PYTHON PROGRAMMING

Table of Contents

4 CONDITIONAL STATEMENTS

- Conditional Statements
- Simple if Statement
- if..else Statements
- if...elif...else
- Short Hand if

5 LOOPING CONSTRUCTS

- Looping Constructs
- for Loop
- range( ) Function
- Using else statement with for loop
- While loop
- Using else statement with While Loop

6 NESTED LOOPS

- Nested For Loops
- break
- continue
- pass

7 PYTHON ARRAYS

- Arrays
- How to use array in a program
- NumPy
- Basic operations
- Matrix Multiplication

8 PYTHON STRINGS

- Strings
- len( ), replace( ), del( ), upper( )
- String Operators
- Concatenation
- String Repetition Operator
- String Slicing
- Sample Programs
- Built-in String functions

9 PYTHON FUNCTIONS

- **Functions**
- **Advantages of Functions**
- **Types of Functions**
- **User Defined Functions**
- **Return Statement**
- **Lambda Functions**
- **Recursive Functions**
- **Built-in Functions**

10 PYTHON LISTS

- **Introduction to Lists**
- **List methods in Python**
- **Nested List**

11 PYTHON TUPLES

- **Tuples**
- **Pack Unpack**
- **Tuples method**

12 PYTHON SETS AND DICTIONARY

- **Sets**
- **Set Operations**
- **Dictionary – Key Value Pairs**

13 PYTHON CLASSES AND OBJECTS

- **Classes and Objects**
- **Creating Classes- Variables , Methods**

14 PYTHON MODULES AND PACKAGES

- **Modules**
- **Library**
- **Important Packages**

15 PYTHON DATE TIME MODULE

- **Date time module**
- **Various date time format**

MODULE 1

INTRODUCTION TO PYTHON

HISTORY OF PYTHON

Python is a general purpose programming language created by Guido van Rossum at CWI (CentrumWiskunde& Informatica) which is a National Research Institute for Mathematics and Computer Science in Netherland during 1991.

Python got its name from a BBC comedy series – "Monty Python's Flying Circus".

Guido Van Rossum

Born	**31 January 1956**
Nationality	**Dutch**
Occupation	**Computer programmer, author**
Awards	**Advancement of Free Software (2001)**

KEY FEATURES OF PYTHON

- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- Python is a powerful language. Python is easy to learn and understand.
- Python works on different platforms (Windows, Mac, Linux etc).
- Python has a simple syntax similar to the English language.Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python supports automatic garbage collections.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written.
- Python can be treated in a procedural way or an object-oriented way
- It can be easily integrated with C, C++, Java

REAL-WORLD APPLICATIONS OF PYTHON

- Web Development
- Game Development
- Scientific and Numeric Applications
- Data Visualization
- Machine Learning
- Artificial Intelligence
- Embedded Applications
- Data Analysis
- App Development

SYSTEM REQUIREMENTS FOR INSTALLING PYTHON

Operating Systems and CPU architecture:

* Windows 7 or 10

* Mac OS X 10.11 or higher, 64-bit

* Linux: RHEL 6/7, 64-bit (almost all libraries also work in Ubuntu)

RAM and free disk space:

4 GB RAM

5 GB free disk space

Google colab

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.

PyCharm is a cross-platform IDE that provides consistent experience on the Windows, macOS, and Linux operating systems.

PyCharm is available in three editions: **Professional, Community, and Edu.**
The Community and Edu editions are open-source projects and they are free, but they have fewer feature

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.

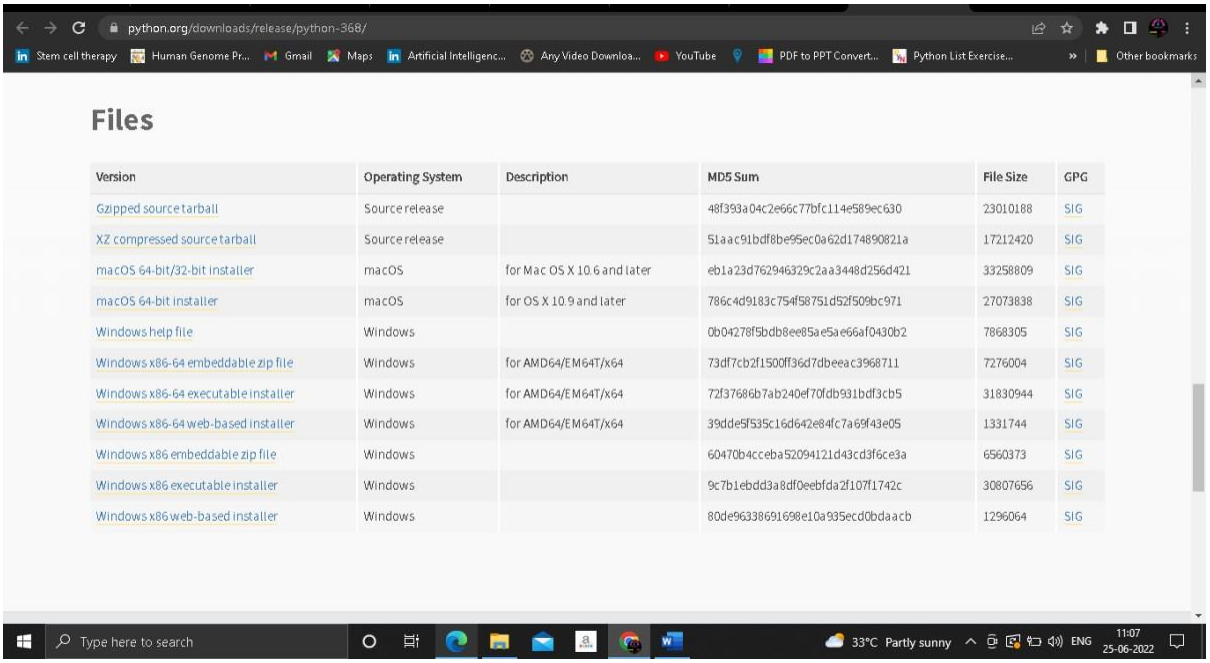
PROCEDURE TO INSTALL PYTHON 3 ON WINDOWS OS:

Install Python 3.6:

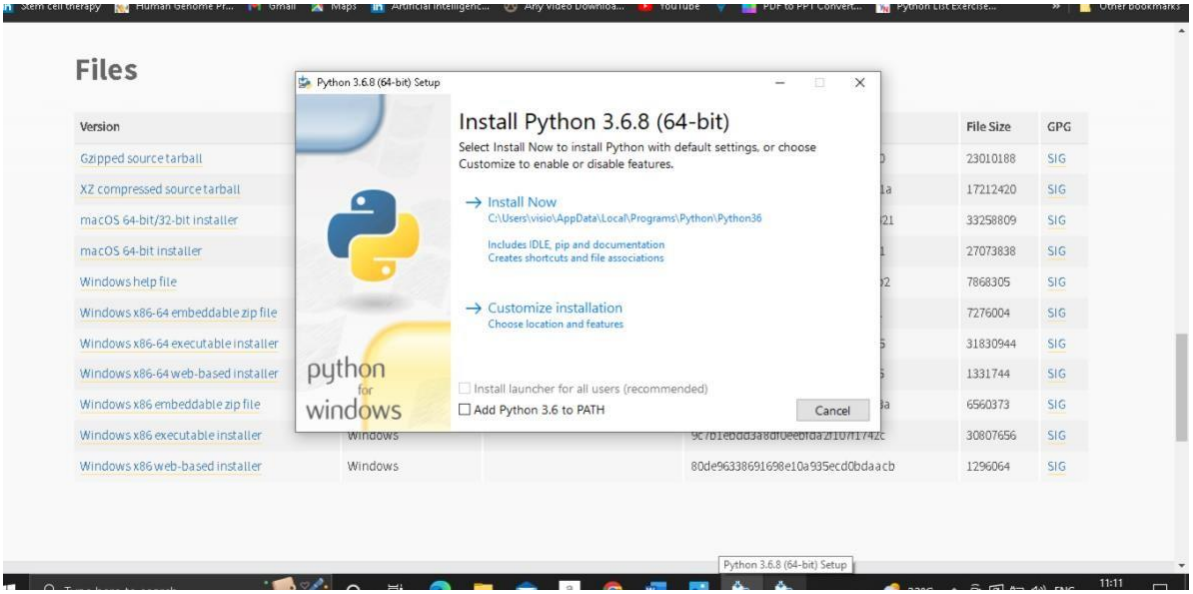
1. To follow the installation procedure, you need to be connected to the Internet.

2. Visit https://www.python.org/downloads/release/python-368/

3. At the bottom locate Windows x86-64 executable installer for 64 bits OS and Windows x86 executable installer for 32 bits OS



4. Click on the located installer file to download.



Note: After double clicking the installer, check mark the option “Add Python 3.6 to PATH”

5. After download completes, double click on the installer file to start the installation procedure.

6. Follow the instructions as per the installer

SCRIPT MODE VS INTERACTIVE MODE

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. Programs written in script mode can be used later.

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.

helloworld.py

```
print("Hello, World!")
```

COMMENTS

The comments make the source code easy to understand. It can also be used to prevent Python from executing code:

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

```
#This is a comment
print("Welcome to python")
```

The multiline comments should be enclosed within a set of ''' '''(triple quotes) which contains more than one line

```
''' This is a comment
print("Welcome to python") '''
```

This will not be executed by Python Interpreter

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 spaces.

MODULE 2

BASICS OF PYTHON

PYTHON CHARACTER SET

Character set is the set of valid characters that a language can recognize. A character represents any letter, digit or any other symbol.

Python has the following character sets:

LETTERS	A-Z, a-z
DIGITS	0-9
SPECIAL SYMBOLS	space,+,-,*,/,**,\, (),{},[],//,==,!=,<,>,:,;,% and etc
WHITESPACES	Blankspace,tabs,carriage return,newline, formfeed
OTHER CHARACTERS	Python can process all ASCII and Unicode characters

INPUT( ) AND 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.

input( ) functions:

The syntax for input( ) function is,

Variable = input ("prompt string")

Where prompt string is the message to the user

Sample code

```
>>>name= input("Enter your name : ")
Enter your name  : Radha
```

```
>>> name=input("Enter your name : ")
    Enter your name : Radha
```

Simple input( ) function without prompt is also available.

