# Avanthi Degree & P.G College

## Narayanaguda –Hyderabad

ESTD-1991

## B.Sc. (Data Science) I Year II Semester

*Problem Solving and Python Programming*

( Paper – II )

Name : _____

Roll No : _____

Group : _____

*The expert in anything was once a beginner*

# Paper – II : Problem Solving and Python Programming

## Unit-I

### Introduction to Computing and Problem Solving:

Fundamentals of Computing – Computing Devices – Identification of Computational Problems – Pseudo Code and Flowcharts – Instructions – Algorithms – Building Blocks of Algorithms.

### Introduction to Python Programming:

Python Interpreter and Interactive Mode– Variables and Identifiers – Arithmetic Operators – Values and Types – Statements, Reading Input, Print Output, Type Conversions, The type() Function and Is Operator, Dynamic and Strongly Typed Language.

### Control Flow Statements:

The if, The if…else,The if…elif…else Decision Control Statements, Nested if Statement, The while Loop, The for Loop, The continue and break Statements.

## Unit-II

### Functions:

Built-In Functions, Commonly Used Modules, Function Definition and Calling the Function, The return Statement and void Function, Scope and Lifetime of Variables, Default Parameters, Keyword Arguments, *args and **kwargs, Command Line Arguments.

### Strings:

Creating and Storing Strings, Basic String Operations, Accessing Characters in String by Index Number, String Slicing and Joining, String Methods, Formatting Strings.

## Unit-III

### Lists:

list operations, list slices, list methods, list loop, mutability, aliasing, cloning lists, list parameters;
Tuples: tuple assignment, tuple as return value;
Dictionaries: operations and methods; advanced list processing - list comprehension;
Illustrative programs: selection sort, insertion sort, merge sort, histogram.

### Files and exception:

Text files, reading and writing files, format operator; command line arguments, errors and exceptions, handling exceptions, modules, packages; Illustrative programs: word count, copy file.

## Unit-IV

### Object-Oriented Programming:

Classes and Objects, Creating Classes in Python, Creating Objects in Python, The Constructor Method, Classes with Multiple Objects, Class Attributes versus Data Attributes, Encapsulation, Inheritance The Polymorphism.

**Functional Programming:** Lambda. Iterators, Generators, List Comprehensions.

## References :

1. Introduction to Python Programming. Gowrishankar S., Veena A. CRC Press, Taylor & Francis Group, 2019

2. Allen B. Downey, ``Think Python: How to Think Like a Computer Scientist'', 2nd edition, Updated for Python 3, Shroff/O'Reilly Publishers, 2016 (http://greenteapress.com/wp/think- python/)

**Suggested Reading :**

1. Learning To Program With Python. Richard L. Halterman. Copyright © 2011
2. Python for Everybody, Exploring Data Using Python 3. Dr. Charles R. Severance. 2016

## *Practical - 2 :* Problem Solving and Python Programming (Lab)

**Installing Python and Setting up the Environment**

Python interpreter can be downloaded for Windows/Linux platform using the link below: https://www.python.org/downloads/

**Exercises**

**I. Programs to demonstrate the usage of operators and conditional statements**

1. Write a program that takes two integers as command line arguments and prints the sum of two integers.
2. Program to display the information: Your name, Full Address, Mobile Number, College Name, Course Subjects
3. Program to find the largest number among 'n' given numbers.
4. Program that reads the URL of a website as input and displays contents of a webpage.

**II. Programs to demonstrate usage of control structures**

5. Program to find the sum of all prime numbers between 1 and 1000.
6. Program that reads set of integers and displays first and second largest numbers.
7. Program to print the sum of first 'n' natural numbers.
8. Program to find the product of two matrices.
9. Program to find the roots of a quadratic equation

**III. Programs to demonstrate the usage of Functions and Recursion**

10. Write both recursive and non-recursive functions for the following:
    a. To find GCD of two integers
    b. To find the factorial of positive integer
    c. To print Fibonacci Sequence up to given number 'n'
    d. To convert decimal number to Binary equivalent
11. Program with a function that accepts two arguments: a list and a number 'n'. It should display all the numbers in the list that are greater than the given number 'n'.
12. Program with a function to find how many numbers are divisible by 2, 3,4,5,6 and 7 between 1 to 1000

**IV. Programs to demonstrate the usage of String functions**

13. Program that accept a string as an argument and return the number of vowels and consonants the string contains.
14. Program that accepts two strings S1, S2, and finds whether they are equal are not.
15. Program to count the number of occurrences of characters in a given string.
16. Program to find whether a given string is palindrome or not

**V. Programs to demonstrate the usage of lists, sets, dictionaries, tuples and files..**

17. Program with a function that takes two lists L1 and L2 containing integer numbers as parameters. The return value is a single list containing the pair wise sums of the numbers in L1 and L2.
18. Program to read the lists of numbers as L1, print the lists in reverse order without using reverse function.
19. Program to find mean, median, mode for the given set of numbers in a list.
20. Program to find all duplicates in the list.
21. Program to o find all the unique elements of a list.
22. Write a program that combine lists L1 and L2 into a dictionary.
23. Program to find max and min of a given tuple of integers.
24. Program to find union, intersection, difference, symmetric difference of given two sets.
25. Program to display a list of all unique words in a text file
26. Program to read the content of a text file and display it on the screen line wise with a line number followed by a colon
27. Program to analyse the two text files using set operations
28. Write a program to print each line of a file in reverse order.

**VI. Programs to demonstrate the usage of Object Oriented Programming**

29. Program to implement the inheritance
30. Program to implement the polymorphism

**VII. Programs to search and sort the numbers**

31. Programs to implement Linear search and Binary search
32. Programs to implement Selection sort, Insertion sort

# Chapter 1:
# Introduction to Computing and Problem Solving:

## 1.1 Fundamentals of Computing – Computing Devices

Computing devices are the electronic devices which take inputs, process the inputs and then calculate results from the inputs. One of the most common computing device is computer which we all know about it.

This is the vast field and it covers huge amount of devices today which we are using in one or another form and these devices ranging from the simple Calculator, Printers, Digital Cameras, Mobile phones, Smart phones , PC Desktops, Laptops and many more. These all do the math calculations on the streams of binary digits (bits i.e. 0 and 1 ) and produce the results in different forms that we need.

### 1.1.1 Processors

Every computing device will have the electronic calculator i.e. **processor.** This executes calculations from the instructions given to them in the form of programming code.

### 1.1.2 Programming

Programming is the basic for executing computations. There can be two kinds of programming for computers and that are Operating systems and Applications.

An O*perating system (OS)* helps the people to interact with the computing machine

*Applications* also known as apps, are examples of specialized programs that enable the computer to carry out specific tasks.

### 1.1.3 Data Storage

Data storage is basically storing the programs as well as data on different memory devices used in the computing devices today and these programs and data which are stored can be used later for our needs. There are different storage devices using today.

### 1.1.4 Types of General-purpose Computers

There are basically two kinds of General-purpose computers and they are

1. **Multi-user Computers**
2. **Dedicated Computers**

#### Multi-user Computers

As the name suggests that this is a kind of computer that can handle multi users at a time. Mainframes, minicomputers, and supercomputers all fall into this category.

#### Dedicated Computers

*Dedicated computers* are the computers which can be used by one user at a time. These computers have replaced multi-user computers because they are simple and have less cost. PC (Personal Computer is the most common dedicated computer these days

There are further two kinds of PCs which depends on their sizes and portability.

These PCs are **Desktops** and **Laptops.**

### 1.1.5 OTHER COMPUTING DEVICES

A personal computer is designed to be a general-purpose device. That is, it can be used to solve many different types of problems. several types of devices that represent the latest trends in personal computing.

PORTABLE COMPUTERS, SMARTPHONES, TABLET COMPUTERS.

Computing technology is being integrated into many everyday products. From automobiles to refrigerators to airplanes, etc.

Here are two of the latest ways that computing technologies are being integrated into everyday products:

The Smart House
The Self-Driving Car

## 1.2 Identification of Computational Problems

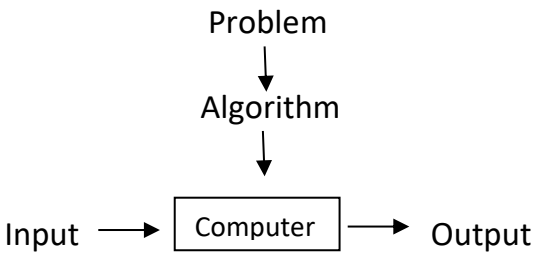
It is any type of calculation that includes both arithmetical and non-arithmetical steps and follows a well-defined model

### 1.2.1 ALGORITHMS :

An algorithm is method of representing step by step procedure for solving a problem.

➔ An algorithm is a part of the plan for the computer program. In fact, an algorithm is 'an effective procedure for solving a problem in a finite number of steps'.

➔ An algorithm is very useful for finding the right answer to a problem or breaking the difficult problem into simple cases.




**Steps for Algorithm Development:-**

1. Initially understand the problem
2. Then, identify the expected output for the problem
3. Identify the necessary input for the problem.
4. Develop the logic that produces the expected output from the selected input.
5. Finally, test the algorithm with various set of inputs.

**Characteristics of Algorithm:-**

An Algorithm must process the following characteristics

1. Finiteness
2. Definiteness
3. Effectiveness
4. Input
5. Output

1. **Finiteness**
   An Algorithm should terminate in a finite number of steps
2. **Definiteness**
   Each step of algorithm must be clear and unambiguous.
3. **Effectiveness**
   Each step must be effective, in the sense that, it should be easily convertible into program
   statement and can be performed exactly in a finite amount of time.
4. **Input**
   Each algorithm must take Zero, one or more quantities as input data
5. **Output**
   Each algorithm must produce at least one output.

## Different ways of stating Algorithms

An algorithm can be written in following two ways:

1. Pseudo code
2. Flow chart

### 1.2.1.1 Pseudo Code :-

- ➔ The word pseudo means imitation and code means instructions written some programming language.
- ➔ Pseudo code describes the structure of program in simple English language.
- ➔ Pseudo code describes the complete logic of some program so that implementation become easy. It does not use any syntax or symbols.

**Example:** Adding two numbers

Step1: Start
Step2: Read input variable 'A'.
Step3: Read input variable 'B'.
Step4: compute the sum of A,B
Step5: Store the sum value in 'C'.
Step6: Print value 'C'
Step7: End

### 1.2.1.2 FLOWCHARTS:-

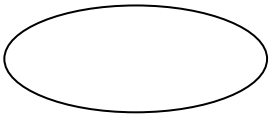
- ➔ A Flowchart is a type of diagram (graphical or symbolic) that represents an algorithm or process.
- ➔ Each step in the process is represented by a different symbol and contains a short description of the process step.
- ➔ The flow chart symbols are linked together with arrows showing the process flow direction.

- ➔ A flowchart describes what operations (and in what sequence) are required to solve a given problem
- ➔ Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields

## Flowchart Symbols

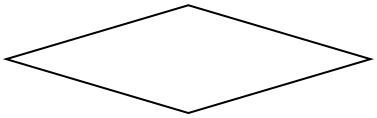
**Terminator / Terminal:**

An oval flow chart shape indicates the start or end of the process, usually containing the word "Start" or "End".

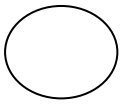
 or 

**Process:** A rectangular flow chart shape indicates a normal/generic process flow step. For example, "Add 1 to X", "M = M*F" or similar

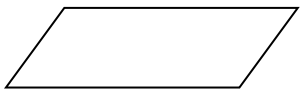
**Decision:** A diamond flow chart symbol is used when a decision needs to be made, commonly a Yes/No question or True/False test.



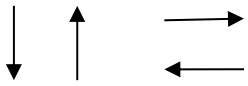
**Connector**: A small, labelled, circular flow chart shape used to indicate a jump in the process flow. Connectors are generally used in complex or multi-sheet diagrams



**Data:** A parallelogram that indicates data input or output (I/O) for a process. Examples: Get X from the user, Display X.



**Arrow:** used to show the flow of control in a process.



## 1.3 BUILDING BLOCKS OF ALGORITHMS (statements, state, control flow, functions)

Algorithms can be constructed from basic building blocks namely, sequence, selection and iteration.

### 1.3.1 Statements:

Statement is a single action in a computer. In a computer statements might include some of the following actions

- input data-information given to the program
- process data-perform operation on a given input
- output data-processed result

### 1.3.2 State:

Transition from one process to another process under specified condition with in a time is called state.

### 1.3.3. Control flow:

The process of executing the individual statements in a given order is called control flow.

The control can be executed in three ways

1. sequence
2. selection
3. iteration

#### 1.3.3.1 Sequence:

All the instructions are executed one after another is called sequence execution.

#### 1.3.3.1 Selection:

- A selection statement causes the program control to be transferred to a specific part of the program based upon the condition.
- If the conditional test is true, one part of the program will be executed, otherwise it will execute the other part of the program.

#### 1.3.3.2 Iteration:

In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time.

This type of execution is called looping or iteration.

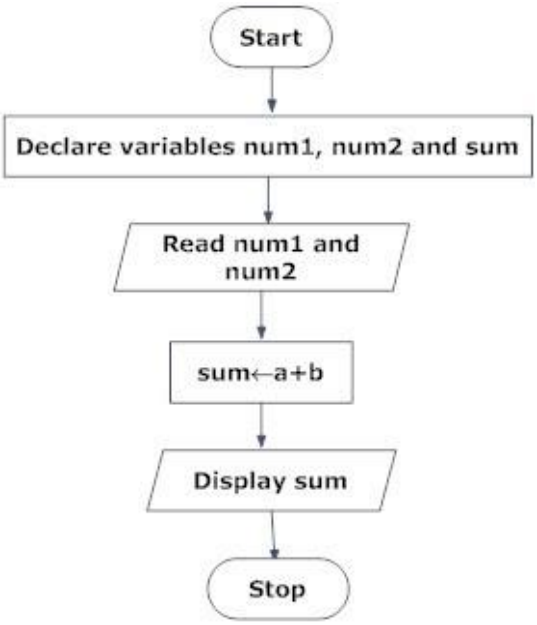
### 1.3.4 Functions:

- Function is a sub program which consists of block of code(set of instructions) that performs a particular task.
- For complex problems, the problem is been divided into smaller and simpler tasks during algorithm design.

## Example 1: To find sum of two numbers

**Pseudo Code Algorithm:-**

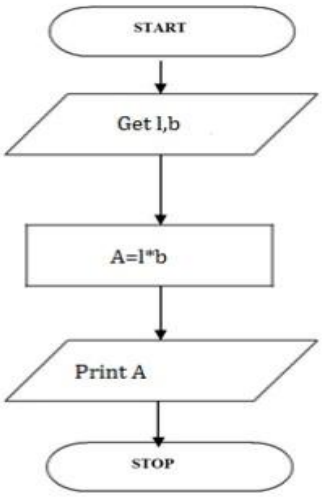
Step 1: Start/begin
Step 2: Declare Num1, Num2 and Sum
Step 3: Read Num1 , Num2
Step 4: Sum ⟵ Num1 + Num2
Step 5: Print or Display Sum
Step 6: Stop/End




## Example 2: To find Area of Rectangle

**Pseudo Code Algorithm:-**

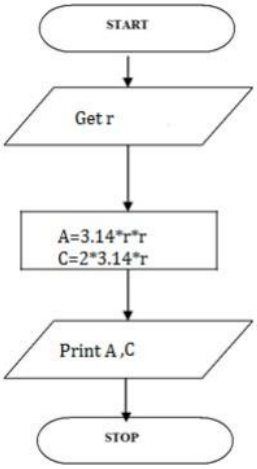
Step 1: Start
Step 2: get l,b values
Step 3: Calculate A=l*b
Step 4: Display A
Step 5: Stop




## Example 3: Write an algorithm for Calculating area and circumference of circle

**Pseudo Code Algorithm:-**

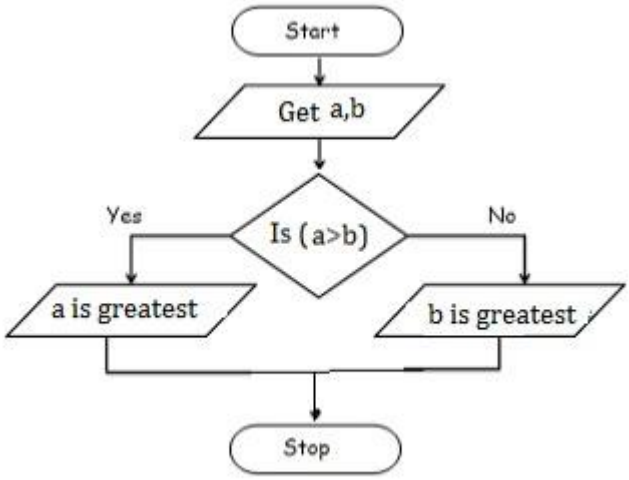
Step 1: Start
Step 2: get r value
Step 3: Calculate A=3.14*r*r
Step 4: Calculate C=2.3.14*r
Step 5: Display A,C
Step 6: Stop



## Example 4: Greatest of two numbers

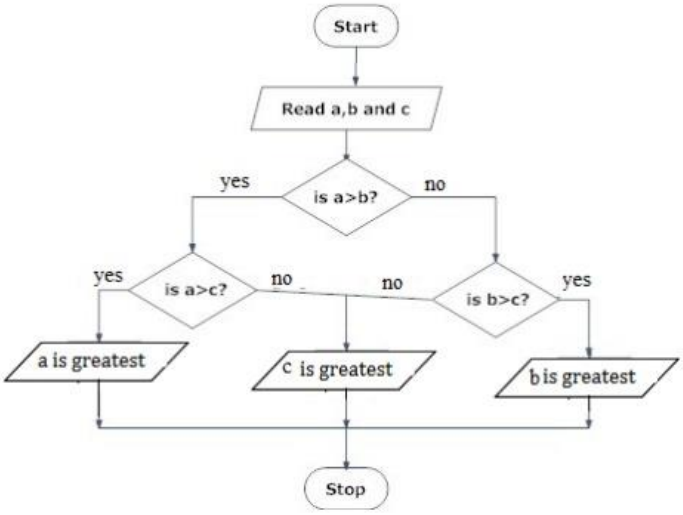
**Pseudo Code Algorithm:-**

Step 1: Start
Step 2: get a,b value
Step 3: check if(a>b) print a is greater
Step 4: else b is greater
Step 5: Stop




## Example 5: Write an algorithm & Flow chart to find the largest among three different numbers entered by user.

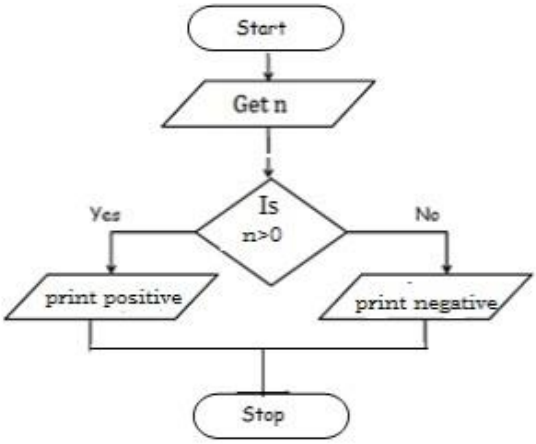
Step1: Start
Step2: Get A, B, C
Step3: if(A>B) goto Step4 else goto step5
Step4: If(A>C) print A else print C
Step5: If(B>C) print B else print C
Step6: Stop
.




## Example 6: To check positive or negative number

**Pseudo Code Algorithm:-**

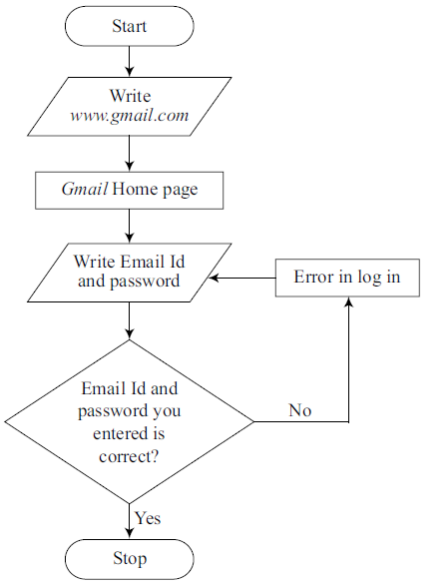
Step 1: Start
Step 2: get num
Step 3: check if(num>0) print a is positive
Step 4: else num is negative
Step 5: Stop



**Example 7: Write an algorithm and Draw a flow chart to log in to your Gmail Account**

**Pseudo Code Algorithm:-**

Step 1: Start/begin
Step 2: Go to www.gmail.com.
Step 3: . Enter your email id and password
Step 4: Click the Sign in button
Step 5: Stop




**Assignment:-**

1. To check odd or even number
2. Write an algorithm to find sum of a given number
3. Write an algorithm to find factorial of a given number
4. Check given number is prime or not
5. GCD of two numbers

# Chapter 2:
# Introduction to Python Programming:

## 2.0 Python Introduction

- Python is the world's most popular and fastest-growing computer programming language. It is a multi-purpose and high-level programming language.
- Python was invented by **Guido Van Rossum** in the year 1989, but it was introduced into the market on 20th February 1991.

- The Python programming language has been used by many people like Software Engineers, Data Analysts, Network Engineers, Mathematicians, Accountants, Scientists, and many more. Using Python, one can solve complex problems in less time with fewer lines of code.

## Features of Python

The Python is a very popular programming language with the following features.

- Python is easy to learn and easy to understand.
- The Python is an interpreted programming language. It executes the code line by line.
- The Python is a cross-platform programming language. It can be used with any operating system like Windows, Linux, MAC OS, etc.
- The Python is a free and open-source programming language.
- The Python is an Object-Oriented, Procedural and Functional programming language
- The Python is a multi-purpose programming language
- The Python is a high-level programming language
- Python has a huge Community to get help all over the globe.
- Python has a large Ecosystem of Libraries, Frameworks, and Tools to work with it.

