Chapter 4

Introduction to Problem Solving

4.1 Introduction

"Computer Science is a science of abstraction -creating the right model for a problem and devising the appropriate mechanizable techniques to solve it."

–A. Aho and J. Ullman

Today, computers are all around us. We use them for doing various tasks in a faster and more accurate manner. For example, using a computer or smartphone, we can book train tickets online.

India is a big country and we have an enormous railway network. Thus, railway reservation is a complex task. Making reservation involves information about many aspects, such as details of trains (train type, types of berth and compartments in each train, their schedule, etc.), simultaneous booking of tickets by multiple users and many other related factors.

It is only due to the use of computers that today, the booking of the train tickets has become easy. Online booking of train tickets has added to our comfort by enabling us to book tickets from anywhere, anytime.

We usually use the term computerisation to indicate the use of computer to develop software in order to automate any routine human task efficiently. Computers are used for solving various day-to-day problems and thus problem solving is an essential skill that a computer science student should know. It is pertinent to mention that computers themselves cannot solve a problem. Precise step-by-step instructions should be given by us to solve the problem. Thus, the success of a computer in solving a problem depends on how correctly and precisely we define the problem, design a solution (algorithm) and implement the solution (program) using a programming language. Thus, problem solving is the process of identifying a problem, developing an algorithm for the identified problem and finally implementing the algorithm to develop a computer program.

In this chapter

GIGO (Garbage In Garbage Out)
The correctness of the output that a computer gives depends upon the correctness of input provided.

4.2 STEPS FOR PROBLEM SOLVING

Suppose while driving, a vehicle starts making a strange noise. We might not know how to solve the problem right away. First, we need to identify from where the noise is coming? In case the problem cannot be solved by us, then we need to take the vehicle to a mechanic. The mechanic will analyse the problem to identify the source of the noise, make a plan about the work to be done and finally repair the vehicle in order to remove the noise. From the above example, it is explicit that, finding the solution to a problem might consist of multiple steps.

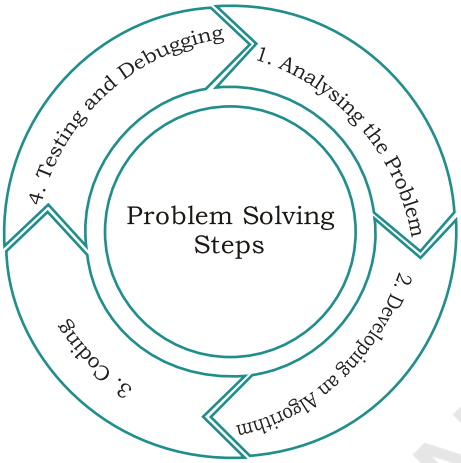


Figure 4.1: Steps for problem solving

When problems are straightforward and easy, we can easily find the solution. But a complex problem requires a methodical approach to find the right solution. In other words, we have to apply problem solving techniques. Problem solving begins with the precise identification of the problem and ends with a complete working solution in terms of a program or software. Key steps required for solving a problem using a computer are shown in Figure 4.1 and are discussed in following subsections.

4.2.1 Analysing the problem

It is important to clearly understand a problem before we begin to find the solution for it. If we are not clear as to what is to be solved, we may end up developing a program which may not solve our purpose. Thus, we need to read and analyse the problem statement carefully in order to list the principal components of the problem and decide the core functionalities that our solution should have. By analysing a problem, we would be able to figure out what are the inputs that our program should accept and the outputs that it should produce.

Algorithm
A set of exact steps which when followed, solve the problem or accomplish the required task.

4.2.2 Developing an Algorithm

It is essential to device a solution before writing a program code for a given problem. The solution is represented in natural language and is called an algorithm. We can imagine an algorithm like a very well-written recipe for

a dish, with clearly defined steps that, if followed, one will end up preparing the dish.

We start with a tentative solution plan and keep on refining the algorithm until the algorithm is able to capture all the aspects of the desired solution. For a given problem, more than one algorithm is possible and we have to select the most suitable solution. The algorithm is discussed in section 4.3.

4.2.3 Coding

After finalising the algorithm, we need to convert the algorithm into the format which can be understood by the computer to generate the desired solution. Different high level programming languages can be used for writing a program.

It is equally important to record the details of the coding procedures followed and document the solution. This is helpful when revisiting the programs at a later stage. Coding is explained in detail in section 4.8.

4.2.4 Testing and Debugging

The program created should be tested on various parameters. The program should meet the requirements of the user. It must respond within the expected time. It should generate correct output for all possible inputs. In the presence of syntactical errors, no output will be obtained. In case the output generated is incorrect, then the program should be checked for logical errors, if any.

Software industry follows standardised testing methods like unit or component testing, integration testing, system testing, and acceptance testing while developing complex applications. This is to ensure that the software meets all the business and technical requirements and works as expected. The errors or defects found in the testing phases are debugged or rectified and the program is again tested. This continues till all the errors are removed from the program.

Once the software application has been developed, tested and delivered to the user, still problems in terms of functioning can come up and need to be resolved from time to time. The maintenance of the solution, thus, involves fixing the problems faced by the user,

Activity 4.1

What sequence of steps will you follow to compute the LCM of two numbers?

answering the queries of the user and even serving the request for addition or modification of features.

4.3 ALGORITHM

In our day-to-day life we perform activities by following certain sequence of steps. Examples of activities include getting ready for school, making breakfast, riding a bicycle, wearing a tie, solving a puzzle and so on. To complete each activity, we follow a sequence of steps. Suppose following are the steps required for an activity 'riding a bicycle':

1) remove the bicycle from the stand,
2) sit on the seat of the bicycle,
3) start peddling,
4) use breaks whenever needed and
5) stop on reaching the destination.

Let us now find Greatest Common Divisor (GCD) of two numbers 45 and 56.

Note: GCD is the largest number that divides both the given numbers.

Step 1: Find the numbers (divisors) which can divide the given numbers

```
Divisors of 45 are: 1, 3, 5, 9, 15, and 45
Divisors of 54 are: 1, 2, 3, 6, 9, 18, 27, and 54
```

Step 2: Then find the largest common number from these two lists.

```
Therefore, GCD of 45 and 54 is 9
```

Hence, it is clear that we need to follow a sequence of steps to accomplish the task. Such a finite sequence of steps required to get the desired output is called an algorithm. It will lead to the desired result in a finite amount of time, if followed correctly. Algorithm has a definite beginning and a definite end, and consists of a finite number of steps.

The origin of the term Algorithm is traced to Persian astronomer and mathematician, Abu Abdullah Muhammad ibn Musa Al-Khwarizmi (c. 850 AD) as the Latin translation of Al-Khwarizmi was called 'Algorithmi'.

4.3.1 Why do we need an Algorithm?

A programmer writes a program to instruct the computer to do certain tasks as desired. The computer then follows the steps written in the program code. Therefore, the programmer first prepares a roadmap of the program to be written, before actually writing the code. Without

a roadmap, the programmer may not be able to clearly visualise the instructions to be written and may end up developing a program which may not work as expected.

Such a roadmap is nothing but the algorithm which is the building block of a computer program. For example, searching using a search engine, sending a message, finding a word in a document, booking a taxi through an app, performing online banking, playing computer games, all are based on algorithms.

Writing an algorithm is mostly considered as a first step to programming. Once we have an algorithm to solve a problem, we can write the computer program for giving instructions to the computer in high level language. If the algorithm is correct, computer will run the program correctly, every time. So, the purpose of using an algorithm is to increase the reliability, accuracy and efficiency of obtaining solutions.

(A) Characteristics of a good algorithm

- Precision — the steps are precisely stated or defined.
- Uniqueness — results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
- Finiteness — the algorithm always stops after a finite number of steps.
- Input — the algorithm receives some input.
- Output — the algorithm produces some output.

(B) While writing an algorithm, it is required to clearly identify the following:

- The input to be taken from the user
- Processing or computation to be performed to get the desired result
- The output desired by the user

4.4 Representation of Algorithms

Using their algorithmic thinking skills, the software designers or programmers analyse the problem and identify the logical steps that need to be followed to reach a solution. Once the steps are identified, the need is to

write down these steps along with the required input and desired output. There are two common methods of representing an algorithm —flowchart and pseudocode. Either of the methods can be used to represent an algorithm while keeping in mind the following:

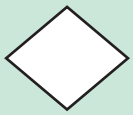
- it showcases the logic of the problem solution, excluding any implementational details
- it clearly reveals the flow of control during execution of the program

4.4.1 Flowchart — Visual Representation of Algorithms

A flowchart is a visual representation of an algorithm. A flowchart is a diagram made up of boxes, diamonds and other shapes, connected by arrows. Each shape represents a step of the solution process and the arrow represents the order or link among the steps.

There are standardised symbols to draw flowcharts. Some are given in Table 4.1.

Table 4.1 Shapes or symbols to draw flow charts

Flowchart symbol	Function	Description
	Start/End	Also called "Terminator" symbol. It indicates where the flow starts and ends.
	Process	Also called "Action Symbol," it represents a process, action, or a single step.
	Decision	A decision or branching point, usually a yes/no or true/false question is asked, and based on the answer, the path gets split into two branches.
	Input/Output	Also called data symbol, this parallelogram shape is used to input or output data
	Arrow	Connector to show order of flow between shapes.

Example 4.1: Write an algorithm to find the square of a number.

Before developing the algorithm, let us first identify the input, process and output:

- Input: Number whose square is required
- Process: Multiply the number by itself to get its square
- Output: Square of the number

Algorithm to find square of a number.

Step 1: Input a number and store it to num
Step 2: Compute num * num and store it in square
Step 3: Print square

The algorithm to find square of a number can be represented pictorially using flowchart as shown in Figure 4.2.

Think and Reflect

What will happen if an algorithm does not stop after a finite number of steps?

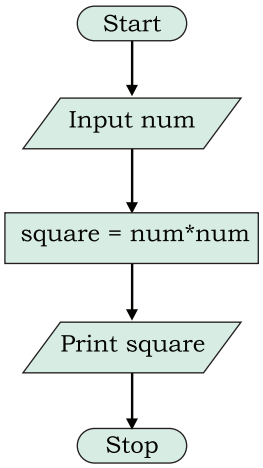


Figure 4.2: Flowchart to calculate square of a number

Activity 4.2

Draw a flowchart that represents the attainment of your career goal.

Example 4.2: Draw a flowchart to solve the problem of a non-functioning light bulb

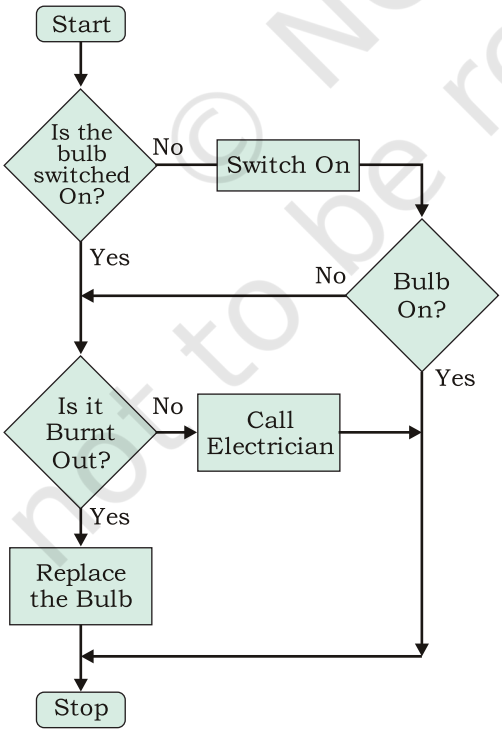


Figure 4.3: Flowchart to solve the problem of a non-functioning light bulb

4.4.2 Pseudocode

A pseudocode (pronounced Soo-doh-kohd) is another way of representing an algorithm. It is considered as a non-formal language that helps programmers to write algorithm. It is a detailed description of instructions that a computer must follow in a particular order. It is intended for human reading and cannot be executed directly by the computer. No specific standard for writing a pseudocode exists. The word "pseudo" means "not real," so "pseudocode" means "not real code". Following are some of the frequently used keywords while writing pseudocode:

- INPUT
- COMPUTE
- PRINT
- INCREMENT
- DECREMENT
- IF/ELSE
- WHILE
- TRUE/FALSE

Example 4.3: Write an algorithm to display the sum of two numbers entered by user, using both pseudocode and flowchart.

Pseudocode for the sum of two numbers will be:

```
INPUT num1
INPUT num2
COMPUTE Result = num1 + num2
PRINT Result
```

The flowchart for this algorithms is given in Figure 4.4.

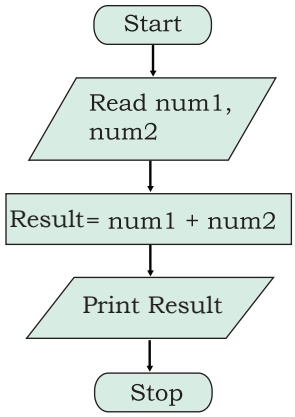


Figure 4.4: Flowchart to display sum of two numbers

Activity 4.3

Write a pseudocode for creating a scoreboard for a hockey match.

Example 4.4: Write an algorithm to calculate area and perimeter of a rectangle, using both pseudocode and flowchart.

Pseudocode for calculating area and perimeter of a rectangle.

```
INPUT length
INPUT breadth
COMPUTE Area = length * breadth
PRINT Area
COMPUTE Perim = 2 * (length + breadth)
PRINT Perim
```

The flowchart for this algorithm is given in Figure 4.5.

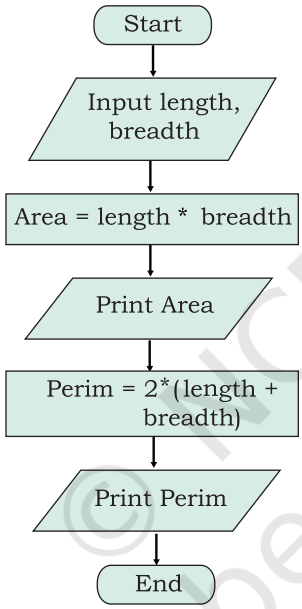


Figure 4.5: Flowchart to calculate area and perimeter of a rectangle

(A) *Benefits of Pseudocode*

Before writing codes in a high level language, a pseudocode of a program helps in representing the basic functionality of the intended program. By writing the code first in a human readable language, the programmer safeguards against leaving out any important step. Besides, for non-programmers, actual programs are difficult to read and understand, but pseudocode helps them to review the steps to confirm that the proposed implementation is going to achieve the desire output.

4.5 FLOW OF CONTROL

The flow of control depicts the flow of events as represented in the flow chart. The events can flow in a sequence, or on branch based on a decision or even repeat some part for a finite number of times.

4.5.1 Sequence

If we look at the examples 4.3 and 4.4, the statements are executed one after another, i.e., in a sequence. Such algorithms where all the steps are executed one after the other are said to execute in sequence. However, statements in an algorithm may not always execute in a sequence. We may sometimes require the algorithm to either do some routine tasks in a repeated manner or behave differently depending on the outcomes of previous steps. In this section, we are going to learn how to write algorithms for such situations.

4.5.2 Selection

Consider the map of a neighbourhood as shown in Figure 4.6. Let us assume that the pink building with the red roof is the school; the yellow painted house at the far end of the map is a house.

Think and Reflect

Can you list some of the routine activities in your daily life where decision making is involved?

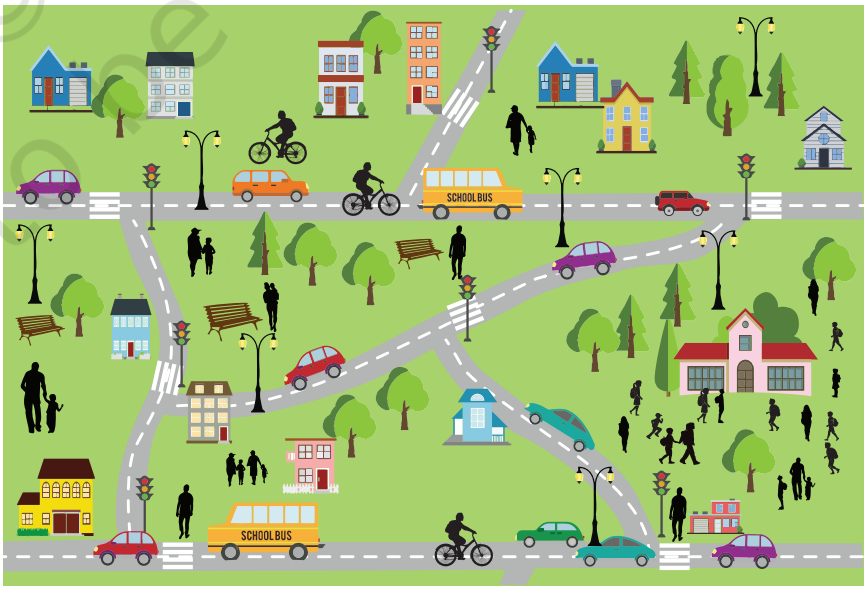


Figure 4.6: Decision making in real life

Notes

With reference to Figure 4.6, let us answer the following questions :

- Is there a predefined route for walking from home to school?
- Can we have a different route while coming back?

As seen from the map, there can be multiple routes between home and school. We might take the shortest route in the morning. But while coming back home in the afternoon, the shortest route might have heavy traffic. Therefore, we could take another route with less traffic. Hence, the above problem involves some decision-making based on certain conditions.

Let us look at some other examples where decision making is dependent on certain conditions. For example,

(i) Checking eligibility for voting.

Depending on their age, a person will either be allowed to vote or not allowed to vote:

- If age is greater than or equal to 18, the person is eligible to vote
- If age is less than 18, the person is not eligible to vote

(ii) Let us consider another example

If a student is 8 years old and the student likes Maths

put the student in Group A

Otherwise

Put the student in Group B

In which group will these students go as per the above condition?

Outcome

- 8-year-old Ravi who does not like Maths: Group B
- 8-year-old Priti who likes Maths: Group A
- 7-year-old Anish who likes Maths: Group B

In these examples, any one of the alternatives is selected based on the outcome of a condition. Conditionals are used to check possibilities. The program checks one or more conditions and perform operations (sequence of actions) depending on true or false value of the condition. These true or false values are called binary values.

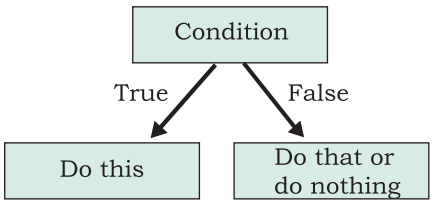


Figure 4.7: Actions depending on true or false of a condition

Conditionals are written in the algorithm as follows:

```
If <condition> then
     steps to be taken when the
condition is true/fulfilled
```

There are situations where we also need to take action when the condition is not fulfilled (Figure 4.7). To represent that, we can write:

```
If <condition> is true then
steps to be taken when the condition is
true/fulfilled
otherwise
     steps to be taken when the condition is
false/not fulfilled
```

In programming languages, 'otherwise' is represented using Else keyword. Hence, a true/false conditional is written using if-else block in actual programs.

Example 4.5: Let us write an algorithm to check whether a number is odd or even.

- Input: Any number
- Process: Check whether the number is even or not
- Output: Message "Even" or "Odd"

Pseudocode of the algorithm can be written as follows:

```
PRINT "Enter the Number"
INPUT number
IF number MOD 2 == 0 THEN
     PRINT "Number is Even"
ELSE
     PRINT "Number is Odd"
```

The flowchart representation of the algorithm in shown in Figure 4.8.

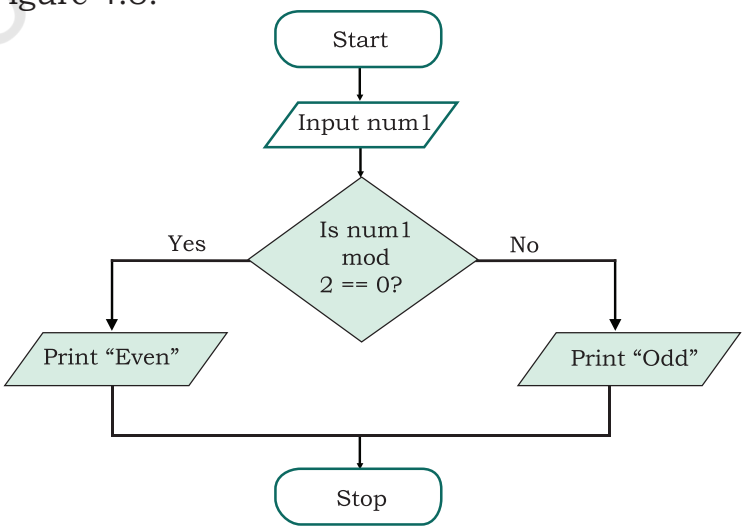


Figure 4.8: Flowchart to check whether a number is even or odd

Example 4.6: Let us write a pseudocode and draw a flowchart where multiple conditions are checked to categorise a person as either child (<13), teenager (>=13 but <20) or adult (>=20),based on age specified:

- Input: Age
- Process: Check Age as per the given criteria
- Output: Print either "Child", "Teenager", "Adult"

Pseudocode is as follows:

```
INPUT Age
IF Age < 13 THEN
        PRINT "Child"
  ELSE IF Age < 20 THEN
        PRINT "Teenager"
  ELSE
        PRINT "Adult"
```

The flowchart representation of the algorithm in shown in Figure 4.9

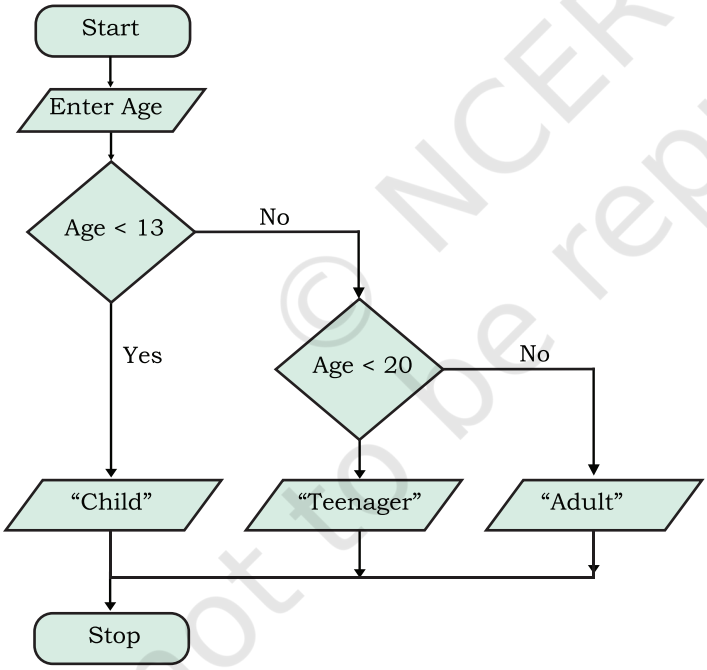


Figure 4.9: Flowchart to check multiple conditions

Example 4.7: Algorithm for a card game called "Dragons and Wizards".

Make two teams DRAGONS and WIZARDS

The rules for the game are as follows:

- If the card drawn is a diamond or a club, Team DRAGONS gets a point
- If the card drawn is a heart which is a number, Team WIZARDS gets a point

NOTES

- If the card drawn is a heart that is not a number, Team DRAGONS gets a point
- For any other card, Team WIZARDS gets a point
- The team with highest point is the winner

Let us identify the following for a card:
Input: shape, value
Process: Increment in respective team scores by one based on the outcome of the card drawn, as defined in the rules.
Output: Winning team
Now let us write the conditionals for this game:

```
IF (shape is diamond) OR (shape is club)
     Team DRAGONS gets a point
ELSE IF (shape is heart) AND (value is
number)
     Team WIZARDS gets a point
ELSE IF (shape is heart) AND (value is not a
number)
     Team DRAGONS gets a point
ELSE
     Team WIZARDS gets a point
```

The pseudocode for the program can be as follows:
Note: Dpoint (for Dragon) and Wpoint (for Wizard) store points scored by the respective teams.

```
INPUT shape
INPUT value
SET Dpoint = 0, Wpoint = 0
IF (shape is diamond) OR (shape is club) THEN
     INCREMENT Dpoint
ELSE IF (shape is heart) AND (value is
number)THEN
     INCREMENT Wpoint
ELSE IF (shape is heart) AND (value is not a
number)THEN
     INCREMENT Dpoint
ELSE
     INCREMENT Wpoint
END IF
IF Dpoint > Wpoint THEN
  PRINT "Dragon team is the winner"
ELSE
  PRINT "Wizard team is the winner"
```

4.5.3 Repetition

When giving directions to go someplace, we say something like, “walk 50 steps then turn right”. Or “Walk till the next

crossing then take a right turn". Consider some other examples like:

- Clap your hands five times
- Walk 10 steps ahead
- Jump on the spot till you get tired

These are the kind of statements we use, when we want something to be done repeatedly, for a given number of times. Likewise, suppose 10 cards need to be withdrawn in the previous card game (example 4.7), then the pseudocode needs to be repeated 10 times to decide the winner. All these are examples of repetitions. In programming, repetition is also known as iteration or loop. A loop in an algorithm means execution of some program statements repeatedly till some specified condition is satisfied.

