Python EBook

By @Curious_.programmer

Chapter 1: Introduction

1.1 What Is Python?

Python is a programming language that is commonly used to create websites and applications, automate processes, and do data analysis. Python is a general- purpose programming language, which means it can be used to construct a wide range of applications and isn't specialized for any particular problem. Because of its flexibility and beginner-friendliness, it has become one of the most widely used programming languages today.

Python is a general purpose, dynamic, high-level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is easy to learn yet powerful and versatile scripting language, which makes it attractive for Application Development.
Python's syntax and dynamic typing with its interpreted nature make it an ideal language for scripting and rapid application development.

1.2 Why Python Is Popular?

Python's popularity is due to a variety of factors. Here's a closer look at what makes it so useful to programmers

It features a straightforward syntax that resembles normal English, making it easy to read and comprehend. This speeds up the development of projects as well as the improvement of existing ones.

It's adaptable. Python may be used for a variety of purposes, including web development and machine learning.

It's user-friendly, making it popular among new programmers.

It's free to use and distribute, even for commercial, because it's open source.

The Python module and library archive—bundles of code produced by third-party users to extend Python's capabilities—is large and expanding.

Python has a strong community that adds to the library of modules and libraries and serves as a resource for other programmers. Because of the large support network, finding a solution to a stumbling block is quite simple; someone has almost certainly encountered the same issue before.

1.3 What can be done using Python?

Python is widely used for web and software development, task automation, data analysis, and data visualization. Python has been embraced by many non- programmers, such as accountants and scientists, for a range of common activities, such as arranging money, due to its relative ease of learning.

Data analysis and machine Learning

Python has become a data science standard, allowing data analysts and other professionals to do complicated statistical computations, generate data visualization's, construct machine learning algorithms, manage and analyses data, and perform other data-related activities using the language.

Web development

Python is frequently used to create a website's or application's back end—the bits that the user does not see. Sending data to and from servers, processing data and connecting with databases, URL routing, and maintaining security are all things that Python can help with in web development. For web development, Python has a number of frameworks. Django and Flask are two popular frameworks

Automation or scripting

If you find yourself repeating a job, you may make it more efficient by automating it with Python. Scripting is the process of writing code to create these automated operations. Automation may be used in the coding industry to check for mistakes across many files, convert files, do simple math, and eliminate duplicates in data.

1.4 Why learn Python?

Python provides many useful features to the programmer. These features make it most popular and widely used language. We have listed below few-essential feature of Python.

- Interpreted Language
- Object-Oriented Language
- Open Source Language
- Extensible
- Learn Standard Library
- GUI Programming Support
- Integrated
- Embeddable
- Dynamic Memory Allocation

1.5 Python Installation

1.5.1 Downloading installing Python

1. Go to www.python.org/downloads/
2. Download Python as per your system requirement

Installing python on windows

1. Click on Python Releases for Windows, select the link for the latest Python3 Release – Python 3.x.x

2. Scroll to the bottom and select either Windows x86-64 executable Installer for 64-bit or Windows x86 executable installer for 32-bit

Installing python on Linux

1. Open the Ubuntu Software Center folder
2. Select Developer Tools from the All Software drop-down list box
3. Double-click the Python 3.3.4 entry
4. Click Install
5. Close the Ubuntu Software Center folder

1.6 Python First Program

Unlike the other programming languages, Python provides the facility to execute the code using few lines.

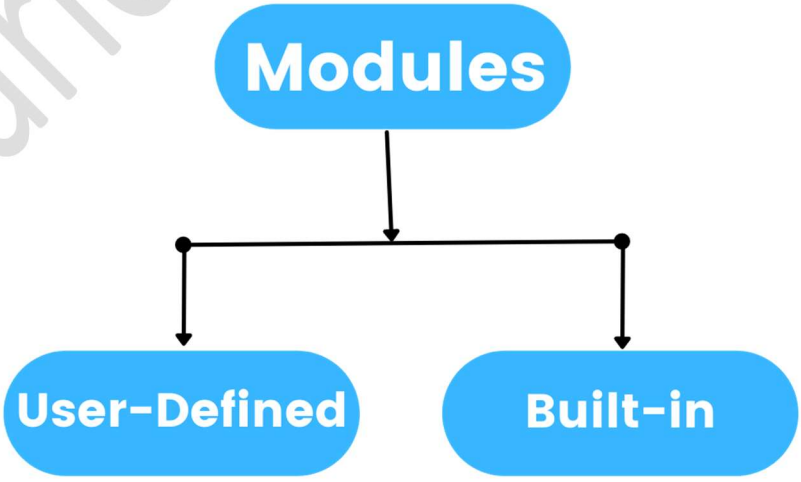
We can do this using one statement in Python.

Chapter 2 :
Modules, Commets,pip

2.1 Modules in Python

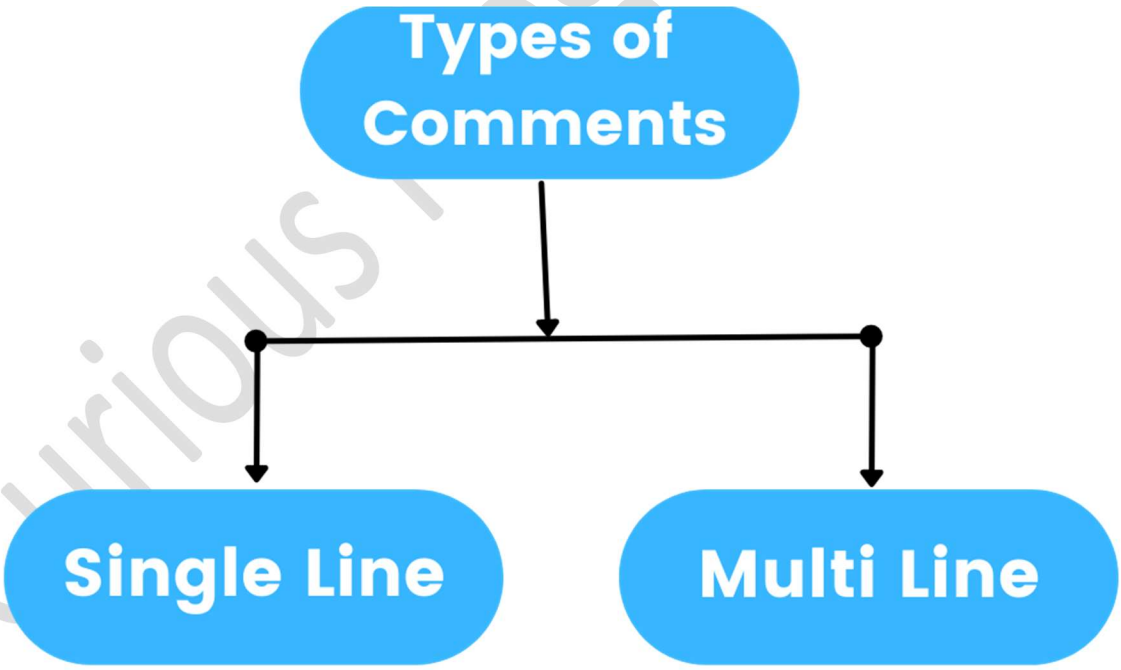
- A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables.