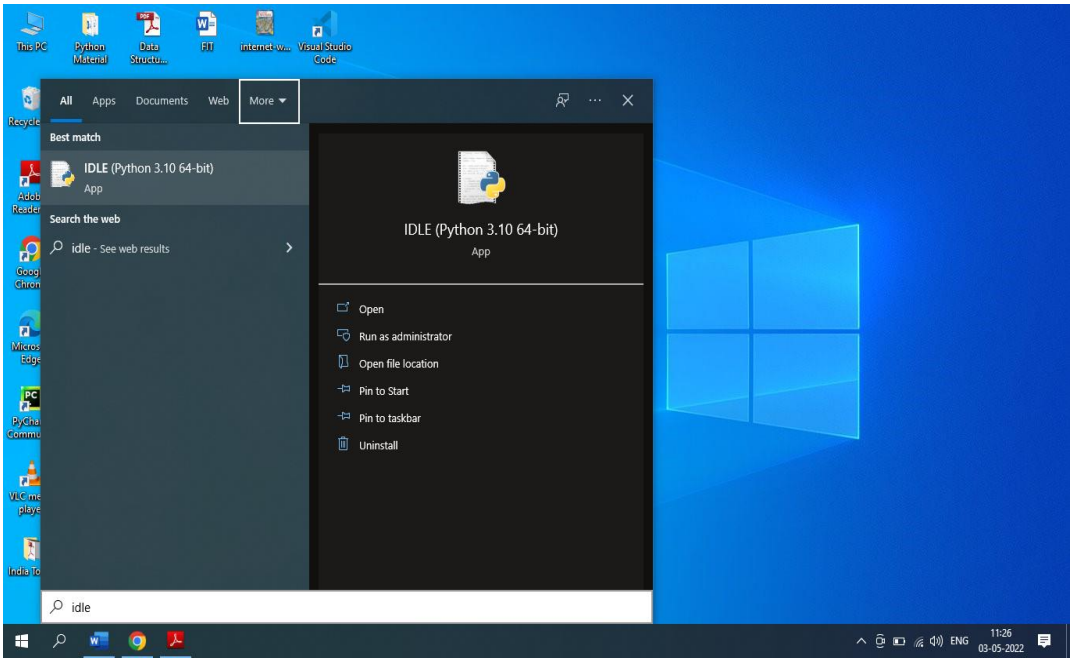
## 2.1 Python Interpreter

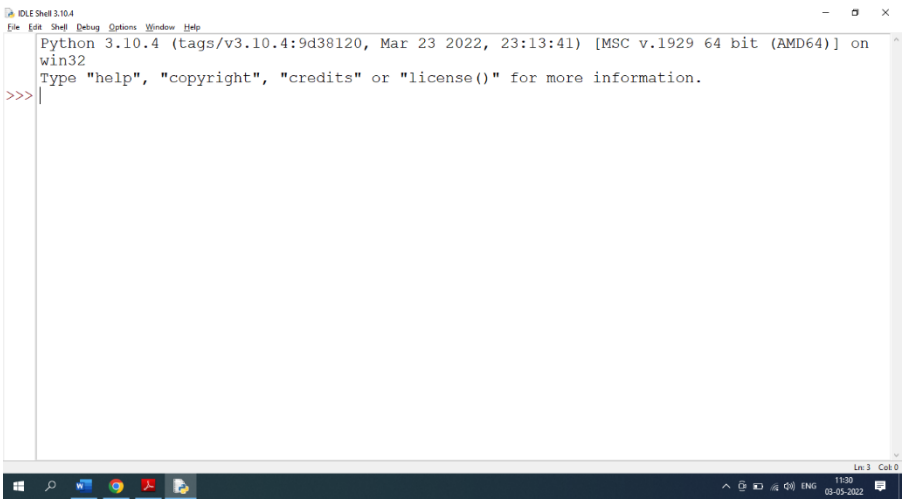
Python Interpreter is a program that reads and executes Python code. It uses 2 modes of Execution.

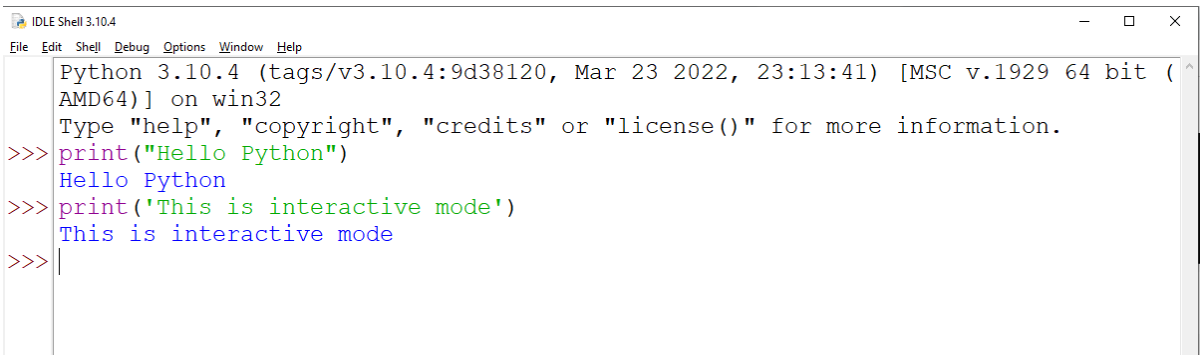
1. Interactive mode
2. Script mode

### 2.1.1 Python Interactive Mode

- In the interactive mode, We can query some statements, and python will interpret/execute them and give you the output.
- Interactive mode is useful to test out our code before keeping them to the scripts.
- We can do operations like simple calculations, and printing some variables and etc.
- To bring up the interactive python shell, Search for `IDLE` in windows.
- For Linux and Mac user, Goto `Terminal` and bring up the interactive python shell by running `python` command.



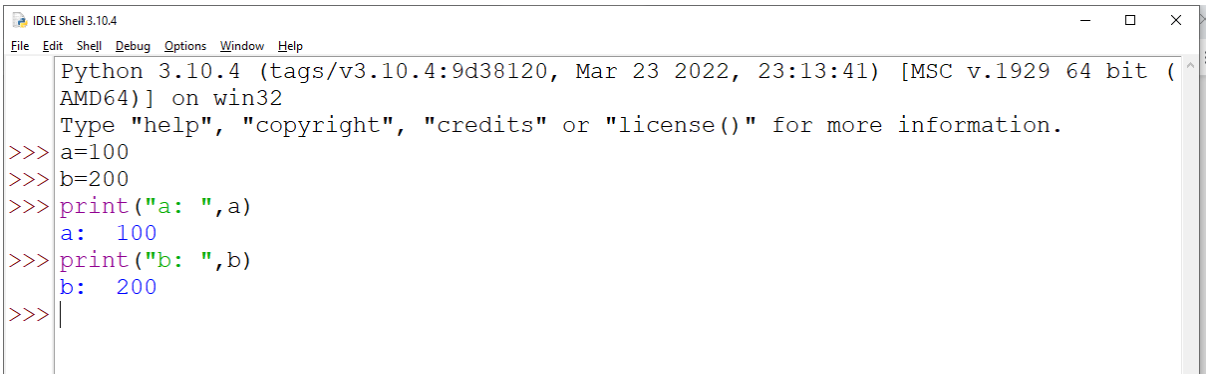



We can write our statements into interactive shell, In above case we written `print("Hello Python")` statement and pressed `enter`.

Then python immediately executed our `print` statement and displayed the output.

This is called **interactive mode.**

Where we can perform simple operations and get the results.




### Disadvantages of interactive mode

- The interactive mode is not suitable for large programs.
- The interactive mode doesn't save the statements. Once we make a program it is for that time itself, we cannot use it in the future. In order to use it in the future, we need to retype all the statements.
- Editing the code written in interactive mode is a tedious task. We need to revisit all our previous commands and if still, we could not edit we need to type everything again.

## 2.1.2 Python Script Mode

Script mode in Python is where we first write the Python program inside a script file and execute it after that.

We can execute the script of code either using the **Command prompt** or using **Python IDE** installed in our system.

In order to run a code in script mode follow the following steps.

**Step 1:** Make a file using a text editor. You can use any text editor of your choice (Here I use notepad).

**Step 2:** After writing the code save the file using **".py"** extension.

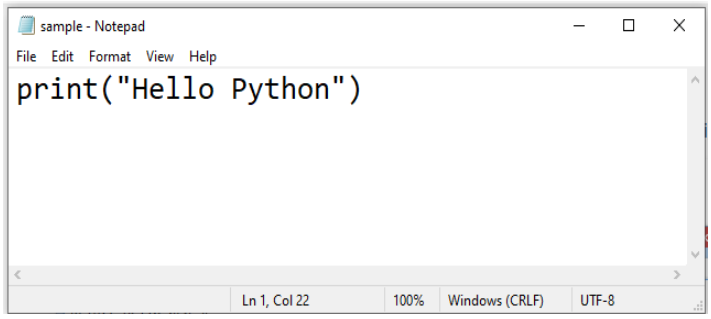
**Step 3:** Now open the command prompt and command directory to the one where your file is stored.

**Step 4:** Type python "filename.py" and press enter.

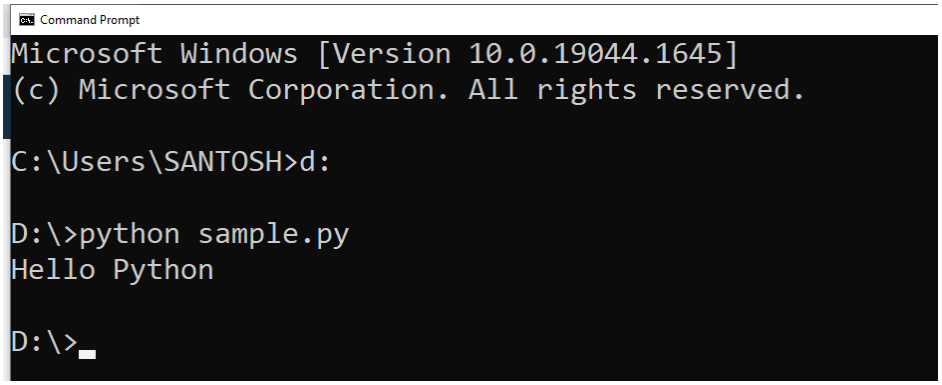
**Step 5:** You will see the output on your command prompt.

**Example:**

In order to execute "Hello Python" using script mode we first make a file and save it




Now we use the command prompt to execute this file.




## 2.2 Variables

- In Python, a variable is a named memory where a programmer can store data and retrieve for future use using the same name.
- In Python, variables are created without specifying any data type.
- There is no specific keyword used to create a variable. Variables are created directly by specifying the variable name with a value.

We use the following syntax to create a variable.

Syntax:

```
variable_name = value
```

**Note :**

When a variable is defined, we must create it with a value. If the value is not assigned the variable gives an error stating that variable is not defined.

## 2.3 Identifiers

An identifier is a name given to a variable, function, class or module.

Identifiers may be one or more characters in the following format:

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).

  Examples : `myCountry, other_1 and good_morning`
- A Python identifier can begin with an alphabet (A – Z and a – z and _ ).
- An identifier cannot start with a digit but is allowed everywhere else

  Example : `1num is invalid`

  `num1 is valid`
- Keywords cannot be used as identifiers
- One cannot use spaces and special symbols like !, @, #, $, % etc. as identifiers.
- Identifier can be of any length.

## 2.4 Python IO Operators

In Python, input and output operations (OI Operations) are performed using two built-in functions. Following are the two built-in functions to perform output operations and input operations.

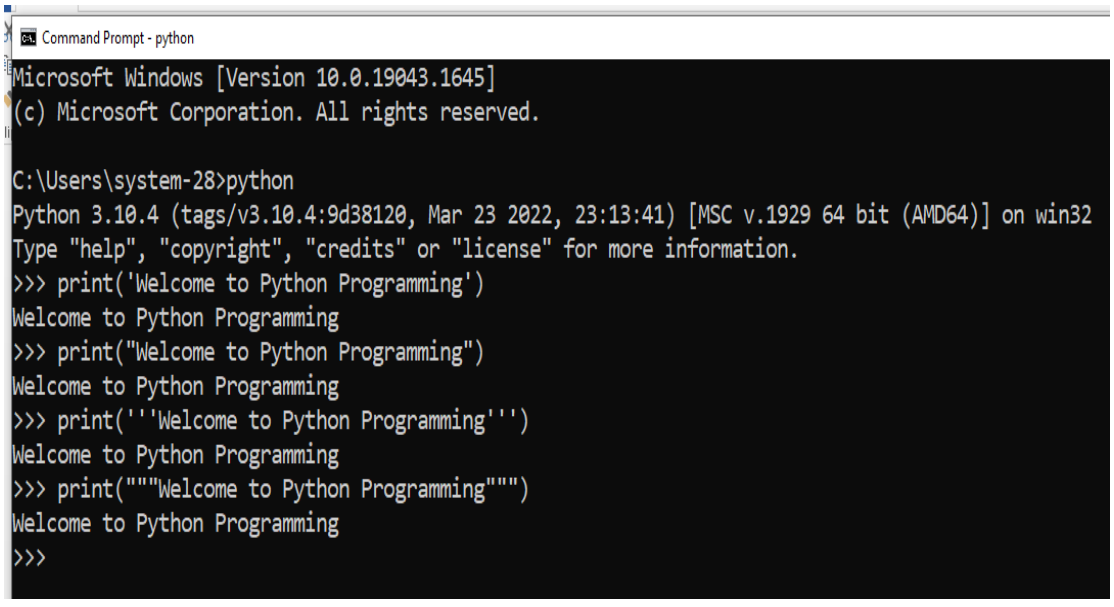
**print( )** - Used for output operation.
**input( )** - Used for input operations.

### Output Operations using print() function

The built-in function **`print( )`** is used to output the given data to the standard output device (Screen).

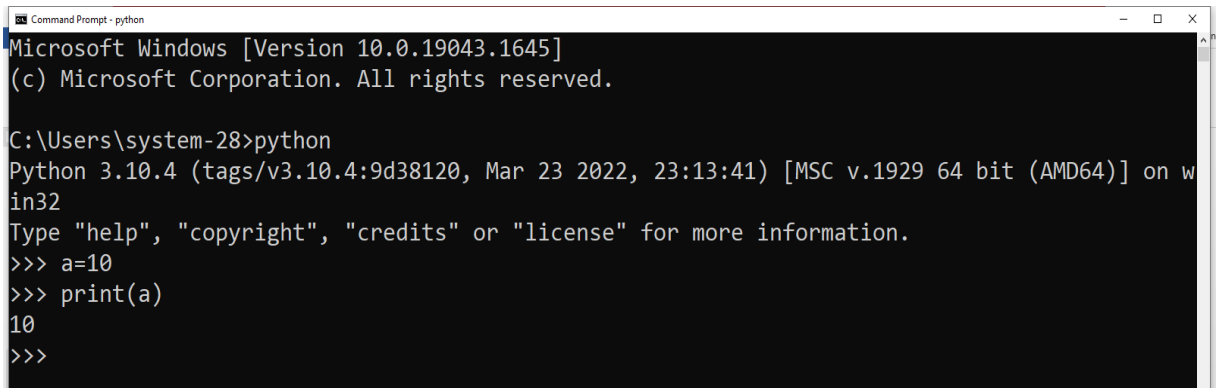
### Displaying a message using print( ) function

In Python, using the print( ) function we can display a text message. The print( ) takes a string as an argument and displays the same.



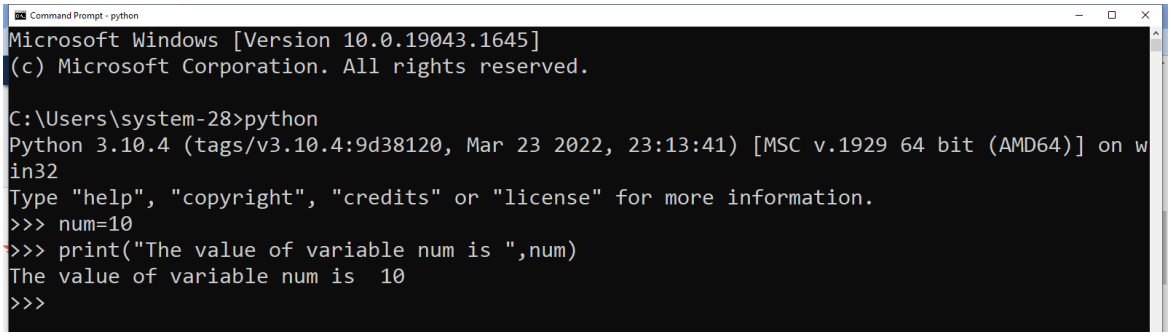
## Displaying a variable value using `print( )` function

In Python, the built-in function **`print( )`** function is also used to display variables value. The following example shows that how to display variable value using **`print( )`** function.

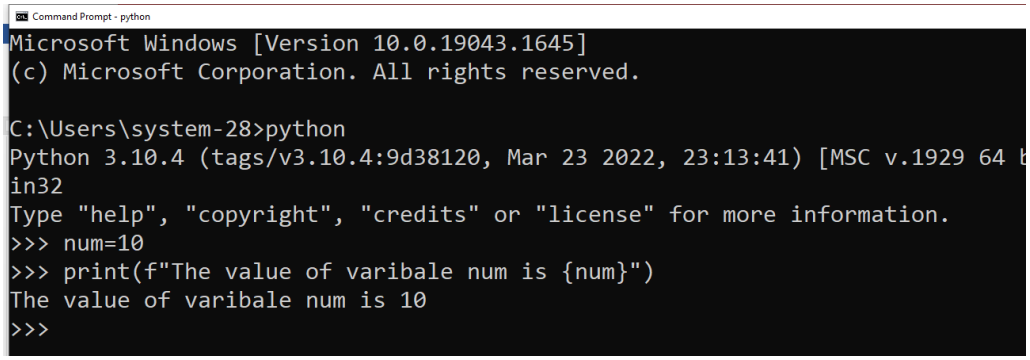



## Formatted `print( )` function to display a combination of message and value

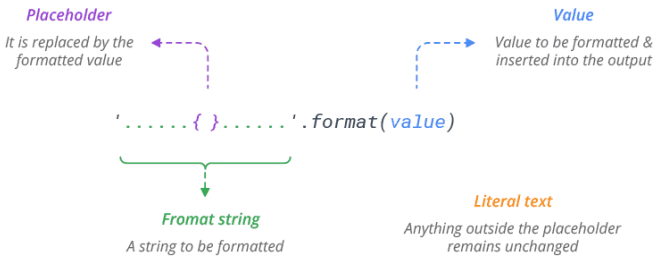
The built-in function **`print( )`** is also used to display the combination of message and variable value. Let's look at the following example.




In Python, we can use the *formatted string* that is prefixed with character " *f* ".
In the formatted string, the variable values are included using curly braces ({ }).




The format strings will contain the curly braces { } and the format() method will use those curly braces { } as placeholders to replace with the content of the parameters.



```
Command Prompt - python
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\system-28>python
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64
in32
Type "help", "copyright", "credits" or "license" for more information.
>>> num=10
>>> print("The value of the num variable is {0}".format(num))
The value of the num variable is 10
>>>
```

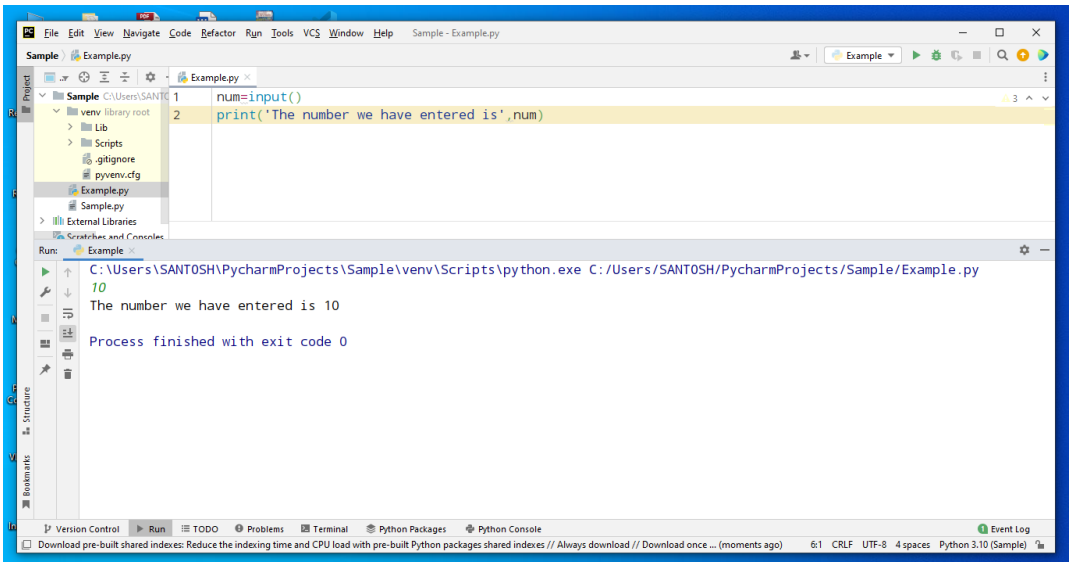
## Input Operations using input() function

The Python provides a built-in function **input( )** to perform input operations. The input( ) function can be used either with some message or without a message.

When input( ) function is used without a message, it simply prompts for the input value. When the user enters the input value it reads the entered value as a string and assigns it to the left-hand-side variable.

**Example :**

```
num=input()
print("The number we have entered is ", num)
```

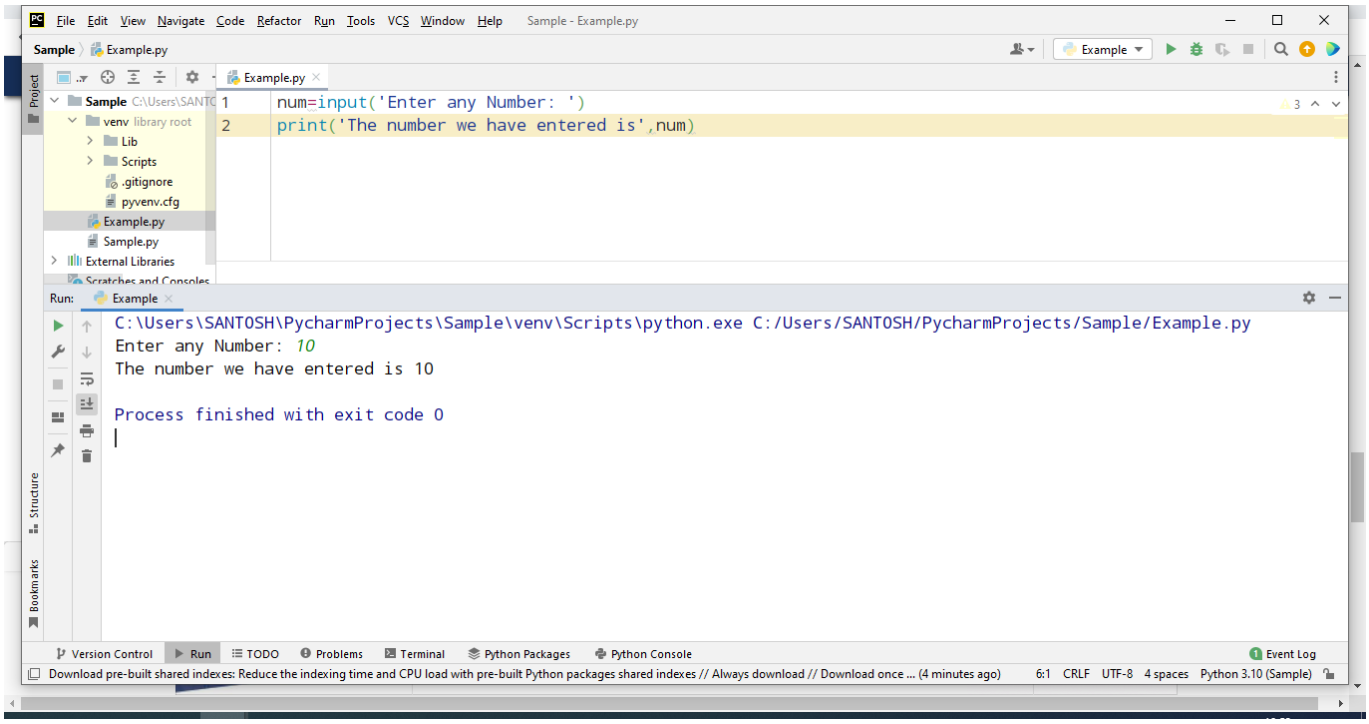



Here, the major problem is that the user does not have any information regarding what is the next step the user has to do? To solve this problem, we use the input( ) function with a message which tells the user that what the user has to do?

