



---

# BASICS OF PYTHON PROGRAMMING

---

CLASS VIII TERM II



## INDEX

SNO	CONTENT	PAGE NUMBER
1	Basics of Programming in Python	2
2	Input and Output	8
3	Tokens	13
4	Control Structures – Decision Control statements	22
5	Control Structures – Iterative Statements	28
6	Lab Exercises	30
7	Bibliography	37

# Chapter 1

## Basics of Programming in Python

### Introduction

Python is a general purpose programming language created by Guido Van Rossum from CWI (Centrum Wiskunde & Informatica) which is a National Research Institute for Mathematics and Computer Science in Netherlands. The language was released in 1991. Python got its name from a BBC comedy series from seventies- “Monty Python’s Flying Circus”. Python is a case sensitive programming language.

### Key Features of Python

- ✓ It is a general purpose programming language which can be used for both scientific and non-scientific programming.
- ✓ It is a platform independent programming language.
- ✓ The programs written in Python are easily readable and understandable.

The version 3.x of Python IDLE (Integrated Development Learning Environment) is used to develop and run Python code. It can be downloaded from the web resource [www.python.org](http://www.python.org).

### Programming in Python

In Python, programs can be written in two ways namely interactive mode and Script mode. The Interactive mode allows us to write codes in Python command prompt (>>>) whereas in script mode programs can be written and stored as separate file with the extension .py and executed. Script mode is used to create and edit python source file.

### Interactive mode Programming

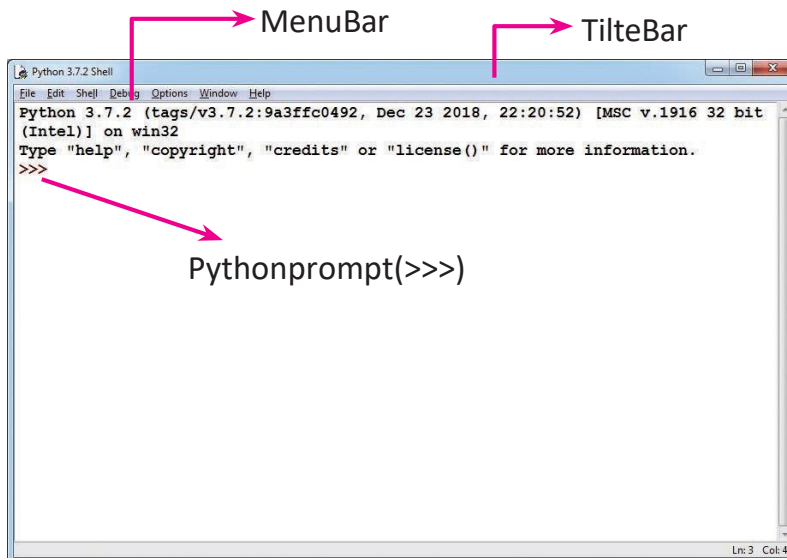
In interactive mode Python code can be directly typed and the interpreter displays the result(s) immediately. The interactive mode can also be used as a **simple calculator**

## Invoking Python IDLE

The following command can be used to invoke Python IDLE from Window OS.

**Start→AllPrograms→Python3.x→IDLE(Python3.x)**

Now **Python IDLE** window appears as shown below



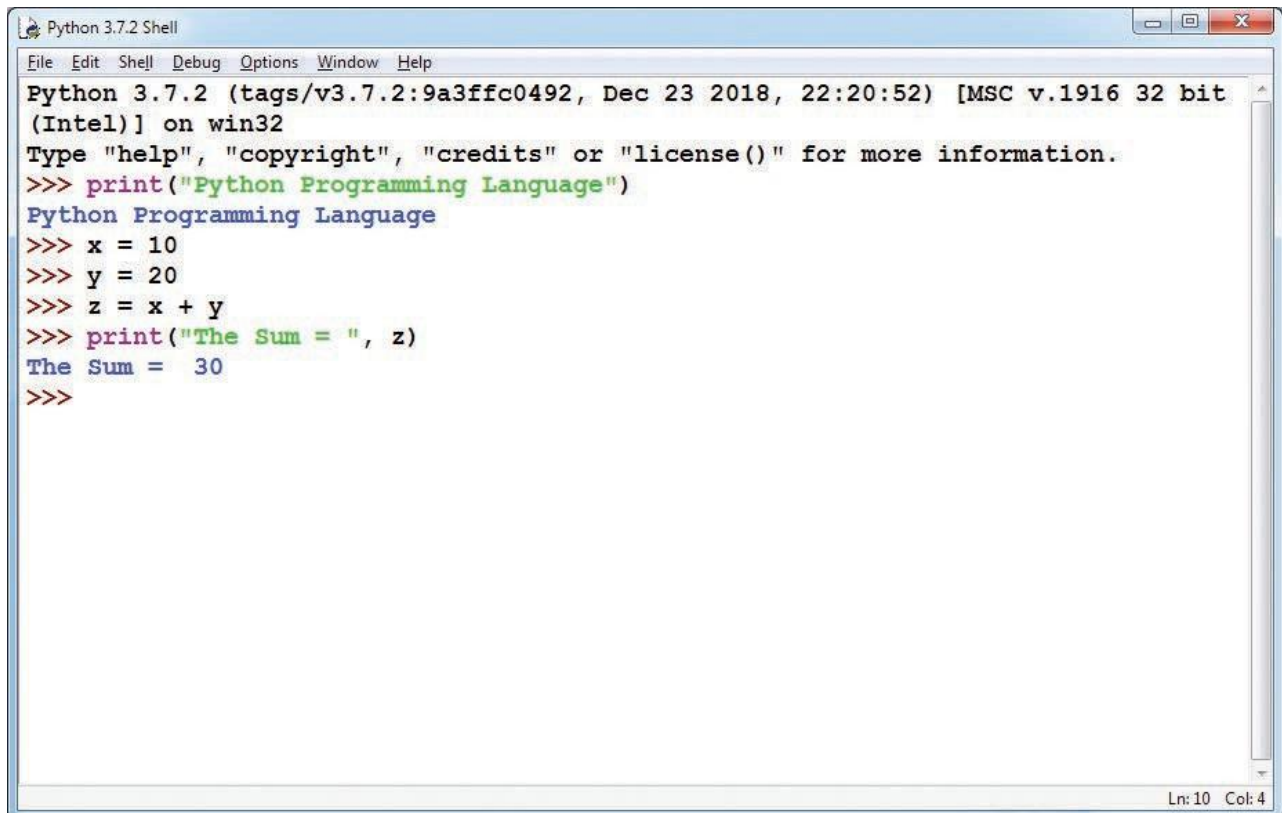
The prompt (>>>) indicates that Interpreter is ready to accept instructions. Therefore, the prompt on screen means IDLE is working in interactive mode. Now let us try as a simple calculator by using a simple mathematical expressions.