- A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.



2.2 Comments in Python

- Python Comment is an essential tool for the programmers. Comments are generally used to explain the code.
- We can easily understand the code if it has a proper explanation.
- A good programmer must use the comments because in the future anyone wants to modify the code as well as implement the new module; then, it can be done easily.

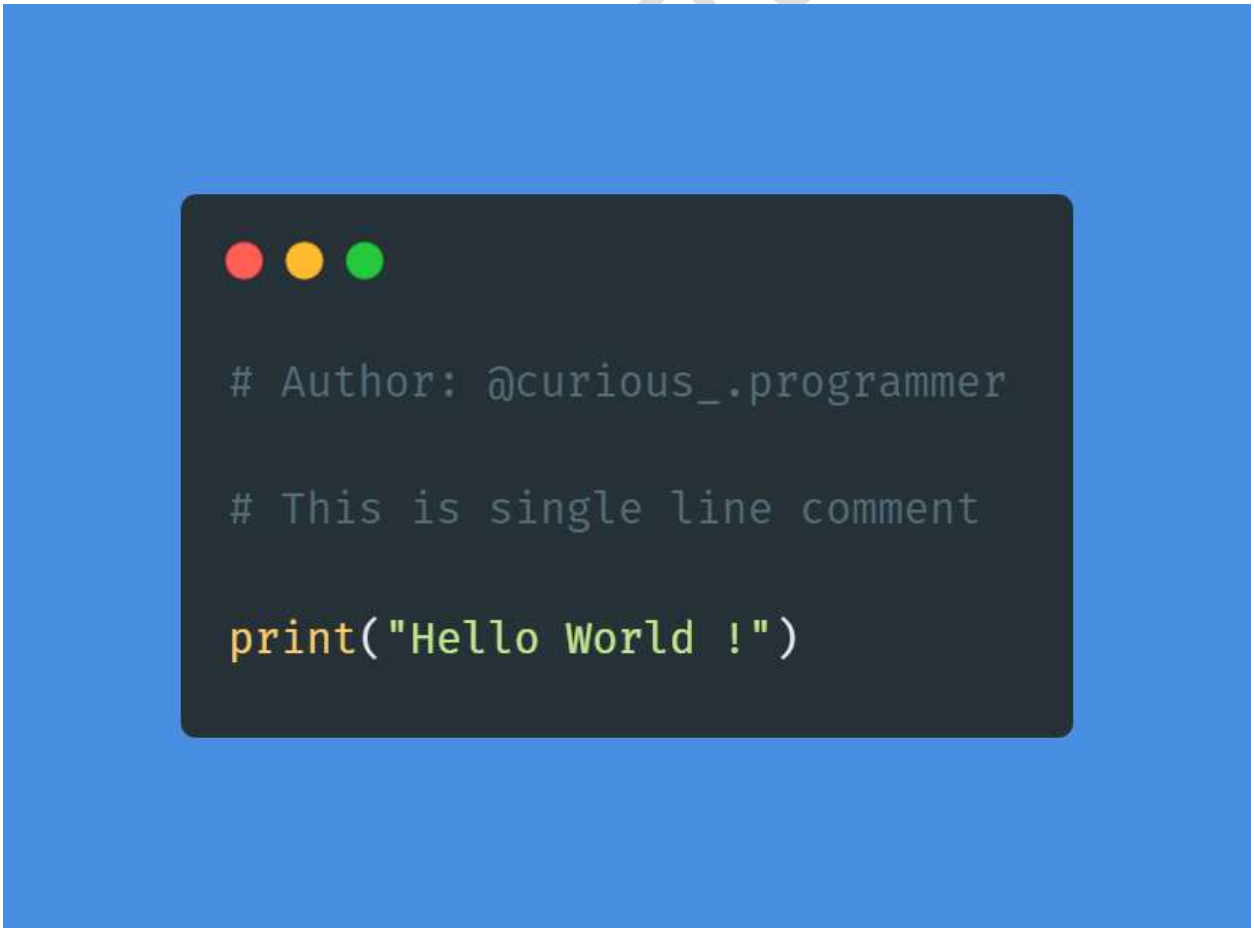


Single line comment

- Python single line comment starts with the hashtag symbol (#) with no white spaces and lasts till the end of the line.

- If the comment exceeds one line then put a hashtag on the next line and continue the comment

- Example:



Multiline Comments

- Python does not provide the option for multiline comments. However, there are different ways through which we can write multiline comments.

- Python ignores the string literals that are not assigned to a variable so we can use these string literals as a comment.

- Example:

2.3 What is pip?

- pip is a package-management system written in Python used to install and manage software packages.

- It connects to an online repository of public packages, called the Python Package Index.

- pip can also be configured to connect to other package repositories, provided that they comply to Python Enhancement Proposal 503.

Chapter 3: Variables, datatypes and Keywords

3.1 Variables in python

- Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

- In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.

- Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

Example:

3.2 Identifier in python

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program.

Rules:

1. The first character of the variable must be an alphabet or underscore (_).
2. All the characters except the first character may be an alphabet of lower-case (a-z), upper-case (A-Z), underscore, or digit (0-9).
3. Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).

4. Identifier name must not be similar to any keyword defined in the language.
5. Examples of valid identifiers: a123, _n, n_9, etc.
6. Examples of invalid identifiers: 1a, n%4, n 9, etc.

3.3 Data Types in python

- Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

```
a = 5
```

- The variable a holds integer value five and we did not define its type. Python interpreter will automatically interpret variables a as an integer type.

- Python enables us to check the type of the variable used in the program. Python provides us the type() function, which returns the type of the variable passed.

Python Data Types:

1) **Numbers:** Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type.
   a) Python supports three types of numeric data-
      i) **Integer -** Integer value can be any length such as integers 10, 2, 29, -20, -150 etc.
      ii) **Float-** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc.
      iii) **Complex number-** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively

2) **Sequence Type:**
      i) **String:** The string can be defined as the sequence of characters represented in the quotation marks. In Python, we can use single, double, or triple quotes to define a string.
      ii) **List:** Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].
      iii) **Tuple:** A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are

separated with a comma (,) and enclosed in parentheses ().

3) **Boolean**: Boolean type provides two built-in values, True and False. These values are used to determine the given statement true or false. It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'

4) **Set:** Python Set is the unordered collection of the data type. It is inerrable, mutable (can modify after creation), and has unique elements. In set, the order of the elements is undefined; it may return the changed sequence of the element.

5) **Dictionary:** Dictionary is an unordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type, whereas value is an arbitrary Python object.