When input( ) function is used with a message, it prompts with a given message for the input value. When the user enters the input value it reads the entered value as a string and assigns it to the left-hand-side variable.

**Example :**

```
num=input('Enter any Number : ')
print("The number we have entered is ", num)
```




Always the **`input( )`** function reads input value as **`string`** value only.
To read the value of any other data type, there is no input function in Python.
Explicitly we need to convert to the required data type using type casing.

### Declaring multiple variables in a single statement :

In Python, it is possible to define more than one variable using a single statement. When multiple variables are created using a single statement, the variables and their corresponding value must be separated with a comma symbol. Let's look at the following code which creates two variables with two different values.

**Example :**

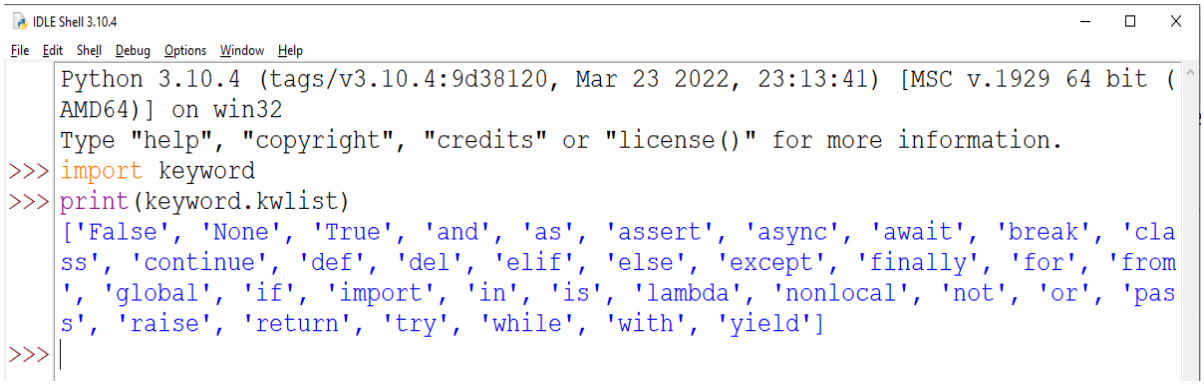
```
name,rollno= ('Santhosh',1)
print("Student name is ",name," and " , "Roll number is ",rollno )
```

## 2.5 Keywords

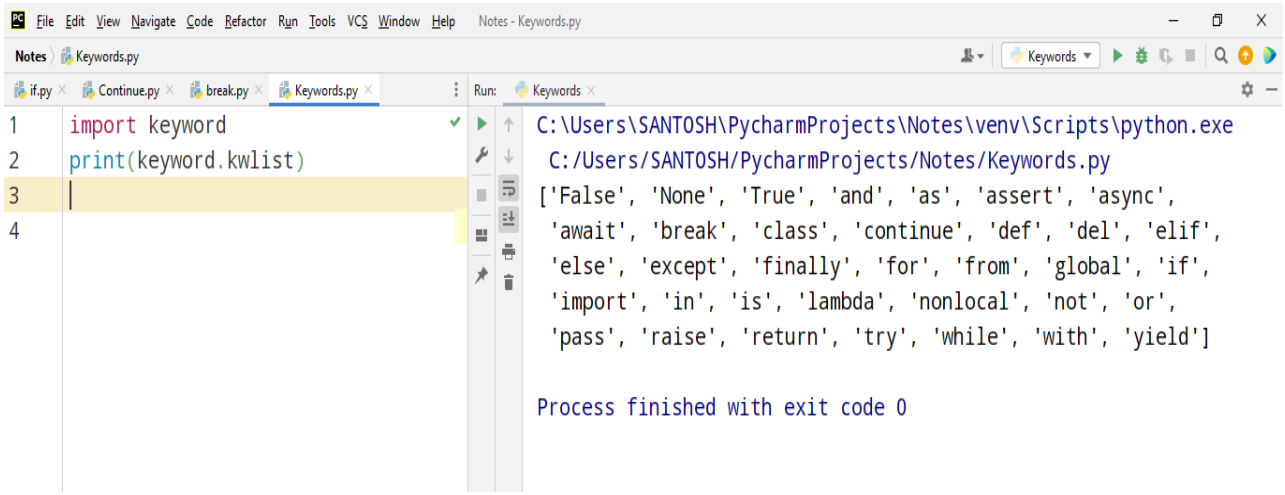
- Keywords are the reserved words in the Python programming language. All keywords are designated with a special meaning to each.
- The meaning of all keywords is fixed and it cannot be modified or removed. All the keywords need to be used as they have defined (Lower case or Upper case).
- In Python 3.10's , there are 35 keywords .

**Following is the Python code used to display all the keywords in Python 3.10.**

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




**Sample program to display all the keywords in Python using PyCharm IDE**




## Points to be Remembered

- All the keywords must be used as they have defined.
- Keywords should not be used as identifiers like variable names, functions name, class name, etc.
- The meaning of each keyword is fixed, we cannot modify or remove it.

## 2.6 Python Data Types

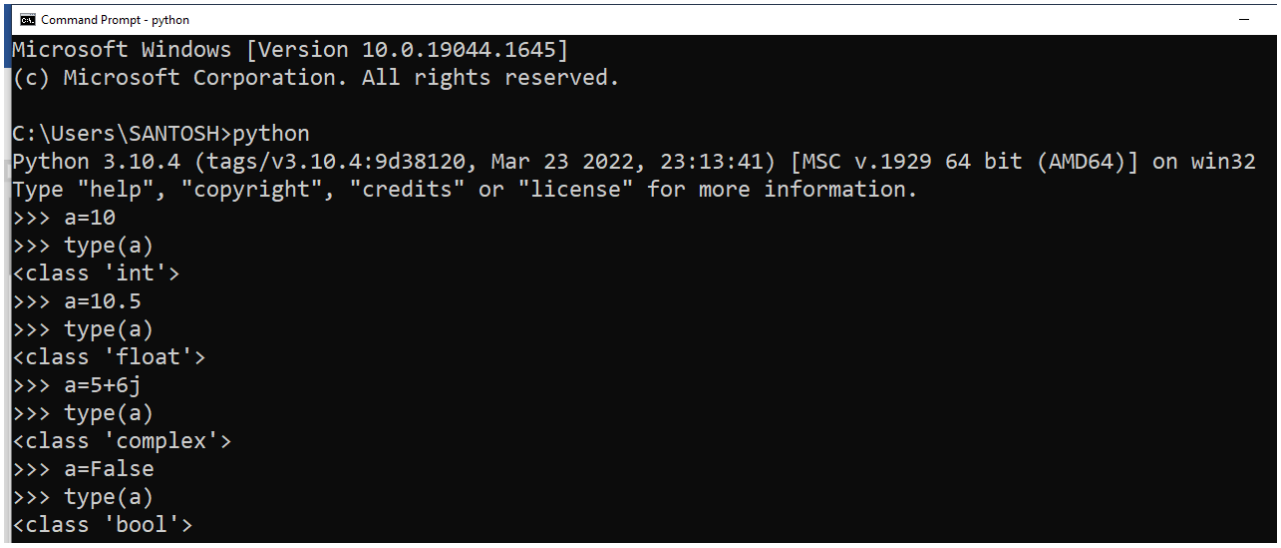
Data types specify the type of data like numbers and characters to be stored and Manipulated within a program. Basic data types of Python are

- Numbers
- Boolean
- Strings
- None

### 2.6.1 Numbers /Numeric

The Python programming language provides four numeric / Numbers data types. They are as follows.

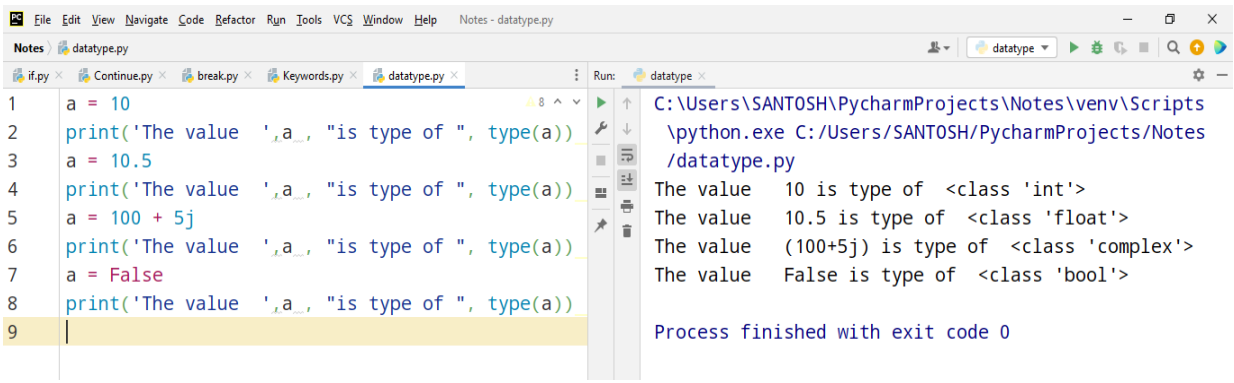
- **int** - All the numbers without fraction part (Example : 10). For int, there is no upper limit.
- **float** - All the numbers with a fraction part (Example : 10.5). It's accurate up to 15 decimal places
- **complex** – Complex numbers are written in the form $x + yj$ , where *x* is the real part and *y* is the imaginary part. (Example : $5 + 10j$ ).
- **bool** - boolean values True and False.




Example - Python code to illustrate 'Numeric / Numbers' data type

```
a = 10
print('The value  ',a , "is type of ", type(a))
a = 10.5
print('The value  ',a , "is type of ", type(a))
a = 100 + 5j
print('The value  ',a , "is type of ", type(a))
a = False
print('The value  ',a , "is type of ", type(a))
```

When we run the above code, it produces the output as follows.



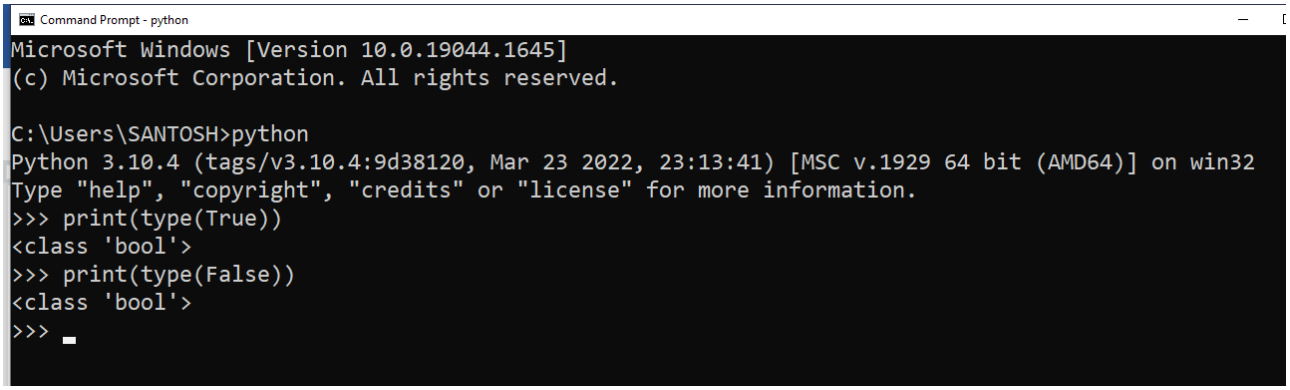
## 2.6.2 Boolean

The python data type `bool` is used to store two values i.e `True` and `False`.
Bool is used to test whether the result of an expression is true or false. The Boolean values, `True` and `False` are treated as reserved words.

### Syntax

To check the boolean value of an expression or a variable, pass it as a parameter to the bool function:

```
print(bool(expression))
    or
print(expression)
```



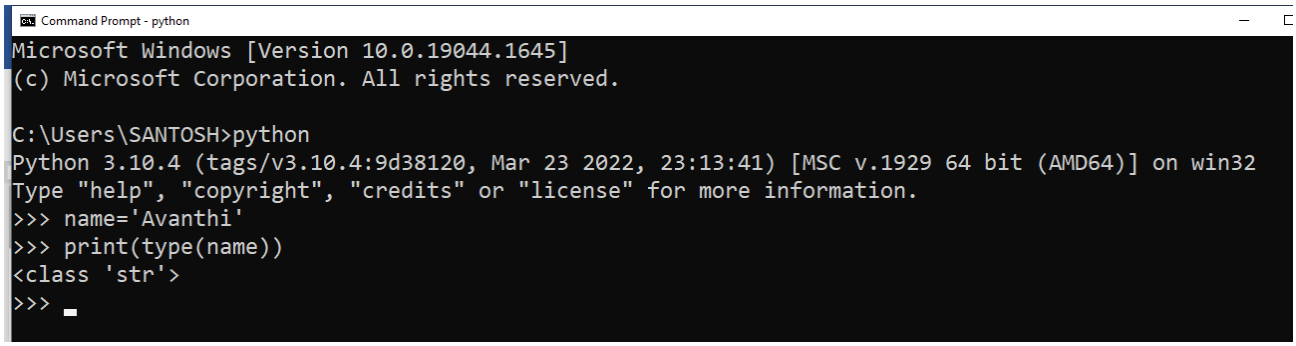

## 2.6.3 Strings

A string consists of a sequence of one or more characters, which can include letters, numbers, and other types of characters.

A string can also contain spaces. You can use single quotes or double quotes to represent strings and it is also called a string literal.

Multiline strings can be denoted using triple quotes, ‘‘‘ or “ “ “.

The string data type in Python is called as **`'str '`**.



## Tips!

- When a string contains a single quotation as a character in it, then enclose it with double quotes.

  (Example - "It's cool")

- When a string contains double quotation as a character in it, then enclose it with single quotes.

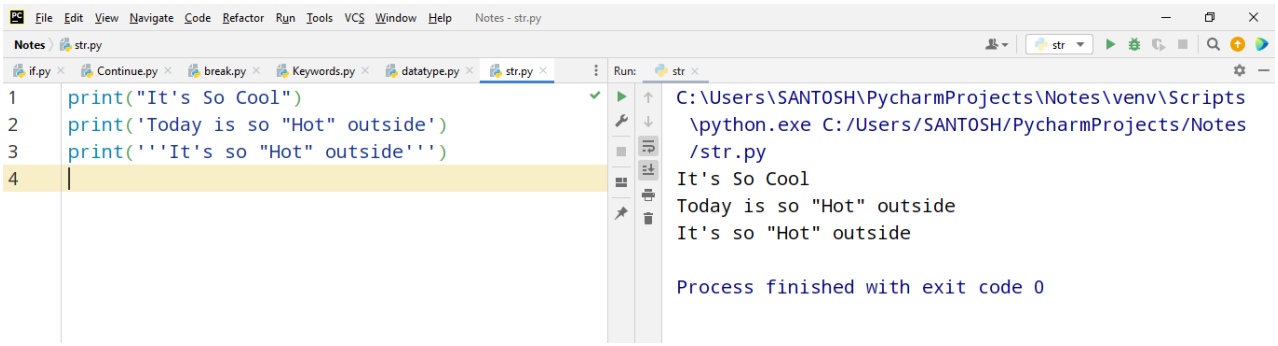
  (Example - 'Today is so "hot" outside')

- When a string contains both double quotation and a single quotation as characters in it, then enclose it with triple quotes.

  (Example - '''It's so "hot" outside''')

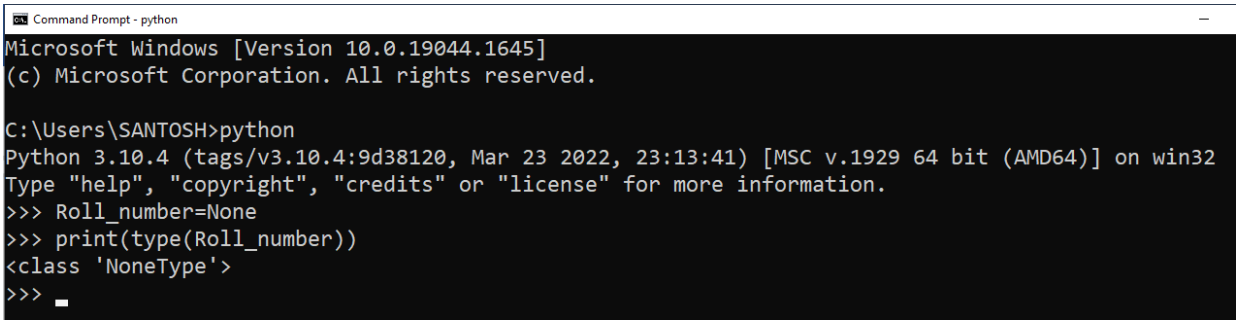
```
print("It's So Cool")
print('Today is so "Hot" outside')
print('''It's so "Hot" outside''')
```

When we run the above code, it produces the output as follows.




## 2.6.4 None

***None*** is another special data type in Python. *None* is frequently used to represent the absence of a value . In Python, **'None'** is the object which represents nothing. When we want a value to hold nothing, we do assign it with value **'None'**.



## 2.7 Indentation

- Indentation in Python is used to create a group of statements. Many popular languages such as C, C++ and Java uses braces ({ }) to define a block of code and is for readability, Python use indentation.
- The leading whitespaces (space and tabs) at the start of a line is used to determine the indentation level of the line. You have to increase the indent level to group the statements for that code block.
- Python uses indentation to indicate a block of code.

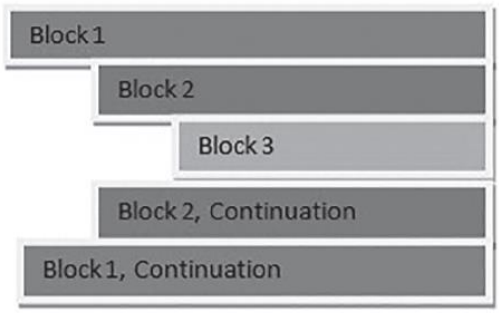



FIGURE 2.3
Code blocks and indentation in Python.

## 2.8 Statements and Expressions

A statement is an instruction that the Python interpreter can execute. Python program consists of a sequence of statements

For example, z = 1 is an assignment statement.

Expression is an arrangement of values and operators which are evaluated to make a new value. Expressions are statements as well.

For example,

```
>>> 8 + 2
10
```

## 2.9 Comments

- Comments are an important part of any program. A comment is a text that describes what the program or a particular part of the program is trying to do and is ignored by the Python interpreter.
- Comments are used to help you and other programmers understand, maintain, and debug the program.
- Python uses two types of comments:

```
single-line comment
multiline comments
```

### 2.9.1 Single Line Comment

In Python, use the hash (#) symbol to start writing a comment. Hash (#) symbol makes all text following it on the same line into a comment.

For example,

```
#This is single line Python comment
```

### 2.9.2 Multi Line Comment

If the comment extends multiple lines, then one way of commenting those lines is to use hash (#) symbol at the beginning of each line.

For example,

```
# This is
# multiline comments
# in Python
```

Another way of doing this is to use triple quotes, either `'''` or `"""`. These triple quotes are generally used for multiline strings.

However, they can be used as a multiline comment as well.

For example,

```
'''This is
   multiline comment
   in Python using triple quotes'''
```

## 2.10 Python Typecasting / Type Conversions

- In all programming languages, frequently, we need to convert the values from one data type to another data type.
- The process of converting a value from one data type to another data type is called **Typecasting** or simply **Casting.**
- In Python, the typecasting is performed using built-in functions. As all the data types in Python are organized using classes, the type casting is performed using constructor functions.

