2020
EDITION

Textbook for CBSE Class XII

Computer Science

With

Python

PREETI ARORA

- Advanced Python Concepts
- Modules
- Recursion
- Data Structures
- Pyplot
- Computer Networks
- MySQL
- Society, Law & Ethics
- Project Work

Web Support
at
sultan-chand.com/ws/python12
• Presentations • Projects & Program Codes
• Sample Papers • Model Test Papers
• Practical File • Viva Voce

SCS
sultan chand

2020
EDITION

Textbook for CBSE Class XII

Computer Science

With

Python

PREETI ARORA
DOEACC 'A' level, M.Sc–IT, M.Tech–IT
Sr. Computer Science Teacher

SCS
sultan chand
Educational Publishers

SULTAN CHAND & SONS (P) LTD
Educational Publishers
4859/24, Darya Ganj, New Delhi-110 002
Phones : 4354 6000 (100 Lines), 2324 3939
Fax : (011) 4354 6004, 2325 4295
E-mail : scs@sultanchandebooks.com
Buy books online at : www.sultan-chand.com

ISBN: 978-93-89174-43-4

First Edition 2019
Second Thoroughly Revised Edition 2020

©All rights reserved.

No part of this book may be reproduced or copied in any form or by any means (graphic, electronic or mechanical, including photocopying, recording, taping, or information retrieval system) or reproduced on any disc, tape, perforated media or any other information storage device, etc., without the prior written permission of the publishers. Breach of this condition is liable for legal action. **Anyone who brings information regarding any such reproduction will be handsomely rewarded.**

Publication of **Key** to this book is strictly prohibited.

Every effort has been made to avoid errors or omissions in this publication. In spite of this, some errors might have crept in. Any mistake, error or discrepancy noted may be brought to our notice which shall be taken care of in the next edition. It is notified that neither the publishers nor the author or seller will be responsible for any damage or loss of action to anyone, of any kind, in any manner, therefrom.

For faulty binding, misprints or for missing pages, etc., the publishers' liability is limited to replacement within one month of the purchase by a similar edition. All expenses in this connection are to be borne by the purchaser.

All disputes are subject to Delhi jurisdiction only.

PREFACE

Programming is very important for learning to innovate and create eco-friendly solutions to global problems. Programming is also important in our day-to-day life to enhance the power of computers and internet. An important step towards learning innovative programming solutions is through Python programming which this book has at its core.

The thoroughly revised **Computer Science with Python** for Class XII provides an in-depth understanding of the Computer Science (083) curriculum and strictly adheres to the guidelines laid down by the CBSE. The book deals with advanced concepts of Python programming language, Relational Database Management System (SQL), Python-MySQL Connectivity, Computer Networks, Data Visualizations and Cyber Ethics.

Python is a popular object-oriented language used both for stand-alone programs and scripting applications in a variety of domains. This book presents a contemporary approach to programming with stress on principles of good programming, such as clarity, legibility and efficiency in program design. Thus, an interactive programming style has been emphasized/expressed throughout the book.

The key concept of this book is that it teaches advanced Python programming concepts and usage of several Python libraries, such as plotting graphs and charts using Python Pyplot, establishing Python-MySQL connectivity, implementing Python data structures (Stacks and Queues) with a student-centred approach. With easy-to-understand examples, practical implementations and other tools, the student will learn how to design the logic of programs and then implement those programs using Python.

The text of the book has been presented in a friendly and easy-to-comprehend language. The book contains example programs that are concise and practical besides diagrams and examples from real-life applications. Each chapter provides tested, debugged and error-free codes with screenshots. Based on the CBSE curriculum, the book has been divided into four units:

Unit I: Programming and Computational Thinking (PCT-2) – Chapters 1-7

This unit contains seven chapters covering review of Python studied in Class XI, the concept of functions and modules in Python, passing arrays (lists) to functions, Recursion, Mathematical and String built-in libraries/modules besides creating and using Python libraries, Text and Binary file-handling operations involving read, write, append, read-write operations, standard input-output and error streams, relative and absolute paths for a file.

Apart from the above, the idea of efficiency of a Python program and time taken by a program to execute has been discussed in depth through appropriate programming examples. Implementing Python Data Structures, *viz.* Stacks and Queues using Lists, Data Visualization using Pyplot in terms of Line chart, Bar chart and Pie chart have also been thoroughly discussed.

Unit II: Computer Networks (CN) – Chapter 8

This unit covers the concepts of computer networking, network devices, network stack, TCP and idea of retransmission, network congestion, basic network tools, application layer concepts, secure communication, several important protocols that govern the working of internet and networks and detailed summary of new network technologies such as cloud computing and IoT (Internet of Things).

Unit III: Data Management (DM-2) – Chapters: 9, 10 & 12

This unit covers development of Django web application, Interface Python with an SQL database and SQL commands, aggregation functions along with important SQL clauses such as group by, having and order by.

Unit IV: Society, Law and Ethics (SLE-2) – Chapter 11

This unit deals with intellectual property rights, plagiarism, digital rights management, privacy laws, frauds and cybercrimes such as phishing, illegal downloading, child pornography, cyber forensics and IT Act, 2000. It also explains related concepts of technology and society, e-waste management, and gender and disability issues while teaching and using computers.

The book has five appendices containing a project on Library Management System, installation guide to Django, Viva Voce questions, Sample Question Paper (Solved) and a Model Test Paper (Unsolved).

As part of our Web Support, Presentations on Python, Chapter-wise Program Codes, Projects based on Python-MySQL Connectivity, Practical File, Sample Papers, Model Test Papers for practice and Viva Voce questions are available online and can be accessed at sultan-chand.com/ws/python12. Besides, subject-related updates, if any, will also be made available online in due course.

I am confident that students and teachers will benefit immensely by making the best use of this book.

Your feedback is important to me. Any suggestions for the improvement of this book will be highly appreciated and duly acknowledged.

My special thanks are due to Ms. Rinku Kumari and Ms. Harmeet Kaur for their valuable feedback during the course of my writing this book.

Last but not the least, I express my deep gratitude to my esteemed publishers, Sultan Chand & Sons (P) Ltd, for their patience, guidance and support.

AUTHOR

CONTENTS

1. Review of Python Basics 1.1–1.64

1.1 Introduction . . . 1.1
1.2 Structure of a Python Program . . . 1.3
1.3 Variables and Data Types . . . 1.4
1.3.1 Dynamic Typing . . . 1.6
1.4 Keywords . . . 1.6
1.5 Mutable and Immutable Types . . . 1.7
1.6 Operators and Operands . . . 1.8
1.7 Input and Output (Python's Built-in Functions) . . . 1.11
1.7.1 Type Casting (Explicit Conversion) . . . 1.12
1.8 Comments in Python . . . 1.14
1.9 Flow of Execution . . . 1.15
1.9.1 Sequential Statements . . . 1.15
1.9.2 Selection/Conditional Statements . . . 1.15
1.9.3 Looping Constructs . . . 1.20
1.10 Strings . . . 1.24
1.10.1 String Operations . . . 1.25
1.10.2 String Slicing . . . 1.25
1.10.3 Built-in String Methods . . . 1.26
1.11 Lists . . . 1.30
1.11.1 List Comprehension . . . 1.31
1.11.2 List Slicing . . . 1.32
1.11.3 Built-in List Functions and Methods . . . 1.33
1.11.4 Copying Lists . . . 1.34
1.12 Tuples . . . 1.35
1.12.1 Iterating Through a Tuple . . . 1.36
1.12.2 Tuple Slicing . . . 1.36
1.13 Dictionary . . . 1.38
1.13.1 Iterating Through a Dictionary . . . 1.39
1.13.2 Updating Dictionary Elements . . . 1.39
1.13.3 Built-in Dictionary Functions . . . 1.40
1.14 Sorting Techniques . . . 1.41
1.14.1 Bubble Sort . . . 1.41
1.14.2 Insertion Sort . . . 1.44

2. Functions 2.1–2.60

2.1 Introduction . . . 2.1
2.2 Functions . . . 2.2
2.2.1 Built-in Functions . . . 2.3
2.2.2 Modules . . . 2.11
2.2.3 User-defined Functions . . . 2.20
2.3 How Function Returns A Value . . . 2.25
2.4 Parameters and Arguments in Functions . . . 2.29
2.4.1 Types of Arguments . . . 2.31
2.5 Passing Arrays/Lists to Functions . . . 2.35
2.6 Scope of Variables . . . 2.36

2.7 Using main() as a Function . . . 2.38
2.8 Flow of Execution of Program Containing Function Call . . . 2.40
2.9 Recursion . . . 2.40
2.9.1 Recursion Vs Iteration . . . 2.44

3. Using Python Libraries 3.1–3.30

3.1 Introduction . . . 3.1
3.2 Module in Python . . . 3.2
3.3 Importing Python Modules . . . 3.3
3.4 Namespaces in Python . . . 3.7
3.5 Name Resolution (Resolving Scope of a Name) . . . 3.8
3.6 Module Aliasing . . . 3.9
3.7 Member Aliasing . . . 3.10
3.8 Package/Library . . . 3.13
3.9 Locating Modules . . . 3.17
3.10 Standard Python Libraries . . . 3.17
3.10.1 Datetime Library/Module . . . 3.17

4. Data File Handling 4.1–4.38

4.1 Introduction . . . 4.1
4.2 Why Use Files . . . 4.2
4.3 Data File Operations . . . 4.2
4.4 Opening and Closing Files . . . 4.4
4.4.1 open()—Opening a File . . . 4.4
4.4.2 close()—Closing a File . . . 4.7
4.5 Reading from a File . . . 4.8
4.6 Writing to File . . . 4.11
4.7 Appending to File . . . 4.15
4.8 Binary File Operations . . . 4.18
4.9 Relative and Absolute Paths . . . 4.21
4.10 Standard File Streams . . . 4.23

5. Program Efficiency 5.1–5.12

5.1 Introduction . . . 5.1
5.2 What is Algorithm Analysis . . . 5.1
5.3 Program Efficiency in Terms of Time . . . 5.4
5.4 Efficiency on the Basis of Number of Operations . . . 5.6

6. Data Structures in Python 6.1–6.28

6.1 Introduction . . . 6.1
6.2 Stack . . . 6.1
6.3 Implementation of Stack Using List . . . 6.3
6.3.1 Creating a Stack . . . 6.3
6.3.2 Adding Elements into a Stack . . . 6.4
6.3.3 Checking for Empty Stack . . . 6.4
6.3.4 Deleting an Element from a Stack . . . 6.5
6.3.5 Traversing Stack Elements . . . 6.5
6.4 Queue . . . 6.10
6.5 Implementation of Queue as List . . . 6.11
6.5.1 Creating a Queue using List . . . 6.11
6.5.2 Inserting an Element into a Queue . . . 6.12
6.5.3 Checking for an Empty Queue . . . 6.13
6.5.4 Deleting Elements from a Queue . . . 6.14
6.5.5 Traversal of Queue Elements . . . 6.16

7. Data Visualization Using Pyplot 7.1–7.34

7.1 Introduction . . . 7.1
7.2 Matplotlib . . . 7.2
7.3 NumPy . . . 7.3
7.4 Installing Matplotlib . . . 7.3
7.5 Types of Visualizations . . . 7.4
7.6 Basic Visualization Rules . . . 7.5
7.7 Line Plot/Chart . . . 7.5
7.7.1 Multiple Plots . . . 7.8
7.7.2 Multiple Views . . . 7.9
7.8 Bar Plot/Chart . . . 7.17
7.9 Pie Plot/Chart . . . 7.20

8. Computer Networks 8.1–8.74

8.1 Introduction . . . 8.1
8.2 Computer Network—A Brief Overview . . . 8.2
8.2.1 Advantages of Computer Networks . . . 8.2
8.3 Structure of a Network . . . 8.4
8.4 Types of Networks . . . 8.4
8.4.1 Personal Area Network (PAN) . . . 8.5
8.4.2 Local Area Network (LAN) . . . 8.5
8.4.3 Metropolitan Area Network (MAN) . . . 8.6
8.4.4 Wide Area Network (WAN) . . . 8.6
8.5 Wired and Wireless Networks . . . 8.7
8.6 Client-Server Architecture . . . 8.14
8.7 New Technologies . . . 8.15
8.7.1 Cloud Computing . . . 8.15
8.7.2 Internet of Things (IoT) . . . 8.17
8.8 Network Devices . . . 8.18
8.8.1 Network Interface Card (NIC) . . . 8.18
8.8.2 Hub . . . 8.19
8.8.3 Switch . . . 8.19
8.8.4 Repeater . . . 8.20
8.8.5 Gateway . . . 8.20
8.8.6 Router . . . 8.21
8.8.7 Wireless Access Point (WAP) . . . 8.21
8.8.8 Setting up a Computer Network—An example . . . 8.21
8.9 Network Stack . . . 8.23
8.10 Modulation . . . 8.23
8.10.1 Amplitude Modulation . . . 8.24
8.10.2 Frequency Modulation . . . 8.25
8.11 Collision in Wireless Network . . . 8.25
8.12 Error Checking . . . 8.26
8.13 MAC . . . 8.28
8.14 Routing . . . 8.29
8.15 IP Addresses (v4 and v6) . . . 8.30
8.15.1 Internet Protocol Version 4 (IPv4) . . . 8.31
8.15.2 Internet Protocol Version 6 (IPv6) . . . 8.31
8.16 Domain Name System (DNS) . . . 8.31

8.17 TCP: Basic Idea of Retransmission and Rate Modulation When there is Congestion (Analogy to a Road Network) . . . 8.34
8.18 Retransmission in TCP . . . 8.35
8.19 Protocols: 2G, 3G, 4G, Wi-Fi . . . 8.35
8.19.1 What makes a protocol have a higher bandwidth . . . 8.38
8.20 Basic Network Tools . . . 8.38
8.20.1 traceroute . . . 8.38
8.20.2 ping . . . 8.39
8.20.3 ipconfig . . . 8.39
8.20.4 nslookup . . . 8.40
8.20.5 Whois . . . 8.40
8.20.6 netstat . . . 8.41
8.20.7 Speed Test . . . 8.41
8.21 Application Layer . . . 8.42
8.21.1 HTTP . . . 8.43
8.21.2 Working of Electronic mail (Email) . . . 8.44
8.22 Secure Communication . . . 8.47
8.23 Network Applications . . . 8.48
8.23.1 Remote Desktop . . . 8.48
8.23.2 Remote Login . . . 8.48
8.23.3 FTP . . . 8.49
8.23.4 SMTP . . . 8.50
8.23.5 VoIP . . . 8.51
8.23.6 POP/IMAP . . . 8.52
8.23.7 Session Control Protocol (SCP) . . . 8.53
8.23.8 SSH Protocol . . . 8.53
8.23.9 NFC . . . 8.53

9. Interface Python with SQL **9.1–9.26**

9.1 Introduction . . . 9.1
9.2 Python-MySQL Connectivity . . . 9.2
9.3 Why Python . . . 9.2
9.4 Installing MySQL-Connector . . . 9.3
9.4.1 MySQLdb . . . 9.4
9.5 Establishing Connection . . . 9.6
9.6 Creating Cursor Object . . . 9.7
9.7 Creating a Database . . . 9.8
9.8 Closing Cursor and Connection . . . 9.20
9.9 Operations on a Table in a Nutshell . . . 9.21

10. More on SQL **10.1–10.28**

10.1 Introduction . . . 10.1
10.2 Functions in MySQL . . . 10.1
10.3 Aggregate Functions in SQL . . . 10.3
10.4 Sorting in SQL—Order By . . . 10.9
10.5 Group By . . . 10.10
10.5.1 HAVING Clause . . . 10.10
10.6 Aggregate Functions & Conditions on Groups (HAVING Clause) . . . 10.12

11. Society, Law and Ethics **11.1–11.26**

11.1 Introduction . . . 11.1
11.2 Intellectual Property Rights . . . 11.2

11.3 Plagiarism . . . 11.2
11.4 Digital Rights Management . . . 11.3
11.5 Licensing . . . 11.5
11.6 Open Source and Open Data . . . 11.8
11.7 Privacy Laws . . . 11.9
11.8 Cybercrime . . . 11.10
11.8.1 Phishing . . . 11.11
11.8.2 Illegal Downloading . . . 11.11
11.8.3 Child Pornography . . . 11.12
11.8.4 Cyber Scams and Frauds . . . 11.12
11.8.5 Cyber Forensics . . . 11.13
11.9 Information Technology Act, 2000 . . . 11.14
11.10 Unique IDs and Biometrics . . . 11.15
11.11 Impact of Technology Change on Society . . . 11.15
11.12 E-Waste Management . . . 11.16
11.13 Gender and Disability Issues while Teaching and Using Computers . . . 11.18

12. Web Development with Django **12.1–12.34**

12.1 Introduction . . . 12.1
12.2 What is a Framework . . . 12.1
12.3 What is Django . . . 12.2
12.4 Django Web Framework . . . 12.3
12.5 How Django Works . . . 12.4
12.6 Django Installation . . . 12.5
12.7 Web Server . . . 12.5
12.8 Creating Projects . . . 12.6
12.9 Creating Apps of a Django Project . . . 12.15
12.10 GET and POST Methods . . . 12.17
12.10.1 Difference between GET and POST Methods . . . 12.17
12.10.2 Minimal Django-based Web Application that Parses a GET . . . 12.20
12.10.3 Minimal Django-based Web Application that Parses a POST . . . 12.23
12.11 Working with Flat Files and CSV Files . . . 12.25
12.11.1 Write the Fields to a Flat File . . . 12.25
12.11.2 Write the Fields to a CSV File . . . 12.26
12.11.3 Read the Fields from a CSV File . . . 12.27

APPENDICES **A.1–A.28**

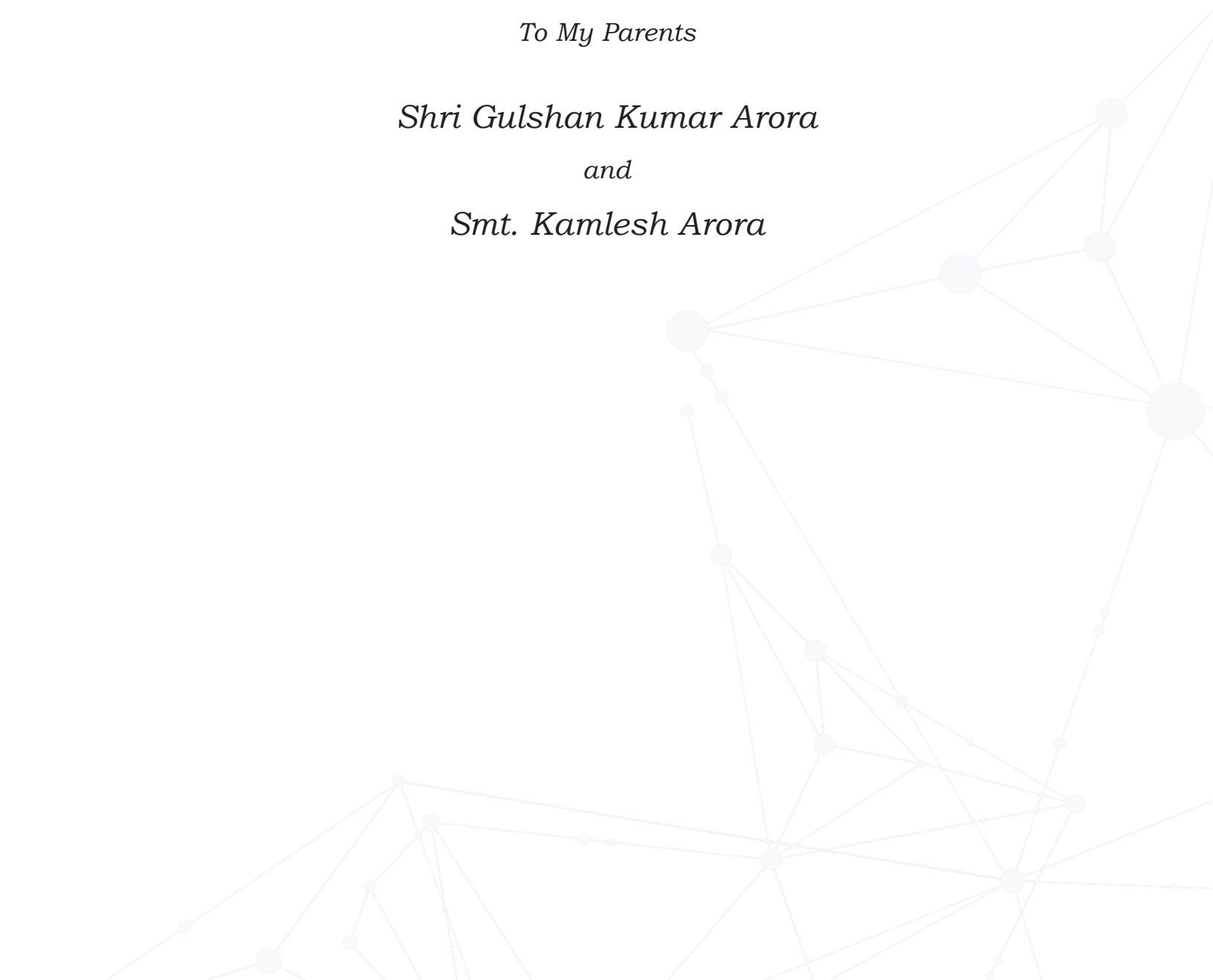
Appendix A: Project Work (Project on Student Registration System) . . . A.1
Appendix B: DJANGO Installation . . . A.6
Appendix C: Viva Voce . . . A.11
Appendix D: Sample Question Paper (Solved) . . . A.16
Appendix E: Model Test Paper (Unsolved) . . . A.24

To My Parents

Shri Gulshan Kumar Arora

and

Smt. Kamlesh Arora



1 Review of Python Basics

1.1 INTRODUCTION

You are aware of the fact that Python is a powerful, modern, high-level programming language. You have learnt the basics of Python programming in Class XI. In this chapter, we will briefly recapitulate the Python concepts that you have already learnt in the previous class.

Python is an interpreted language as its programs are executed by an interpreter. Thus, Python interpreter should be installed on the computer system to write and run Python programs. We have also learnt that Python IDLE (Integrated Development and Learning Environment) provides two working modes—interactive mode (popularly known as Python shell) and script mode.

CTM: The Python IDLE tool offers an interactive and a more efficient platform to write your code in Python.

Using **Interactive (Python Shell window) mode**, the Python commands are directly typed at >>> (command prompt), and as soon as we press the Enter key, the interpreter displays the result(s) immediately, which is known as **Displaying**.