The input ( ) accepts all data as string or characters but 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.

```
s =int(input("Enter any number"))

Enter any number12
```

print( ) functions

To print or display output, Python provides print( ) function.

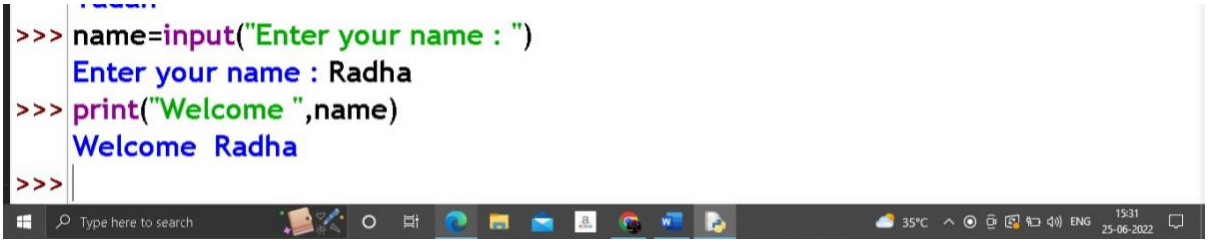
Sample code

```
print ("Hello World!")
```

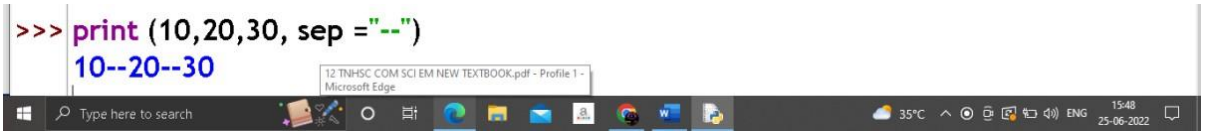
output

```
Hello World!
```

Sample code



The **sep** parameter is primarily used to format the strings that need to be printed on the console and add a separator between strings to be printed. This feature was newly introduced in Python 3.x version.




```
print(10,20,30,sep='%')
output
10%20%30
```

The end parameter is primarily used to append any string at the end of the output of the print statement in python.

```
print(100,200,300,end="$")
output
100 200 300$
```

TOKENS

A token is the smallest individual unit in a python program. All statements and instructions in a program are built with tokens.

Python has the following tokens:

(i) Keyword

(ii) Identifiers

(iii) Literals

(iv) Operators and

(v) Punctuators

Keywords

Keywords are the reserved words in Python. We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language. In Python, keywords are case sensitive.

Few keywords are given below.

Elif	**Del**	**True**	**False**	**Except**
Global	For	If	While	import
And	Try	Or	Return	Pass
Class	In	Not	Is	lambda

Identifiers :

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

• 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

num, total_marks, num1

Example of invalid identifiers

12num, name$, total-mark, pass

LITERALS:

Literal is a raw data given to a variable or constant. In Python, there are various types of literals

String literals:

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes to create a string.

Example:

"python programming…"

Numeric Literals:

Numeric literals can belong to 3 different numerical types

** Integer,

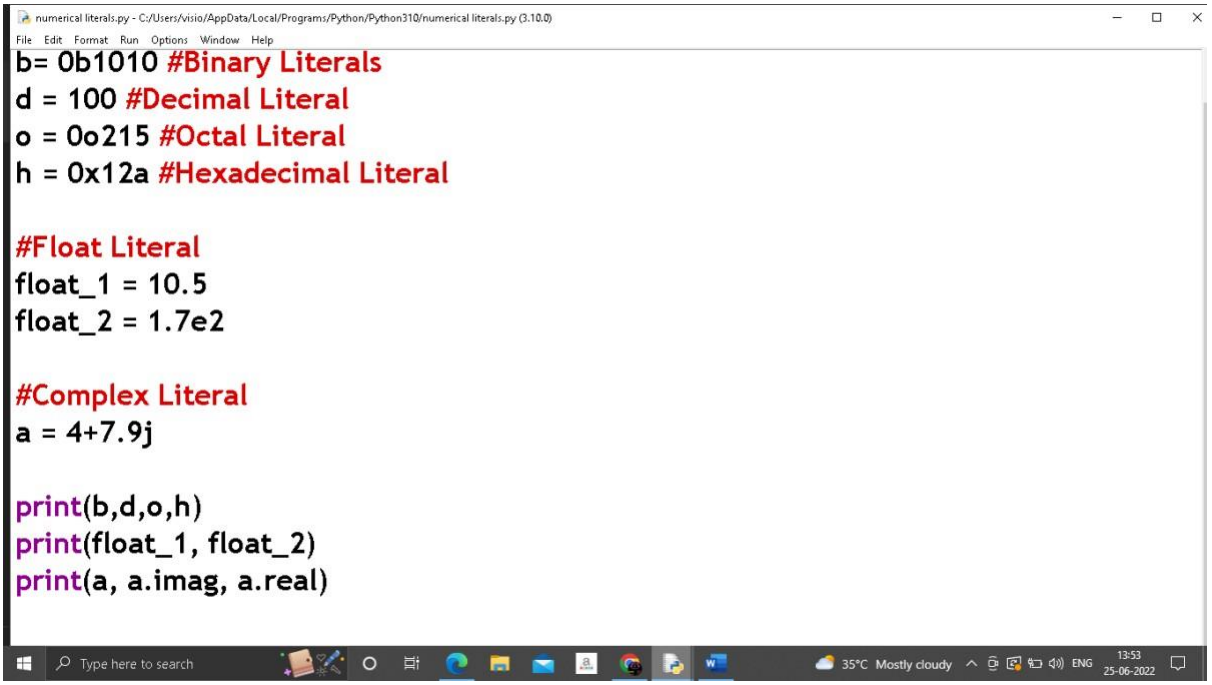
** Float and

** Complex Numbers

Sample code

 # Program to demonstrate Numeric Literals

Execute it and give output



Punctuators

Punctuators are the symbols with specific meanings.Punctuators are used in python to organize the structures, statements and expressions.