3.4 operator in Python

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Python provides variety of operators, which are described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic operators

Arithmetic operators are used with numeric values to perform common mathematical operations

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators:

Assignment operators are used to assign values to variables:

Operator	**Example**	**Same As**
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3

>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	**Name**	**Example**
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y

is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1

^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

3.5 Input In Python

Python input() Function

Python input() function is used to get input from the user. It prompts for the user input and reads a line. After reading data, it converts it into a string and returns that. It throws an error EOFError if EOF is read.

Parameters

prompt: It is a string message which prompts for the user input

Return

It returns user input after converting into a string.
Let's see some examples of input() function to understand it's functionality.

Python input() Function Example

Input-

```
# Author: @curious_.programmer
# Python Input Function Example

val = input("Enter a value : ")
#Showing Result
print("You Entered: ",val)
```

Output:

```
Enter a value : 4
You Entered:  4
>
```

Chapter 4:
List,Tuples,set,Dictionary and Type Conversion

4.1 list In Python

Lists are just like dynamic sized arrays, declared in other languages (vector in C++ and ArrayList in Java).

Lists need not be homogeneous always which makes it a most powerful tool in Python.

A single list may contain Data Types like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets [].

Characteristics of Lists

The list has the following characteristics:

- The lists are ordered.
- The element of the list can access by index.
- The lists are the mutable type.
- The lists are mutable types.
- A list can store the number of various elements.

A list can be define as below

```
# Author: @curious_.programmer
# A list can be Define as Below

L1 = ["Yadnyesh", 102, "INDIA"]
L2 = [1,2,3,4,5,6,7,8]
```

4.2 Tuples in Python

Python Tuple is used to store the sequence of immutable Python objects.

The tuple is similar to lists since the value of the items stored in the list can be changed, whereas the tuple is immutable, and the value of the items stored in the tuple cannot be changed.

Creating a tuple

A tuple can be written as the collection of comma-separated (,) values enclosed with the small () brackets. The parentheses are optional but it is good practice to use. A tuple can be defined as follows.

Example-

```
# Author: @curious_.programmer
# Tuples in Python

T1 = (101, "Yadnyesh", 19)
t2 = ("Apple", "banana", "cat")

T3 = 10, 20, 30, 40, 50
```

4.3 Set in Python

A Python set is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements. Sets are mutable which means we can modify it after its creation.

Unlike other collections in Python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together, or we can get the list of elements by looping through the set.

Creating a set

The set can be created by enclosing the comma-separated immutable items with the curly braces {}. Python also provides the set() method, which can be used to create the set by the passed sequence.

Example 1: Using curly braces

```
# Author: @curious_.programmer
# Set In python using Curly Braces

Days = {"Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"}

print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

Example 2: Using set() method

```
# Author: @curious_.programmer
# Set In python using Set Method

Days = set(["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"])
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

4.4 Dictionary in Python

Python Dictionary is used to store the data in a key-value pair format. The dictionary is the data type in Python, which can simulate the real-life data arrangement where some specific value exists for some particular key. It is the mutable data-structure. The dictionary is defined into element Keys and values.

- Keys must be a single element
- Value can be any type such as list, tuple, integer, etc.

Creating the dictionary

The dictionary can be created by using multiple key-value pairs enclosed with the curly brackets {}, and each key is separated from its value by the colon (:).The syntax to define the dictionary is given below.

Syntax:

```
Dict = {"Name": "Tom", "Age": 22}
```

In the above dictionary Dict, The keys Name and Age are the string that is an immutable object.

Let's see an example to create a dictionary and print its content.

Example-

```
# Author: @curious_.programmer
# Dictionary in Python

Employee = {"Name": "John", "Age": 29,
"salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
```

Accessing the dictionary values

We have discussed how the data can be accessed in the list and tuple by using the indexing.

However, the values can be accessed in the dictionary by using the keys as keys are unique in the dictionary.

The dictionary values can be accessed in the following way.

Code:

```
Employee = {"Name": "YADNYESH", "Age": 29, "salary":25000}
print(type(Employee))
print("printing Employee data .... ")
print("Name : %s" %Employee["Name"])
print("Age : %d" %Employee["Age"])
print("Salary : %d" %Employee["salary"])
print("Company : %s" %Employee["Company"])
```

Output:

```
<class 'dict'>
printing Employee data ....
Name : YADNYESH
Age : 29
Salary : 25000
>
```

Adding dictionary values

The dictionary is a mutable data type, and its values can be updated by using the specific keys.

The value can be updated along with key Dict[key] = value. The update() method is also used to update an existing value.

Note: If the key-value already present in the dictionary, the value gets updated. Otherwise, the new keys added in the dictionary.

Let's see an example to update the dictionary values.

Programme

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Adding elements to dictionary one at a time
Dict[0] = 'Peter'
Dict[2] = 'Joseph'
Dict[3] = 'Ricky'
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Adding set of values
# with a single Key
# The Emp_ages doesn't exist to dictionary
Dict['Emp_ages'] = 20, 33, 24
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[3] = 'Java'
print("\nUpdated key value: ")
print(Dict)
```

Output-

```
Empty Dictionary:
{}

Dictionary after adding 3 elements:
{0: 'Peter', 2: 'Joseph', 3: 'Ricky'}

Dictionary after adding 3 elements:
{0: 'Peter', 2: 'Joseph', 3: 'Ricky', 'Emp_ages': (20, 33, 24)}

Updated key value:
{0: 'Peter', 2: 'Joseph', 3: 'Java', 'Emp_ages': (20, 33, 24)}
```

4.5 Python Type Casting

Type Casting is the method to convert the variable data type into a certain data type in order to the operation required to be performed by users. In this article, we will see the various technique for typecasting.

There can be two types of Type Casting in Python –

- Implicit Type Casting
- Explicit Type Casting

Implicit Type Casting

In this, methods, Python converts data type into another data type automatically. In this process, users don't have to involve in this process.

Program

```
# Python program to demonstrate

# implicit type Casting

# Python automatically converts

# a to int

a = 7
```

```
print(type(a))


# Python automatically converts

# b to float

b = 3.0

print(type(b))


# Python automatically converts

# c to float as it is a float addition

c = a + b

print(c)

print(type(c))