For example,

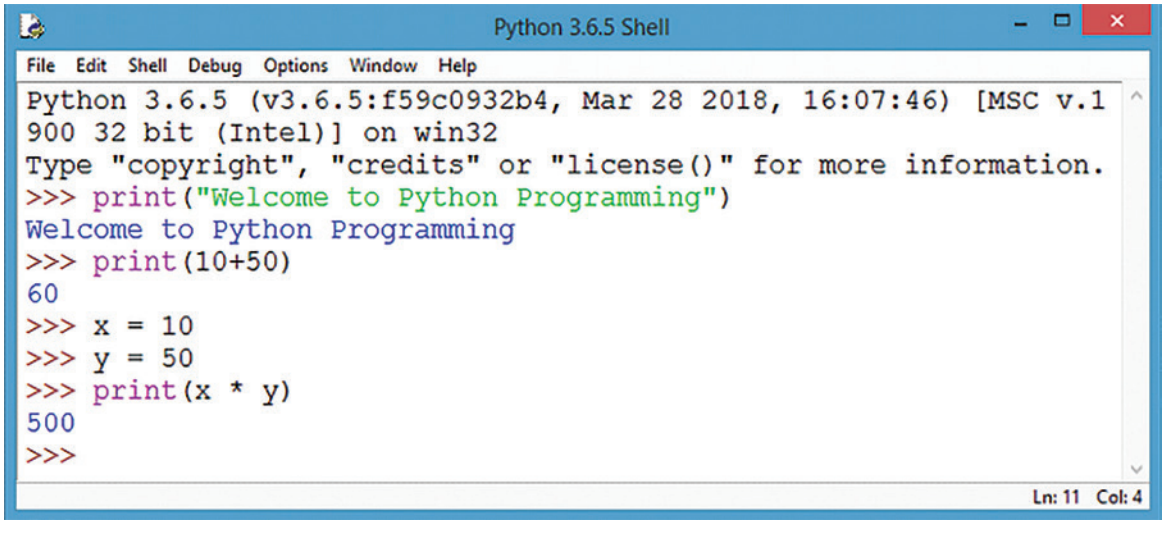
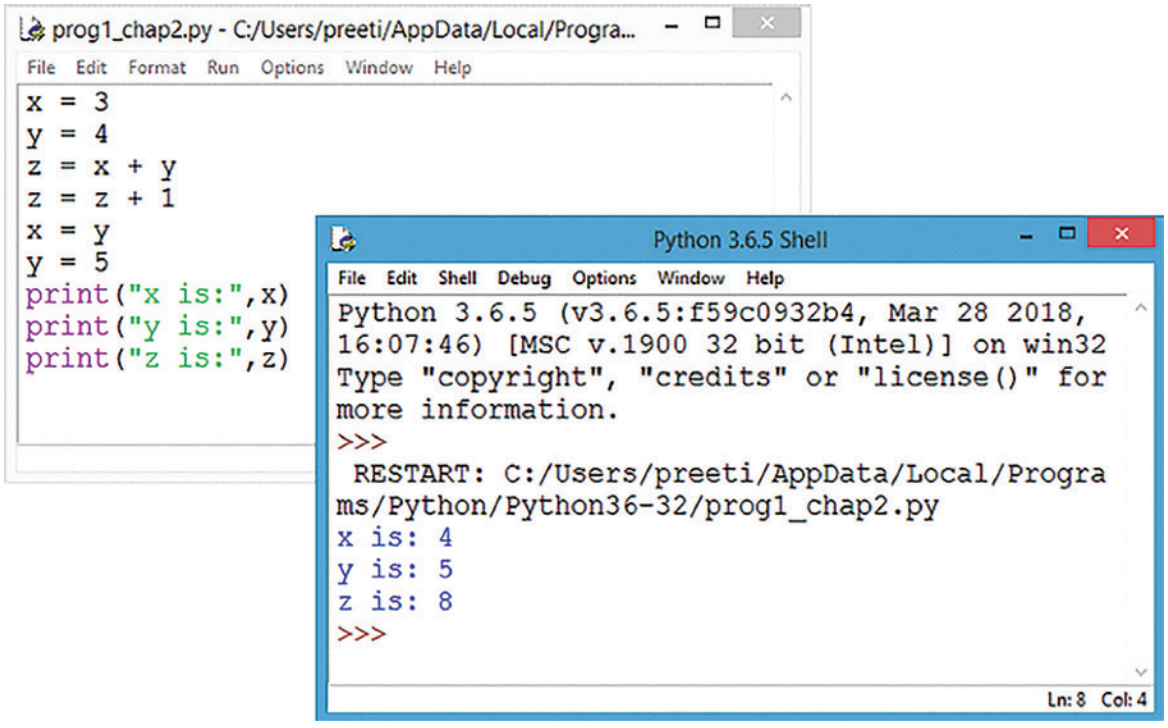


Fig. 1.1: Working with Python Shell

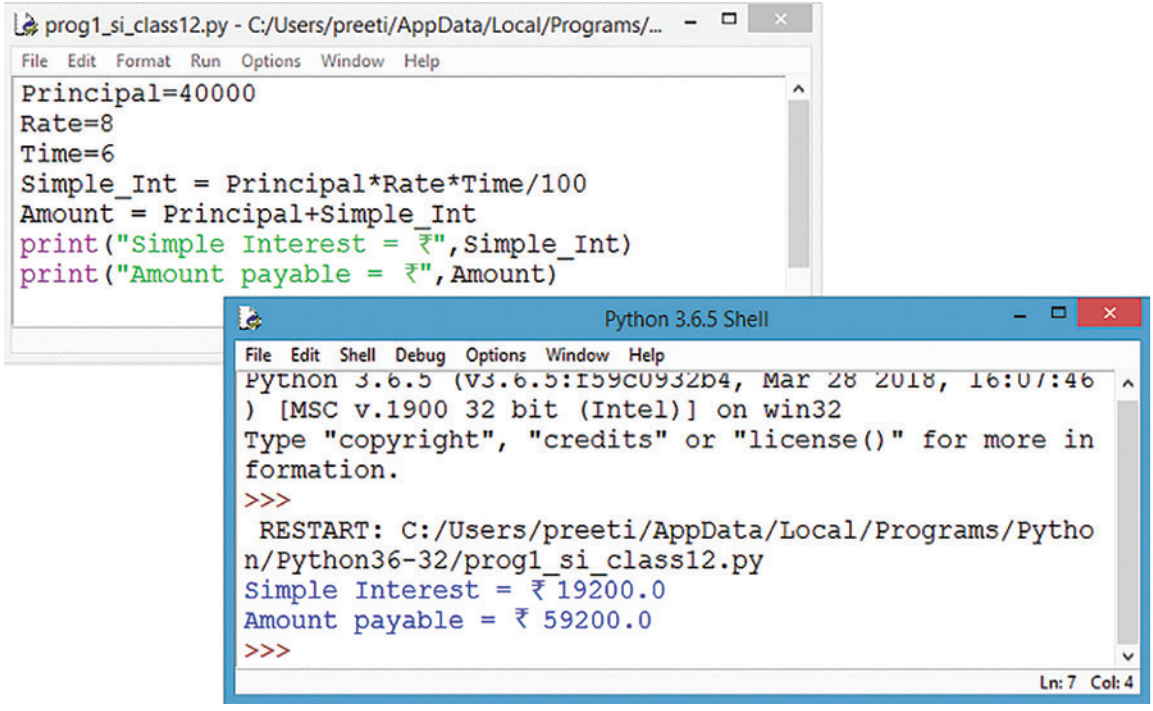
Script Mode

The drawback of interactive mode is that we cannot save the commands that we type. This can be overcome using the script mode. In order to switch over to script mode, click on the **File** menu option -> **New** option or press the shortcut key **Ctrl + N** from the shell window.

Example 1:

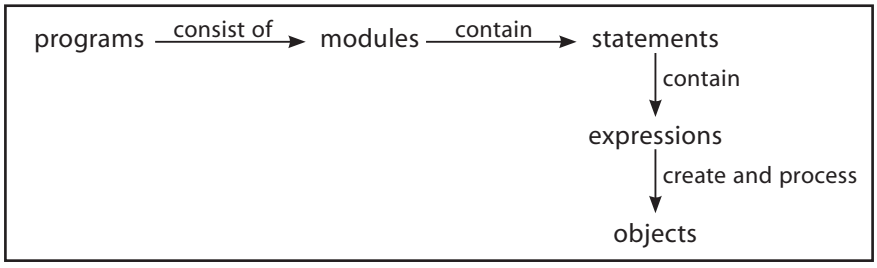


Example 2: Lokesh has taken a loan of ₹ 40000 from Vinod at the rate of 8% per annum. After 6 years, he wants to repay the loan in full including interest. Write the Python code (in script mode) to calculate and display the interest and total amount to be paid by Lokesh to settle his accounts.

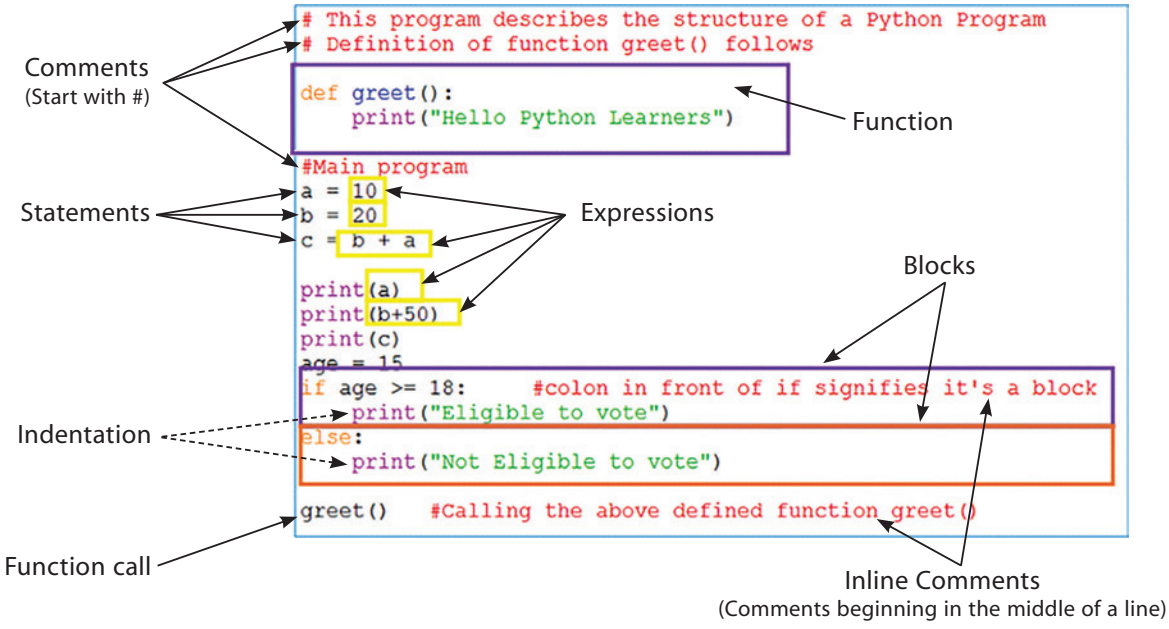


1.2 STRUCTURE OF A PYTHON PROGRAM

A Python program constitutes several elements such as statements, expressions, functions, comments, etc., which are synchronized in the manner as shown below:



Let us see the several components of a basic Python program.



As shown in the snippet given above, the several components that a Python program holds are:

- **Expressions:** An expression **represents** something, like a number, a string, or an element. Any value is an expression.
- **Statements:** Anything that **does something** is a statement. Any assignment to a variable or function call is a statement. Any value contained in that statement is an expression.
- **Comments:** Comments are the additional information provided against a statement or a chunk of code for the better clarity of the code. Interpreter ignores the comments and does not count them in commands.

  Symbols used for writing comments include Hash (#) or Triple Double Quotation marks (“””). **Hash (#)** is used in writing **single-line comments** that do not span multiple lines. **Triple Quotation Marks (‘’’ or “””)** are used to write **multiple-line comments**. Three triple quotation marks to start the comment and again three quotation marks to end the comment.
- **Functions:** Function is a set of instructions defined under a particular name, which once written can be called whenever and wherever required.
- **Block(s):** A block refers to a group of statements which are part of another statement or function. All statements inside a block are indented at the same level.

1.3 VARIABLES AND DATA TYPES

A variable is like a container that stores values you can access or change. The purpose of using variables is to allow the stored values to be used later on. We have learnt that any object or variable in Python is a name that refers to a value at a particular memory location and possesses three components:

- **A Value:** It represents any number or a letter or a string. To assign any value to a variable, we use assignment operator (=).
- **An Identity:** It refers to the address of the variable in memory which does not change once created. To retrieve the address (identity) of a variable, the command used is:

```
>>>id(variable_name)
```

- **A Type:** We are not required to explicitly declare a variable with its type. Whenever we declare a variable with some value, Python automatically allocates the relevant data type associated with it.

  Hence, the data type of a variable is according to the value it holds.

  For example, `>>> x = 20`

  The above statement signifies 'x' to be of integer type since it has been assigned an integer value 20.

  Data types are classified as follows (Fig.1.2):

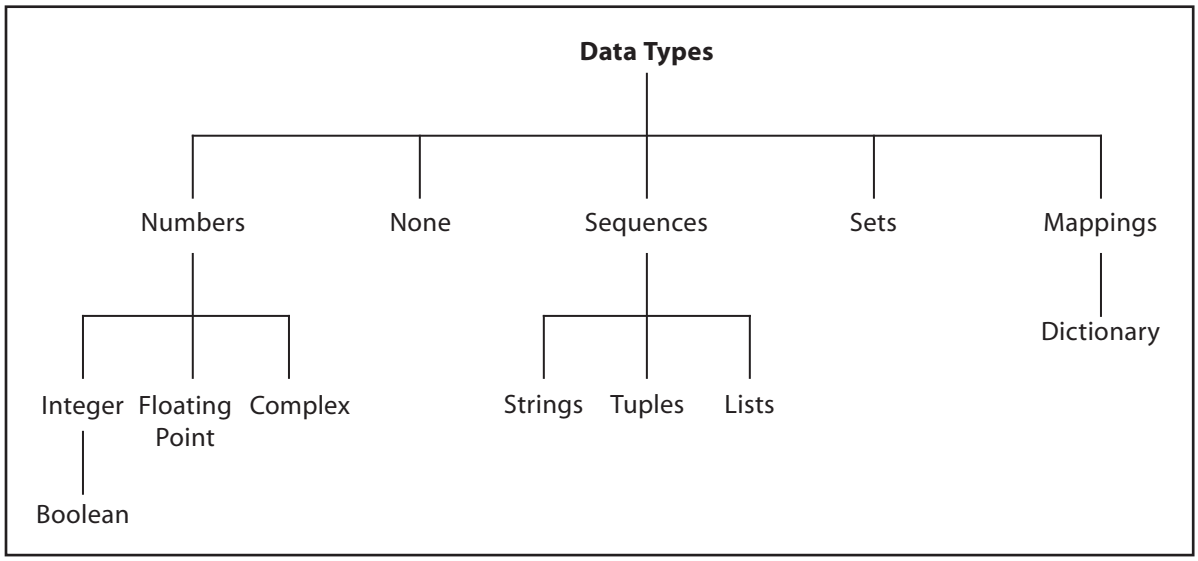
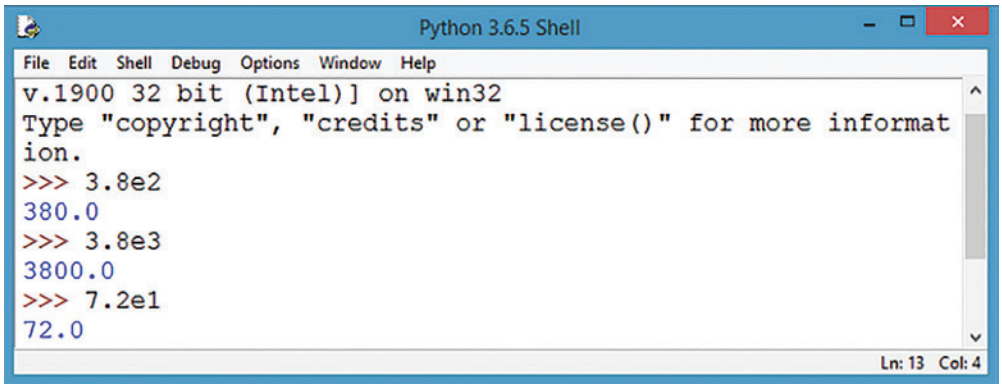


Fig. 1.2: Classification of Data Types in Python

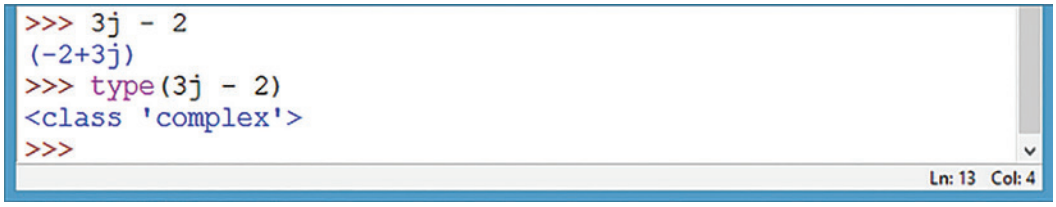
1. **Number or Numeric Data Type:** Numeric data type is used to store numeric values. It is further classified into three subtypes:

   (a) **Integer and Long:** To store whole numbers, *i.e.*, decimal digits without a fraction part. They can be positive or negative. **Examples:** 566, –783, –3, 44, etc.

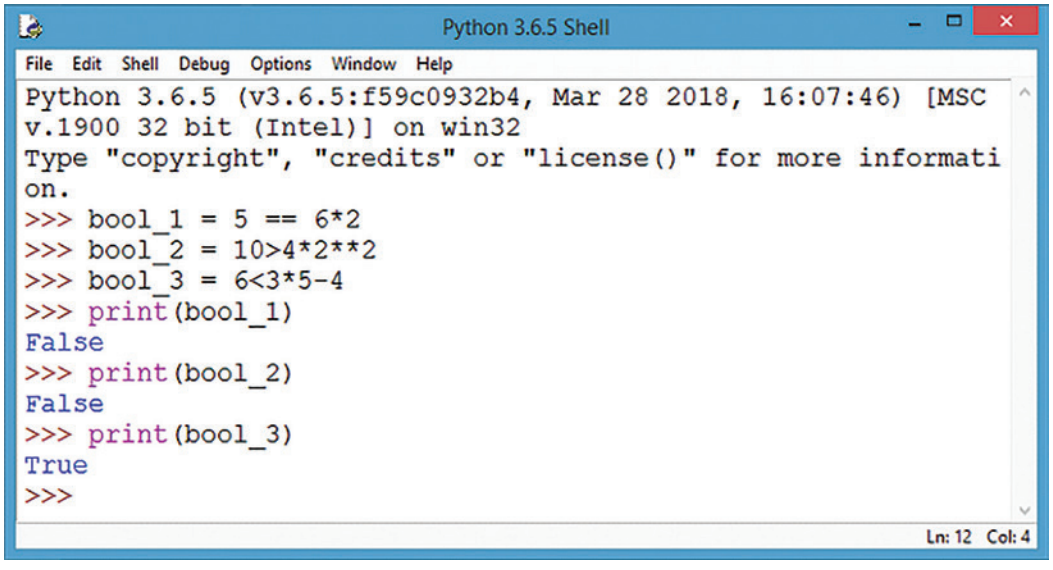
(b) **Float/Floating Point:** Floating point numbers signify real numbers. This data type is used to store numbers with a fraction part. They can be represented in scientific notation where the uppercase or lowercase letter 'e' signifies the 10^{th} power:



(c) **Complex Numbers:** Complex numbers are pairs of real and imaginary numbers. They take the form 'a + bj', where 'a' is the float and 'b' is the real part of the complex number.



(d) **Boolean:** Boolean data type is used in situations where comparisons to be made always result in either a true or a false value.



2. **None:** This is a special data type with an unidentified value. It signifies the absence of value in a situation, represented by None. Python doesn't display anything when we give a command to display the value of a variable containing value as None.
3. **Sequence:** A sequence is an ordered collection of items, indexed by integers (both positive as well as negative). The three types of sequence data types available in Python are Strings, Lists and Tuples, which we will discuss in successive topics.

4. **Sets:** Set is an unordered collection of values of any type with no duplicate entry. It is immutable.
5. **Mappings:** This data type is unordered and mutable. Dictionaries in Python fall under Mappings. A dictionary represents data in key-value pairs and accessed using keys, which are immutable. Dictionary is enclosed in curly brackets ({ }).

1.3.1 Dynamic Typing

One of the salient features of Python is dynamic typing. It refers to declaring a variable multiple times with values of different data types as and when required. It allows you to redefine a variable with different data types such as numeric, string, etc.

For example, consider the statement:

```
x = 20
```

The above statement signifies a variable 'x' of integer type as it holds an integer value. Now, suppose later in the program, you re-assign a value of different type to variable 'x' then, Python shall generate no error and allows the re-assignment with different set of values. *For example,*

```
x = 20
print(x)
x = "Computer Science"
print(x)
x = 3.276
print(x)
```

The above code on execution shall display the output as:

```
>>>20
>>>Computer Science
>>>3.276
```

In the above example, we have assigned three different values to the variable 'x' with different types. This process is referred to as **Dynamic typing**.

CTM: Each and every element in Python is referred to as an object.

1.4 KEYWORDS

Keywords are the reserved words used by a Python interpreter to recognize the structure of a program. As these words have specific meanings for the interpreter, they cannot be used as variable names or for any other purpose. For checking/displaying the list of keywords available in Python, we have to write the following two statements:

```
import keyword
print(keyword.kwlist)
```

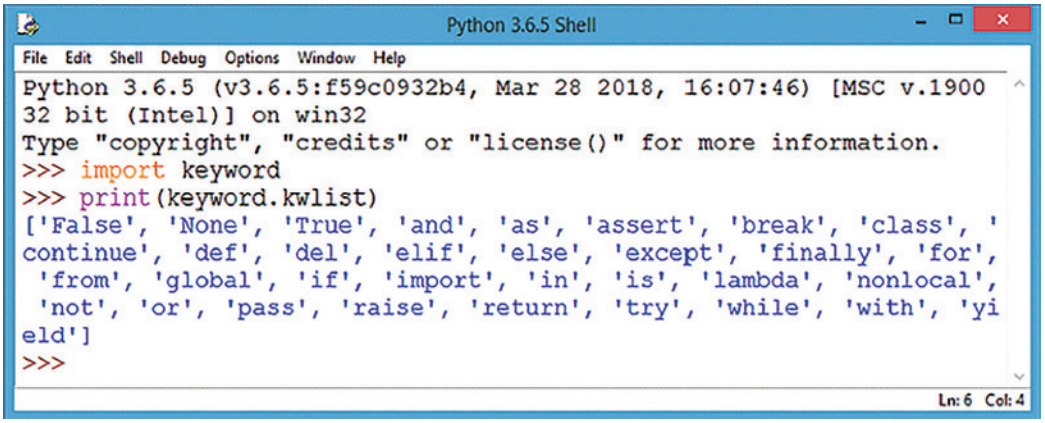


Fig. 1.3: Keywords in Python

CTM: All these keywords are in small letters, except for False, None, True, which start with capital letters.

1.5 MUTABLE AND IMMUTABLE TYPES

In certain situations, we may require changing or updating the values of certain variables used in a program. However, for certain data types, Python does not allow us to change the values once a variable of that type has been created and assigned values.

Variables whose values can be changed after they are created and assigned are called **mutable**.

Variables whose values cannot be changed after they are created and assigned are called **immutable**. When an attempt is made to update the value of an immutable variable, the old variable is destroyed and a new variable is created by the same name in memory.

Python data types can be classified into mutable and immutable as under:

- **Examples of mutable objects:** list, dictionary, set, etc.
- **Examples of immutable objects:** int, float, complex, bool, string, tuple, etc.

For example, int is an immutable type which, once created, cannot be modified.

Consider a variable 'x' of integer type:

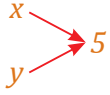
```
>>>x = 5
```

This statement will create a value 5 referenced by x.

Now, we create another variable 'y' which is copy of the variable 'x'.

```
>>>y = x
```

The above statement will make y refer to value 5 of x. We are creating an object of type int. Identifiers x and y point to the same object.



Now, we give another statement as:

```
>>>x = x + y
```

The above statement shall result in adding up the value of x and y and assigning to x.

Thus, x gets rebuilt to 10.

$x \longrightarrow 10$

$y \longrightarrow 5$

The object in which x was tagged is changed. Object x = 5 was never modified. **An immutable object doesn't allow modification after creation. Another example of immutable object is a string.**

```
>>>str = "strings immutable"
>>>str[0] = 'p'
>>>print(str)
```

This statement shall result in TypeError on execution.

TypeError: 'str' object does not support item assignment.

This is because of the fact that strings are immutable. On the contrary, a mutable type object such as a list can be modified even after creation, whenever and wherever required.

```
new_list = [10, 20, 30]
print(new_list)
```

Output:

```
[10, 20, 30]
```

Suppose we need to change the first element in the above list as:

```
new_list = [10, 20, 30]
new_list[0] = 100
```

print(new_list) will make the necessary updating in the list new_list and shall display the output as:

```
[100, 20, 30]
```

This operation is successful since lists are mutable.

Python handles mutable and immutable objects differently. Immutable objects are quicker to access than mutable objects. Also, immutable objects are fundamentally expensive to "change" because doing so involves creating a copy. Changing mutable objects is cheap.

1.6 OPERATORS AND OPERANDS

Python allows programmers to manipulate data or operands through operators. Operators are the symbols that operate upon these operands to form an expression. Operators available in Python are categorized as follows:

Table 1.1: Operators in Python

Arithmetic Operators
Assignment Operators
Relational or Comparison Operators
Logical Operators
Identity Operators
Bitwise Operators
Membership Operators

➢ **Arithmetic/Mathematical Operators:** + (addition), – (subtraction), * (multiplication), / (Division), ** (Exponent), % (Modulus), // (Floor Division).

Operator	Description	Example(s)
+ (unary)	To explicitly express a positive number	Value of +3 is +3
– (unary)	To represent a negative number	Value of –3 is –3
+ (binary)	To add two numbers	Value of 23+3.5 is 26.5
– (binary)	To subtract one value from the other	Value of 45 – 32 is 13
*	To find the product of two numbers	Value of 3.2*6 is 19.2
/	To find the quotient when one value is divided by the other.	• Value of 3/2 is 1.5 • Value of –3/2 is –1.5 • Value of 10.0/3 is 3.3333333333333335
//	(Floor division) To find the integar part of the quotient when one number is divided by the other. The result is always the largest integer less than or equal to the actual quotient.	• Value of 3//2 is 1 • Value of –3//2 is –2 • Value of 10.0//3 is 3.0
%	(Remainder) To find the remainder when one value is divided by the other.	• Value of 3%2 is 1 • Value of 10%6 is 4 • Value of 6%10 is 6 • Value of 10.3%3.2 is 0.70000000000000002
**	(Exponent) To raise a number to the power of another number. The expression a**b is used to find the value of a^b.	• Value of 3**2 is 9 • Value of 3**–2 is 0.1111111111111111 • Value of 10.2**3.1 is 1338.6299344200029

➢ **Assignment Operators:** = (Assignment), += (Add and Assign), –= (Subtract and Assign), *= (Multiply and Assign), /= (Divide and Assign Quotient), **= (Exponent and Assign), %= (Divide and Assign Remainder), //= (Floor division and Assign).

Operator	Description	Example

+= (Add and Assign)	Evaluates R-value and adds it to L-value. The final result is assigned to L-value.	`x=5`
		`y = 10`
		`x += 2*y`
		`print("x=",x, "and y=",y)`
		gives the result: `x= 25 and y= 10`

–= (Subtract and Assign)	Evaluates R-value and subtracts it from L-value. The final result is assigned to L-value.	`x=5`
		`y = 10`
		`x –= 2*y`
		`print("x=",x, "and y=",y`
		gives the result: `x= –15 and y= 10`

*= (Multiply and Assign)	Evaluates R-value and multiplies it with L-value. The final result is assigned to L-value.	`x=5`
		`y = 10`
		`x *= 2*y`
		`print("x=",x, "and y=",y)`
		gives the result: `x= 100 and y= 10`

/= (Divide and Assign Quotient)	Evaluates R-value and divides the L-value with R-value. The quotient is assigned to L-value.	x=5 y = 10 x /= y print("x=",x, "and y=",y) gives the result: x= 0.5 and y= 10
%= (Divide and Assign Remainder)	Evaluates R-value and divides the L-value with R-value. The remainder is assigned to L-value.	x=5 x%= 4 print("x=",x) gives the result: x=1
//= (Floor division and Assign)	Evaluates R-value and divides (floor division) the L-value with R-value. The quotient is assigned to L-value.	x=5 x //= 4 print("x=",x) gives the result: x=1
**= (Exponent and Assign)	Evaluates R-value and calculates (L-value)$^{R\text{-value}}$. The final result is assigned to L-value.	x=5 x **= 4 print("x=",x) gives the result: x= 625

➢ **Relational/Comparison Operators:** > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), == (equal to), != (not equal to).