Some of the Punctuators are: [ ] { } ( ) -= += *= //= **= /=

Escape Sequence

Python allows to have certain non-graphic characters in string values. Such characters cannot be typed directly from keyboard. e.g. backspace, tabs, carriage return. These can be represented by escape sequence. An escape sequence is represented by backslash followed by one or more characters. Following are the list of escape sequence:

\" - Double quotes

\' - Single quotes

\\ - Backslash

\b - Backspace

\n - New line

\r - Carriage return

\t - Horizontal tab

\v - Vertical tab

Boolean literals

Used to represent one of the two boolean values, that is True and False.

For example:

```
>>>bool=(6>10)
>>>print (bool)
        False
```

MODULE 3

OPERATORS

OPERATORS

Operators are used to perform operations on variables and values.

TYPES OF OPERATORS

Various types of operators available in Python are

- Arithmetic Operators
- Relational or Comparative Operators
- Logical Operators
- Assignment Operators
- Conditional Operators
- Identity Operators
- Membership Operators
- Bitwise Operators

Precedence of Operators :

The precedence of operators determines which operator is executed first if there is more than one operator in an expression.

Precedence:

P – Parentheses

E– Exponentiation

M – Multiplication (Multiplication and division have the same precedence)

D – Division

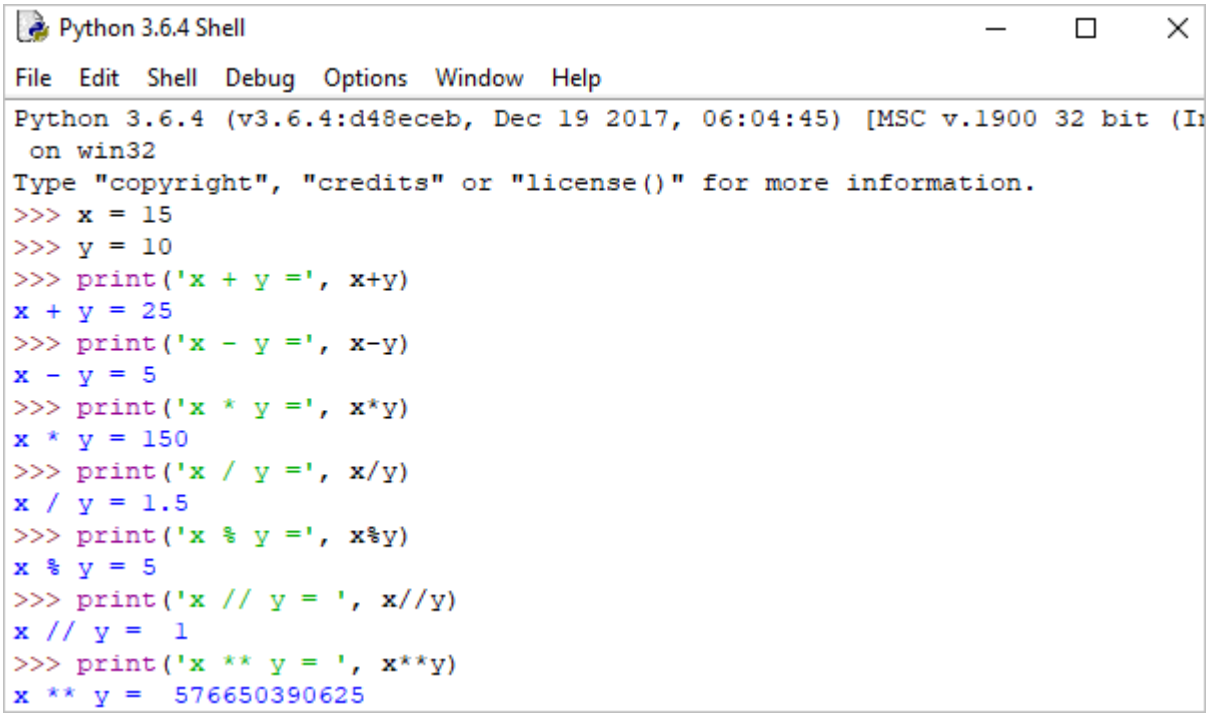
A – Addition (Addition and subtraction have the same precedence)

S – Subtraction

Arithmetic Operators :

Arithmetic Operators are used to perform mathematical calculations like addition, subtraction, multiplications , divisions. Operator is the symbol for calculations and operand is the value on which the operator acts.