Think and Reflect

Can you list some of the routine activities in your daily life where repetition or iteration is involved?

Example 4.8: Write pseudocode and draw a flowchart to accept 5 numbers and find their average.

The flowchart representation is shown in Figure 4.10.

Pseudocode will be as follows:

Step 1: SET count = 0, sum = 0
Step 2: WHILE count < 5 , REPEAT steps 3 to 5
Step 3: INPUT a number to num
Step 4: sum = sum + num
Step 5: count = count + 1
Step 6: COMPUTE average = sum/5
Step 7: PRINT average

In example 4.8, a counter called "count" keeps track of number of times the loop has been repeated. After every iterationof the loop, the value of count is incremented by 1 until it performs the set number of repetitions, given in the iteration condition.

There are situations when we are not aware beforehand about the number of times a set

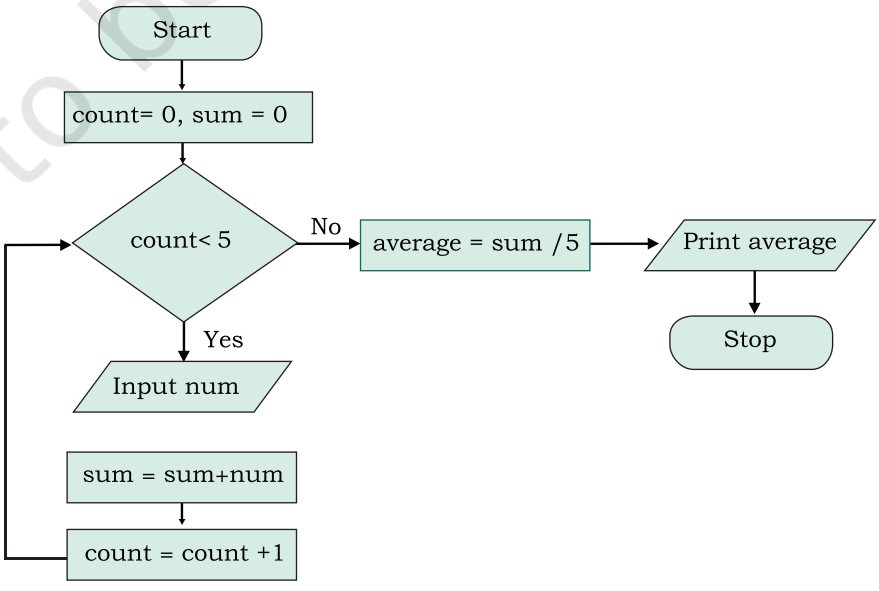


Figure 4.10: Flowchart to Calculate the Average of 5 Numbers

of statements need to be repeated. Such requirements of unknown number of repetitions are handled using WHILE construct.

Example 4.9: Write pseudocode and draw flowchart to accept numbers till the user enters 0 and then find their average.

Pseudocode is as follows:

Step 1: SET count = 0, sum = 0
Step 2: INPUT num
Step 3: WHILE num is not equal to 0, REPEAT Steps 4 to 6
Step 4: sum = sum + num
Step 5: count = count + 1
Step 6: INPUT num
Step 7: COMPUTE average = sum/count
Step 8: PRINT average
The flowchart representation is shown in Figure 4.11.

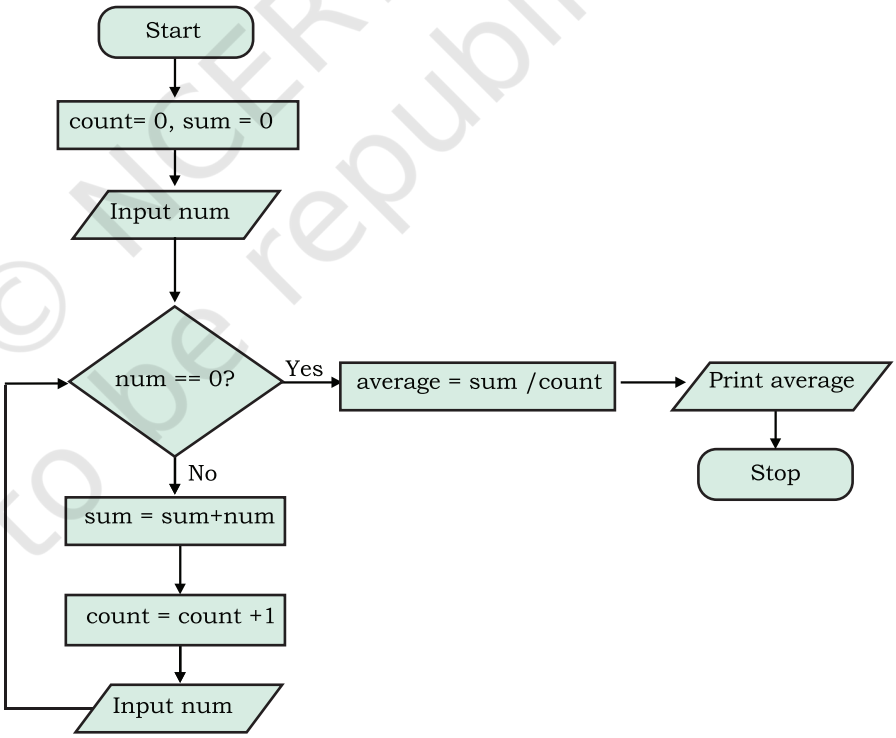


Figure 4.11: Flowchart to accept numbers till the user enters 0

In this example, we do not know how many numbers a user is going to enter before entering 0. This is handled by checking the condition repeatedly till the condition becomes false.

Activity 4.4

Let us answer the following questions using the pesudocode given in example 4.9:

1) What will the sum when the input are 6, 7, 4, 8, 2, 5, 0, 3, 1.
2) What will be the value of count?
3) Why did we use the input statement to enter num twice?
4) Why did we divide sum by count?
5) Can there be any other approach?

4.6 Verifying Algorithms

Can you imagine what would happen if a banking software does not work correctly? Suppose functioning of the online money transfer module is not programmed correctly, and it credits into the account only half the amount transacted! What happens if the account is debited instead of being credited. Such a faulty software will mess up the working of the complete system and cause havoc! Today software are used in even more critical services — like in the medical field or in space shuttles. Such software needs to work correctly in every situation. Therefore, the software designer should make sure that the functioning of all the components are defined correctly, checked and verified in every possible way.

Think and Reflect

Why is verification of algorithm an important step in problem solving?

When we were told that the formula for the sum of first N natural numbers is $\frac{N(N+1)}{2}$, how did we verify it? Well, we can check this for small numbers, for which we can manually calculate the sum. Let N = 6, then the sum is 1 + 2 + 3 + 4 + 5 + 6 = 21

Using formula we get sum = $\frac{6x(6+1)}{2}$

We can try with some more numbers this way to ensure that the formula works correctly.

In the same way, when we have written an algorithm, we want to verify that it is working as expected. To verify, we have to take different input values and go through all the steps of the algorithm to yield the desired output for each input value and may modify or improve as per the need. The method of taking an input and running through the steps of the algorithm is sometimes called *dry run*. Such a dry run will help us to:

1. Identify any incorrect steps in the algorithm
2. Figure out missing details or specifics in the algorithm

Activity 4.5

Write an algorithm to take as input the measurement of length and breadth in feet and inches (e.g., 5 ft 6 inch) of a rectangular shape and calculate its area and perimeter.

It is important to decide the type of input value to be used for the simulation. In case all possible input values are not tested, then the program will fail. What if there is some other case for which it does not work? Let us look at some examples.

Write an algorithm to calculate the time taken to go from place A to C (T_total) via B where time taken to

NOTES

go from A to B (T1) and B to C (T2) are given. That is, we want the algorithm to add time given in hours and minutes. One way to write the algorithm is:

```
PRINT value for T1
INPUT hh1
INPUT mm1
PRINT value for T2
INPUT hh2
INPUT mm2
hh_total = hh1 + hh2 (Add hours)
mm_total = mm1 + mm2 (Add mins)
Print T_total as hh_total, mm_total
```

Now let us verify. Suppose the first example we take is T1 = 5 hrs 20 mins and T2 = 7 hrs 30 mins. On dry run, we get the result 12 hrs and 50 mins. This looks fine.

Now let us take another example where T1 = 4 hrs 50 mins and T2 = 2 hrs 20 mins, and we end up getting the result as 6 hrs 70 mins which is not how we measure time. The result should have been 7 hrs 10 mins.

With this second example we realise that our algorithm will work only when mm1 + mm2 (mm_total) < 60. For all other cases, it will give us output not the way we want. When mm_total >= 60, the algorithm should increase the sum of hours (hh_total) by 1 and reduce mm_total by 60, i.e., (mm_total - 60). So the modified algorithm will be:

```
PRINT value for T1
INPUT hh1
INPUT mm1
PRINT value for T2
INPUT hh2
INPUT mm2
hh_total = hh1 + hh2 (Add hours)
mm_total = mm1 + mm2 (Add mins)
hh_total = hh1 + hh2 (Add hours)
mm_total = mm1 + mm2 (Add mins)
IF (mm_total >= 60) THEN
     hh_total = hh_total + 1
     mm_total = mm_total - 60
PRINT T_total as hh_total, mm_total
```

Now we can simulate through algorithm for T1 = 4 hrs 50 mins and T2 = 2 hrs 20 mins, and get T_total = 7 hrs and 10 mins, which means the algorithm is working correctly.

Suppose we develop some software without verifying the underlying algorithm and if there are errors in the algorithm, then the software developed will not run. Hence, it is important to verify an algorithm since the effort required to catch and fix a mistake is minimal.

4.7 Comparison of Algorithm

There can be more than one approach to solve a problem using computer and hence we can have more than one algorithm. Then one may ask which algorithm should be used?

Consider the problem of finding whether a given number is prime or not. Prime numbers are of great importance in computer science as they find application in databases, security, file compression or decompression, modulation or demodulation, etc. There can be four different ways to write algorithms to check whether a given number is prime or not as shown below:

(i) Starting with divisor 2, divide the given number (dividend) and check if there are any factors. Increase the divisor in each iteration and repeat the previous steps as long as divisor < dividend. If there is a factor, then the given number is not prime

(ii) In (i), instead of testing all the numbers till the dividend, only test up to half of the given value (dividend) because the divisor can not be more than half of the dividend

(iii) In method (i), only test up to the square root of the dividend (numbers)

(iv) Given a prior list of prime number till 100, divide the given number by each number in the list. If not divisible by any number, then the number is a prime else it is not prime

All these four methods can check if a given number is prime or not. Now the question is which of these methods is better or efficient?

Algorithm (i) requires large number of calculations (means more processing time) as it checks for all the numbers as long as the divisor is less than the number. If the given number is large, this method will take more time to give the output.

The spirit of problem solving by decomposition is to 'divide and conquer'. In words of *Howard Raffa*, a famous mathematician:

"Decompose a complex problem into simpler problems get one's thinking straight in these simpler problems, put these analyses together with logical glue"

Algorithm (ii) is more efficient than (i) as it checks for divisibility till half the number, and thus it reduces the time for computation of the prime number. Algorithm (iii) is even more efficient as it checks for divisibility till square root of the number, thereby further reducing the time taken.

As algorithm (iv) uses only the prime numbers smaller than the given number for divisibility, it further reduces the calculations. But in this method we require to store the list of prime numbers first. Thus it takes additional memory even though it requires lesser calculations.

Hence, algorithms can be compared and analysed on the basis of the amount of processing time they need to run and the amount of memory that is needed to execute the algorithm. These are termed as time complexity and space complexity, respectively. The choice of an algorithm over another is done depending on how efficient they are in terms of proecssing time required (time complexity) and the memory they utilise (space complexity).

4.8 CODING

Once an algorithm is finalised, it should be coded in a high-level programming language as selected by the programmer. The ordered set of instructions are written in that programming language by following its syntax. Syntax is the set of rules or grammar that governs the formulation of the statements in the language, such as spellings, order of words, punctuation, etc.

The machine language or low level language consisting of 0s and 1s only is the ideal way to write a computer program. Programs written using binary digits are directly understood by the computer hardware, but they are difficult to deal with and comprehend by humans. This led to the invention of high-level languages which are close to natural languages and are easier to read, write, and maintain, but are not directly understood by the computer hardware. An advantage of using high-level languages is that they are portable, i.e., they can run on different types of computers with little

Notes

or no modifications. Low-level programs can run on only one kind of computer and have to be rewritten in order to run on another type of system. A wide variety of high-level languages, such as FORTRAN, C, C++, Java, Python, etc., exist.

A program written in a high-level language is called source code. We need to translate the source code into machine language using a compiler or an interpreter, so that it can be understood by the computer. We have learnt about the compiler and interpreter in Chapter 1.

There are multiple programming languages available and choosing the one suitable for our requirements requires us to consider many factors. It depends on the platform (OS) where the program will run. We need to decide whether the application would be a desktop application, a mobile application or a web application. Desktop and mobile applications are generally developed for a particular operating system and for certain hardware whereas the web applications are accessed in different devices using web browsers and may use resources available over cloud.

Besides, programs are developed not only to work on a computer, mobile or a web browser, but it may also be written for embedded systems like digital watch, mp3 players, traffic signals or vehicles, medical equipments and other smart devices. In such cases, we have to look for other specialised programming tools or sometimes write programs in assembly languages.

4.9 Decomposition

Sometimes a problem may be complex, that is, its solution is not directly derivable. In such cases, we need to decompose it into simpler parts. Let us look at the Railway reservation system we talked about earlier. The complex task of designing a good railway reservation system is seen as designing the different components of the system and then making them work with each other effectively.

The basic idea of solving a complex problem by decomposition is to 'decompose' or break down a complex problem into smaller sub problems as shown

in Figure 4.12. These sub problems are relatively easier to solve than the original problem. Finally, the sub-problems are combined in a logical way to obtain the solution for the bigger, main problem.

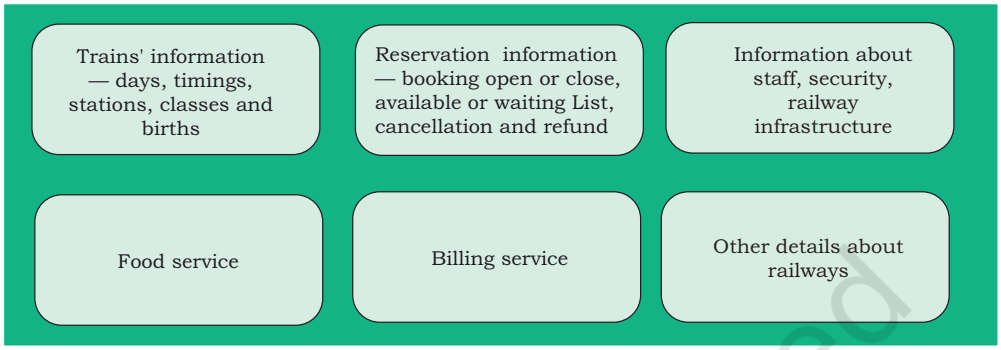


Figure 4.12: Railway reservation system

Breaking down a complex problem into sub problems also means that each sub problem can be examined in detail. Each sub problem can be solved independently and by different persons (or teams). Having different teams working on different sub problems can also be advantageous because specific sub problems can be assigned to teams who are experts in solving such problems.

There are many real life problems which can be solved using decomposition. Examples include solving problems in mathematics and science, events management in school, weather forecasting, delivery management system, etc.

Once the individual sub problems are solved, it is necessary to test them for their correctness and integrate them to get the complete solution.

SUMMARY

- An algorithm is defined as a step-by-step procedure designed to perform an operation which will lead to the desired result, if followed correctly.
- Algorithms have a definite beginning and a definite end, and a finite number of steps.
- A good algorithm, which is precise, unique and finite, receives input and produces an output.

- In order to write effective algorithms we need to identify the input, the process to be followed and the desired output.
- A flowchart is a type of diagram that represents the algorithm graphically using boxes of various kinds, in an order connected by arrows.
- An algorithm where all the steps are executed one after the other is said to execute in sequence.
- Decision making involves selection of one of the alternatives based on outcome of a condition.
- An algorithm may have a certain set of steps, which are repeating for a finite number of times, such an algorithm is said to be iterative.
- There can be more than one approach to solve a problem and hence we can have more than one algorithm for a particular problem.
- The choice of algorithm should be made on the basis of time and space complexity.

Exercise

1. Write pseudocode that reads two numbers and divide one by another and display the quotient.
2. Two friends decide who gets the last slice of a cake by flipping a coin five times. The first person to win three flips wins the cake. An input of 1 means player 1 wins a flip, and a 2 means player 2 wins a flip. Design an algorithm to determine who takes the cake?
3. Write the pseudocode to print all multiples of 5 between 10 and 25 (including both 10 and 25).
4. Give an example of a loop that is to be executed a certain number of times.
5. Suppose you are collecting money for something. You need ₹ 200 in all. You ask your parents, uncles and aunts as well as grandparents. Different people may give either ₹ 10, ₹ 20 or even ₹ 50. You will collect till the total becomes 200. Write the algorithm.
6. Write the pseudocode to print the bill depending upon the price and quantity of an item. Also print

NOTES

Bill GST, which is the bill after adding 5% of tax in the total bill.

7. Write pseudocode that will perform the following:
   a) Read the marks of three subjects: Computer Science, Mathematics and Physics, out of 100
   b) Calculate the aggregate marks
   c) Calculate the percentage of marks
8. Write an algorithm to find the greatest among two different numbers entered by the user.
9. Write an algorithm that performs the following: Ask a user to enter a number. If the number is between 5 and 15, write the word GREEN. If the number is between 15 and 25, write the word BLUE. if the number is between 25 and 35, write the word ORANGE. If it is any other number, write that ALL COLOURS ARE BEAUTIFUL.
10. Write an algorithm that accepts four numbers as input and find the largest and smallest of them.
11. Write an algorithm to display the total water bill charges of the month depending upon the number of units consumed by the customer as per the following criteria:
    - for the first 100 units @ 5 per unit
    - for next 150 units @ 10 per unit
    - more than 250 units @ 20 per unit

    Also add meter charges of 75 per month to calculate the total water bill .
12. What are conditionals? When they are required in a program?
13. Match the pairs

Flowchart Symbol	Functions
	Flow of Control
	Process Step
	Start/Stop of the Process

Data

Decision Making

14. Following is an algorithm for going to school or college. Can you suggest improvements in this to include other options?

Reach_School_Algorithm

a) Wake up

b) Get ready

c) Take lunch box

d) Take bus

e) Get off the bus

f) Reach school or college

15. Write a pseudocode to calculate the factorial of a number (Hint: Factorial of 5, written as $5! = 5 \times 4 \times 3 \times 2 \times 1$) .

16. Draw a flowchart to check whether a given number is an Armstrong number. An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself. For example, 371 is an Armstrong number since 3**3 + 7**3 + 1**3 = 371.

17. Following is an algorithm to classify numbers as "Single Digit", "Double Digit" or "Big".

Classify_Numbers_Algo

```
INPUT Number
IF Number < 9
     "Single Digit"
Else If Number < 99
     "Double Digit"
Else
     "Big"
```

Verify for (5, 9, 47, 99, 100 200) and correct the algorithm if required

18. For some calculations, we want an algorithm that accepts only positive integers upto 100.

NOTES

Accept_1to100_Algo

```
INPUT Number
IF (0<= Number) AND (Number <= 100)
     ACCEPT
Else
     REJECT
```

a) On what values will this algorithm fail?

b) Can you improve the algorithm?

CBSE Class 11 Study Material

- Printable Worksheets for Class 11

NCERT Solutions for Class 11

- NCERT Solutions for class 11 Maths
- NCERT Solutions for class 11 Physics
- NCERT Solutions for class 11 Chemistry
- NCERT Solutions for class 11 Biology
- NCERT Solutions for class 11 English
- NCERT Solutions for Class 11 English Woven Words Essay
- NCERT Solutions for Class 11 English Woven Short Stories
- NCERT Solutions for Class 11 English Woven Words Poetry
- NCERT Solutions for class 11 Accountancy
- NCERT Solutions for class 11 Business Studies
- NCERT Solutions for class 11 Economics
- NCERT Solutions for class 11 Computer Science – Python
- Class 11 Hindi Aroh (आरोह भाग 1)
- Class 11 Hindi Vitan (वितान भाग 1)

- Class 11 Sanskrit
- Class 11 History
- Class 11 Geography
- Class 11 Indian Economic Development
- Class 11 Statistics for Economics
- Class 11 Political Science
- Class 11 Psychology
- Class 11 Sociology
- Class 11 Entrepreneurship

- **Maths formulas for Class 11**
- Hindi Grammar for Class 11
- Class 11 English Hornbill Summaries
- Class 11 English Snapshots Summaries
- CBSE Sample Papers for Class 11
- NCERT Exemplar Class 11 Maths Solutions
- NCERT Exemplar Class 11 Physics Solutions
- NCERT Exemplar Class 11 Chemistry Solutions
- NCERT Exemplar Class 11 Biology Solutions
- RD Sharma Class 11 Solutions
- **CBSE Class 11 and 12 Revised Syllabus**
- MCQ Questions

- CBSE Class 11 Physics Manual
- CBSE Class 11 Chemistry Manual
- Trigonometry Formulas
- Integration Formulas
- JEE Main Study Material
- NEET Study Material

- CBSE Class 11 Notes
- Class 11 Maths Notes
- Class 11 Physics Notes
- Class 11 Chemistry Notes
- Class 11 Biology Notes
- Class 11 English Notes
- Class 11 English Woven Words Short Stories
- CBSE Class 11 English Woven Words Essay
- CBSE Class 11 English Woven Words Poetry
- CBSE Class 11 English Snapshots
- CBSE Class 11 English Hornbill
- Class 11 Business Studies Notes
- Class 11 Accountancy Notes
- Class 11 Psychology Notes
- Class 11 Entrepreneurship Notes
- Class 11 Economics Notes

- Class 11 Indian Economic Development Notes
- Statistics for Economics Class 11 Notes
- Class 11 Political Science Notes
- Class 11 History Notes
- Sociology Class 11 Notes
- Geography Class 11 Notes

NCERT Books for Class 11

- Class 11 NCERT Maths Books
- Class 11 Physics NCERT Book
- Class 11 Chemistry NCERT Book
- Class 11 Biology NCERT Book
- Class 11 Political Theory Part-I
- Class 11 NCERT Business Studies Books
- Class 11 India Constitution at Work
- NCERT Geography Book Class 11
- NCERT Class 11 History Book
- Class 11 India Economic Development
- Class 11 NCERT English Books
- NCERT Sanksrit Books Class 11
- Class 11 Computer and Communication Technology Book
- Class 11 NCERT Accountancy Books

- Class 11 Statistics
- Class 11 Introduction to Psychology
- Class 11 Introducing Sociology
- Class 11 Understanding Society
- Class 11 Fine Arts
- Class 11 Heritage Craft Books
- Class 11 Nai Awaz
- Class 11 Dhanak
- Class 11 The story of Graphic Design
- Class 11 Human Ecology and Family Sciences

Chapter 5

Getting Started with Python

"Computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better."

– Donald Knuth

5.1 Introduction to Python

We have written algorithms for different problems in Chapter 4. Let us now move a step further and create programs using any version of Python 3. But before learning about Python programming language, let us understand what is a programming language and how it works.

An ordered set of instructions to be executed by a computer to carry out a specific task is called a program, and the language used to specify this set of instructions to the computer is called a programming language.