Operator	Description	Example (assuming x=6, y=2)
== (Equality)	Compares two values for equality. Returns True if they are equal, otherwise returns False.	(x==y) returns False
!= (Inequality)	Compares two values for inequality. Returns True if they are unequal, otherwise returns False.	(x!=y) returns True
< (Less than)	Compares two values. Returns True if first value is less than the second, otherwise returns False.	(x<y) returns False
<= (Less than or equal to)	Compares two values. Returns True if first value is less than or equal to the second, otherwise returns False.	(x<=y) returns False
> (Greater than)	Compares two values. Returns True if first value is greater than the second, otherwise returns False.	(x>y) returns True
>= (Greater than or equal to)	Compares two values. Returns True if first value is greater than or equal to the second, otherwise returns False.	(x>=y) returns True

➢ **Logical Operators:** or, and, not.

Operator	Description	Example (assuming x=6, y=2)
not	Negates a condition and returns True if the condition is false, otherwise returns False.	not(x > y) returns False
and	Combines two conditions and returns True if both the conditions are true, otherwise returns False.	(x >3 and y<2) returns False
or	Combines two conditions and returns True if at least one of the conditions is true, otherwise returns False.	(x>3 or y<2) returns True

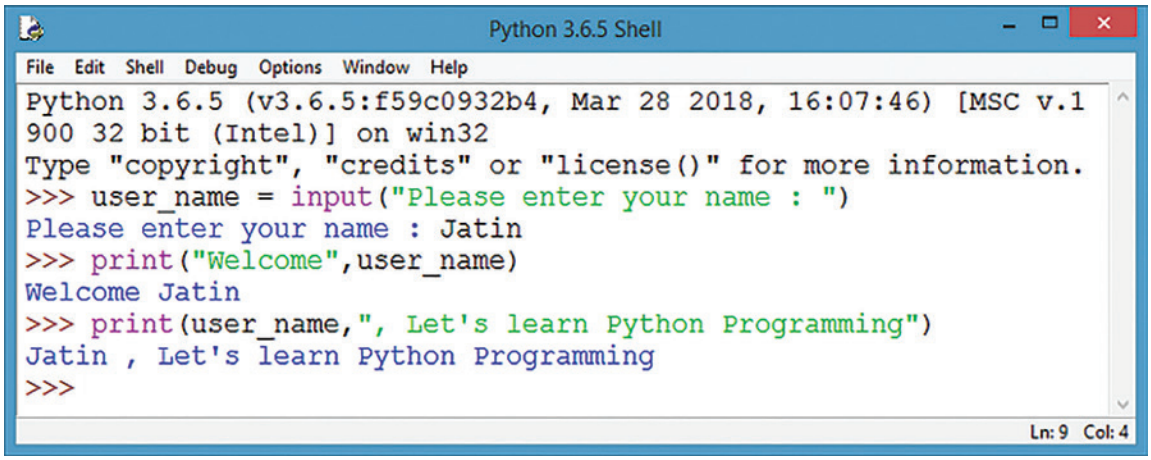
1.7 INPUT AND OUTPUT (PYTHON'S BUILT-IN FUNCTIONS)

In order to provide the required set of values, data is to be fetched from a user to accomplish the desired task. Thus, Python provides the following I/O (Input-Output) built-in library functions:

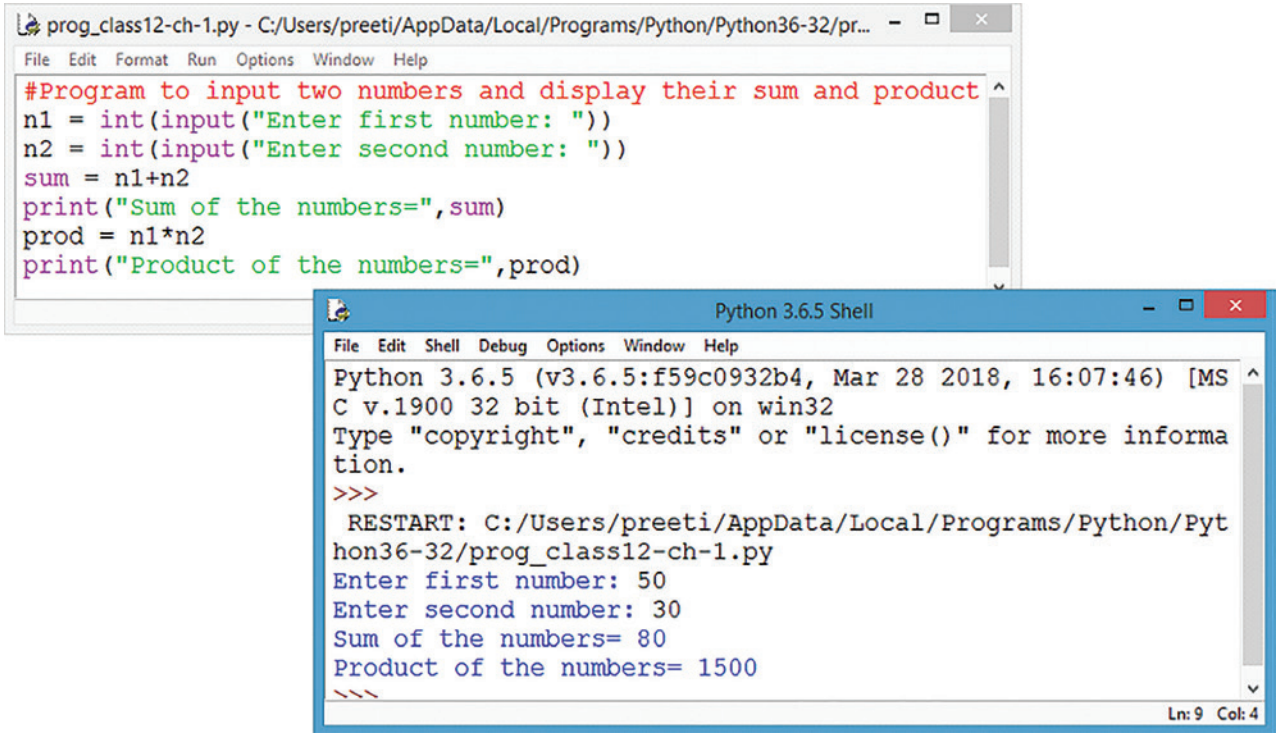
1. **input():** The input() function accepts and returns the user's input as a string and stores it in the variable which is assigned with the assignment operator. It is important to remember that while working with input(), the input fetched is always a string; so, in order to work with numeric values, use of appropriate conversion function (int) becomes mandatory.

The input() function takes one string argument (called prompt). During execution, input() shows the prompt to the user and waits for the user to input a value from the keyboard. When the user enters a value, input() returns this value to the script. In almost all the cases, we store this value in a variable.

Example 3: Display a welcome message to the user.



Example 4: Input two numbers from the user and display their sum and product.



In the above program, we have used the **int() method**. int() takes a number, expression or a string as an argument and returns the corresponding integer value. int() method behaves as per the following criteria:

(a) If the argument is an integer value, `int()` returns the same integer. *For example*, `int(12)` returns 12.

(b) If the argument is an integer expression, the expression is evaluated and `int()` returns this value. *For example*, `int(12+34)` returns 46.

(c) If the argument is a float number, `int()` returns the integer part (before the decimal point) of the number. *For example*, `int(12.56)` returns 12.

2. **eval():** eval() method takes a string as an argument, evaluates this string as a number, and returns the numeric result (int or float as the case may be). If the given argument is not a string or if it cannot be evaluated as a number, then eval() results in an error.

1.7.1 Type Casting (Explicit Conversion)

As and when required, we can change the data type of a variable in Python from one type to another. Such data type conversion can happen in two ways:

- either explicitly (forced) when the programmer specifies for the interpreter to convert a data type into another type; or
- implicitly, when the interpreter understands such a need by itself and does the type conversion automatically.

➢ **Explicit Conversion**

Explicit conversion, also called type casting, happens when data type conversion takes place deliberately, *i.e.*, the programmer forces it in the program. The general form of an explicit data type conversion is:

(new_data_type) (expression)

With explicit type conversion, there is a risk of data loss since we are forcing an expression to be of a specific type.

For example, converting a floating value of x = 50.75 into an integer type, *i.e.*, int(x) will discard the fractional part .75 and shall return the value as 50.

```
>>> x = 50.75
>>> print(int(x))
50
```

Following are some of the functions in Python that are used for explicitly converting an expression or a variable into a different type.

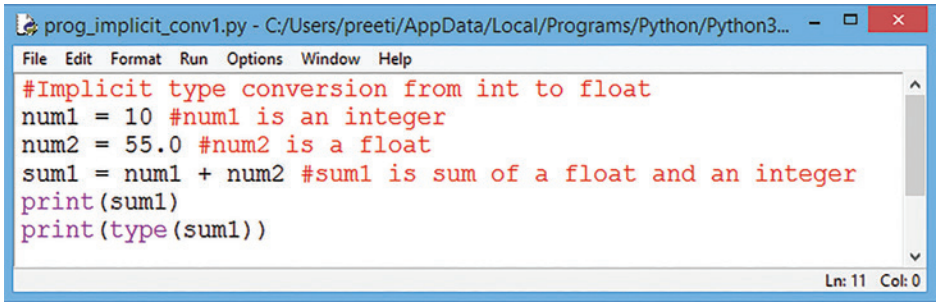
Table 1.2: Explicit type conversion functions in Python

Function	Description
`int(x)`	Converts x into an integer.
`float(x)`	Converts x into a floating-point number.
`str(x)`	Converts x into a string representation.
`chr(x)`	Converts x into a character.
`unichr(x)`	Converts x into a Unicode character.

➢ Implicit Conversion

Implicit conversion, also known as coercion, happens when data type conversion is done automatically by Python and is not instructed by the programmer.

Example 5: Program to illustrate implicit type conversion from int to float.



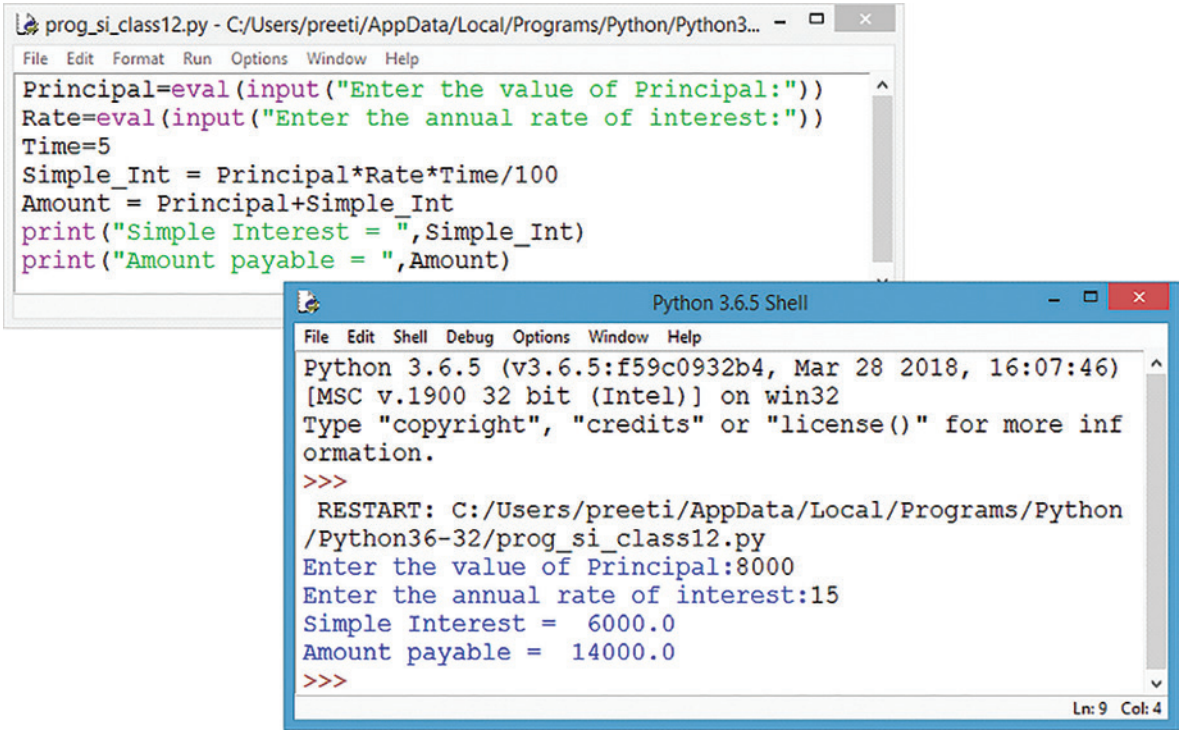

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Pyt
hon/Python37-32/prog_implicit_conv1.py
65.0
<class 'float'>
>>>
```

In the above example, an integer value stored in variable num1 is added to a float value stored in variable num2, and the result gets automatically converted into a float value stored in variable sum1 without explicitly telling the system. This is an example of implicit data conversion.

The reason for the float value not converted into an integer instead is due to type promotion that allows performing operations (whenever possible) by converting data into a wider-sized data type without any loss of information.

Example 6: Write a Python code to calculate simple interest and amount payable by inputting the value of principal amount and rate from the user for a time period of 5 years.

(Formula used: *Simple Interest = Principal * Rate * Time/100*)

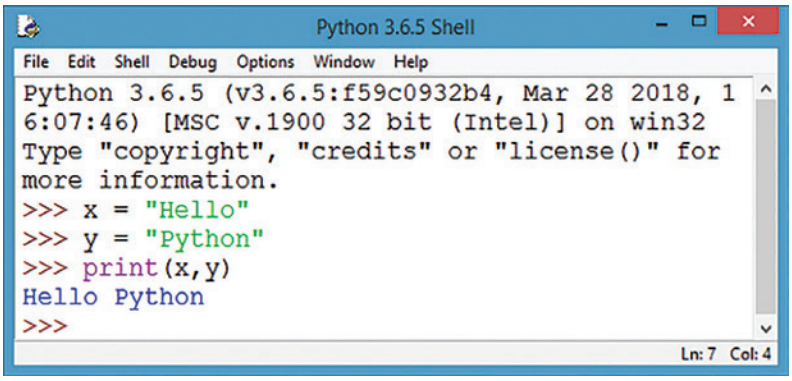


3. **print():** print() is a function which is used to display the specified content on the screen. The content (called argument) is specified within the parentheses. print() statement without any arguments will simply jump to the next line.

   With Python version 3.x onwards, print() is considered a function rather than a statement. Always enclose the text/parameters within the rounded parentheses inside the print() function.

CTM: A print() method without any argument, *i.e.*, without any value or name or expression shall print/ display a blank line.

Example 7:



1.8 COMMENTS IN PYTHON

Comments are statements in a script that are ignored by the Python interpreter and, therefore, have no effect on the actual output of the code. Comments make the code more readable and understandable. A ***comment (single-line)*** in Python starts with a ***hash symbol (#)*** anywhere in a line and extends till the end of the line.

Anything written after '#' in a line is ignored by Python interpreter. In case of multi-line comments, we can use triple quoted strings (single or double strings): ''' ''' or """ """.

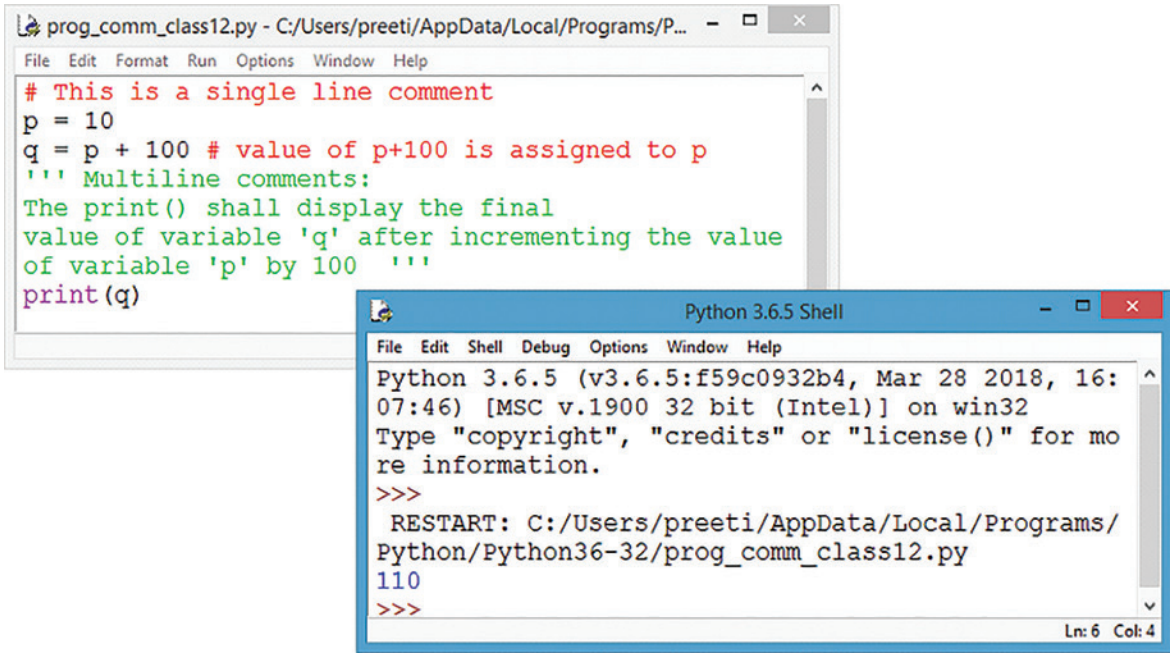


Fig. 1.4: Single and Multi-line Comments in Python

1.9 FLOW OF EXECUTION

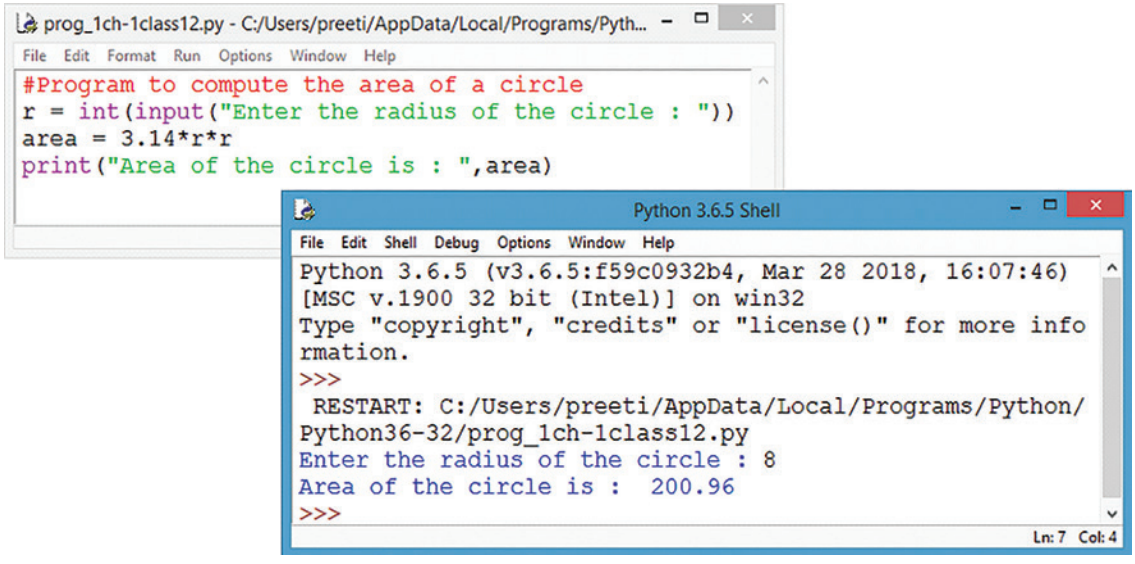
Execution in a Python program begins with the very first statement. The way in which the statements are executed defines the flow of execution in a Python program, which is categorized as under:

(i) Sequential statements
(ii) Selection/Conditional statements
(iii) Iteration or Looping constructs

1.9.1 Sequential Statements

Sequential, as the name suggests, signifies the way where statements in a Python program are executed one after the other, *i.e.*, from the first statement till the last statement.

Example 8: Program to calculate the area of a circle.



As is evident from the above program, all the statements are executed one after the other.

1.9.2 Selection/Conditional Statements

Conditional statements are used to perform actions or calculations based on whether a condition is evaluated as true or false. In programming, decision-making or selection can be achieved through the conditional statement. The simplest form is the ***if statement***. if statement increases the utility of a program (Table 1.3).

Table 1.3: if statement

Syntax	Example

`if condition:`
`    statement(s)`
Header
	`Age= int(input("Enter your age:"))`
	`if Age>=18:`
	`    print("Eligible to vote")`

	`number=int(input("Enter a number:"))`
	`if number>0:`
	`    print("Number is positive")`

- If the condition is true, then the indented statement gets executed.
- The indented statement implies that its execution is dependent on the header.

CTM: Do not forget to place colon (:) after if condition or if statement as shown in the given syntax.

The second type of conditional ***if*** statement, called ***if...else***, provides an alternative execution. ***if...else*** allows for two possibilities and the condition determines which branch gets executed (Table 1.4).

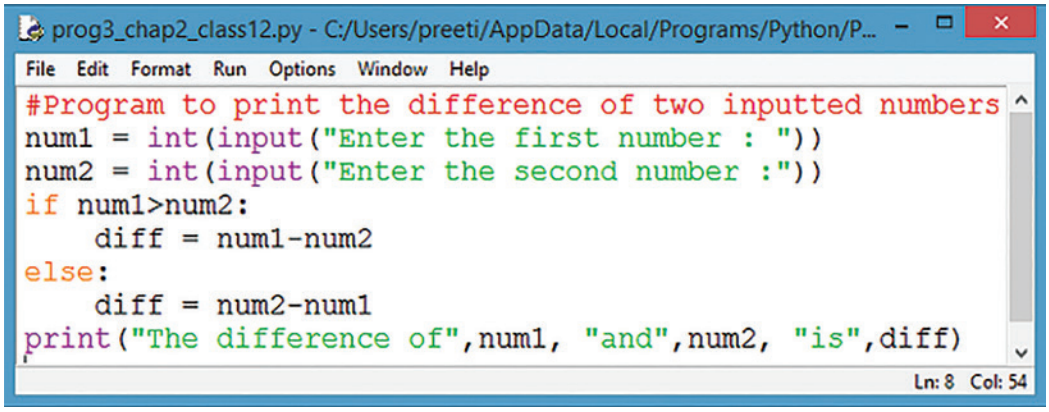
Table 1.4: if...else statement

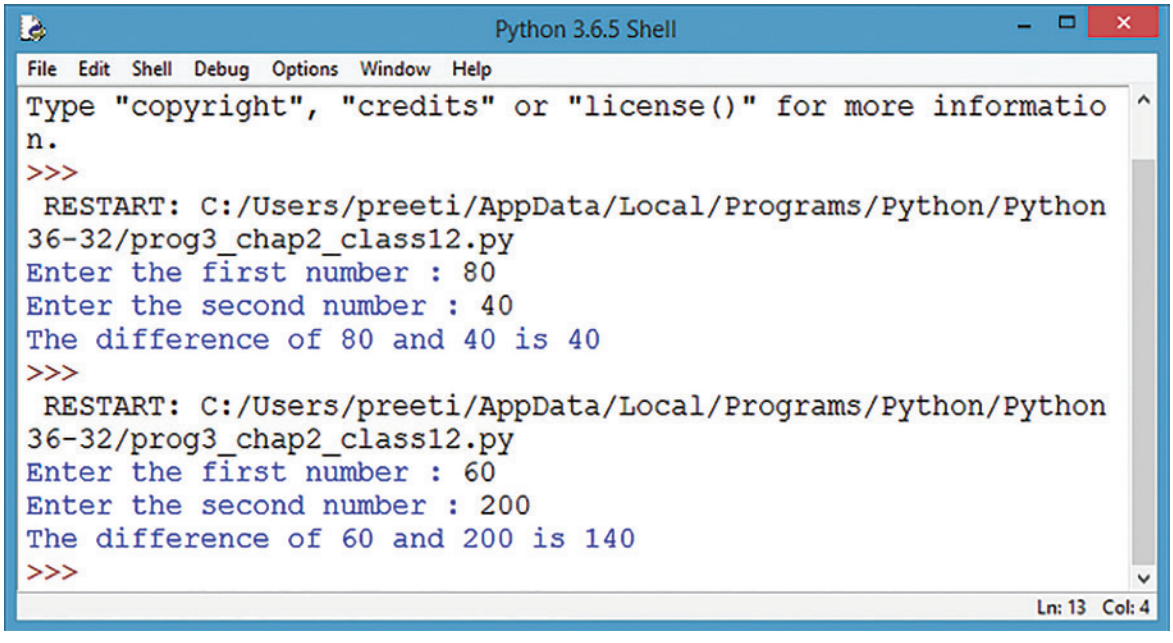
Syntax	Example

if condition:
  statement(s)
else:
  statement(s)
	Age= int(input("Enter your age:"))
	if Age>=18:
	  print("Eligible to vote")
	else:
	  print("Not eligible to vote")

	number=int(input("Enter a number:"))
	if number>0:
	  print("Number is positive")
	else:
	  print("Number is negative")

Example 9: Program to calculate and display the difference of two inputted numbers.





As shown in the above program, the flow of program is based on a single condition and the statements following the respective block for 'if' and 'else' are executed accordingly. In case there are multiple conditions to be handled, we have to switch over to ***multiple if-elif-else*** statements as shown in Table 1.5.

Table 1.5: if...elif...else statement

Syntax	Example

`if condition:`
`  statement(s)`
`elif condition:`
`  statement(s)`
`elif condition:`
`  statement(s)`
`else:`
`  statement(s)`
	`number=int(input("Enter a number:"))`
	`if number > 0:`
	`  print("Number is positive")`
	`elif number < 0:`
	`  print("Number is negative")`
	`else:`
	`  print("Number is zero")`

	`signal = input("Enter a colour:")`
	`if signal=="red" or signal=="RED":`
	`  print("STOP")`
	`elif signal=="orange" or signal=="ORANGE":`
	`  print("Be Slow")`
	`elif signal=="green" or signal=="GREEN":`
	`  print("Go!")`

In if...elif...else construct,

- else is optional
- Number of elif is dependent on the number of conditions
- If the first condition is false, the next is checked, and so on. If one of the conditions is true, the corresponding statement(s) executes, and the statement ends.

POINT TO REMEMBER

The condition in an if or elif clause can be any Python expression, even a statement that returns a value (even a None value).

Another modification to multiple if...elif...else construct is the ***Nested if...else statement***.

The nested if...else statement allows us to check for multiple test expressions and execute different codes for more than two conditions. We can have many levels of nesting inside if... else statements.

Example 10: Program to illustrate nested if...else through a four-function calculator.

prog_nested_if1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-...

File Edit Format Run Options Window Help