Operator	Description
+ (Addition)	It is used to add two operands. a = 20, b = 20 a+b = 40
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value results negative. a = 30, b = 20 a - b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. a = 40, b = 20 a/b = 2.0
*** (Multiplication)**	It is used to multiply one operand with the other. a = 20, b = 10 a * b = 200
% (remainder) Modulo	It returns the remainder after dividing the first operand by the second operand. a = 200, b = 100 a%b = 0
**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to the second operand.
//(Floor division)	It gives the quotient produced by dividing the two operands. Integer division



TRY IT YOURSELF

```
x = 15
y = 3
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x%y)
print(x//y)
print(x**y)
```

Relational Operators or Comparative operators

A Relational operator is also called as Comparative operator which checks the relationship between two operands.

```
>Greater than:
returns True if the left operand is greater than the right
x> y

<Less than:
 returns True if the left operand is less than the right
 x< y

==Equal to:
 returns True if both operands are equal
 x == y

!=Not equal to
 returns True if operands are not equal
  x != y

>=Greater than or equal to:
 returns True if left operand is greater than or equal to the right
 x >= y

<=Less than or equal to:
 returnsTrue if left operand is less than or equal to the right
 x <= y
```

Find output for the following code snippets

```
x = 5
y = 8
print("x == y:", x == y)
print("x != y:", x != y)
print("x < y:", x < y)
print("x > y:", x > y)
```

```
print("x <= y:", x <= y)
print("x >= y:", x >= y
```

```
output
x == y: False
x != y: True
x < y: True
x > y: False
x <= y: True
x >= y: False
```

TRY IT YOURSELF

```
print(20.2 )
print(20.2 < 21.3)
print(20.2 >21.3< 22.4)
print(20.2 < 21.3 < 22.4 < 23.5)
```

Logical Operators

There are three logical operators that are used to compare values. They evaluate expressions down to Boolean values, returning either True or False. These operators are and, or, and not

Operator	What it means	Example
and	True if both are true	x and y
or	True if at least one is true	x or y
not	True only if false	not x

Program code

```
x = 15
print(x > 3 and x < 20)
print(x > 3 or x < 4)
print(not(x > 3 and x < 20))
```

output

```
True
True
False
```

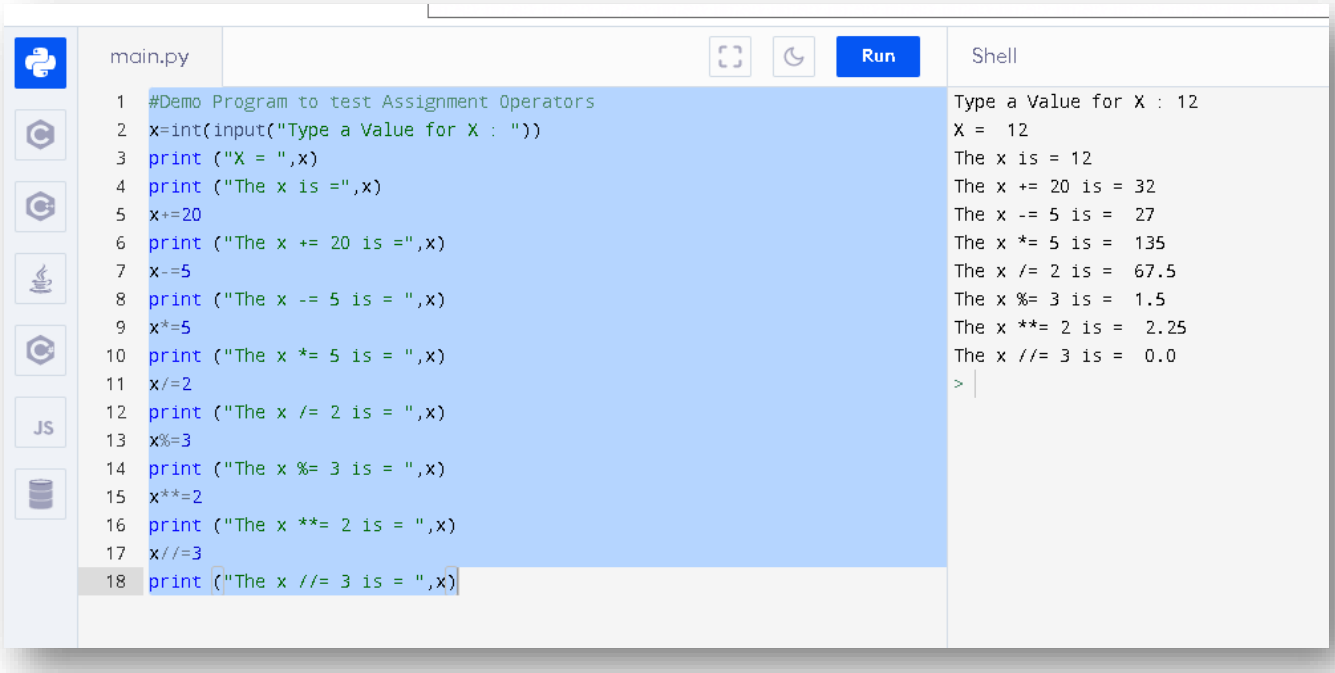
Assignment operator

In Python, = is a simple assignment operator to assign values to variable. There are various compound operators in Python like +=, -=, *=, /=, %=, **= and //=

Operators	Descriptions
=	Assigns values from right side operand to left side operand.
+=	Adds right side operand to the left side operand and assign the result to left side operand.
-=	Subtracts right side operand to the left side operand and assign the result to left side operand.
*=	multiplies right side operand to the left side operand and assign the result to left side operand.
/=	divides right side operand to the left side operand and assign the quotient to left side operand.
%=	divides right side operand to the left side operand and assign remainder the result to left side operand.
**=	Performs exponential (power) calculation on operators and assign value to the left operand
//=	Perform floor division on operators and assign value to the left operand

Find output

#Demo Program to test Assignment Operators



CONDITIONAL OPERATORS

Ternary operator is also known as conditional operator that evaluate something based on a condition being true or false. It simply allows testing a condition in a single line replacing the multiline if-else making the code compact.