As we know that computers understand the language of 0s and 1s which is called machine language or low level language. However, it is difficult for humans to write or comprehend instructions using 0s and 1s. This led to the advent of high-level programming languages like Python, C++, Visual Basic, PHP, Java that are easier to manage by humans but are not directly understood by the computer.

A program written in a high-level language is called source code. Recall from Chapter 1 that language translators like compilers and interpreters are needed to translate the source code into machine language. Python uses an interpreter to convert its instructions into machine language, so that it can be understood by the computer. An interpreter processes the program statements one by one, first translating and then executing. This process is continued until an error is encountered or the whole program is executed successfully. In both the cases, program execution will stop. On the contrary, a compiler translates the entire source code, as a whole, into the object code. After scanning the whole program, it generates error messages, if any.

In this chapter

- Introduction to Python
- Python Keywords
- Identifiers
- Comments
- Data Types
- Operators
- Expressions
- Statement
- Input and Output
- Type Conversion
- Debugging

5.1.1 Features of Python

- Python is a high level language. It is a free and open source language.
- It is an interpreted language, as Python programs are executed by an interpreter.
- Python programs are easy to understand as they have a clearly defined syntax and relatively simple structure.
- Python is case-sensitive. For example, NUMBER and number are not same in Python.
- Python is portable and platform independent, means it can run on various operating systems and hardware platforms.
- Python has a rich library of predefined functions.
- Python is also helpful in web development. Many popular web services and applications are built using Python.
- Python uses indentation for blocks and nested blocks.

Downloading Python

The latest version of Python 3 is available on the official website:

https://www.python.org/

5.1.2 Working with Python

To write and run (execute) a Python program, we need to have a Python interpreter installed on our computer or we can use any online Python interpreter. The interpreter is also called Python shell. A sample screen of Python interpreter is shown in Figure 5.1:

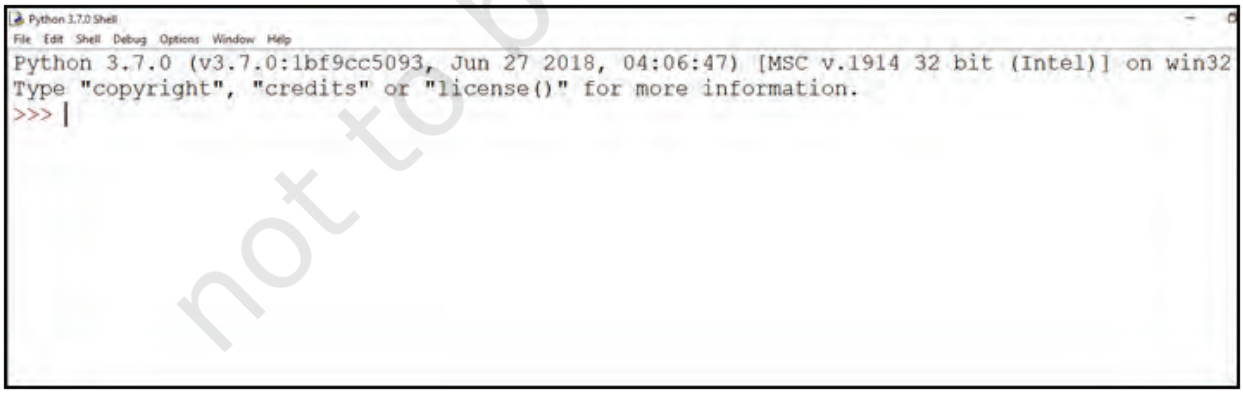


Figure 5.1: Python interpreter or shell

In the above screen, the symbol >>> is the Python prompt, which indicates that the interpreter is ready to take instructions. We can type commands or statements on this prompt to execute them using a Python interpreter.

5.1.3 Execution Modes

There are two ways to use the Python interpreter:

a) Interactive mode
b) Script mode

Interactive mode allows execution of individual statement instantaneously. Whereas, Script mode allows us to write more than one instruction in a file called Python source code file that can be executed.

(A) Interactive Mode

To work in the interactive mode, we can simply type a Python statement on the >>> prompt directly. As soon as we press enter, the interpreter executes the statement and displays the result(s), as shown in Figure 5.2.

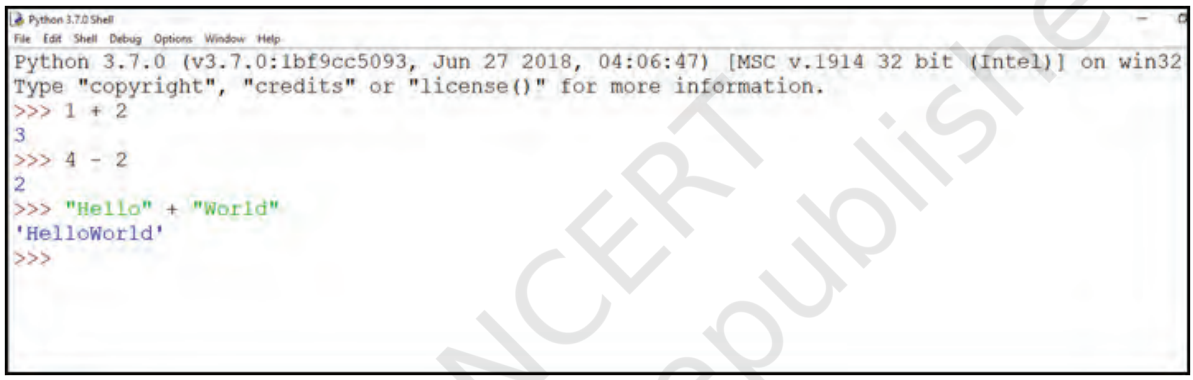


Figure 5.2: Python interpreter in interactive mode

Working in the interactive mode is convenient for testing a single line code for instant execution. But in the interactive mode, we cannot save the statements for future use and we have to retype the statements to run them again.

(B) Script Mode

In the script mode, we can write a Python program in a file, save it and then use the interpreter to execute it. Python scripts are saved as files where file name has extension ".py". By default, the Python scripts are saved in the Python installation folder. To execute a script, we can either:

a) Type the file name along with the path at the prompt. For example, if the name of the file is prog5-1.py, we type prog5-1.py. We can otherwise open the program directly from IDLE as shown in Figure 5.3.
b) While working in the script mode, after saving the file, click [Run]->[Run Module] from the menu as shown in Figure 5.4.

c) The output appears on shell as shown in Figure 5.5.

Program 5-1 Write a program to show print statement in script mode.

Figure 5.3: Python source code file (prog5-1.py)

Figure 5.4: Execution of Python in Script mode using IDLE

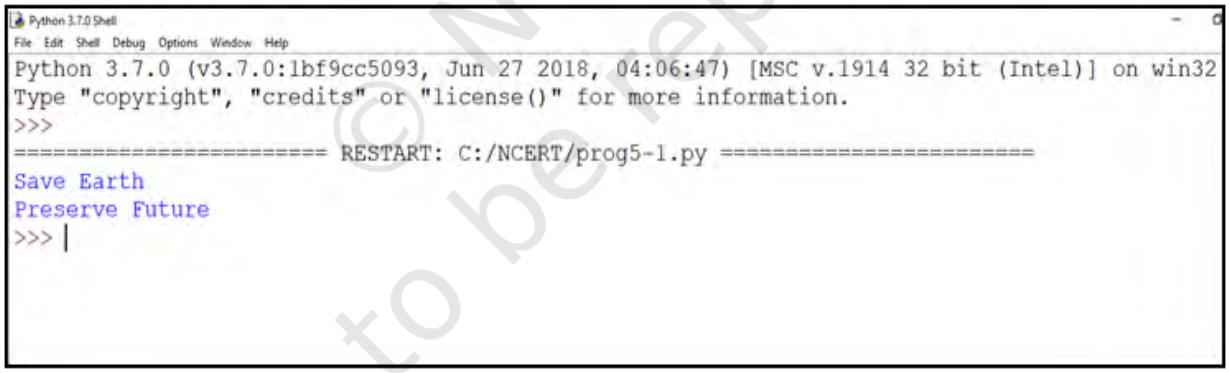


Figure 5.5: Output of a program executed in script mode

5.2 PYTHON KEYWORDS

Keywords are reserved words. Each keyword has a specific meaning to the Python interpreter, and we can use a keyword in our program only for the purpose for which it has been defined. As Python is case sensitive, keywords must be written exactly as given in Table 5.1.

Table 5.1 Python keywords

False	class	finally	is	return
None	continue	for	lambda	try

NOTES

True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

5.3 IDENTIFIERS

In programming languages, identifiers are names used to identify a variable, function, or other entities in a program. The rules for naming an identifier in Python are as follows:

- The name should begin with an uppercase or a lowercase alphabet or an underscore sign (_). This may be followed by any combination of characters a–z, A–Z, 0–9 or underscore (_). Thus, an identifier cannot start with a digit.
- It can be of any length. (However, it is preferred to keep it short and meaningful).
- It should not be a keyword or reserved word given in Table 5.1.
- We cannot use special symbols like !, @, #, $, %, etc., in identifiers.

For example, to find the average of marks obtained by a student in three subjects, we can choose the identifiers as `marks1, marks2, marks3` and `avg` rather than a, b, c, or A, B, C.

```
avg = (marks1 + marks2 + marks3)/3
```

Similarly, to calculate the area of a rectangle, we can use identifier names, such as `area, length, breadth` instead of single alphabets as identifiers for clarity and more readability.

```
area = length * breadth
```

5.4 VARIABLES

A variable in a program is uniquely identified by a name (identifier). Variable in Python refers to an object — an item or element that is stored in the memory. Value of a variable can be a string (e.g., 'b', 'Global Citizen'), numeric (e.g., 345) or any combination of alphanumeric characters (CD67). In Python we can use an assignment statement to create new variables and assign specific values to them.

```
gender    = 'M'
message   = "Keep Smiling"
price     = 987.9
```

Program 5-2 Write a program to display values of variables in Python.

```
#Program 5-2
#To display values of variables
message = "Keep Smiling"
print(message)
userNo = 101
print('User Number is', userNo)
```

Output:

```
Keep Smiling
User Number is 101
```

In the program 5-2, the variable message holds string type value and so its content is assigned within double quotes " " (can also be within single quotes ' '), whereas the value of variable userNo is not enclosed in quotes as it is a numeric value.

Variable declaration is implicit in Python, means variables are automatically declared and defined when they are assigned a value the first time. Variables must always be assigned values before they are used in expressions as otherwise it will lead to an error in the program. Wherever a variable name occurs in an expression, the interpreter replaces it with the value of that particular variable.

Program 5-3 Write a Python program to find the area of a rectangle given that its length is 10 units and breadth is 20 units.

```
#Program 5-3
#To find the area of a rectangle
length = 10
breadth = 20
area = length * breadth
print(area)
```

Output:

```
200
```

5.5 COMMENTS

Comments are used to add a remark or a note in the source code. Comments are not executed by interpreter.

They are added with the purpose of making the source code easier for humans to understand. They are used primarily to document the meaning and purpose of source code and its input and output requirements, so that we can remember later how it functions and how to use it. For large and complex software, it may require programmers to work in teams and sometimes, a program written by one programmer is required to be used or maintained by another programmer. In such situations, documentations in the form of comments are needed to understand the working of the program.

In Python, a comment starts with # (hash sign). Everything following the # till the end of that line is treated as a comment and the interpreter simply ignores it while executing the statement.

Example 5.1

```
#Variable amount is the total spending on
#grocery
amount = 3400
#totalMarks is sum of marks in all the tests
#of Mathematics
totalMarks = test1 + test2 + finalTest
```

Program 5-4 Write a Python program to find the sum of two numbers.

```
#Program 5-4
#To find the sum of two numbers
num1 = 10
num2 = 20
result = num1 + num2
print(result)
```

Output:

```
30
```

5.6 Everything is an Object

Python treats every value or data item whether numeric, string, or other type (discussed in the next section) as an object in the sense that it can be assigned to some variable or can be passed to a function as an argument.

Every object in Python is assigned a unique identity (ID) which remains the same for the lifetime of that object. This ID is akin to the memory address of the object. The function `id()` returns the identity of an object.

In the context of Object Oriented Programming (OOP), objects are a representation of the real world, such as employee, student, vehicle, box, book, etc. In any object oriented programming language like C++, JAVA, etc., each object has two things associated with it: (i) data or attributes and (ii) behaviour or methods. Further there are concepts of class and class hierarchies from which objects can be instantiated. However, OOP concepts are not in the scope of our present discussions.

Python also comes under the category of object oriented programming. However, in Python, the definition of object is loosely casted as some objects may not have attributes or others may not have methods.

Example 5.2

```
>>> num1 = 20
>>> id(num1)
1433920576          #identity of num1
>>> num2 = 30 - 10
>>> id(num2)
1433920576          #identity of num2 and num1
                          #are same as both
refers to                              #object 20
```

5.7 DATA TYPES

Every value belongs to a specific data type in Python. Data type identifies the type of data values a variable can hold and the operations that can be performed on that data. Figure 5.6 enlists the data types available in Python.

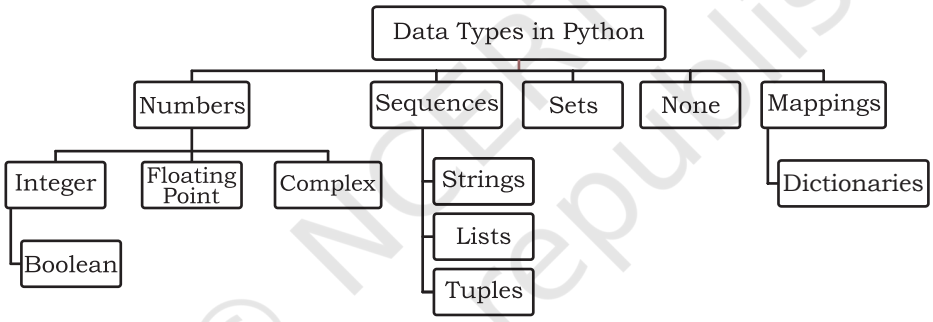


Figure 5.6: Different data types in Python

5.7.1 Number

Number data type stores numerical values only. It is further classified into three different types: `int`, `float` and `complex`.

Table 5.2 Numeric data types

Type/ Class	Description	Examples
`int`	integer numbers	–12, –3, 0, 125, 2
`float`	real or floating point numbers	–2.04, 4.0, 14.23
`complex`	complex numbers	3 + 4i, 2 – 2i

Boolean data type (`bool`) is a subtype of integer. It is a unique data type, consisting of two constants, `True` and `False`. Boolean `True` value is non-zero, non-null and non-empty. Boolean `False` is the value zero.

NOTES

Let us now try to execute few statements in interactive mode to determine the data type of the variable using built-in function `type()`.

Example 5.3

```
>>> num1 = 10
>>> type(num1)
<class 'int'>

>>> num2 = -1210
>>> type(num2)
<class 'int'>

>>> var1 = True
>>> type(var1)
<class 'bool'>

>>> float1 = -1921.9
>>> type(float1)
<class 'float'>
>>> float2 = -9.8*10**2
>>> print(float2, type(float2))
-980.0000000000001 <class 'float'>

>>> var2 = -3+7.2j
>>> print(var2, type(var2))
(-3+7.2j) <class 'complex'>
```

Variables of simple data types like integers, float, boolean, etc., hold single values. But such variables are not useful to hold a long list of information, for example, names of the months in a year, names of students in a class, names and numbers in a phone book or the list of artefacts in a museum. For this, Python provides data types like tuples, lists, dictionaries and sets.

5.7.2 Sequence

A Python sequence is an ordered collection of items, where each item is indexed by an integer. The three types of sequence data types available in Python are Strings, Lists and Tuples. We will learn about each of them in detail in later chapters. A brief introduction to these data types is as follows:

(A) String

String is a group of characters. These characters may be alphabets, digits or special characters including spaces. String values are enclosed either in single quotation

NOTES

marks (e.g., 'Hello') or in double quotation marks (e.g., "Hello"). The quotes are not a part of the string, they are used to mark the beginning and end of the string for the interpreter. For example,

```
>>> str1 = 'Hello Friend'
>>> str2 = "452"
```

We cannot perform numerical operations on strings, even when the string contains a numeric value, as in str2.

(B) List

List is a sequence of items separated by commas and the items are enclosed in square brackets [ ].

Example 5.4

```
#To create a list
>>> list1 = [5, 3.4, "New Delhi", "20C", 45]
#print the elements of the list list1
>>> print(list1)
[5, 3.4, 'New Delhi', '20C', 45]
```

(C) Tuple

Tuple is a sequence of items separated by commas and items are enclosed in parenthesis ( ). This is unlike list, where values are enclosed in brackets [ ]. Once created, we cannot change the tuple.

Example 5.5

```
#create a tuple tuple1
>>> tuple1 = (10, 20, "Apple", 3.4, 'a')
#print the elements of the tuple tuple1
>>> print(tuple1)
(10, 20, "Apple", 3.4, 'a')
```

5.7.3 Set

Set is an unordered collection of items separated by commas and the items are enclosed in curly brackets { }. A set is similar to list, except that it cannot have duplicate entries. Once created, elements of a set cannot be changed.

Example 5.6

```
#create a set
>>> set1 = {10,20,3.14,"New Delhi"}
>>> print(type(set1))
<class 'set'>
>>> print(set1)
{10, 20, 3.14, "New Delhi"}
#duplicate elements are not included in set
```

```
>>> set2 = {1,2,1,3}
>>> print(set2)
{1, 2, 3}
```

5.7.4 None

None is a special data type with a single value. It is used to signify the absence of value in a situation. `None` supports no special operations, and it is neither `False` nor 0 (zero).

Example 5.7

```
>>> myVar = None
>>> print(type(myVar))
<class 'NoneType'>
>>> print(myVar)
None
```

5.7.5 Mapping

Mapping is an unordered data type in Python. Currently, there is only one standard mapping data type in Python called dictionary.

(A) Dictionary

Dictionary in Python holds data items in key-value pairs. Items in a dictionary are enclosed in curly brackets { }. Dictionaries permit faster access to data. Every key is separated from its value using a colon (:) sign. The key : value pairs of a dictionary can be accessed using the key. The keys are usually strings and their values can be any data type. In order to access any value in the dictionary, we have to specify its key in square brackets [ ].

Example 5.8

```
#create a dictionary
>>> dict1 = {'Fruit':'Apple',
'Climate':'Cold', 'Price(kg)':120}
>>> print(dict1)
{'Fruit': 'Apple', 'Climate': 'Cold',
'Price(kg)': 120}
>>> print(dict1['Price(kg)'])
120
```

5.7.6 Mutable and Immutable Data Types

Sometimes we may require to change or update 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 shown in Figure 5.7.

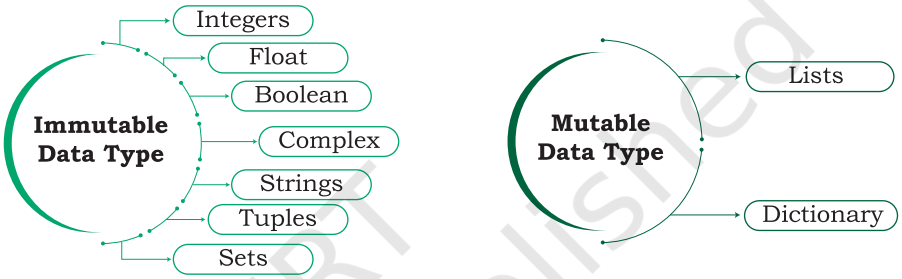


Figure 5.7: Classification of data types

Let us now see what happens when an attempt is made to update the value of a variable.

```
>>> num1 = 300
```

This statement will create an object with value 300 and the object is referenced by the identifier `num1` as shown in Figure 5.8.

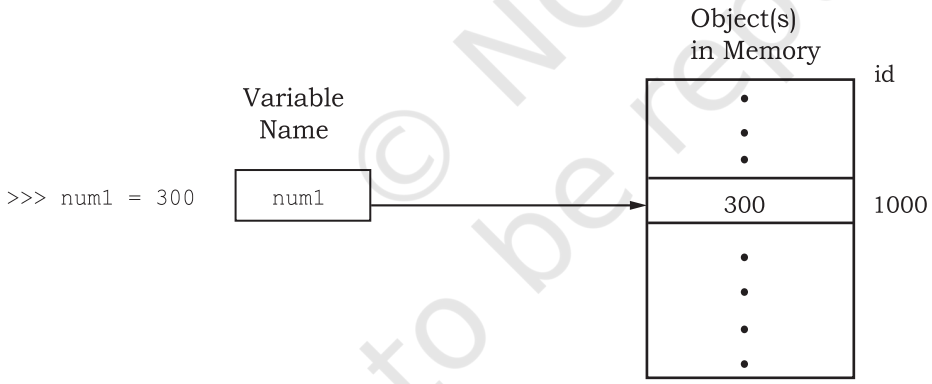


Figure 5.8: Object and its identifier

```
>>> num2 = num1
```

The statement `num2 = num1` will make `num2` refer to the value 300, also being referred by `num1`, and stored at memory location number, say 1000. So, `num1` shares the referenced location with `num2` as shown in Figure 5.9.

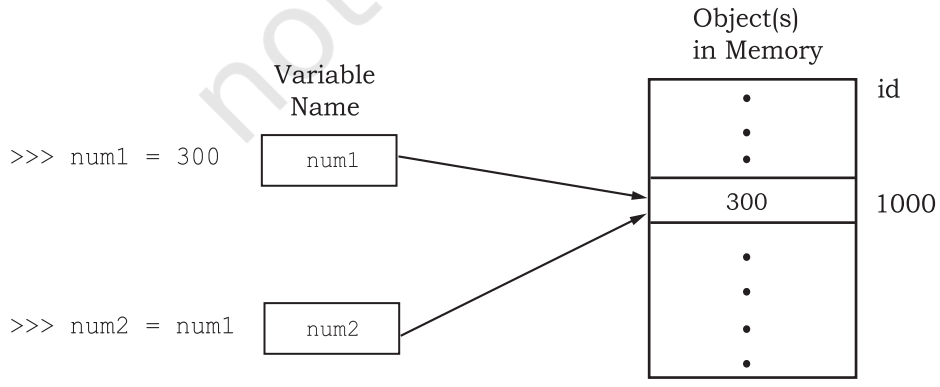


Figure 5.9: Variables with same value have same identifier

In this manner Python makes the assignment effective by copying only the reference, and not the data:

```
>>> num1 = 300
>>> num2 = num1
```

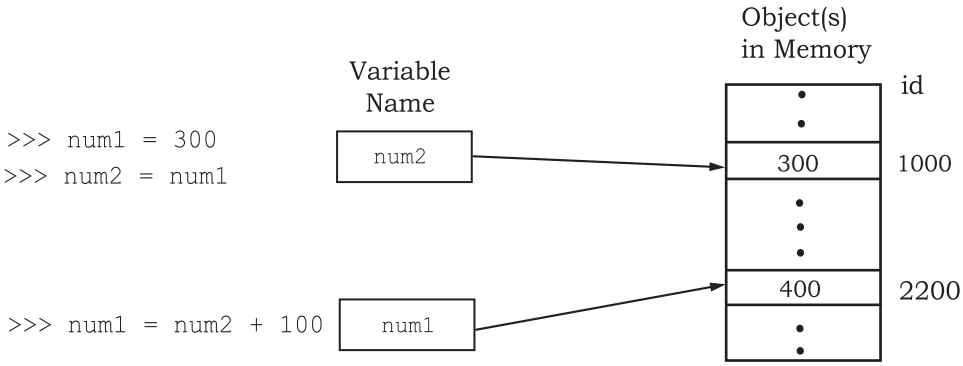


Figure 5.10: Variables with different values have different identifiers

```
>>> num1
= num2 +
100
```

This statement 1 `num1 = num2 + 100` links the variable `num1` to a new object stored at memory location number say 2200 having a value 400. As `num1` is an integer, which is an immutable type, it is rebuilt, as shown in Figure 5.10.

Python compares strings lexicographically, using ASCII value of the characters. If the first character of both the strings are same, the second character is compared, and so on.

5.7.7 Deciding Usage of Python Data Types

It is preferred to use lists when we need a simple iterable collection of data that may go for frequent modifications. For example, if we store the names of students of a class in a list, then it is easy to update the list when some new students join or some leave the course. Tuples are used when we do not need any change in the data. For example, names of months in a year. When we need uniqueness of elements and to avoid duplicacy it is preferable to use sets, for example, list of artefacts in a museum. If our data is being constantly modified or we need a fast lookup based on a custom key or we need a logical association between the key : value pair, it is advised to use dictionaries. A mobile phone book is a good application of dictionary.