# Python automatically converts

# d to float as it is a float multiplication

d = a * b

print(d)

print(type(d))
```

Output:

Explicit Type Casting

In this method, Python need user involvement to convert the variable data type into certain data type in order to the operation required.

Mainly in type casting can be done with these data type function:

Int() : Int() function take float or string as an argument and return int type object.

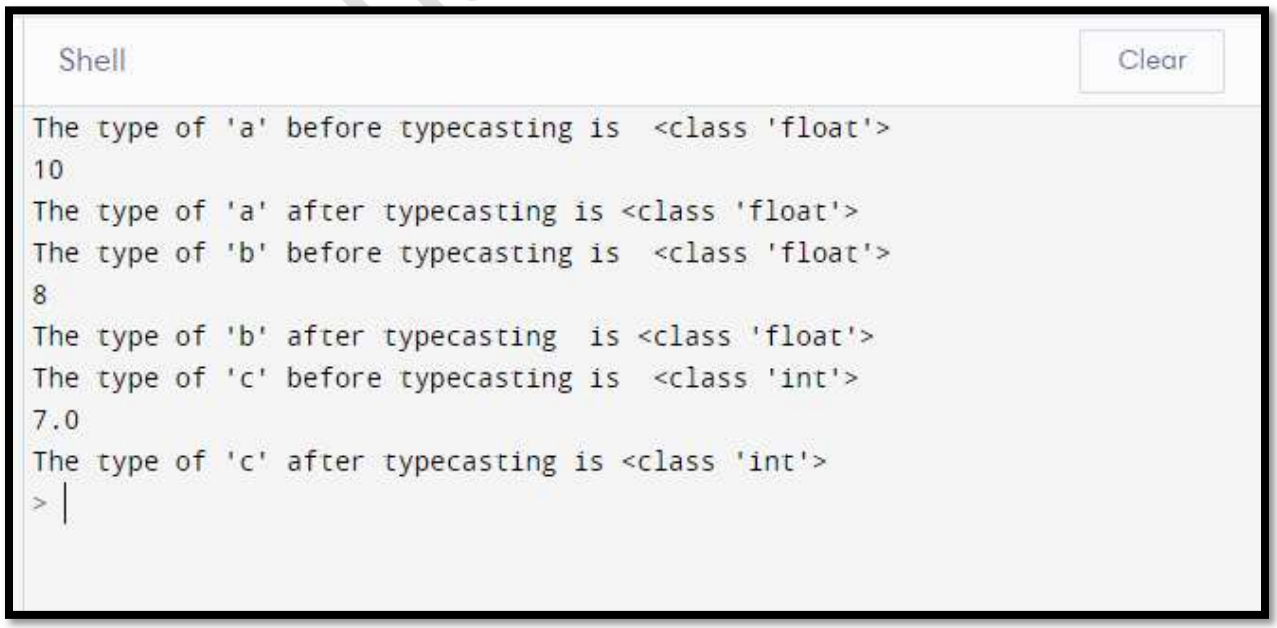
float() : float() function take int or string as an argument and return float type object.

str() : str() function take float or int as an argument and return string type object.

Program

```
#program to demonstrate explicit type conversion
#initializing the value of a
a=10.6
print("The type of 'a' before typecasting is ",type(a))
print(int(a))
print("The type of 'a' after typecasting is",type(a))
#initializing the value of b
b=8.3
print("The type of 'b' before typecasting is ",type(b))
print(int(b))
print("The type of 'b' after typecasting  is",type(b))
#initializing the value of c
c=7
print("The type of 'c' before typecasting is ",type(c))
print(float(c))
print("The type of 'c' after typecasting is",type(c))
```

Output:



Chapter 5: Flow Control

There comes situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code. Decision-making statements in programming languages decide the direction of the flow of program execution

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

Indentation in Python

For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block.

If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python Indentation is the most used part of the python language since it declares the block of code.

All the statements of one block are intended at the same level indentation. We will see how the actual indentation takes place in decision making and other stuff in python.

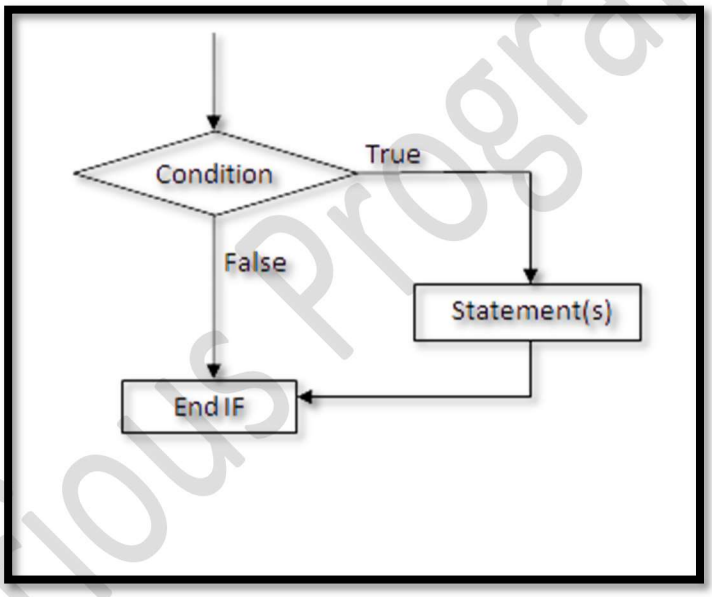
5.1 If Statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

Syntax:

```
# Author @Curious_.programmer
# Syntx of if
if expression:
  satatements
```

Flowchart:

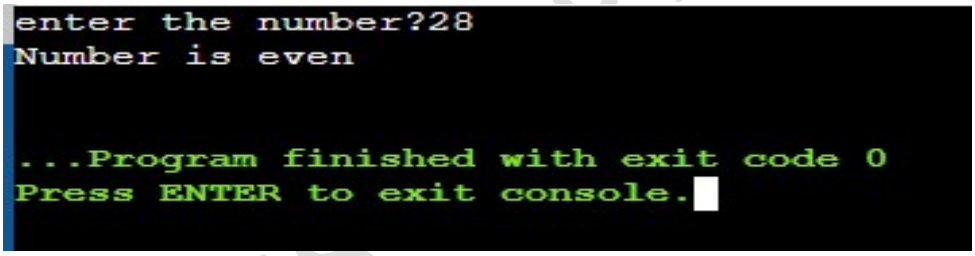


Example:

```
# Author @Curious_.programmer
# Example of if

num = int(input("Enter a Number : "))
if num%2 ==0:
  print("Number is Even
```

Output:



5.2 If-else Statement

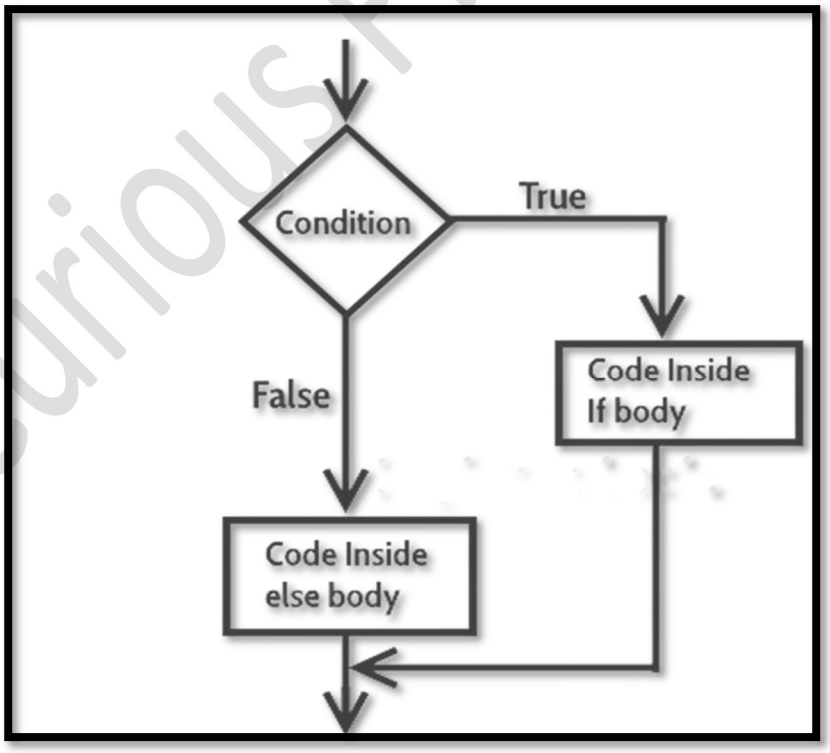
The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

Syntax:

```
# Author @Curious_.programmer
# Syntx of if-else
if expression:
  # Block of code
else:
  # Another Block of Statement (else-Block)
```

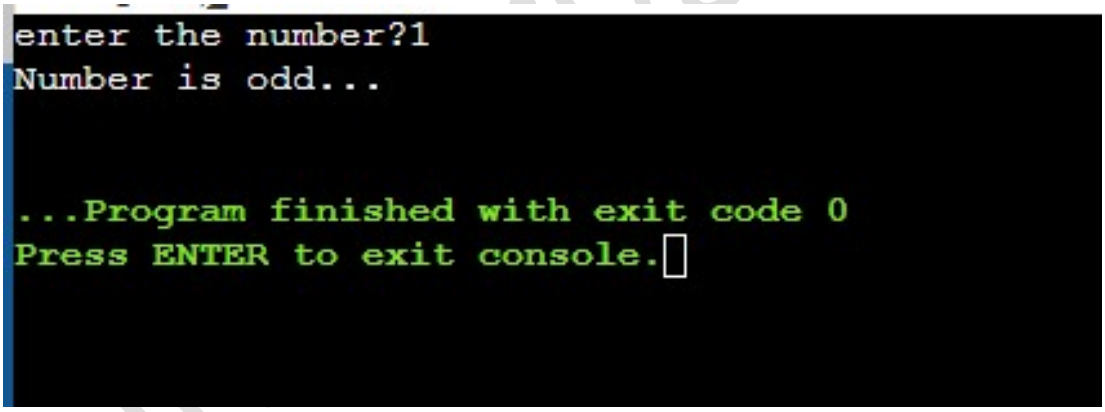
Flowchart:



Program

```python
# Author @Curious_Programmer
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even ... ")
else:
    print("Number is odd ... ")
```

Output:



5.3 elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.

We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

Syntax

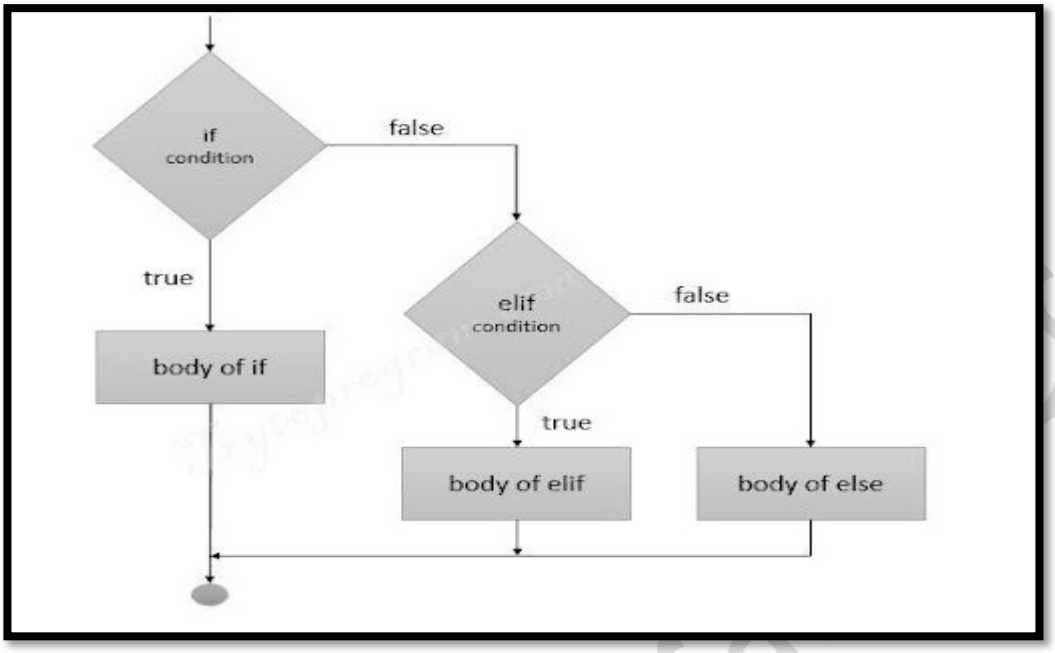
```
# Author @Curious_Programmer
if expression 1:
    # block of statements

elif expression 2:
    # block of statements

elif expression 3:
    # block of statements

else:
    # block of statements
```

Flowchart:



Program

```
# Author @Curious_Programmer
number = int(input("Enter the number?"))
if number==10:
    print("number is equals to 10")
elif number==50:
    print("number is equal to 50");
elif number==100:
    print("number is equal to 100");
else:
    print("number is not equal to 10, 50 or 100");
```

Output

```
Enter the number?10
number is equals to 10


...Program finished with exit code 0
Press ENTER to exit console.
```

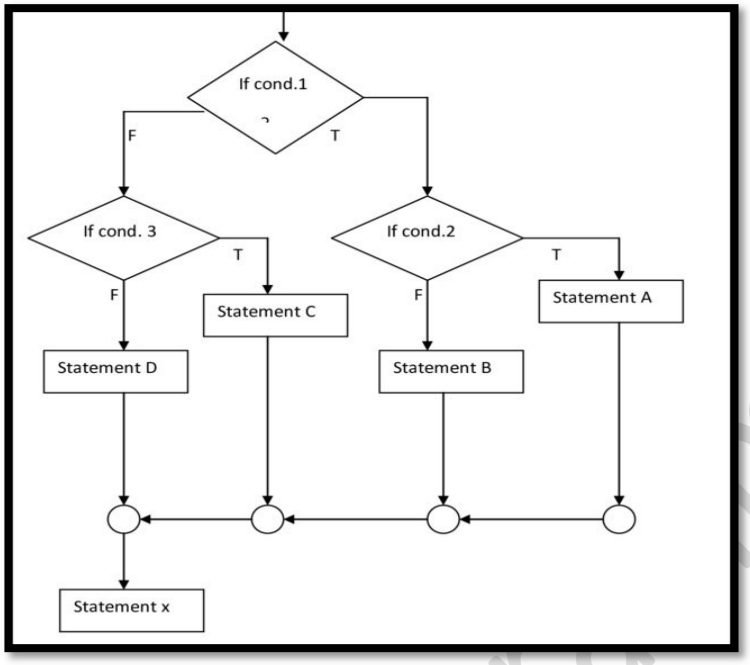
5.4 Nested If

A nested if is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, Python allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
# if Block is end here
```