Syntax

Variable Name = [on_true] if [Test expression] else [on_false]

Code for finding maximum among 2 numbers using conditional operators

```
a=100
b=50
large=a if a>b else b
print(large)
```

output

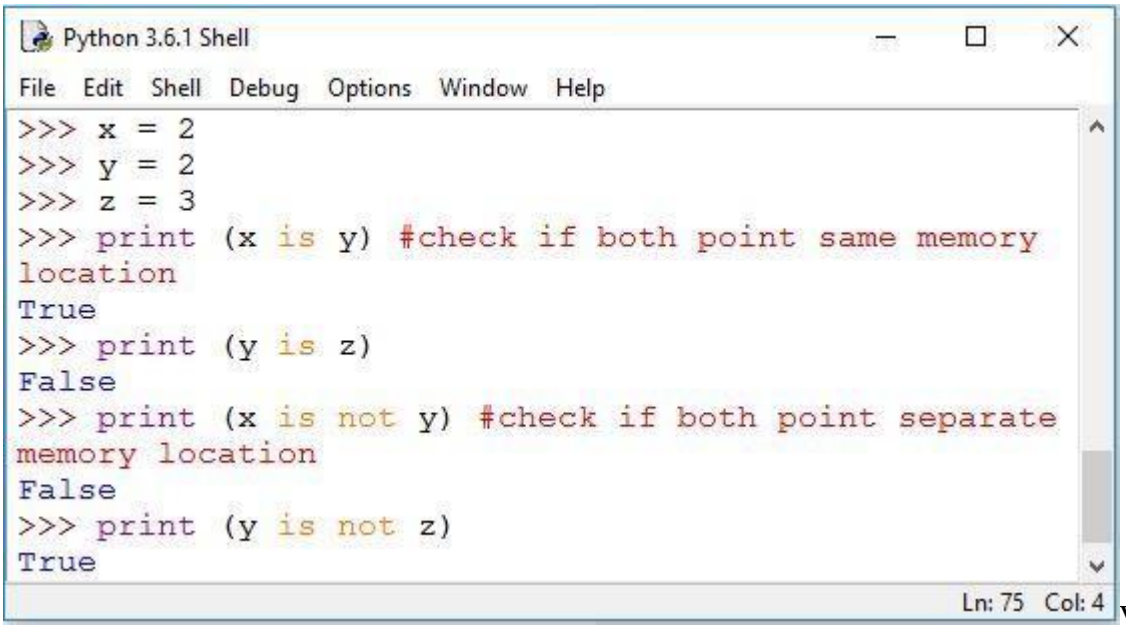
```
100
```

IDENTITY OPERATORS

Identity operators compare the memory locations of two objects

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

PROGRAM CODE :



v

TRY IT YOURSELF

```
>>> a=b=3.1
>>> a is b

>>> id(a)

>>> id(b)

>>>c,d=3.1,3.1
>>> c is d

>>> id(c)
>>> id(d)
```

MEMBERSHIP OPERATORS

Membership operator is an operator which test for membership in a sequence, such as string, list, tuple etc

Operator	Description
in	Evaluate to true, if it find a variable in the specified sequence; otherwise false.
not in	Evaluate to true, if it does not find a variable in the specified sequence; otherwise false

```
>>> 5 in [0,5,10,15]
True
>>> 6 in [0,5,10,15]
False
>>> 5 not in [0,5,10,15]
False
>>> 6 not in [0,5,10,15]
True
```

Find output

Sample code

```
vowels = ['A', 'E', 'I', 'O', 'U']
ch = input('Please Enter a Capital Letter:\n')
if ch in vowels:
        print('You entered a vowel character')
else:
        print('You entered a consonants character')
```

MODULE 4

CONDITIONAL STATEMENTS

CONDITIONAL STATEMENTS

Conditional Statement in Python perform different computations or actions depending on whether a specific Boolean constraint evaluates to true or false. Conditional statements are handled by IF statements in Python.

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision

Conditional statements are handled by if statements in Python

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.

It has a code body that only executes if the condition in the if statement is true. The statement can be a single line or a block of code.

Syntax:

```
if test expression:
    statement(s)
```

Here, the program evaluates the test expression and will execute statement(s) only if the test expression is True.

If the test expression is False, the statement(s) is not executed.

In Python, the body of the if statement is indicated by the indentation

Sample Code :

```
num = 5
if num > 0:
        print(num, "is a positive number.")
print("This statement is true.")
#When we run the program, the output will be:
5 is a positive number.
This statement is true.
```

if…else statement :

The if…else statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed and if it fails , another block of code will be executed.