5.8 OPERATORS

An operator is used to perform specific mathematical or logical operation on values. The values that the operators work on are called operands. For example, in the expression `10 + num`, the value `10`, and the variable `num` are operands and the + (plus) sign is an operator. Python supports several kinds of operators whose categorisation is briefly explained in this section.

5.8.1 Arithmetic Operators

Python supports arithmetic operators that are used to perform the four basic arithmetic operations as well as modular division, floor division and exponentiation.

Table 5.3 Arithmetic Operators in Python

Operator	Operation	Description	Example (Try in Lab)

+	Addition	Adds the two numeric values on either side of the operator
		
		This operator can also be used to concatenate two strings on either side of the operator
			>>> num1 = 5
			>>> num2 = 6
			>>> num1 + num2
			11
			>>> str1 = "Hello"
			>>> str2 = "India"
			>>> str1 + str2
			'HelloIndia'

-	Subtraction	Subtracts the operand on the right from the operand on the left	>>> num1 = 5
			>>> num2 = 6
			>>> num1 - num2
			-1

*	Multiplication	Multiplies the two values on both side of the operator
		
		Repeats the item on left of the operator if first operand is a string and second operand is an integer value
			>>> num1 = 5
			>>> num2 = 6
			>>> num1 * num2
			30
			>>> str1 = 'India'
			>>> str1 * 2
			'IndiaIndia'

/	Division	Divides the operand on the left by the operand on the right and returns the quotient	>>> num1 = 8
			>>> num2 = 4
			>>> num2 / num1
			0.5

%	Modulus	Divides the operand on the left by the operand on the right and returns the remainder	>>> num1 = 13
			>>> num2 = 5
			>>> num1 % num2
			3

//	Floor Division	Divides the operand on the left by the operand on the right and returns the quotient by removing the decimal part. It is sometimes also called integer division.	>>> num1 = 13
			>>> num2 = 4
			>>> num1 // num2
			3
			>>> num2 // num1
			0

**	Exponent	Performs exponential (power) calculation on operands. That is, raise the operand on the left to the power of the operand on the right	>>> num1 = 3
			>>> num2 = 4
			>>> num1 ** num2
			81

5.8.2 Relational Operators

Relational operator compares the values of the operands on its either side and determines the relationship among

them. Assume the Python variables `num1 = 10, num2 = 0, num3 = 10, str1 = "Good", str2 = "Afternoon"` for the following examples:

Table 5.4 Relational operators in Python

Operator	Operation	Description	Example (Try in Lab)

==	Equals to	If the values of two operands are equal, then the condition is True, otherwise it is False	`>>> num1 == num2`
			`False`
			`>> str1 == str2`
			`False`

!=	Not equal to	If values of two operands are not equal, then condition is True, otherwise it is False	`>>> num1 != num2`
			`True`
			`>>> str1 != str2`
			`True`
			`>>> num1 != num3`
			`False`

>	Greater than	If the value of the left-side operand is greater than the value of the right-side operand, then condition is True, otherwise it is False	`>>> num1 > num2`
			`True`
			`>>> str1 > str2`
			`True`

<	Less than	If the value of the left-side operand is less than the value of the right-side operand, then condition is True, otherwise it is False	`>>> num1 < num3`
			`False`
			`>>> str2 < str1`
			`True`

>=	Greater than or equal to	If the value of the left-side operand is greater than or equal to the value of the right-side operand, then condition is True, otherwise it is False	`>>> num1 >= num2`
			`True`
			`>>> num2 >= num3`
			`False`
			`>>> str1 >= str2`
			`True`

<=	Less than or equal to	If the value of the left operand is less than or equal to the value of the right operand, then is True otherwise it is False	`>>> num1 <= num2`
			`False`
			`>>> num2 <= num3`
			`True`
			`>>> str1 <= str2`
			`False`

5.8.3 Assignment Operators

Assignment operator assigns or changes the value of the variable on its left.

Table 5.5 Assignment operators in Python

Operator	Description	Example (Try in Lab)

=	Assigns value from right-side operand to left-side operand	`>>> num1 = 2`
		`>>> num2 = num1`
		`>>> num2`
		`2`
		`>>> country = 'India'`
		`>>> country`
		`'India'`

+=	It adds the value of right-side operand to the left-side operand and assigns the result to the left-side operand
	Note: x += y is same as x = x + y
		>>> num1 = 10
		>>> num2 = 2
		>>> num1 += num2
		>>> num1
		12
		>>> num2
		2
		>>> str1 = 'Hello'
		>>> str2 = 'India'
		>>> str1 += str2
		>>> str1
		'HelloIndia'

-=	It subtracts the value of right-side operand from the left-side operand and assigns the result to left-side operand
	Note: x -= y is same as x = x - y
		>>> num1 = 10
		>>> num2 = 2
		>>> num1 -= num2
		>>> num1
		8

*=	It multiplies the value of right-side operand with the value of left-side operand and assigns the result to left-side operand
	Note: x *= y is same as x = x * y
		>>> num1 = 2
		>>> num2 = 3
		>>> num1 *= 3
		
		>>> num1
		6
		>>> a = 'India'
		>>> a *= 3
		>>> a
		'IndiaIndiaIndia'

/=	It divides the value of left-side operand by the value of right-side operand and assigns the result to left-side operand
	Note: x /= y is same as x = x / y
		>>> num1 = 6
		>>> num2 = 3
		>>> num1 /= num2
		>>> num1
		2.0

%=	It performs modulus operation using two operands and assigns the result to left-side operand
	Note: x %= y is same as x = x % y
		>>> num1 = 7
		>>> num2 = 3
		>>> num1 %= num2
		>>> num1
		1

//=	It performs floor division using two operands and assigns the result to left-side operand
	Note: x //= y is same as x = x // y
		>>> num1 = 7
		>>> num2 = 3
		>>> num1 //= num2
		>>> num1
		2

**=	It performs exponential (power) calculation on operators and assigns value to the left-side operand
	Note: x **= y is same as x = x ** y
		>>> num1 = 2
		>>> num2 = 3
		>>> num1 **= num2
		>>> num1
		8

5.8.4 Logical Operators

There are three logical operators supported by Python. These operators (`and, or, not`) are to be written in lower case only. The logical operator evaluates to either `True` or `False` based on the logical operands on either side. Every value is logically either `True` or `False`. By default, all values are `True` except `None`, `False`, 0 (zero), empty collections "", (), [], {}, and few other special values. So if we say `num1 = 10, num2 = -20`, then both `num1` and `num2` are logically `True`.

Table 5.6 Logical operators in Python

Operator	Operation	Description	Example (Try in Lab)

`and`	Logical AND	If both the operands are `True`, then condition becomes `True`	`>>> True and True`
			`True`
			`>>> num1 = 10`
			`>>> num2 = -20`
			`>>> bool(num1 and num2)`
			`True`
			`>>> True and False`
			`False`
			`>>> num3 = 0`
			`>>> bool(num1 and num3)`
			`False`
			`>>> False and False`
			`False`

`or`	Logical OR	If any of the two operands are `True`, then condition becomes `True`	`>>> True or True`
			`True`
			`>>> True or False`
			`True`
			`>>> bool(num1 or num3)`
			`True`
			`>>> False or False`
			`False`

`not`	Logical NOT	Used to reverse the logical state of its operand	`>>> num1 = 10`
			`>>> bool(num1)`
			`True`
			`>>> not num1`
			`>>> bool(num1)`
			`False`

5.8.5 Identity Operators

Identity operators are used to determine whether the value of a variable is of a certain type or not. Identity operators can also be used to determine whether two

variables are referring to the same object or not. There are two identity operators.

Table 5.7 Identity operators in Python

Operator	Description	Example (Try in Lab)

is	Evaluates `True` if the variables on either side of the operator point towards the same memory location and `False` otherwise. `var1` is `var2` results to `True` if id(`var1`) is equal to id(`var2`)	`>>> num1 = 5`
		`>>> type(num1) is int`
		`True`
		`>>> num2 = num1`
		`>>> id(num1)`
		`1433920576`
		`>>> id(num2)`
		`1433920576`
		`>>> num1 is num2`
		`True`

is not	Evaluates to `False` if the variables on either side of the operator point to the same memory location and `True` otherwise. `var1` is not `var2` results to `True` if id(`var1`) is not equal to id(`var2`)	`>>> num1 is not num2` `False`

5.8.6 Membership Operators

Membership operators are used to check if a value is a member of the given sequence or not.

Table 5.8 Membership operators in Python

Operator	Description	Example (Try in Lab)

in	Returns `True` if the variable/value is found in the specified sequence and `False` otherwise	`>>> a = [1,2,3]`
		`>>> 2 in a`
		`True`
		`>>> '1' in a`
		`False`

not in	Returns `True` if the variable/value is not found in the specified sequence and `False` otherwise	`>>> a = [1,2,3]`
		`>>> 10 not in a`
		`True`
		`>>> 1 not in a`
		`False`

5.9 EXPRESSIONS

An expression is defined as a combination of constants, variables, and operators. An expression always evaluates to a value. A value or a standalone variable is also considered as an expression but a standalone operator is not an expression. Some examples of valid expressions are given below.

```
(i) 100                    (iv) 3.0 + 3.14
(ii) num                    (v) 23/3 -5 * 7(14 -2)
(iii) num - 20.4           (vi) "Global" + "Citizen"
```

5.9.1 Precedence of Operators

Evaluation of the expression is based on precedence of operators. When an expression contains different kinds of operators, precedence determines which operator should be applied first. Higher precedence operator is evaluated before the lower precedence operator. Most of the operators studied till now are binary operators. Binary operators are operators with two operands. The unary operators need only one operand, and they have a higher precedence than the binary operators. The minus (-) as well as + (plus) operators can act as both unary and binary operators, but `not` is a unary logical operator.

```
#Depth is using - (minus) as unary operator
Value = -Depth
#not is a unary operator, negates True
print(not(True))
```

The following table lists precedence of all operators from highest to lowest.

Table 5.9 Precedence of all operators in Python

Order of Precedence	Operators	Description
1	**	Exponentiation (raised to the power)
2	~ ,+, -	Complement, unary plus and unary minus
3	* ,/, %, //	Multiply, divide, modulo and floor division
4	+, -	Addition and subtraction
5	<= ,< ,> ,>=	Relational operators
6	== ,!=	Equality operators
7	=, %=, /=, //=, -=, +=, *=, **=	Assignment operators
8	is, is not	Identity operators
9	in, not in	Membership operators
10	not, or, and	Logical operators

Note:

a) Parenthesis can be used to override the precedence of operators. The expression within () is evaluated first.

b) For operators with equal precedence, the expression is evaluated from left to right.

Example 5.9 How will Python evaluate the following expression?

```
20 + 30 * 40
```

NOTES

Solution:

```
= 20 + (30 * 40)          #Step 1
```
#precedence of * is more than that of +
```
= 20 + 1200               #Step 2
= 1220                    #Step 3
```

Example 5.10 How will Python evaluate the following expression?

```
20 - 30 + 40
```

Solution:

The two operators (–) and (+) have equal precedence. Thus, the first operator, i.e., subtraction is applied before the second operator, i.e., addition (left to right).

```
= (20 - 30) + 40    #Step 1
= -10 + 40          #Step 2
= 30                #Step 3
```

Example 5.11 How will Python evaluate the following expression?

```
(20 + 30) * 40
```

Solution:

```
= (20 + 30) * 40          # Step 1
```
#using parenthesis(), we have forced precedence of + to be more than that of *
```
= 50 * 40                 # Step 2
= 2000                    # Step 3
```

Example 5.12 How will the following expression be evaluated in Python?

```
15.0 / 4 + (8 + 3.0)
```

Solution:

```
= 15.0 / 4 + (8.0 + 3.0)  #Step 1
= 15.0 / 4.0 + 11.0       #Step 2
= 3.75 + 11.0             #Step 3
= 14.75                   #Step 4
```

5.10 STATEMENT

In Python, a statement is a unit of code that the Python interpreter can execute.

Example 5.13

```
>>> x = 4              #assignment statement
>>> cube = x ** 3      #assignment statement
>>> print (x, cube)    #print statement
4 64
```

5.11 INPUT AND OUTPUT

Sometimes, a program needs to interact with the user's to get some input data or information from the end user and process it to give the desired output. In Python, we have the `input()` function for taking the user input. The `input()` function prompts the user to enter data. It accepts all user input as string. The user may enter a number or a string but the `input()` function treats them as strings only. The syntax for `input()` is:

```
input ([Prompt])
```

`Prompt` is the string we may like to display on the screen prior to taking the input, and it is optional. When a prompt is specified, first it is displayed on the screen after which the user can enter data. The `input()` takes exactly what is typed from the keyboard, converts it into a string and assigns it to the variable on left-hand side of the assignment operator (=). Entering data for the input function is terminated by pressing the enter key.

Example 5.14

```
>>> fname = input("Enter your first name: ")
Enter your first name: Arnab
>>> age = input("Enter your age: ")
Enter your age: 19
>>> type(age)
<class 'str'>
```

The variable `fname` will get the string 'Arnab', entered by the user. Similarly, the variable age will get the string '19'. We can typecast or change the datatype of the string data accepted from user to an appropriate numeric value. For example, the following statement will convert the accepted string to an integer. If the user enters any non-numeric value, an error will be generated.

Example 5.15

```
#function int() to convert string to integer
>>> age = int( input("Enter your age:"))
Enter your age: 19
>>> type(age)
<class 'int'>
```

Python uses the `print()` function to output data to standard output device — the screen. We will learn about function in Chapter 7. The function `print()` evaluates the expression before displaying it on the screen. The `print()`

Observe that a plus sign does not add any space between the two strings while a comma inserts a space between two strings in a print statement.

outputs a complete line and then moves to the next line for subsequent output. The syntax for `print()` is:

```
print(value [, ..., sep = ' ', end = '\n'])
```

- `sep`: The optional parameter `sep` is a separator between the output values. We can use a character, integer or a string as a separator. The default separator is space.
- `end`: This is also optional and it allows us to specify any string to be appended after the last value. The default is a new line.

Example 5.16

Statement	Output
`print("Hello")`	`Hello`
`print(10*2.5)`	`25.0`
`print("I" + "love" + "my" + "country")`	`Ilovemycountry`
`print("I'm", 16, "years old")`	`I'm 16 years old`

The third print function in the above example is concatenating strings, and we use + (plus) between two strings to concatenate them. The fourth print function also appears to be concatenating strings but uses commas (,) between strings. Actually, here we are passing multiple arguments, separated by commas to the print function. As arguments can be of different types, hence the print function accepts integer (16) along with strings here. But in case the print statement has values of different types and '+' is used instead of comma, it will generate an error as discussed in the next section under explicit conversion.

5.12 TYPE CONVERSION

Consider the following program

```
num1 = input("Enter a number and I'll double
it: ")
num1 = num1 * 2
print(num1)
```

The program was expected to display double the value of the number received and store in variable `num1`. So if a user enters 2 and expects the program to display 4 as the output, the program displays the following result:

```
Enter a number and I'll double it: 2
22
```

NOTES

This is because the value returned by the input function is a string ("2") by default. As a result, in statement `num1 = num1 * 2`, `num1` has string value and `*` acts as repetition operator which results in output as "22". To get `4` as output, we need to convert the data type of the value entered by the user to integer. Thus, we modify the program as follows:

```
num1 = input("Enter a number and I'll double
it: ")
num1 = int(num1)   #convert string input to
                   #integer
num1 = num1 * 2
print(num1)
```

Now, the program will display the expected output as follows:

```
Enter a number and I'll double it: 2
4
```

Let us now understand what is type conversion and how it works. 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 to another type; or implicitly, when the interpreter understands such a need by itself and does the type conversion automatically.

5.12.1 Explicit Conversion

Explicit conversion, also called type casting happens when data type conversion takes place because the programmer forced 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 loss of information since we are forcing an expression to be of a specific type. For example, converting a floating value of `x = 20.67` into an integer type, i.e., `int(x)` will discard the fractional part `.67`. Following are some of the functions in Python that are used for explicitly converting an expression or a variable to a different type.

Table 5.10 Explicit type conversion functions in Python

Function	Description
`int(x)`	Converts x to an integer
`float(x)`	Converts x to a floating-point number

`str(x)`	Converts x to a string representation
`chr(x)`	Converts x to a character
`unichr(x)`	Converts x to a Unicode character

Program 5-5 Program of explicit type conversion from int to float.

```
#Program 5-5
#Explicit type conversion from int to float
num1 = 10
num2 = 20
num3 = num1 + num2
print(num3)
print(type(num3))
num4 = float(num1 + num2)
print(num4)
print(type(num4))
```

Output:

```
30
<class 'int'>
30.0
<class 'float'>
```

Program 5-6 Program of explicit type conversion from float to int.

```
#Program 5-6
#Explicit type conversion from float to int
num1 = 10.2
num2 = 20.6
num3 = (num1 + num2)
print(num3)
print(type(num3))
num4 = int(num1 + num2)
print(num4)
print(type(num4))
```

Output:

```
30.8
<class 'float'>
30
<class 'int'>
```

Program 5-7 Example of type conversion between numbers and strings.

```
#Program 5-7
#Type Conversion between Numbers and Strings
priceIcecream = 25
priceBrownie = 45
totalPrice = priceIcecream + priceBrownie
print("The total is Rs." + totalPrice )
```

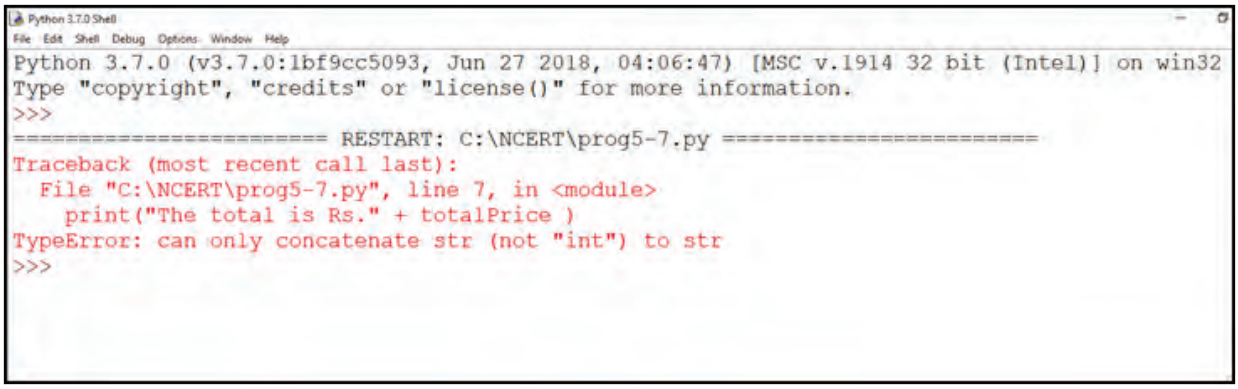


Figure 5.11: Output of program 5-7

On execution, program 5-7 gives an error as shown in Figure 5.11, informing that the interpreter cannot convert an integer value to string implicitly. It may appear quite intuitive that the program should convert the integer value to a string depending upon the usage. However, the interpreter may not decide on its own when to convert as there is a risk of loss of information. Python provides the mechanism of the explicit type conversion so that one can clearly state the desired outcome. Program 5-8 works perfectly using explicit type casting:

Program 5-8 Program to show explicit type casting.

```
#Program 5-8
#Explicit type casting
priceIcecream = 25
priceBrownie = 45
totalPrice = priceIcecream + priceBrownie
print("The total in Rs." + str(totalPrice))
```

Output:

```
The total in Rs.70
```

Similarly, type casting is needed to convert float to string. In Python, one can convert string to integer or float values whenever required.

Program 5-9 Program to show explicit type conversion.

```
#Program 5-9
#Explicit type conversion
icecream = '25'
brownie = '45'
#String concatenation
price = icecream + brownie
print("Total Price Rs." + price)
#Explicit type conversion - string to integer
```

```
price = int(icecream)+int(brownie)
print("Total Price Rs." + str(price))
```

Output:

```
Total Price Rs.2545
Total Price Rs.70
```

5.12.2 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.

Program 5-10 Program to show implicit conversion from int to float.

```
#Program 5-10
#Implicit type conversion from int to float

num1 = 10            #num1 is an integer
num2 = 20.0          #num2 is a float
sum1 = num1 + num2   #sum1 is sum of a float
and an integer
print(sum1)
print(type(sum1))
```

Output:

```
30.0
<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 was automatically converted to a float value stored in variable `sum1` without explicitly telling the interpreter. This is an example of implicit data conversion. One may wonder why was the float value not converted to an integer instead? This 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.

5.13 DEBUGGING

A programmer can make mistakes while writing a program, and hence, the program may not execute or may generate wrong output. The process of identifying and removing such mistakes, also known as bugs or errors, from a program is called debugging. Errors occurring in programs can be categorised as:

i) Syntax errors

ii) Logical errors

iii) Runtime errors

5.13.1 Syntax Errors

Like other programming languages, Python has its own rules that determine its syntax. The interpreter interprets the statements only if it is syntactically (as per the rules of Python) correct. If any syntax error is present, the interpreter shows error message(s) and stops the execution there. For example, parentheses must be in pairs, so the expression (10 + 12) is syntactically correct, whereas (7 + 11 is not due to absence of right parenthesis. Such errors need to be removed before the execution of the program

5.13.2 Logical Errors

A logical error is a bug in the program that causes it to behave incorrectly. A logical error produces an undesired output but without abrupt termination of the execution of the program. Since the program interprets successfully even when logical errors are present in it, it is sometimes difficult to identify these errors. The only evidence to the existence of logical errors is the wrong output. While working backwards from the output of the program, one can identify what went wrong.

For example, if we wish to find the average of two numbers 10 and 12 and we write the code as 10 + 12/2, it would run successfully and produce the result 16. Surely, 16 is not the average of 10 and 12. The correct code to find the average should have been (10 + 12)/2 to give the correct output as 11.

Logical errors are also called semantic errors as they occur when the meaning of the program (its semantics) is not correct.

5.13.3 Runtime Error

A runtime error causes abnormal termination of program while it is executing. Runtime error is when the statement is correct syntactically, but the interpreter cannot execute it. Runtime errors do not appear until after the program starts running or executing.

For example, we have a statement having division operation in the program. By mistake, if the denominator entered is zero then it will give a runtime error like "division by zero".

Let us look at the program 5-11 showing two types of runtime errors when a user enters non-integer value

or value '0'. The program generates correct output when the user inputs an integer value for num2.

Program 5-11 Example of a program which generates runtime error.