### Example1:

```
>>>5+10  
15  
>>>5+50*10  
505  
>>>5**2  
25
```

### Example2:

```
>>>print ("Python Programming Language")  
Python Programming Language  
>>>x=10  
>>>y=20  
>>>z=x+y  
>>>print("TheSum =",z)  
The Sum = 30
```



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Python Programming Language")
Python Programming Language
>>> x = 10
>>> y = 20
>>> z = x + y
>>> print("The Sum = ", z)
The Sum = 30
>>>
```

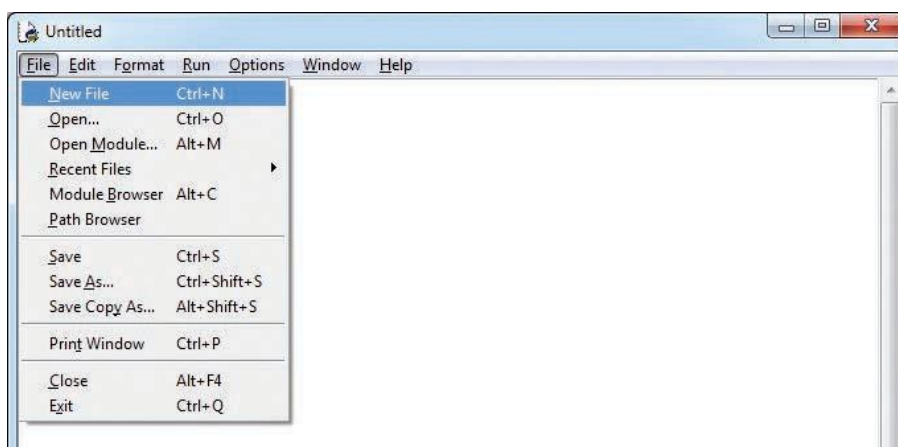
Python Interactive Window

### Script mode Programming

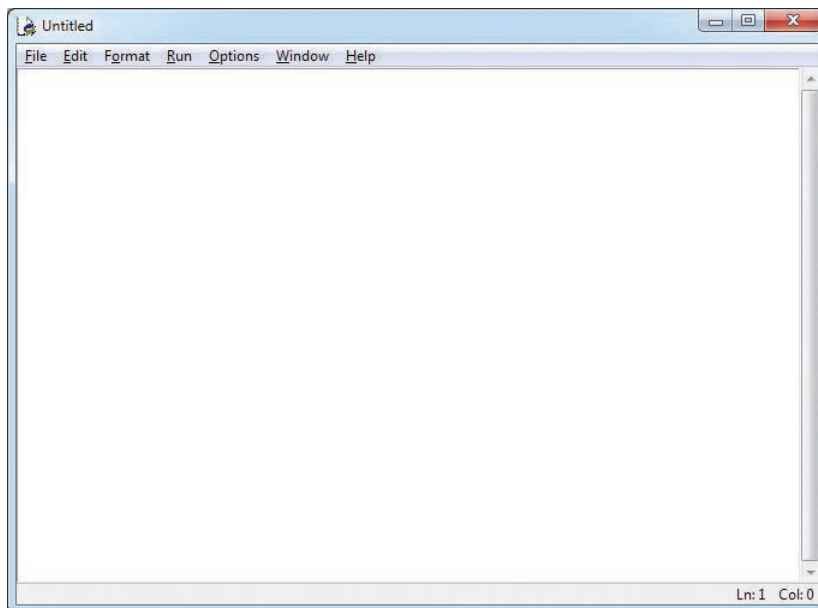
Basically, a script is a text file containing the Python statements. Python Scripts are reusable code. Once the script is created, it can be executed again and again without retyping. The Scripts are editable.

### Creating Scripts in Python

Step 1: Choose **File** → **New File** or press **Ctrl + N** in Python shell window



Step 2: An **untitled** blank script text editor will be displayed on screen as shown below



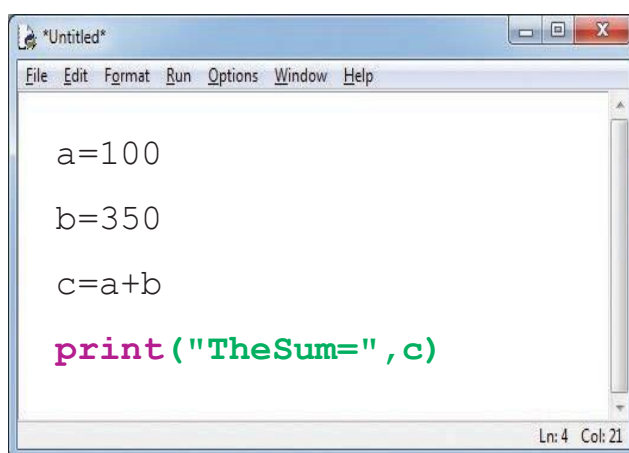
Step 3: Type the following code in the script editor.