Syntax :

```
if test expression:
    statement1
else:
   statement2
```

Sample code

```
num = 5
if num>= 0:
        print("Positive ")
else:
        print("Negative number")

output : Positive
```

if….elif….else

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.

Syntax of if...elif...else

```
if test expression:
    Body of if
elif test expression:
    Body of elif
else:
    Body of else
```

Sample Code :

```
x=float(input("1st Number: "))
y=float(input("2nd Number: "))
z=float(input("3rd Number: "))

if x>y and x>z:
    maximum=x
elif y>x and y>z:
    maximum=y
else:
    maximum=z

print(maximum)
```

OUTPUT:

```
1st Number: 1
2nd Number: 2
3rd Number: 3
3.0
```

Short Hand if

If you have only one statement to execute, you can put it on the same line as the if statement.

Example

```
if a > b: print("a is greater than b")
```

TRY IT YOURSELF

1.Write a program to check whether a person is eligible for voting or not. (accept age from user)

2.Write a python program that inputs an integer in the range (0 – 999) and prints whether it is a 1 or 2 or 3 digit number.

3. Write a program to display "Hello" if a number entered by user is a multiple of five , otherwise print "Bye".

MODULE 5

LOOPING CONSTRUCTS

LOOPING CONSTRUCTS

Loop executes set of statements repeatedly certain number of times.

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.

Python provides two types of looping constructs:

- while loop
- for loop

For Loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). It is also an entry check loop.

Syntax 1

```
for iterating_var in sequence:
    statement(s)
```

Syntax 2

```
for counter_variable in sequence:
        statements-block 1
[else: # optional block
        statements-block 2]
```

sample code 1

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

Output:

```
P
y
t
h
o
n
```

range( ) function

The range() is an in-built function in Python.

The **range()** function is used to generate the sequence of the numbers. for loop uses the range() function in the sequence to specify the initial, final and increment values.

range() generates a list of values starting from start till stop-1.

The syntax of range() is as follows:

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

It has three parameters, in which two are optional:

- **start:** It's an optional parameter used to define the starting point of the sequence. By default, it's zero.
- **stop:** It's a mandatory parameter, used to define the stopping point of the sequence
- **step:** It's also an optional parameter used to specify the incrementation on each iteration; by default, the value is one.

Float Arguments in Range

By default, the range() function only allow integers as parameters. If you pass float value, then it throws the following error:

```
TypeError:  'float' object cannot be interpreted as an integer
```

If we pass the range(10),
it will generate the numbers from 0 to 9

```
sum(range(10))
```

output

45

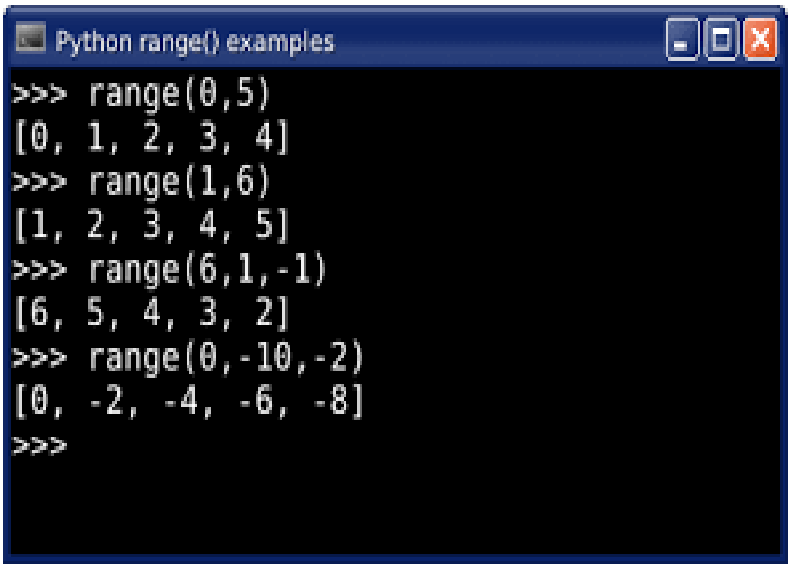
```
sum(range(4))
```

output

6

```
for i in range(10,20,2):
    print(i)
```

```
#OutPut
10
12
14
16
18
```



Sample code

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

Output:

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

Using else statement with for loop

Python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed. This feature is available only in Python.

```
for i in range(0,3):
    print(i)
else:
    print("for loop completely exhausted")
```

Output:

```
0

1

2

for loop completely exhausted
```

While loop

We generally use this loop when we don't know the number of times to iterate beforehand.

Syntax of while Loop in Python

```
while test_expression:
    Body of while
```

In the while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False.

Sample Code

```
i = 1
while i< 6:
   print(i)
   i += 1
```

sample code

```
count =0
while count <5:
   print count," is less than 5"
   count = count +1
```

output

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
```

Using else Statement with While Loop

Python supports to have an **else** statement associated with a loop statement.

- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise else statement gets executed

```
count =0
while count <5:
   print count," is less than 5"
   count = count +1
else:
   print count," is not less than 5"
```

When the above code is executed, it produces the following result −

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than
```

Real World Examples of Loop

1. Software program in a mobile device allows user to unlock the mobile with 5 password attempts. After that it resets mobile device.
2. You put your favorite song on a repeat mode. It is also a loop.