```
#Program 5-11
#Runtime Errors Example
num1 = 10.0
num2 = int(input("num2 = "))
#if user inputs a string or a zero, it leads
to runtime error
print(num1/num2)
```

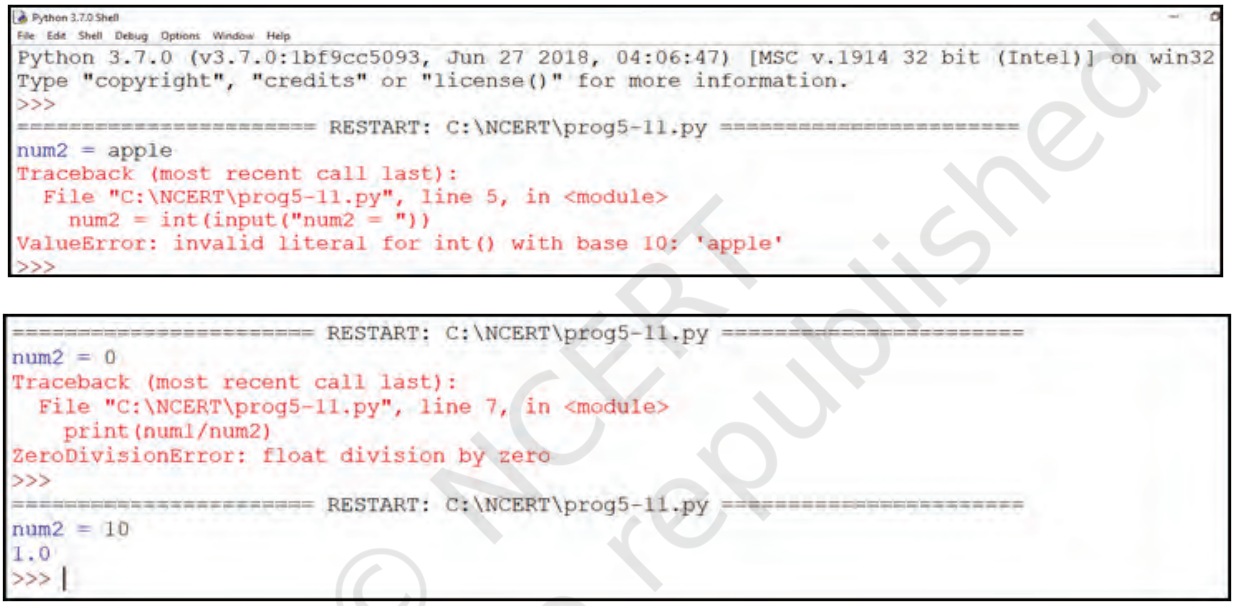


Figure 5.11: Output of program 5-11

SUMMARY

- Python is an open-source, high level, interpreter-based language that can be used for a multitude of scientific and non-scientific computing purposes.
- Comments are non-executable statements in a program.
- An identifier is a user defined name given to a variable or a constant in a program.
- The process of identifying and removing errors from a computer program is called debugging.
- Trying to use a variable that has not been assigned a value gives an error.
- There are several data types in Python — integer, boolean, float, complex, string, list, tuple, sets, None and dictionary.

NOTES

- Datatype conversion can happen either explicitly or implicitly.
- Operators are constructs that manipulate the value of operands. Operators may be unary or binary.
- An expression is a combination of values, variables and operators.
- Python has `input()` function for taking user input.
- Python has `print()` function to output data to a standard output device.

EXERCISE

1. Which of the following identifier names are invalid and why?

i	`Serial_no.`	v	`Total_Marks`
ii	`1st_Room`	vi	`total-Marks`
iii	`Hundred$`	vii	`_Percentage`
iv	`Total Marks`	viii	`True`

2. Write the corresponding Python assignment statements:
   a) Assign 10 to variable `length` and 20 to variable `breadth`.
   b) Assign the average of values of variables `length` and `breadth` to a variable `sum`.
   c) Assign a list containing strings 'Paper', 'Gel Pen', and 'Eraser' to a variable `stationery`.
   d) Assign the strings 'Mohandas', 'Karamchand', and 'Gandhi' to variables `first`, `middle` and `last`.
   e) Assign the concatenated value of string variables `first`, `middle` and `last` to variable `fullname`. Make sure to incorporate blank spaces appropriately between different parts of names.
3. Write logical expressions corresponding to the following statements in Python and evaluate the expressions (assuming variables `num1`, `num2`, `num3`, `first`, `middle`, `last` are already having meaningful values):
   a) The sum of 20 and –10 is less than 12.
   b) `num3` is not more than 24.

NOTES

c) 6.75 is between the values of integers num1 and num2.

d) The string 'middle' is larger than the string 'first' and smaller than the string 'last'.

e) List Stationery is empty.

4. Add a pair of parentheses to each expression so that it evaluates to True.

a) `0 == 1 == 2`

b) `2 + 3 == 4 + 5 == 7`

c) `1 < -1 == 3 > 4`

5. Write the output of the following:

a)
```
num1 = 4
num2 = num1 + 1
num1 = 2
print (num1, num2)
```

b)
```
num1, num2 = 2, 6
num1, num2 = num2, num1 + 2
print (num1, num2)
```

c)
```
num1, num2 = 2, 3
num3, num2 = num1, num3 + 1
print (num1, num2, num3)
```

6. Which data type will be used to represent the following data values and why?

a) Number of months in a year

b) Resident of Delhi or not

c) Mobile number

d) Pocket money

e) Volume of a sphere

f) Perimeter of a square

g) Name of the student

h) Address of the student

7. Give the output of the following when num1 = 4, num2 = 3, num3 = 2

a)
```
num1 += num2 + num3
print (num1)
```

b)
```
num1 = num1 ** (num2 + num3)
print (num1)
```

c)
```
num1 **= num2 + num3
```

d)
```
num1 = '5' + '5'
print(num1)
```

NOTES

e) `print(4.00/(2.0+2.0))`

f)
```
num1 = 2+9*((3*12)-8)/10
print(num1)
```

g)
```
num1 = 24 // 4 // 2
print(num1)
```

h)
```
num1 = float(10)
print (num1)
```

i)
```
num1 = int('3.14')
print (num1)
```

j) `print('Bye'  == 'BYE')`

k) `print(10 != 9 and 20 >= 20)`

l) `print(10 + 6 * 2 ** 2 != 9//4 -3 and 29 >= 29/9)`

m) `print(5 % 10 + 10 < 50 and 29 <= 29)`

n) `print((0 < 6) or (not(10 == 6) and (10<0)))`

8. Categorise the following as syntax error, logical error or runtime error:

   a) `25 / 0`

   b) `num1 = 25; num2 = 0; num1/num2`

9. A dartboard of radius 10 units and the wall it is hanging on are represented using a two-dimensional coordinate system, with the board's center at coordinate (0,0). Variables x and y store the x-coordinate and the y-coordinate of a dart that hits the dartboard. Write a Python expression using variables x and y that evaluates to `True` if the dart hits (is within) the dartboard, and then evaluate the expression for these dart coordinates:

   a) `(0,0)`

   b) `(10,10)`

   c) `(6, 6)`

   d) `(7,8)`

10. Write a Python program to convert temperature in degree Celsius to degree Fahrenheit. If water boils at 100 degree C and freezes as 0 degree C, use the program to find out what is the boiling point and freezing point of water on the Fahrenheit scale.

    (Hint: T(°F) = T(°C) × 9/5 + 32)

11. Write a Python program to calculate the amount payable if money has been lent on simple interest.

NOTES

Principal or money lent = P, Rate of interest = R% per annum and Time = T years. Then Simple Interest (SI) = (P x R x T)/ 100.

Amount payable = Principal + SI.

P, R and T are given as input to the program.

12. Write a program to calculate in how many days a work will be completed by three persons A, B and C together. A, B, C take x days, y days and z days respectively to do the job alone. The formula to calculate the number of days if they work together is xyz/(xy + yz + xz) days where x, y, and z are given as input to the program.
13. Write a program to enter two integers and perform all arithmetic operations on them.
14. Write a program to swap two numbers using a third variable.
15. Write a program to swap two numbers without using a third variable.
16. Write a program to repeat the string "GOOD MORNING" n times. Here 'n' is an integer entered by the user.
17. Write a program to find average of three numbers.
18. The volume of a sphere with radius r is $4/3\pi r^3$. Write a Python program to find the volume of spheres with radius 7cm, 12cm, 16cm, respectively.
19. Write a program that asks the user to enter their name and age. Print a message addressed to the user that tells the user the year in which they will turn 100 years old.
20. The formula $E = mc^2$ states that the equivalent energy (E) can be calculated as the mass (m) multiplied by the speed of light (c = about 3×10^8 m/s) squared. Write a program that accepts the mass of an object and determines its energy.
21. Presume that a ladder is put upright against a wall. Let variables length and angle store the length of the ladder and the angle that it forms with the ground as it leans against the wall. Write a Python program to compute

Notes

the height reached by the ladder on the wall for the following values of length and angle:

a) 16 feet and 75 degrees
b) 20 feet and 0 degrees
c) 24 feet and 45 degrees
d) 24 feet and 80 degrees

Case Study-based Question

Schools use "Student Management Information System" (SMIS) to manage student-related data. This system provides facilities for:

- recording and maintaining personal details of students.
- maintaining marks scored in assessments and computing results of students.
- keeping track of student attendance.
- managing many other student-related data. Let us automate this process step by step.

Identify the personal details of students from your school identity card and write a program to accept these details for all students of your school and display them in the following format.

Name of School

Student Name: PQR Roll No: 99
Class: XI Section: A
Address : Address Line 1
Address Line 2

City: ABC Pin Code: 999999
Parent's/ Guardian's Contact No: 9999999999

Documentation Tips

It is a fact that a properly documented program is easy to read, understand and is flexible for future development. Therefore, it is important that one pays extra attention to documentation while coding. Let us assess the documentation done by us in our case study program and also find out whether our friends also pay similar attention to documentation or not.

NOTES

Following is a checklist of good documentation points:

- Objective of the program is clearly stated in the beginning.
- Objective of each function is clearly mentioned in the beginning of each function.
- Comments are inserted at the proper place so as to enhance the understandability and readability of the program.
  (***Note:*** Over commenting doesn't help)
- Variables and function names are meaningful and appropriate.
- Single letter variable names are not used.
- Program name is meaningful.
  (***Note:*** It is not proper to use your name as program name, for example, 'raman. py' or 'namya.py' to denote your program code. It is more appropriate to use the program name as 'bankingProject.py' for a banking related program or 'admProcess' for an admission related program.)
- Program code is properly indented.
- Same naming conventions are used throughout the program.
  (***Note:*** Some of the naming conventions are firstNum, first_num, to denote the variable having first number).

Let's do this exercise for our peer's case studies as well and provide a feedback to them.

A relevant peer feedback helps in improving the documentation of the projects. It also helps in identifying our mistakes and enriches us with better ideas used by others.

CBSE Class 11 Study Material

- Printable Worksheets for Class 11

NCERT Solutions for Class 11

- NCERT Solutions for class 11 Maths
- NCERT Solutions for class 11 Physics
- NCERT Solutions for class 11 Chemistry
- NCERT Solutions for class 11 Biology
- NCERT Solutions for class 11 English
- NCERT Solutions for Class 11 English Woven Words Essay
- NCERT Solutions for Class 11 English Woven Short Stories
- NCERT Solutions for Class 11 English Woven Words Poetry
- NCERT Solutions for class 11 Accountancy
- NCERT Solutions for class 11 Business Studies
- NCERT Solutions for class 11 Economics
- NCERT Solutions for class 11 Computer Science – Python
- Class 11 Hindi Aroh (आरोह भाग 1)
- Class 11 Hindi Vitan (वितान भाग 1)

- Class 11 Sanskrit
- Class 11 History
- Class 11 Geography
- Class 11 Indian Economic Development
- Class 11 Statistics for Economics
- Class 11 Political Science
- Class 11 Psychology
- Class 11 Sociology
- Class 11 Entrepreneurship

- **Maths formulas for Class 11**
- Hindi Grammar for Class 11
- Class 11 English Hornbill Summaries
- Class 11 English Snapshots Summaries
- CBSE Sample Papers for Class 11
- NCERT Exemplar Class 11 Maths Solutions
- NCERT Exemplar Class 11 Physics Solutions
- NCERT Exemplar Class 11 Chemistry Solutions
- NCERT Exemplar Class 11 Biology Solutions
- RD Sharma Class 11 Solutions
- **CBSE Class 11 and 12 Revised Syllabus**
- MCQ Questions

- CBSE Class 11 Physics Manual
- CBSE Class 11 Chemistry Manual
- Trigonometry Formulas
- Integration Formulas
- JEE Main Study Material
- NEET Study Material

- CBSE Class 11 Notes
- Class 11 Maths Notes
- Class 11 Physics Notes
- Class 11 Chemistry Notes
- Class 11 Biology Notes
- Class 11 English Notes
- Class 11 English Woven Words Short Stories
- CBSE Class 11 English Woven Words Essay
- CBSE Class 11 English Woven Words Poetry
- CBSE Class 11 English Snapshots
- CBSE Class 11 English Hornbill
- Class 11 Business Studies Notes
- Class 11 Accountancy Notes
- Class 11 Psychology Notes
- Class 11 Entrepreneurship Notes
- Class 11 Economics Notes

- Class 11 Indian Economic Development Notes
- Statistics for Economics Class 11 Notes
- Class 11 Political Science Notes
- Class 11 History Notes
- Sociology Class 11 Notes
- Geography Class 11 Notes

NCERT Books for Class 11

- Class 11 NCERT Maths Books
- Class 11 Physics NCERT Book
- Class 11 Chemistry NCERT Book
- Class 11 Biology NCERT Book
- Class 11 Political Theory Part-I
- Class 11 NCERT Business Studies Books
- Class 11 India Constitution at Work
- NCERT Geography Book Class 11
- NCERT Class 11 History Book
- Class 11 India Economic Development
- Class 11 NCERT English Books
- NCERT Sanksrit Books Class 11
- Class 11 Computer and Communication Technology Book
- Class 11 NCERT Accountancy Books

- Class 11 Statistics
- Class 11 Introduction to Psychology
- Class 11 Introducing Sociology
- Class 11 Understanding Society
- Class 11 Fine Arts
- Class 11 Heritage Craft Books
- Class 11 Nai Awaz
- Class 11 Dhanak
- Class 11 The story of Graphic Design
- Class 11 Human Ecology and Family Sciences

Chapter 6

Flow of Control

6.1 Introduction

"Don't you hate code that's not properly indented? Making it [indenting] part of the syntax guarantees that all code is properly indented."

– G. van Rossum

In Figure 6.1, we see a bus carrying the children to school. There is only one way to reach the school. The driver has no choice, but to follow the road one milestone after another to reach the school. We learnt in Chapter 5 that this is the concept of sequence, where Python executes one statement after another from beginning to the end of the program. These are the kind of programs we have been writing till now.

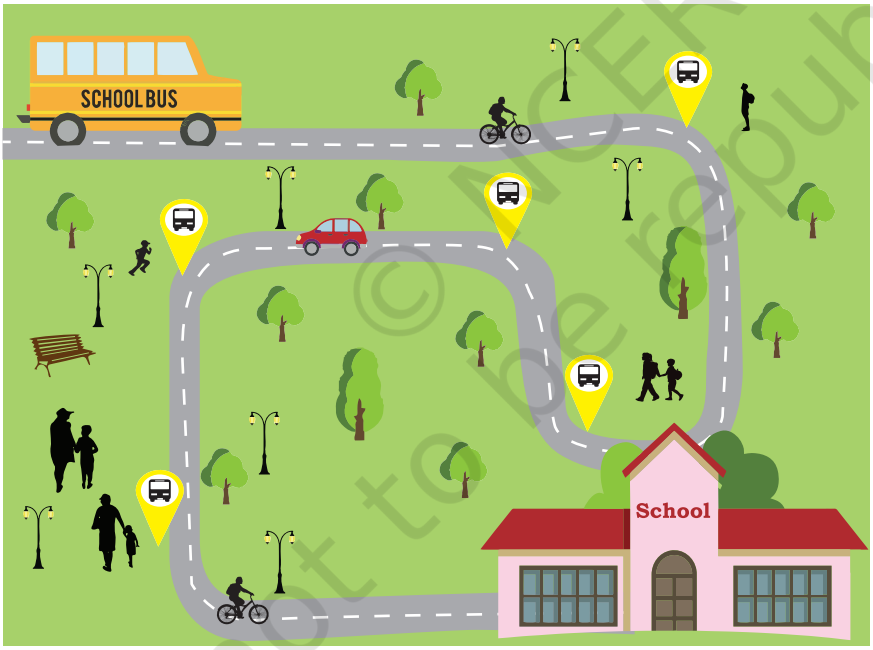


Figure 6.1: Bus carrying students to school

Let us consider a program 6-1 that executes in sequence, that is, statements are executed in an order in which they are written.

The order of execution of the statements in a program is known as flow of control. The flow of control can be implemented using control structures. Python supports two types of control structures—selection and repetition.

In this chapter

- » *Introduction to Flow of Control*
- » *Selection*
- » *Indentation*
- » *Repetition*
- » *Break and Continue Statements*
- » *Nested Loops*

Program 6-1 Program to print the difference of two numbers.

```
#Program 6-1
#Program to print the difference of two input numbers
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
diff = num1 - num2
print("The difference of",num1,"and",num2,"is",diff)
```

Output:

```
Enter first number 5
Enter second number 7
The difference of 5 and 7 is -2
```

6.2 SELECTION

Now suppose we have ₹10 to buy a pen. On visiting the stationery shop, there are a variety of pens priced at ₹10 each. Here, we have to decide which pen to buy. Similarly, when we use the direction services of a digital map, to reach from one place to another, we notice that sometimes it shows more than one path like the least crowded path, shortest distance path, etc. We decide the path as per our priority. A decision involves selecting from one of the two or more possible options. In programming, this concept of decision making or selection is implemented with the help of `if..else` statement.

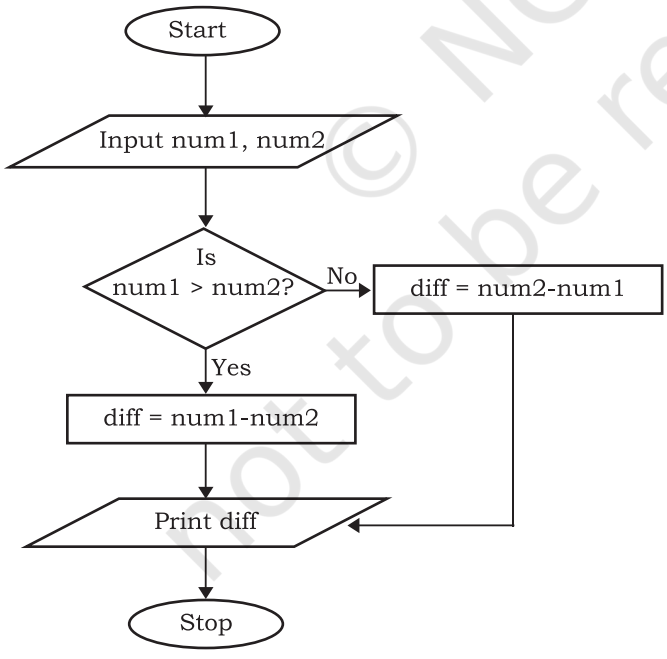


Figure 6.2: Flow chart depicting decision making

Now, suppose we want to display the positive difference of the two numbers num1 and num2 given at program 6-1. For that, we need to modify our approach. Look at the flowchart shown in Figure 6.2 that subtracts the smaller number from the bigger number so that we always get a positive difference. This selection is based upon the values that are input for the two numbers num1 and num2.

The syntax of `if` statement is:

```
if condition:
     statement(s)
```

In the following example, if the age entered by the user is greater than 18, then print that the user is eligible to vote. If the condition is true, then the indented statement(s) are executed. The indentation implies that its execution is dependent on the condition. There is no limit on the number of statements that can appear as a block under the `if` statement.

Example 6.1

```
age = int(input("Enter your age "))
if age >= 18:
    print("Eligible to vote")
```

A variant of `if` statement called `if..else` statement allows us to write two alternative paths and the control condition determines which path gets executed. The syntax for `if..else` statement is as follows.

```
if condition:
     statement(s)
else:
     statement(s)
```

Let us now modify the example on voting with the condition that if the age entered by the user is greater than 18, then to display that the user is eligible to vote. Otherwise display that the user is not eligible to vote.

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

Now let us use the same concept to modify program 6-1, so that it always gives a positive difference as the output. From the flow chart in Figure 6.2, it is clear that we need to decide whether num1 > num2 or not and take action accordingly.

We have to specify two blocks of statements since num1 can be greater than num2 or vice-versa as shown in program 6-2.

Many a times there are situations that require multiple conditions to be checked and it may lead to many alternatives. In such cases we can chain the conditions using `if..elif` (`elif` means `else..if`).

Program 6-2 Program to print the positive difference of two numbers.

```
#Program 6-2
#Program to print the positive difference of two numbers
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
if num1 > num2:
     diff = num1 - num2
else:
     diff = num2 - num1
print("The difference of",num1,"and",num2,"is",diff)
```

Output:

```
Enter first number: 5
Enter second number: 6
The difference of 5 and 6 is 1
```

The syntax for a selection structure using `elif` is as shown below.

```
if condition:
     statement(s)
elif condition:
     statement(s)
elif condition:
     statement(s)
else:
     statement(s)
```

Example 6.2 Check whether a number is positive, negative, or zero.

```
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")
```

Example 6.3 Display the appropriate message as per the colour of signal at the road crossing.

```
signal = input("Enter the 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!")
```

Number of `elif` is dependent on the number of conditions to be checked. If the first condition is false, then the next condition is checked, and so on. If one of the conditions is true, then the corresponding indented block executes, and the `if` statement terminates.

Let us write a program to create a simple calculator to perform basic arithmetic operations on two numbers. The program should do the following:

- Accept two numbers from the user.
- Ask user to input any of the operator (+, -, *, /). An error message is displayed if the user enters anything else.
- Display only positive difference in case of the operator "-".
- Display a message "Please enter a value other than 0" if the user enters the second number as 0 and operator '/' is entered.

Program 6-3 Write a program to create a simple calculator performing only four basic operations.

```
#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 one 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("Error! Division by zero is not allowed. Program
terminated")
    else:
        result = val1/val2
else:
    print("Wrong input,program terminated")
print("The result is ",result)
```

Output:

```
Enter value 1: 84
Enter value 2: 4
Enter any one of the operator (+,-,*,/): /
The result is 21.0
```

In the program, 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.

6.3 INDENTATION

In most programming languages, the statements within a block are put inside curly brackets. However, Python uses indentation for block as well as for nested block structures. Leading whitespace (spaces and tabs) at the beginning of a statement is called indentation. In Python, the same level of indentation associates statements into a single block of code. The interpreter checks indentation levels very strictly and throws up syntax errors if indentation is not correct. It is a common practice to use a single tab for each level of indentation.

In the program 6-4, the `if-else` statement has two blocks of statements and the statements in each block are indented with the same amount of spaces or tabs.

Program 6-4 Program to find the larger of the two pre-specified numbers.

```
#Program 6-4
#Program to find larger of the two numbers
num1 = 5
num2 = 6
if num1 > num2:                          #Block1
        print("first number is larger")
        print("Bye")
else:                                    #Block2
        print("second number is larger")
        print("Bye Bye")
```

Output:

```
second number is larger
Bye Bye
```

6.4 Repetition

Often, we repeat a tasks, for example, payment of electricity bill, which is done every month. Figure 6.3 shows the life cycle of butterfly that involves four stages, i.e., a butterfly lays eggs, turns into a caterpillar, becomes a pupa, and finally matures as a butterfly. The cycle starts again with laying of eggs by the butterfly.

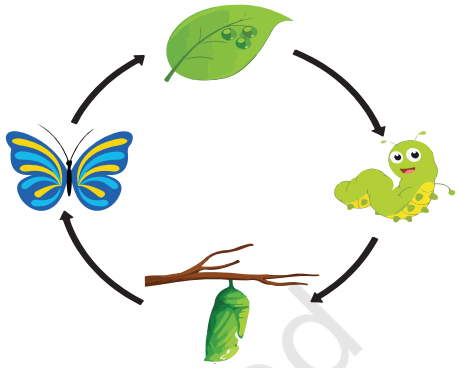


Figure 6.3: Iterative process occurring in nature

This kind of repetition is also called iteration. Repetition of a set of statements in a program is made possible using looping constructs. To understand further, let us look at the program 6-5.

Program 6-5 Write a program to print the first five natural numbers.