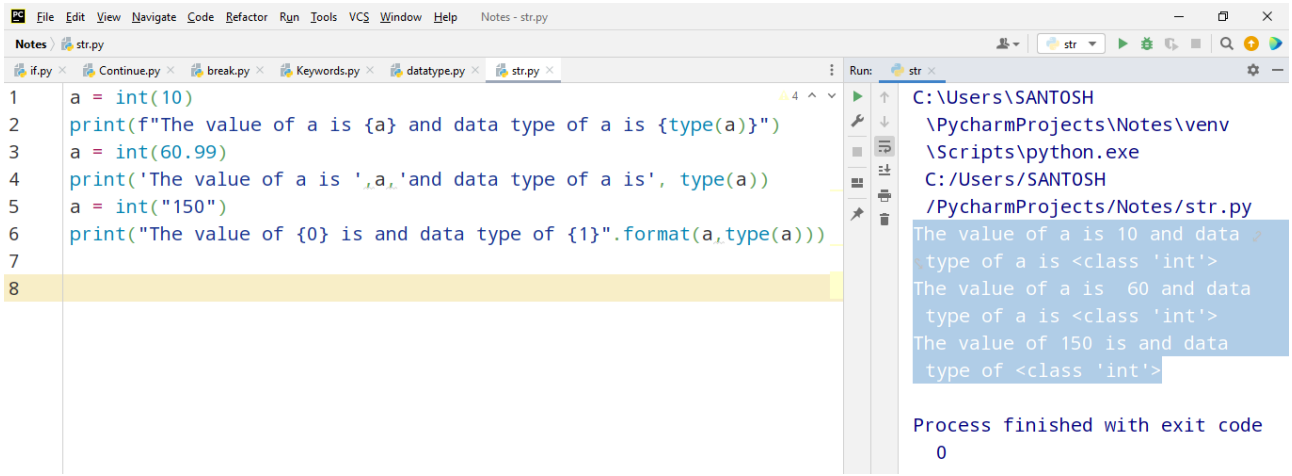
The following are the constructor functions used to perform typecasting.

| S.No. | Function | Description |
|---|---|---|
| 1 | `int( )` | It is used to convert an integer literal, float literal, and string literal (String must represent a whole number) to an integer value. |
| 2 | `float( )` | It is used to convert an integer literal, float literal, and string literal (String must represent a whole number) to a float value. |
| 3 | `str( )` | It is used to convert a value of any data type integer literals and float literals to a string value. |

## Example - int( ) Typecasting

```
a = int(10)
print(f"The value of a is {a} and data type of a is {type(a)}")
a = int(60.99)
print('The value of a is ',a,'and data type of a is', type(a))
a = int("150")
print("The value of {0} is and data type of {1}".format(a,type(a)))
```

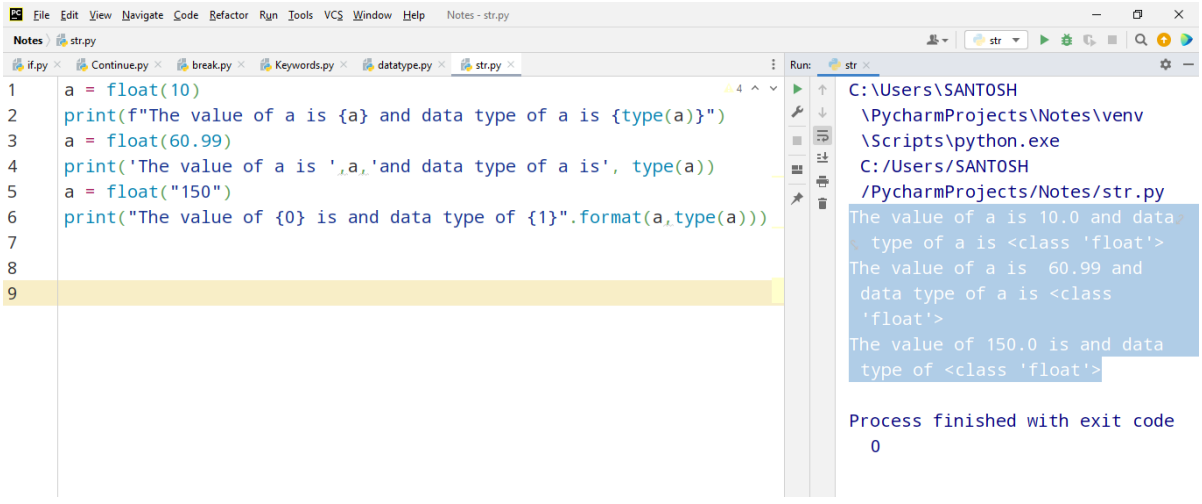
When we run the above code, it produces the output as follows




## Example - float( ) Typecasting

```
a = float(10)
print(f"The value of a is {a} and data type of a is {type(a)}")
a = float(60.99)
print('The value of a is ',a,'and data type of a is', type(a))
a = float("150")
print("The value of {0} is and data type of {1}".format(a,type(a)))
```

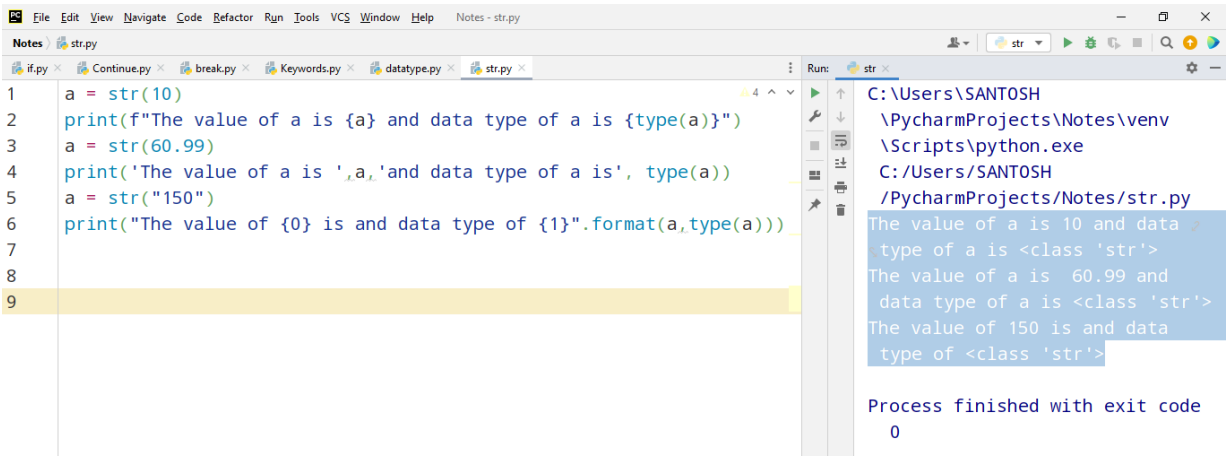
When we run the above code, it produces the output as follows



### Example - str( ) Typecasting

```
a = str(10)
print(f"The value of a is {a} and data type of a is {type(a)}")
a = str(60.99)
print('The value of a is ',a,'and data type of a is', type(a))
a = str("150")
print("The value of {0} is and data type of {1}".format(a,type(a)))
```

When we run the above code, it produces the output as follows.




**Points to be Remembered**

- In Python, when an integer value is cast to float value, then it is appended with the fractional part containing zeros (0).
- In Python, when a float value s cast to an integer it rounding down to the previous whole number.

## 2.11 Operators

In Python, an operator is a symbol used to perform arithmetical and logical operations. In other words, an operator can be defined as a symbol used to manipulate the value of an operand. Here, an operand is a value or variable on which the operator performs its task. For example, `'+'` is a symbol used to perform mathematical addition operation.

Consider the expression `a = 10 + 30`.
Here, variable `'a'`, values `'10'` and `'30'` are known as **Operands** and the symbols `'='` and `'+'` are known as **Operators.**

# Types of Operators in Python

In Python, there is a rich set of operators, and they are classified as follows.

- Arithmetic Operators ( `+, -, *, /, %, **, //` )
- Assignment Operators ( `=, +=, -=, *=, /=, %=, **=, //=` )
- Comparison Operators ( `<, <=, >, >=, ==, !=` )
- Logical Operators ( `and, or, not` )
- Identity Operators ( `is, is not` )
- Membership Operators ( `in, not in` )
- Bitwise Operators ( `&, |, ^, ~, <<, >>` )

## 2.11.1 Arithmetic Operators in Python

In Python, the arithmetic operators are the operators used to perform a basic arithmetic operation between two variables or two values.
Arithmetic operators are used to execute arithmetic operations such as addition, subtraction, division, multiplication etc.

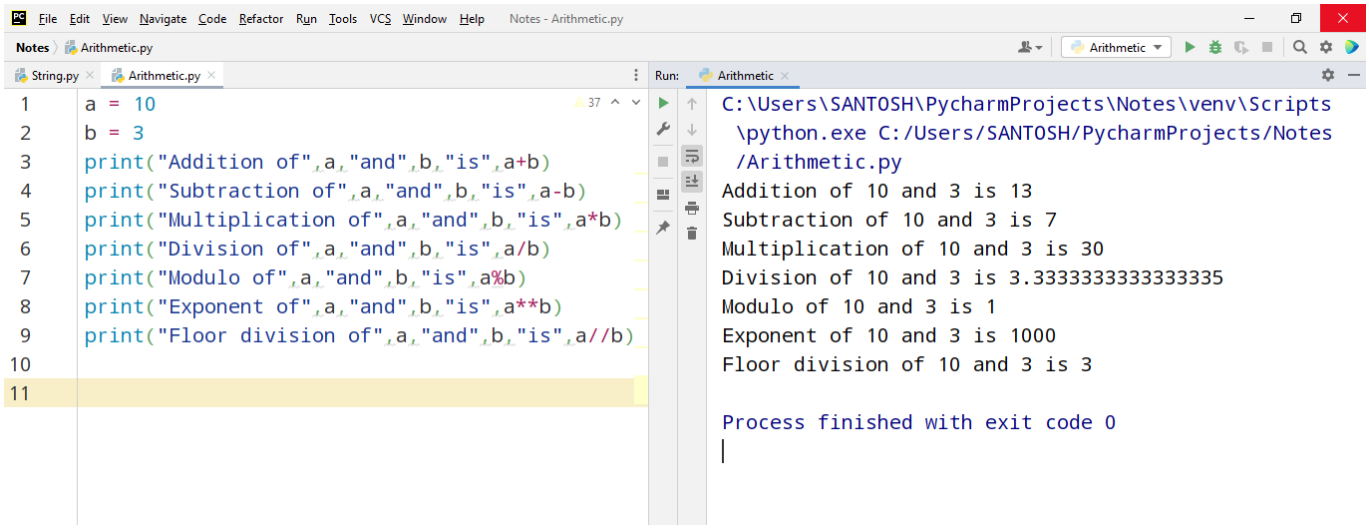
The following table presents the list of arithmetic operations in Python along with their description.

To understand the example let's consider two variables `a` with value `10` and `b` with value `3.`

| Operator | Operator Name | Description | Example |
|---|---|---|---|
| + | Addition operator | Adds two operands, producing their sum | a + b = 13 |
| - | Subtraction operator | Subtracts the two operands, producing their difference. | a - b = 7 |
| * | Multiplication operator | Produces the product of the operands | a * b = 30 |
| / | Division operator | Produces the quotient of its operands where the left operand is the dividend and the right operand is the divisor. | a / b = 3.3333333333333335 |
| % | Modulus operator | Divides left hand operand by right hand operand and returns a remainder | a % b = 1 |
| ** | Exponent operator | Performs exponential (power) calculation on operators. | a ** b = 1000 |
| // | Floor division operator | Returns the integral part of the quotient | a // b = 3 |

## Example - Arithmetic Operators in Python

```
a = 10
b = 3
print("Addition of",a,"and",b,"is",a+b)
print("Subtraction of",a,"and",b,"is",a-b)
print("Multiplication of",a,"and",b,"is",a*b)
print("Division of",a,"and",b,"is",a/b)
print("Modulo of",a,"and",b,"is",a%b)
print("Exponent of",a,"and",b,"is",a**b)
print("Floor division of",a,"and",b,"is",a//b)
```




## 2.11.2 Assignment Operators in Python

In Python, the assignment operators are the operators used to assign the right-hand side value to the left-hand side variable. The following table presents the list of assignment operations in Python along with their description.

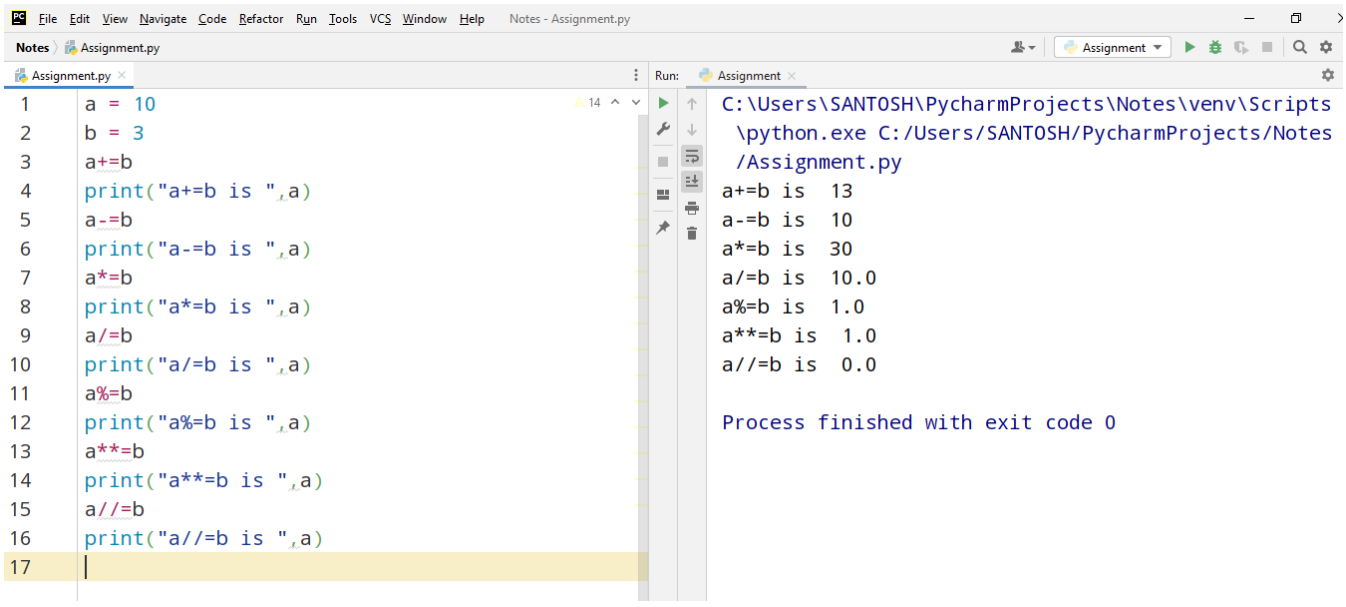
To understand the example let's consider two variables `a` with value `10` and `b` with value `3.`

| Operator | Operator Name | Description | Example |
|---|---|---|---|
| = | Assignment | Assigns right-hand side value to left-hand side variable | a = 10 |
| += | Addition Assignment | Adds right operand to the left operand and assign the result to left operand | a += b ( a = a + b ) |
| -= | Subtraction Assignment | Subtracts right operand from the left operand and assign the result to left operand | a -= b ( a = a - b ) |
| *= | Multiplication Assignment | Multiplies right operand with the left operand and assign the result to left operand | a *= b ( a = a * b ) |
| /= | Division Assignment | Divides left operand with the right operand and assign the result to left operand | a /= b ( a = a / b ) |

| | | | |
|---|---|---|---|
| %= | Remainder Assignment | Takes modulus using two operands and assign the result to left operand | a %= b ( a = a % b ) |
| **= | Exponentiation Assignment | Performs exponential calculation on operators and assigns value to the left operand | a **= b ( a = a ** b ) |
| //= | Floor Division Assignment | Performs floor division on operators and assign value to the left operand | a //= b ( a = a // b ) |

## Example - Assignment Operators in Python

```
a = 10
b = 3
a+=b
print("a+=b is ",a)
a-=b
print("a-=b is ",a)
a*=b
print("a*=b is ",a)
a/=b
print("a/=b is ",a)
a%=b
print("a%=b is ",a)
a**=b
print("a**=b is ",a)
a//=b
print("a//=b is ",a)
```



## 2.11. 3 Comparison Operators in Python

- In Python, the comparison operators are used to compare two values. In other words, comparison operators are used to checking the relationship between two variables or values.
- The comparison operators are also known as **Relational Operators**.

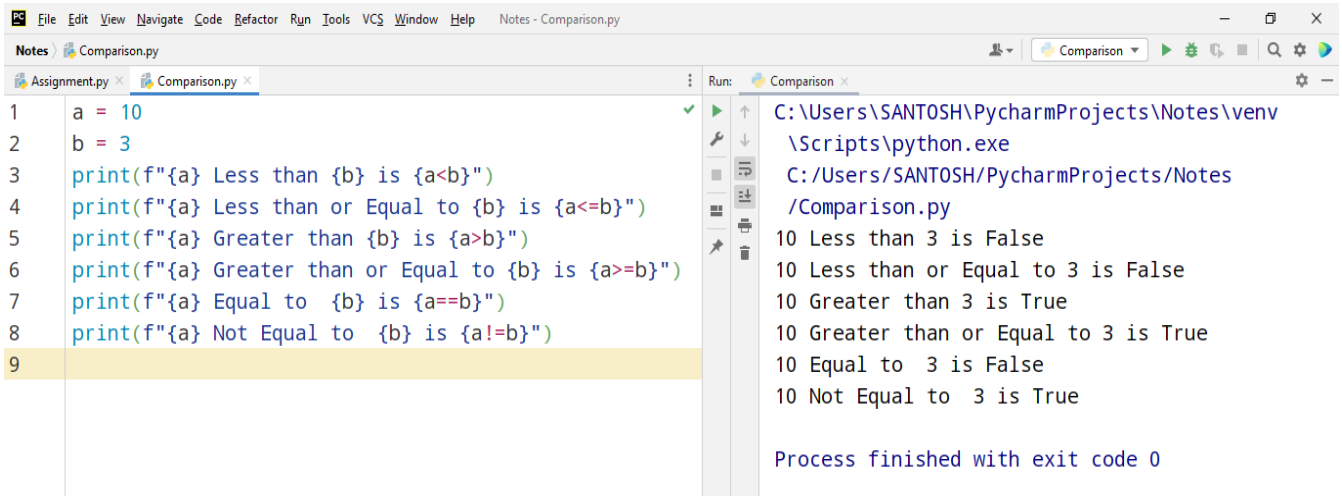
The following table presents the list of comparison operations in Python along with their description.

To understand the example let's consider two variables **a** with value **10** and **b** with value **3**.

| Operator | Operator Name | Description | Example |
|---|---|---|---|
| < | Less than | Returns **True** if left-hand side value is smaller than right-hand side value, otherwise returns **False.** | a < b (False) |
| <= | Less than or Equal to | Returns **True** if left-hand side value is smaller than or equal to right-hand side value, otherwise returns **False.** | a <= b (False) |
| > | Greater than | Returns **True** if left-hand side value is larger than right-hand side value, otherwise returns **False.** | a > b (True) |
| >= | Greater than or Equal to | Returns **True** if left-hand side value is larger than or equal to right-hand side value, otherwise returns **False.** | a >= b (True) |
| == | Equal to | Returns **True** if left-hand side value is equal to right-hand side value, otherwise returns **False**. | a == b (False) |
| != | Not Equal to | Returns **True** if left-hand side value is not equal to right-hand side value, otherwise returns **False.** | a != b (True) |

### Example - Comparison Operators in Python

```
a = 10
b = 3
print(f"{a} Less than {b} is {a<b}")
print(f"{a} Less than or Equal to {b} is {a<=b}")
print(f"{a} Greater than {b} is {a>b}")
print(f"{a} Greater than or Equal to {b} is {a>=b}")
print(f"{a} Equal to  {b} is {a==b}")
print(f"{a} Not Equal to  {b} is {a!=b}")
```




## 2.11.4 Logical Operators

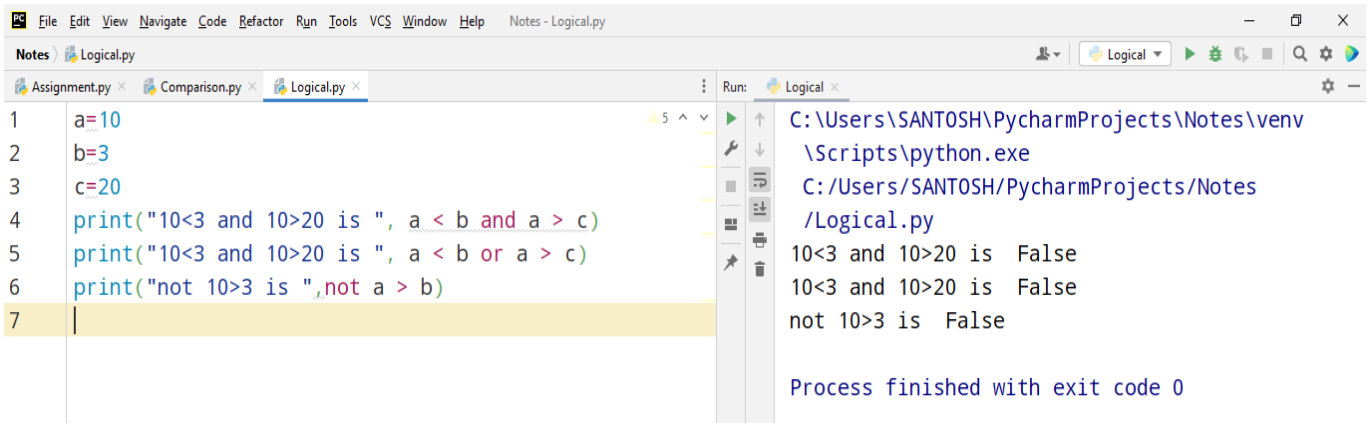
In Python, the logical operators are used to merge multiple conditions into a single condition. The following table presents the list of logical operations in Python along with their description.

To understand the example let's consider two variables **a** with value **10** and **b** with value **3.**

| Operator | Operator Name | Description | Example |
|---|---|---|---|
| and | Logical AND | Returns **True** if all the conditions are True, | a < b and a > c |
| or | Logical OR | Returns **False** if all the conditions are | a < b or a > c |
| not | Logical NOT | Returns **True** if the condition is False, | not a > b |

### Example - Logical Operators in Python

```
a=10
b=3
c=20
print("10<3 and 10>20 is ", a < b and a > c)
print("10<3 and 10>20 is ", a < b or a > c)
print("not 10>3 is ",not a > b)
```



## 2.11.5 Identity Operators

In Python, identity operators are used to comparing the memory locations of two objects or variables.

There are two identity operators in Python. The following table presents the list of identity operations in Python along with their description.

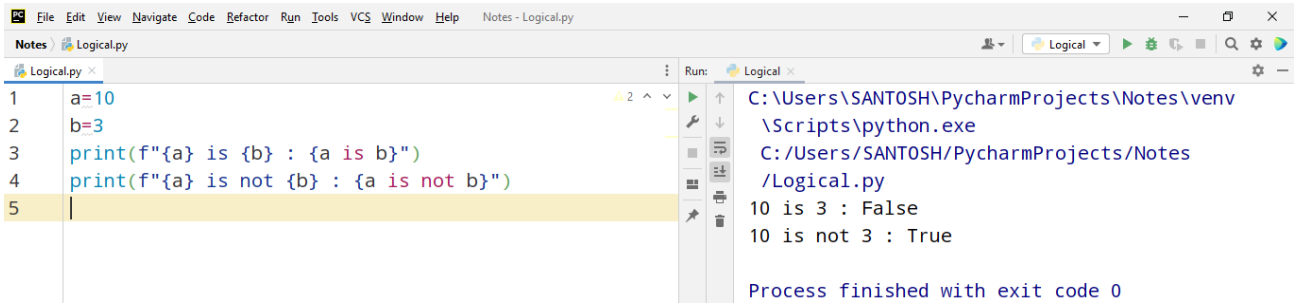
To understand the example let's consider two variables `a` with value `10` and `b` with value `3.`

| Operator | Operator Name | Description | Example |
|---|---|---|---|
| is | is identical | Returns `True` if the variables on either side of the operator point to the same object otherwise returns `False.` | `a is b` (False) |
| is not | is not identical | Returns `False` if the variables on either side of the operator point to the same object otherwise returns `True`. | `a is not b` (True) |

Example - Identity Operators in Python

```
a=10
b=3
print(f"{a} is {b} : {a is b}")
print(f"{a} is not {b} : {a is not b}")
```

When we run the above code, it produces the output as follows.




## 2.11.6 Membership Operators

In Python, the membership operators are used to test whether a value is present in a sequence. Here the sequence may be String, List, or Tuple.