```
a=100
```

```
b = 350
```

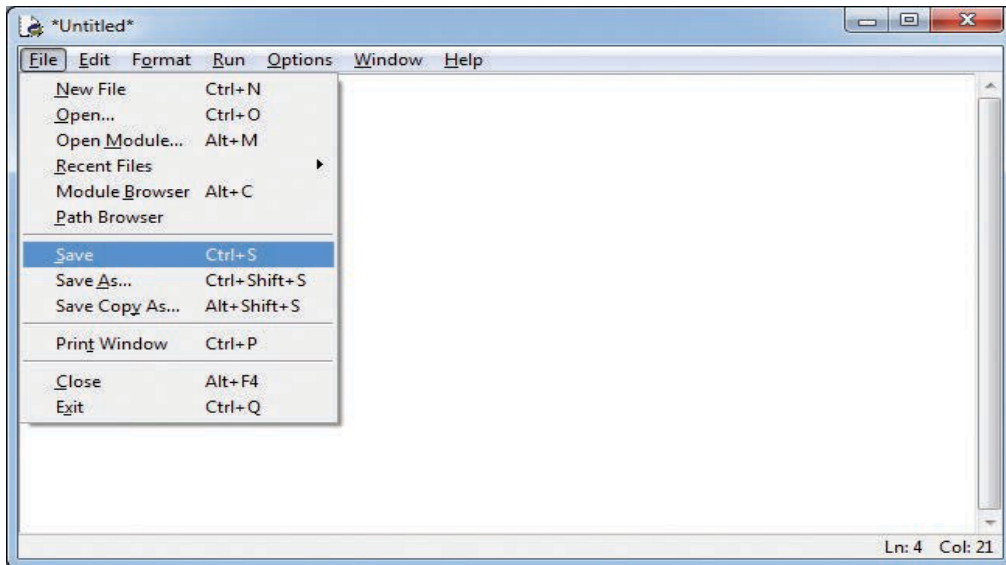
```
c = a+b
```

```
print ("The Sum=", c)
```

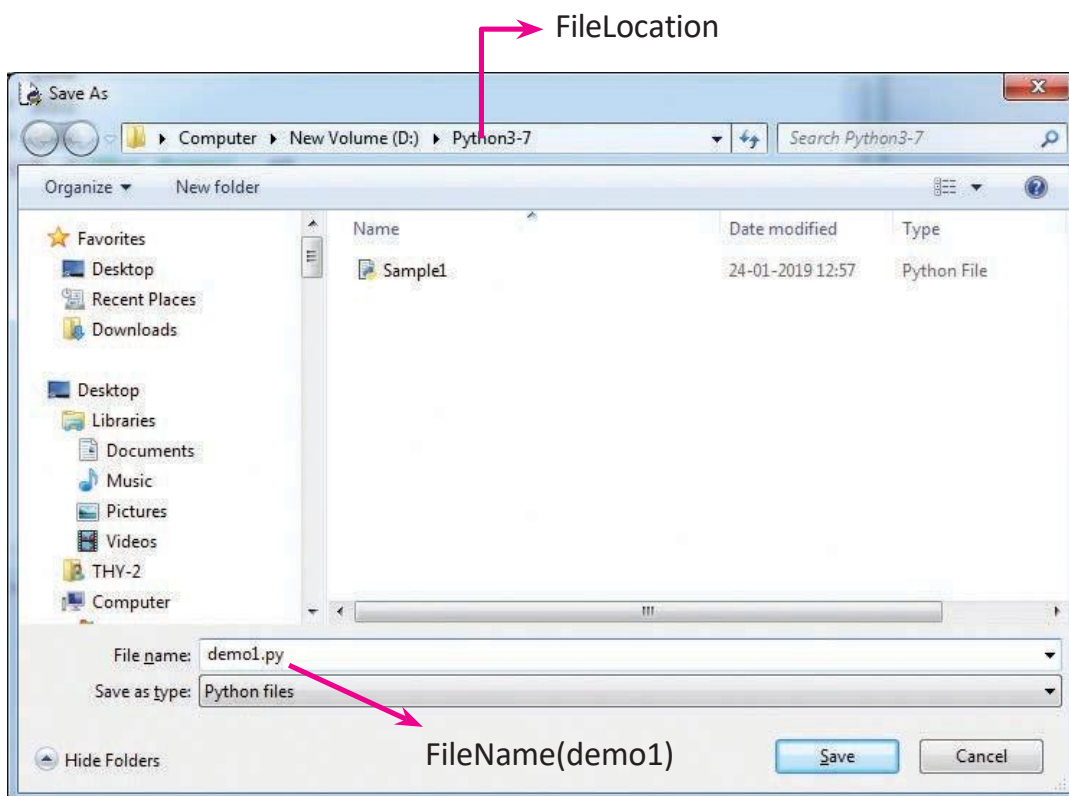


Step 4: Saving Python script

a) Choose **File** → **Save** or Press **Ctrl + S**



b) Now, **Save As** dialog box appears on the screen as shown

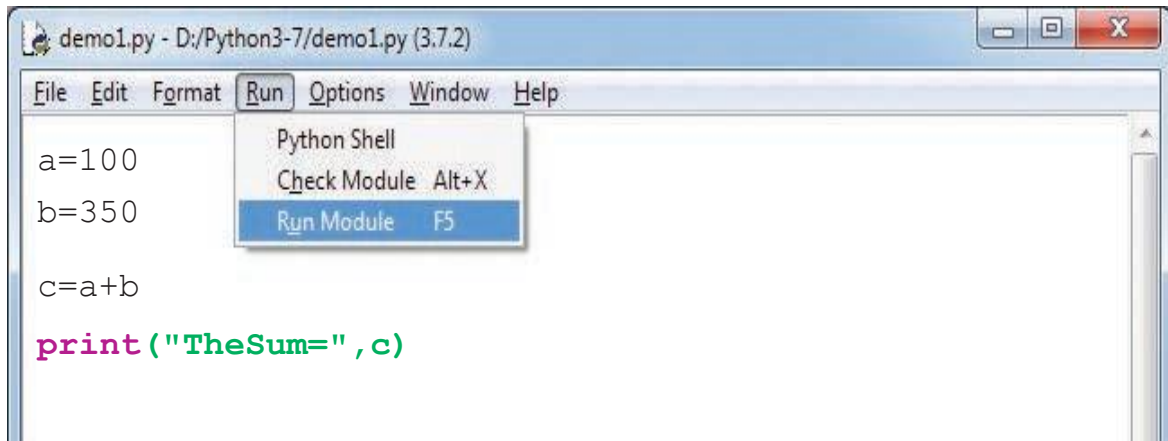


In the **Save As** dialog box, select the location where you want to save your Python code, and type the file name in **File Name** box. Python files are by default saved with extension **.py**. Thus, while creating Python scripts using Python Script editor, no need to specify the file extension.

Finally, click **Save** button to save your Python script.

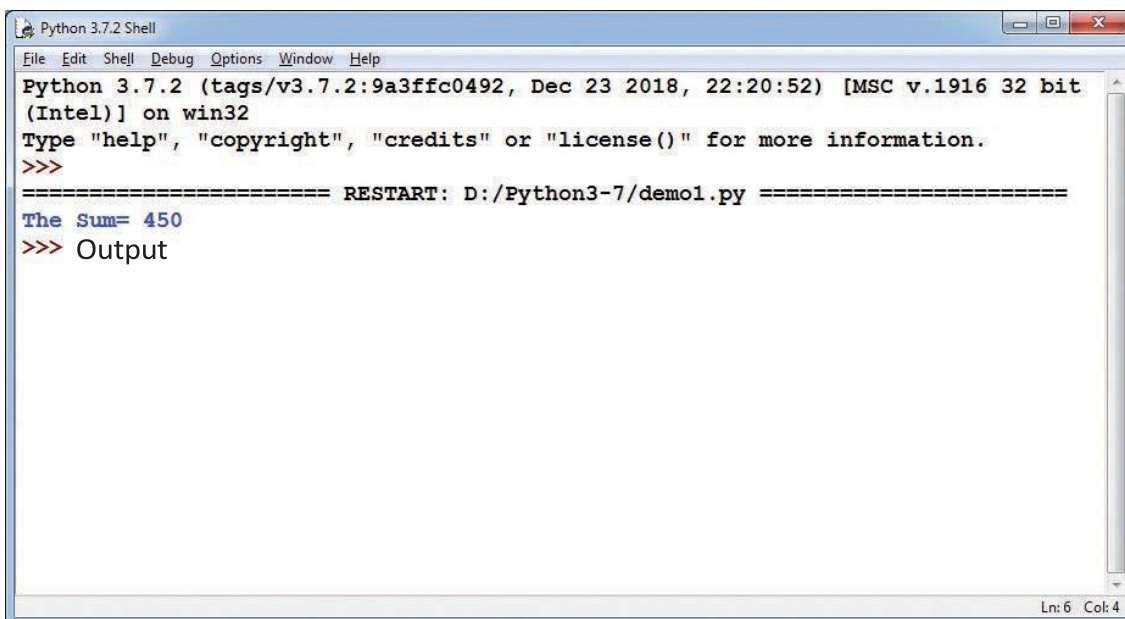
## Executing Python Script

Choose **Run** → **Run Module** or Press **F5**



If your code has any error, it will be shown in red color in the IDLE window, and Python describes the type of error occurred. To correct the errors, go back to Script editor, make corrections, save the file using **Ctrl + S** or **File** → **Save** and execute it again.

For all error free code, the output will appear in the IDLE window of Python as shown below





## Chapter 2 Input and Output

### Input and Output Functions

A program needs to interact with the user to accomplish the desired task; this can be achieved using **Input-Output functions**. The **input()** function helps to enter data at run time by the user and the output function **print()** is used to display the result of the program on the screen after execution.

#### The **print()** function

In Python, the **print()** function is used to display result on the screen.

The syntax for **print()** is as follows: `print(" Content to be displayed ")`

#### Example