Flowchart:



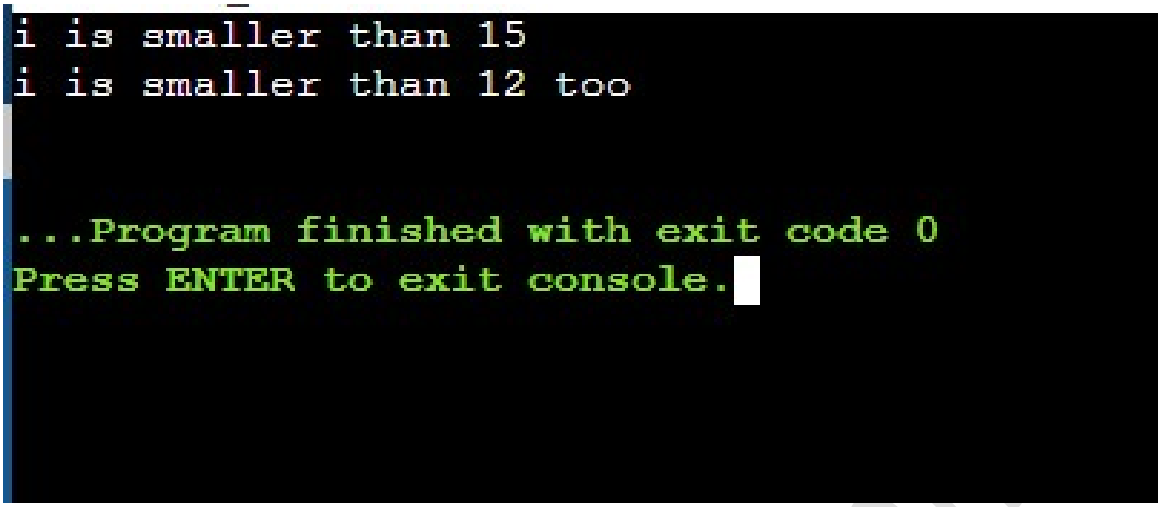
Program

```
# python program to illustrate nested Ifstatement
i = 10
if (i == 10):

    # First if statement
    if (i < 15):
        print("i is smaller than 15")

    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (i < 12):
        print("i is smaller than 12 too")
    else:
        print("i is greater than 15")
```

Output:



Chapter 6: Loops

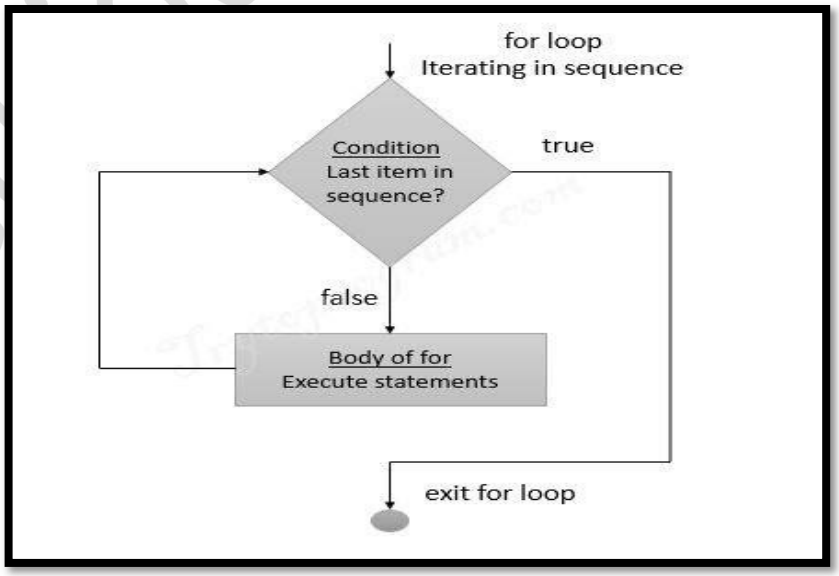
The for loop in Python is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

```
# Author @Curious_Programmer

for iterating_var in sequance:
  statement(S)
```

Flowchart:



For loop Using Sequence

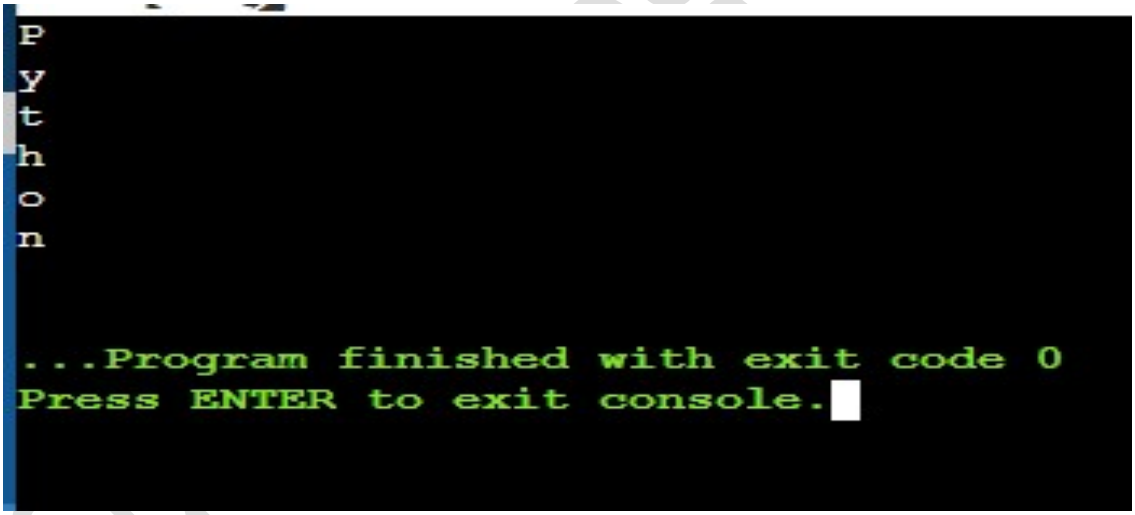
Example-1: Iterating string using for loop

Program

```python
# Author @Curious_Programmer

str = "python"
for i in str:
  print(i)
```

Output



Example- 2: Program to print the table of the given number.