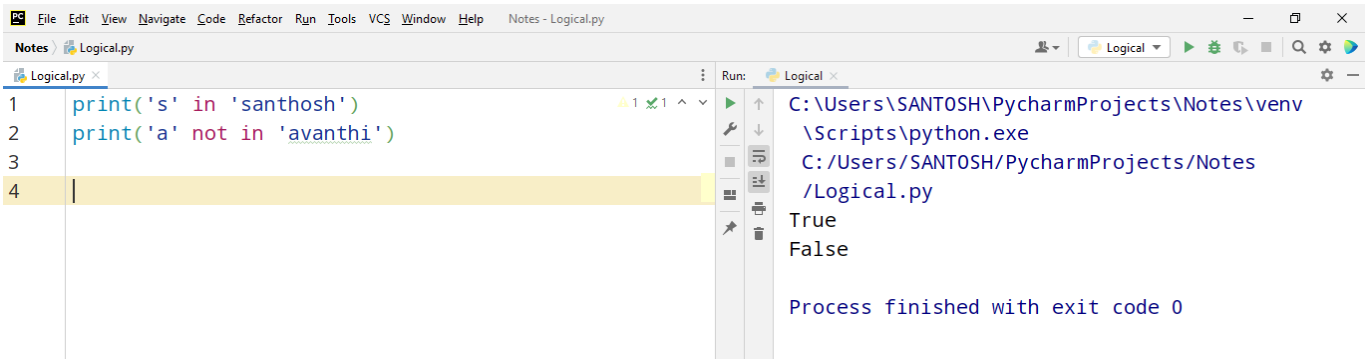
There are two membership operators in Python. The following table presents the list of membership operations in Python along with their description.

| Operator | Operator Name | Description | Example |
|---|---|---|---|
| in | in | Returns `True` if it finds a variable in the specified sequence, otherwise returns `False`. | a in list |
| not in | not in | Returns `True` if it does not finds a variable in the specified sequence, otherwise returns `False.` | a not in list |

### Example - Membership Operators in Python

```
print('s' in 'santhosh')
print('a' not in 'avanthi')
```

When we run the above code, it produces the output as follows.




## 2.11.7 Bitwise Operators in Python

In Python, the bitwise operators are used to performs bit by bit operation. There are six bitwise operators in Python.
The following table presents the list of bitwise operations in Python along with their description.

| Operator | Operator Name | Description | Example |
|---|---|---|---|
| & | Binary AND | Result is one in each bit position for which the corresponding bits of both operands are 1s. | p & q = 12 (means 0000 1100) |
| \| | Binary OR | Result is one in each bit position for which the corresponding bits of either or both operands are 1s. | p \| q = 61 (means 0011 1101) |
| ^ | Binary XOR | Result is one in each bit position for which the corresponding bits of either but not both operands are 1s. | (p ^ q) = 49 (means 0011 0001) |
| ~ | Binary Ones Complement | Inverts the bits of its operand. | (~p) = –61 (means 1100 0011 in 2's complement form due to a signed binary number. |
| << | Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. | p << 2 = 240 (means 1111 0000) |
| >> | Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | p >> 2 = 15 (means 0000 1111) |

*Note:* The value of p is 60 and q is 13.

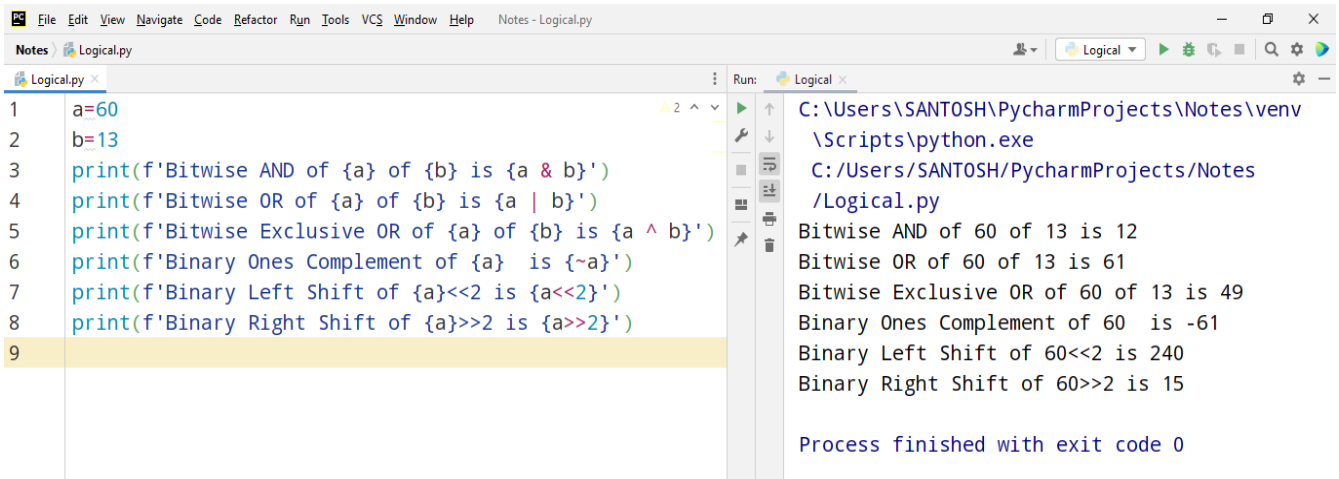
Bitwise Truth Table

| P | Q | P & Q | P \| Q | P ^ Q | ~ P |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | |

| Bitwise and ( & ) | | | | Bitwise or ( \| ) | | | |
|---|---|---|---|---|---|---|---|
| a | = | 0011 1100 | → (60) | a | = | 0011 1100 | → (60) |
| b | = | 0000 1101 | → (13) | b | = | 0000 1101 | → (13) |
| a & b | = | 0000 1100 | → (12) | a \| b | = | 0011 1101 | → (61) |
| **Bitwise exclusive or ( ^ )** | | | | **One's Complement ( ~ )** | | | |
| a | = | 0011 1100 | → (60) | a | = | 0011 1100 | → (60) |
| b | = | 0000 1101 | → (13) | ~ a | = | 1100 0011 | → (-61) |
| a ^ b | = | 0011 0001 | → (49) | | | | |
| **Binary left shift ( << )** | | | | **Binary right shift ( >> )** | | | |
| a | = | 0011 1100 | → (60) | a | = | 0011 1100 | → (60) |
| a << 2 | = | 1111 0000 | → (240) | a >> 2 | = | 0000 1111 | → (15) |
| left shift of 2 bits | | | | right shift of 2 bits | | | |

## Example - bitwise Operators in Python

```
a=60
b=13
print(f'Bitwise AND of {a} of {b} is {a & b}')
print(f'Bitwise OR of {a} of {b} is {a | b}')
print(f'Bitwise Exclusive OR of {a} of {b} is {a ^ b}')
print(f'Binary Ones Complement of {a}  is {~a}')
print(f'Binary Left Shift of {a}<<2 is {a<<2}')
print(f'Binary Right Shift of {a}>>2 is {a>>2}')
```



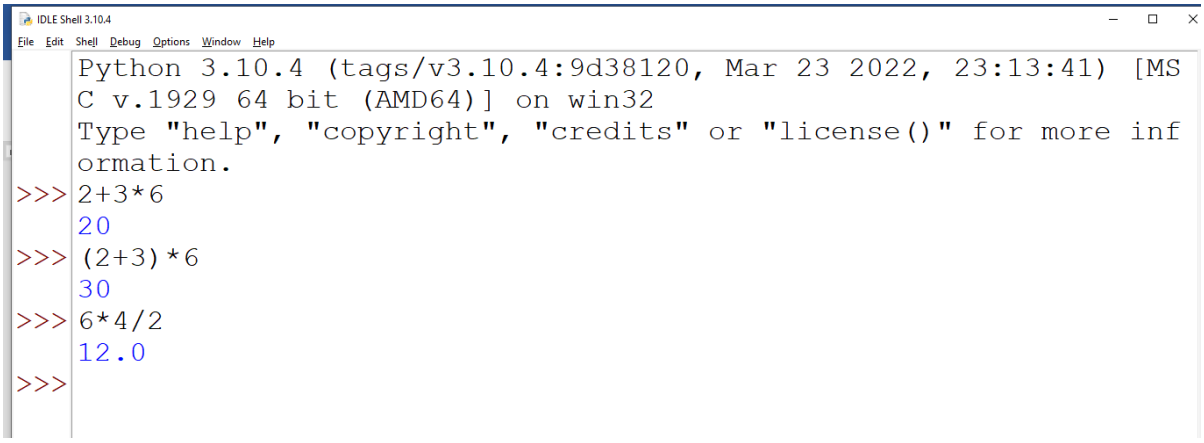
## 2.12 Precedence and Associativity

- Operator precedence determines the way in which operators are parsed with respect to each other.
- Operators with higher precedence become the operands of operators with lower precedence. Associativity determines the way in which operators of the same precedence are parsed. Almost all the operators have left-to-right associativity.
- Operator precedence is listed in TABLE 2.9 starting with the highest precedence to lowest precedence

**TABLE 2.9**

Operator Precedence in Python

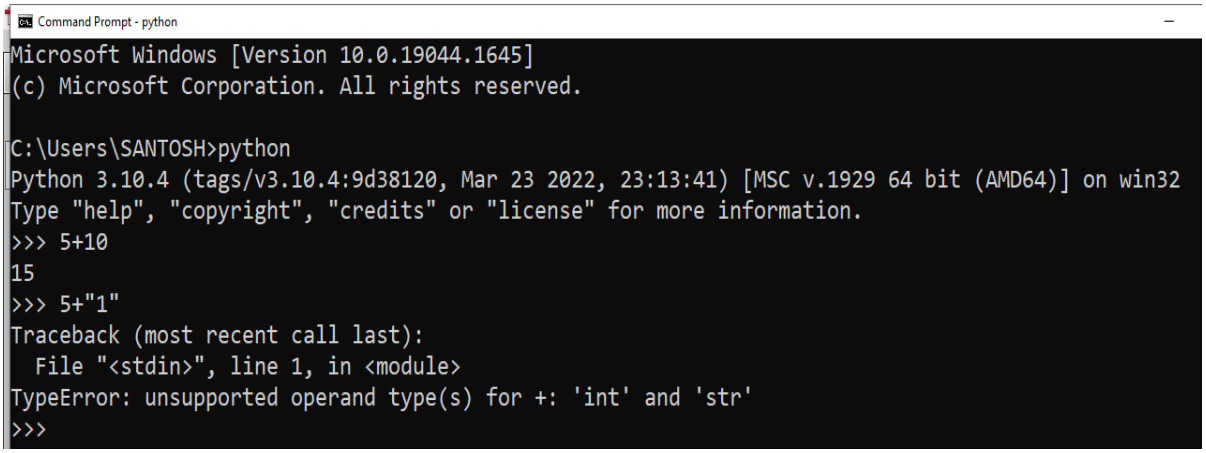
| Operators | Meaning |
|---|---|
| () | Parentheses |
| ** | Exponent |
| +x, –x, ~x | Unary plus, Unary minus, Bitwise NOT |
| *, /, //, % | Multiplication, Division, Floor division, Modulus |
| +, – | Addition, Subtraction |
| <<, >> | Bitwise shift operators |
| & | Bitwise AND |
| ^ | Bitwise XOR |
| \| | Bitwise OR |
| ==, !=, >, >=, <, <=, is, is not, in, not in | Comparisons, Identity, Membership operators |
| not | Logical NOT |
| and | Logical AND |
| or | Logical OR |



## 2.13 Dynamic and Strongly Typed Language

- Python is a dynamic language as the type of the variable is determined during run-time by the interpreter.
- Python is also a strongly typed language as the interpreter keeps track of all the variables types. In a strongly typed language, you are simply not allowed to do anything that's incompatible with the type of data you are working with.

For Example.




In Python, Traceback is printed when an error occurs. The last line tells us the kind of error that occurred which in our case is the unsupported operand type(s).

# Chapter-3
# Control Flow Statements

In Python, the default execution flow of a program is a sequential order. But the sequential order of execution flow may not be suitable for all situations.

Sometimes, we may want to jump from line to another line, we may want to skip a part of the program, or sometimes we may want to execute a part of the program again and again.

To solve this problem, Python provides control statements.

In Python, the control statements are the statements which will tell us that in which order the instructions are getting executed.
The control statements are used to control the order of execution according to our requirements.
Python provides several control statements, and they are classified as follows.

## Sequential Control Flow Statements:

This refers to the line by line execution, in which the statements are executed sequentially, in the same order in which they appear in the program.

## Decision Control Flow Statements / Selection Control Statements

- In Python, the selection statements are also known as decision making statements or branching statements.
- The selection statements are used to select a part of the program to be executed based on a condition.
- Python provides the following selection statements.
    - `if statement`
    - `if…else statement`
    - `if…elif…else statement`
    - `nested if statement`

## Loop Control Flow Statements:

- The iterative statements are also known as looping statements or repetitive statements.
- The iterative statements are used to execute a part of the program repeatedly as long as the given condition is True.
- Using iterative statements reduces the size of the code, reduces the code complexity, makes it more efficient, and increases the execution speed.
- Python provides the following iterative statements.
    - `while statement`
    - `for statement`

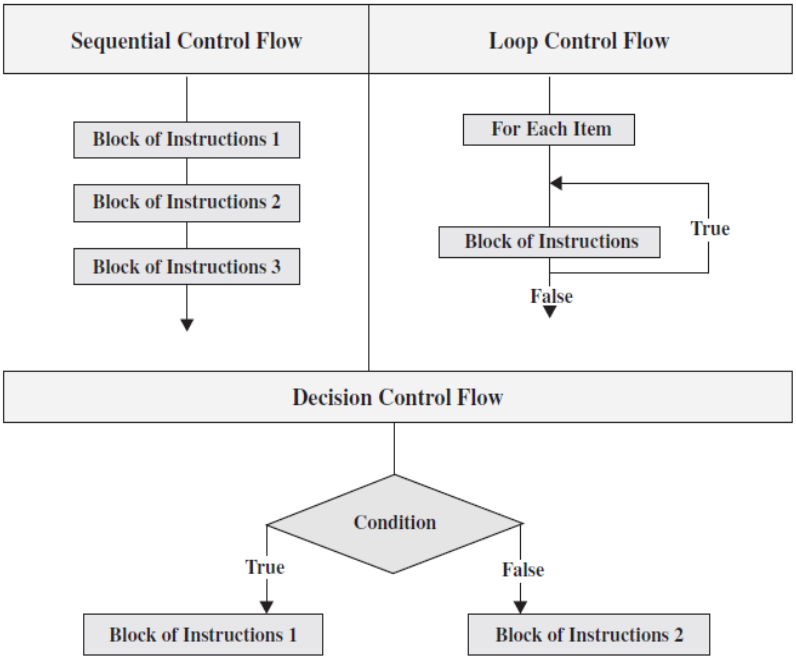
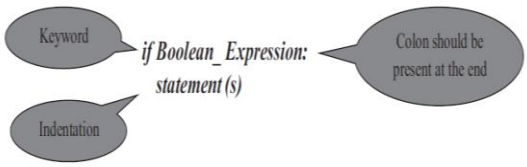



FIGURE 3.1
Forms of control flow statements.

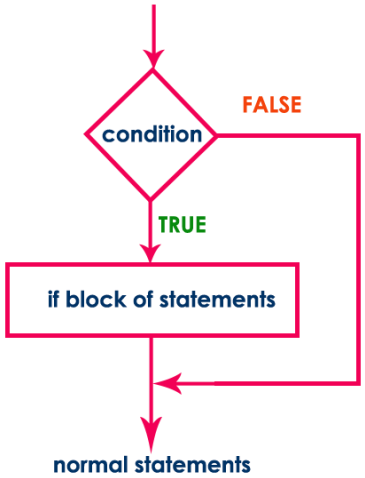
## 3.1.1 if Decision Control Flow statement

If the Boolean expression (Condition) evaluates to *True* then statements in the *if* block will be executed; otherwise the result is *False* then none of the statements are executed.

The syntax for ***if*** statement is,




The execution flow of if statement is as follows.




**Syntax**

```
if condition:
     Statement_1
     Statement_2
     Statement_3
      ...
```

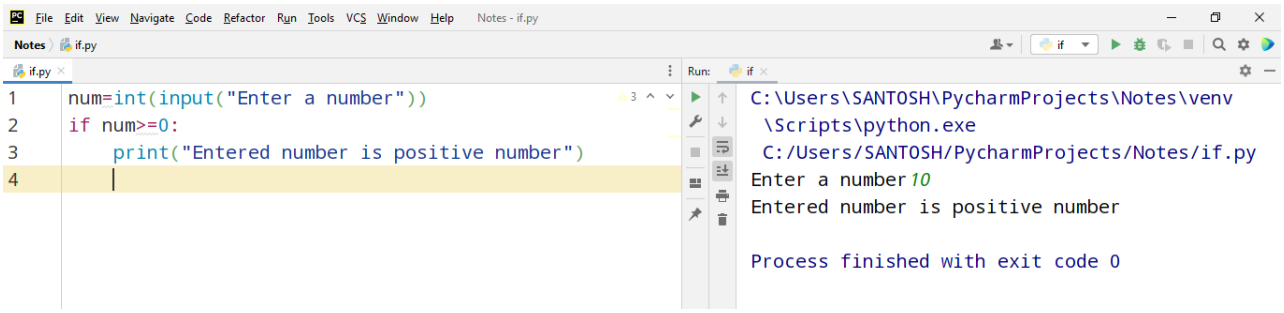
- The ***if*** decision control flow statement starts with ***if*** keyword and ends with a colon.
- The expression in an ***if*** statement should be a Boolean expression.
- In Python, the if block statements are determined through indentation.

```
Command Prompt - python
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\SANTOSH>python
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> if 20>10:
...    print("20 is greater than 10")
...
20 is greater than 10
>>>
```

## Example 1: Program Reads a Number and Prints a Message If It Is Positive

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

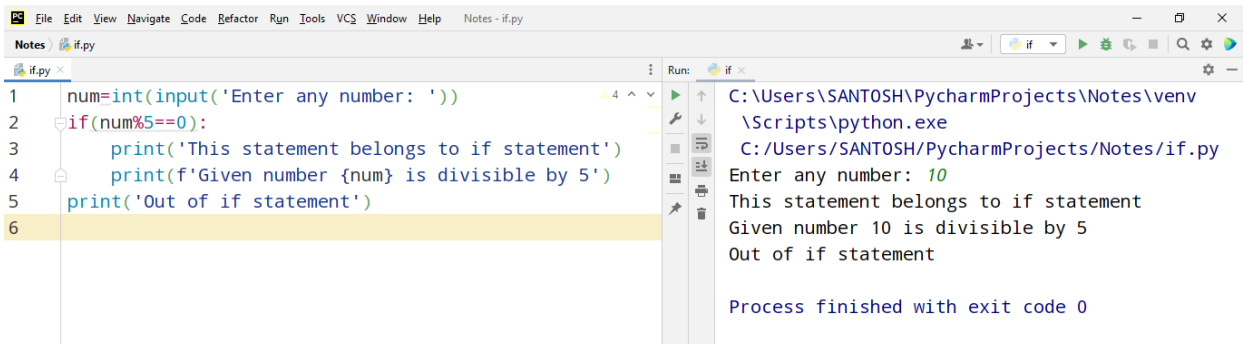
When we run the above code, it produces the output as follows.




## Example 2: Program to Check Whether a Number is Divisible by 5

```
num=int(input('Enter any number: '))
if(num%5==0):
    print('This statement belongs to if statement')
    print(f'Given number {num} is divisible by 5')
print('Out of if statement')
```

When we run the above code, it produces the output as follows



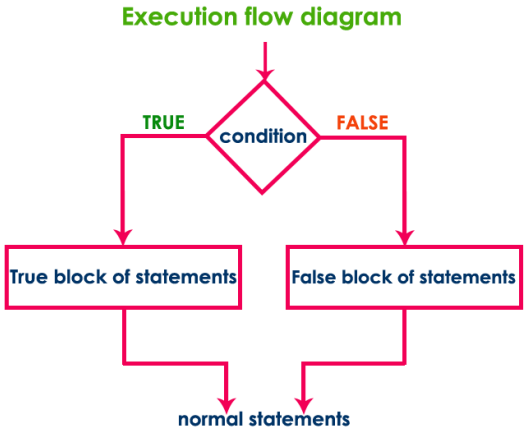
```
num=int(input('Enter any number: '))
if(num%5==0):
    print('This statement belongs to if statement')
    print(f'Given number {num} is divisible by 5')
print('Out of if statement')
```

```
C:\Users\SANTOSH\PycharmProjects\Notes\venv
 \Scripts\python.exe
 C:/Users/SANTOSH/PycharmProjects/Notes/if.py
Enter any number: 12
Out of if statement

Process finished with exit code 0
```

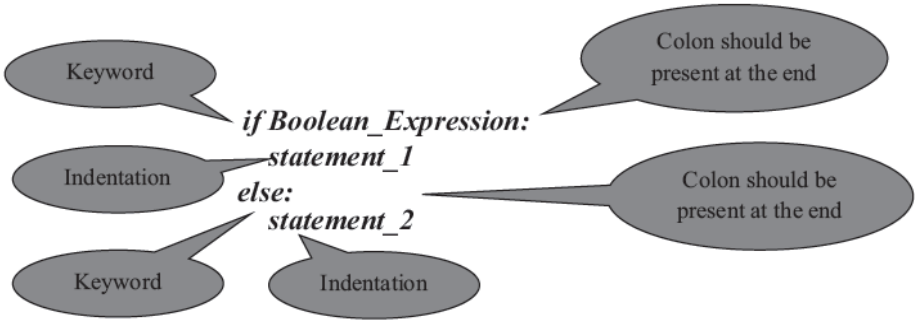
In the above execution, the number 12 is not divisible by 5. So, the condition becomes False and the condition is evaluated to False. Then the if statement ignores the execution of its block of statements.

## 3.1.2 if..else Decision Control Flow statement

- In Python, we use the if-else statement to test a condition and pick the execution of a block of statements out of two blocks based on that condition result.
- The if-else statement checks the given condition then decides which block of statements to be executed based on the condition result.
- If the condition is True, then the true block of statements is executed and if it is False, then the false block of statements is executed.
- The execution flow of if-else statement is as follows.