```
#Program to create a four function calculator
result = 0
val1= float(input("Enter value 1: "))
val2= float(input("Enter value 2: "))
op = input("Enter any of the operator (+,-,*,/): ")
if op == "+":
    result = val1 + val2
elif op == "-":
    if val1>val2:
        result = val1-val2
    else:
        result = val2 - val1
elif op == "*":
    result = val1 * val2
elif op == "/":
    if val2 == 0:
        print("Please enter a value other than 0")
    else:
        result = val1/val2
else:
    print("Please enter any one of the operator(+,-,*,/)")
print("The result is: ",result)
```

Ln: 27 Col: 0

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
1)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_nested_
if1.py
Enter value 1: 200
Enter value 2: 400
Enter any of the operator (+,-,*,/): *
The result is:  80000.0
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_nested_
if1.py
Enter value 1: 200
Enter value 2: 400
Enter any of the operator (+,-,*,/): -
The result is:  200.0
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_nested_
if1.py
Enter value 1: 200
Enter value 2: 400
Enter any of the operator (+,-,*,/): /
The result is:  0.5
```

```
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_nested_
if1.py
Enter value 1: 200
Enter value 2: 400
Enter any of the operator (+,-,*,/): +
The result is:  600.0
>>>
```

Ln: 27 Col: 4

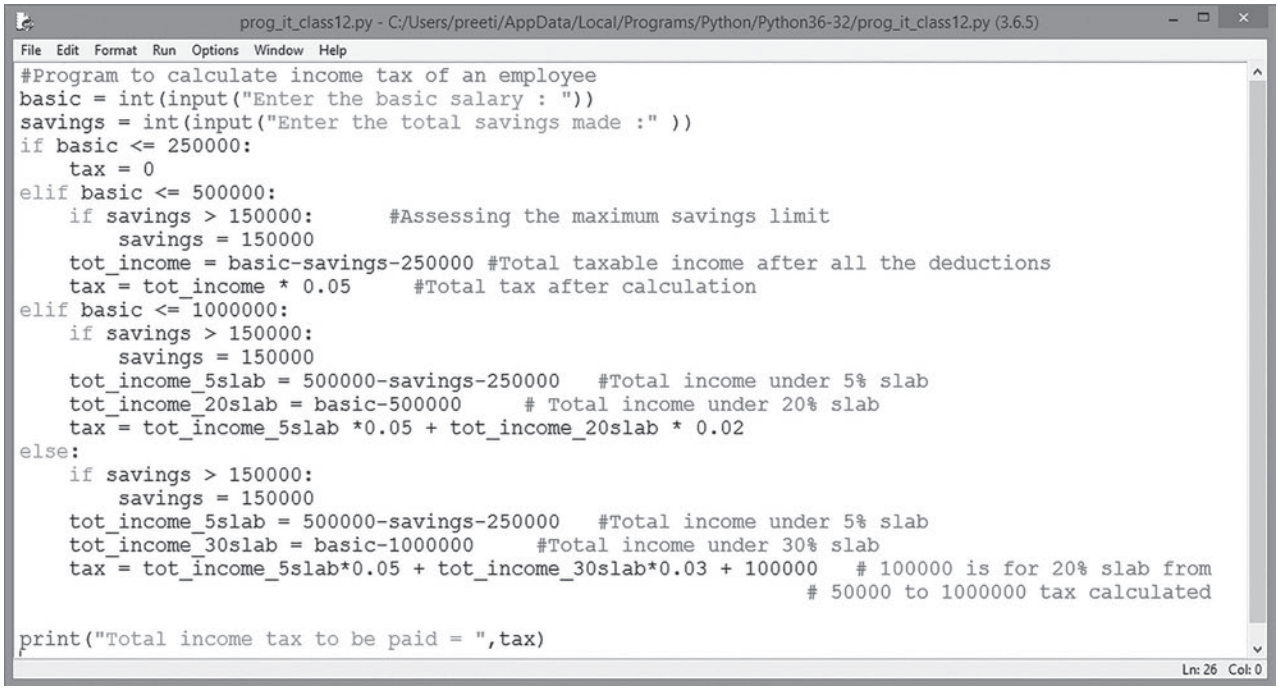
As we can see, for the operators "–" and "/", there exists an if...else condition within the elif block. This is called nested if. We can have many levels of nesting inside if...else statements.

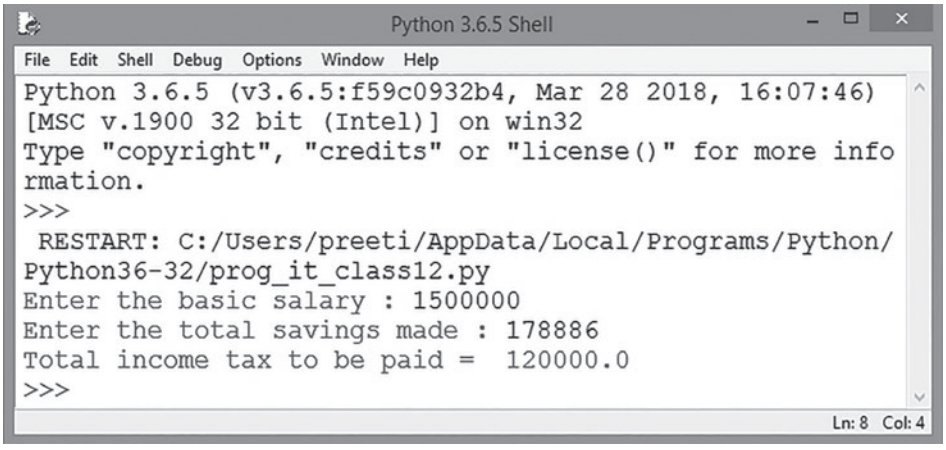
Syntax	Example

if condition:
  statement(s)
  if condition:
  statement(s)
  elif condition:
  statement(s)
  else:
  statement(s)
elif condition:
  statement(s)
else:
  statement(s)
	var = 100
	if var < 200:
	  print("value is less than 200")
	  if var == 150:
	  print("value is 150")
	  elif var == 100:
	  print("value is 100")
	  elif var == 50:
	  print("value is 50")
	  elif var < 50:
	  print("value is less than 50")
	else:
	  print("Could not find true")
	print("Good bye!")

Example 11: Program to calculate income tax of an employee on the basis of basic salary and total savings inputted by the user (using nested if...else statement) as per the given slabs:

- No tax for individuals with income less than ₹ 2.5 lakh
- 0%–5% tax with income ₹ 2.5 lakh to ₹ 5 lakh for different age groups
- 20% tax with income ₹ 5 lakh to ₹ 10 lakh
- Investments up to ₹ 1.5 lakh under Sec 80C can save up to ₹ 45,000 in taxes.





1.9.3 Looping Constructs

Looping constructs are the programming constructs that provide the capability to execute a set of statements in a program repetitively, based on a condition. The statements in a loop are executed again and again as long as a particular logical condition remains true.

Python provides two types of loops: the 'for loop' and the 'while loop' represented as counting loop and conditional loop respectively. These looping statements allow a block of statements to be repeated a number of times on the basis of a loop control variable.

for Loop

The for statement is used to iterate/repeat itself over a range of values or a sequence one by one. The for loop is executed for each of these items in the range. With every iteration of the loop, the control variable checks whether each of the values in the range has been traversed or not. When all the items in the range are exhausted, the body of the loop is not executed any more; the control is then transferred to the statement immediately following the for loop.

Syntax of for Loop:

```
for <Control_variable> in <sequence/ items in range>:
    <statements in body of loop>
else: # optional
    <statements>
```

Here,

- sequence may be a list, string, tuple
- 'else' statement will be executed after all the iterations of for loop, if provided
- Control_variable is a variable that takes a new value from the range each time the loop is executed.

The examples given below exhibiting the implementation of 'for' loop use a most commonly used built-in Python library function, **range()**, which we are going to discuss first.

range():

The range() is a built-in function in Python and is used to create a list containing a sequence of numbers starting with the ***start*** and ending with one less than the ***stop***.

Syntax:

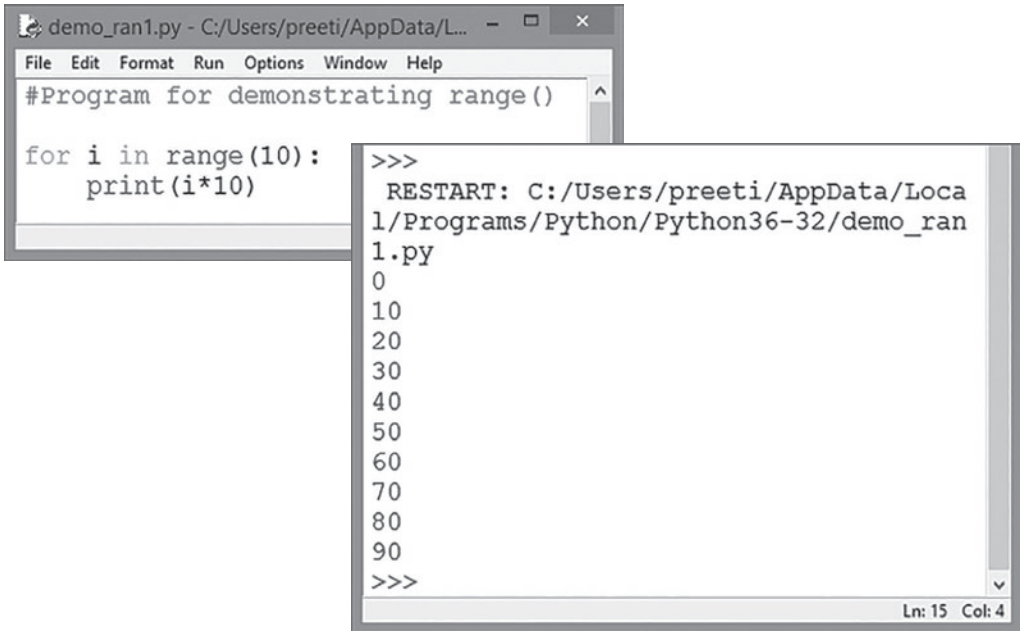
```
range(start, stop, step)
```

The start and step parameters are optional. By default, the list starts from 0, and in every iteration, it is incremented by one but we can specify a different increment value by using the ***step*** parameter. range() is often used in *for* loops.

Command	Output
>>> range(10)	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 30, 5)	[0, 5, 10, 15, 20, 25]
>>> range(0, -9, -1)	[0, -1, -2, -3, -4, -5, -6, -7, -8]

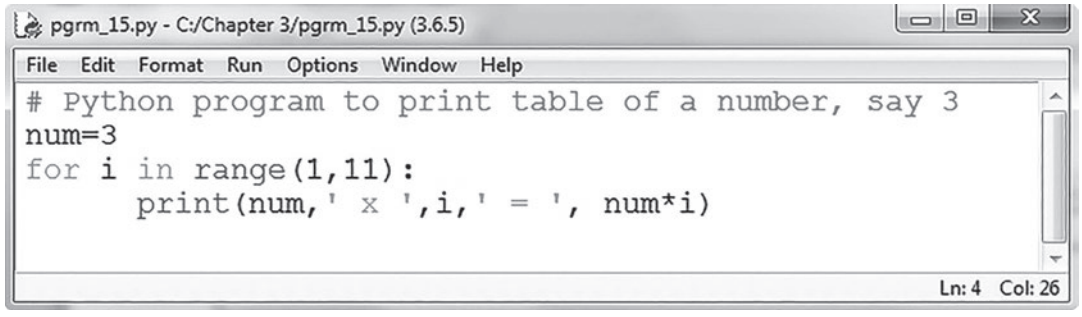
The arguments of range() function must be integers. The *step* parameter can be any positive or negative integer other than zero.

For example,

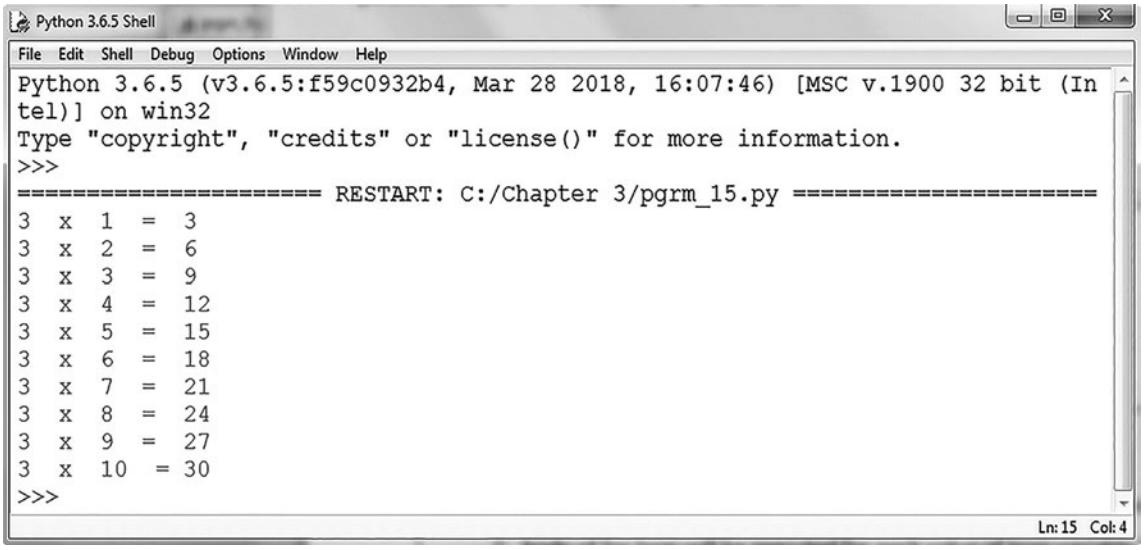


As observed from the above example, using for loop does not require loop variable (i) to be defined explicitly beforehand.

Example 12: Program to generate the table of a number using for loop.

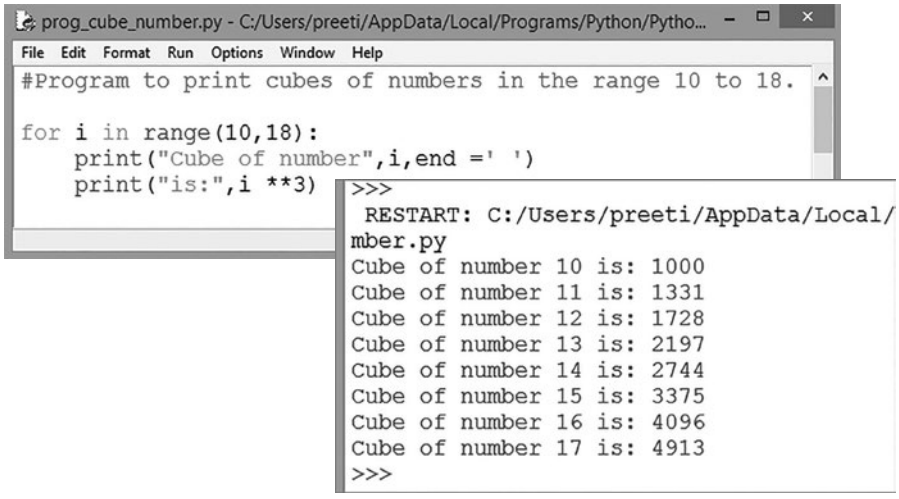


In the above program, the range function has been used which will generate values 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Also, 'i' is the loop control variable and will be assigned each generated value, for which the 'body of for loop' will be executed.

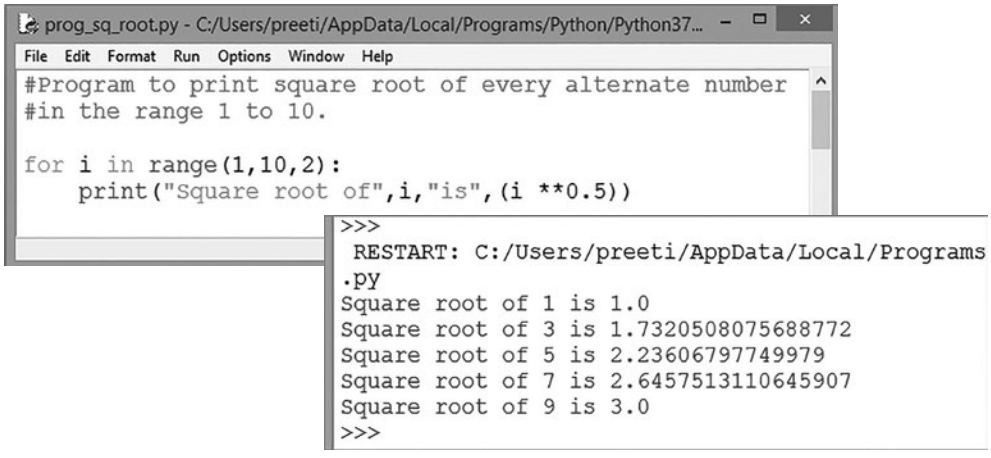


CTM: The statements given in the line after colon (:) are at the same indentation level which forms the body of a loop.

Example 13: Write a program to print cubes of numbers in the range 10 to 18.



Example 14: Write a program to print square root of every alternate number in the range 1 to 10.



while Loop

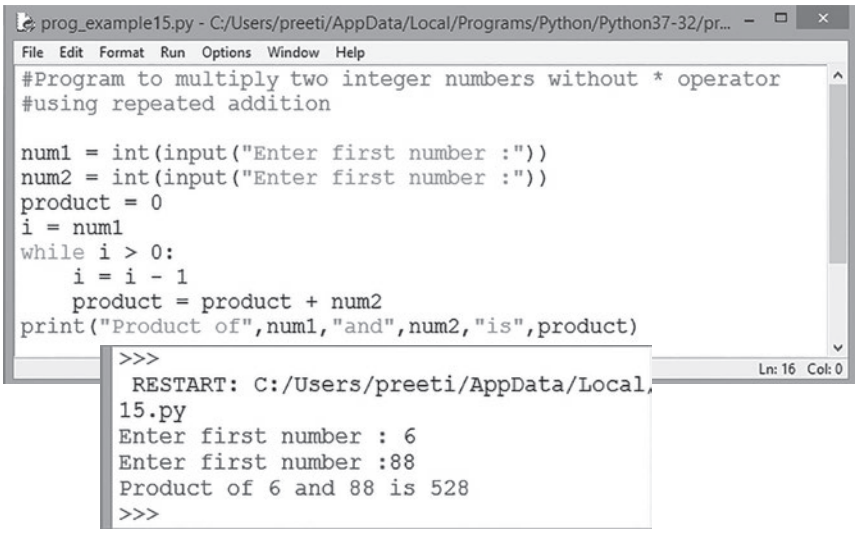
The while loop repeatedly executes the set of statements till the defined condition is true. As soon as the condition evaluates to false, control passes to the first line written after the loop.

Syntax of while Loop:

```
while <test_expression>:
    Body of while
else: # optional
    Body of else
```

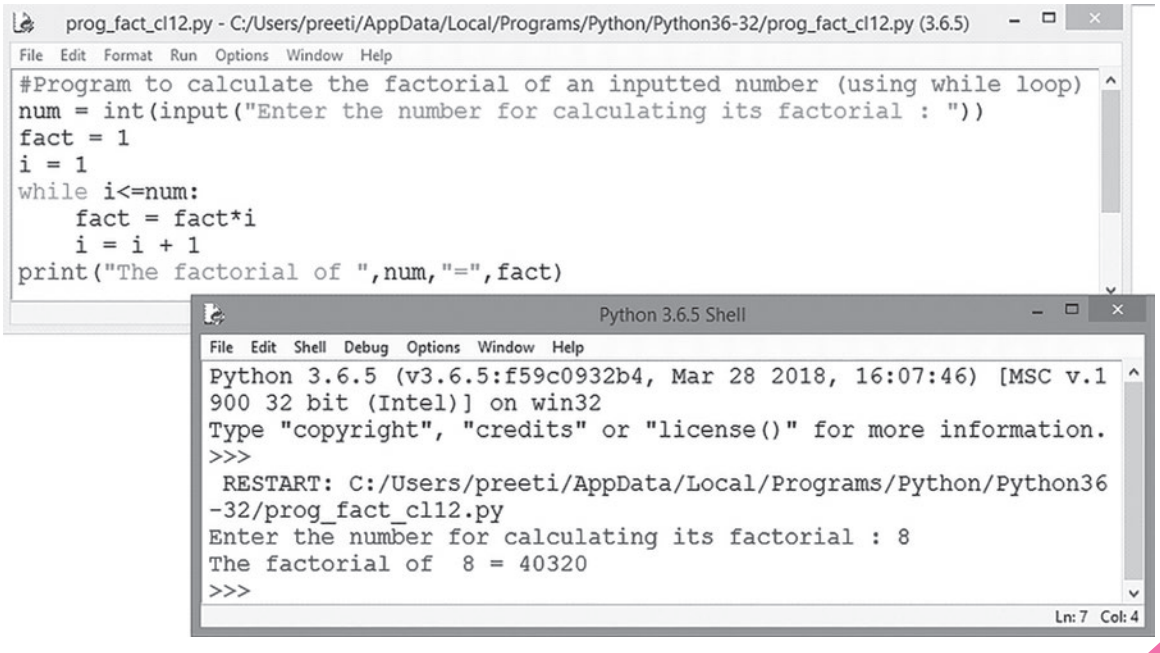
- 'while' is a reserved word
- test_expression is the condition to be checked for true/false value
- '*Body of while*' consists of single or multiple statements or even an empty statement, *i.e.*, pass statement
- '*else*' is optional and executed when test condition evaluates to false.

Example 15: Write a program to calculate the product of two inputted integers without using * operator, instead using repeated addition.



Generally, a while loop is used when you don't know in advance about how many times the loop will be executed, but the control expression/termination condition is known.

Example 16: Program to calculate factorial of a number using while loop.



Explanation:

In the above program, factorial of an inputted number is calculated.

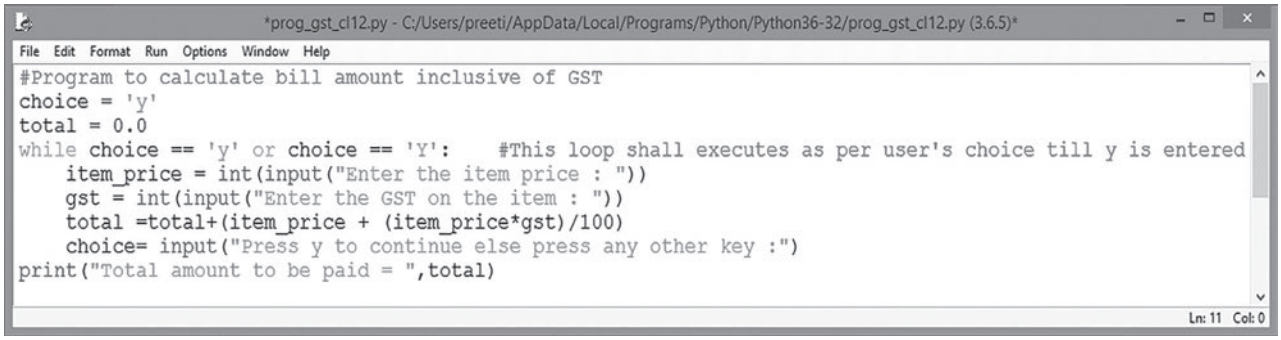
The factorial of a number is the product of all the integers from 1 to that number.

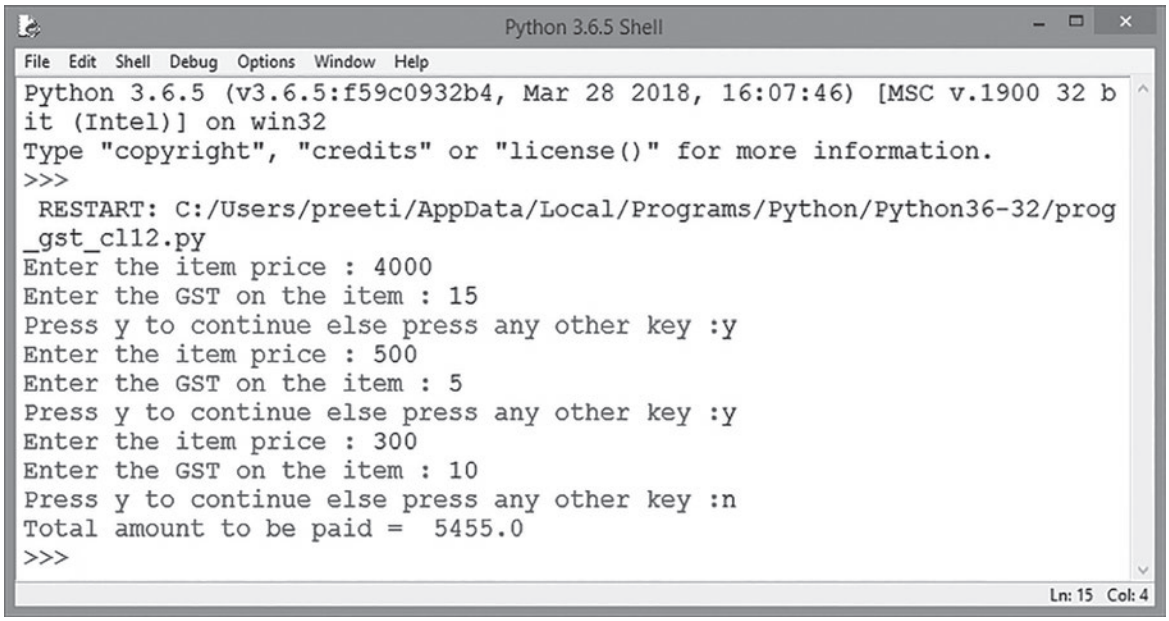
For example, the factorial of 8 (denoted as 8!) is 1*2*3*4*5*6*7*8 = 40320.