Program:

```
# Author @Curious_Programmer

list = [1,2,3,4,5,6,7,8,9,10]
n = 5
for i in list:
  c = n*i
  print(c)
```

Output

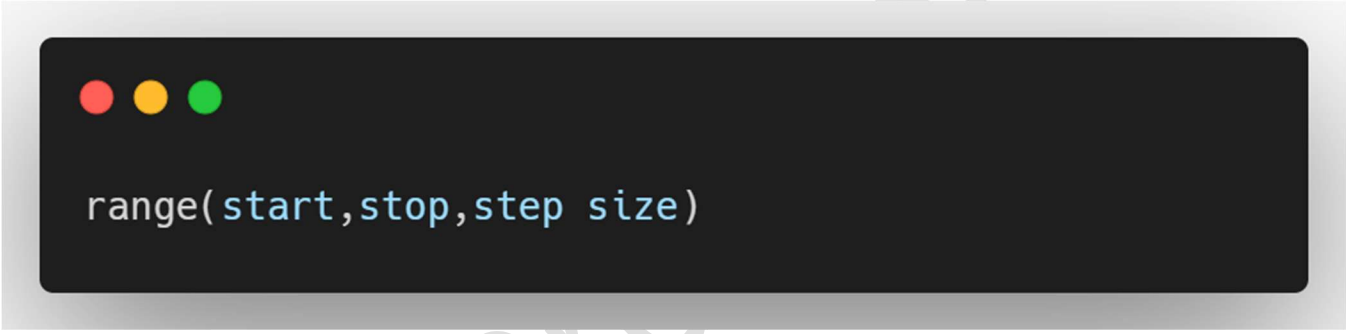
```
5
10
15
20
25
30
35
40
45
50


...Program finished with exit code 0
Press ENTER to exit console.
```

For loop Using range() function

The **range()** function is used to generate the sequence of the numbers. If we pass the range(10), it will generate the numbers from 0 to 9. The syntax of the range() function is given below.

Syntax:



- The start represents the beginning of the iteration.
- The stop represents that the loop will iterate till stop-1. The range(1,5) will generate numbers 1 to 4 iterations. It is optional.
- The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

Example

```
for i in range(10):
    print(i,end = ' ')
```

Output

```
0 1 2 3 4 5 6 7 8 9

...Program finished with exit code 0
Press ENTER to exit console.
```

Nested for loop

Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax is given below.

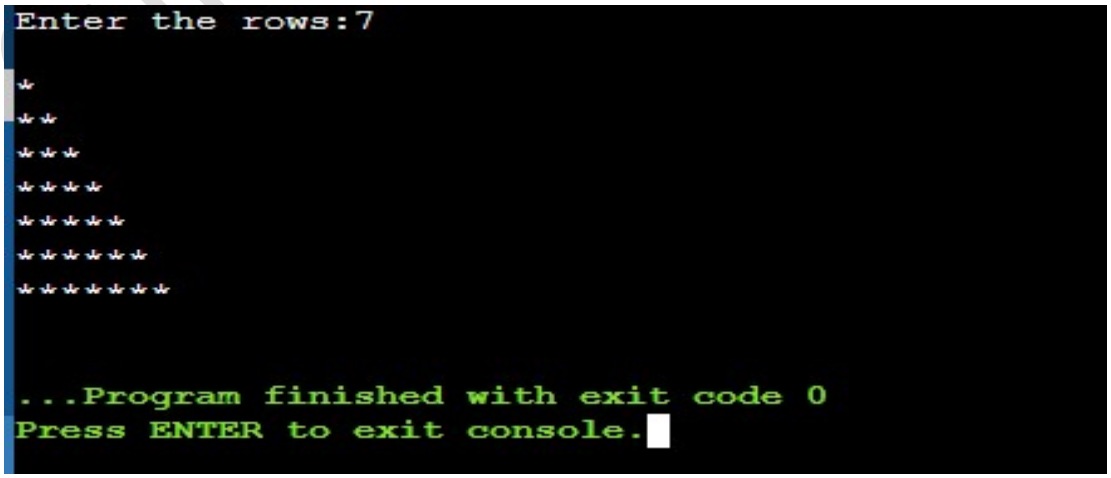
Syntax:

```
for iterating_var1 in sequence:  #outer loop
    for iterating_var2 in sequence:  #inner loop
        #block of statements
#Other statements
```

Example

```
# User input for number of rows
rows = int(input("Enter the rows:"))
# Outer loop will print number of rows
for i in range(0,rows+1):
# Inner loop will print number of Astrisk
    for j in range(i):
        print("*",end = '')
    print()
```

output:



Chapter 7: Strings

In Python, Strings are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.

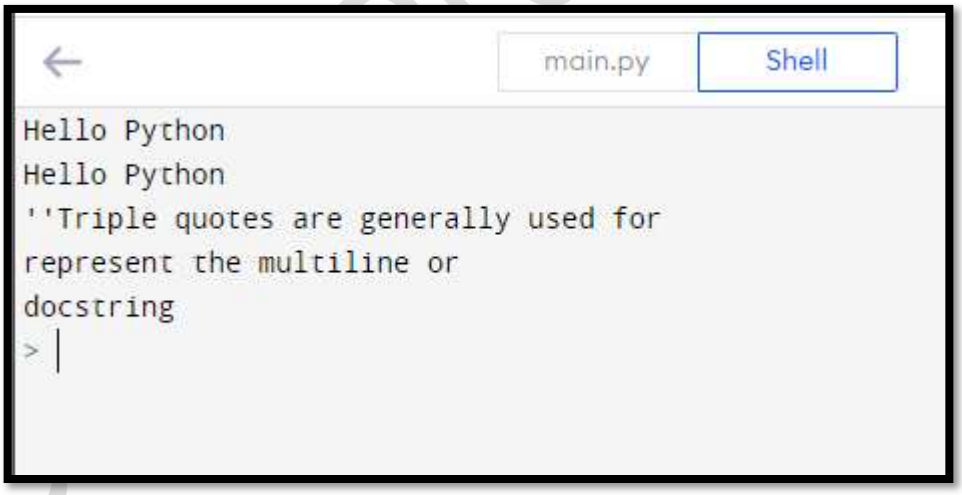
Each character is encoded in the ASCII or Unicode character. So we can say that Python strings are also called the collection of Unicode characters.

7.1 Creating String in Python

```
#Using single quotes
str1 = 'Hello Python'
print(str1)
#Using double quotes
str2 = "Hello Python"
print(str2)

#Using triple quotes
str3 = '''Triple quotes are generally used for
        represent the multiline or
docstring'''
print(str3)
```

Output:



7.2 Python String Formatting

Escape Sequence

Let's suppose we need to write the text as - They said, "Hello what's going on?"- the given statement can be written in single quotes or double quotes but it will raise the **SyntaxError** as it contains both single and double-quotes.

Example:

```
1. str = "They said, "Hello what's going on?""
2. print(str)
```

Output:

```
SyntaxError: invalid syntax
```

We can use the triple quotes to accomplish this problem but Python provides the escape sequence.

The backslash(/) symbol denotes the escape sequence. The backslash can be followed by a special character and it interpreted differently. The single quotes inside the string must be escaped. We can apply the same as in the double quotes.