The syntax for *if...else* statement is,



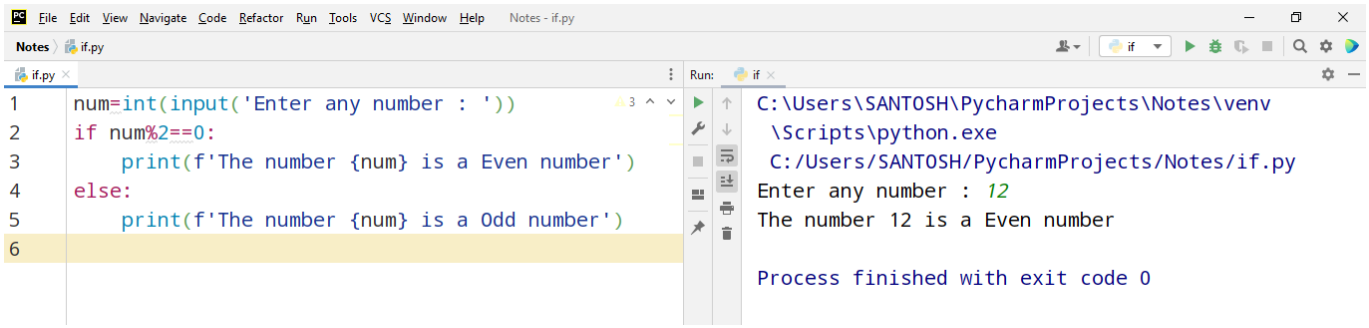
### Syntax

```
if condition:
        Statement_1
        Statement_2
        Statement_3
        ...
else:
        Statement_4
        Statement_5
        ...
```

**Example 1: Program to Find If a Given Number Is Odd or Even**

```
num=int(input('Enter any number : '))
if num%2==0:
    print(f'The number {num} is a Even number')
else:
    print(f'The number {num} is a Odd number')
```

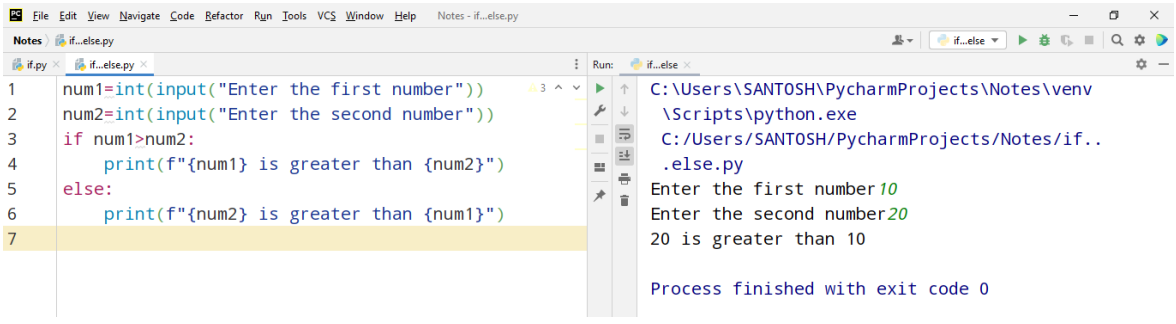
When we run the above code, it produces the output as follows.




**Example 2: Program to Find the Greater of Two Numbers**

```
num1=int(input("Enter the first number"))
num2=int(input("Enter the second number"))
if num1>num2:
    print(f"{num1} is greater than {num2}")
else:
    print(f"{num2} is greater than {num1}")
```

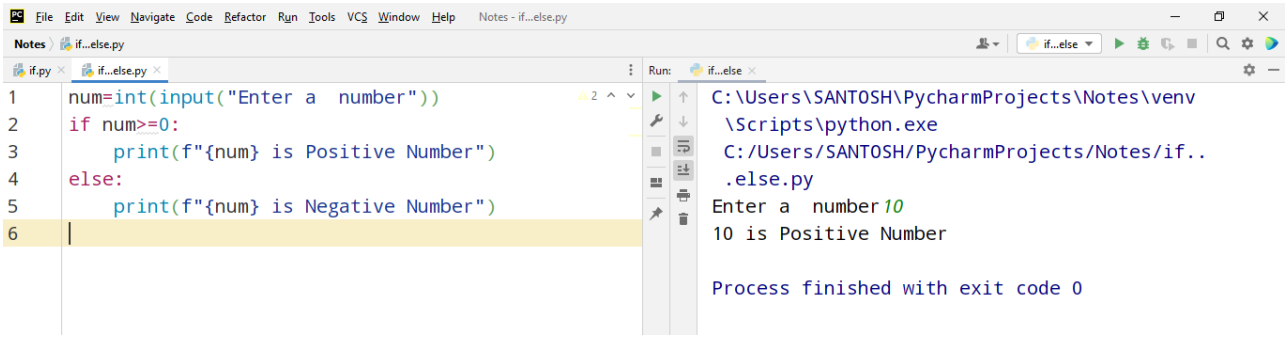
When we run the above code, it produces the output as follows.



Example 3: **Program Reads a Number and Checks if the number is positive or negative**

```
num=int(input("Enter a  number"))
if num>=0:
    print(f"{num} is Positive Number")
else:
    print(f"{num} is Negative Number")
```

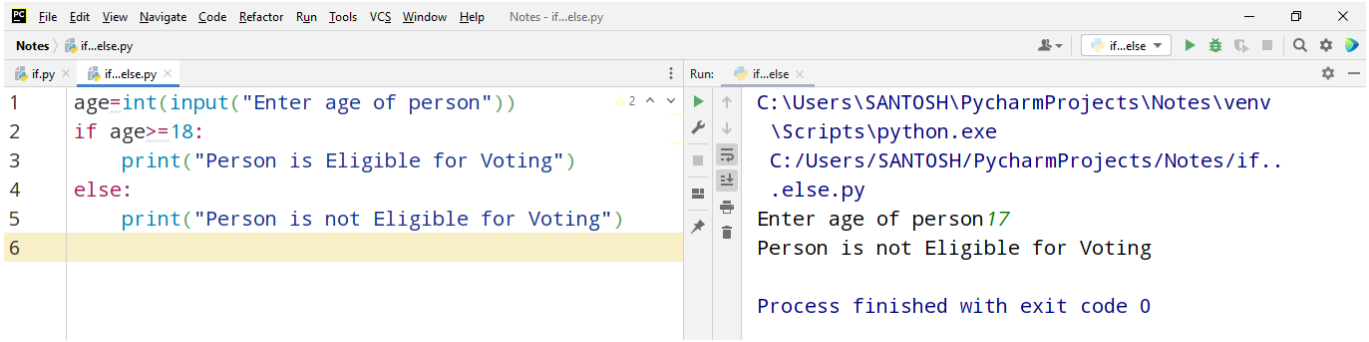
When we run the above code, it produces the output as follows.




Example 4: **Program Reads a Person age and check whether the person is eligible to vote or not**

```
age=int(input("Enter age of person"))
if age>=18:
    print("Person is Eligible for Voting")
else:
    print("Person is not Eligible for Voting")
```

When we run the above code, it produces the output as follows



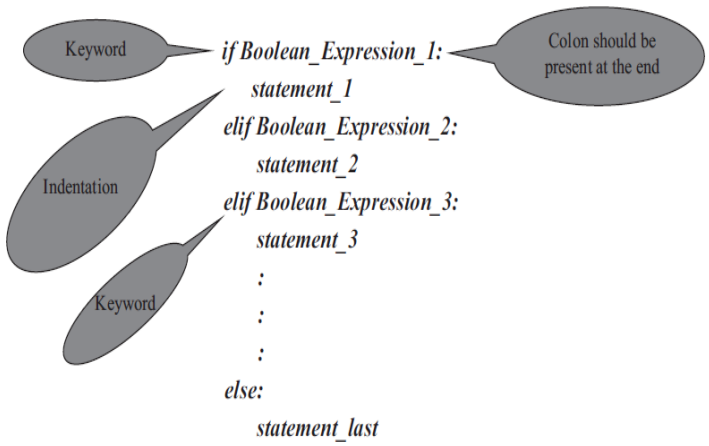

### 3.1.3 The *if…elif…else* Decision Control Statement

In Python, When we want to test multiple conditions we use `elif` statement. The keyword `'elif'` is short for 'else if'.

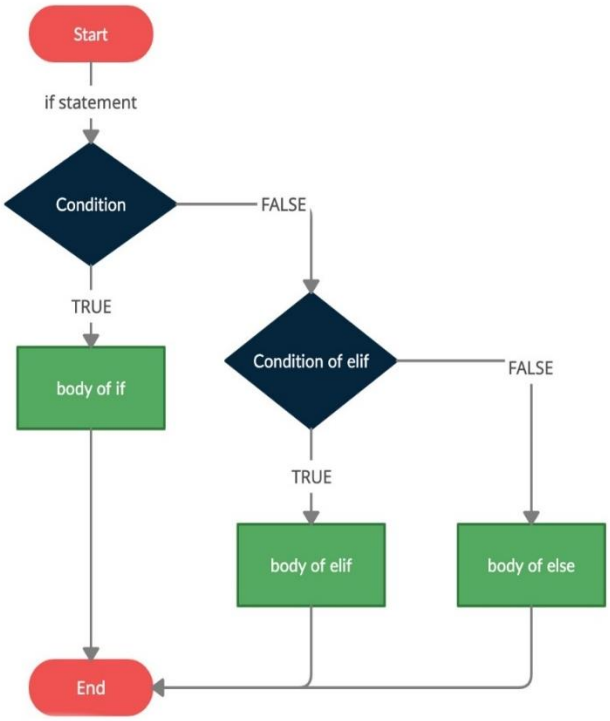
The ***if…elif…else*** is also called as multi-way decision control statement. When you need to choose from several possible alternatives, then an *elif* statement is used along with an *if* statement.

The *else* statement must always come last, and will again act as the default action.

The syntax for *if...elif...else* statement is,

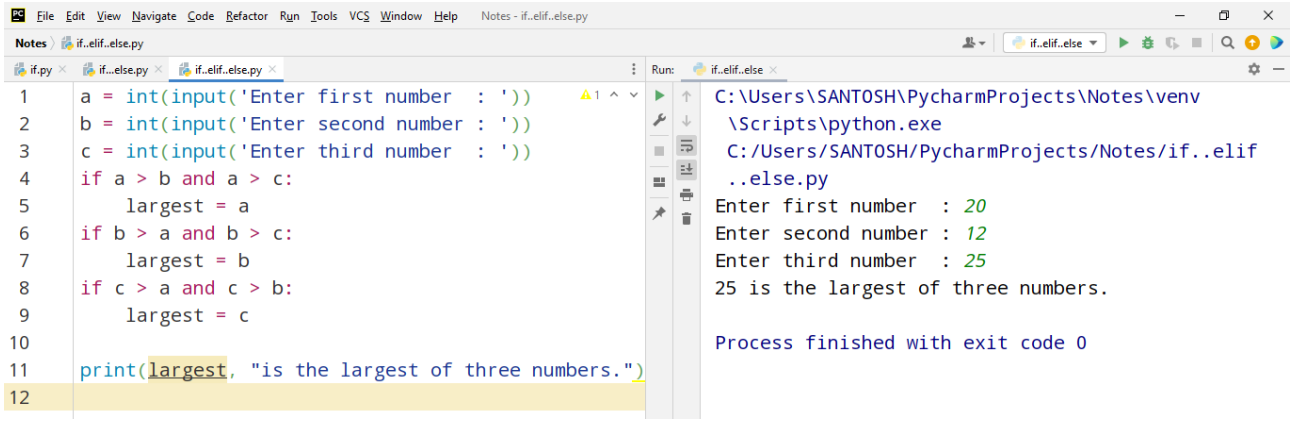



```
if condition_1:
        Statement_1
        Statement_2
        Statement_3
        ...
elif condition_2:
        Statement_4
        Statement_5
        Statement_6
        ...
else:
       Statement_7
       Statement_8

        ...
```




Example 1: **Program to find the largest number among the three input numbers**

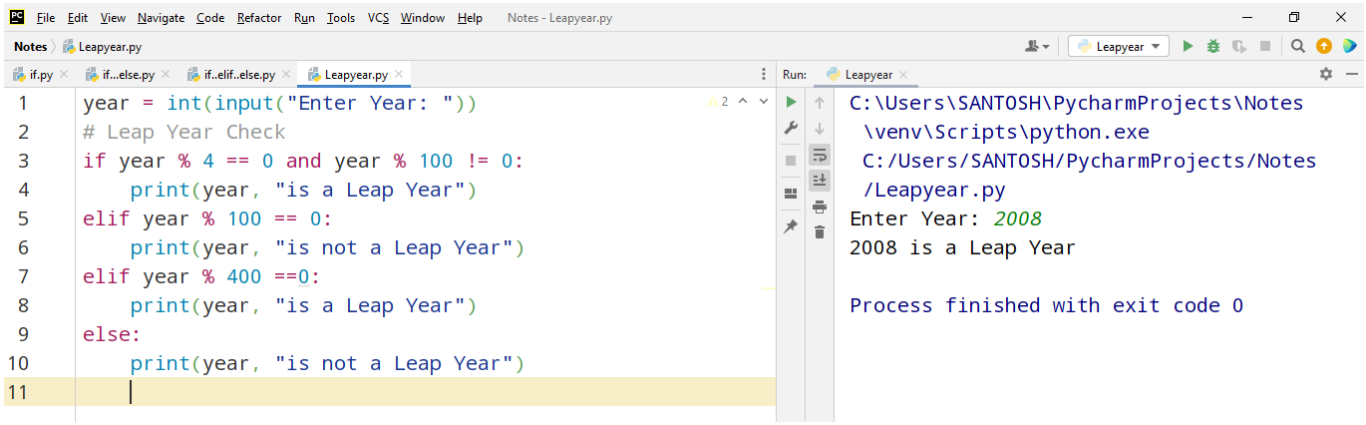
```
a = int(input('Enter first number  : '))
b = int(input('Enter second number : '))
c = int(input('Enter third number  : '))
if a > b and a > c:
    largest = a
if b > a and b > c:
    largest = b
if c > a and c > b:
    largest = c

print(largest, "is the largest of three numbers.")
```




Example 2: **Program to Check whether the entered year is leap year or not.**

```
year = int(input("Enter Year: "))
# Leap Year Check
if year % 4 == 0 and year % 100 != 0:
    print(year, "is a Leap Year")
elif year % 100 == 0:
    print(year, "is not a Leap Year")
elif year % 400 ==0:
    print(year, "is a Leap Year")
else:
    print(year, "is not a Leap Year")
```

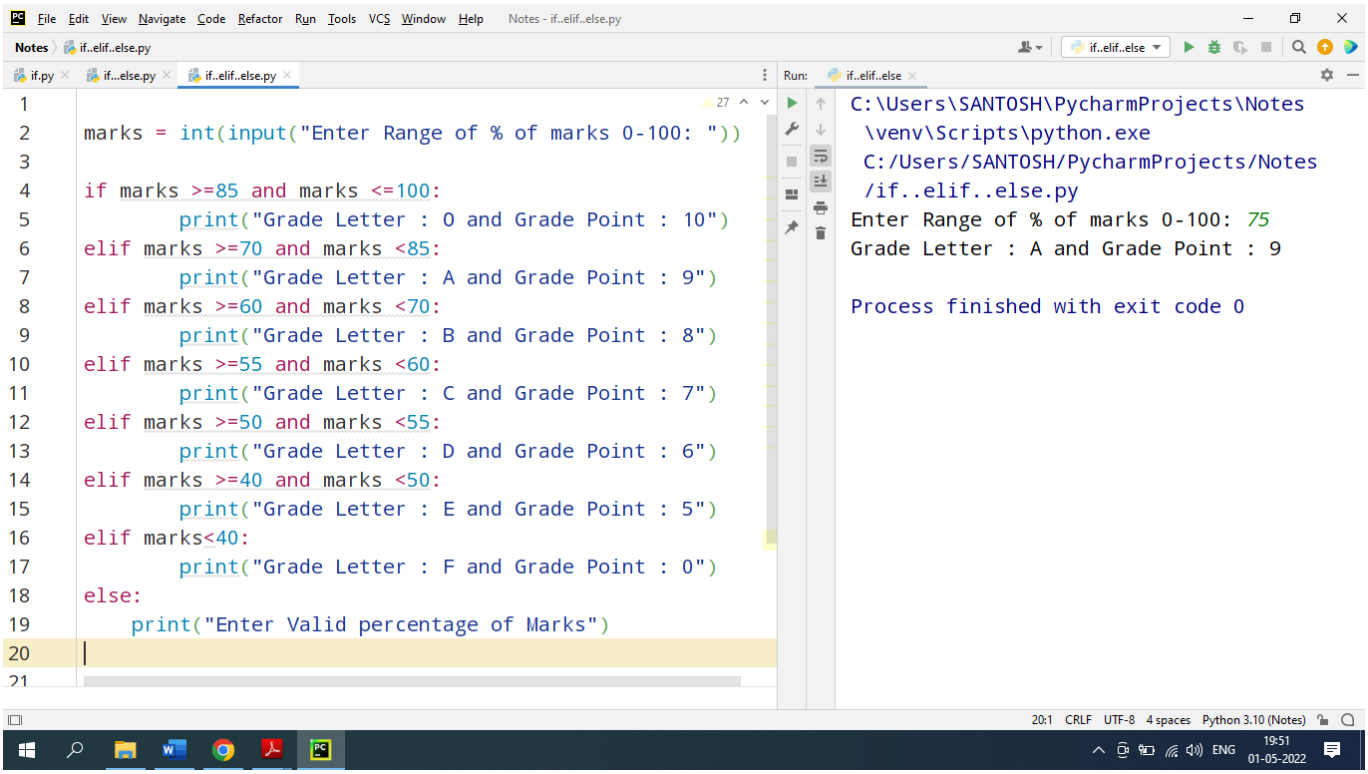
When we run the above code, it produces the output as follows



Example 3: **Program to Print the *Grade Letter and Grade Point* based on Student Marks percentage and Read marks by the user.**

```
marks = int(input("Enter Range of % of marks 0-100: "))

if marks >=85 and marks <=100:
        print("Grade Letter : O and Grade Point : 10")
elif marks >=70 and marks <85:
        print("Grade Letter : A and Grade Point : 9")
elif marks >=60 and marks <70:
        print("Grade Letter : B and Grade Point : 8")
elif marks >=55 and marks <60:
        print("Grade Letter : C and Grade Point : 7")
elif marks >=50 and marks <55:
        print("Grade Letter : D and Grade Point : 6")
elif marks >=40 and marks <50:
        print("Grade Letter : E and Grade Point : 5")
elif marks<40:
        print("Grade Letter : F and Grade Point : 0")
else:
    print("Enter Valid percentage of Marks")
```

When we run the above code, it produces the output as follows



### *3.1.4 Nested if Statement*

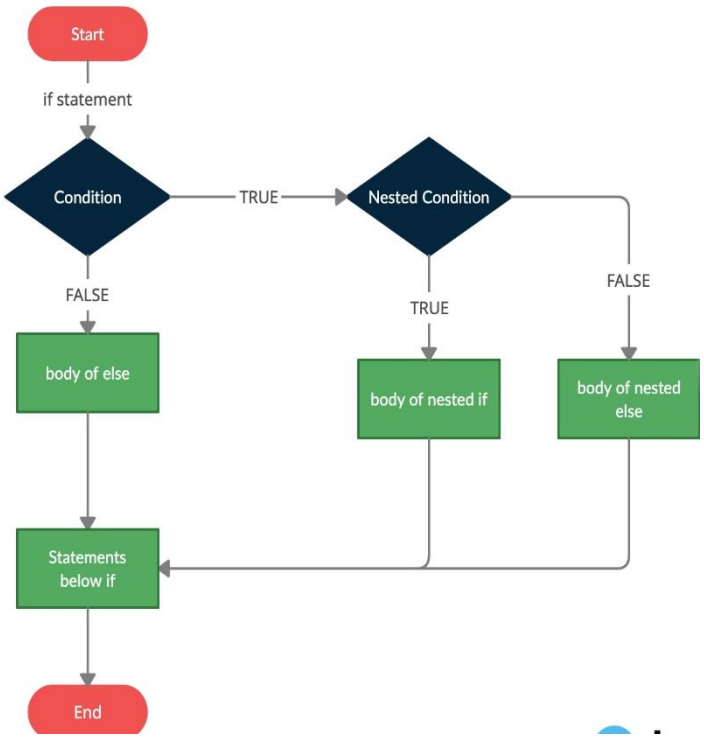
In some situations, you have to place an *if* statement inside another statement. An *if* statement that contains another *if* statement either in its *if* block or *else* block is called a Nested *if* statement.

The syntax of the nested *if* statement is,

Nested if

```
if Boolean_Expression_1:
    if Boolean_Expression_2:
        statement_1
    else:
        statement_2
else:
    statement_3
```

If the *Boolean_Expression_1* is evaluated to *True*, then the control shifts to *Boolean_Expression_2* and if the expression is evaluated to *True*, then statement_1 is executed,
If the *Boolean_Expression_2* is evaluated to *False* then the statement_2 is executed.
If the *Boolean_Expression_1* is evaluated to *False,* then statement_3 is executed.

