3 Introduction to Data and Programming with Python



3.4 Conditional Statements

Learning Scope

Introduction, Normal flow of control, Blocks and Indentation in Python, Conditional statements (if, multiple use of if statement, if-else, if-elif-else), case and switch in Python, Python Mathematical functions: max( ), min( ), pow( ), sqrt( ), etc., Types of errors (syntax error, logical error, run time error).

INTRODUCTION

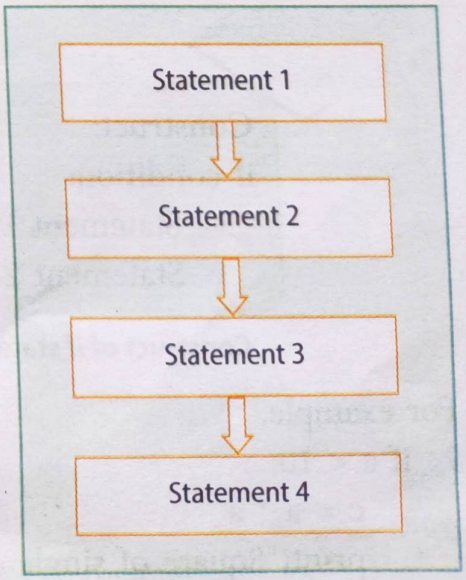
In the previous chapter, you learnt the fundamental elements of Python programming such as character set, constants and variables, keywords, assignments, operators, etc. and user-friendly coding by using input statement. In this segment, we will take a tour of conditional statements and some Python built-in functions in brief.

CONDITIONAL STATEMENTS

In the previous class, you learnt to write simple programs using sequential set of statements. In this case, when a command is issued to execute a program, the control reaches the first line of the program. Thereafter, it moves to the next line and so on. The control keeps on executing the lines, one after the other, unless it reaches the end of the program. When the execution is over, the control finally returns to the system. This sequential movement of the flow of control from one statement to the other is termed as *normal flow of control.*

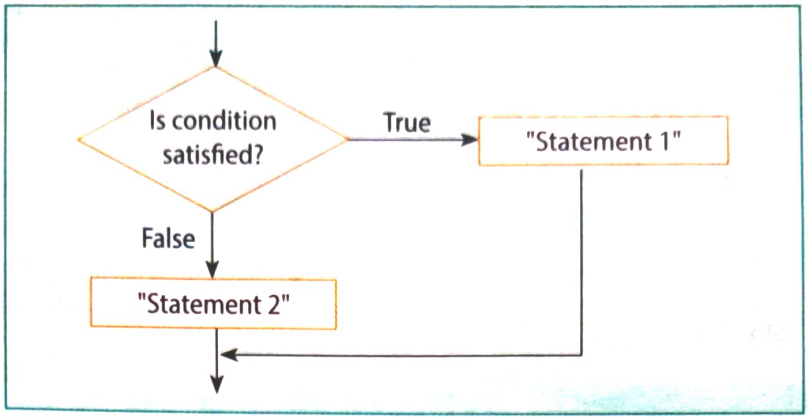
In a normal flow of control, all the statements are executed one after the other (by default). It means that the execution of a program will start from 'Statement 1' and then will move to 'Statement 2', 'Statement 3' and so on till the last statement of the program. It is the simplest way of executing a program, where there are no restrictions to the flow of control during the process.

But, this normal flow of control may not be helpful, if the complexity of the program increases. You may need to transfer the control to a defined statement while ignoring some statements unexecuted. This type of construct is known as conditional statement.



Normal flow of control

Let us understand the concept of conditional statements in brief:



Here, the control checks the condition. Once, the condition is verified, it will execute either 'Statement 1' or 'Statement 2' and will allow the control to execute the statements selectively.

In this segment, we will discuss the different constructs of conditional statements in brief.

Types of Conditional Statements

Sometimes, the user may like to execute a statement(s) by ignoring other statement(s) to yield the desired result. This can be done by using conditional flow of control. Python language allows to create conditional flow of control with the help of if statement. The if statement can be used in different forms, as per the requirements of the users or the suitability of the program logic. They are:

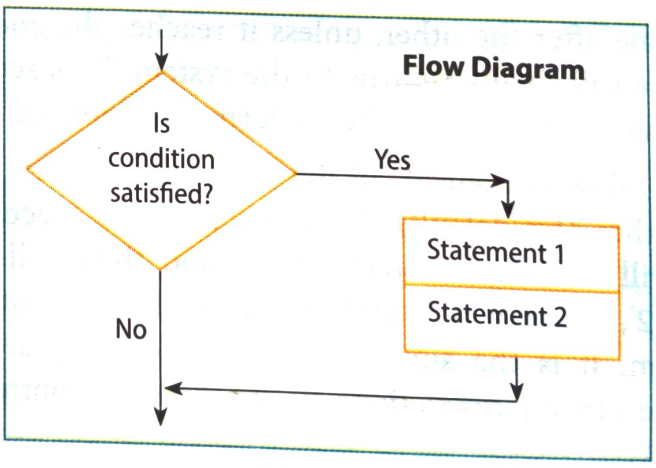
- if statement
- if and only if statement (multiple use of if)
- if-else statement
- if-elif-else statement

if-statement

You can use if statement to check a specific condition. If the condition is true, it executes a statement or a set of statements under the same *indented block*. When the given condition is false, the control ignores the execution of the indented block.

```
Construct:
if condition:
    Statement 1
    Statement 2
```

Construct of *if* statement



Flowchart

For example,

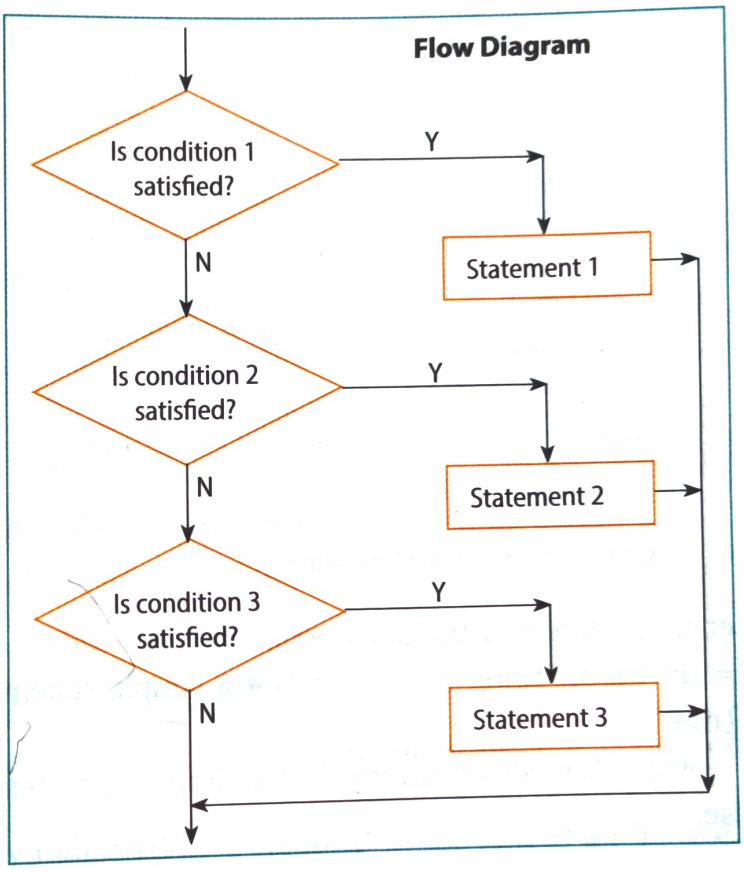
```
if a < 10:
    c = a * a
    print("Square of single digit number is", c)
```

At first, the condition a<10 is checked. If it is true, then the square of 'a' will be calculated as well as displayed on the screen because both the statements are written under the same indentation. In case the condition is false, the execution of the complete block shall be ignored.

if and only if statements (multiple use of if)

If quite a few 'if' statements are used in a code, known as 'if and only if' statements. It is a logical situation, in which a statement or a set of statements indented under same block are executed, if a specific condition stands true.

Refer the given construct. The 'statement 1' will get executed only when 'condition 1' is true. In case, 'condition 1' is false, the control moves to check whether 'condition 2' is true or not. If 'condition 2' is true then 'statement 2' will be executed otherwise, it will check 'condition 3' and so on.