Factorial is not defined for negative numbers and the factorial of zero is one, *i.e.*, 0! = 1

To calculate the factorial, input of the number is accepted from the user. A factorial variable 'fact' is initialized to 1. A while loop is used to multiply the number to find the factorial. The process continues until the value of control/loop var. i becomes equal to the inputted number 'num'. In the last statement, the factorial of the given number is printed.

Example 17: Program to calculate the total amount payable by the customer on purchase of any item with GST levied on it. Develop a user-friendly approach for the program using while loop.





1.10 STRINGS

In Python, a string is a sequence of characters enclosed within quotes. Python treats single quotes and double quotes as equal. An individual character in a string is accessed using a subscript (index). The subscript should always be an integer (positive/negative) and begin with 0.

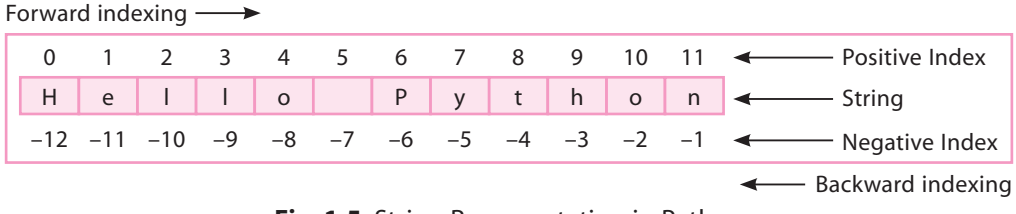


Fig. 1.5: String Representation in Python

Strings are immutable, *i.e.*, we cannot modify/change the contents of a string after its creation. In other words, we can say, Item assignment is not supported in strings.

1.10.1 String Operations

String can be manipulated using operators like concatenate (+), repetition (*), and membership operators like in and not in. Let us take a quick look at the important string operations available in Python:

Operator	Name	Description
+	Concatenation	Adds or joins two strings
*	Repetition	Concatenates multiple copies of the same string
in/not in	Membership	Returns true if a character exists in the given string
[:]	Range(start, stop, [step])	Extracts the characters from the given range
[ ]	Slice[n : m]	Extracts the characters from the given index

Let us understand these operations using the example given below:

Example 18: Consider two strings:

```
str1 = "Hello"
str2 = "Python"
```

Observe the result obtained after performing important string operations.

1.10.2 String Slicing

Slicing is used to retrieve a subset of values. A slice of a string is nothing but a substring. This extracted substring is termed as **slice**. A chunk of characters can be extracted from a string using slice operator with three indices in square brackets separated by colon ([:]).

The syntax is:

Syntax:

```
String_name[start:end:step]
```

Here,

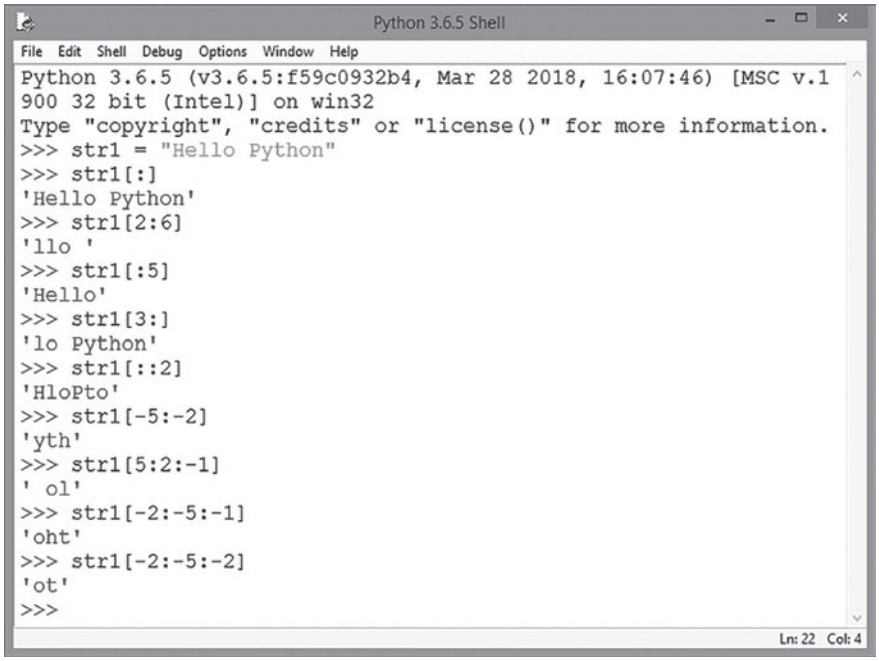
- start—starting integer where the slicing starts
- end—position till which the slicing takes place. The slicing stops at index end-1.
- step—integer value which determines the increment between each index for slicing.

start, stop values are optional. If a single parameter to passed, start and end are set to None.

Example 19: Consider a string str1 with the following content:

```
str1 = "Hello Python"
```

The various slice operations and the output retrieved are shown below:



1.10.3 Built-in String Methods

Python provides the following built-in methods to manipulate strings:

Method	Description	Example

isalpha()	Returns True if the string contains only letters, otherwise returns False.
	Syntax:
	`str.isalpha()`
		`>>> str = "Good"`
		`>>> print(str.isalpha())`
		`True`
		#Returns True as no special character or digit is present in the string.
		`>>> str1="This is a string"`
		`>>> print(str1.isalpha())`
		`False`
		#Returns False as the string contains spaces.
		`>>> str1="Working with...Python!!"`
		`>>> print(str1.isalpha())`
		`False`
		#Returns False as the string contains special characters and spaces.

isdigit()	This method returns True if string contains only digits, otherwise False.
	Syntax:
	`str.isdigit()`
		`>>> str1="123456"`
		`>>> print(str1.isdigit())`
		`True`
		#Returns True as the string contains only digits.
		`>>> str1 = "Ram bagged 1st position"`
		`>>> print(str1.isdigit())`
		`False`
		#Returns False because apart from digits, the string contains letters and spaces.

lower()	Converts all the uppercase letters in the string to lowercase.
	Syntax:
	`str.lower()`
		`>>> str1= "Learning PYTHON"`
		`>>> print(str1.lower())`
		`learning python`
		#Converts uppercase letters only to lowercase.
		`>>> str1= "learning python"`
		`>>> print(str1.lower())`
		`learning python`
		#if already in lowercase, then it will simply return the string.

islower()	Returns True if all letters in the string are in lowercase.
	Syntax:
	`str.islower()`
		`>>> str1 ="python"`
		`>>> print(str1.islower())`
		`True`
		`>>> str1 = "Python"`
		`>>> print(str1.islower())`
		`False`

upper()	Converts lowercase letters in the string to uppercase.
	Syntax:
	`str.upper()`
		`>>> var1= "Welcome"`
		`>>> print(var1.upper())`
		`WELCOME`
		`>>> var1= "WELCOME"`
		`>>> print(var1.upper())`
		`WELCOME`
		#if already in uppercase, then it will simply return the string.

isupper()	Returns True if the string is in uppercase.
	Syntax:
	`str.isupper()`
		`>>> str1= "PYTHON"`
		`>>> print(str1.isupper())`
		`True`
		`>>> str1= "PythOn"`
		`>>> print(str1.isupper())`
		`False`

lstrip()
or
lstrip(chars)
	Returns the string after removing the space(s) from the left of the string.
	Syntax:
	`str.lstrip()`
	or
	`str.lstrip(chars)`
	chars (optional) – a string specifying the set of characters to be removed from the left. All combinations of characters in the *chars* argument are removed from the left of the string until the left character of the string mismatches.
		`>>> str1= " Green Revolution"`
		`>>> print(str1.lstrip())`
		`Green Revolution`
		#Here no argument was given, hence it removed all leading whitespaces from the left of the string.
		`>>> str2= "Green Revolution"`
		`>>> print(str2.lstrip("Gr"))`
		`een Revolution`
		`>>> str2= "Green Revolution"`
		`>>> print(str2.lstrip("rG"))`
		`een Revolution`
		#Here all elements of the given argument are matched with the left of the str2 and, if found, are removed.

rstrip()
or
rstrip(chars)
	This method removes all the trailing whitespaces from the right of the string.
	Syntax:
	`rstrip( )`
	or
	`str.rstrip(chars)`
	chars (optional) – a string specifying the set of characters to be removed from the right. All combinations of characters in the *chars* argument are removed from the right of the string until the right character of the string mismatches.
		`>>> str1= "Green Revolution"`
		`>>> print(str1.rstrip())`
		`Green Revolution`
		#Here no argument was given, hence it removed all leading whitespaces from the right of the string.
		`>>> str1= "Computers"`
		`>>> print(str1.rstrip("rs"))`
		`Compute`
		
		#Here the letters 'rs' are passed as an argument; it is matched from the right of the string and removed from the right of str1.

isspace()	Returns True if string contains only whitespace characters, otherwise returns False.
	Syntax:
	`str.isspace()`
		`>>> str1= "  "`
		`>>> print(str1.isspace())`
		`True`
		`>>> str1="  Python "`
		`>>> print(str1.isspace())`
		`False`

istitle()	The istitle() method doesn't take any arguments. It returns True if string is properly "titlecased", else returns False if the string is not a "titlecased" string or an empty string.
	Syntax:
	`str.istitle()`
		`>>> str1= "All Learn Python"`
		`>>> print(str1.istitle())`
		`True`
		`>>> s= "All learn Python"`
		`>>> print(s.istitle())`
		`False`
		`>>> s= "This Is @ Symbol"`
		`>>> print(s.istitle())`
		`True`
		`>>> s= "PYTHON"`
		`>>> print(s.istitle())`
		`False`

join(sequence)	Returns a string in which the string elements have been joined by a string separator.
	Syntax:
	`str.join(sequence)`
	sequence – Join() takes an argument which is of sequence data type capable of returning its elements one at a time. This method returns a string, which is the concatenation of each element of the string and the string separator between each element of the string.
		`>>> str1= "12345"`
		`>>> s= "-"`
		`>>> s.join(str1)`
		`'1-2-3-4-5'`
		`>>> str2= "abcd"`
		`>>> s= "#"`
		`>>> s.join(str2)`
		`'a#b#c#d'`

swapcase()	This method converts and returns all uppercase characters to lowercase and vice versa of the given string. It does not take any argument.
	Syntax:
	`str.swapcase()`
	The swapcase() method returns a string with all the cases changed.
		`>>> str1= "Welcome"`
		`>>> str1.swapcase()`
		`'wELCOME'`
		`>>> str2= "PYTHON"`
		`>>> str2.swapcase()`
		`'python'`
		`>>> s= "pYThoN"`
		`>>> s.swapcase()`
		`'PythOn'`

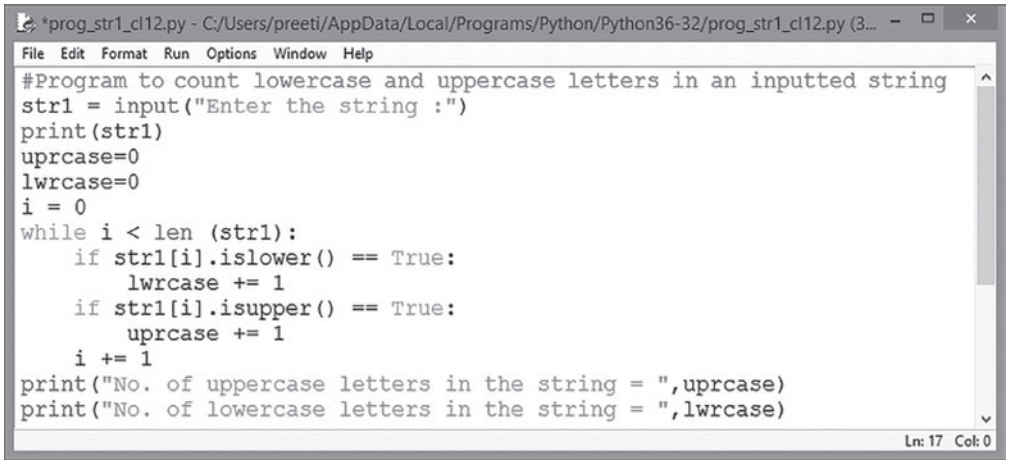
partition (Separator)	Partition method is used to split the given string using the specified separator and return a tuple with three parts: Substring before the separator; separator itself; a substring after the separator.
	Syntax:
	`str.partition(Separator)`
	Separator: This argument is required to separate a string.
	If the separator is not found, it returns the string itself, followed by two empty strings within the parentheses, as tuple.
		`>>> str3= "xyz@gmail.com"`
		`>>> str3.partition(' ')`
		`('xyz@gmail.com', '', '')`
		#Here separator is not found, returns the string itself, followed by two empty strings.
		
		`>>> str2= "Hardworkpays"`
		`>>> str2.partition('work')`
		`('Hard', 'work', 'pays')`
		#Here str2 is separated into three parts—
		1) the substring before the separator, *i.e.*, 'Hard'
		2) the separator itself, *i.e.*, 'work', and
		3) the substring part after the separator, *i.e.*, 'pays'.
		
		`>>> str5= str3.partition('@')`
		`>>> print(str5)`
		`('xyz', '@', 'gmail.com')`
		
		`>>> str4= str2.partition('–')`
		`>>> print(str4)`
		`('Hardworkpays', '', '')`

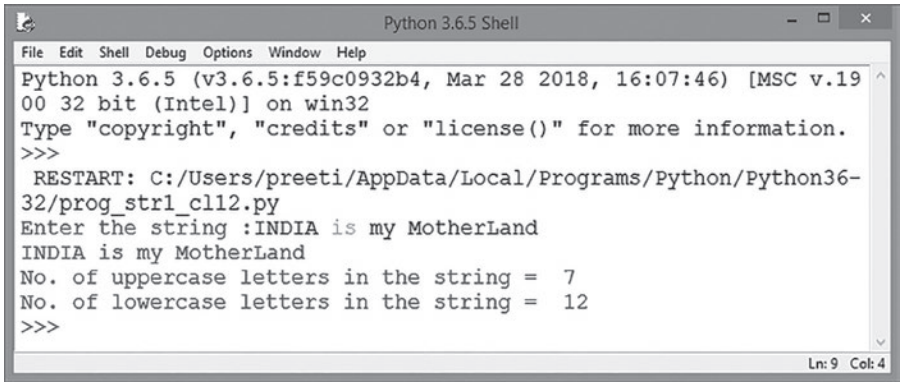
In internal storage or the memory of computer, the characters are stored in integer value. A specific value is used for a given character and it is based on ASCII code. There are different numbers assigned to capital letters and small letters. Python provides two functions for character encoding: ord() and chr().		

ord()	**ord()** – function returns the ASCII code of the character.	`>>> ch= 'b'`
		`>>> ord(ch)`
		`98`
		`>>> ord('A')`
		`65`

chr()	**chr()** – function returns character represented by the inputted ASCII number.	`>>> chr(97)`
		`'a'`
		`>>> chr(66)`
		`'B'`

Example 20: Program to input a string and count the number of uppercase and lowercase letters.





Explanation:

The above program counts the total number of lowercase and uppercase letters in an inputted string. The string can contain a mix of characters, numbers or any other special characters.

The inputted string is stored in the variable 'str 1'. Two variables are initialized to 0 for storing the number of uppercase and lowercase letters respectively. The loop shall iterate till the end of the string which is taken into account for calculating its length. For checking if a character is lowercase or uppercase, we have used two inbuilt methods, islower() & isupper() of the string library. If the character is in lowercase, the counter var 'lwrcase' shall be incremented and for uppercase, var 'uprcase' shall be incremented and finally the count shall be displayed using appropriate print() statements.

1.11 LISTS

Like strings, lists are a sequence of values. A list is a data type that can be used to store any type and number of variables and information. The values in the list are called elements or items or list members.

A list in Python is formed by enclosing the values inside square brackets ([ ]). **Unlike strings, lists are mutable,** *i.e.*, values in a list can be changed or modified and can be accessed using index value enclosed in square brackets.

Syntax:

```
[list_name] = [item1,item2,item3...,itemn]
```

For example, `L1 = [10, 20, 30, 40]`

Consider list1 containing 10 elements.

```
list1 = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

This list in computer memory shall be represented as shown in Fig. 1.6 below:

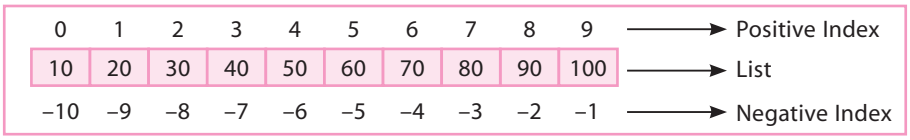
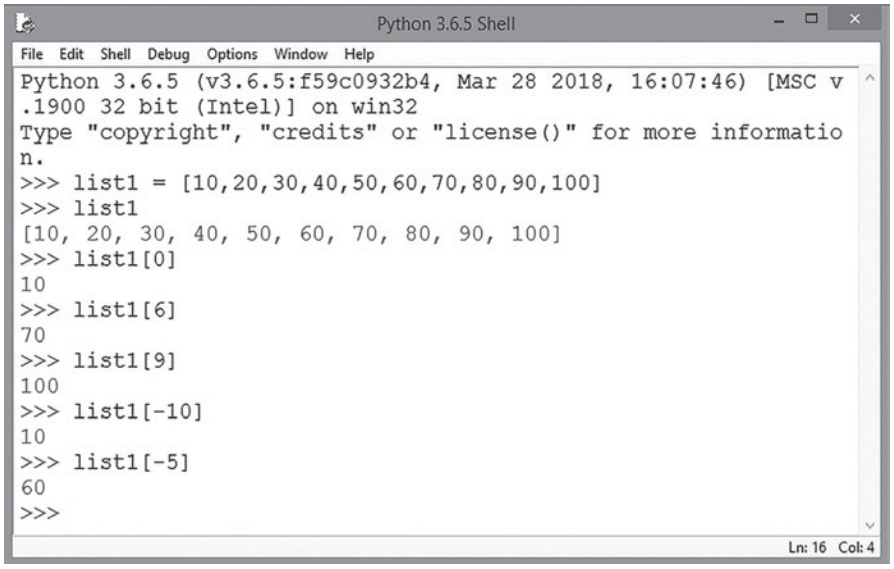


Fig. 1.6: List Representation in Python

On the basis of the index values, the elements in the list are accessed and retrieved:



1.11.1 List Comprehension

List comprehension is an elegant and concise way of creating a new list from an existing list in Python. List comprehension consists of an expression followed by 'for statement' inside square brackets.

Syntax:

new_list = [expression for item in list if conditional]

This is equivalent to:

for item in list:

if conditional:

expression:

For example,

```
>>>L1= [i * 1 for i in range(5) if i % 2 == 0]
>>>print(L1)
[0, 2, 4]
```

This corresponds to : L1=[]

`for i in range(5):` → iterate (it will iterate for 0,1,2,3,4)

`if i % 2 == 0:` → filter (then for each iteration, it will check condition which holds true for (0, 2 and 4)

`L1[i].append(i*i)` → transform (alter the value of i)

After transformation, each value of i will be added to list L1.

1.11.2 List Slicing

List slices are the sub-part of a list extracted out. List slices can be created using indexes. Slicing is used to retrieve a subset of values. A slice of a list is basically its sub-list. While indexing is used to obtain individual items, slicing allows us to obtain a subset of items. When we enter a range that we want to extract, it is called range slicing.

Syntax:

```
list[start: stop: step]
```

where:

start is the starting point

stop is the stopping point, which is not included

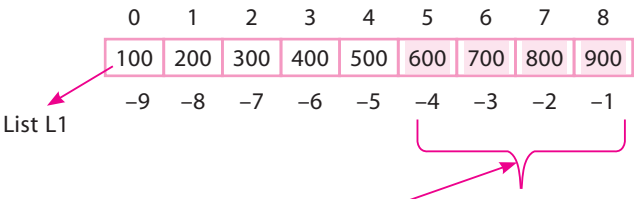
step is the step size, also known as stride, and is optional. Its default value is 1.

CTM: In list slice, if the start value is missing, then it will start from the 0th index; if stop is missing, then it will print till the end; if stop index is given, then it will stop at index stop – 1, *i.e.*, stop is excluded.

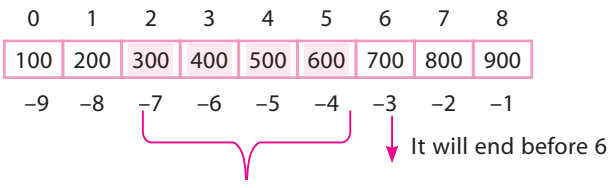
For example,

Consider the following list:

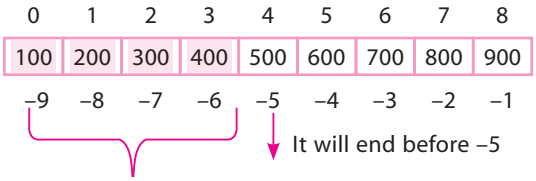
```
>>>L1=[100,200,300,400,500,600,700,800,900]
```




```
>>>L1[5:]
[600,700,800,900]
>>>L1[2:6]
```




```
[300,400,500,600]
>>>L1[-9:-5]
```




```
[100,200,300,400]
```

We can also reverse the values in list.

```
>>>L1[::-1]
```

Output:

```
[900,800,700,600,500,400,300,200,100]
>>>L1= ['MY LAB', [1,2,3], 'Y',(3,4,6),'TABLE',50]
>>>L1[2:3]
['Y']
>>>L1[1:2]
[[1, 2, 3]]
>>>L1[3][1]
4
```

1.11.3 Built-in List Functions and Methods

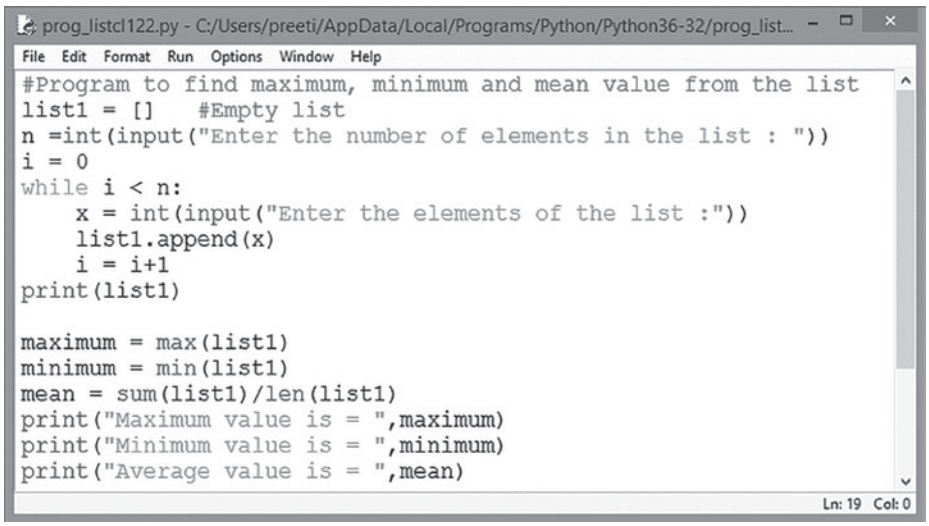
Python offers the following built-in functions which we have already discussed in detail in Class XI:

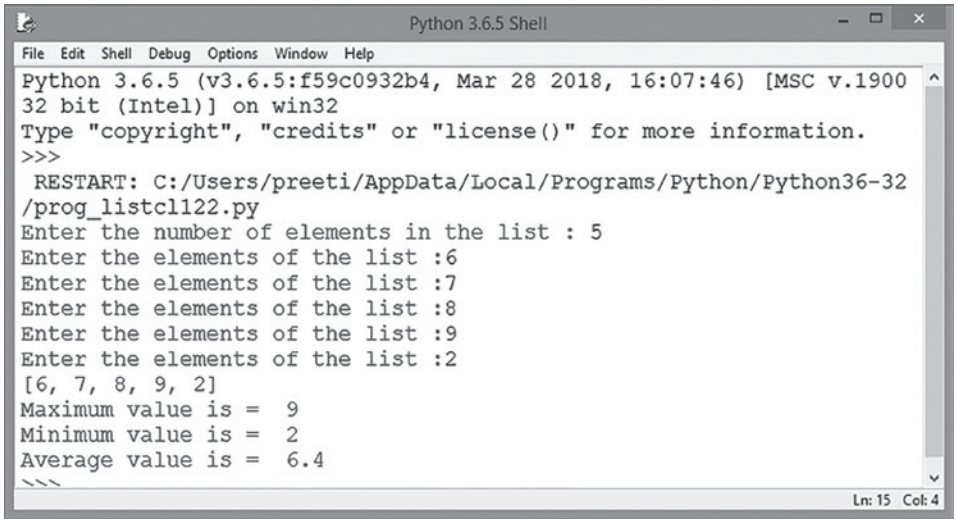
S.No.	Function	Description
1.	cmp(list1, list2)	Compares elements from both the lists
2.	len(list)	Returns the total length of the list
3.	max(list)	Returns the item with maximum value in the list
4.	min(list)	Returns the item with minimum value in the list
5.	list(seq)	Converts a tuple into list
6.	sum(list)	Sums up all the numeric values present in the list

Python also offers the following list methods:

Method	Result
`appends(`*`item`*`)`	Adds *item* to the end of the list.
`index(`*`item`*`)`	Returns the index of the first element whose value is equal to the item. A ValueError exception is raised if item is not found in the list.
`insert(`*`index, item`*`)`	Inserts *item* into the list at the specified *index*. When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position towards the end of the list. No exceptions will occur if we specify an invalid index. If we specify an index beyond the end of the list, the item will be added to the end of the list. If we use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
`sort()`	Sorts the items in the list so that they appear in ascending order (from the lowest value to the highest value).
`remove(`*`item`*`)`	Removes the first occurrence of item from the list. A ValueError exception is raised if item is not found in the list.
`reverse()`	Reverses the order of the items in the list.

Example 21: Program to find the maximum, minimum and mean value from the inputted list.





1.11.4 Copying Lists

Given a list, the simplest way to make a copy of the list is to assign it to another list.