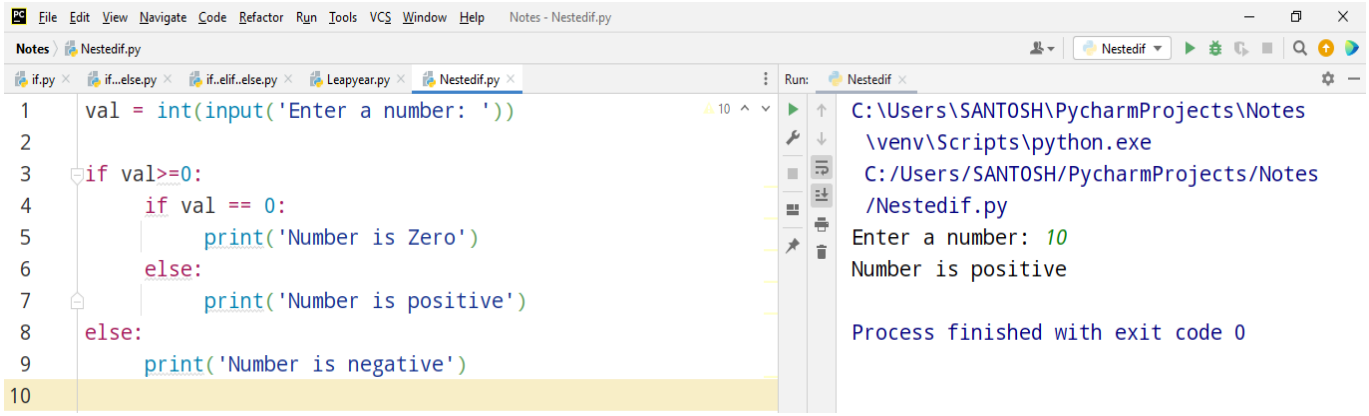



```
if condition_1:
    if condition_2:
        Statement_1
        Statement_2
        Statement_3
        ...
    else:
        Statement_4
        Statement_5
        Statement_6
        ...
else:
       Statement_7
       Statement_8
```

Example 1: **Program to Check sign of a user entered number**

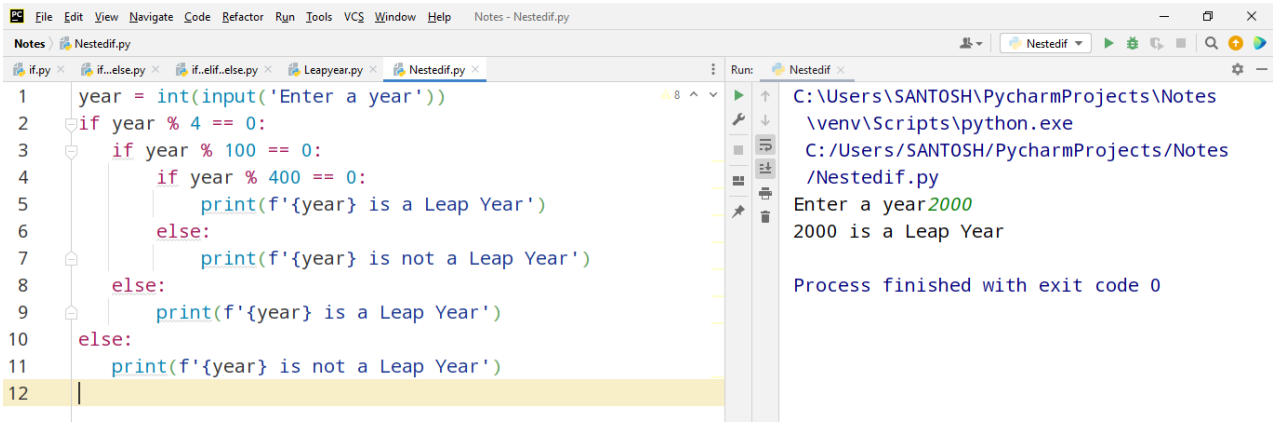
```
val = int(input('Enter a number: '))

if val>=0:
     if val == 0:
          print('Number is Zero')
     else:
          print('Number is positive')
else:
     print('Number is negative')
```

When we run the above code, it produces the output as follows




Example 2: **Program to Check whether the entered year is leap year or not using Nested if**

```
year = int(input('Enter a year'))
if year % 4 == 0:
   if year % 100 == 0:
       if year % 400 == 0:
           print(f'{year} is a Leap Year')
       else:
           print(f'{year} is not a Leap Year')
   else:
       print(f'{year} is a Leap Year')
else:
   print(f'{year} is not a Leap Year')
```




All years which are perfectly divisible by 4 are leap years except for century years (years ending with 00) which is a leap year only it is perfectly divisible by 400.
For example, years like 2012, 2004, 1968 are leap years but 1971, 2006 are not leap years.
Similarly, 1200, 1600, 2000, 2400 are leap years but 1700, 1800, 1900 are not.

## 3. 2 Loop Control Flow Statements:

In Python, the iterative statements are also known as looping statements or repetitive statements. The iterative statements are used to execute a part of the program repeatedly as long as a given condition is True.

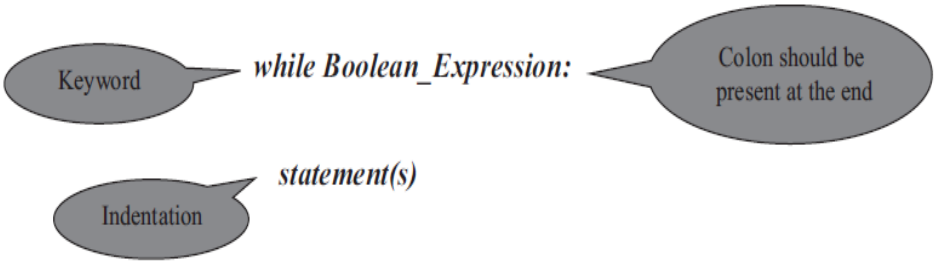
Python provides the following iterative statements.

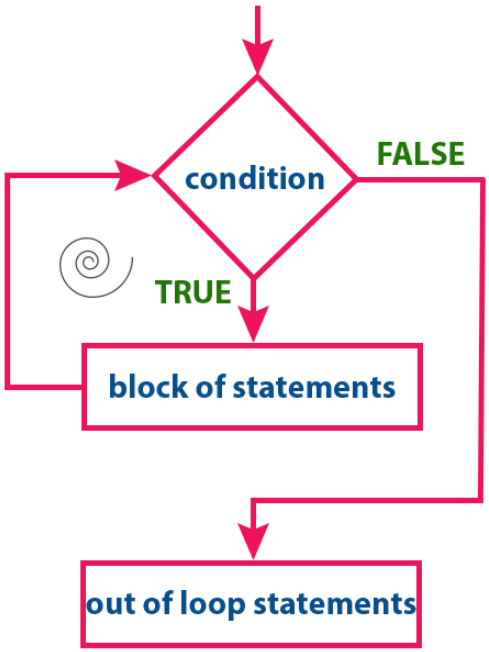
- while statement
- for statement

### *3.2.1 while loop*

- In Python, the while statement is used to execute a set of statements repeatedly.
- In Python, the while statement is also known as entry control loop statement because in the case of the while statement, first, the given condition is verified then the execution of statements is determined based on the condition result.

The syntax for *while* loop is,





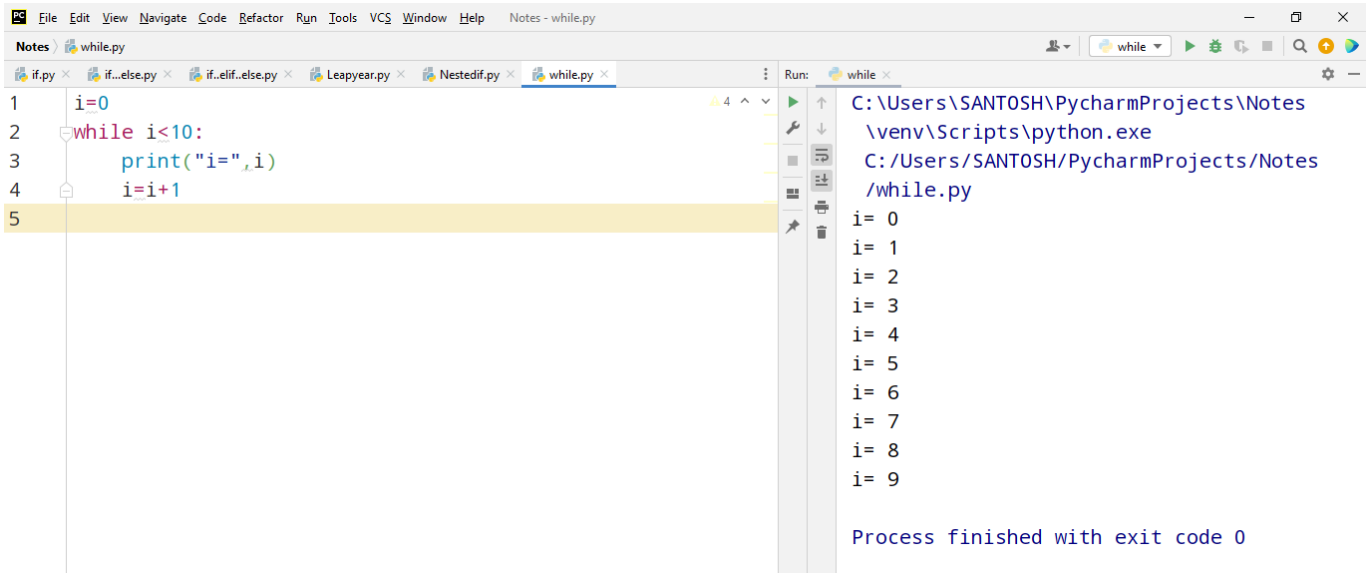

## Syntax

```
while condition:
        Statement_1
        Statement_2
        Statement_3
        ...
```

Example 1: **Write Python Program to Display First 10 Numbers Using while Loop Starting from 0**

```
i=0
while i<10:
    print("i=",i)
    i=i+1
```

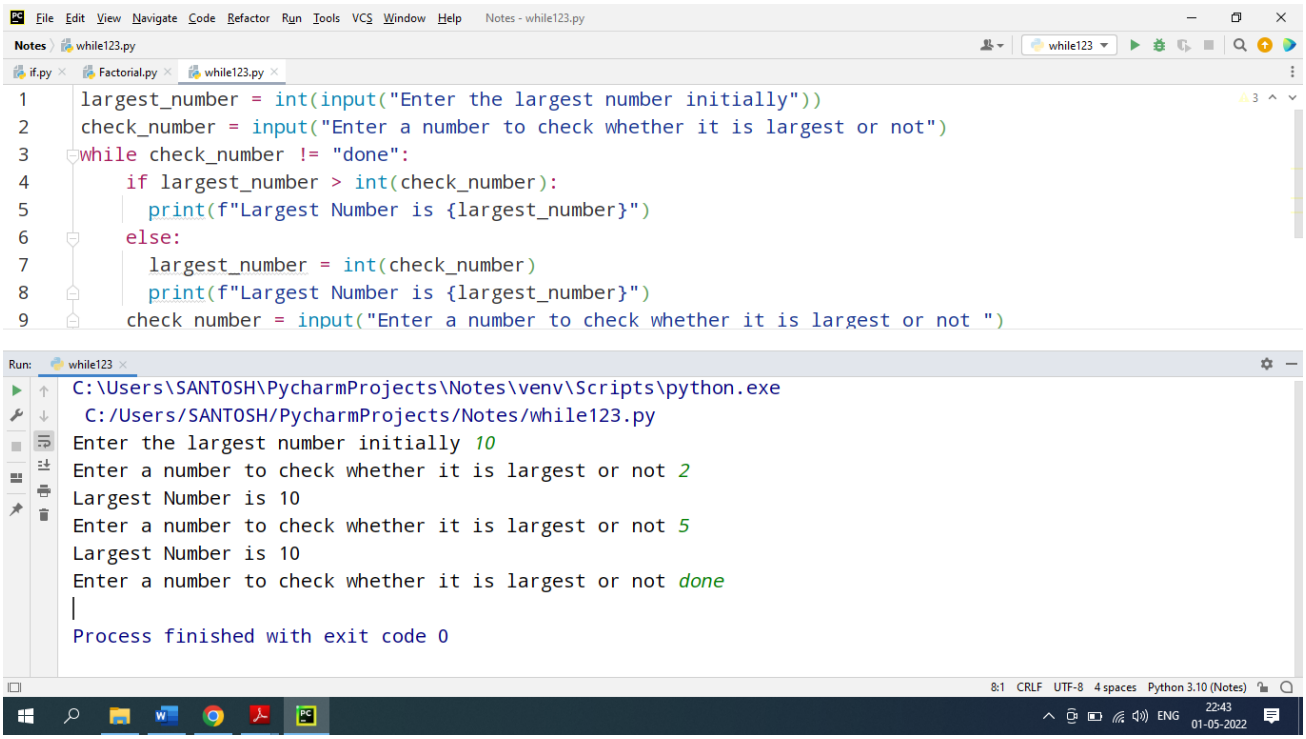
When we run the above code, it produces the output as follows



## Example 2 : **Program to Repeatedly Check for the Largest Number Until the User Enters "done"**

```
largest_number = int(input("Enter the largest number initially"))
check_number = input("Enter a number to check whether it is
                      largest or not")
while check_number != "done":
    if largest_number > int(check_number):
       print(f"Largest Number is {largest_number}")
    else:
      largest_number = int(check_number)
      print(f"Largest Number is {largest_number}")
    check_number = input("Enter a number to check whether it is
                          largest or not")
```

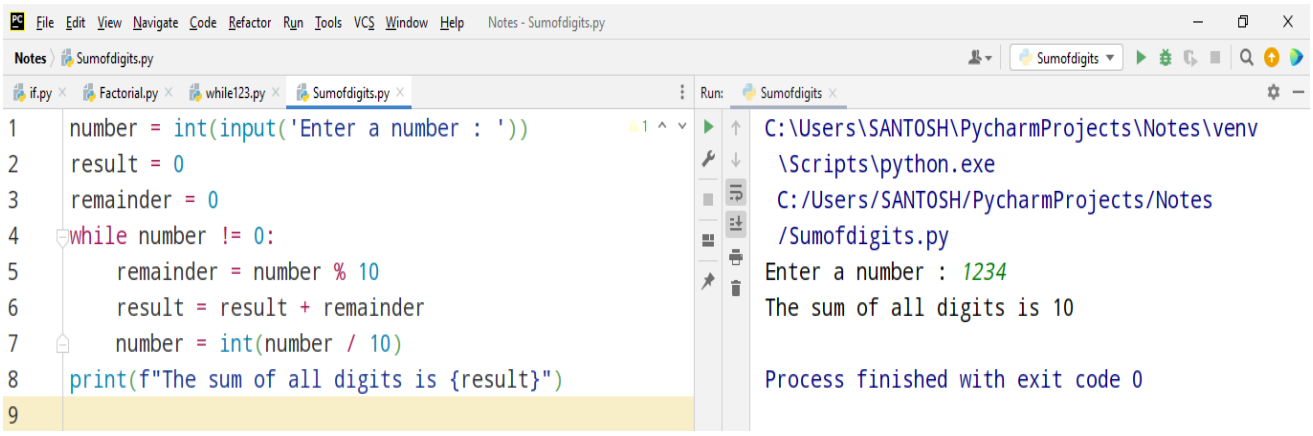
When we run the above code, it produces the output as follows




## Example 3 : **Write Python Program to Find the Sum of Digits in a Number using while**

```
number = int(input('Enter a number : '))
result = 0
remainder = 0
while number != 0:
    remainder = number % 10
    result = result + remainder
    number = int(number / 10)
print(f"The sum of all digits is {result}")
```

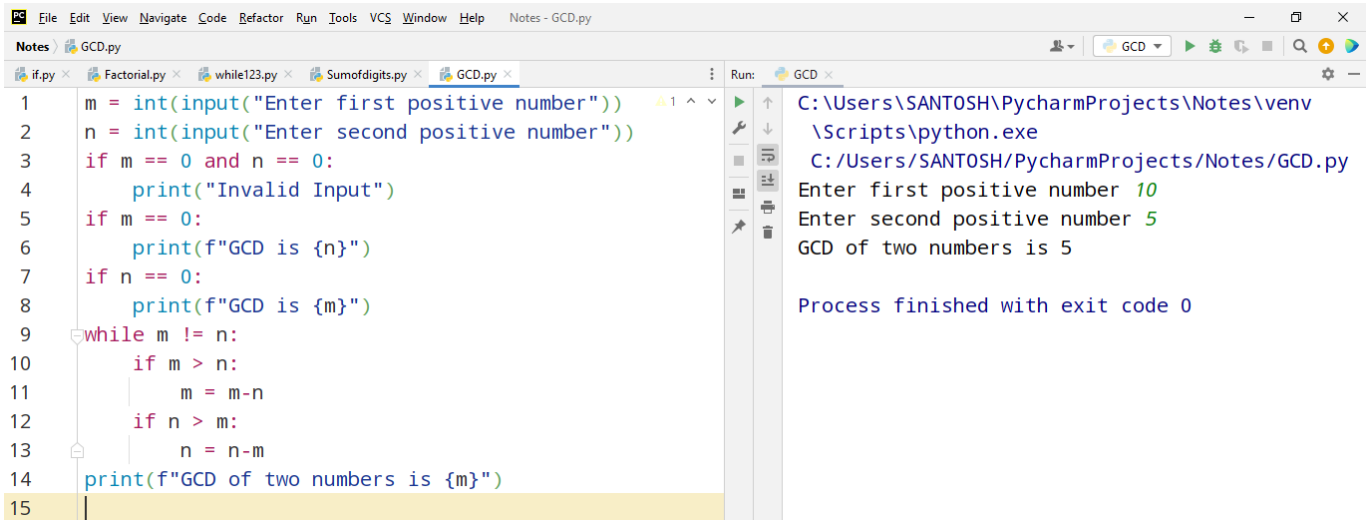
When we run the above code, it produces the output as follows




## Example 4 : **Program to Find the GCD of Two Positive Numbers**

```
m = int(input("Enter first positive number"))
n = int(input("Enter second positive number"))
if m == 0 and n == 0:
    print("Invalid Input")
if m == 0:
    print(f"GCD is {n}")
if n == 0:
    print(f"GCD is {m}")
while m != n:
    if m > n:
        m = m-n
    if n > m:
        n = n-m
print(f"GCD of two numbers is {m}")
```

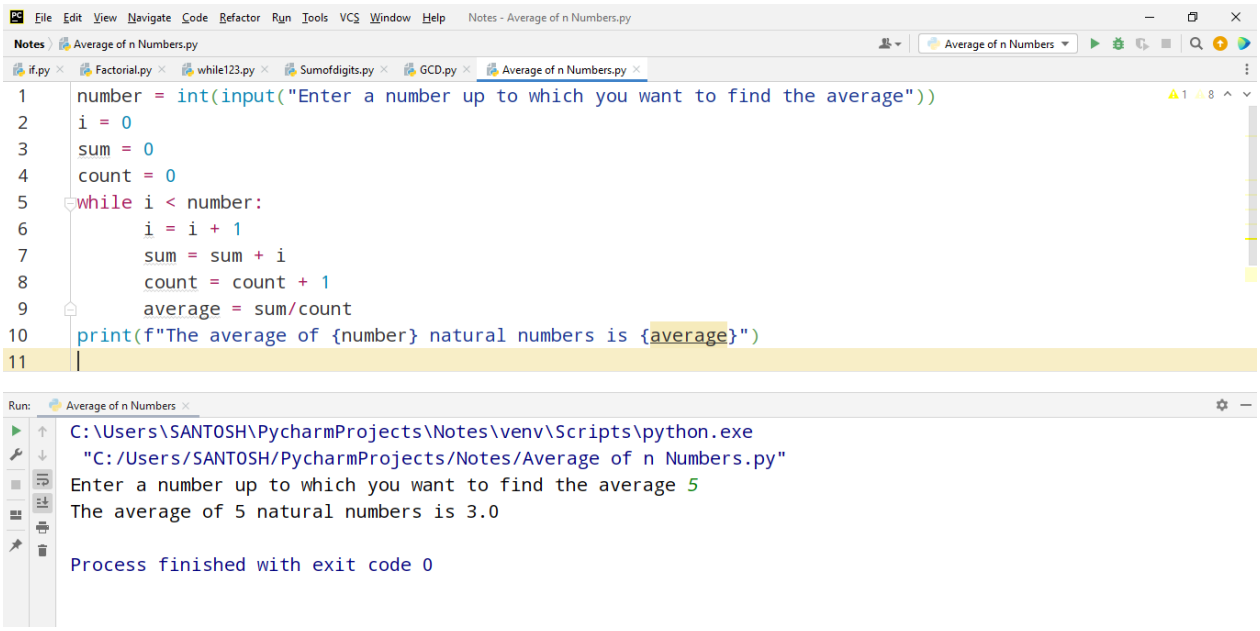
When we run the above code, it produces the output as follows



## Example 5 : Write a Program to Find the Average of *n* Natural Numbers Where *n* Is the Input from the User

```
number = int(input("Enter a number up to which you want to
                find the average"))
i = 0
sum = 0
count = 0
while i < number:
      i = i + 1
      sum = sum + i
      count = count + 1
      average = sum/count
 print(f"The average of {number} natural numbers is
       {average}")
```

When we run the above code, it produces the output as follows

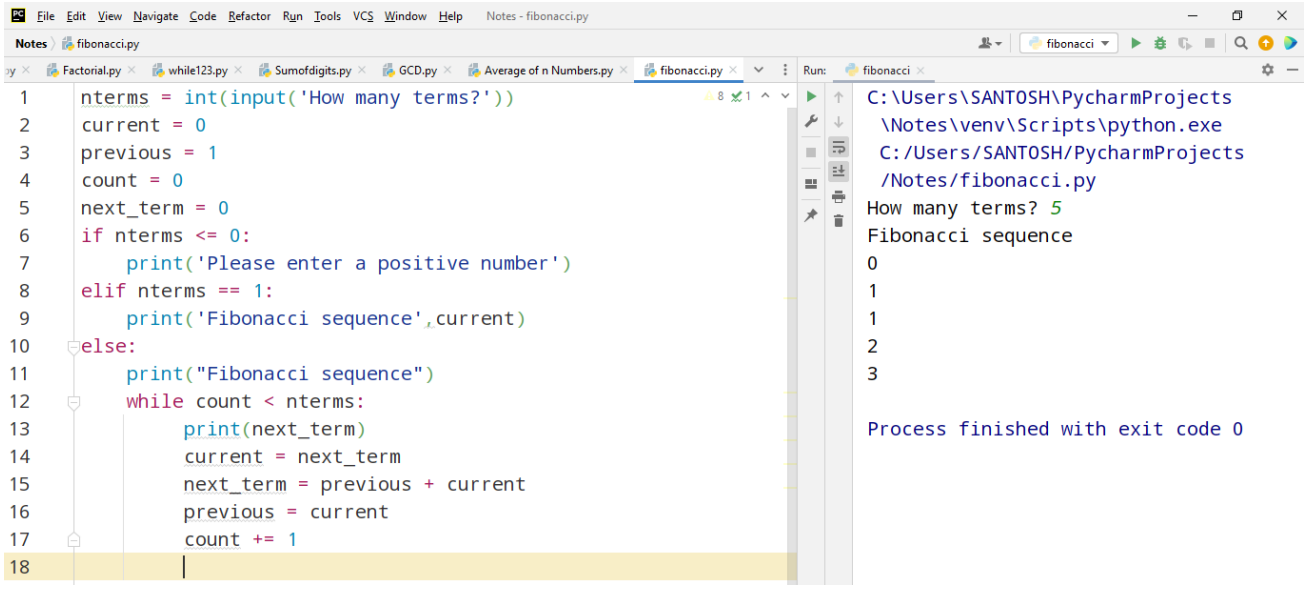



## Example 6: Write a Program to Display the Fibonacci Sequences up to *n*th Term Where *n* is Provided by the User

```
nterms = int(input('How many terms?'))
current = 0
previous = 1
count = 0
next_term = 0
if nterms <= 0:
    print('Please enter a positive number')
elif nterms == 1:
    print('Fibonacci sequence',current)
else:
    print("Fibonacci sequence")
```

```
while count < nterms:
      print(next_term)
      current = next_term
      next_term = previous + current
      previous = current
      count += 1
```

When we run the above code, it produces the output as follows

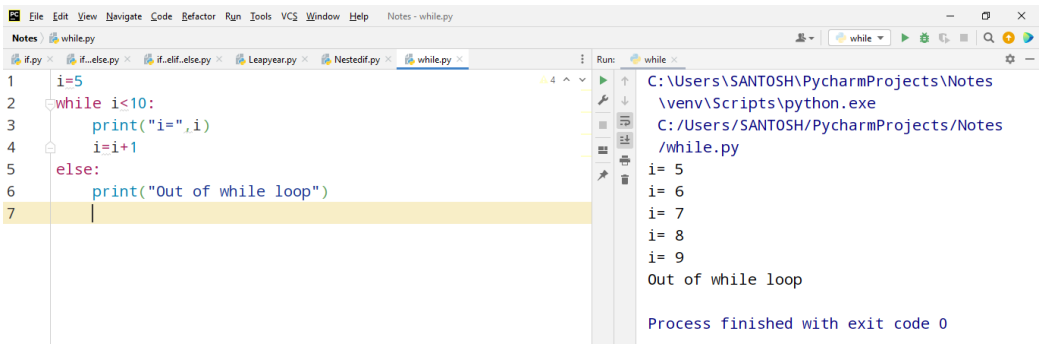



## while statement with 'else' clause in Python

In Python, the else clause can be used with a while statement. The else block is gets executed whenever the condition of the while statement is evaluated to false. But, if the while loop is terminated with **break** statement then else doesn't execute.