Flowchart

Construct:

```
if condition 1:
    Statement 1
if condition 2:
    Statement 2
if condition 3:
    Statement 3
```

Construct of multiple use of *if* statement

For example,

```
if (marks >=80):
    print("Distinction")
if (marks >= 60 and marks <80):
    print("First Division")
if (marks >= 50 and marks < 60):
    print("Second Division")
if (marks >=40 and marks< 50):
    print("Third Division")
if (marks < 40):
    print("Grade not allotted")
```

Indentation

Refer the example shown above. A specific grade will be displayed based on the marks.

Note

Indentation refers to as the aligning a statement or group of statements which are the part of another statement. Python programming is very specific about indentation. The code will result in an error, if the indentation of statements is not properly done.

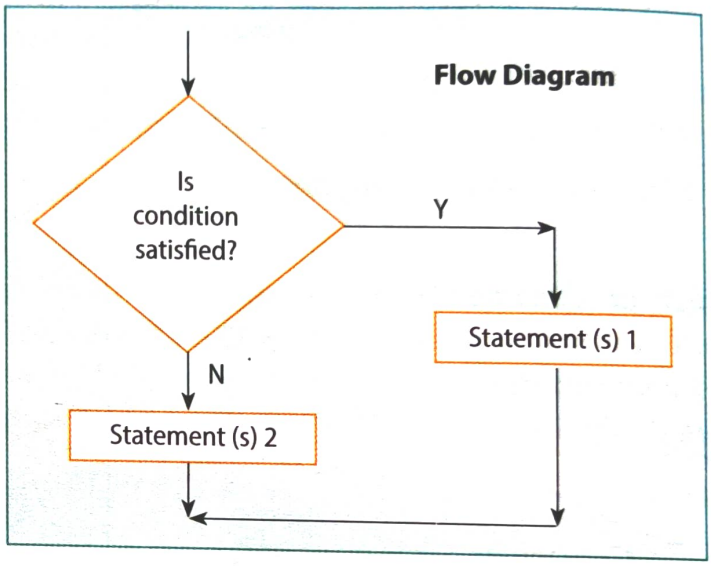
if-else

In a logical situation, you can use if-else statement when either of the two actions is to be carried out, depending upon certain conditions. If the condition is true, the control shall execute the task of one block of indented statements otherwise, it shall execute another block of indented statements.

Construct:

```
if condition:
    Statement(s) 1
else:
    Statement(s) 2
```

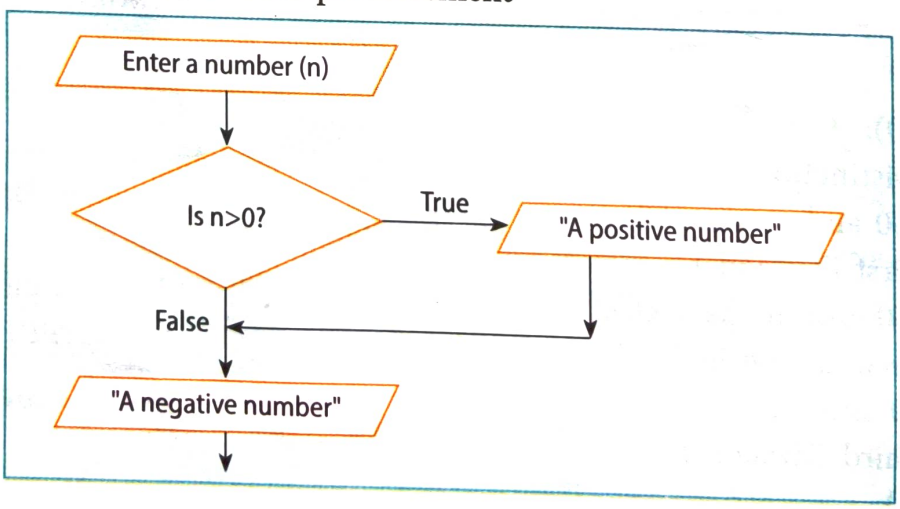
Construct of if-else statement



Flowchart

Let us understand the above concept with code:

```
n= int(input("Enter a number")) # a simple statement
if(n>0):
    print("A positive number") # a simple statement
else:
    print("A negative number") # a simple statement
```



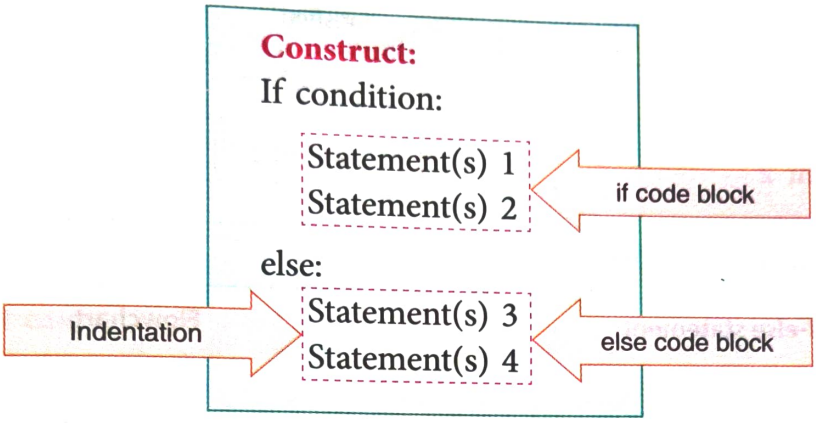
Here, you may observe that the control checks the condition whether the number is greater than zero or not. If the number is greater than zero then the program will display "A positive number" and if the number is less than zero then "A negative number" is displayed.

Blocks and Indentation in Python

In Python, indentation refers to as the adding white space before a statement to a particular block of code. In other words, it is the alignment of all the statements with the same space to the left, belong to the same code block.

Unlike in other programming languages, it is marked with the use of braces { } or keywords to define code blocks. However, Python relies on indentation levels to determine the grouping of statements.

For example,



Key Points about Indentation in Python:

- **Indentation Level**
  - ➢ All lines of code that are part of the same block must have the same indentation level.
  - ➢ The standard indentation is four spaces, but any consistent number of spaces or tabs can be used.
- **Code Blocks**
  - ➢ A code block is a group of statements that are executed together.
    For example, the body of a function, the statements inside an if condition, else condition or the content of a loop.
  - ➢ The start of a new code block is typically indicated by a colon (:) at the end of a line (e.g., after if, for, while, def, etc.), followed by an indented line.

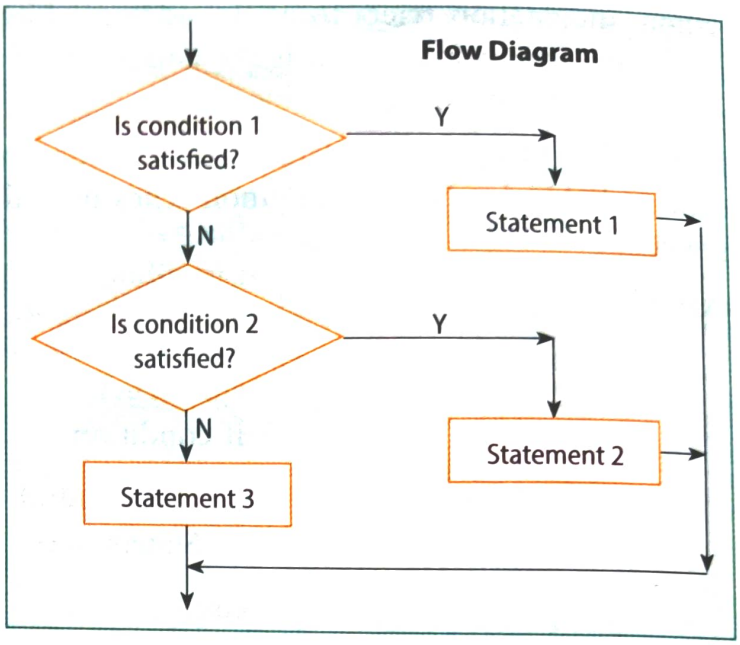
if-elif-else

It is also a type of conditional construct, where multiple 'if else' statements are connected with the help of a keyword '*elif* '. It allows the user to execute a statement or a block of statements when the given condition in the first 'if ' statement is true. When the first condition is false, it checks another condition and then takes the necessary action. At the end, the logical construct stops with 'else' statement.