```
#Program 6-5
#Print first five natural numbers
print(1)
print(2)
print(3)
print(4)
print(5)
```

Output:

```
1
2
3
4
5
```

What should we do if we are asked to print the first 100,000 natural numbers? Writing 100,000 print statements would not be an efficient solution. It would be tedious and not the best way to do the task. Writing a program having a loop or repetition is a better solution. The program logic is given below:

1. Take a variable, say count, and set its value to 1.
2. Print the value of count.
3. Increment the variable (count += 1).

4. Repeat steps 2 and 3 as long as count has a value less than or equal to 100,000 (count <= 100,000).

Looping constructs provide the facility 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 particular logical condition remains true. This condition is checked based on the value of a variable called the loop's control variable. When the condition becomes false, the loop terminates. It is the responsibility of the programmer to ensure that this condition eventually does become false so that there is an exit condition and it does not become an infinite loop. For example, if we did not set the condition count <= 100000, the program would have never stopped. There are two looping constructs in Python - `for` and `while`.

6.4.1 The 'For' Loop

The `for` statement is used to iterate over a range of values or a sequence. The `for` loop is executed for each of the items in the range. These values can be either numeric, or, as we shall see in later chapters, they can be elements of a data type like a string, list, or tuple.

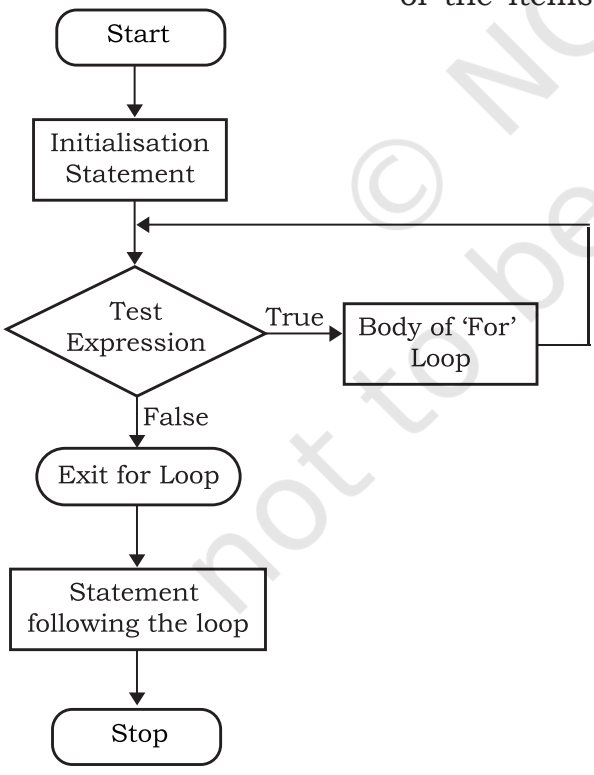


Figure 6.4: Flow chart of for loop

With every iteration of the loop, the control variable checks whether each of the values in the range have been traversed or not. When all the items in the range are exhausted, the statements within loop are not executed; the control is then transferred to the statement immediately following the `for` loop. While using `for` loop, it is known in advance the number of times the loop will execute. The flowchart depicting the execution of a `for` loop is given in Figure 6.4.

(A) Syntax of the For Loop

```
for <control-variable> in <sequence/
items in range>:
    <statements inside body of the
     loop>
```

Program 6-6 Program to print the characters in the string 'PYTHON' using `for` loop.

```
#Program 6-6
#Print the characters in word PYTHON using for loop
for letter in 'PYTHON':
    print(letter)
```

Output:

```
P
Y
T
H
O
N
```

Program 6-7 Program to print the numbers in a given sequence using `for` loop.

```
#Program 6-7
#Print the given sequence of numbers using for loop
count = [10,20,30,40,50]
for num in count:
    print(num)
```

Output:

```
10
20
30
40
50
```

Program 6-8 Program to print even numbers in a given sequence using `for` loop.

```
#Program 6-8
#Print even numbers in the given sequence
numbers = [1,2,3,4,5,6,7,8,9,10]
for num in numbers:
    if (num % 2) == 0:
        print(num,'is an even Number')
```

Output:

```
2  is an even Number
4  is an even Number
```

```
6  is an even Number
8  is an even Number
10 is an even Number
```

Note: Body of the loop is indented with respect to the `for` statement.

(B) The Range() Function

The range() is a built-in function in Python. Syntax of range() function is:

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

It is used to create a list containing a sequence of integers from the given start value upto stop value (excluding stop value), with a difference of the given step value. We will learn about functions in the next chapter. To begin with, simply remember that function takes parameters to work on. In function range(), start, stop and step are parameters.

The start and step parameters are optional. If start value is not specified, by default the list starts from 0. If step is also not specified, by default the value increases by 1 in each iteration. All parameters of range() function must be integers. The step parameter can be a positive or a negative integer excluding zero.

Example 6.4

```
#start and step not specified
>>> list(range(10)-)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

#default step value is 1
>>> list(range(2, 10))
[2, 3, 4, 5, 6, 7, 8, 9]

#step value is 5
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]

#step value is -1. Hence, decreasing
#sequence is generated
>>> range(0, -9, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8]
```

The function range() is often used in for loops for generating a sequence of numbers.

Program 6-9 Program to print the multiples of 10 for numbers in a given range.

```
#Program 6-9
#Print multiples of 10 for numbers in a given range
for num in range(5):
    if num > 0:
        print(num * 10)
```

Output:

```
10
20
30
40
```

6.4.2 The 'While' Loop

The `while` statement executes a block of code repeatedly as long as the control condition of the loop is true. The control condition of the `while` loop is executed before any statement inside the loop is executed. After each iteration, the control condition is tested again and the loop continues as long as the condition remains true. When this condition becomes false, the statements in the body of loop are not executed and the control is transferred to the statement immediately following the body of `while` loop. If the condition of the `while` loop is initially false, the body is not executed even once.

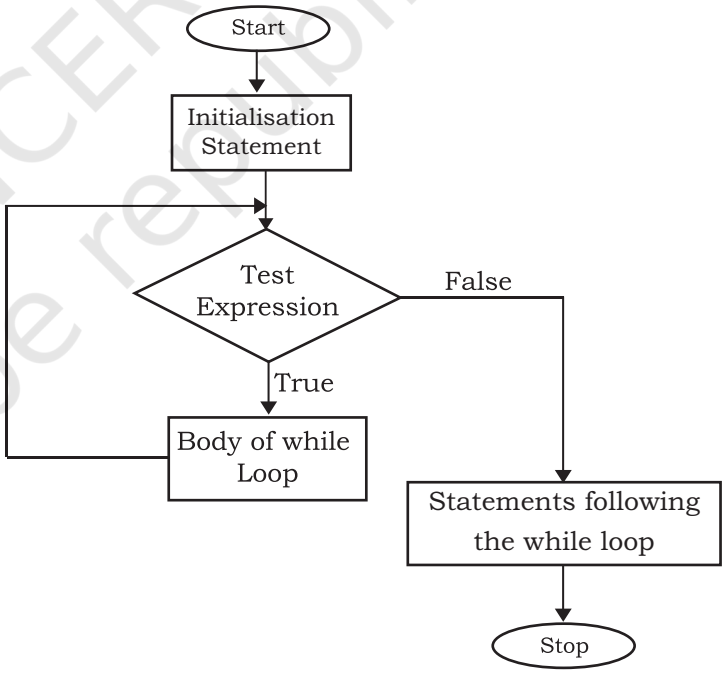


Figure 6.5: Flow chart of while Loop

The statements within the body of the `while` loop must ensure that the condition eventually becomes false; otherwise the loop will become an infinite loop, leading to a logical error in the program. The flowchart of `while` loop is shown in Figure 6.5.

Syntax of `while` Loop

```
while test_condition:
  body of while
```

Program 6-10 Program to print first 5 natural numbers using `while` loop.

```
#Program 6-10
#Print first 5 natural numbers using while loop
count = 1
while count <= 5:
     print(count)
     count += 1
```

Output:

```
1
2
3
4
5
```

Program 6-11 Program to find the factors of a whole number using `while` loop.

```
#Program 6-11
#Find the factors of a number using while loop
num = int(input("Enter a number to find its factor: "))
print (1, end=' ')   #1 is a factor of every number
factor = 2
while factor <= num/2 :
     if num % factor == 0:
#the optional parameter end of print function specifies the delimeter
#blank space(' ') to print next value on same line
          print(factor, end=' ')
     factor += 1
print (num, end=' ') #every number is a factor of itself
```

Output:

```
Enter a number to find its factors : 6
1 2 3 6
```

Note: Body of the loop is indented with respect to the `while` statement. Similarly, the statements within `if` are indented with respect to positioning of `if` statement.

6.5 BREAK AND CONTINUE STATEMENT

Looping constructs allow programmers to repeat tasks efficiently. In certain situations, when some particular condition occurs, we may want to exit from a loop (come

out of the loop forever) or skip some statements of the loop before continuing further in the loop. These requirements can be achieved by using `break` and `continue` statements, respectively. Python provides these statements as a tool to give more flexibility to the programmer to control the flow of execution of a program.

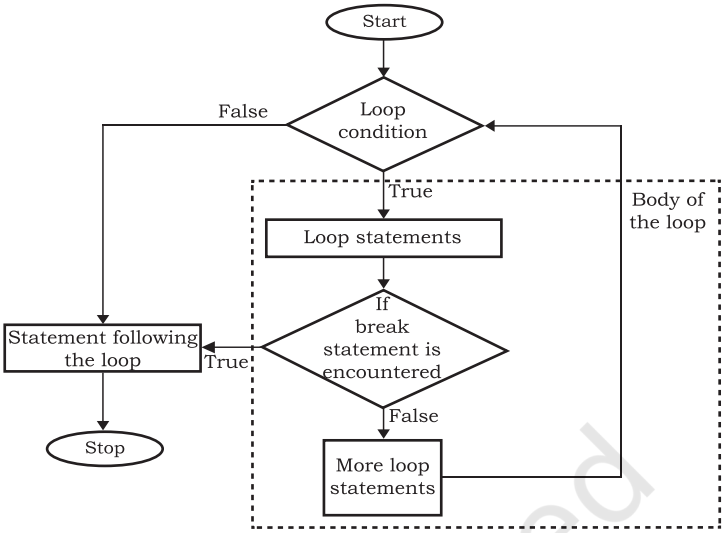


Figure 6.5: Flowchart for using break statement in loop

6.5.1 Break Statement

The `break` statement alters the normal flow of execution as it terminates the current loop and resumes execution of the statement following that loop.

Program 6-12 Program to demonstrate use of `break` statement.

```
#Program 6-12
#Program to demonstrate the use of break statement in loop
num = 0
for num in range(10):
     num = num + 1
     if num == 8:
          break
     print('Num has value ' + str(num))
print('Encountered break!! Out of loop')
```

Output:

```
Num has value 1
Num has value 2
Num has value 3
Num has value 4
Num has value 5
Num has value 6
Num has value 7
Encountered break!! Out of loop
```

Note: When value of `num` becomes 8, the `break` statement is executed and the `for` loop terminates.

Program 6-13 Find the sum of all the positive numbers entered by the user. As soon as the user enters a neagtive number, stop taking in any further input from the user and display the sum .

```
#Program 6-13
#Find the sum of all the positive numbers entered by the user
#till the user enters a negative number.
entry = 0
sum1 = 0
print("Enter numbers to find their sum, negative number ends the
loop:")
while True:
#int() typecasts string to integer
     entry = int(input())
     if (entry < 0):
          break
     sum1 += entry
print("Sum =", sum1)
```

Output:

```
Enter numbers to find their sum, negative number ends the loop:
3
4
5
-1
Sum = 12
```

Program 6-14 Program to check if the input number is prime or not.

```
#Program 6-14
#Write a Python program to check if a given number is prime or not.
num = int(input("Enter the number to be checked: "))
flag = 0                    #presume num is a prime number
if num > 1 :
     for i in range(2, int(num / 2)):
          if (num % i == 0):
               flag = 1      #num is a not prime number
               break         #no need to check any further

     if flag == 1:
          print(num , "is not a prime number")
     else:
          print(num , "is a prime number")
```

```
else :
          print("Entered number is <= 1, execute again!")
```

Output 1:

```
Enter the number to be checked: 20
20 is not a prime number
```

Output 2:

```
Enter the number to check: 19
19 is a prime number
```

Output 3:

```
Enter the number to check: 2
2 is a prime number
```

Output 4:

```
Enter the number to check: 1
Entered number is <= 1,  execute again!
```

6.5.2 Continue Statement

When a `continue` statement is encountered, the control skips the execution of remaining statements inside the body of the loop for the current iteration and jumps to the beginning of the loop for the next iteration. If the loop's condition is still true, the loop is entered again, else the control is transferred to the statement immediately following the loop. Figure 6.7 shows the flowchart of `continue` statement.

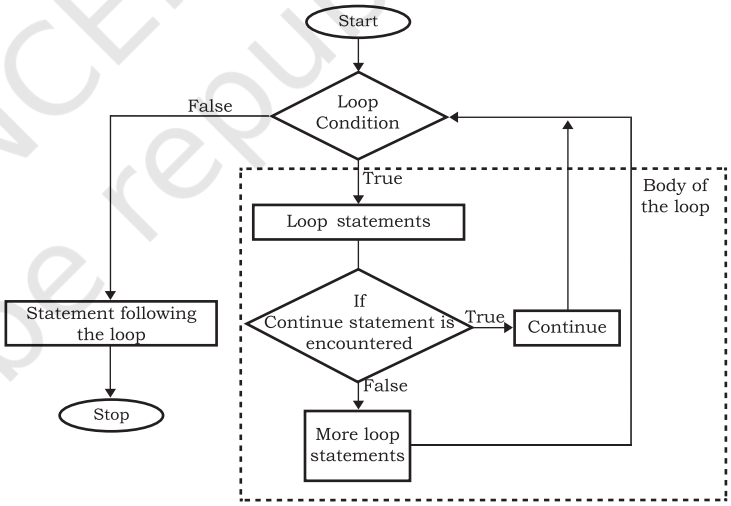


Figure 6.6: Flow chart of continue statement

Program 6-15 Program to demonstrate the use of continue statement.

```
#Program 6-15
#Prints values from 0 to 6 except 3
num = 0
for num in range(6):
     num = num + 1
     if num == 3:
```

```
            continue
        print('Num has value ' + str(num))
print('End of loop')
```

Output:

```
Num has value 1
Num has value 2
Num has value 4
Num has value 5
Num has value 6
End of loop
```

Observe that the value 3 is not printed in the output, but the loop continues after the `continue` statement to print other values till the `for` loop terminates.

6.6 NESTED LOOPS

A loop may contain another loop inside it. A loop inside another loop is called a nested loop.

Program 6-16 Program to demonstrate working of nested `for` loops.

```
#Program 6-16
#Demonstrate working of nested for loops
for var1 in range(3):
    print( "Iteration " + str(var1 + 1) + " of outer loop")
    for var2 in range(2):    #nested loop
        print(var2 + 1)
    print("Out of inner loop")
print("Out of outer loop")
```

Output:

```
Iteration 1 of outer loop
1
2
Out of inner loop
Iteration 2 of outer loop
1
2
Out of inner loop
Iteration 3 of outer loop
1
2
Out of inner loop
Out of outer loop
```

Python does not impose any restriction on how many loops can be nested inside a loop or on the levels of nesting. Any type of loop (`for/while`) may be nested within another loop (`for/while`).

Program 6-17 Program to print the pattern for a number input by the user.

```
#Program 6-17
#Program to print the pattern for a number input by the user
#The output pattern to be generated is
#1
#1 2
#1 2 3
#1 2 3 4
#1 2 3 4 5
num = int(input("Enter a number to generate its pattern = "))
for i in range(1,num + 1):
     for j in range(1,i + 1):
          print(j, end = " ")
     print()
```

Output:

```
Enter a number to generate its pattern = 5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Program 6-18 Program to find prime numbers between 2 to 50 using nested for loops.

```
#Program 6-18
#Use of nested loops to find the prime numbers between 2 to 50

num = 2
for i in range(2, 50):
     j= 2
     while ( j <= (i/2)):
          if (i % j == 0):      #factor found
               break            #break out of while loop
          j += 1
     if ( j > i/j) :           #no factor found
```

```
            print ( i, "is a prime number")
print ("Bye Bye!!")
```

Output:

```
2 is a prime number
3 is a prime number
5 is a prime number
7 is a prime number
11 is a prime number
13 is a prime number
17 is a prime number
19 is a prime number
23 is a prime number
29 is a prime number
31 is a prime number
37 is a prime number
41 is a prime number
43 is a prime number
47 is a prime number
Bye Bye!!
```

Program 6-19 Write a program to calculate the factorial of a given number.

```
#Program 6-19
#The following program uses a for loop nested inside an if..else
#block to calculate the factorial of a given number

num = int(input("Enter a number: "))
fact = 1
# check if the number is negative, positive or zero
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    for i in range(1, num + 1):
        fact = fact * i
    print("factorial of ", num, " is ", fact)
```

Output:

```
Enter a number: 5
Factorial of  5  is  120
```

Notes

Summary

- The `if` statement is used for selection or decision making.
- The looping constructs `while` and `for` allow sections of code to be executed repeatedly under some condition.
- `for` statement iterates over a range of values or a sequence.
- The statements within the body of `for` loop are executed till the range of values is exhausted.
- The statements within the body of a `while` are executed over and over until the condition of the `while` is false.
- If the condition of the `while` loop is initially false, the body is not executed even once.
- The statements within the body of the `while` loop must ensure that the condition eventually becomes false; otherwise, the loop will become an infinite loop, leading to a logical error in the program.
- The `break` statement immediately exits a loop, skipping the rest of the loop's body. Execution continues with the statement immediately following the body of the loop. When a `continue` statement is encountered, the control jumps to the beginning of the loop for the next iteration.
- A loop contained within another loop is called a nested loop.

Exercise

1. What is the difference between `else` and `elif` construct of `if` statement?
2. What is the purpose of range() function? Give one example.
3. Differentiate between `break` and `continue` statements using examples.
4. What is an infinite loop? Give one example.
5. Find the output of the following program segments:

(i)
```
a = 110
while a > 100:
    print(a)
    a -= 2
```

NOTES

(ii)
```
for i in range(20,30,2):
    print(i)
```

(iii)
```
country = 'INDIA'
for  i in country:
    print (i)
```

(iv)
```
i = 0; sum = 0
while i < 9:
    if i % 4 == 0:
        sum = sum + i
    i = i + 2
print (sum)
```

(v)
```
for x in range(1,4):
    for y in range(2,5):
        if x * y > 10:
            break
        print (x * y)
```

(vi)
```
var = 7
while var > 0:
    print ('Current variable value: ', var)
    var = var -1
    if var == 3:
        break
    else:
        if var == 6:
            var = var -1
            continue
    print ("Good bye!")
```

PROGRAMMING EXERCISES

1. Write a program that takes the name and age of the user as input and displays a message whether the user is eligible to apply for a driving license or not. (the eligible age is 18 years).
2. Write a function to print the table of a given number. The number has to be entered by the user.
3. Write a program that prints minimum and maximum of five numbers entered by the user.
4. Write a program to check if the year entered by the user is a leap year or not.
5. Write a program to generate the sequence: –5, 10, –15, 20, –25..... upto n, where n is an integer input by the user.
6. Write a program to find the sum of 1+ 1/8 + 1/27.....$1/n^3$, where n is the number input by the user.

NOTES

7. Write a program to find the sum of digits of an integer number, input by the user.
8. Write a function that checks whether an input number is a palindrome or not.

   [***Note***: A number or a string is called palindrome if it appears same when written in reverse order also. For example, 12321 is a palindrome while 123421 is not a palindrome]
9. Write a program to print the following patterns:

i)	* * * * * * * * * * * * *	ii)	1 2 1 2 3 2 1 2 3 4 3 2 1 2 3 4 5 4 3 2 1 2 3 4 5
iii)	1 2 3 4 5 1 2 3 4 1 2 3 1 2 1	iv)	* * * * * * * *

10. Write a program to find the grade of a student when grades are allocated as given in the table below.

Percentage of Marks	Grade
Above 90%	A
80% to 90%	B
70% to 80%	C
60% to 70%	D
Below 60%	E

Percentage of the marks obtained by the student is input to the program.

CASE STUDY-BASED QUESTIONS

Let us add more functionality to our SMIS developed in Chapter 5.

6.1 Write a menu driven program that has options to

- accept the marks of the student in five major subjects in Class X and display the same.
- calculate the sum of the marks of all subjects. Divide the total marks by number of subjects (i.e. 5), calculate percentage = total marks/5 and display the percentage.

NOTES

- Find the grade of the student as per the following criteria:

Criteria	Grade
percentage > 85	A
percentage < 85 && percentage >= 75	B
percentage < 75 && percentage >= 50	C
percentage > 30 && percentage <= 50	D
percentage <30	Reappear

Let's peer review the case studies of others based on the parameters given under "DOCUMENTATION TIPS" at the end of Chapter 5 and provide a feedback to them.

CBSE Class 11 Study Material

- Printable Worksheets for Class 11

NCERT Solutions for Class 11

- NCERT Solutions for class 11 Maths
- NCERT Solutions for class 11 Physics
- NCERT Solutions for class 11 Chemistry
- NCERT Solutions for class 11 Biology
- NCERT Solutions for class 11 English
- NCERT Solutions for Class 11 English Woven Words Essay
- NCERT Solutions for Class 11 English Woven Short Stories
- NCERT Solutions for Class 11 English Woven Words Poetry
- NCERT Solutions for class 11 Accountancy
- NCERT Solutions for class 11 Business Studies
- NCERT Solutions for class 11 Economics
- NCERT Solutions for class 11 Computer Science – Python
- Class 11 Hindi Aroh (आरोह भाग 1)
- Class 11 Hindi Vitan (वितान भाग 1)

- Class 11 Sanskrit
- Class 11 History
- Class 11 Geography
- Class 11 Indian Economic Development
- Class 11 Statistics for Economics
- Class 11 Political Science
- Class 11 Psychology
- Class 11 Sociology
- Class 11 Entrepreneurship

- **Maths formulas for Class 11**
- Hindi Grammar for Class 11
- Class 11 English Hornbill Summaries
- Class 11 English Snapshots Summaries
- CBSE Sample Papers for Class 11
- NCERT Exemplar Class 11 Maths Solutions
- NCERT Exemplar Class 11 Physics Solutions
- NCERT Exemplar Class 11 Chemistry Solutions
- NCERT Exemplar Class 11 Biology Solutions
- RD Sharma Class 11 Solutions
- **CBSE Class 11 and 12 Revised Syllabus**
- MCQ Questions

- CBSE Class 11 Physics Manual
- CBSE Class 11 Chemistry Manual
- Trigonometry Formulas
- Integration Formulas
- JEE Main Study Material
- NEET Study Material

- CBSE Class 11 Notes
- Class 11 Maths Notes
- Class 11 Physics Notes
- Class 11 Chemistry Notes
- Class 11 Biology Notes
- Class 11 English Notes
- Class 11 English Woven Words Short Stories
- CBSE Class 11 English Woven Words Essay
- CBSE Class 11 English Woven Words Poetry
- CBSE Class 11 English Snapshots
- CBSE Class 11 English Hornbill
- Class 11 Business Studies Notes
- Class 11 Accountancy Notes
- Class 11 Psychology Notes
- Class 11 Entrepreneurship Notes
- Class 11 Economics Notes

- Class 11 Indian Economic Development Notes
- Statistics for Economics Class 11 Notes
- Class 11 Political Science Notes
- Class 11 History Notes
- Sociology Class 11 Notes
- Geography Class 11 Notes

NCERT Books for Class 11

- Class 11 NCERT Maths Books
- Class 11 Physics NCERT Book
- Class 11 Chemistry NCERT Book
- Class 11 Biology NCERT Book
- Class 11 Political Theory Part-I
- Class 11 NCERT Business Studies Books
- Class 11 India Constitution at Work
- NCERT Geography Book Class 11
- NCERT Class 11 History Book
- Class 11 India Economic Development
- Class 11 NCERT English Books
- NCERT Sanksrit Books Class 11
- Class 11 Computer and Communication Technology Book
- Class 11 NCERT Accountancy Books

- Class 11 Statistics
- Class 11 Introduction to Psychology
- Class 11 Introducing Sociology
- Class 11 Understanding Society
- Class 11 Fine Arts
- Class 11 Heritage Craft Books
- Class 11 Nai Awaz
- Class 11 Dhanak
- Class 11 The story of Graphic Design
- Class 11 Human Ecology and Family Sciences

Chapter 7

Functions

7.1 Introduction

Till now we have written some programs and might have realised that as the problem gets complex, the number of lines in a program increase, which makes the program look bulky and difficult to manage. Consider the following problem statement:

There is a company that manufactures tents as per user's requirements. The shape of the tent is cylindrical surmounted by a conical top.

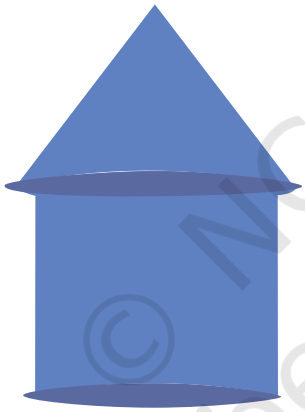


Figure 7.1: Shape of a tent

The company performs the following tasks to fix the selling price of each tent.

1. Accept user requirements for the tent, such as
   a) height
   b) radius
   c) slant height of the conical part
2. Calculate the area of the canvas used
3. Calculate the cost of the canvas used for making the tent
4. Calculate the net payable amount by the customer that is inclusive of the 18% tax

The company has created a computer program for quick and accurate calculation for the payable amount as shown in program 7-1.