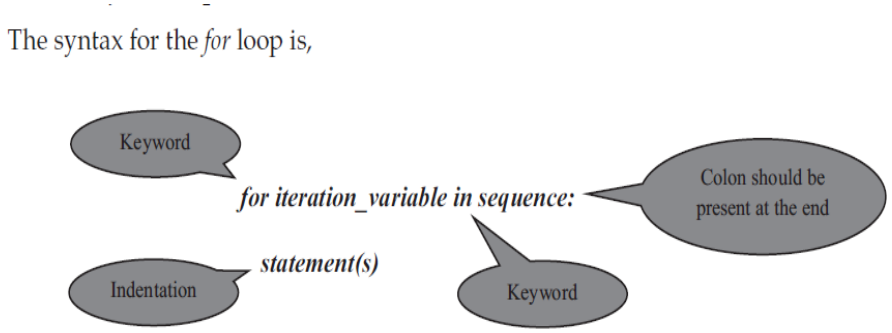
```
i=5
while i<10:
     print("i=",i)
     i=i+1
else:
     print("Out of while loop")
```

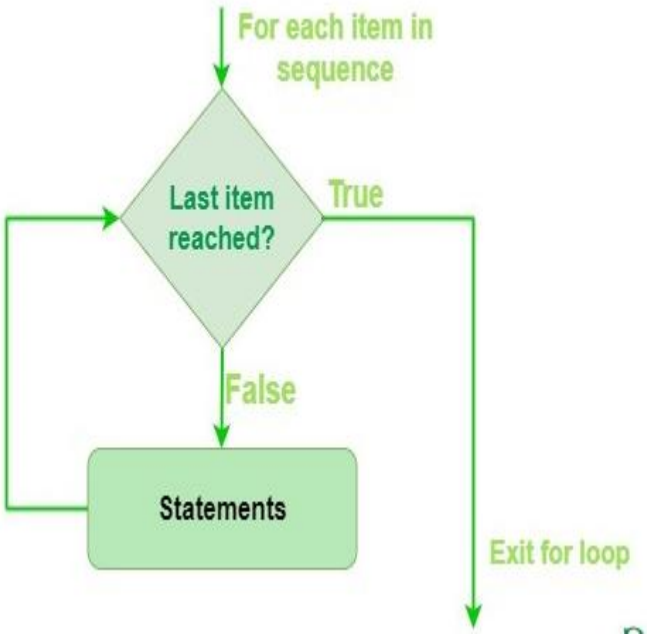
When we run the above code, it produces the output as follows



### *3.2.2 for loop*

In Python, the for statement is used to iterate through a sequence like a list, a tuple, a set, a dictionary, or a string.
The for statement is used to repeat the execution of a set of statements for every element of a sequence.
The general syntax of for statement in Python is as follows.




**Syntax**

```
for <variable> in <sequence>:
        Statement_1
        Statement_2
        Statement_3
        ...
```

The *range()* function generates a sequence of numbers which can be iterated through using *for* loop.

The syntax for *range()* function is,

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

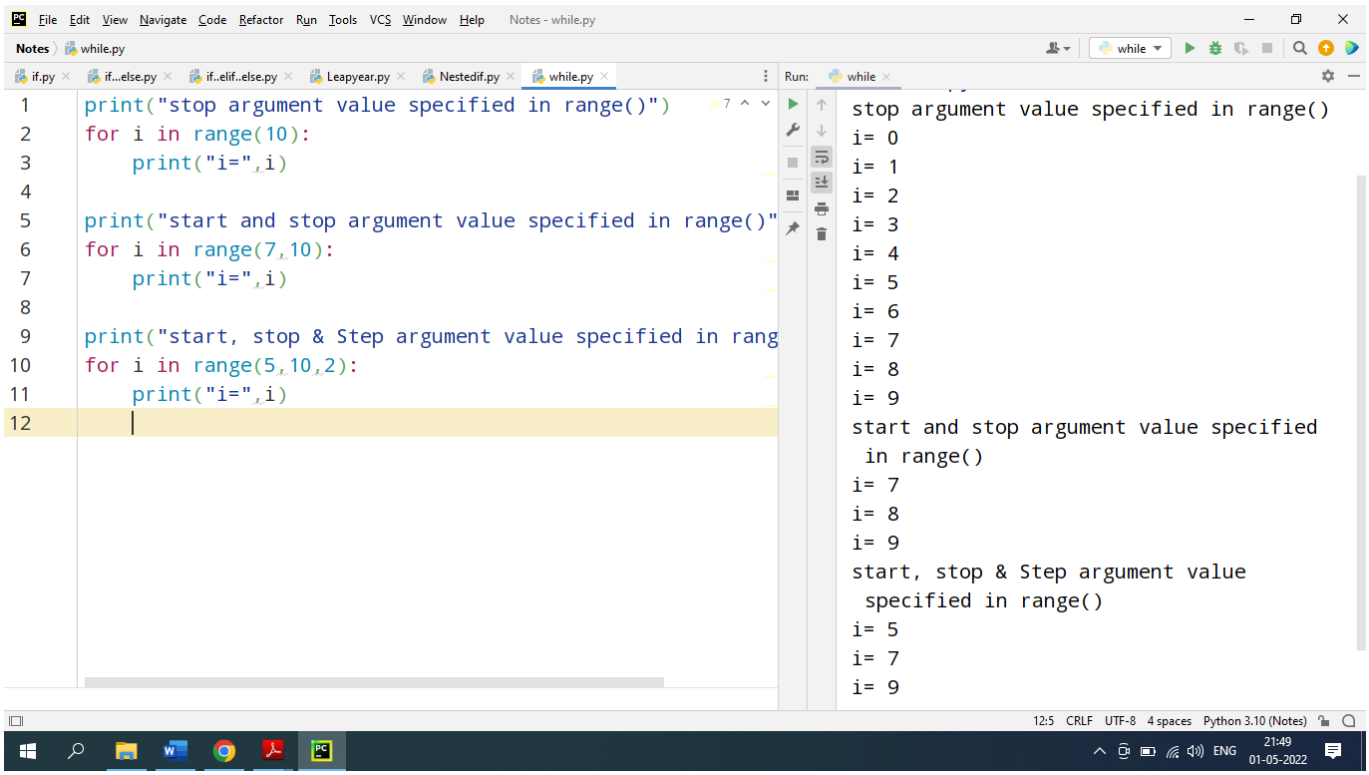
Both **start** and **step** arguments are optional and the range argument value should always be an integer.

***start*** → value indicates the beginning of the sequence. If the **start** argument is not specified, then the sequence of numbers start from zero by default.

***stop*** → Generates numbers up to this value but not including the number itself.

***step*** → indicates the difference between every two consecutive numbers in the sequence. The **step** value can be both negative and positive but not zero.

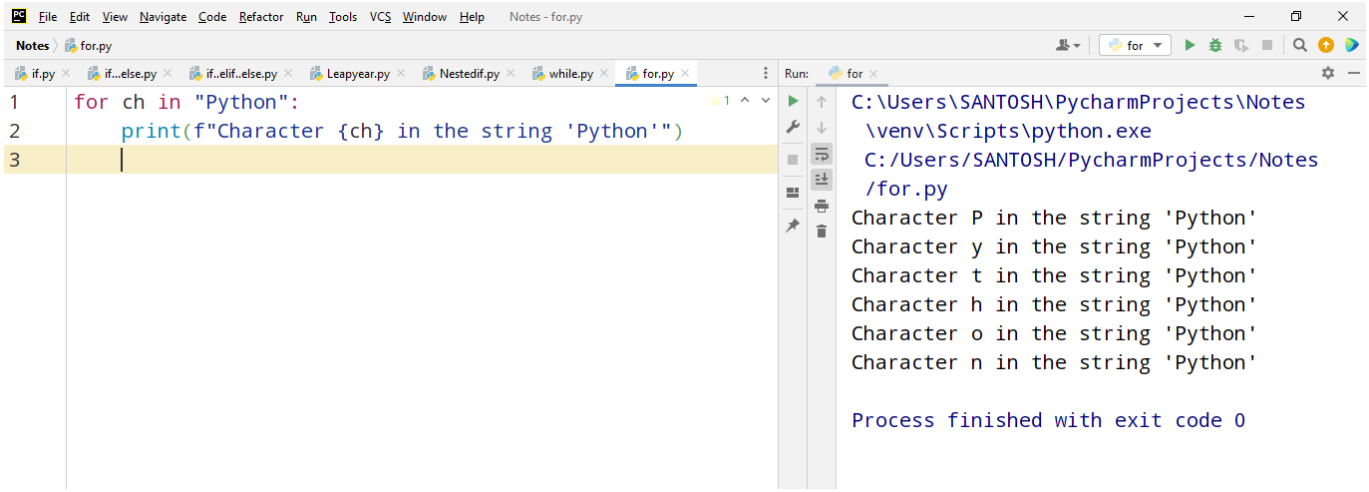
```
print("stop argument value specified in range()")
for i in range(10):
    print("i=",i)

print("start and stop argument value specified in range()")
for i in range(7,10):
    print("i=",i)

print("start, stop & Step argument value specified in
range()")
for i in range(5,10,2):
    print("i=",i)
```




## Example 2: **Program to Iterate through Each Character in the String Using for Loop**

```
for ch in "Python":
    print(f"Character {ch} in the string 'Python'")
```

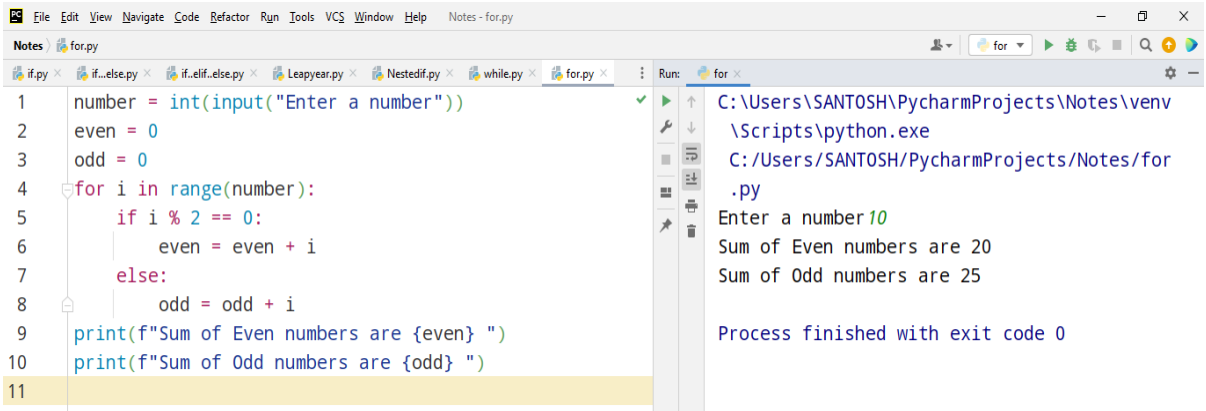
When we run the above code, it produces the output as follows




Example 3: **Write a Program to Find the Sum of All Odd and Even Numbers up to a Number Specified by the User**

```
number = int(input("Enter a number"))
even = 0
odd = 0
for i in range(number):
    if i % 2 == 0:
        even = even + i
    else:
        odd = odd + i
print(f"Sum of Even numbers are {even} ")
print(f"Sum of Odd numbers are {odd} ")
```

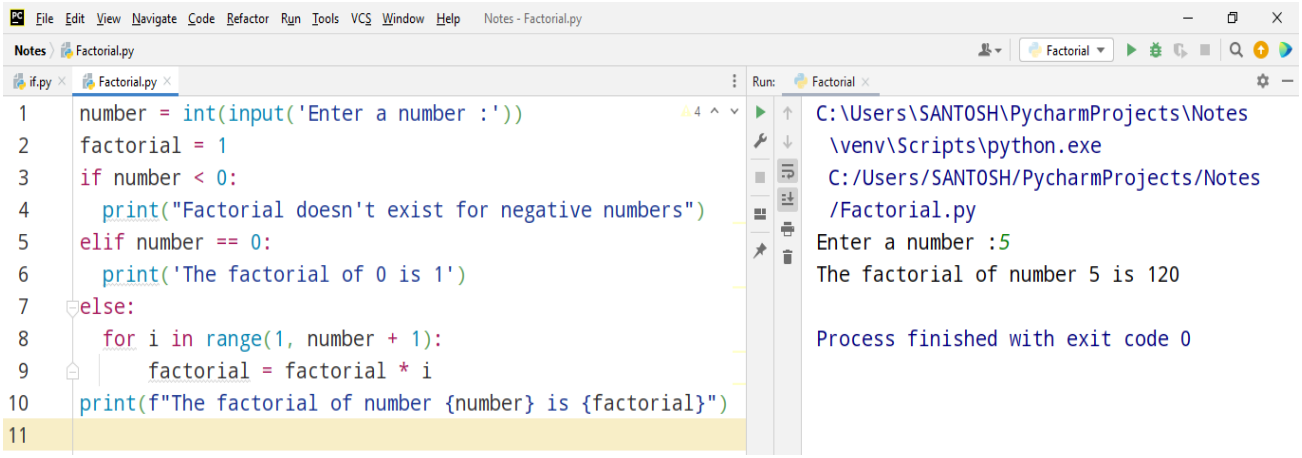
When we run the above code, it produces the output as follows



Example 4: **Write a Program to Find the Factorial of a Number**

```
number = int(input('Enter a number :'))
factorial = 1
if number < 0:
  print("Factorial doesn't exist for negative numbers")
elif number == 0:
  print('The factorial of 0 is 1')
else:
  for i in range(1, number + 1):
      factorial = factorial * i
print(f"The factorial of number {number} is {factorial}")
```

When we run the above code, it produces the output as follows




## *3.3 The continue and break Statements*

In Python, there are control statements that do not need any condition to control the program execution flow. These control statements are called as ***unconditional control statements***.

Python programming language provides the following ***unconditional control statements...***

- ***break***
- ***continue***

### 3.3.1 break statement :

**break** statement is also used to terminate looping statements like while and for. When the **break** statement is encountered inside the looping statement, the execution control moves out of the looping statements.

Syntax : **break**;

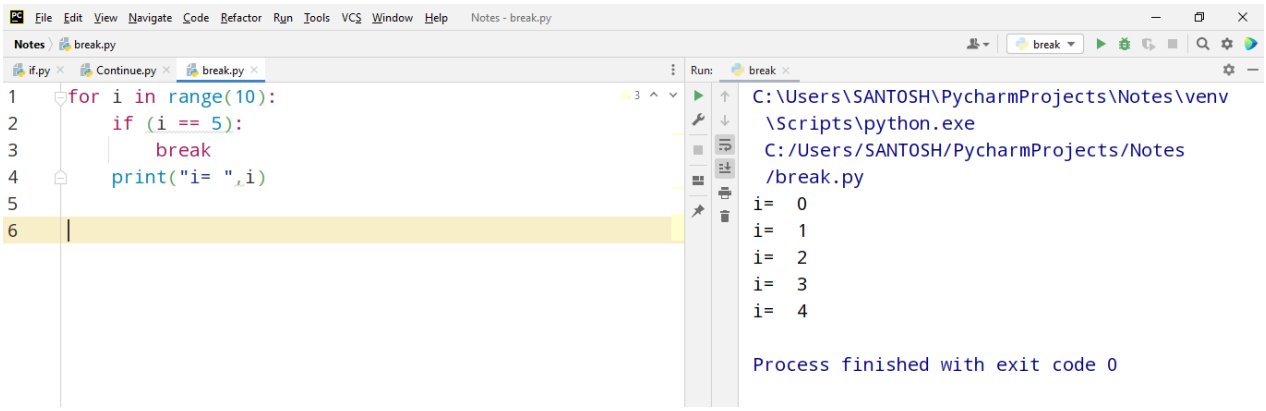
(or)

**break**

## Example 1: **Program on break using for loop**

```
for i in range(10):
    if (i == 5):
        break
    print("i= ",i)
```

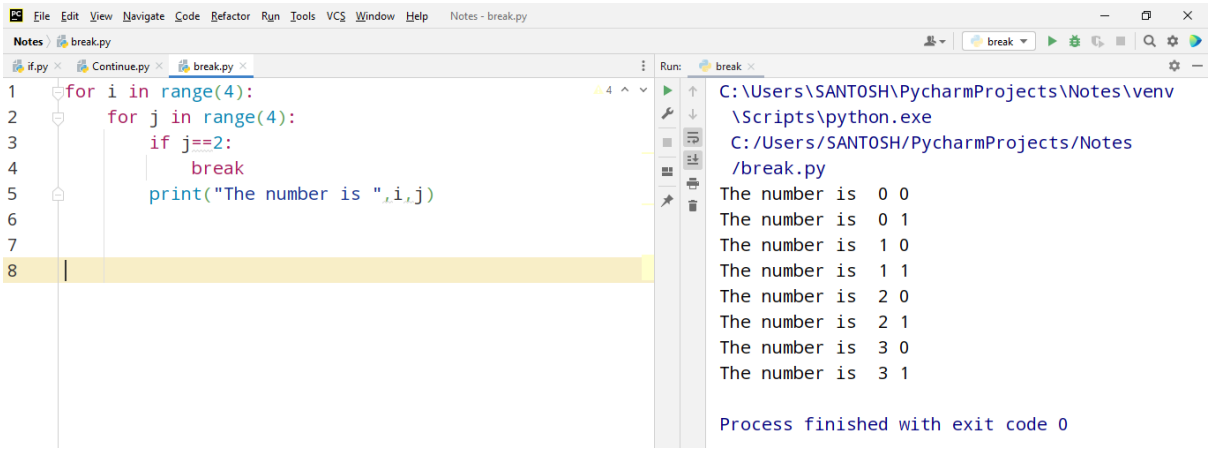
When we run the above code, it produces the output as follows




## Example 2: **Program on break using for loop**

```
for i in range(4):
    for j in range(4):
        if j==2:
            break
        print("The number is ",i,j)
```

When we run the above code, it produces the output as follows



## 3.3.2 continue statement

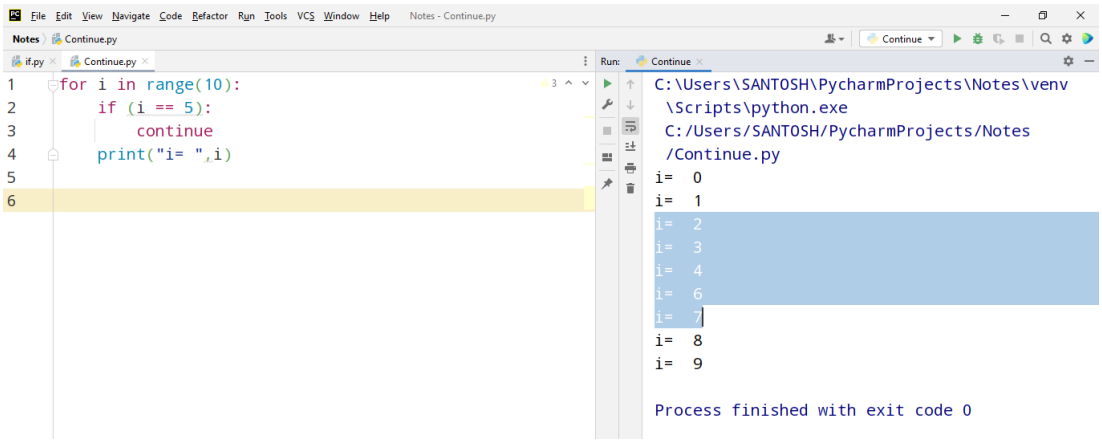
The **continue** statement is used to move the program execution control to the beginning of the looping statement.

When the **continue** statement is encountered in a looping statement, the execution control skips the rest of the statements in the looping block and directly jumps to the beginning of the loop. The **continue** statement can be used with looping statements like while, do-while and for.

### Example 1: **Program on continue using for loop**

```
for i in range(10):
    if (i == 5):
        continue
    print("i= ",i)
```

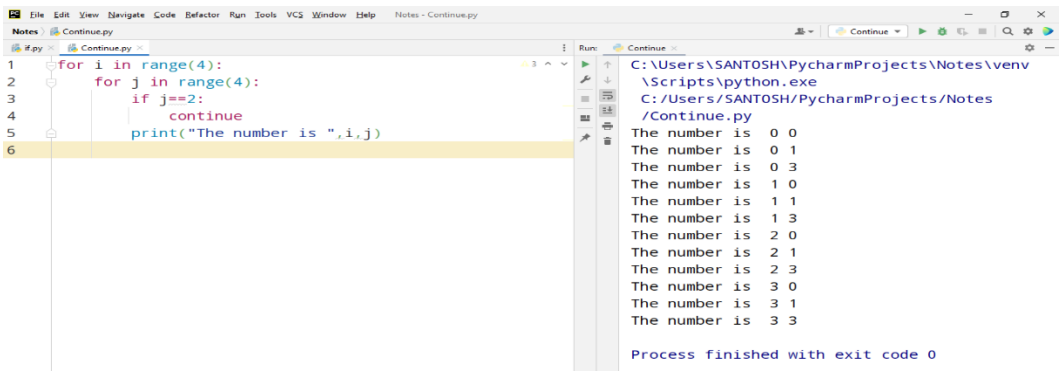
When we run the above code, it produces the output as follows




### Example 2: **Program on continue using for loop**

```
for i in range(4):
    for j in range(4):
        if j==2:
            continue
        print("The number is ",i,j)
```

When we run the above code, it produces the output as follows



# Unit-I Questions

1. What is Computing Device? What are features of computing device?
2. Explain in details about Identification of Computational Problems (Algorithm, Pseudo code and Flowchart)
3. Define a Variable and identifier . How to assign values to Variables.
4. Explain different Operators n Python with examples
5. Explain precedence and associativity of operators with examples
6. Explain Type conversion in Python with examples.
7. Write a short note on data type in Python
8. Briefly explain the Conditional / Decision Making statements available in python
9. Briefly explain the Looping / Iterator statements available in python
10. What is the purpose of using break and continue?
11. Explain the use of range( ) function with an example.
12. Write a short note on Dynamic and Strongly Typed Language