The construct is given as:

```
Construct:
if condition1:
    Statement 1
elif condition 2:
    Statement 2
else:
    Statement 3
```

Construct of if-elif-else statement



Flowchart

If the first condition is true, then 'Statement 1' will be executed and the other conditions will be ignored. However, if the first condition is false, then the second condition is checked and if it happens to be true, then 'Statement 2' will be executed and 'Statement 3' shall be ignored. If first and second condition happen to be false, then 'Statement 3' will be executed.

For example,

```
if (a < 10):
    print("Single digit number")
elif (a >= 10 and a <100):
    print("Double digit number")
else:
    print("Multiple digit number")
```

Refer to the above example. Here, if condition 'a < 10' is true and then it will display "Single digit number". If it is false, then will check another condition 'a >= 10 and a < 100'. If the condition is true, will display "Double-digit number" otherwise, it will display "Multiple-digit number".

INTRODUCING MATCH AND CASE IN PYTHON 3.10

Until version 3.10, Python never had a feature that implemented what the switch statement does in other programming languages. So, if you want to execute multiple conditional statements, you would have to use the elif keyword.

With Python 3.10, it has implemented a switch case feature called "structural pattern matching". The match statement and case keywords were introduced, providing a more elegant solution similar to traditional switch case statements like in other languages. Thus, you can design a program by implementing this feature with the match and case keywords.

Syntax:

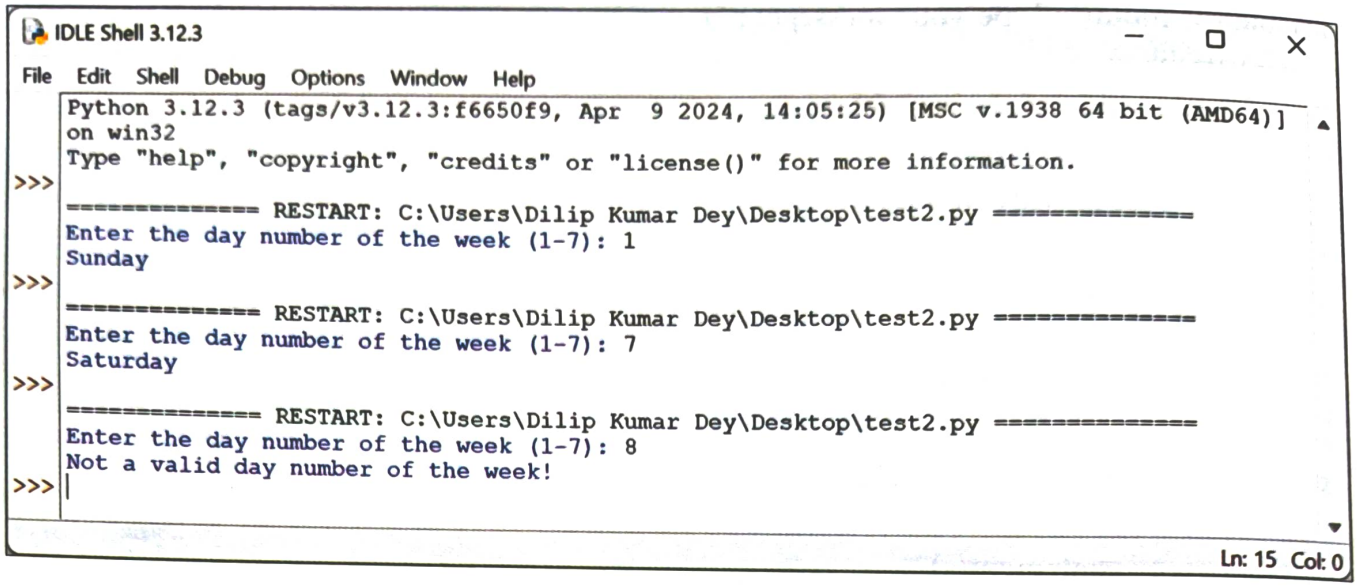
```
<variable> = input("<Type your message>: ")
match <variable>:
   case <input value of the variable>:
        .....
        .....statement(s).....
        .....
   case <input value of the variable>:
        .....
        .....statement(s).....
        .....
   case <input value of the variable>:
        .....
        .....statement(s).....
        .....
   case _: # default case
      print(<default message>)
```

You probably noticed that the program did not add a break keyword to each of the cases, as it is done in other programming languages. This is the advantage of Python's native switch statement has over those of other languages. The break functionality of break keyword is done behind the scenes (by default).

Let's design a program where we want to categorise days of the week by identifying a specific day number, taking an integer value as input. Here's how we can use the match statement:

```
day = int(input("Enter the day number of the week (1-7): "))
match day:
   case 1:
      print("Sunday")
   case 2:
      print("Monday")
   case 3:
      print("Tuesday")
   case 4:
      print("Wednesday")
   case 5:
      print("Thursday")
   case 6:
      print("Friday")
   case 7:
      print("Saturday")
   case _:
      print("Not a valid day number of the week!")
```

Output:



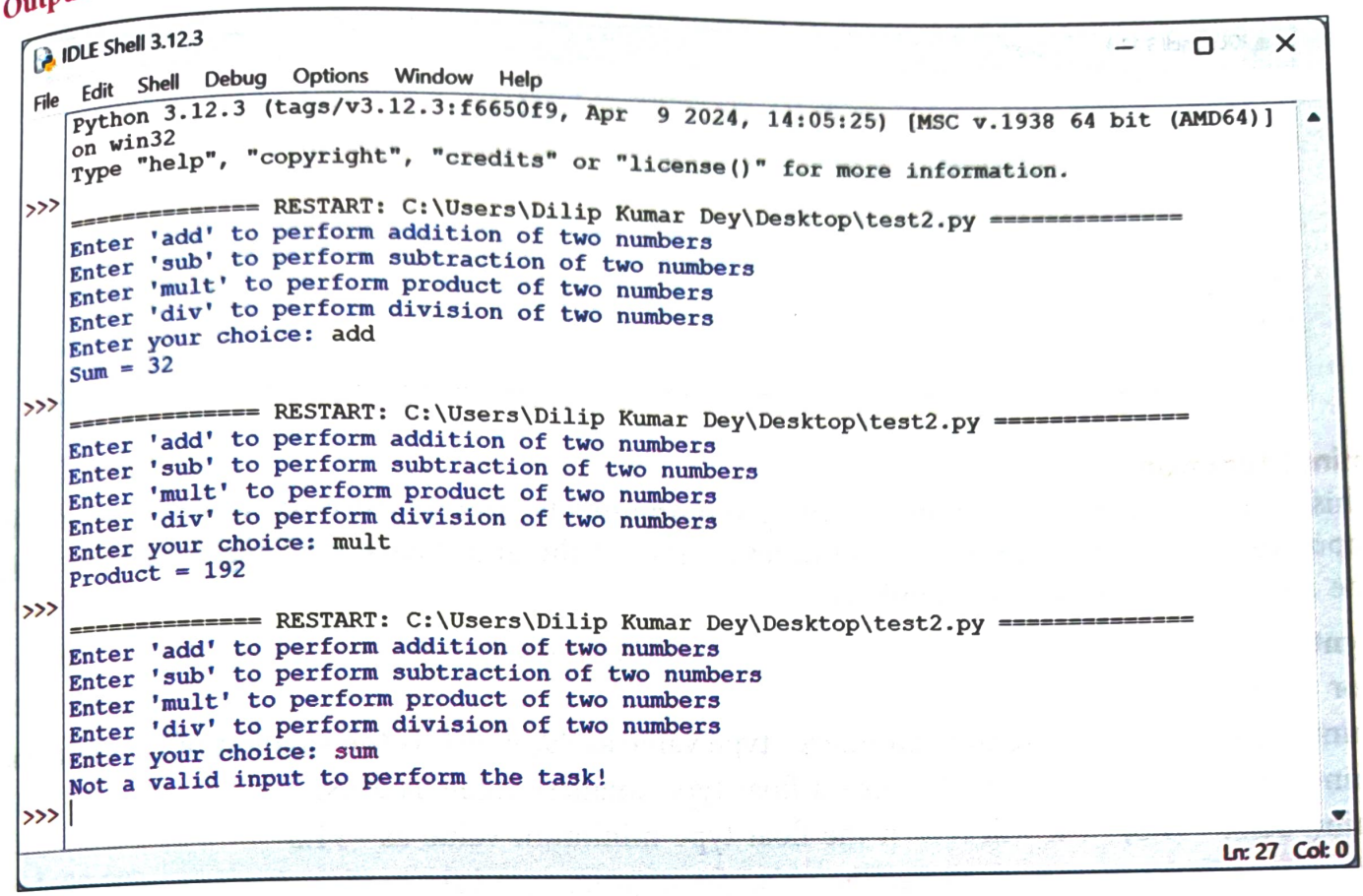
Explanation:

- The first line prompts the user to input the day number of the week.
- The match statement is used to compare the user's input with different cases.
- If the user's input matches between 1 to 7 (both inclusive), it prints the day of the week.
- The case _: is a default case that matches anything not covered by the previous cases.
- If the user's input doesn't match any of the defined cases, it prints an appropriate message.

