect. Python would not be able to find your Python script there unless you have saved it in the system folder, which is also rather unlikely.*

1. **From the folder where I have saved my Python script**

**Correct. You must change to the folder where you have saved your Python script so that Python can find your file.**

1. From any arbitrary folder

*Incorrect. Python would not be able to find your Python script unless you are accidentally in the folder where you have saved your Python script.*

1. Which line of code is not a valid Python syntax?
   1. print("This is {a} wrong syntax!")

*Incorrect. The syntax is correct since the curly bracket and its content will be treated as part of the printing string.*

* 1. **0 = 0**

**Correct. The left-hand side of a value-assignment syntax must be a variable name. A number there is invalid since Python interprets it as assigning a value to a number.**

* 1. 0 == 1

*Incorrect. This syntax is valid since it is a Boolean expression.*

* 1. y = a / int(b)

*Incorrect. This is a valid syntax since we can carry out a division in which the denominator is being converted to an integer.*

1. Which of the following is a valid Python variable name?
   1. **iamavariablelol**

**Correct. This variable name is valid since it starts with a character and contains no invalid character.**

* 1. :)iamavariable

*Incorrect. This variable name is invalid since it starts with a colon and contains invalid characters such as closing round bracket.*

* 1. 007imavariable

*Incorrect. This variable name is invalid since it starts with a number.*

* 1. i-am-a-variable-lol

*Incorrect. This variable name contains invalid characters such as hyphen.*

1. What is the value of int(-0.5) in Python?
   1. -1

*Incorrect. The int() function does not round down a negative number.*

* 1. 0.5

*Incorrect. The int() function is not used to convert a value to absolute number.*

* 1. **0**

**Correct. The int() function will take away all the decimal places.**

* 1. 1

*Incorrect. The int() function does not round down a negative number and then convert it to absolute number.*

1. What does the .format() method do?
   1. It formats the font of the string in a print() function.

*Incorrect. The .format() method does not format the font of a string.*

* 1. It replaces the expression in a curly bracket within a string by its value.

*Incorrect. This would be done by the f-instruction in the print() function for string formatting.*

* 1. It replaces the curly bracket by a round bracket in a string.

*Incorrect. The .format() method does not replace the curly bracket by a round bracket in a string unless the code explicitly requires Python to do so.*

* 1. **It replaces the curly bracket within a string by the value of the expression in the .format() method.**

**Correct. The value of the expression in the .format() method will be used to replace the curly bracket within the printing string.**

1. Which of the following Boolean expression is false?
   1. -5 <= 0

*Incorrect. The expression is true since it requires -5 to be smaller or equal to 0.*

* 1. 10 \*\* 2 == 100

*Incorrect. Since 10 \*\* 2 = 100, the left-hand side and the right-hand side are equal.*

* 1. **int(-0.5) < int(-0.2)**

**Correct. Since int(-0.5) is 0 and int(-0.2) is also 0, 0 < 0 is a false relation.**

* 1. 18 / 3 > 1 + 4

*Incorrect. Since 18 / 3 = 6 and 1 + 4 = 5, 6 > 5 is a true relation.*

1. Which Boolean expression is equivalent to not(a and b)?
   1. not a and not b

*Incorrect. not a and not b is equivalent to not(a or b)*

* 1. **not a or not b**

**Correct. not a or not b is equivalent to not(a and b)**

* 1. not a and b

*Incorrect. not a and b is equivalent to not(a or not b)*

* 1. a or b

*Incorrect. a or b is equivalent to not(not a and not b)*

1. Which statement is correct regarding the if-elif-else statement block?
   1. The if-elif-else statement must end with an end-statement.

*Incorrect. No end-statement is needed for an if-elif-else statement block.*

* 1. There must be an else-statement in every if-elif-else statement block.

*Incorrect. An if-elif-else statement block does not necessarily require an else-statement.*

* 1. If-elif-else statement block can also start with an elif- or else-statement.

*Incorrect. An if-elif-else statement block must start with an if-statement.*

* 1. **Behind each if-, elif-, and else-statement must be a colon before the instruction block starts.**

**Correct. Behind every if-, elif-, and else-statement must be a colon.**

1. Which values will be printed on the screen given the following Python code?

counter = 0

while counter <= 3:

print(counter)

counter = counter + 1

* 1. **0, 1, 2, 3**

**Correct. Since the print() function comes before the increment of the counter, the counter will be printed starting from its initial value 0 and goes until 3 with an increase of 1 in each iteration.**

* 1. 1, 2, 3, 4

*Incorrect. Since the print() function comes before the increment of the counter, the counter will be printed starting from its initial value 0 and not 1.*

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

*Incorrect. Since the loop will only continue to run if the value in counter is smaller or equal to 3, 4 cannot be printed based on this code.*

* 1. 0, 2, 3

*Incorrect. Since the increment of the counter can only be 1 for each iteration of the loop, a jump from 0 to 2 is impossible based on this code.*

1. Which of the following is correct about for-loops in Python?
   1. We need to initiate a counter before the loop starts.

*Incorrect. We only need to initiate a counter before a while-loop starts.*

* 1. We must write a line to increase the counter by one within the loop.

*Incorrect. We only need to write a line to increase the counter by one within a while-loop.*

* 1. We need an exit condition for the loop.

*Incorrect. A for-loop does not need any exit condition since it runs through a finite list. Once the list comes to an end, Python will exit the loop.*

* 1. **We can use the range() function to generate a list of numbers as the index of the for-loop iterations.**

**Correct. Every for-loop needs a list for it to run through. One type of list is a list of integers that can serve as the index for the for-loop iterations, and we can generate such a list by the range() function.**