```
print("string to be displayed as output")  
  
print (variable )  
  
print("String to be displayed as output",variable)  
print("String1",variable,"String2",variable,"String3" .....)
```

### Example

```
>>>print("Welcome to Python Programming")
      Welcome to Python Programming

>>>x=5

>>>y=6

>>>z=x+y

>>>print(z)

      11

>>>print("The sum =",z)
      The sum = 11
```

The **print ( )** evaluates the expression before printing it on the monitor. The print ( ) displays an entire statement which is specified within print ( ). **Comma ( , )** is used as a separator in **print ( )** to print more than one item.

### Input() Function

In Python, **input( )** function is used to accept data as input at run time.

The syntax for **input()** function is,

```
variable = input( " Prompt String")
```

Where, **prompt string** in the syntax is a statement or message to the user, to know what input can be given.

If a prompt string is used, it is displayed on the monitor; the user can provide expected data from the input device.

The **input( )** takes whatever is typed from the keyboard and stores the entered data in the given variable.

If prompt string is not given in **input( )** no message is displayed on the screen, thus, the user will not know what is to be typed as input.

#### Example1:input() with prompt string

```
>>>city=input("Enter Your City:")Enter Your City: Madurai
>>>print("I am from",city) I am from Madurai
```

#### Example2:input() without prompt string

```
>>>city=input()
Rajarajan
>>>print("I am from ",city)
I am from Rajarajan
```

Note that in example-2, the **input( )** is not having any prompt string, thus the user will not know what is to be typed as input. If the user inputs irrelevant data as given in the above example, then the output will be unexpected. So, to make your program more interactive, provide prompt string with **input( )**.The **input ( )** accepts all data as string or characters but

#### Example3:

```
x=int(input("EnterNumber1:"))
y=int(input("EnterNumber2:"))
print ("The sum = ", x+y)
```

#### Output:

```
EnterNumber1:34
EnterNumber2:56 The sum = 90
```

not as numbers. If a numerical value is entered, the input values should be explicitly converted into numeric data type. The `int( )` function is used to convert string data as integer data explicitly.

#### Example4:Alternate method for the above program

```
x,y=int(input("EnterNumber1:")),int(input("EnterNumber2:"))  
print("X=",x,"Y=",y)
```

#### Output:

```
EnterNumber1:30
```

```
EnterNumber2:50
```

```
X=30 Y=50
```

### Comments in Python

Comments are statements that we use to make the code easier to understand. They are not read by the Python interpreter when it executes the code.

In Python, comments begin with hash symbol (**#**). The lines that begins with **#** are considered as comments and ignored by the Python interpreter.

Comments may be single line or no multi-lines. The multiline comments should be enclosed within a set of `''' '''`(triple quotes) as given below.

*# It is Single line Comment*

*''' It is multiline comment*

*which contains more than one line '''*

## Indentation

Python uses whitespace such as **spaces** and **tabs** to define program blocks whereas other languages like C, C++, java use curly braces { } to indicate blocks of codes for class, functions or body of the loops and block of selection command.

The number of whitespaces (spaces and tabs) in the indentation is not fixed, but all statements within the block must be indented with same amount of spaces.

## Chapter 3

### Tokens

Tokens are the smallest units of a Python program and are the building blocks of its syntax. They are a set of one or more characters that have a meaning together and cannot be broken down further without losing their significance.

Python breaks each logical line into a sequence of elementary lexical components known as **Tokens**. The normal token types are

- 1) Identifiers,
- 2) Keywords,
- 3) Operators,
- 4) Delimiters and
- 5) Literals.

Whitespace separation is necessary between tokens, identifiers or keywords.

#### Identifiers

An Identifier is a name used to identify a variable, function, class, module or object.

Rules to be followed when creating an identifier are-

- An identifier must start with an alphabet (A..Z or a..z) or underscore (`_`).
- Identifiers may contain digits (0 .. 9)
- Python identifiers are case sensitive i.e. uppercase and lowercase letters are distinct.
- Identifiers must not be a **python** keyword.

Python does not allow punctuation character such as `%, $, @` etc., within identifiers

### Example of valid identifiers

Sum, total\_marks, regno, num1

### Example of invalid identifiers

12Name, name\$, total-mark, continue

### Keywords

**Keywords** are special words used by Python interpreter to recognize the structure of program. As these words have specific meaning for interpreter, they cannot be used for any other purpose. List of keywords in Python are:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
As	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

### Operators

In computer programming languages operators are special symbols which represent computations, conditional matching etc. The value of an operator used is called **operands**.

Operators are categorized as Arithmetic, Relational, Logical, Assignment etc.

Value and variables when used with operator are known as **operands**.

## Arithmetic Operators

An arithmetic operator is a mathematical operator that takes two operands and performs a calculation on them. They are used for simple arithmetic. Most computer languages contain a set of such operators that can be used within equations to perform different types of sequential calculations.

Python supports the following Arithmetic operators.

Operator - Operation	Examples	Result
Assume a=100 and b=10. Evaluate the following expressions		
+ (Addition)	>>> a + b	110
- (Subtraction)	>>>a - b	90
* (Multiplication)	>>> a*b	1000
/ (Division)	>>> a / b	10.0
% (Modulus)	>>> a % 30	10
** (Exponent)	>>> a ** 2	10000
// (Floor Division)	>>> a//30 (Integer Division)	3



### #Demo Program to test Arithmetic Operators