We can also design a program by considering string value as input from the user. the program is designed as under:

```
# Python switch case statement by taking string input
a=24; b=8
print("Enter 'add' to perform addition of two numbers")
print("Enter 'sub' to perform subtraction of two numbers")
print("Enter 'mult' to perform product of two numbers")
print("Enter 'div' to perform division of two numbers")
choice = input("Enter your choice: ")
match choice:
    case 'add':
        print("Sum =",(a+b))
    case 'sub':
        print("Difference =",(a-b))
    case 'mult':
        print("Product =",(a*b))
    case 'div':
        print("Division =",(a/b))
    case _:
        print("Not a valid input to perform the task!")
```

Output:



PYTHON BUILT-IN MATHEMATICAL FUNCTIONS

Sometimes, we need to include built-in functions (predefined set of instructions) in coding to perform certain specific tasks. As a result, it not only enhances the capability of processing, but also reduces the complexity of programming logic. In this section, we will discuss some of the built-in functions, used for mathematical operations. However, it must be noted that there are some mathematical functions where you need to import math module to perform the task such as sqrt( ), ceil( ), floor( ), etc.

Let us discuss the following functions to perform different types of mathematical calculations in brief.

max( ) function

This function is used to find the largest of the given arguments. It returns a value depending upon the arguments (i.e., if the arguments are int types then the return value will be an int).

Syntax: *<variable> = Function name (argument1, argument2,.....)*

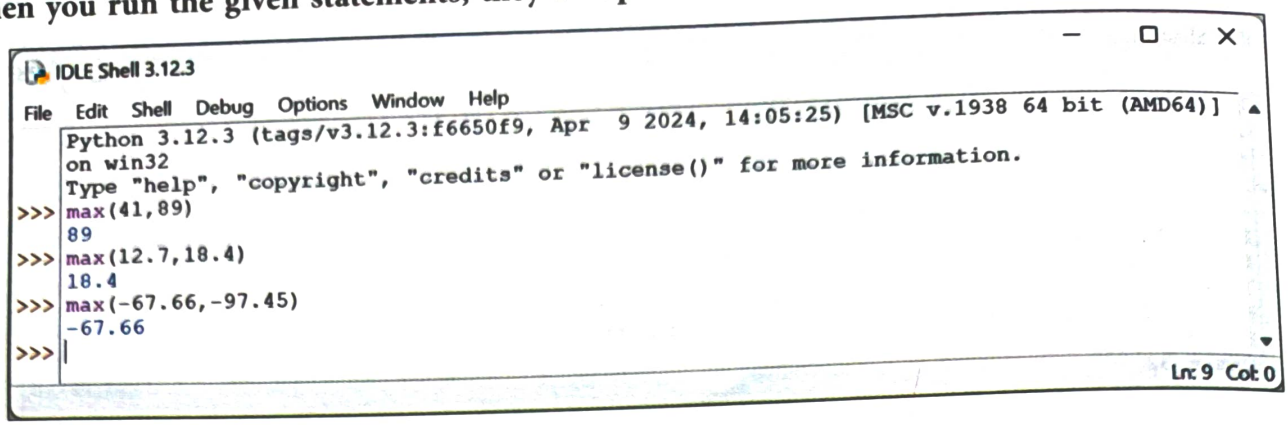
For example,

max(41,89) : It will return an int type largest value as 89.

max(12.7,18.4) : This will return a float type largest value as 18.4.

max(–67.66, –97.45) : This function will return largest value float type as –67.66.

When you run the given statements, they will produce the following results:



min( ) function

This function returns the smallest of the given arguments. The return value will depend on the input values of the arguments of the function (i.e., if the arguments are integer numbers then the result will be an integer number)

Syntax: *<variable> = Function name (argument1, argument2, ...)*

For example,

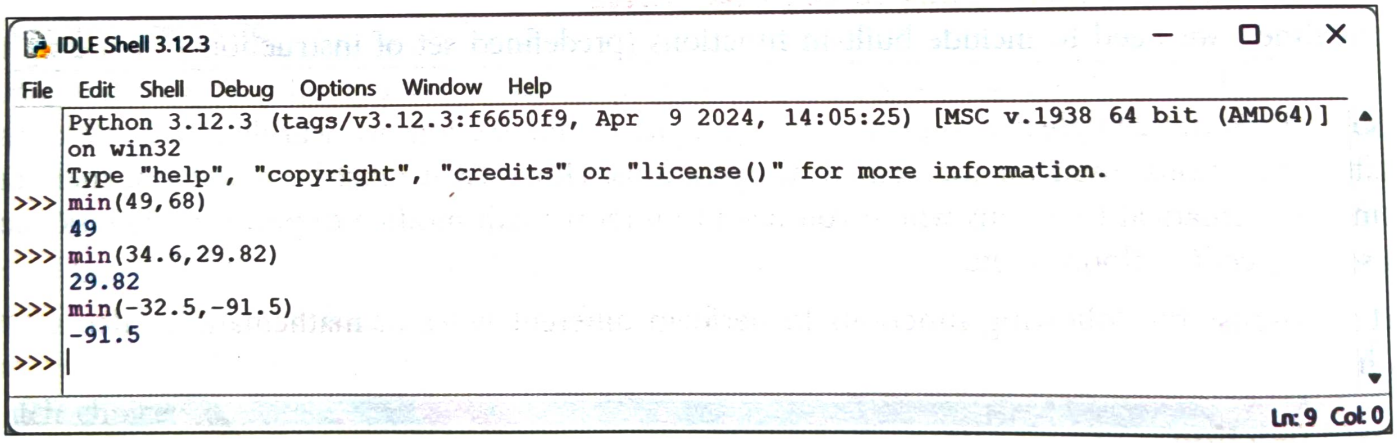
min(49,68) : It returns an integer type value as 49, which is the smallest of its argument.

min(34.6,29.82) : This will return a float type smallest value as 29.82

min(–32.5, –91.5) : It will return the float type minimum value as –91.5

Note: Higher the magnitude, lower the value for negative numbers produced.

When you run the above statements, they will produce the following results:



pow( ) function

This function is used to find the power 'b' raised to a specified base 'a' (i.e.; ab). It will return the data of type *int* or *float*, depending upon the output.

Syntax: *<variable> = Function name (number1, number2)*

(The arguments number1 and number2 represent base and power respectively.)