```
>>> list1 = [1,2,3]
>>> list2 = list1
>>> list1
[1, 2, 3]
>>> list2
[1, 2, 3]
>>>
```

The statement list2 = list1 does not create a new list. Rather, it just makes list1 and list2 refers to the same list object. Here, list2 actually becomes an alias of list1. Therefore, any changes made to either of them will be reflected in the other list.

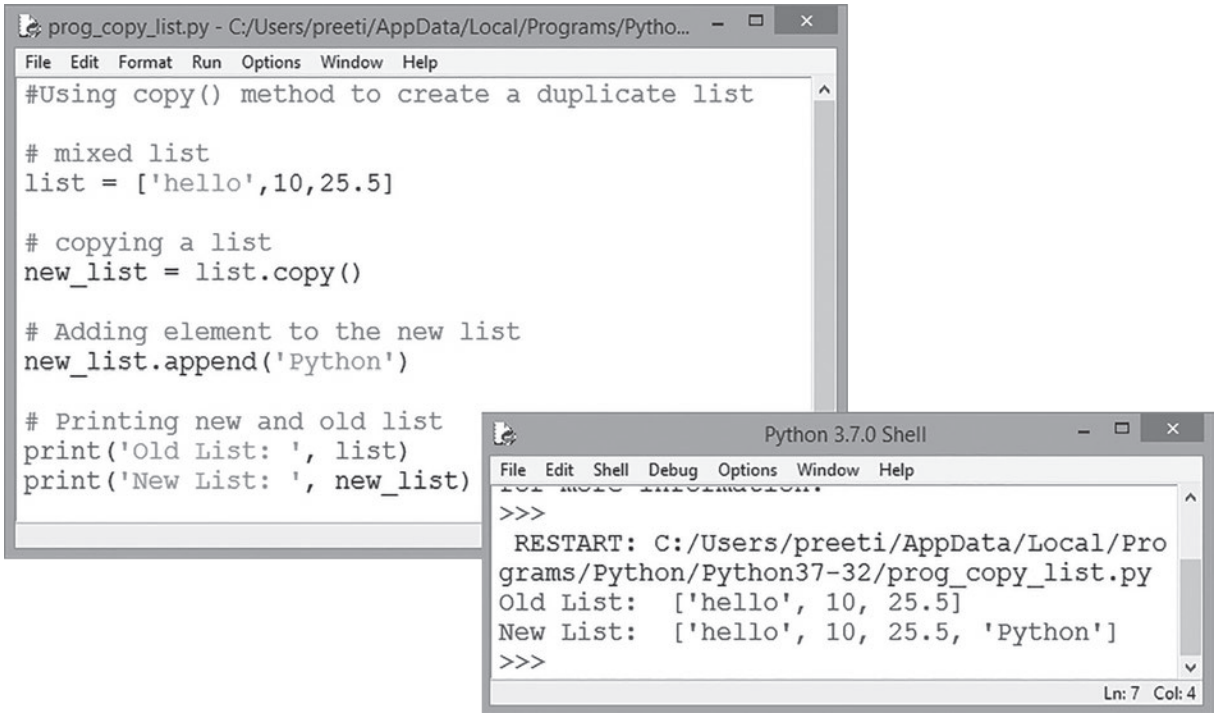
```
>>> list1.append(10)
>>> list1
[1, 2, 3, 10]
>>> list2
[1, 2, 3, 10]
>>>
```

We can also create a copy or clone of the list as a distinct object by three methods. The first method uses slicing, the second method uses built-in function list() and the third method uses copy() function of Python library copy.

For example, we can create a list using copy() method. This method creates a duplicate copy of the existing list but does not modify the original list. The syntax of copy() method is:

```
new_list = list.copy()
```

The copy() method doesn't take any parameters. However, it returns a list but doesn't modify the original list.



1.12 TUPLES

A tuple is a collection of Python objects separated by commas. In other words, **a tuple is a sequence of immutable Python objects**. The difference between lists and tuples is that the contents of tuples cannot be changed. Tuples are represented by parentheses ( ), whereas lists use square brackets [ ].

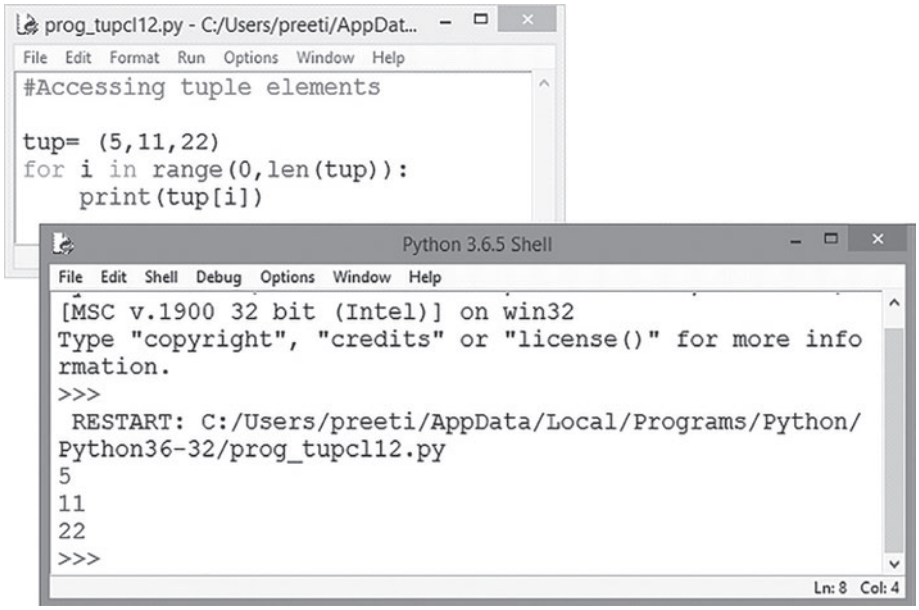
For example,

```
========== RESTART: Shell ==========
>>> tup = tuple()
>>> print(tup)   #an empty tuple
()
>>> tup1=("compsc","infopractices",2017,2018)
>>> tup2=(5,11,22,44,9,66)
>>> print("tup1[0]:",tup1[0])
tup1[0]: compsc
>>> print("tup2[1:5]:",tup2[1:5])
tup2[1:5]: (11, 22, 44, 9)
>>>
Ln: 19 Col: 4
```

1.12.1 Iterating Through a Tuple

Elements of the tuple can be accessed sequentially using loop.

For example, tuple elements can be accessed using for loop with range() function.



CTM: Tuple indices start at 0.

Like lists, tuples also work well with basic operations as shown in the table given below:

Python Expression	Results	Description
`len((1, 2))`	`2`	Length
`(1, 2) + (4, 5)`	`(1, 2, 4, 5)`	Concatenation
`('CS',) * 2`	`('CS', 'CS')`	Repetition
`5 in (1, 2, 3)`	`False`	Membership
`for x in (4,2,3) : print (x, end = ' ')`	`4 2 3`	Iteration

1.12.2 Tuple Slicing

Slicing is used to retrieve a subset of values. A slice of a tuple is basically its sub-tuple.

Syntax:

```
tuple_name[start: stop: step]
```

where,

start is the starting point

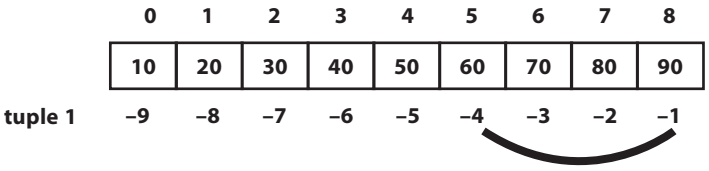
stop is the stopping point

step is the step size, also known as stride

If we omit the first index, slice starts from the 0^{th} index value and omission of stop will take it to the end. Default value of step is 1. However, it includes the first element but excludes the last element.

The first value in a tuple is at index 0. Just like lists, we can use the index values in combination with square brackets [] to access items in a tuple:

For example,




```
========== RESTART: Shell ==========
>>> tuple1 = (10,20,30,40,50,60,70,80,90)
>>> tuple1[5:]
(60, 70, 80, 90)
>>>
```

Ln: 13 Col: 4

Python includes the following tuple functions:

S.No.	Function	Description
1.	`cmp(tuple1, tuple2)`	Compares elements from both the tuples.
2.	`len(tuple)`	Returns the total length of the tuple.
3.	`max(tuple)`	Returns the item from the tuple with the maximum value.
4.	`min(tuple)`	Returns the item from the tuple with the minimum value.
5.	`tuple(seq)`	Converts a list into tuple.

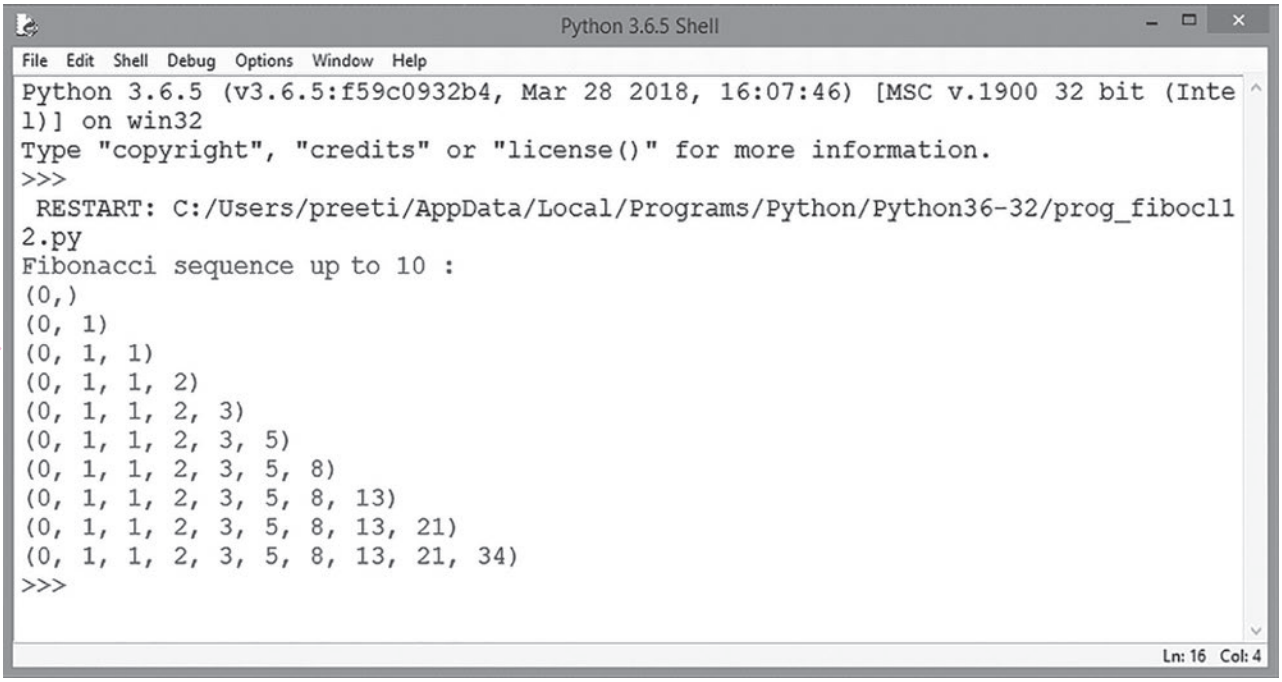
Example 22: Program to generate Fibonacci series in a tuple.

prog_fibocl12.py - C:/Users/preeti/AppData/Local/Programs/Python/Python36-32/prog_fibocl12.py (3.6.5)

File Edit Format Run Options Window Help

```
#Fibonacci series in tuple.
nterms= 10
n1 = 0
n2 = 1
count = 0
tup=()
# check if the number of terms is valid
if nterms<= 0:
    print("Please enter a positive integer")
elif nterms== 1:
    print("Fibonacci sequence up to",nterms,":")
    print(n1)
else:
    print("Fibonacci sequence up to",nterms,":")
while count < nterms:
    tup=tup+(n1,)
    nth = n1 + n2
# update values
    n1 = n2
    n2 = nth
    count += 1
    print(tup)
```

Ln: 26 Col: 4



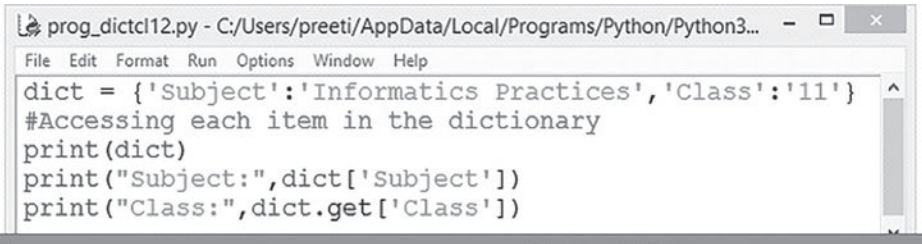
1.13 DICTIONARY

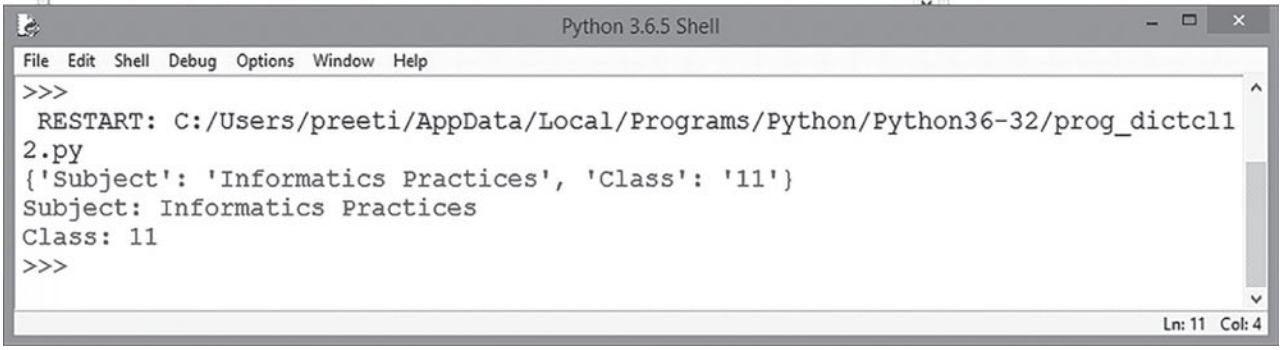
A dictionary is like a list except that in a list we have to access it using an index, whereas items in a dictionary can be accessed using a unique key, which can be a number, string or a tuple. The items in a dictionary can be changed but keys are an immutable data type. Each key is separated from its value by a colon (:), the items are separated by commas, and the entire elements (key-value pair) are enclosed in curly braces { }.

Syntax:

```
<dictionary_name> = {'key1': 'value1','key2': 'value2','key3': 'value3'…
                     'keyn': 'valuen'}
```

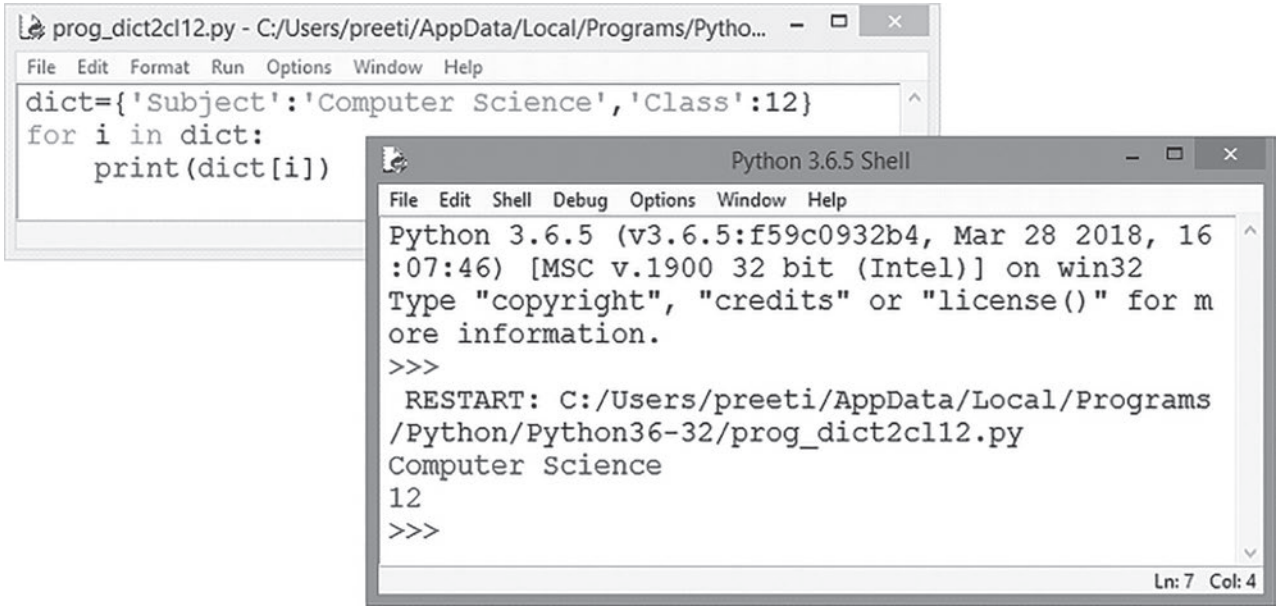
For example, creating and accessing the elements of dictionary, dict.





1.13.1 Iterating Through a Dictionary

The following example will show how dictionary items can be accessed through for loop.



1.13.2 Updating Dictionary Elements

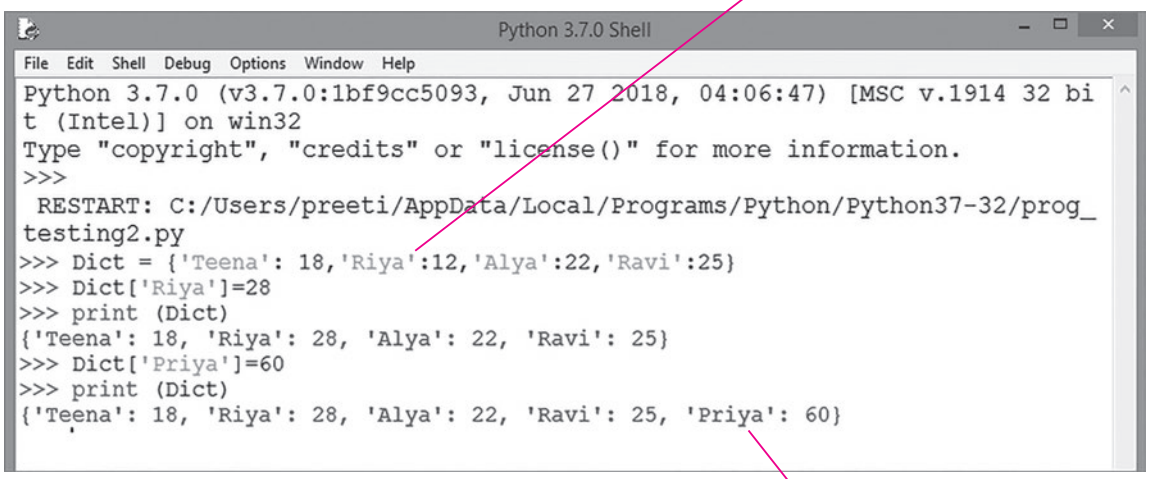
You can also update a dictionary by modifying an existing key-value pair or by merging another dictionary with an existing one.

Syntax:

```
<Dictionary>[<key>]=<value>
```

For example,

If key is there in the dictionary then it will update the value of that particular key in the dictionary.



If key is not there in the dictionary then it will add key-value pair in the dictionary.

1.13.3 Built-in Dictionary Functions

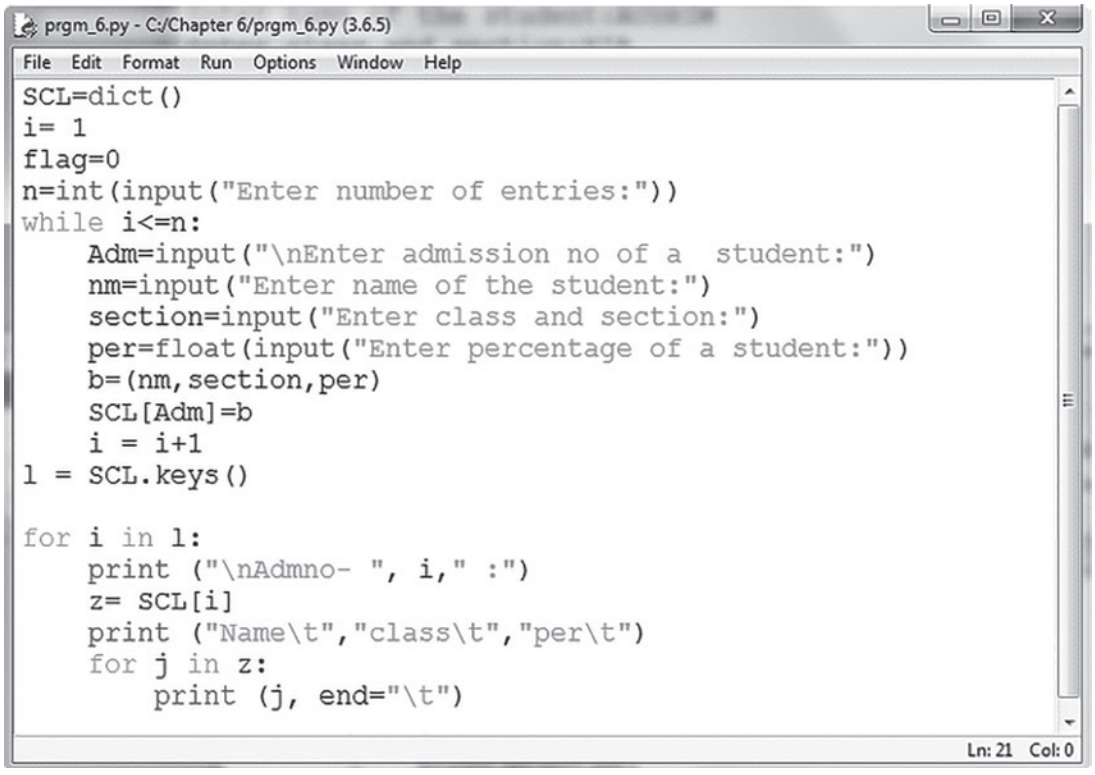
Python offers the following Dictionary functions:

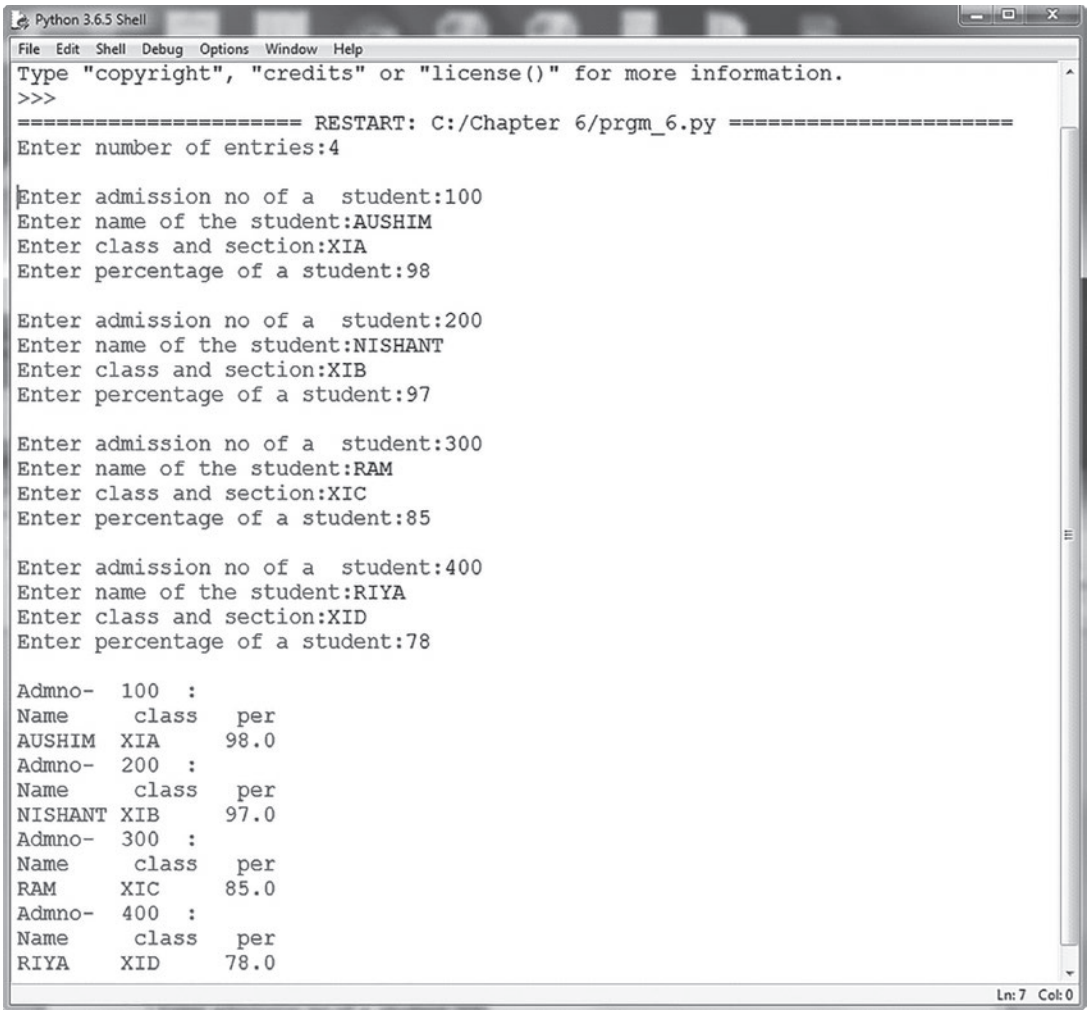
S.No.	Function	Description
1.	`cmp(dict1, dict2)`	Compares elements from both the dictionaries.
2.	`len(dict)`	Returns the total number of items present in the dictionary.
3.	`str(dict)`	Produces a printable string representation of a dictionary.
4.	`type(variable)`	Returns the type of the variable passed as an argument.

Python offers the following built-in Dictionary methods:

S.No.	Method & Description
1.	dict.clear() Removes all elements of dictionary dict
2.	dict.copy() Returns a shallow copy of dictionary dict
3.	dict.items() Returns a list of dict's (key, value) tuple pairs
4.	dict.keys() Returns a list of dictionary dict's keys
5.	dict.setdefault(key, default = None) Similar to get(), but will set dict[key] = default if key is not already in dict
6.	dict.update(dict2) Adds dictionary dict2's key-value pairs to dict
7.	dict.values() Returns a list of dictionary dict's values

Example 23: Program to store students' information like admission number, roll number, name and marks in a dictionary, and display information on the basis of admission number.





Explanation:

The above program fetches the name, section, percentage of a student on the basis of the admission no. Here, admission no. shall act as the key to the student dictionary. keys() returns all the keys present in the dictionary which will be used to display the records of all the students on the basis of their admission no. (Adm) and are printed using for loop.

1.14 SORTING TECHNIQUES

Sorting is to arrange the list in an ascending or descending order. Sorting algorithm specifies the way to arrange data in a particular order.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. There are various sorting techniques that can be used to sort the list in ascending or descending order. Here we will discuss two techniques:

1. Bubble Sort
2. Insertion Sort

1.14.1 Bubble Sort

Bubble Sort is one of the simple sorting techniques where it traverses the whole list by comparing its adjacent elements, sorting them and swapping the elements until the whole list is sorted.

Algorithm

1 START

2 read the array elements and store it in an array a[]

3 store the length in a variable n.

4 for i=0 to n–1 repeat steps 5 and 6

5 for j=0 to n–i–1 repeat step 4

6 if a[j] > a[j+1] then swap elements

7 display the sorted list

8 END

Steps:

1. Bubble sort algorithm starts by comparing the first 2 elements in the list. If and only if the first element is greater than the second, these elements will be swapped. This ensures that the greater of the first 2 elements is in the second position. In the next step, the second element is compared with the third element, and following the same logic, they are swapped if and only if the second element is greater than the third element. With these 2 steps, it is ensured that the greatest of the first 3 elements is in the 3rd position. If this process is repeated until the last element, the greatest element will end up in the last position in the list. This completes the first step in Bubble Sort.
2. After step 1 gets completed, the largest element in array will get placed in the last position of the list.
3. If there are n elements to be sorted, then the process mentioned above should be repeated n–1 times to get the required result.
4. For performance optimization, in the second step, last element and the second last element are not compared since this comparison was already done in the previous step, and the last element is already present in its correct position.
5. Similarly, in step 3, second last and third last elements are not compared and so on.

Let's see it with the help of an example:

Sorting in ascending order using bubble sort.

Suppose list is [42, 29, 74, 11, 65, 58]

First pass

[42, 29, 74, 11, 65, 58] ⟶ [29, 42, 74, 11, 65, 58] **it will swap since 29<42**

[29, 42, 74, 11, 65, 58] ⟶ [29, 42, 74, 11, 65, 58] no swapping as 42<74

[29, 42, 74, 11, 65, 58] ⟶ [29, 42, 11, 74, 65, 58] it will swap since 11<74

[29, 42, 11, 74, 65, 58] ⟶ [29, 42, 11, 65, 74, 58] it will swap since 65<74

[29, 42, 11, 65, 74, 58] ⟶ [29, 42, 11, 65, 58, 74] it will swap since 58<74

The largest element is settled at its correct place *i.e.*, at the end. So the first five elements form the unsorted part while the last element forms the sorted part and will be compared for the next pass.

Second pass

[29, 42, 11, 65, 58, 74] → [29, 42, 11, 65, 58, 74] no swapping as 29<42

[29, 42, 11, 65, 58, 74] → [29, 11, 42, 65, 58, 74] it will swap since 11<42

[29, 11, 42, 65, 58, 74] → [29, 11, 42, 65, 58, 74] no swapping as 65>42

[29, 11, 42, 65, 58, 74] → [29, 11, 42, 58, 65, 74] it will swap since 58<65

The next largest element is settled at its correct place. So the first four elements form the unsorted part while the last two elements form the sorted part and will be compared for the next pass.

Third pass

[29, 11, 42, 58, 65, 74] → [11, 29, 42, 58, 65, 74] it will swap since 29>11

[11, 29, 42, 58, 65, 74] → [11, 29, 42, 58, 65, 74] no swapping as 29<42

[11, 29, 42, 58, 65, 74] → [11, 29, 42, 58, 65, 74] no swapping as 42<58

The next largest element is settled at its correct place. So the first three elements form the unsorted part while the last three elements form the sorted part.

Fourth pass

[11, 29, 42, 58, 65, 74] → [11, 29, 42, 58, 65, 74] no swapping as 29>11

[11, 29, 42, 58, 65, 74] → [11, 29, 42, 58, 65, 74] no swapping as 42>29

The next largest element is settled at its correct place. So the first two elements form the unsorted part while the last four elements form the sorted part.

Fifth pass

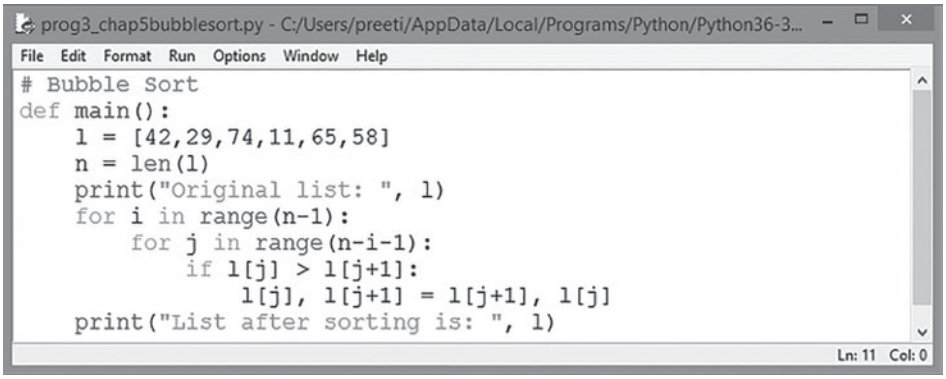
[11, 29, 42, 58, 65, 74] → [11, 29, 42, 58, 65, 74] no swapping as 11<29

The next largest element is settled at its correct place. And finally the list is sorted.

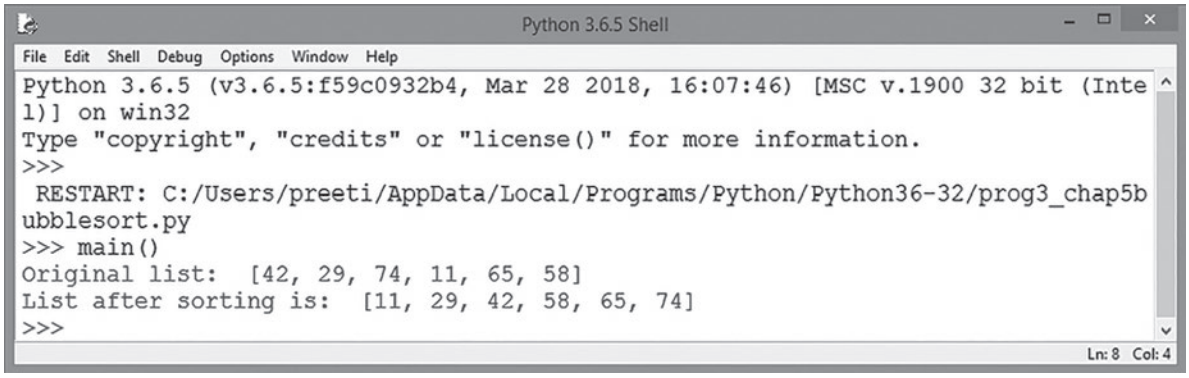
Here, total number of items, say, n is 6, so outer loop will be executed n–1 times, *i.e.*, 5 times. There will be 5 passes through the list.

For each pass or outer loop iteration, it compares adjacent items and swaps those that are not in order. Each pass places the next largest value in its correct place and there is no need to compare these elements as they are already placed at their correct place.

Code:



**********Output of the program**********



1.14.2 Insertion Sort

Consider you have report cards of 10 students and you have to sort these in ascending order according to roll number.

Well, you will have to go through each report card from the starting or the back, and find its right position by comparing its roll number with the roll number on every other report card. Once you find the right position, you will insert the report card there.

Similarly, if more new report cards are provided to you, you can easily repeat the same process and insert the new report cards and get the report cards sorted too.

This is exactly how insertion sort works. It starts from the index 1(not 0), and each index starting from index 1 is like a new report card, that you have to place at the proper position in the sorted sub-array on the left.

Algorithm for insertion sort

Step 1: START

Step 2: Enter the elements in an array A, store the length of array in N.

Step 3: For i=1 to N, repeat steps 4 to 8, else step 8

Step 4: J=A.index(i)

Step 5: While (A[j–1]>A[j] and j>0), repeat steps 6 to 7, else step 3

Step 6: A[j-1], A[j] =A[j], A[j–1]

Step 7: j=j-1

Step 8: print the list

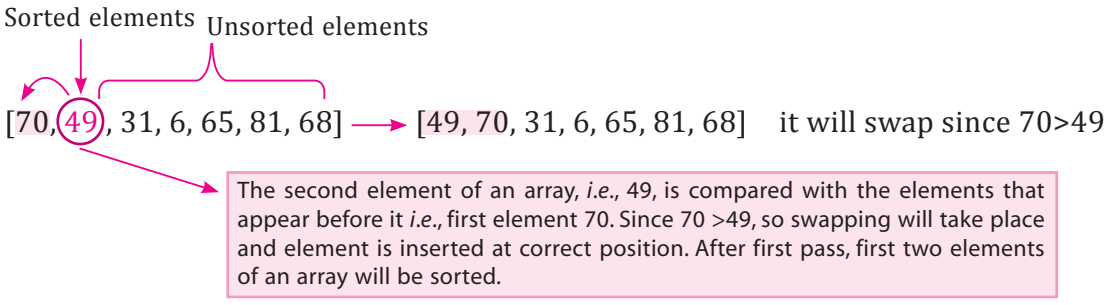
Step 9: END

Let's see it with the help of an example:

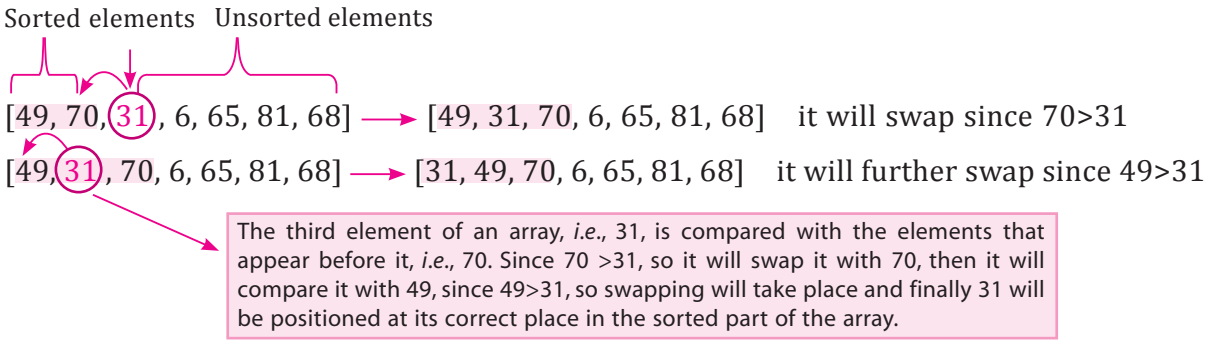
Sorting in ascending order using insertion sort.

Suppose list is [70,49,31,6,65,81,68]

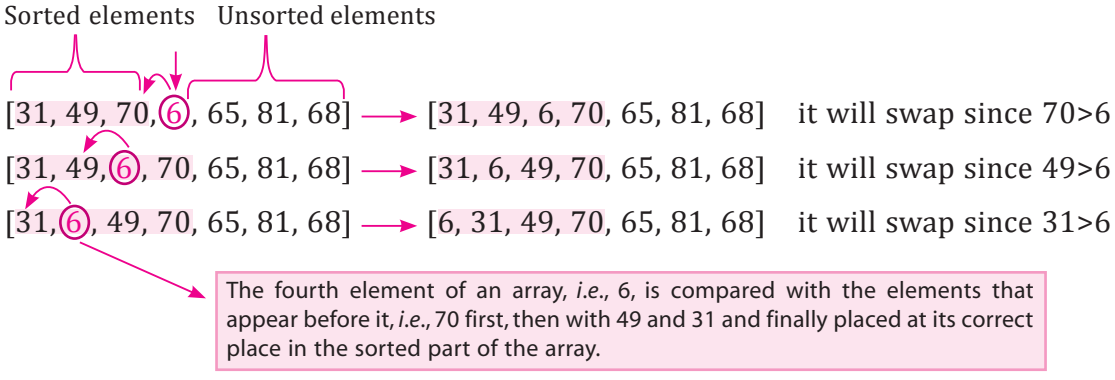
First pass



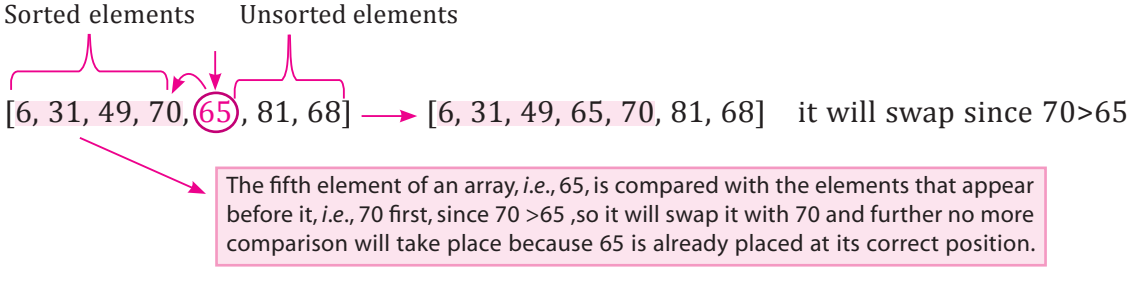
Second pass



Third pass



Fourth pass



Fifth pass

Sorted elements Unsorted elements

[6, 31, 49, 65, 70, 81, 68] → [6, 31, 49, 65, 70, 81, 68] no swapping will take place since 70<81

The sixth element of an array, *i.e.*, 81, is compared with the elements that appear before it, *i.e.*, 70 first. Since 70 <81, so it will not swap as this is already placed at its correct position.

Sixth pass

Sorted elements Unsorted elements

[6, 31, 49, 65, 70, 81, 68] → [6, 31, 49, 65, 70, 68, 81] it will swap since 81>68

[6, 31, 49, 65, 70, 68, 81] → [6, 31, 49, 65, 68, 70, 81] it will swap since 70>68

The seventh element of an array, *i.e.*, 68, is compared with the elements that appear before it, *i.e.*, 81 first and then with 70 and gets its correct position in the sorted array.

Now the list is sorted. The total number of items (*say n*) is 7, so outer loop will be executed n–1 times, *i.e.*, 6 times. There will be 6 passes through the list.

For each pass, it not only compares adjacent items and swaps, but also slides up each larger element until it gets to the correct location in the sorted part of the array.

Code:

prog4_chap5insertion.py - C:/Users/preeti/AppData/Local/Progr...

File Edit Format Run Options Window Help

```
# Insertion Sort
a = [70,49,31,6,65,81,68]
print("Original list : ", a)
for i in a:
    j = a.index(i)
    while j>0:
        if a[j-1] > a[j]:
            a[j-1], a[j] = a[j], a[j-1]
        else:
            break
        j = j-1
print("List after sorting : ",a)
```

Ln: 13 Col: 0

**********Output of the program*********

Python 3.6.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900
 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python36-32
/prog4_chap5insertion.py
Original list :  [70, 49, 31, 6, 65, 81, 68]
List after sorting :  [6, 31, 49, 65, 68, 70, 81]
>>>
```

Ln: 7 Col: 4

Steps:

1. Compare the current element in the iteration (say A) with the previous adjacent element to it. If it is in order, then continue the iteration, step 5 else, go to step 2.
2. Swap the two elements (the current element in the iteration (A) and the previous adjacent element to it).
3. Compare A with its new previous adjacent element. If they are not in order, then proceed to step 4.
4. Swap if they are not in order and repeat steps 3 and 4.
5. Continue the iteration.

Application:insertion sort

Insertion sort is efficient for small data sets but quite inefficient for large lists. Insertion sort is adaptive; this means that it reduces its total number of steps if a partially-sorted array is provided as input, which eventually increases its overall efficiency. It does not require additional memory.

Application:bubble sort

It is comparatively simpler to understand than some other sorting algorithms. Bubble sort is the fastest and the easiest sorting method available. It is the fastest sorting method for an extremely small and/or nearly sorted set of data.

Efficiency of any sorting algorithm is measured by counting number of algorithmic operations.

MEMORY BYTES

- A complete set of instructions written using a programming language is termed as a Program/Code/Program Code.
- Python is a powerful and flexible programming language. It uses concise and easy-to-learn syntax which enables programmers to write more codes and develop more complex programs in a much shorter time.
- It is a platform-independent programming language.
- Python interpreter executes one statement (command) at a time.
- Python provides two different ways to work with—Interactive mode and Script mode.
- Interactive mode does not save commands in the form of a program and the output is placed between commands as it is displayed as soon as we press the Enter key after typing one or more commands.
- Interactive mode is suitable for testing code.
- Python is a case-sensitive language. Thus, Ram and ram are two different names in Python.
- Python is an interpreted language.
- Python's interactive interpreter is also called Python Shell.
- print() statement outputs an entire (complete) line and then goes to the next line for subsequent output(s).
- A Python program can contain various components like expressions, statements, comments, functions, blocks and indentation.
- An expression is a legal combination of symbols that represents a value.
- A statement is a programming instruction.
- Comments are non-executable, additional information added in a program for readability.
- In Python, comments begin with the hash (#) symbol/character.
- A variable in Python is defined only when some value is assigned to it.

- Python supports dynamic typing, *i.e.*, a variable can hold values of different types at different times.
- A function is a named block of statements that can be invoked by its name.
- The input() function evaluates the data input and takes the result as numeric type.
- The if statement is a decision-making statement.
- The looping constructs while and for statements allow sections of code to be executed repeatedly.
- for statement iterates over a range of values or a sequence.
- The statements within the body of a while loop are executed over and over again until the condition of the while becomes false or remains true.
- A string is a sequence of characters.
- We can create strings simply by enclosing characters in quotes (single, double or triple).
- Positive subscript helps in accessing the string from the beginning.
- Negative subscript helps in accessing the string from the end.
- '+' operator joins or concatenates the strings on both sides of the operator.
- The * operator creates a new string concatenating multiple copies of the same string.
- List is a sequence data type.
- A list is a mutable sequence of values which can be of any type and are indexed by integer.
- A list is created by placing all the items (elements) inside a square bracket [], separated by commas.
- A list can even have another list as an item. This is called nested list.
- Another way of creating tuple is built-in function list().
- Traversing a list means accessing each element of a list. This can be done by using looping statement, either for or while.
- List slicing allows us to obtain a subset of items.
- copy() creates a list from another list. It does not take any parameter.
- A tuple is an immutable sequence of values which can be of any type and are indexed by an integer.
- Creating a tuple is as simple as putting values separated by a comma. Tuples can be created using parentheses ().
- To create a tuple with a single element, the final comma is necessary.
- Python provides various operators like '+','*','in', 'not in', etc., which can be applied to tuples.
- In a dictionary, each key maps a value. The association of a key and a value is called a key-value pair.
- To create a dictionary, key-value pairs are separated by a comma and are enclosed in two curly braces {}. In key-value pair, each key is separated from its value by a colon (:).
- We can add new elements to an existing dictionary, extend it with single pair of values or join two dictionaries into one.
- We can also update a dictionary by modifying an existing key-value pair or by merging another dictionary with an existing one.
- Python provides us with a number of dictionary methods like: len(), pop(), items(), keys(), values(), get(), etc.
- keys() method in Python dictionary returns an object that displays a list of all the keys in the dictionary.
- Bubble sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.
- Insertion sort is an in-place sorting algorithm.
- In Insertion sort, an element gets compared and inserted into the correct position in the list.

OBJECTIVE TYPE QUESTIONS

1. Fill in the blanks.

(a) ..... is the Python operator responsible for declaring variables.

(b) The built-in function randrange() belongs to ..... module.

(c) A ..... operator does not directly operate on data but produces a left-to-right evaluation of expression.

(d) median() method belongs to ..... module in Python.

(e) The reserved words in Python are called ..... and these cannot be used as names or identifiers.

(f) An ..... is a symbol used to perform an action on some value.

(g) A file that contains a collection of related functions and other definitions is called ..... .

(h) The modules in Python have the ..... extension.

(i) A ..... is just a module that contains some useful definitions.

(j) Each object in Python has three key attributes—a ....., a ..... and an ..... .

(k) In Python, the non-zero value is treated as ..... and zero value is treated as ..... .

(l) Keys of a dictionary must be ..... .

(m) In ....., the adjoining values in a sequence are compared and exchanged repeatedly until the entire array is sorted.

(n) Logical operators are used to combine two or more ..... expressions.

(o) The ..... function returns the length of a specified list.

Answers: (a) Assignment (=) operator (b) random (c) comma (,)
(d) statistics (e) keywords (f) operator
(g) module (h) .py (i) library
(j) type, value, id (k) true, false (l) unique
(m) Bubble sort (n) relational (o) len()

2. State whether the following statements are True or False.

(a) The two statements x = int(22.0/7) and x = int(22/7.0) yield the same results.

(b) The given statement: x + 1 = x is a valid statement.

(c) List slice is a list in itself.

(d) Relational operators return either true or false.

(e) break, continue, pass are the three conditional statements.

(f) The % (modulus) operator cannot be used with the float data type.

(g) The range() function is used to specify the length of a for-in loop.

(h) Assignment operator can be used in place of equality operator in the test condition.

(i) Comments in Python begin with a "$" symbol.

(j) In print() function, if you use a concatenate operator (+) between two strings, both the strings are joined with a space in between them.

(k) If we execute Python code using prompt ">>>" then we call it an interactive interpreter.

(l) Lists are immutable while strings are mutable.

(m) The keys of a dictionary must be of immutable types.

(n) Lists and strings in Python support two-way indexing.

(o) Tuples can be nested and can contain other compound objects like lists, dictionaries and other tuples.

Answers: (a) True (b) False (c) True (d) True (e) False (f) True
(g) True (h) False (i) False (j) False (k) True (l) False
(m) True (n) True (o) True

3. Multiple Choice Questions (MCQs)

(a) Which of the following is not considered a valid identifier in Python?
(i) two2 (ii) _main (iii) hello_rsp1 (iv) 2 hundred

(b) What will be the output of the following code— print("100+200")?
(i) 300 (ii) 100200 (iii) 100+200 (iv) 200

(c) Which amongst the following is a mutable datatype in Python?

(i) int (ii) string (iii) tuple (iv) list

(d) pow() function belongs to which library?

(i) math (ii) string (iii) random (iv) maths

(e) Which of the following statements converts a tuple into a list?

(i) len(string) (ii) list(string) (iii) tup(list) (iv) dict(string)

(f) The statement: bval = str1 > str2 shall return ..... as the output if two strings str1 and str2 contains "Delhi" and "New Delhi".

(i) True (ii) Delhi (iii) New Delhi (iv) False

(g) What will be the output generated by the following snippet?

```
a = [5,10,15,20,25]
k = 1
i = a[1] + 1
j = a[2] + 1
m = a[k+1]
print(i,j,m)
```

(i) 11 15 16 (ii) 11 16 15 (iii) 11 15 15 (iv) 16 11 15

(h) The process of arranging the array elements in a specified order is termed as:

(i) Indexing (ii) Slicing (iii) Sorting (iv) Traversing

(i) Which of the following Python functions is used to iterate over a sequence of numbers by specifying a numeric end value within its parameters?

(i) range() (ii) len() (iii) substring() (iv) random()

(j) What is the output of the following?

```
d = {0: 'a', 1: 'b', 2: 'c'}
for i in d:
  print(i)
```

(i)
```
0
1
2
```

(ii)
```
a
b
c
```

(iii)
```
0
a
1
b
2
c
```

(iv)
```
2
a
2
b
2
c
```

(k) What is the output of the following?

```
x = 123
for i in x:
  print(i)
```

(i) 1 2 3 (ii) 123 (iii) error (iv) infinite loop

(l) Which arithmetic operators cannot be used with strings?

(i) + (ii) * (iii) – (iv) All of the above

(m) What will be the output when the following code is executed?

```
>>>str1="helloworld"
>>>str1[:-1]
```

(i) dlrowolleh (ii) hello (iii) world (iv) helloworld

(n) What is the output of the following statement?

```
print("xyyzxyzxzxyy".count('yy', 1))
```

(i) 2 (ii) 0 (iii) 1 (iv) Error

(o) Suppose list1 = [0.5 * x for x in range(0, 4)], list1 is:

(i) [0, 1, 2, 3] (ii) [0, 1, 2, 3, 4]

(iii) [0.0, 0.5, 1.0, 1.5] (iv) [0.0, 0.5, 1.0, 1.5, 2.0]

(p) Which is the correct form of declaration of dictionary?

(i) Day={1:'monday',2:'tuesday',3:'wednesday'}

(ii) Day=(1;'monday',2;'tuesday',3;'wednesday')

(iii) Day=[1:'monday',2:'tuesday',3:'wednesday']

(iv) Day={1'monday',2'tuesday',3'wednesday']

(q) Identify the valid declaration of L: `L = [1, 23, 'hi', 6]`

(i) list (ii) dictionary (iii) array (iv) tuple

Answers: (a) (iv) (b) (iii) (c) (iv) (d) (i) (e) (ii) (f) (iv)
(g) (ii) (h) (iii) (i) (i) (j) (i) (k) (iii) (l) (iii)
(m) (i) (n) (i) (o) (iii) (p) (i) (q) (i)

SOLVED QUESTIONS

1. What is Python?

Ans. Python is a high-level, interpreted, dynamic, Object-Oriented Programming Language that supports GUI programming.