```
a=100
b=10
print ("The Sum      = ",a+b)
print ("The Difference = ",a-b)
print ("The Product   = ",a*b)
print("TheQuotient=",a/b)
print ("The Remainder= ",a%30)
print ("The Exponent= ",a**2)
print ("The Floor Division =",a//30)
```

#### Output:

```
TheSum      =110
TheDifference =90
TheProduct  =1000
TheQuotient =10.0
TheRemainder =10
The Exponent =10000
The Floor Division = 3
```

### Relational or Comparative Operator

A Relational operator is also called as **Comparative** operator which checks the relationship between two operands. If the relation is true, it returns **True**; otherwise it returns **False**.

Python supports following relational operators

Operator - Operation	Examples	Result
Assume the value of a=100 and b=35. Evaluate the following expressions.		
== (is Equal)	>>> a==b	False

> (Greater than)	>>> a > b	True
< (Less than)	>>> a < b	False
>= (Greater than or Equal to)	>>> a >= b	True
<= (Less than or Equal to)	>>> a <= b	False
!= (Not equal to)	>>> a != b	True

#Demo Program to test Relational Operators

```
a=int (input("Enter a Value for A:"))
```

```
b=int (input("Enter a Value for B:"))
```

```
print ("A = ",a," and B = ",b)
```

```
print ("The a==b = ",a==b)
```

```
print ("The a > b = ",a>b)
```

```
print ("The a < b = ",a<b)
```

```
print("Thea>=b=",a>=b)
```

```
print("Thea<=b=",a<=b)
```

```
print ("The a != b = ",a!=b)
```

### Output:

EnterValueforA:35

EnterValueforB:56

A=35 and B =56

The a==b =False

The a>b =False

The a<b =True  
 The a>=b =False  
 The a<=b =False  
 The a!=b =True

### Logical Operator

In python, Logical operators are used to perform logical operations on the given relational expressions. There are three logical operators they are **and**, **or** and **not**.

Operator	Example	Result
Assume a = 97 and b = 35, Evaluate the following Logical expressions		
Or	>>> a>b or a==b	True
And	>>> a>b and a==b	False
Not	>>> not a>b	False i.e. Not True

Example – Code	Example - Result
<pre>a=int (input("Enter a Value for A:")) b=int (input("Enter a Value for B:")) print ("A = ",a, " and b = ",b) print ("The a &gt; b or a == b = ",a&gt;b or a==b) print ("The a &gt; b and a == b = ",a&gt;b and a==b) print ("The not a &gt; b = ",not a&gt;b)</pre>	<pre>Enter a Value for A:50 Enter a Value for B:40 A = 50 and b = 40 The a &gt; b or a == b = True The a &gt; b and a == b = False The not a &gt; b = False</pre>

## Assignment Operator

In Python, = is a simple assignment operator to assign values to variable.

Let **a = 5** and **b = 10** assigns the value 5 to **a** and 10 to **b**

These two assignment statement can also be given as **a,b=5,10** that assigns the value 5 and 10 on the right to the variables a and b respectively.

There are various compound operators in Python like +=, -=, \*=, /=, %=, \*\*= and //= are also available.

## Operators used in Strings : + and \*

The + operator is used to concatenate strings and \* operator is used to replicate strings

Example

Output

```
a="Namaste"
```

```
NamasteEveryone
```

```
b="Everyone"
```

```
print(a+b)
```

```
print(a*2)
```

```
NamasteNamaste
```

## Delimiters

Python uses the symbols and symbol combinations as delimiters in expressions, lists, dictionaries and strings. Following are the delimiters.

(	)	[	]	{	}
,	:	.	'	=	;
+=	-=	*=	/=	//=	%=
&=	=	^=	>>=	<<=	**=

## Literals

Literal is a raw data given to a variable or constant. In Python, there are various types of literals like Numeric, String, Boolean. A Boolean literal can have any of the two values: True or False.

Ex:

```
A=10
```

In the above example 10 is a numeric literal

```
B="Happy"
```

Happy is a string literal

## Escape Sequences

In Python strings, the backslash "`\`" is a special character, also called the "**escape**" character. It is used in representing certain whitespace characters: "`\t`" is a tab, "`\n`" is a newline, and "`\r`" is a carriage return. For example to print the message "It's raining", the Python command is