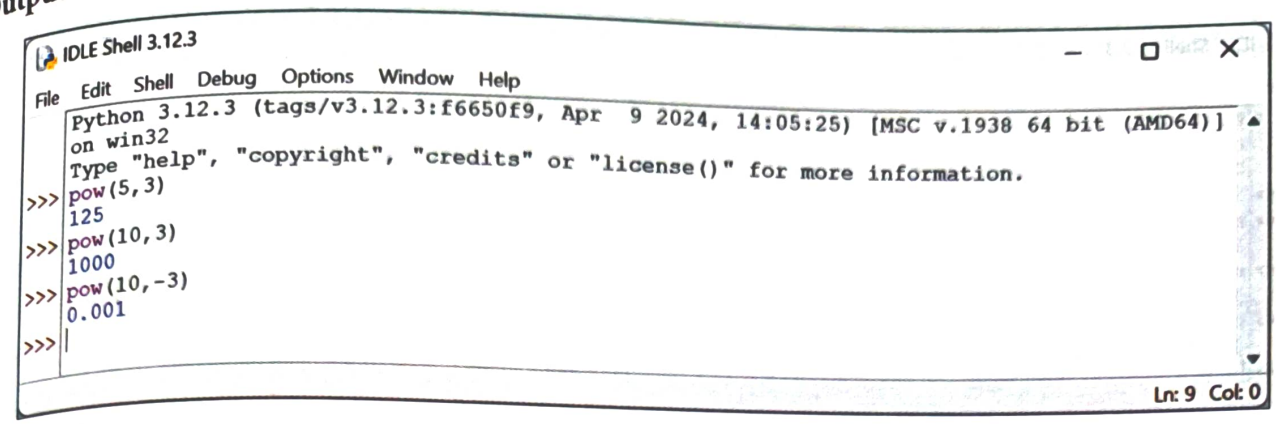
For example,

pow(5,3) : The function will return an *int* type value as 125

pow(10,3) : It will return an *int* type value as 1000

pow(10, –3) : It will return a *float* type value as 0.001

Output of the given statements is shown as:



abs( ) function

This function is used to return an absolute value (i.e., only the magnitude of the number). In other words, the absolute value of any number is always positive. It returns the value as an int or a float type, depending upon the given argument.

Syntax: *<variable> = Function name (argument)*

For example,

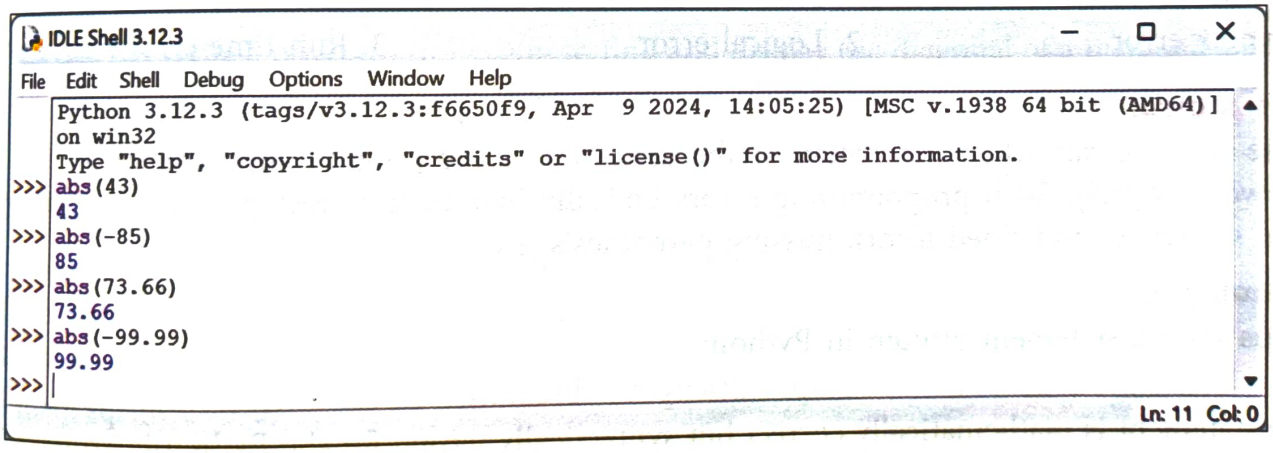
abs(43)

abs(–85)

abs(73.66)

abs(–99.99)

Output of the above statements is shown as:



sqrt( ) function

This function is used to find the square root of a number p for p>0. It always returns a float type value for a given argument. In this case, you need to import math module to get the output of a given argument.

Syntax: *<variable> = math.Function name (number)*

For example,

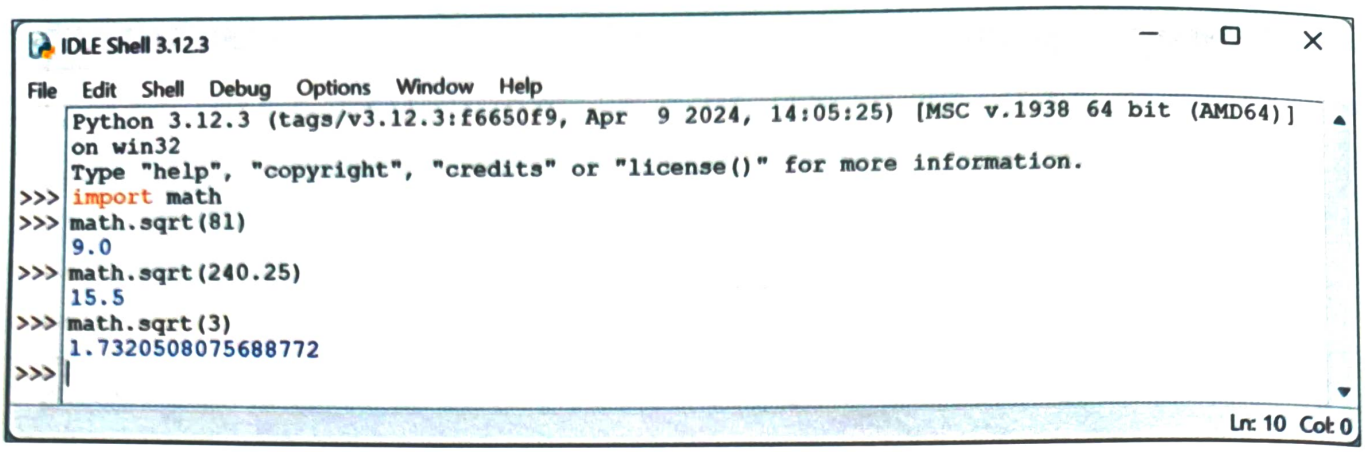
import math

math.sqrt(81) : It will return a float type value as 9.0

math.sqrt(240.25) : It will return a float type value as 15.5

math.sqrt(3) : It will return a float type value as 1.732……..

Output of the given statements is shown as:



Note

Square root of a negative value is an imaginary number.

If you use a function as:

>>> math.sqrt(-3.25)

It will display a *ValueError*.

TYPES OF ERRORS

When you execute a set of codes and it either results in an error or does not produce the desired result, it is because an error has been encountered in the code. Moreover, as a programmer you should be aware of the types of errors that may occur while writing codes. In general, there are three types of errors which occur while designing codes. They are:

1. Syntax error
2. Logical error
3. Run time error

Syntax error

These errors occur when the syntax or the grammar of the programming statements are not followed properly. Such programming errors typically involve incorrect punctuation, incorrect word sequence, undefined terms, missing parenthesis, etc.

For example,

Let us refer a statement written in Python:

p = 2(length + breadth)

This statement is mathematically correct but syntactically wrong for the Python interpreter. The reason for the syntax error is simply the missing '*' sign between 2 and (length + breadth). The correct statement will be:

p = 2*(length + breadth)

Note

If a code is syntactically wrong for Python, it will not execute unless the errors are corrected.

Logical error

A logical error is an error which is made while planning the program's logic. It may be improper sequencing of the statements, wrong placement of the operators or wrongly setting the hierarchies. When a logical error occurs, the computer does not kno r that an error has

been made. It simply follows the code instructions and executes them, but the result may not be correct. This problem occurs due to wrong logic and thus, it does not produce the desired output.

For example;

In a Python code, you want to find the sum of two numbers, but the instruction is given as:

```
a, b = 15, 28
c = a * b
print(("Sum of two number=", c)
```

The snippet shown above, will print the product of 'a' and 'b' whereas, you wanted to find the sum of 'a' and 'b'. This is a logical error.

Run time error

Even after errors are removed and the program is interpreted, it does not mean that the program will produce the correct result. The mistakes, other than syntax error or logical error, which interrupt the correct execution of the program, are referred to as *Run Time* errors. These errors may occur due to the division of a number by zero or to find out the square root of a negative number, etc.

For example,

The expression a = (b + c)/d

If the value of d = 0, then the system will show as *ZeroDivisionError* which is a run time error. It is an error because a number cannot be divided by zero.