"Once you succeed in writing the programs for [these] complicated algorithms, they usually run extremely fast. The computer doesn't need to understand the algorithm, its task is only to run the programs."

– R. Tarjan

In this chapter

- *Introduction to Functions*
- *User Defined Functions*
- *Scope of a Variable*
- *Python Standard Library*

Program 7-1 Program to calculate the payable amount for the tent.

```
#Program 7-1
#Program to calculate the payable amount for the tent without
#functions

print( "Enter values for the cylindrical part of the tent in
meters\n")
h = float(input("Enter height of the cylindrical part: "))
r = float(input("Enter radius: "))

l = float(input("Enter the slant height of the conical part in
meters: "))
csa_conical = 3.14*r*l          #Area of conical part
csa_cylindrical = 2*3.14*r*h    #Area of cylindrical part

# Calculate area of the canvas used for making the tent
canvas_area = csa_conical + csa_cylindrical
print("The area of the canvas is",canvas_area,"m^2")

#Calculate cost of the canvas
unit_price = float(input("Enter the cost of 1 m^2 canvas: "))
total_cost= unit_price * canvas_area
print("The total cost of canvas = ",total_cost)

#Add tax to the total cost to calculate net amount payable by the
#customer
tax = 0.18 * total_cost;
net_price = total_cost + tax
print("Net amount payable = ",net_price)
```

Another approach to solve the above problem is to divide the program into different blocks of code as shown in Figure 7.2.

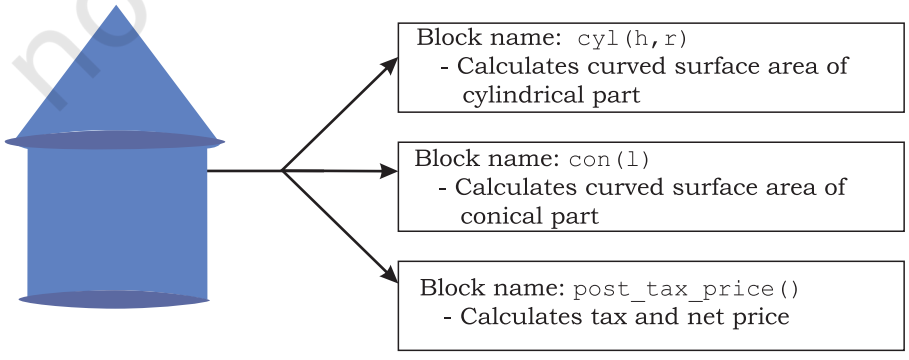


Figure 7.2: Calculation of the cost of the tent

The process of dividing a computer program into separate independent blocks of code or separate sub-problems with different names and specific functionalities is known as modular programming. In this chapter, we will learn about the benefits of this approach.

7.2 FUNCTIONS

In programming, the use of function is one of the means to achieve modularity and reusability. Function can be defined as a named group of instructions that accomplish a specific task when it is invoked. Once defined, a function can be called repeatedly from different places of the program without writing all the codes of that function everytime, or it can be called from inside another function, by simply writing the name of the function and passing the required parameters, if any (Section 7.3). The programmer can define as many functions as desired while writing the code. The program 7-1 is rewritten using user defined functions as shown in program 7-2.

Program 7-2 Program to calculate the payable amount for the tent using user defined functions.

```
#Program 7-2
#Program to calculate the cost of tent
#function definition
def cyl(h,r):
     area_cyl = 2*3.14*r*h     #Area of cylindrical part
     return(area_cyl)

#function definition
def con(l,r):
     area_con = 3.14*r*l       #Area of conical part
     return(area_con)

#function definition
def post_tax_price(cost):      #compute payable amount for the tent
     tax = 0.18 * cost;
     net_price = cost + tax
     return(net_price)

print("Enter values of cylindrical part of the tent in meters:")
h = float(input("Height: "))
r = float(input("Radius: "))
```

```
 csa_cyl = cyl(h,r)  #function call

l = float(input("Enter slant height of the conical area in meters: "))
csa_con = con(l,r)  #function call

#Calculate area of the canvas used for making the tent
canvas_area = csa_cyl + csa_con
print("Area of canvas = ",canvas_area," m^2")

#Calculate cost of canvas
unit_price = float(input("Enter cost of 1 m^2 canvas in rupees: "))
total_cost = unit_price * canvas_area
print("Total cost of canvas before tax = ",total_cost)
print("Net amount payable (including tax) = ",post_tax_price(total_
cost))
```

If we compare program 7-1 and 7-2, it is evident that program 7-2 looks more organised and easier to read.

7.2.1 The Advantages of Function

Suppose in further the company decides to design another type of tent whose base is rectangular, while the upper part remains the same. In such a scenario, some part of the existing code can be reused by calling the function `con(l,r)`. If the company develops other products or provides services, and where 18% tax rate is to be applied, the programmer can use the function `post_tax_price(cost)`directly.

Thus, following are the advantages of using functions in a program:

- Increases readability, particularly for longer code as by using functions, the program is better organised and easy to understand.
- Reduces code length as same code is not required to be written at multiple places in a program. This also makes debugging easier.
- Increases reusability, as function can be called from another function or another program. Thus, we can reuse or build upon already defined functions and avoid repetitions of writing the same piece of code.
- Work can be easily divided among team members and completed in parallel.

7.3 USER DEFINED FUNCTIONS

Taking advantage of reusability feature of functions, there is a large number of functions already available

in Python under standard library (section 7.5). We can directly call these functions in our program without defining them. However, in addition to the standard library functions, we can define our own functions while writing the program. Such functions are called user defined functions. Thus, a function defined to achieve some task as per the programmer's requirement is called a user defined function.

7.3.1 Creating User Defined Function

A function definition begins with `def` (short for define). The syntax for creating a user defined function is as follows:

```
def<Function name> ([parameter 1, parameter 2,....]):
    set of instructions to be executed
    [return <value>]
```

Function Header

Function Body (Should be indented within the function header)

- The items enclosed in "[ ]" are called parameters and they are optional. Hence, a function may or may not have parameters. Also, a function may or may not return a value.
- Function header always ends with a colon (:).
- Function name should be unique. Rules for naming identifiers also applies for function naming.
- The statements outside the function indentation are not considered as part of the function.

Program 7-3 Write a user defined function to add 2 numbers and display their sum.

```
#Program 7-3
#Function to add two numbers
#The requirements are listed below:
  #1. We need to accept 2 numbers from the user.
  #2. Calculate their sum
  #3. Display the sum.

#function definition
def addnum():
    fnum = int(input("Enter first number: "))
    snum = int(input("Enter second number: "))
    sum = fnum + snum
    print("The sum of ",fnum,"and ",snum,"is ",sum)

#function call
addnum()
```

In order to execute the function addnum(), we need to call it. The function can be called in the program by writing function name followed by () as shown in the last line of program 7-3.

Output:

```
Enter first number: 5
Enter second number: 6
The sum of 5 and 6 is 11
```

7.3.2 Arguments and Parameters

In the above example, the numbers were accepted from the user within the function itself, but it is also possible for a user defined function to receive values at the time of being called. An argument is a value passed to the function during the function call which is received in corresponding parameter defined in function header.

Program 7-4 Write a program using a user defined function that displays sum of first *n* natural numbers, where *n* is passed as an argument.

```
#Program 7-4
#Program to find the sum of first n natural numbers
#The requirements are:
  #1. n be passed as an argument
  #2. Calculate sum of first n natural numbers
  #3. Display the sum

#function header
def sumSquares(n):          #n is the parameter
     sum = 0
     for i in range(1,n+1):
          sum = sum + i
     print("The sum of first",n,"natural numbers is: ",sum)

num = int(input("Enter the value for n: "))
#num is an argument referring to the value input by the user
sumSquares(num)          #function call
```

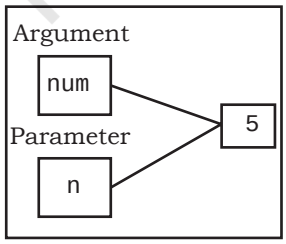


Figure 7.3: Both argument and parameter refers to the same value

Let us assume that the user has input 5 during the execution of the program 7-4. So, num refers to the value 5. It is then used as an argument in the function:

```
sumSquares(num)
```

Since the function is called, the control is transferred to execute the function

```
def sumSquares(n):
```

where parameter n also refers to the value 5 which num is referring to as shown in Figure 7.3.

Since both num and n are referring to the same value, they are bound to have the same identity. We can use the id() function to find the identity of the object that the argument and parameter are referring to. Let us understand this with the help of the following example.

Program 7-5 Write a program using user defined function that accepts an integer and increments the value by 5. Also display the id of argument (before function call), id of parameter before increment and after increment.

```
#Program 7-5
#Function to add 5 to a user input number
#The requirements are listed below:
  #1. Display the id()of argument before function call.
  #2. The function should have one parameter to accept the argument
  #3. Display the value and id() of the parameter.
  #4. Add 5 to the parameter
  #5. Display the new value and id()of the parameter to check
      #whether the parameter is assigned a new memory location or
      #not.
def incrValue(num):
    #id of Num before increment
    print("Parameter num has value:",num,"\nid =",id(num))
    num = num + 5
    #id of Num after increment
    print("num incremented by 5 is",num,"\nNow id is ",id(num))
number = int(input("Enter a number: "))
print("id of argument number is:",id(number))          #id of Number
incrValue(number)
```

Output:

```
Enter a number: 8
id of argument number is: 1712903328
Parameter num has value: 8
id = 1712903328
num incremented by 5 is 13
Now id is 1712903408
```

number and num have the same id

The id of Num has changed.

Let us understand the above output through illustration (see Figure 7.4):

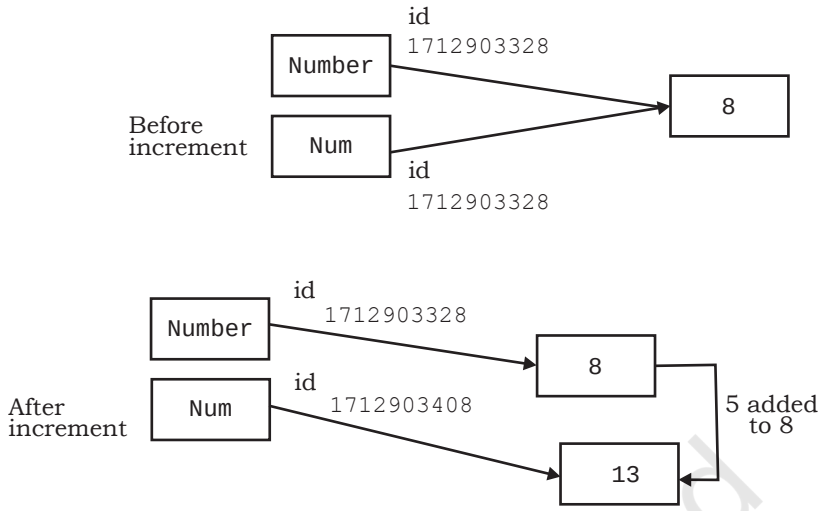


Figure 7.4: ID of argument and parameter before and after increment

Both argument and parameter can have the same name as shown in program 7-6.

Program 7-6 Write a program using a user defined function myMean() to calculate the mean of floating values stored in a list.

```
#Program 7-6
#Function to calculate mean
#The requirements are listed below:
  #1. The function should have 1 parameter (list containing floating
      #point values)
  #2. To calculate mean by adding all the numbers and dividing by
      #total number of elements

def myMean(myList): #function to compute means of values in list
     total = 0
     count = 0
     for i in myList:
          total = total + i          #Adds each element i to total
          count = count + 1          #Counts the number of elements
     mean = total/count              #mean is calculated
     print("The calculated mean is:",mean)
myList = [1.3,2.4,3.5,6.9]
#Function call with list "myList" as an argument
myMean(myList)
```

Output:

```
The calculated mean is: 3.5250000000000004
```

Program 7-7 Write a program using a user defined function calcFact() to calculate and display the factorial of a number num passed as an argument.

```
#Program 7-7
#Function to calculate factorial
#The requirements are listed below:
  #1. The function should accept one integer argument from user.
  #2. Calculate factorial. For example:
  #3. Display factorial

def calcFact(num):
     fact = 1
     for i in range(num,0,-1):
          fact = fact * i
     print("Factorial of",num,"is",fact)

num = int(input("Enter the number: "))
calcFact(num)
```

Output:

```
Enter the number: 5
Factorial of 5 is 120
```

Note: Since multiplication is commutative 5! = 5*4*3*2*1 = 1*2*3*4*5

(A) String as Parameters

In programs 7-5 to 7-7, the arguments passed are of numeric type only. However, in some programs, user may need to pass string values as an argument, as shown in program 7-8.

Program 7-8 Write a program using a user defined function that accepts the first name and lastname as arguments, concatenate them to get full name and displays the output as:

```
Hello full name
```

For example, if first name is Gyan and lastname is Vardhan, the output should be:

```
Hello Gyan Vardhan
```

```
#Program 7-8
#Function to display full name
#The requirements are listed below:
   #1. The function should have 2 parameters to accept first name and
       #last name.
   #2. Concatenate names using + operator with a space between first
       #name and last name.
   #3. Display full name.
```

```
def fullname(first,last):
#+ operator is used to concatenate strings
     fullname = first + " " + last
     print("Hello",fullname)
#function ends here
first = input("Enter first name: ")
last = input("Enter last name: ")
#function call
fullname(first,last)
```

Output:

```
Enter first name: Gyan
Enter last name: Vardhan
Hello Gyan Vardhan
```

(B) Default Parameter

Python allows assigning a default value to the parameter. A default value is a value that is predecided and assigned to the parameter when the function call does not have its corresponding argument.

Program 7-9 Write a program that accepts numerator and denominator of a fractional number and calls a user defined function mixedFraction() when the fraction formed is not a proper fraction. The default value of denominator is 1. The function displays a mixed fraction only if the fraction formed by the parameters does not evaluate to a whole number.

```
#Program 7-9
#Function to display mixed fraction for an improper fraction
#The requirements are listed below:
  #1. Input numerator and denominator from the user.
  #2. Check if the entered numerator and denominator form a proper
      #fraction.
  #3. If they do not form a proper fraction, then call
      #mixedFraction().
  #4. mixedFraction()display a mixed fraction only when the fraction
      #does not evaluate to a whole number.

def mixedFraction(num,deno = 1):
     remainder = num % deno
#check if the fraction does not evaluate to a whole number
     if remainder!= 0:
          quotient = int(num/deno)
          print("The mixed fraction=", quotient,"(",remainder, "/",
deno,")")
     else:
```

```
            print("The given fraction evaluates to a whole number")
#function ends here
num = int(input("Enter the numerator: "))
deno = int(input("Enter the denominator: "))
print("You entered:",num,"/",deno)
if num > deno:    #condition to check whether the fraction is
improper
     mixedFraction(num,deno)      #function call
else:
     print("It is a proper fraction")
```

Output:

```
Enter the numerator: 17
Enter the denominator: 2
You entered: 17 / 2
The mixed fraction =  8 ( 1 / 2 )
```

In the above program, the denominator entered is 2, which is passed to the parameter "deno" so the default value of the argument deno is overwritten.

Let us consider the following function call:

```
mixedFraction(9)
```

Here, num will be assigned 9 and deno will use the default value 1.

***Note*:**

- A function argument can also be an expression, such as

```
mixedFraction(num+5,deno+5)
```

In such a case, the argument is evaluated before calling the function so that a valid value can be assigned to the parameter.

- The parameters should be in the same order as that of the arguments.

The default parameters must be the trailing parameters in the function header that means if any parameter is having default value then all the other parameters to its right must also have default values. For example,

```
def mixedFraction(num,deno = 1)
def mixedFraction(num = 2,deno = 1)
```

Let us consider few more function definition headers:

```
#incorrect as default must be the last
#parameter
def calcInterest(principal = 1000, rate,
time = 5):
#correct
def calcInterest(rate,principal = 1000,
time = 5):
```

7.3.3 Functions Returning Value

A function may or may not return a value when called. The return statement returns the values from the function. In the examples given so far, the function performs calculations and display result(s).They do not return any value. Such functions are called void functions. But a situation may arise, wherein we need to send value(s) from the function to its calling function. This is done using `return` statement.

The `return` statement does the following:

- returns the control to the calling function.
- return value(s) or `None`.

Program 7-10 Write a program using user defined function calcPow() that accepts base and exponent as arguments and returns the value $Base^{exponent}$ where Base and exponent are integers.

```
#Program 7-10
#Function to calculate and display base raised to the power exponent
#The requirements are listed below:
   #1. Base and exponent are to be accepted as arguments.
   #2. Calculate Base^exponent
   #3. Return the result (use return statement )
   #4. Display the returned value.
def calcpow(number,power):          #function definition
     result = 1
     for i in range(1,power+1):
          result = result * number
     return result

base = int(input("Enter the value for the Base: "))
expo = int(input("Enter the value for the Exponent: "))
answer = calcpow(base,expo)          #function call
print(base,"raised to the power",expo,"is",answer)
```

Output:

```
Enter the value for the Base: 5
Enter the value for the Exponent: 4
5 raised to the power 4 is 625
```

So far we have learnt that a function may or may not have parameter(s) and a function may or may not return any value(s). In Python, as per our requirements, we can have the function in either of the following ways:

- Function with no argument and no return value
- Function with no argument and with return value(s)
- Function with argument(s) and no return value

- Function with argument(s) and return value(s)

7.3.4 Flow of Execution

Flow of execution can be defined as the order in which the statements in a program are executed. The Python interpreter starts executing the instructions in a program from the first statement. The statements are executed one by one, in the order of appearance from top to bottom.

When the interpreter encounters a function definition, the statements inside the function are not executed until the function is called. Later, when the interpreter encounters a function call, there is a little deviation in the flow of execution. In that case, instead of going to the next statement, the control jumps to the called function and executes the statement of that function. After that, the control comes back the point of function call so that the remaining statements in the program can be executed. Therefore, when we read a program, we should not simply read from top to bottom. Instead, we should follow the flow of control or execution. It is also important to note that a function must be defined before its call within a program.

Program 7-11 Program to understand the low of execution using functions.

```
#Program 7-11
#print using functions
helloPython()                              #Function Call

def helloPython():                         #Function definition
     print("I love Programming")
```

On executing the above code the following error is produced:

```
Traceback (most recent call last):
  File "C:\NCERT\Prog 7-11.py", line 3, in <module>
    helloPython()                              #Function Call
NameError: name 'helloPython' is not defined
```

The error 'function not defined' is produced even though the function has been defined. When a function call is encountered, the control has to jump to the function definition and execute it. In the above program, since the function call precedes the function definition, the interpreter does not find the function definition and hence an error is raised.

That is why, the function definition should be made before the function call as shown below:

```
def helloPython(): #Function definition
    print("I love Programming")

helloPython()      #Function Call
```

```
[2]     def Greetings(Name):        #Function Header
[3]         print("Hello "+Name)


[1]     Greetings("John")           #Function Call
[4]     print("Thanks")
```

```
[4]     def RectangleArea(l,b):     #Function Header
[5]     return l*b


[1]     l = input("Length: ")
[2]     b = input("Breadth: ")
[3][6]  Area = RectangleArea(l,b)   #Function Call
[7]     print(Area)
[8]     print("thanks")
```

Figure 7.5: Order of execution of statements

Figure 7.5 explains the flow of execution for two programs. The number in square brackets shows the order of execution of the statements.

Sometime, a function needs to return multiple values which may be returned using tuple. Program 7-12 shows a function which returns two values area and perimeter of rectangle using tuple.

Program 7-12 Write a program using user defined function that accepts length and breadth of a rectangle and returns the area and perimeter of the rectangle.

```
#Program 7-12
#Function to calculate area and perimeter of a rectangle
#The requirements are listed below:
  #1. The function should accept 2 parameters.
  #2. Calculate area and perimeter.
  #3. Return area and perimeter.

def calcAreaPeri(Length,Breadth):
     area = length * breadth
     perimeter = 2 * (length + breadth)
     #a tuple is returned consisting of 2 values area and perimeter
     return (area,perimeter)

l = float(input("Enter length of the rectangle: "))
b = float(input("Enter breadth of the rectangle: "))
#value of tuples assigned in order they are returned
area,perimeter = calcAreaPeri(l,b)
print("Area is:",area,"\nPerimeter is:",perimeter)
```

Output:

```
Enter Length of the rectangle: 45
Enter Breadth of the rectangle: 66
Area is: 2970.0
Perimeter is: 222.0
```

Multiple values in Python are returned through a tuple. (Ch. 10)

Program 7-13 Write a program that simulates a traffic light . The program should consist of the following:

1. A user defined function trafficLight( ) that accepts input from the user, displays an error message if the user enters anything other than RED, YELLOW, and GREEN. Function light() is called and following is displayed depending upon return value from light().
   a) "STOP, your life is precious" if the value returned by light() is 0.
   b) "Please WAIT, till the light is Green " if the value returned by light() is 1
   c) "GO! Thank you for being patient" if the value returned by light() is 2.
2. A user defined function light() that accepts a string as input and returns 0 when the input is RED, 1 when the input is YELLOW and 2 when the input is GREEN. The input should be passed as an argument.
3. Display " SPEED THRILLS BUT KILLS" after the function trafficLight( ) is executed.

```
#Program 7-13
#Function to simulate a traffic light
#It is required to make 2 user defined functions trafficLight() and
#light().

def trafficLight():
     signal = input("Enter the colour of the traffic light: ")
     if (signal not in ("RED","YELLOW","GREEN")):
          print("Please enter a valid Traffic Light colour in
CAPITALS")
     else:
          value = light(signal)          #function call to light()
          if (value == 0):
               print("STOP, Your Life is Precious.")
          elif (value == 1):
               print ("PLEASE GO SLOW.")
          else:
```

```
                print("GO!,Thank you for being patient.")
#function ends here

def light(colour):
     if (colour == "RED"):
          return(0);
     elif (colour == "YELLOW"):
          return (1)
     else:
          return(2)
#function ends here

trafficLight()
print("SPEED THRILLS BUT KILLS")
```

Output:

```
Enter the colour of the traffic light: YELLOW
PLEASE GO SLOW.
SPEED THRILLS BUT KILLS
```

7.4 SCOPE OF A VARIABLE

A variable defined inside a function cannot be accessed outside it. Every variable has a well-defined accessibility. The part of the program where a variable is accessible can be defined as the scope of that variable. A variable can have one of the following two scopes:

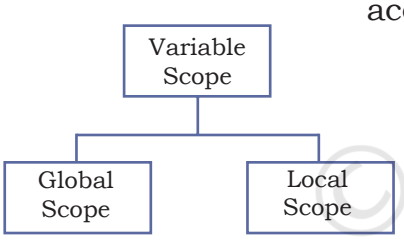


Figure 7.6: Scope of a variable

A variable that has global scope is known as a global variable and a variable that has a local scope is known as a local variable.

(A) Global Variable

In Python, a variable that is defined outside any function or any block is known as a global variable. It can be accessed in any functions defined onwards. Any change made to the global variable will impact all the functions in the program where that variable can be accessed.

(B) Local Variable

A variable that is defined inside any function or a block is known as a local variable. It can be accessed only in the function or a block where it is defined. It exists only till the function executes.

Program 7-14 Program to access any variable outside the function

```
#Program 7-14
#To access any variable outside the function
num = 5
def myFunc1( ):
    y = num + 5
    print("Accessing num -> (global) in myFunc1, value = ",num)
    print("Accessing y-> (local variable of myFunc1) accessible, value=",y)

myFunc1()
print("Accessing num outside myFunc1 ",num)
print("Accessing y outside myFunc1 ",y)
```

Output:

```
Accessing num -> (global) in myFunc1, value =  5
Accessing y-> (local variable of myFunc1) accessible, value =  10
Accessing num outside myFunc1  5
Traceback (most recent call last):
  File "C:\NCERT\Prog 7-14.py", line 9, in <module>
    print("Accessing y outside myFunc1 ",y)
NameError: name 'y' is not defined
```

y generates error when it is accessed outside myfunc1()

→ Global variable output, → Local variable output

***Note*:**

- Any modification to global variable is permanent and affects all the functions where it is used.
- If a variable with the same name as the global variable is defined inside a function, then it is considered local to that function and hides the global variable.
- If the modified value of a global variable is to be used outside the function, then the keyword `global` should be prefixed to the variable name in the function.