Example

```
# using triple quotes
print('''''They said, "What's there?"''')

# escaping single quotes
print('They said, "What\'s going on?"')

# escaping double quotes
print("They said, \"What's going on?\"")
```

Output:

```
''They said, "What's there?"
They said, "What's going on?"
They said, "What's going on?"


...Program finished with exit code 0
Press ENTER to exit console.
```

The list of an escape sequence is given below:

Sr.	Escape Sequence	Description	Example

1.	\newline	It ignores the new line.	`print("Python1 \`
			`Python2 \`
			`Python3")`
			Output:
			`Python1 Python2 Python3`

2.	\\	Backslash	`print("\\")`
			Output:
			`\`

3.	\'	Single Quotes	`print('\'')`
			Output:
			`'`

4.	\\''	Double Quotes	`print("\"")`
			Output:
			`"`

5.	\a	ASCII Bell	`print("\a")`

6.	\b	ASCII Backspace(BS)	`print("Hello \b World")`
			Output:
			`Hello World`

7.	\f	ASCII Formfeed	`print("Hello \f World!")` `Hello  World!`

8.	\n	ASCII Linefeed	`print("Hello \n World!")`
			Output:
			`Hello`
			` World!`

9.	\r	ASCII Carriege Return(CR)	`print("Hello \r World!")` **Output:**

			`World!`

10.	\t	ASCII Horizontal Tab	`print("Hello \t World!")`
			Output:
			`Hello  World!`

11.	\v	ASCII Vertical Tab	`print("Hello \v World!")`
			Output:
			`Hello`
			`World!`

12.	\ooo	Character with octal value	`print("\110\145\154\154\157")`
			Output:
			`Hello`

13	\xHH	Character with hex value.	`print("\x48\x65\x6c\x6c\x6f")`
			Output:
			`Hello`

7.3 The format() method

The **format()** method is the most flexible and useful method in formatting strings. The curly braces {} are used as the placeholder in the string and replaced by the **format()** method argument. Let's have a look at the given an example.

```
# Using Curly braces
print("{} and {} both are the best
friend".format("Devansh","Abhishek"))

#Positional Argument
print("{1} and {0} best players
".format("Virat","Rohit"))

#Keyword Argument
print("{a},{b},{c}".format(a = "James", b =
"Peter", c = "Ricky"))
```

Output:

```
Devansh and Abhishek both are the best friend
Rohit and Virat best players
James,Peter,Ricky


...Program finished with exit code 0
Press ENTER to exit console.
```

7.4 Python String functions

Method	Description
capitalize()	It capitalizes the first character of the String. This function is deprecated in python3

casefold()	It returns a version of s suitable for case-less comparisons.
center(width ,fillchar)	It returns a space padded string with the original string centred with equal number of left and right spaces.
count(string,begin,end)	It counts the number of occurrences of a substring in a String between begin and end index.
decode(encoding = 'UTF8', errors = 'strict')	Decodes the string using codec registered for encoding.
encode()	Encode S using the codec registered for encoding. Default encoding is 'utf-8'.
endswith(suffix ,begin=0,end=len(string))	It returns a Boolean value if the string terminates with given suffix between begin and end.
expandtabs(tabsize = 8)	It defines tabs in string to multiple spaces. The default space value is 8.
find(substring ,beginIndex, endIndex)	It returns the index value of the string where substring is found between begin index and end index.

format(value)	It returns a formatted version of S, using the passed value.
index(subsring, beginIndex, endIndex)	It throws an exception if string is not found. It works same as find() method.
isalnum()	It returns true if the characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise, it returns false.
isalpha()	It returns true if all the characters are alphabets and there is at least one character, otherwise False.
isdecimal()	It returns true if all the characters of the string are decimals.
isdigit()	It returns true if all the characters are digits and there is at least one character, otherwise False.
isidentifier()	It returns true if the string is the valid identifier.

Chapter 8: Functions

Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code, which can be called whenever required.

Python allows us to divide a large program into the basic building blocks known as a function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the Python program.

The Function helps to programmer to break the program into the smaller part. It organizes the code very effectively and avoids the repetition of the code. As the program grows, function makes the program more organized.

Python provide us various inbuilt functions like range() or print(). Although, the user can create its functions, which can be called user-defined functions.
There are mainly two types of functions.

1. **User-define functions** - The user-defined functions are those define by the user to perform the specific task.
2. **Built-in functions** - The built-in functions are those functions that are pre-defined in Python.

Advantage of Functions in Python

- There are the following advantages of Python functions.
- Using functions, we can avoid rewriting the same logic/code again and again in a program.
- We can call Python functions multiple times in a program and anywhere in a program.
- We can track a large Python program easily when it is divided into multiple functions.
- Reusability is the main achievement of Python functions.
- However, Function calling is always overhead in a Python program.

8.1 Creating a Function

```
# Author @Curious_Programmer

def my_function(parametres):
  function_block
return expression
```

- Let's understand the syntax of functions definition.
- The def keyword, along with the function name is used to define the function.
- The identifier rule must follow the function name.
- A function accepts the parameter (argument), and they can be optional.
- The function block is started with the colon (:), and block statements must be at the same indentation.
- The return statement is used to return the value. A function can have only one return

8.2 Function Calling

In Python, after the function is created, we can call it from another function. A function must be defined before the function call; otherwise, the Python interpreter gives an error. To call the function, use the function name followed by the parentheses.

Example:

```
#function definition
def hello_world():
    print("hello world")
# function calling
hello_world()
```

8.3 The return statement

The return statement is used at the end of the function and returns the result of the function. It terminates the function execution and transfers the result where the function is called. The return statement cannot be used outside of the function

Syntax:

return [expression_list]

Example:

```
# Author @Curious_Programmer

def hello_world():
  print("hello World")
# function calling
hello_world()
```

8.4 Arguments in function

The arguments are types of information which can be passed into the function. The arguments are specified in the parentheses. We can pass any number of arguments, but they must be separate them with a comma.

Consider the following example, which contains a function that accepts a string as the argument.

Example

```
1. #defining the function
2. def func (name):
3.     print("Hi ",name)
4. #calling the function
5. func("yash")
```

types of argument

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

8.5 Scope of variables

The scopes of the variables depend upon the location where the variable is being declared. The variable declared in one part of the program may not be accessible to the other parts.

In python, the variables are defined with the two types of scopes.

1. Global variables
2. Local variables

The variable defined outside any function is known to have a global scope, whereas the variable defined inside a function is known to have a local scope.

Example 1 Local Variable

```
def print_message():
    message = "hello !! I am going to print a message." # the
variable message is local to the function itself
    print(message)
print_message()
print(message) # this will cause an error since a local
variable cannot be accessible here.
```

Output:

```
hello !! I am going to print a message.
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    print(message) # this will cause an error since a local variable cannot be accessible here.
NameError: name 'message' is not defined


...Program finished with exit code 0
Press ENTER to exit console.
```

Example 2 Global Variable

```python
def calculate(*args):
    sum=0
    for arg in args:
        sum = sum +arg
    print("The sum is",sum)
sum=0
calculate(10,20,30) #60 will be printed as the sum
print("Value of sum outside the function:",sum) # 0 will be
printed  Output:
```

Output:

```
The sum is 60
Value of sum outside the function: 0


...Program finished with exit code 0
Press ENTER to exit console.
```