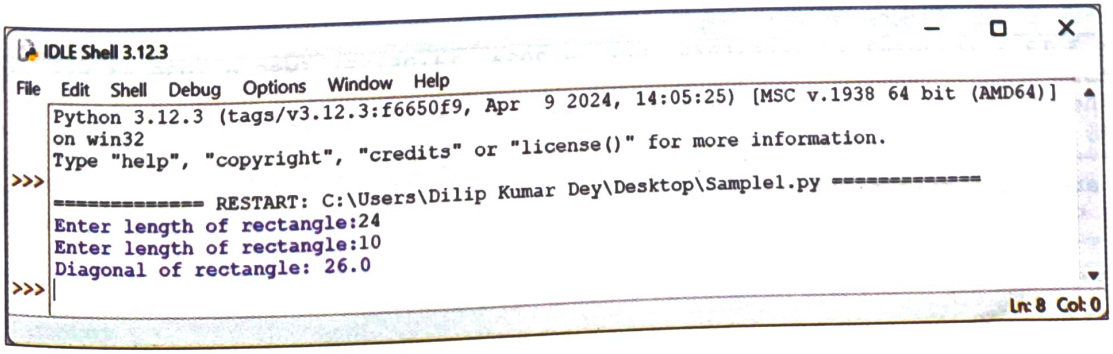
Solved Python Programs

Program 1 Write a program in Python to find and display the diagonal of a rectangle taking length and breadth of the rectangle as inputs.

[Hint: d = $\sqrt{l^2 + b^2}$]

```
import math
# a program to display area of a rectangle
a = int(input("Enter length of rectangle:"))
b = int(input("Enter length of rectangle:"))
d = math.sqrt(a*a+b*b)
print("Diagonal of rectangle:", d)
```

Execution:

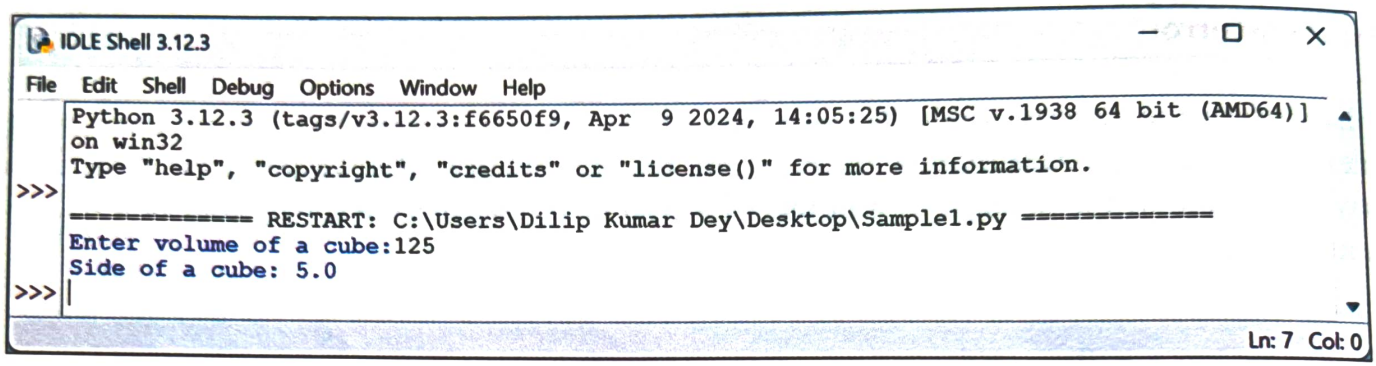


Program 2 Write a program in Python to calculate and display the side of a cube taking its volume as an input.

[Hint: side = $\sqrt[3]{volume}$

```
# a program to display diagonal of a cube
v = int(input("Enter volume of a cube:"))
side = pow(v,1/3)
print("Side of a cube:", side)
```

Execution:



Program 3 Calculate the radius of a circle by using the formula:

Area = $\frac{22}{7}$ r2; where r = radius of the circle

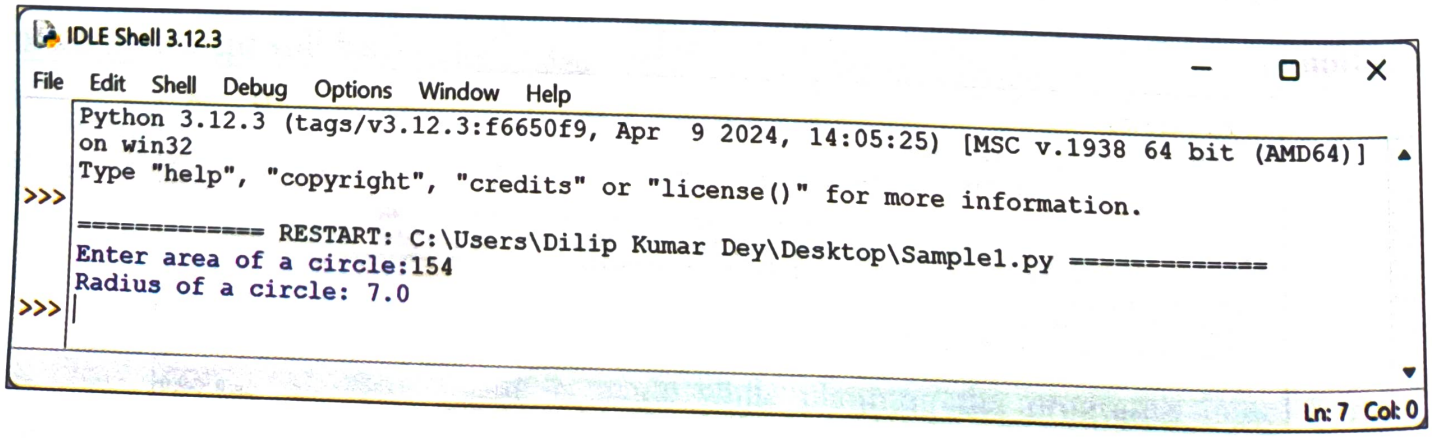
Hence, the radius can be calculated as:

r = $\sqrt{\frac{7 * area}{22}}$

Write a program in Python to calculate and display the radius of a circle by taking the area as input.

```
import math
# a program to display radius of a circle
area = int(input("Enter area of a circle:"))
r = math.sqrt(7*area/22)
print("Radius of a circle:", r)
```

Execution:



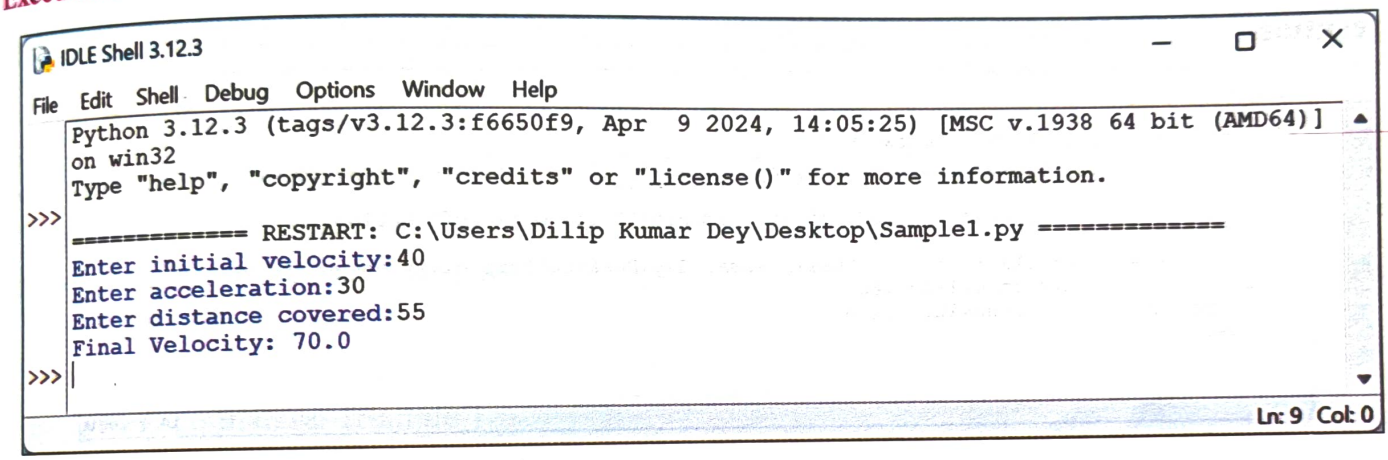
Program 4 The final velocity of a vehicle can be calculated by using the formula:

$$v2 = u2 + 2as$$

where u = initial velocity, a = acceleration, s = distance covered
Write a code in Python to calculate and display the final velocity by taking the initial velocity, acceleration and the distance covered as inputs.

```
import math
# a program to calculate final velocity
u = int(input("Enter initial velocity:"))
a = int(input("Enter acceleration:"))
s = int(input("Enter distance covered:"))
v = math.sqrt(u*u+2*a*s)
print("Final Velocity:", v)
```