```
>>> print("It\'s raining")
```

 will give the output as It's raining

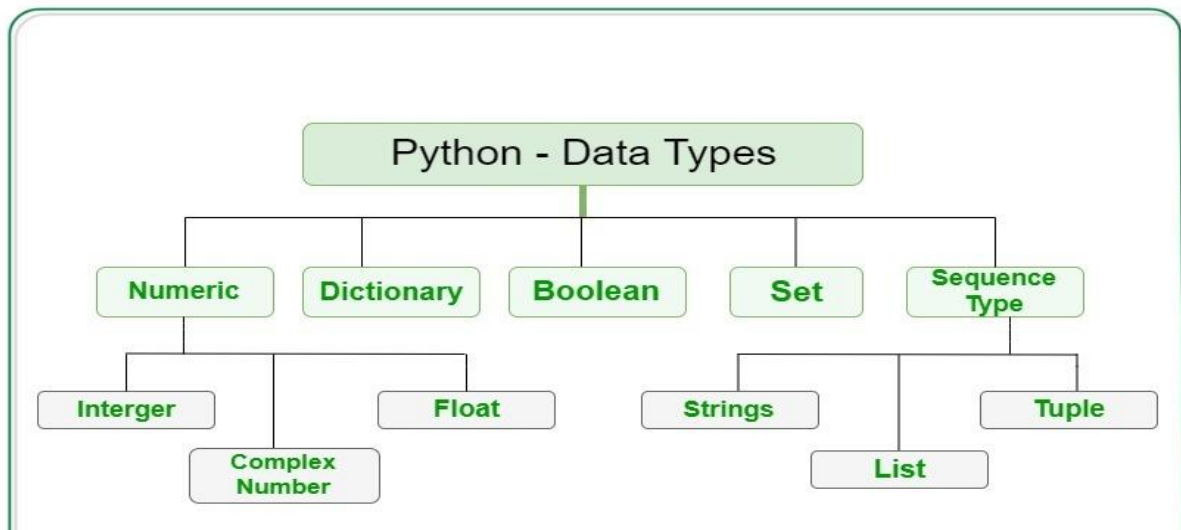
Python supports the following escape sequence characters.

Escape sequence character	Description	Example	Output
<code>\\</code>	Backslash	<code>&gt;&gt;&gt; print("\\test")</code>	<code>\test</code>
<code>\'</code>	Single-quote	<code>&gt;&gt;&gt; print("Doesn\'t")</code>	Doesn't
<code>\"</code>	Double-quote	<code>&gt;&gt;&gt; print("\"Python\"")</code>	"Python"

\n	New line	print("Python","\n","Lang..")	Python Lang..
\t	Tab	print("Python","\t","Lang..")	Python Lang..

## Python Data types

All data values in Python are objects and each object or value has type. Python has Built-in or Fundamental data types such as Number, String, Boolean, tuples, lists, sets and dictionaries etc.



The data types which will be handled here are integer, float and string.

**Integers** – This data type can contain positive or negative whole numbers (without fractions or decimals). In Python there is no limit to how long an integer value can be.

**Float** – It is a real number with a floating point representation. It is specified by a decimal point.

**String** – are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double quote or triple quote.

## Chapter 4

### Control Structures - Decision Control Statements

#### Introduction

Programs may contain set of statements. These statements are the executable segments that yield the result. In general, statements are executed sequentially, that is the statements are executed one after another.

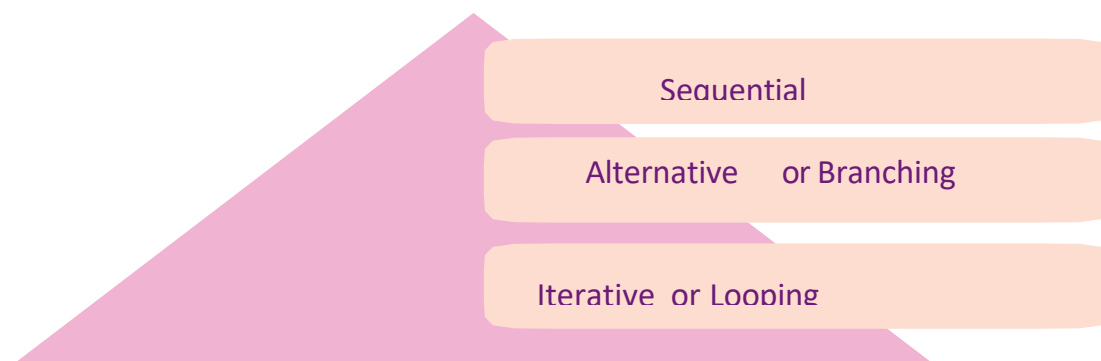
There may be situations in our real life programming where we need to skip a segment or set of statements and execute another segment based on the test of a condition. This is called **alternative** or **branching**.

Also, we may need to execute a set of statements multiple times, called **iteration** or **looping**. In this chapter we are to focus on the various control structures in Python, their syntax and learn how to develop the programs using them.

#### Control Structures

A program statement that causes a jump of control from one part of the program to another is called **control structure** or **control statement**. These control statements are compound statements used to alter the control flow of the process or program depending on the state of the process.

Control Structures can be Sequence , Selection or Iterative statements.



## Sequential Statement

A **sequential statement** is composed of a sequence of statements which are executed one after another. A code to print your name, address and phone number is an example of sequential statement.

### Example

```
#Program to print your name and address-example for sequential statement  
print("Hello!This is Shyam")  
print("43,SecondLane,NorthCarStreet,TN")
```

### Output

```
Hello!This is Shyam  
43,SecondLane,NorthCarStreet,TN
```

## Alternative or Branching statement

In our day-to-day life we need to take various decisions and choose an alternate path to achieve our goal. May be we would have taken an alternate route to reach our destination when we find the usual road by which we travel is blocked. This type of decision making is what we are to learn through alternative or branching statement. Checking whether the given number is positive or negative, even or odd can all be done using alternative or branching statement.

**Python provides the following types of alternative or branching statements:**

Simple if statement

If else statement

If elif statement



## Simple if statement

**Simple** if is the simplest of all decision making statements.

Condition should be in the form of relational or logical expression.

### Syntax:

```
if<condition>:  
    statements-block1
```

In the above syntax if the condition is true statements - block 1 will be executed.

### Example

```
#Program to check the age and print whether eligible for voting x=int  
(input("Enter your age :"))
```

```
If x >=18:
```

```
    print("You are eligible for voting")
```

#### Output1:

```
Enter your age:34
```

```
You are eligible for voting
```

#### Output2:

```
Enter your age: 16
```

```
>>>
```

As you can see in the second execution no output will be printed, only the Python prompt will be displayed because the program does not check the alternative process when the condition fails.

## if... else statement

The **if .. else** statement provides control to check the true block as well as the false block. Following is the syntax of '**if..else**' statement.

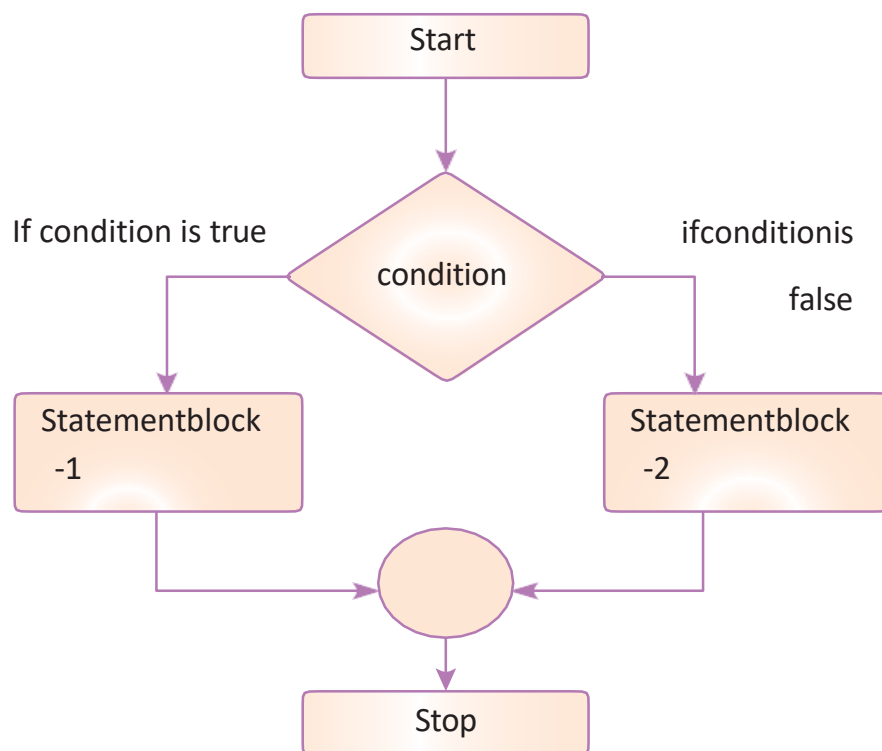
### Syntax:

```
if <condition>:
```

```
    statements-block1
```

```
else:
```

```
    statements-block2
```



### if..elif...else statement:

When we need to construct a chain of **if** statement(s) then '**elif**' clause can be used instead of '**else**'.

#### Syntax:

```
if <condition-1>:  
    statements-block1  
  
elif <condition-2>:  
    statements-block2  
  
else:
```

#### Example :#Program to check if the accepted number is odd or even

```
a=int(input("Enter any number:"))  
if a%2==0:  
    print(a,"is an even number")  
else:  
    print(a,"is an odd number")
```

#### Output1:

```
Enter any number:56  
56 is an even number
```

#### Output2:

```
Enter any number:67  
67 is an odd number
```

In the syntax of **if..elif..else** mentioned above, condition-1 is tested if it is true then statements-block1 is executed, otherwise the control checks condition-2, if it is true statements-block2 is executed and even if it fails statements-block n mentioned in **else** part is executed.

#### Example

Write a program to assign grade as per the following rules

Avg >90 Grade A

Avg between 80 to 90 B

Avg less than 80 C

```
tot= int(input("enter the total of 5 subject marks"))
```

```
avg= tot // 5
```

```
if avg > 90:
```

```
    print("Grade A")
```

```
elif avg >80:
```

```
    print("Grade B")
```

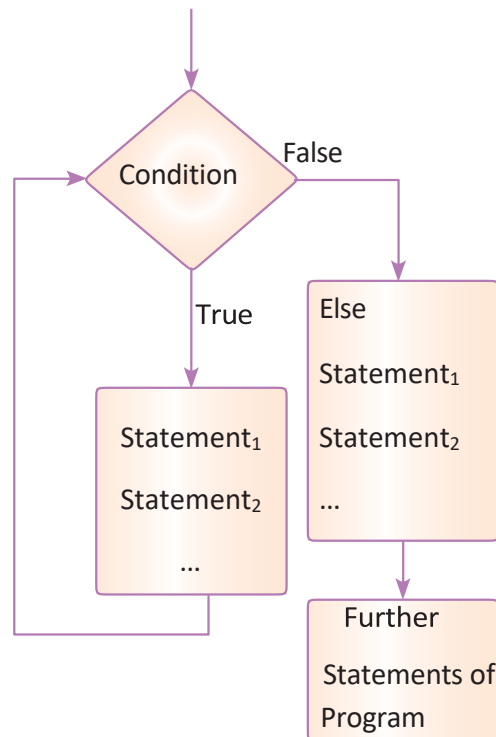
```
else:
```

```
    print("Grade C")
```

## Chapter 5

### Control Structures - Iterative statements

Iteration or loop are used in situation when the user need to execute a block of code several of times or till the condition is satisfied. A **loop** statement allows to execute a statement or group of statements multiple times.



**Python provides two types of looping constructs:**

for loop

while loop

for loop

The **for** loop is the most comfortable loop. It is also an entry check loop. The condition is checked in the beginning and the body of the loop(statements-block 1) is executed if it is only True otherwise the loop is not executed.

## Syntax

```
for counter_variable in range(start ,stop):
```

```
    Statement block 1
```

Example:

```
for i in range(1,10):
```

```
    print(i)
```

# The above code will display 1 to 9 since the stop value is exclusive.

## while loop

In Python, we use a while loop to repeat a block of code until a certain condition is met. For example,

## Syntax

```
while <condition>:
```

```
    body of the while loop
```

## Example

```
v=1
```

```
while v <=4:
```

```
    print(v)
```

```
    v = v + 1
```

output :

1

2

3

4

## PYTHON – LAB ACTIVITIES

1. Write a program to print “Hello World!”  

```
print("Hello World!")
```
2. Write a program that inputs the name of the user and prints it in the given format  
Example Input: Ram  
Output: Namaste Ram, welcome to Python!  

```
name=input("Enter your name: ")  
print("Namaste",name,"welcome to Python!")
```
3. Write a program to input a number and display its square and cube  
# Get input from the user  

```
number = int(input("Enter a number: "))  
  
# Calculate the square and cube  
square = number ** 2  
cube = number ** 3  
  
# Print the results  
print("The square of",number,"is",square)  
print("The cube of",number,"is",cube)
```
4. Write a program to input 2 numbers and perform addition, subtraction and multiplication on them.  
# Get input from the user  

```
num1 = float(input("Enter the first number: "))  
num2 = float(input("Enter the second number: "))  
  
# Perform arithmetic operations  
addition = num1 + num2  
subtraction = num1 - num2  
multiplication = num1 * num2  
  
# Print the results  
print("Addition: ", num1, "+", num2, "=", addition)
```

```
print("Subtraction: ", num1, "-", num2, "=", subtraction)
print("Multiplication: ", num1, "*", num2, "=", multiplication)
```

5. Write a program to get the length and breadth and calculate the area, perimeter of rectangle

```
# Get input from the user
length = float(input("Enter the length of the rectangle: "))
breadth = float(input("Enter the breadth of the rectangle: "))
```

```
# Calculate area and perimeter
area = length * breadth
perimeter = 2 * (length + breadth)
```

```
# Print the results
print("Area of the rectangle: ", area)
print("Perimeter of the rectangle: ", perimeter)
```

6. Write a program to get the base and height of a triangle and computes its area

```
# Get input from the user
base = float(input("Enter the base of the triangle: "))
height = float(input("Enter the height of the triangle: "))
```

```
# Calculate area of the triangle
area = 0.5 * base * height
```

```
# Print the result
print("Area of the triangle: ", area)
```

7. Write a program to get the principal amount, rate of interest, and time period to compute simple interest.