2. Why is Python interpreted?

Ans. Python is interpreted because the program is processed at runtime by the interpreter and we do not need to compile the program before executing it.

3. Who developed Python?

Ans. Python was developed by Guido van Rossum in the early nineties at the National Research Institute for Mathematics in the Netherlands.

4. Why is Python easy to learn?

Ans. Python has relatively few keywords, simple structure and a clearly defined syntax. This allows the student to understand and work easily in a relatively short period of time.

5. Write any one feature of Python library.

Ans. Python library is very portable and cross-platform compatible with UNIX, Windows and Macintosh.

6. Is Python a compiler language or an interpreter language? [HOTS]

Ans. The normal execution of Python program is interpreted. However, subsets of the language can be compiled.

7. State some distinguishable features of Python.

Ans. Python is a modern, powerful, interpreted language with objects, modules, exceptions (or interrupts) and automatic memory management.

Salient Features of Python are:

- **Simple and Easy:** Python is a simple language that is easy to learn.
- **Free/Open source:** Anybody can use Python without the need to purchase a licence.
- **High-level Language:** Being a high-level language, it can be easily understood by the user without the need to be concerned with low-level details.
- **Portable:** Python codes are machine and platform-independent.
- **Extensible and Embeddable:** Python programs support usage of C/C++ codes.
- **Standard Library:** Python standard library contains pre-written tools for programming.

8. Distinguish between Java and Python.

Ans. Java and Python can be distinguished on the following bases:

1. Python programs run slower than Java codes, but Python saves much time and space. Python programs are 3 to 5 times smaller than Java programs.

2. Python is a loosely-typed dynamic language because no time gets wasted in declaring variable types as in Java.
3. Python is easier to learn than Java.

9. What is the difference between interactive mode and script mode in Python?

Ans. In interactive mode, instructions are given in front of Python prompt (>>>) in Python Shell. Python carries out the given instructions and shows the result there itself.

In script mode, Python instructions are stored in a file, generally with .py extension, and executed together in one go as a unit. The saved instructions are known as Python script or Python program.

10. Differentiate between mutable and immutable objects in Python language with example.

Ans. Every variable in Python holds an instance of an object. There are two types of objects in Python, *i.e.*, Mutable and Immutable objects. Whenever an object is instantiated, it is assigned a unique object id. The type of the object is defined at the runtime and it can't be changed afterwards.

However, its state can be changed if it is a mutable object.

For example, int, float, bool, string, unicode, tuple are immutable objects in Python. In simple words, an immutable object can't be changed after it is created. Lists, dictionaries and sets are mutable types.

11. What will be the value and its type for the given expression:

3.25 + 4?

Ans.
```
>>>result = 3.25 + 4
>>>print(result, 'is', type(result))
7.25 is <class 'float'>
```

Its type is a float as integers are automatically converted into floats as necessary.

12. What will the following command display?

```
print("fractional string to int:", int("3.4"))
```

State the reason for getting the output and what is required to be done?

Ans. The above statement, upon execution, shall result in an error since Python int command cannot convert the string "3.4" into 3.4 as Python cannot perform two consecutive typecasts, therefore, you must convert it explicitly in code. This statement is to be modified such as:

int(float("3.4")) which shall generate the output as 3.

13. Consider the following statements in Python interpreter and describe the output/statement required:

(a) Print the message "Hello World".

(b)
```
a = 10
b = 12
c = a + b
print(c)
```

(c) To retrieve the data type of the inputted string "Hello" stored inside the variable 'a'.

(d) To describe the data type of variable 'b'.

(e) To retrieve the address of variables a and b.

(f) State the output:
```
d = b
d
b
id(d)
id(b)
```

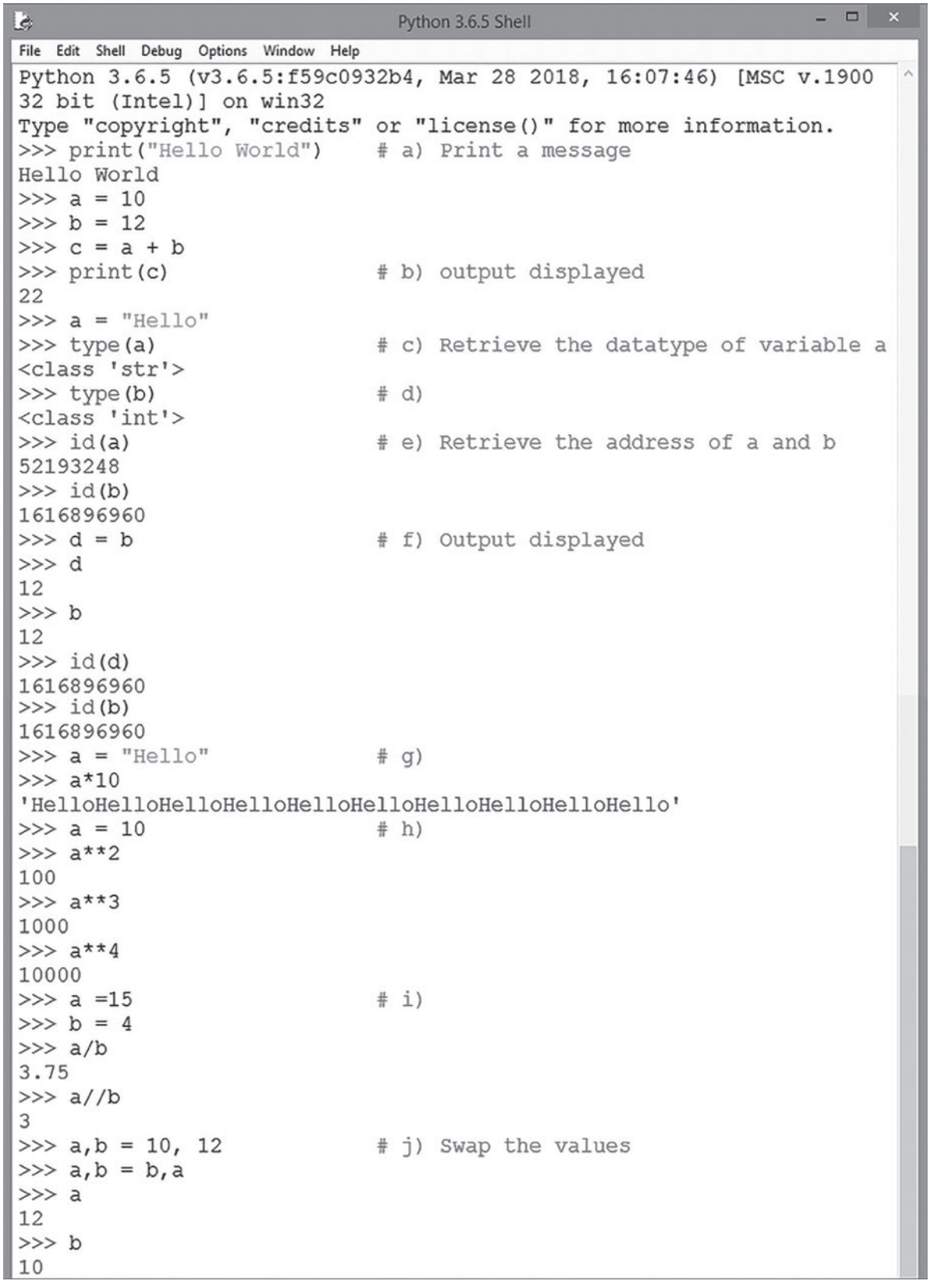
(g)
```
a = "Hello"
a * 10
```

(h) To display the value for a^2, a^3 and a^4

(i) a = 15
b = 4
a/b
a//b

(j) To swap the values of two variables, a and b, using multiple assignment statements.

Ans.



14. How many types of strings are supported in Python?

Ans. Python allows two string types:

1. Single line Strings—Strings that are terminated in a single line.
2. Multi-line Strings—Strings storing multiple lines of text.

15. Convert the following for loop into while loop:

```
for k in range(10,20,5):
print(k)
```

Ans.

```
k=10
while(k<20):
    print(k)
    k+=5
```

16. Find errors in the following code (if any) and correct the code by rewriting it and underlining the corrections:

```
x= int("Enter value for x:"))
for in range[0,11]:
    if x = y
        print x+y
    else:
        Print x-y
```

Ans.

```
x=int(input("Enter value for x:"))
for y in range(0,11):
    if x == y:
        print(x+y)
    else:
        print(x-y)
```

17. Write the output of the following code when executed:

```
Text="gmail@com"
l=len(Text)
ntext=""
for i in range (0,l):
    if Text[i].isupper():
        ntext=ntext+Text[i].lower()
    elif Text[i].isalpha():
        ntext=ntext+Text[i].upper()
    else:
        ntext=ntext+'bb'
print(ntext)
```

Ans. GMAILbbCOM

18. How many times is the word 'HELLO' printed in the following statement?

```
s='python rocks'
for ch in s[3:8]:
    print('Hello')
```

Ans. 5 times

19. Find the output of the following:

```
word = 'green vegetables'
print(word.find('g', 2))
print(word.find('veg', 2))
print(word.find('tab', 4, 15))
```

Ans. Output:

```
8
6
10
```

20. Suppose `L=["abc",[6,7,8], 3,"mouse"]`

Consider the above list and answer the following:

(a) `L[3:]` (b) `L[::2]`

(c) `L[1:2]` (d) `L[1][1]`

Ans. (a) `['mouse']` (b) `['abc', 3]`

(c) `[[6, 7, 8]]` (d) `7`

21. What is a tuple?

Ans. A tuple is an immutable sequence of values which can be of any type and are indexed by an integer.

22. Differentiate between lists and tuples.

Ans. Tuples cannot be changed unlike lists; tuples use parentheses, whereas lists use square brackets.

23. Tuple is an ordered immutable sequence of objects. Justify your answer.

Ans. Tuple is an ordered sequence of objects as each element is accessed by its index and it is immutable as we cannot change the values in place. We cannot update or edit the tuple.

24. What is a dictionary?

Ans. A Python dictionary is a mapping of unique keys to values. It is a collection of key-value pairs. Dictionaries are mutable which means that they can be changed.

25. Find errors, underline them and rewrite the same after correcting the following code:

```
d1=dict[]
i= 1
n=input("Enter number of entries:")
while i<=n:
    a=input("Enter name:")
    b=input("Enter age:")
    d1(a)=b
    i = i+1
    l = d1.key[]
    for i in l:
        print(i,'\t','d1[i]')
```

Ans.

```
d1=dict()
i=1
num=int(input("Enter the no:"))
for i in range(num):
    a = input("Enter name:")
    b = input("Enter age:")
    d1[a] = b
    l = d1.keys()
    i = i + 1
    for i in l:
        print(i,"\t",d1[i])
```

26. Write a Python program to calculate the length of a string.

Ans.

```
def string_length(str1):
    count = 0
    for char in str1:
        count += 1
return count
print(string_length('Python Program'))
```

27. Write a Python program to count the number of characters (character frequency) in a string.

Sample String : 'google.com'

Expected Result : `{'g': 2, 'o': 3, 'l': 1, 'e': 1, '.': 1, 'c': 1, 'm': 1}`

Ans.
```
def char_frequency(str1):
    dict = {}
    for n in str1:
        keys = dict.keys()
        if n in keys:
            dict[n] += 1
        else:
            dict[n] = 1
    return dict
print(char_frequency('google.com'))
```

28. Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.

Sample String : 'restart'

Expected Result : `'resta$t'`

Ans.
```
def change_char(str1):
    char = str1[0]
    str1 = str1.replace(char, '$')
    str1 = char + str1[1:]
    return str1
print(change_char('restart'))
```

29. Write a Python program to remove the n^{th} index character from a nonempty string.

Ans.
```
def remove_char(str, n):
    first_part = str[:n]
    last_part = str[n+1:]
    return first_part + last_part
print(remove_char('Python', 0))
print(remove_char('Python', 3))
print(remove_char('Python', 5))
```

30. Write a Python program that accepts a comma separated sequence of words as input and prints the unique words in sorted form (alphanumerically).

Sample Words : red, white, black, red, green, black

Expected Result : `black, green, red, white, red`

Ans.
```
items = input("Input comma separated sequence of words:")
words = [word for word in items.split(",")]
print(",".join(sorted(list(set(words)))))
```

31. Write a Python function to get a string made of its first three characters of a specified string. If the length of the string is less than 3, then return the original string.

Sample function and result:

first_three('ipy') -> ipy

first_three('python') -> pyt

Ans.
```
def first_three(str):
    return str[:3] if len(str) > 3 else str
print(first_three('ipy'))
print(first_three('python'))
print(first_three('py'))
```

32. Write a Python program to check whether a string starts with specified characters.

Note: In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. *For example*, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence.

Ans.
```
def caesar_encrypt(realText, step):
    outText = []
    cryptText = []
    uppercase = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J','K', 'L',
                 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
                 'Y', 'Z']
    lowercase = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
                 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
                 'w', 'x', 'y', 'z']
    for eachLetter in realText:
        if eachLetter in uppercase:
            index = uppercase.index(eachLetter)
            crypting = (index + step) % 26
            cryptText.append(crypting)
            newLetter = uppercase[crypting]
            outText.append(newLetter)
        elif eachLetter in lowercase:
            index = lowercase.index(eachLetter)
            crypting = (index + step) % 26
            cryptText.append(crypting)
            newLetter = lowercase[crypting]
            outText.append(newLetter)
    return outText
code = caesar_encrypt('abc', 2)
print()
print(code)
print()
```

33. Write a Python program to print the following floating numbers with no decimal places.

Ans.
```
x = 3.1415926
y = -12.9999
print("\nOriginal Number: ", x)
print("Formatted Number with no decimal places: "+"{:.0f}".format(x));
print("Original Number: ", y)
print("Formatted Number with no decimal places: "+"{:.0f}".format(y));
print()
```

34. Which of the following will return the result as a floating point number 2.0?

```
first = 1.0
second = "1"
third = "1.1"
```

1. `first + float(second)`
2. `float(second) + float(third)`
3. `first + int(third)`
4. `first + int(float(third))`
5. `int(first) + int(float(third))`
6. `2.0 * second`

Ans. statements 1 and 4

35. Write a Python program to print the index of the character in a string.

Sample string: Python Program

Expected output:

```
Current character P position at 0
Current character y position at 1
Current character t position at 2
```

Ans.

```
str1 = input('Enter String:')
for index, char in enumerate(str1):
    print("Current character", char, "position at", index )
```

36. Write a Python program to count and display the vowels of a given text.

Ans.

```
def vowel(text):
    vowels = "aeiouAEIOU"
    print(len([letter for letter in text if letter in vowels]))
    print([letter for letter in text if letter in vowels])
vowel('Welcome')
```

Output:

```
3
['e', 'o', 'e']
```

37. Write a Python program to sum all the items in a list.

Ans.

```
def sum_list(items):
    sum_numbers = 0
    for x in items:
        sum_numbers += x
    return sum_numbers
print(sum_list([1,2,-8]))
```

38. What will be the status of the following list after the First, Second and Third pass of the insertion sort method used for arranging the following elements in ascending order?

Note : Show the status of all the elements after each pass very clearly underlining the changes. 16, 19, 11, 15, 10

Ans. First Pass

No Swapping

16	19	11	15	10
16	19	11	15	10

Second Pass

16	19	11	15	10
16	19	11	15	10
11	16	19	15	10

Third Pass

11	16	19	15	10
11	16	19	15	10
11	15	16	19	10

39. What will be the status of the following list after the First, Second and Third pass of the insertion sort method used for arranging the following elements in descending order?

22, 24, 64,34, 80, 43

Note : Show the status of all the elements after each pass and circle the changes.

Ans.

	22	24	–64	34	80	43
Pass 1	(24)	22	–64	34	80	43
Pass 2	24	22	(–64)	34	80	43
Pass 3	(34)	24	22	–64	80	43

40. Write a Python program to get the largest number from a list.

Ans.
```
def max_num_in_list(list):
    max = list[0]
    for a in list:
        if a > max:
            max = a
    return max
print(max_num_in_list([1, 2, -8, 0]))
```

41. Write a Python program to remove duplicates from a list.

a = [10,20,30,20,10,50,60,40,80,50,40]

Ans.
```
a = [10,20,30,20,10,50,60,40,80,50,40]
dup_items = set()
uniq_items = []
for x in a:
    if x not in dup_items:
        uniq_items.append(x)
        dup_items.add(x)
print(dup_items)
```

42. Write a Python function that takes two lists and returns True if they have at least one common member.

Ans.
```
def common_data(list1, list2):
    result = False
    for x in list1:
        for y in list2:
            if x == y:
                result = True
    return result
print(common_data([1,2,3,4,5], [5,6,7,8,9]))
print(common_data([1,2,3,4,5], [6,7,8,9]))
```

43. Write a Python program to shuffle and print a specified list.

Ans.
```
from random import shuffle
color = ['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow']
shuffle(color)
print(color)
```

44. Write a Python program to count the number of elements in a list within a specified range.

Ans.
```
def count_range_in_list(li, min, max):
    ctr = 0
    for x in li:
        if min <= x <= max:
            ctr += 1
    return ctr
list1 = [10,20,30,40,40,40,70,80,99]
print(count_range_in_list(list1, 40, 100))
list2 = ['a','b','c','d','e','f']
print(count_range_in_list(list2, 'a', 'e'))
```

45. Write a Python program to generate groups of five consecutive numbers in a list.

Ans.
```
l = [[5*i + j for j in range(1,6)] for i in range(5)]
print(l)
```

46. Write a Python program to replace the last element in a list with another list.
Sample data : [1, 3, 5, 7, 9, 10], [2, 4, 6, 8]
Expected Output: `[1, 3, 5, 7, 9, 2, 4, 6, 8]`

Ans.
```
num1 = [1, 3, 5, 7, 9, 10]
num2 = [2, 4, 6, 8]
num1[-1:] = num2
print(num1)
```

47. Write a Python program to create a dictionary from two lists without losing duplicate values.
Sample data : ['Class-V', 'Class-VI', 'Class-VII', 'Class-VIII'], [1, 2, 2, 3]
Expected Output: `defaultdict(<class 'set'>, {'Class-V':{1}, 'Class-VI':{2}, 'Class-VII':{2}, 'Class-VIII':{3}})`

Ans.
```
from collections import defaultdict
class_list = ['Class-V', 'Class-VI', 'Class-VII', 'Class-VIII']
id_list = [1, 2, 2, 3]
temp = defaultdict(set)
for c, i in zip(class_list, id_list):
    temp[c].add(i)
print(temp)
```

48. Write a Python program to iterate over dictionaries using for loops.

Ans.
```
d = {'x': 10, 'y': 20, 'z': 30}
for dict_key, dict_value in d.items():
    print(dict_key,'->',dict_value)
```

49. Consider the following unsorted list: 95 79 19 43 52 3. Write the passes of bubble sort for sorting the list in ascending order till the 3rd iteration.

Ans. [79, 19, 43, 52, 3, 95]
[19, 43, 52, 3, 79, 95]
[19, 43, 3, 52, 79, 95]

50. Which of the following is valid arithmetic operator in Python: [CBSE Sample Paper 2019]
(a) //
(b) ?
(c) <
(d) and

Ans. (a) //

51. Write the type of tokens from the following:
(a) if
(b) roll_no

Ans. (a) Key word
(b) Identifier

52. Rewrite the following code in Python after removing all syntax error(s). Underline each correction done in the code.
```
30=To
for K in range(0,To)
IF k%4==0:
print(K*4)
Else:
    print(K+3)
```

Ans.
```
To=30
for K in range(0,To):
    if K%4==0:
        print(K*4)
    else:
        print(K+3)
```

53. Find and write the output of the following Python code:
```
def fun(s):
    k=len(s)
    m=" "
for i in range(0,k):
    if(s[i].isupper()):
        m=m+s[i].lower()
    elif s[i].isalpha():
        m=m+s[i].upper()
    else:
        m=m+'bb'
print(m)
fun('school2@com')
```

Ans. SCHOOLbbbbCOM

54. Find and write the output of the following Python code:
```
def Change(P,Q=30):
    P=P+Q
    Q=P-Q
    print(P,"#",Q)
    return P
R=150
S=100
R=Change(R,S)
print(R,"#",S)
S=Change(S)
```

Ans.
```
250 # 150
250 # 100
130 # 100
```

55. Write a recursive function in Python BinarySearch(Arr,l,R,X) to search the given element X to be searched from the List Arr having R elements where l represents lower bound and R represents upper bound.

Ans.
```
def BinarySearch(Arr,l,R,X):
    if R >= l:
        mid = l + (R-l)//2
        if Arr[mid] == X:
            return mid
        elif Arr[mid] > X:
            return BinarySearch(Arr,l,mid-1,X)
        else:
            return BinarySearch(Arr,mid+1,r,X)
    else:
            return -1 Arr = [ 2, 3, 4, 10, 40 ]
X = int(input('Enter element to be searched:'))
result = BinarySearch(Arr,0,len(Arr)-1,X)
if result != -1:
    print("Element is present at index ", result)
else:
    print("Element is not present in array")
```

UNSOLVED QUESTIONS

1. What are the advantages of Python programming language?
2. In how many different ways can you work in Python?
3. What are the advantages/disadvantages of working in interactive mode in Python?
4. Write Python statement for the following in interactive mode:
   (a) To display sum of 3, 8.0, 6*12
   (b) To print sum of 16, 5.0, 44.0.
5. What are operators? Give examples of some unary and binary operators.
6. What is an expression and a statement?
7. What all components can a Python program contain?
8. What are variables? How are they important for a program?
9. Write the output of the following:

(i)
```
for i in '123':
    print("guru99",i,)
```

(ii)
```
for i in [100,200,300]:
    print(i)
```

(iii)
```
for j in range(10,6,-2):
    print(j*2)
```

(iv)
```
for x in range(1,6):
    for y in range(1,x+1):
        print(x,' ',y)
```

(v)
```
for x in range(10,20):
    if (x == 15):
        break
print(x)
```

(vi)
```
for x in range (10,20):
    if (x % 2 == 0) :
        continue
print(x)
```

10. Write the output of the following program on execution if x = 50:

```
if x>10:
    if x>25:
        print("ok")
    if x>60:
        print("good")
elif x>40:
    print("average")
else:
    print("no output")
```

11. What are the various ways of creating a list?
12. What are the similarities between strings and lists?
13. Why are lists called a mutable data type?
14. What is the difference between insert() and append() methods of a list?
15. Write a program to calculate the mean of a given list of numbers.
16. Write a program to calculate the minimum element of a given list of numbers.

17. Write a code to calculate and display total marks and percentage of a student from a given list storing the marks of a student.

18. Write a Program to multiply an element by 2 if it is an odd index for a given list containing both numbers and strings.

19. Write a Program to count the frequency of an element in a given list.

20. Write a Program to shift elements of a list so that the first element moves to the second index and second index moves to the third index, and so on, and the last element shifts to the first position.

Suppose the list is [10,20,30,40]

After shifting, it should look like: [20,30,40,10]

21. A list Num contains the following elements:

3, 25, 13, 6, 35, 8, 14, 45

Write a function to swap the content with the next value divisible by 5 so that the resultant list will look like:

25, 3, 13, 35, 6, 8, 45, 14

22. Write a program to accept values from a user in a tuple. Add a tuple to it and display its elements one by one. Also display its maximum and minimum value.

23. Write a program to input any values for two tuples. Print it, interchange it and then compare them.

24. Write a Python program to input 'n' classes and names of their class teachers to store them in a dictionary and display the same. Also accept a particular class from the user and display the name of the class teacher of that class.

25. Write a program to store student names and their percentage in a dictionary and delete a particular student name from the dictionary. Also display the dictionary after deletion.

26. Write a Python program to input names of 'n' customers and their details like items bought, cost and phone number, etc., store them in a dictionary and display all the details in a tabular form.

27. Write a Python program to capitalize first and last letters of each word of a given string.

28. Write a Python program to remove duplicate characters of a given string.

29. Write a Python program to compute sum of digits of a given string.

30. Write a Python program to find the second most repeated word in a given string.

31. Write a Python program to change a given string to a new string where the first and last chars have been exchanged.

32. Write a Python program to multiply all the items in a list.

33. Write a Python program to get the smallest number from a list.

34. Write a Python program to append a list to the second list.

35. Write a Python program to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30 (both included).

36. Write a Python program to get unique values from a list.

37. Write a Python program to convert a string to a list.

38. Write a Python script to concatenate the following dictionaries to create a new one:

```
d1 = {'A':1, 'B':2, 'C':3}
d2 = {'D':4}
```

Output should be:

```
{'A':1, 'B':2, 'C':3, 'D':4}
```

39. Write a Python script to check if a given key already exists in a dictionary.

40. Write a Python script to print a dictionary where the keys are numbers between 1 and 15 (both included) and the values are square of keys.

Sample Dictionary

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11: 121, 12: 144, 13: 169, 14: 196, 15: 225}

41. Write a Python script to merge two Python dictionaries.

42. Write a Python program to sort a dictionary by key.

43. Write a Python program to combine two dictionary adding values for common keys.

```
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
```

Sample output: `Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})`

44. Write a Python program to find the highest 3 values in a dictionary.

45. Write a Python program to sort a list alphabetically in a dictionary.

46. Write a Python program to count number of items in a dictionary value that is a list.

47. Consider the following unsorted list: 105, 99, 10, 43, 62, 8. Write the passes of bubble sort for sorting the list in ascending order till the 3rd iteration.

48. What will be the status of the following list after the First, Second and Third pass of the insertion sort method used for arranging the following elements in descending order?

28, 44, 97, 34, 50, 87

Note: Show the status of all the elements after each pass very clearly underlining the changes.