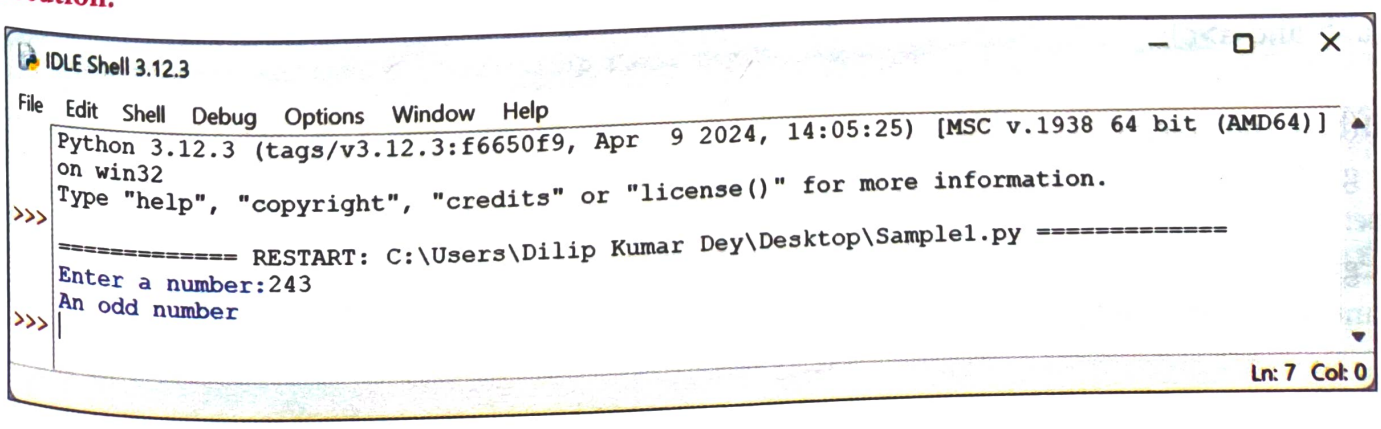
Execution:



Program 5 Write a Python program to input a number and check whether it is an even number or an odd number. Display the message accordingly.

```
# a Python program to check an even or an odd number
num=int(input("Enter a number:"))
if num%2==0:
    print("An even number")
else:
    print("An odd number")
```

Execution:



Program 6 Write a Python program to input temperature in Celsius. Convert the temperature into Fahrenheit. Display the temperature and if the temperature is more than 98.6 then display "Fever", otherwise "Normal".

Hint: $= \frac{c}{5} = \frac{f - 32}{9}$

```
# a Python program to display temperature
c=int(input("Enter temperature in Celsius:"))
f=9*c/5+32
print("Temperature in Fahrenheit:", f)
if f>98.6:
    print("Fever!")
else:
    print("Normal")
```

Execution:

```
IDLE Shell 3.12.3
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Dilip Kumar Dey\Desktop\Sample1.py ==========
Enter temperature in Celsius:40
Temperature in Fahrenheit: 104.0
Fever!
>>>
                                                                    Ln: 8  Col: 0
```

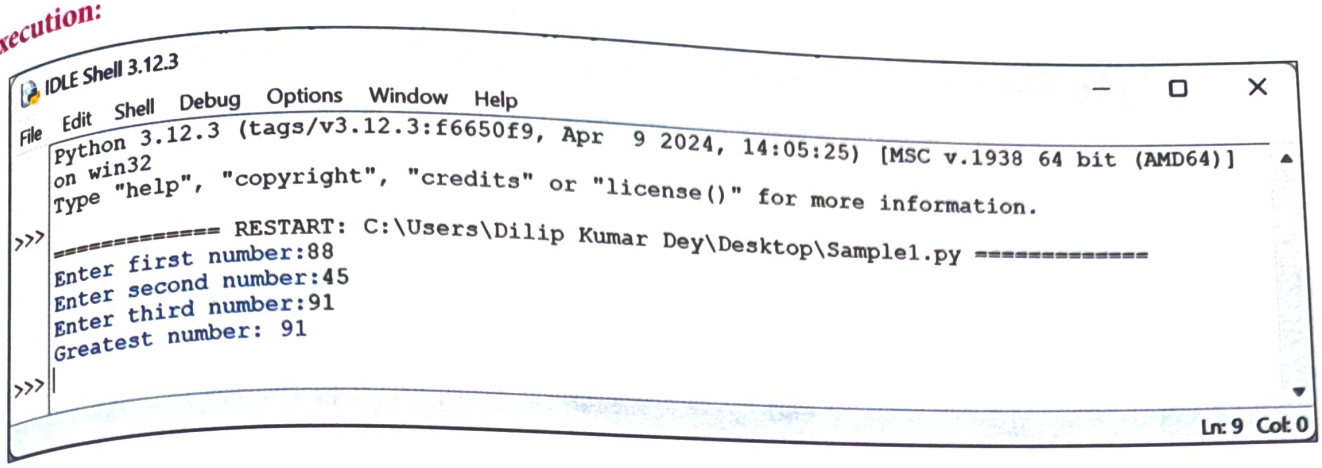
Program 7 Write a program in Python to input three numbers and display the greatest among the three numbers.

Sample Input: First number: 87
Second number: 65
Third number: 34

Output: Greatest Number: 87

```
# a Python program to display greatest number
a=int(input("Enter first number:"))
b=int(input("Enter second number:"))
c=int(input("Enter third number:"))
if(a>b and a>c):
    gr=a
elif(b>a and b>c):
    gr=b
else:
    gr=c
print("Greatest number:", gr)
```

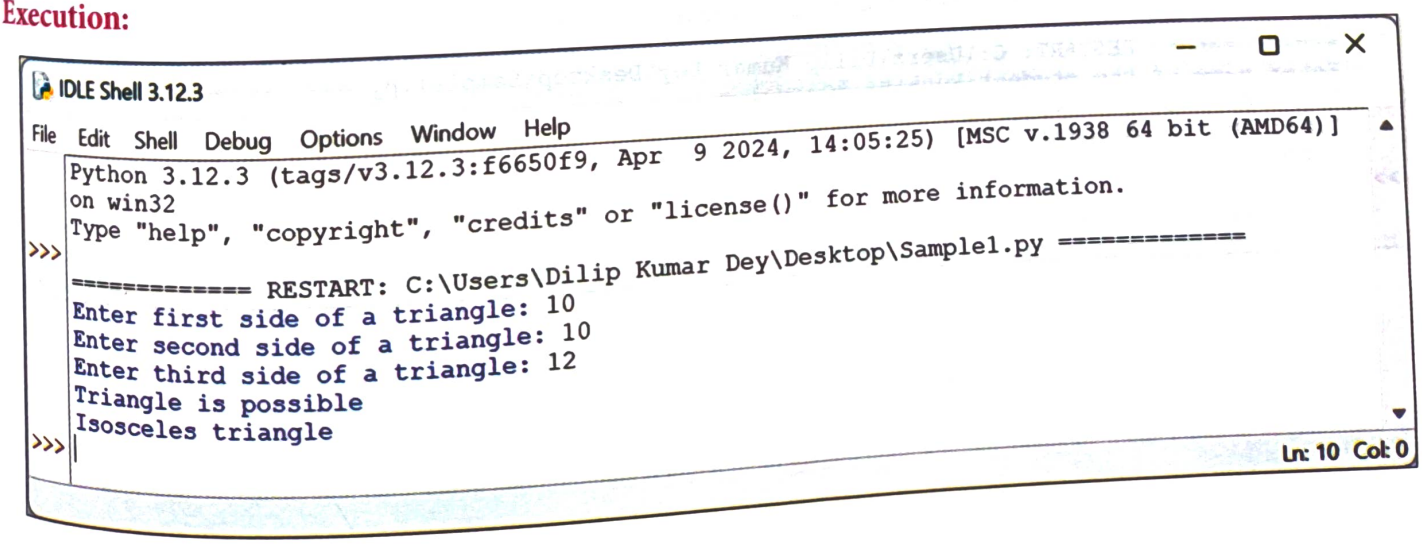
Execution:



Program 8 Write a Python program to enter three sides of a triangle and check whether the triangle is possible or not. If possible then display whether it is an Equilateral, an Isosceles or a Scalene Triangle, otherwise display the message "Triangle is not possible".

```
# a program to display the type of triangle
a = int(input("Enter first side of a triangle: "))
b = int(input("Enter second side of a triangle: "))
c = int(input("Enter third side of a triangle: "))
if((a+b>c)and(b+c>a) and (c+a>b)):
    print("Triangle is possible");
    if((a==b)and(b==c)and(c==a)):
        print("Equilateral triangle")
    elif((a==b)or(b==c)or(c==a)):
        print("Isosceles triangle")
    elif((a!=b)and(b!=c)and(c!=a)):
        print("Scalene triangle")
else:
    print("Triangle not possible")
```

Execution:

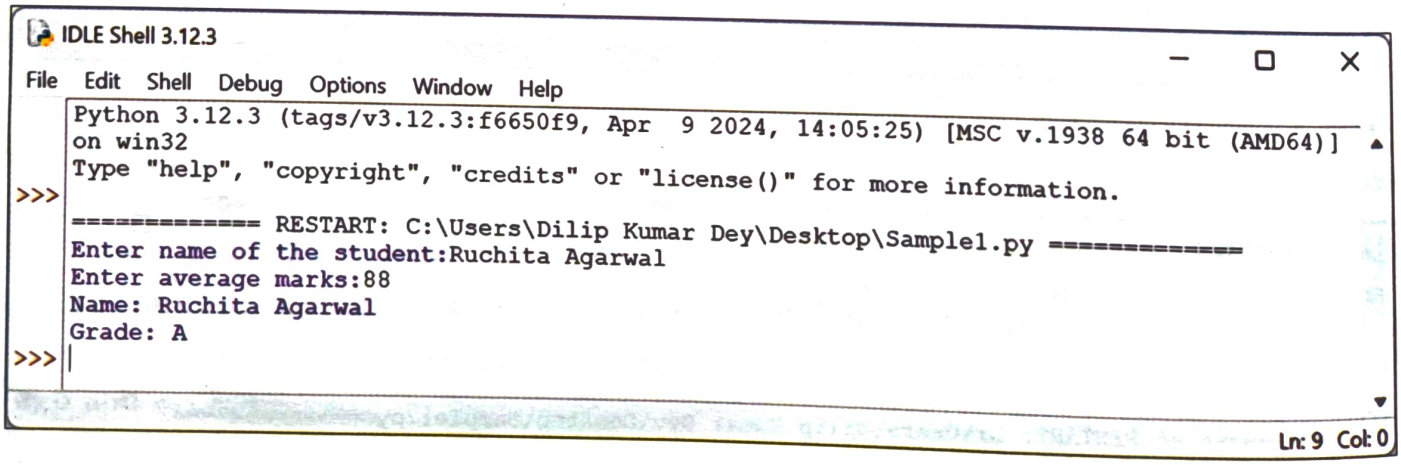


Program 9 Write a program in Python to input name and average marks of a student, obtained in Board Examination. Display the grade as per the table given below:

Average Marks	Grades
>= 90	A+
>= 80 and < 90	A
>= 60 and < 80	B
>= 40 and < 60	C
<40	D

```
# a Python code to display grade
name=input("Enter name of the student:")
avg=int(input("Enter average marks:"))
if(avg>=90):
    gr="A+"
if(avg>=80 and avg<90):
    gr="A"
if(avg>=60 and avg<80):
    gr="B"
if(avg>=40 and avg<60):
    gr="C"
if(avg<40):
    gr="D"
print("Name:", name)
print("Grade:", gr)
```

Execution:

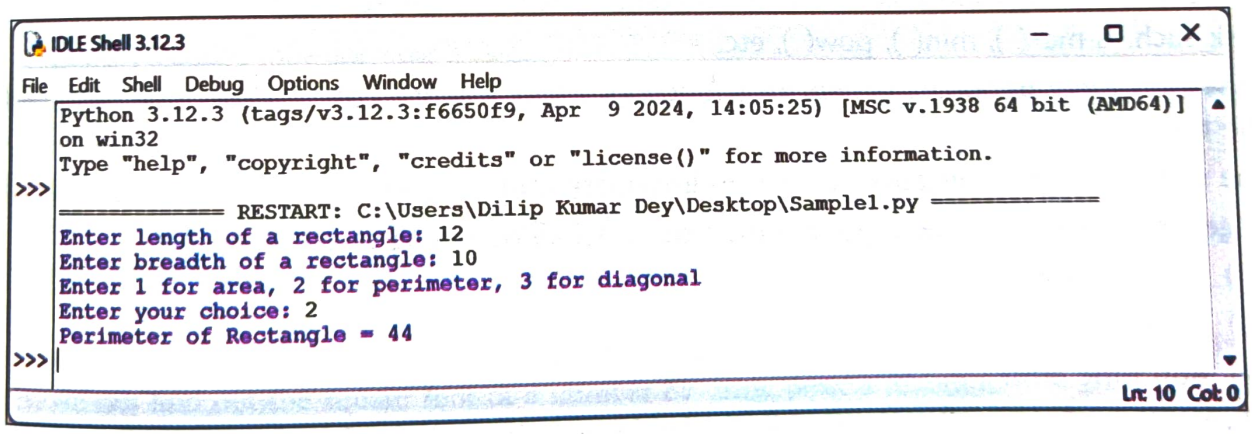


Program 10 Write a program in Python to accept the length and breadth of a rectangle. Calculate and display the area, perimeter and diagonal of the rectangle as per the user's choice.

[Hint: Use match and case]

```
# a program using match and case in Python
import math
a=int(input("Enter length of a rectangle: "))
b=int(input("Enter breadth of a rectangle: "))
print("Enter 1 for area, 2 for perimeter, 3 for diagonal")
choice = int(input("Enter your choice: "))
match choice:
    case 1:
        ar = a*b;
        print("Area of Rectangle =",ar)
    case 2:
        p = 2*(a+b);
        print("Perimeter of Rectangle =",p)
    case 3:
        d= math.sqrt(a*a+b*b);
        print("Diagonal of Rectangle =",d)
    case _:
      print("Wrong Choice!")
```

Execution:





Chapter at a Glance

- The sequential movement of control from one statement to another is termed as normal flow of control.
- Indentation is a system of declaring a statement or a set of statements as a block.
- Conditional statements are used in Python to branch the control based on a logical condition.
- Basically, 'if' statement is used to transfer the control to a specific program location by ignoring some statements.
- When an 'if' statement results in False, the control moves to check another condition by using the 'if-elif' clause.
- Python language allows creating conditional flow of control with the help of if statement. The if statement can be used in different forms as per the requirements of the users or suitability of the program logic. They are:
  (i) if statement
  (ii) if and only if statement
  (iii) if-else statement
  (iv) if- elif- else statement
- The operators are symbols or tokens to perform various mathematical or logical operations.
- The relational operators compare the values of two operands and determine the relationship between them.
- Python language provides three logical operators viz. and, or and not.
- Python provides various mathematical functions that can be used in the codes to make the calculations easier.
- There are some Python mathematical functions that can be used them directly to perform the task such as max( ), min( ), pow( ), etc.
- In Python, there are some mathematical functions where you need to call math module to perform the task.
- The square root of a negative value is an imaginary number and hence it cannot be determined.
- In general, there are three types of errors occur while designing codes. They are:
  (i) Syntax error
  (ii) Logical error
  (iii) Run time error
- When any mathematical expression is divided by zero, it results in ZeroDivisionError. It is because of the fact that a number cannot be divided by zero.