```
# Get input from the user
principal = float(input("Enter the principal amount: "))
rate_of_interest = float(input("Enter the rate of interest: "))
time_period = float(input("Enter the time period (in years): "))
```



```
# Calculate simple interest
simple_interest = (principal * rate_of_interest * time_period) / 100
```

```
# Print the result
print("Simple Interest: ", simple_interest)
```

8. Write a program to get the radius and height and calculate the surface area of the cylinder (7<sup>th</sup>Math)

```
# Get input from the user
radius = float(input("Enter the radius of the cylinder: "))
height = float(input("Enter the height of the cylinder: "))
```

```
# Calculate the surface area of the cylinder
# Surface Area = 2 * π * r * (r + h)
pi = 3.14159 # Approximation of π
surface_area = 2 * pi * radius * (radius + height)
```

```
# Print the result
print("Surface Area of the cylinder: ", surface_area)
```

9. Write a program that reads a number and checks if it is odd or even

```
# Get input from the user
number = int(input("Enter a number: "))
```

```
# Check if the number is even or odd
if number % 2 == 0:
    print(number, "is even.")
else:
    print(number, "is odd.")
```

10. Write a program to input 2 numbers and perform division by checking the non-zero condition.

```
# Get input from the user
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
```

```
# Check for division by zero before performing division
```

```

if num2 != 0:
    division = num1 / num2
else:
    division = "Undefined (cannot divide by zero)"

# Print the results
print("Division: ", num1, "/", num2, "=", division)

```

11. Write a program that reads a number and checks if it is positive, negative, or zero

```

# Get input from the user
number = float(input("Enter a number: "))
# Check if the number is positive, negative, or zero
if number > 0:
    print(number, "is positive.")
elif number < 0:
    print(number, "is negative.")
else:
    print(number, "is zero.")

```

12. Write a program that checks the age of a person to determine if they are eligible to vote, and also checks for invalid age input (ages above 100 and below 0).

```

# Get input from the user
age = int(input("Enter your age: "))
# Check for invalid age
if age < 0 or age > 100:
    print("Invalid age.")
# Check eligibility to vote
elif age >= 18:
    print("Eligible to vote.")
else:
    print("Not eligible to vote.")

```

13. Write a program to solve a quadratic equation of the form  $ax^2+bx+c=0$  and find its roots

```
# Input coefficients a, b, and c
```

```
#2 2 5 - complex roots
```

```
#2 4 2 - Equal roots
```

```
# Get coefficients from the user
```

```
a = float(input("Enter coefficient a: "))
```

```
b = float(input("Enter coefficient b: "))
```

```
c = float(input("Enter coefficient c: "))
```

```
# Calculate the discriminant
```

```
discriminant = (b * b) - (4 * a * c)
```

```
# Check the nature of the roots
```

```
if discriminant > 0:
```

```
    root1 = (-b + (discriminant ** 0.5)) / (2 * a)
```

```
    root2 = (-b - (discriminant ** 0.5)) / (2 * a)
```

```
    print("Roots are real and different.")
```

```
    print("Root 1:", root1)
```

```
    print("Root 2:", root2)
```

```
elif discriminant == 0:
```

```
    root = -b / (2 * a)
```

```
    print("Roots are real and the same.")
```

```
    print("Root:", root)
```

```
else:
```

```
    print("Roots are complex and different.")
```

14. Write a program that read 3 numbers and display the bigger number and no two numbers should be the same.

```
# Get input from the user
```

```
num1 = float(input("Enter the first number: "))
```

```
num2 = float(input("Enter the second number: "))
```

```
num3 = float(input("Enter the third number: "))
```

```
# Check if any two numbers are the same
```

```
if num1 == num2 or num1 == num3 or num2 == num3:
```

```
    print("Error: Please enter three different numbers.")
else:
    # Check which number is the biggest
    if num1 > num2 and num1 > num3:
        print(num1, "is the biggest.")
    elif num2 > num1 and num2 > num3:
        print(num2, "is the biggest.")
    else:
        print(num3, "is the biggest.")
```

15. Write a program to print numbers from 1 to 10

```
# Print numbers from 1 to 10
for i in range(1, 11):
    print(i)
```

16. Write a program to read your name and display it 10 times

```
# Get input from the user
name = input("Enter your name: ")
# Display the name 10 times
for i in range(10):
    print(name)
```

17. Write a program that displays the multiplication table for the first 10 multiples of a given number

```
# Get input from the user
number = int(input("Enter a number: "))
# Display the multiplication table
for i in range(1, 11):
    print(number, "X", i, "=", number * i)
```

18. Write a program that displays the first 10 odd and even numbers

```
# Display the first 10 even numbers
print("First 10 even numbers:")
for i in range(1,10):
    print(i * 2)
```

```
# Display the first 10 odd numbers
print("First 10 odd numbers:")
for i in range(1,10):
    print(i * 2 + 1)
```

19.Code to display the pattern below (Hint:String Multiplication):

```
*
**
***
****
*****
rows=5
for i in range(1, rows + 1):
    print("*" * i)
```

20.Write a python code to find whether the given number is perfect cube

```
# Get input from the user
number = int(input("Enter a number: "))

# Initialize a variable for checking
checkcube = 0

# Check for perfect cube using a for loop
for i in range(number + 1):
    if i **3 == number:
        checkcube = 1

# Display result
if checkcube==1:
    print(number, "is a perfect cube.")
else:
    print(number, "is not a perfect cube.")
```

## Bibliography / References

Textbook of Python Class XII

<https://www.tntextbooks.in/p/12th-books.html>

<https://www.programiz.com/python-programming>

<https://www.w3schools.com/python/>