Program 7-15 Write a program to access any variable outside the function.

```
#Program 7-15
#To access any variable outside the function

num = 5
def myfunc1():
    #Prefixing global informs Python to use the updated global
    #variable num outside the function
    global num
    print("Accessing num =",num)
    num = 10
    print("num reassigned =",num)
#function ends here

myfunc1()
print("Accessing num outside myfunc1",num)
```

Output:

```
Accessing num = 5
num reassigned = 10
Accessing num outside myfunc1 10
```

Global variable num is accessed as the ambiguity is resolved by prefixing global to it

7.5 PYTHON STANDARD LIBRARY

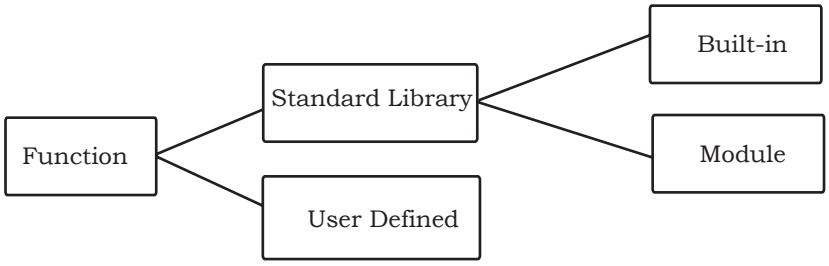


Figure 7.7: Types of functions

Python has a very extensive standard library. It is a collection of many built in functions that can be called in the program as and when required, thus saving programmer's time of creating those commonly used functions everytime.

7.5.1 Built-in functions

Built-in functions are the ready-made functions in Python that are frequently used in programs. Let us inspect the following Python program:

```
#Program to calculate square of a number
a = int(input("Enter a number: ")
b = a * a
print(" The square of ",a ,"is", b)
```

In the above program `input()`, `int()` and `print()` are the built-in functions. The set of instructions to be executed for these built-in functions are already defined in the python interpreter.

Let us consider the following Python statement consisting of a function call to a built in function and answer the given questions:

```
fname = input("Enter your name: ")
```

What is the name of the function being used?

- `input()`

Does the function accept a value or argument?

- Yes, because the parenthesis "()" consists of a string "Enter your name".

Does the function return a value?

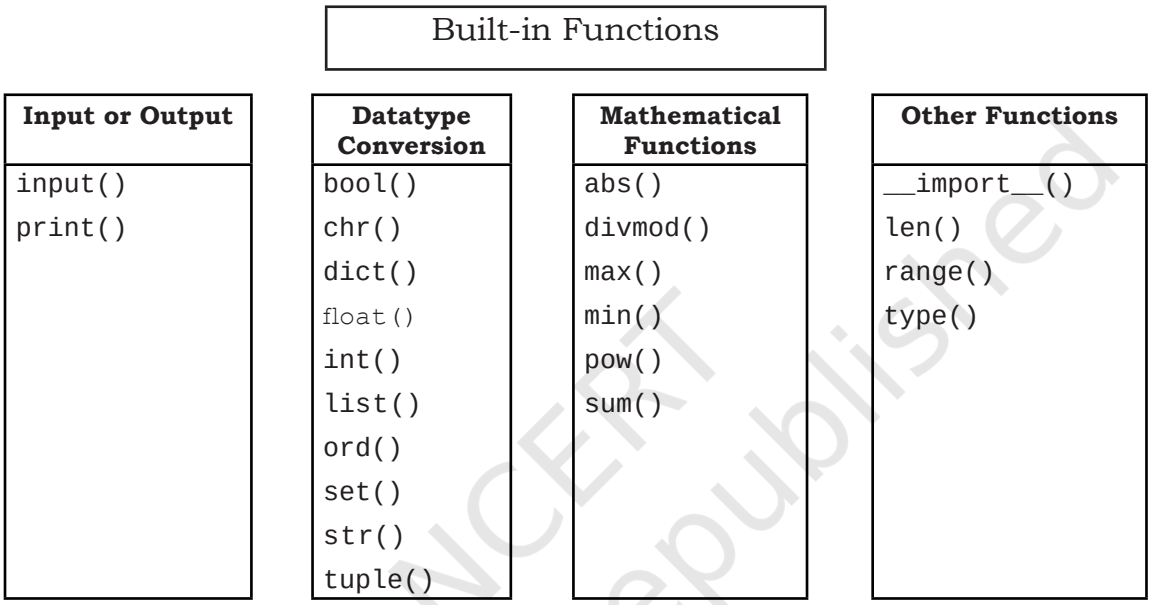
- Yes, since there is an assignment (=) operator preceding the function name, it means that the function returns a value which is stored in the variable fname.

Hence, the function input() accepts a value and returns a value.

Now consider the built-in functions int() and print(), and answer the questions below:

- Does the function accept a value or argument?
- Does the function return a value?

Following is a categorised list of some of the frequently used built-in functions in Python:



We have already used some of the built-in functions. Let us get familiar with some of them as explained in Table 7.1.

Table 7.1 Commonly used built-in functions

Function Syntax	Arguments	Returns	Example Output

abs(x)	x may be an integer or floating point number	Absolute value of x	`>>> abs(4)`
			`4`
			`>>> abs(-5.7)`
			`5.7`

divmod(x,y)	x and y are integers	A tuple: (quotient, remainder)	`>>> divmod(7,2)`
			`(3, 1)`
			`>>> divmod(7.5,2)`
			`(3.0, 1.5)`
			`>>> divmod(-7,2)`
			`(-4, 1)`

max(sequence) or max(x,y,z,...)	x,y,z,.. may be integer or floating point number	Largest number in the sequence/ largest of two or more arguments	`>>> max([1,2,3,4])`
			`4`
			`>>> max("Sincerity")`
			`'y' #Based on ASCII value`

			>>> max(23,4,56) 56

min(sequence)
or
min(x,y,z,...)
	x, y, z,.. may be integer or floating point number	Smallest number in the sequence/ smallest of two or more arguments	>>> min([1,2,3,4])
			1
			>>> min("Sincerity")
			'S'
			#Uppercase letters have lower ASCII values than lowercase letters.
			>>> min(23,4,56)
			4

pow(x,y[,z])	x, y, z may be integer or floating point number	x^y (x raised to the power y)
		if z is provided, then: (x^y) % z
			>>> pow(5,2)
			25.0
			>>> pow(5.3,2.2)
			39.2
			>>> pow(5,2,4)
			1

sum(x[,num])	x is a numeric sequence and num is an optional argument	Sum of all the elements in the sequence from left to right.
		if given parameter, num is added to the sum
			>>> sum([2,4,7,3])
			16
			>>> sum([2,4,7,3],3)
			19
			>>> sum((52,8,4,2))
			66

len(x)	x can be a sequence or a dictionary	Count of elements in x	>>> len("Patience")
			8
			>>> len([12,34,98])
			3
			>>> len((9,45))
			2 >>>len({1:"Anuj",2:"Razia",
			3:"Gurpreet",4:"Sandra"})
			4

7.5.2 Module

Other than the built-in functions, the Python standard library also consists of a number of modules. While a function is a grouping of instructions, a module is a grouping of functions. As we know that when a program grows, function is used to simplify the code and to avoid repetition. For a complex problem, it may not be feasible to manage the code in one single file. Then, the program is divided into different parts under different levels, called modules. Also, suppose we have created some functions in a program and we want to reuse them in another program. In that case, we can save those functions under a module and reuse them. A module is created as a python (.py) file containing a collection of function definitions.

To use a module, we need to import the module. Once we import a module, we can directly use all the functions of that module. The syntax of import statement is as follows:

```
import modulename1 [,modulename2, …]
```

This gives us access to all the functions in the module(s). To call a function of a module, the function name should be preceded with the name of the module with a dot(.) as a separator.
The syntax is as shown below:

```
modulename.functionname()
```

(A) Built-in Modules

Python library has many built-in modules that are really handy to programmers. Let us explore some commonly used modules and the frequently used functions that are found in those modules:

- math
- random
- statistics

Remember, Python is case sensitive. All the module names are in lowercase.

1. Module name : math

It contains different types of mathematical functions. Most of the functions in this module return a float value. Some of the commonly used functions in `math` module are given in Table 7.2. In order to use the math module we need to import it using the following statement:

```
import math
```

Table 7.2 Commonly used functions in math **module**

Function Syntax	Arguments	Returns	Example Output

math.ceil(x)	x may be an integer or floating point number	ceiling value of x	`>>> math.ceil(-9.7)`
			`-9`
			`>>> math.ceil (9.7)`
			`10`
			`>>> math.ceil(9)`
			`9`

math.floor(x)	x may be an integer or floating point number	floor value of x	`>>> math.floor(-4.5)`
			`-5`
			`>>> math.floor(4.5)`
			`4`
			`>>> math.floor(4)`
			`4`

math.fabs(x)	x may be an integer or floating point number	absolute value of x	`>>> math.fabs(6.7)`
			`6.7`
			`>>> math.fabs(-6.7)`
			`6.7`
			`>>> math.fabs(-4)`
			`4.0`

math.factorial(x)	x is a positive integer	factorial of x	`>>> math.factorial(5)` `120`

math.fmod(x,y)	x and y may be an integer or floating point number	x % y with sign of x	`>>> math.fmod(4,4.9)`
			`4.0`
			`>>> math.fmod(4.9,4.9)`
			`0.0`
			`>>> math.fmod(-4.9,2.5)`
			`-2.4`
			`>>> math.fmod(4.9,-4.9)`
			`0.0`

math.gcd(x,y)	x, y are positive integers	gcd (greatest common divisor) of x and y	`>>> math.gcd(10,2)` `2`

math.pow(x,y)	x, y may be an integer or floating point number	x^y (x raised to the power y)	`>>> math.pow(3,2)`
			`9.0`
			`>>> math.pow(4,2.5)`
			`32.0`
			`>>> math.pow(6.5,2)`
			`42.25`
			`>>> math.pow(5.5,3.2)`
			`233.97`

math.sqrt(x)	x may be a positive integer or floating point number	square root of x	`>>> math.sqrt(144)`
			`12.0`
			`>>> math.sqrt(.64)`
			`0.8`

math.sin(x)	x may be an integer or floating point number in radians	sine of x in radians	`>>> math.sin(0)`
			`0`
			`>>> math.sin(6)`
			`-0.279`

2. Module name : random

This module contains functions that are used for generating random numbers. Some of the commonly used functions in `random` module are given in Table 7.3. For using this module, we can import it using the following statement:

```
import random
```

Table 7.3 Commonly used functions in random module

Function Syntax	Argument	Return	Example Output
random.random()	No argument (void)	Random Real Number (float) in the range 0.0 to 1.0	`>>> random.random()` `0.65333522`

random.randint(x,y)	x, y are integers such that x <= y	Random integer between x and y	>>> random.randint(3,7)
			4
			>>> random.randint(-3,5)
			1
			>>> random.randint(-5,-3)
			-5.0

random.randrange(y)	y is a positive integer signifying the stop value	Random integer between 0 and y	>>> random.randrange(5) 4
random.randrange(x,y)	x and y are positive integers signifying the start and stop value	Random integer between x and y	>>> random.randrange(2,7) 2

3. Module name : statistics

This module provides functions for calculating statistics of numeric (Real-valued) data. Some of the commonly used functions in `statistics` module are given in Table 7.4. It can be included in the program by using the following statements:

```
import statistics
```

Table 7.4 Some of the function available through statistics module

Function Syntax	Argument	Return	Example Output
statistics.mean(x)	x is a numeric sequence	arithmetic mean	>>> statistics. mean([11,24,32,45,51]) 32.6
statistics.median(x)	x is a numeric sequence	median (middle value) of x	>>>statistics. median([11,24,32,45,51]) 32
statistics.mode(x)	x is a sequence	mode (the most repeated value)	>>> statistics. mode([11,24,11,45,11]) 11 >>> statistics. mode(("red","blue","red")) 'red'

Note:

- `import` statement can be written anywhere in the program
- Module must be imported only once
- In order to get a list of modules available in Python, we can use the following statement:
  `>>> help("module")`
- To view the content of a module say `math`, type the following:
  `>>> help("math")`

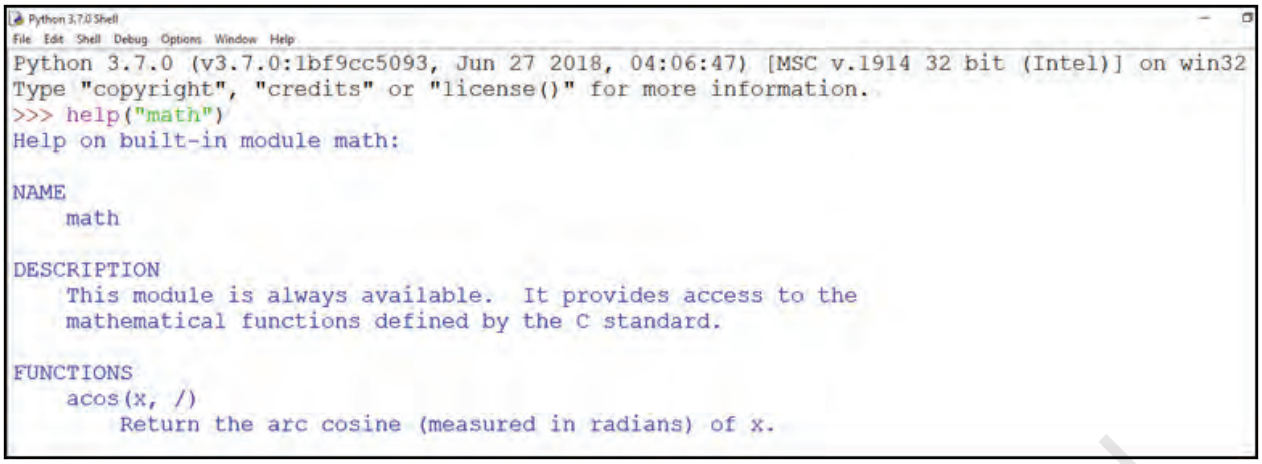


Figure 7.8: Content of module "math"

- The modules in the standard library can be found in the Lib folder of Python.

(B) From Statement

Instead of loading all the functions into memory by importing a module, `from` statement can be used to access only the required functions from a module. It loads only the specified function(s) instead of all the functions in a module.

Its syntax is

```
>>> from modulename import functionname [,
functionname,...]
```

To use the function when imported using "from statement" we do not need to precede it with the module name. Rather we can directly call the function as shown in the following examples:

Example 7.5

```
>>> from random import random
>>> random()      #Function called without
the module name
```

Output:

```
0.9796352504608387
```

Good Programming Practice: Only using the required function(s) rather than importing a module saves memory.

Example 7.6

```
>>> from math import ceil,sqrt
>>> value = ceil(624.7)
>>> sqrt(value)
```

Output:

```
25.0
```

In example 7.2, the ceil value of 624.7 is stored in the variable "value" and then `sqrt` function is applied on the

variable "value". The above example can be rewritten as:

```
>>> sqrt(ceil(624.7))
```

The execution of the function `sqrt()` is dependent on the output of `ceil()` function.

If we want to extract the integer part of 624.7 (we will use `trunc()` function from math module), we can use the following statements.

```
#ceil and sqrt already been imported above
>>> from math import trunc
>>> sqrt(trunc(625.7))
```

Output:

```
25.0
```

A programming statement wherein the functions or expressions are dependent on each other's execution for achieving an output is termed as composition, here are some other examples of composition:

- `a = int(input("First number: "))`
- `print("Square root of ",a ," = ",math.sqrt(a))`
- `print(floor(a+(b/c)))`
- `math.sin(float(h)/float(c))`

Besides the available modules in Python standard library, we can also create our own module consisting of our own functions.

Program 7-16 Create a user defined module `basic_math` that contains the following user defined functions:

1. To add two numbers and return their sum.
2. To subtract two numbers and return their difference.
3. To multiply two numbers and return their product.
4. To divide two numbers and return their quotient and print "Division by Zero" error if the denominator is zero.
5. Also add a docstring to describe the module. After creating module, import and execute functions.

"""Docstrings""" is also called Python documentation strings. It is a multiline comment that is added to describe the modules, functions, etc. They are typically added as the first line, using 3 double quotes.

```
#Program 7-16
#The requirement is:
  #1. Write a docstring describing the module.
  #2. Write user defined functions as per the specification.
  #3. Save the file.
  #4. Import at shell prompt and execute the functions.
```

```
"""
        basic_math Module
       ********************
This module contains basic arithmetic operations
that can be carried out on numbers

"""
#Beginning of module
def addnum(x,y):
    return(x + y)
def subnum(x,y):
    return(x - y)
def multnum(x,y):
    return(x * y)
def divnum(x,y):
    if y == 0:
        print ("Division by Zero Error")
    else:
        return (x/y)          #End of module
```

Output:

```
#Statements for using module basic_math
>>> import basic_math
#Display descriptions of the said module
>>> print(basic_math.__doc__)

        basic_math Module
       ********************
This module contains basic arithmetic operations
that can be carried out on numbers

>>> a = basic_math.addnum(2,5)  #Call addnum() function of the
>>> a                           #basic_math module
7
>>> a = basic_math.subnum(2,5)  #Call subnum() function of the
>>> a                           #basic_ math module
-3
>>> a = basic_math.multnum(2,5) #Call multnum() function of the
>>> a                           #basic_math module
10
>>> a = basic_math.divnum(2,5)  #Call divnum() function of the
>>> a                           #basic_math module
0.4
>>> a = basic_math.divnum(2,0)  #Call divnum() function of the
Zero Divide Error               #basic_math module
```

__doc__ variable stores the docstring. To display docstring of a module we need to import the module and type the following:

```
print(<modulename>.__doc__)          #__ are 2 underscore without space
```

NOTES

SUMMARY

- In programming, functions are used to achieve modularity and reusability.
- Function can be defined as a named group of instructions that are executed when the function is invoked or called by its name. Programmers can write their own functions known as user defined functions.
- The Python interpreter has a number of functions built into it. These are the functions that are frequently used in a Python program. Such functions are known as built-in functions.
- An argument is a value passed to the function during function call which is received in a parameter defined in function header.
- Python allows assigning a default value to the parameter.
- A function returns value(s) to the calling function using return statement.
- Multiple values in Python are returned through a Tuple.
- Flow of execution can be defined as the order in which the statements in a program are executed.
- The part of the program where a variable is accessible is defined as the scope of the variable.
- A variable that is defined outside any particular function or block is known as a global variable. It can be accessed anywhere in the program.
- A variable that is defined inside any function or block is known as a local variable. It can be accessed only in the function or block where it is defined. It exists only till the function executes or remains active.
- The Python standard library is an extensive collection of functions and modules that help the programmer in the faster development of programs.
- A module is a Python file that contains definitions of multiple functions.
- A module can be imported in a program using import statement.
- Irrespective of the number of times a module is imported, it is loaded only once.
- To import specific functions in a program from a module, `from` statement can be used.

NOTES

EXERCISE

1. Observe the following programs carefully, and identify the error:

a)
```
def create (text, freq):
        for i in range (1, freq):
            print text
create(5)          #function call
```

b)
```
from math import sqrt,ceil
def calc():
        print cos(0)
calc()             #function call
```

c)
```
mynum = 9
def add9():
        mynum = mynum + 9
        print mynum
add9()             #function call
```

d)
```
def findValue( vall = 1.1, val2, val3):
        final = (val2 + val3)/ vall
        print(final)
findvalue()        #function call
```

e)
```
def greet():
        return("Good morning")
greet() = message  #function call
```

2. How is math.ceil(89.7) different from math.floor (89.7)?
3. Out of random() and randint(), which function should we use to generate random numbers between 1 and 5. Justify.
4. How is built-in function pow() function different from function math.pow() ? Explain with an example.
5. Using an example show how a function in Python can return multiple values.
6. Differentiate between following with the help of an example:
   a) Argument and Parameter
   b) Global and Local variable
7. Does a function always return a value? Explain with an example.

ACTIVITY-BASED QUESTIONS

Note: Writing a program implies:

- Adding comments as part of documentation
- Writing function definition
- Executing the function through a function call

1. To secure your account, whether it be an email, online bank account or any other account, it is important that we use authentication. Use your programming expertise to create a program using user defined function named login that accepts userid and password as parameters (login(uid,pwd)) that displays a message “account blocked” in case of three wrong attempts. The login is successful if the user enters user ID as "ADMIN" and password as "St0rE@1". On successful login, display a message “login successful”.
2. XYZ store plans to give festival discount to its customers. The store management has decided to give discount on the following criteria:

Shopping Amount	Discount Offered
>=500 and <1000	5%
>=1000 and <2000	8%
>=2000	10%

An additional discount of 5% is given to customers who are the members of the store. Create a program using user defined function that accepts the shopping amount as a parameter and calculates discount and net amount payable on the basis of the following conditions:

Net Payable Amount = Total Shopping Amount – Discount.

3. ‘Play and learn’ strategy helps toddlers understand concepts in a fun way. Being a senior student you have taken responsibility to develop a program using user defined functions to help children master two and three-letter words using English alphabets and addition of single digit numbers. Make sure that you perform a careful analysis of the type of questions that can be included as per the age and curriculum.

NOTES

4. Take a look at the series below:
   1, 1, 2, 3, 5, 8, 13, 21, 34, 55...
   To form the pattern, start by writing 1 and 1. Add them together to get 2. Add the last two numbers: 1+2 = 3.Continue adding the previous two numbers to find the next number in the series. These numbers make up the famed Fibonacci sequence: previous two numbers are added to get the immediate new number.
5. Create a menu driven program using user defined functions to implement a calculator that performs the following:
   a) Basic arithmetic operations(+,-,*,/)
   b) $\log_{10}(x)$,sin(x),cos(x)

SUGGESTED LAB. EXERCISES

1. Write a program to check the divisibility of a number by 7 that is passed as a parameter to the user defined function.
2. Write a program that uses a user defined function that accepts name and gender (as M for Male, F for Female) and prefixes Mr/Ms on the basis of the gender.
3. Write a program that has a user defined function to accept the coefficients of a quadratic equation in variables and calculates its determinant. For example : if the coefficients are stored in the variables a,b,c then calculate determinant as b^2-4ac. Write the appropriate condition to check determinants on positive, zero and negative and output appropriate result.
4. ABC School has allotted unique token IDs from (1 to 600) to all the parents for facilitating a lucky draw on the day of their Annual day function. The winner would receive a special prize. Write a program using Python that helps to automate the task.(Hint: use random module)
5. Write a program that implements a user defined function that accepts Principal Amount, Rate, Time, Number of Times the interest is compounded to calculate and displays compound interest. (Hint: CI = $((P*(1+R/N))^{NT})$

NOTES

6. Write a program that has a user defined function to accept 2 numbers as parameters, if number 1 is less than number 2 then numbers are swapped and returned, i.e., number 2 is returned in place of number1 and number 1 is reformed in place of number 2, otherwise the same order is returned.
7. Write a program that contains user defined functions to calculate area, perimeter or surface area whichever is applicable for various shapes like square, rectangle, triangle, circle and cylinder. The user defined functions should accept the values for calculation as parameters and the calculated value should be returned. Import the module and use the appropriate functions.
8. Write a program that creates a GK quiz consisting of any five questions of your choice. The questions should be displayed randomly. Create a user defined function score() to calculate the score of the quiz and another user defined function remark (scorevalue) that accepts the final score to display remarks as follows:

Marks	Remarks
5	Outstanding
4	Excellent
3	Good
2	Read more to score more
1	Needs to take interest
0	General knowledge will always help you. Take it seriously.

CASE STUDY-BASED QUESTION

For the SMIS system extended in Chapter 6 let us do the following:

1. 7.1 Convert all the functionality in Chapter 5 and 6 using user defined functions.
2. 7.2 Add another user defined function to the above menu to check if the student has short attendance or not. The function should accept total number of working days in a month and check if the student is a defaulter by calculating his or her attendance using the formula: Count of days the student was

NOTES

present or the total number of working days. In case the attendance calculated is less than 78%, the function should return 1 indicating short attendance otherwise the function should return 0 indicating attendance is not short.

Let's peer review the case studies of others based on the parameters given under "DOCUMENTATION TIPS" at the end of Chapter 5 and provide a feedback to them.