TRY IT YOURSELF

1. Write a program to count the occurence of even number and odd number between the range 10 – 55.

MODULE 6

NESTED LOOPS

NESTED FOR LOOPS

Nested for loop is having one or more for loops inside another for loop

Sample code

```
# 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:

```
Enter the rows:5
*
**
***
****
*****
```

TRY IT YOURSELF

1.Program to print multiplication table of given number.

2. write the python program to print the following output

```
1
22
333
4444
55555
```

break

The break is a keyword in python that is used to terminates the loop containing it. If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.One has to note an important point here is that ‘if a loop is left by break, the else part is not executed’

Syntax

```
break
```

program code

```
for letter in 'Python':     # First Example
   if letter == 'h':
      break
    print ('Current Letter :', letter)
```

output

```
Current Letter : P
Current Letter : y
Current Letter : t
```

continue

The continue statement, skips the current iteration and continues with the next iteration of the loop:

syntax

```
        continue
```

coding

```
for letter in 'Python':     # First Example
   if letter == 'h':
      continue
   print ('Current Letter :', letter)
```

output

```
Current Letter : P
Current Letter : y
Current  Letter  :  t
Current Letter : o
Current Letter : n
```

pass

pass statement in Python programming is a null statement. pass statement when executed by the interpreter it is completely ignored. Nothing happens when pass is executed, it results in no operation. pass statement can be used in ‘if ’ clause as well as within loop construct, when you do not want any statements or commands within that block to be executed

The **pass** statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet

pass statement is generally used as a placeholder. When we have a loop or function that is to be implemented in the future and not now, we cannot develop such functions or loops with empty body segment because the interpreter would raise an error. So, to avoid this we can use pass statement to construct a body that does nothing.

Syntax:

Pass

Example

```
for letter in 'Python':     # First Example
  if letter == 'h':
     pass
  print ('Current Letter :', letter)
```

output

```
Current Letter : P
Current Letter : y
Current  Letter  :  t
Current Letter : h
Current Letter : o
Current Letter : n
```

Sample code

```
a=int (input("Enter any number :"))
if (a==0):
        pass
 else:
        print ("non zero value is accepted")
```

Output:

```
Enter any number :3
 non zero value is accepted
```

When the above code is executed if the input value is 0 (zero) then no action will be performed, for all the other input values the output will be as per input

TRY IT YOURSELF

Write a python code to get the following output (adds only positive number)

```
Enter n1: 1.1
Enter n2: 2.2
Enter n3: 5.5
Enter n4: 4.4
Enter n5: -3.4
Enter n6: -45.5
Enter n7: 34.5
Enter n8: -4.2
Enter n9: -1000
Enter n10: 12
Sum = 59.70
```

MODULE 7

PYTHON ARRAYS

ARRAYS

An array is defined as a collection of items of similar data type that are stored at contiguous memory locations. Arrays in Python are Data Structures that can hold multiple values of the same type.

Arrays are also mutable (change) and not fixed in size, which means they can grow and shrink throughout the life of the program. Items can be added and removed, making them very flexible to work with.

Accessing array elements in Python :

Each and every element in an array is given an index or subscript. To access array elements, you need to specify the index values. Array Indexing starts at 0 .

a

12	13	14	15	15					
0	1	2	3	4					n-1

Index

a[0]=12

a[1]=13

Following operations can be performed on arrays:

- **Traversing**
- **Searching**
- **Insertion**
- **Deletion**
- **Sorting**
- **Merging**

How to use array in a program

By using **import array** at the top of the file to include the module array. You would then go on to create an array using **array.array( )**

import array

array.array( )

you can also use **from array import * (** * means importing all the functionalities) and then create an array by writing the **array( )** constructor alone.

from array import *

array( )

NumPy

NumPy is a special Python library with a large collection of high-level mathematical functions to operate on arrays. It also has functions for working with statistical calculations, linear algebra, fourier transform, and matrices.

Basic operations

Example

Get the first element from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

output as single element

```
1
```

#program for array addition

```
import numpy as np
x=np.array([2,3])
y=np.array([7,8])
sum=np.add(x,y)
print(sum)
```

output in array form

```
[ 9 11]
```

Sample code

```
import numpy as np
a = np.array([1, 2, 3, 6])
print(a**2)
```

output

```
[1,  4,  9, 36]
```

```
import numpy as np
np.sin(1)
```

```
0.8414709848078965
```

The value of sin 1 is 0.8414709848, in radian

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(a.transpose())
```

output

```
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

Create an identity matrix of dimension 4-by-4

```
import numpy as np
i = np.eye(4)
print(i)
```

Corresponding Output

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

Obtaining Boolean Array from Binary Array

Convert a binary numpy array (containing only 0s and 1s) to a boolean numpy array

```
import numpy as np
a = np.array([[1, 0, 0],
              [1, 1, 1],
              [0, 0, 0]])
o = a.astype('bool')
print(o)
```

Corresponding Output

```
[[ True False False]
 [ True  True  True]
 [False False False]]
```

Matrix Multiplication

Given 2 numpy arrays as matrices, output the result of multiplying the 2 matrices (as a numpy array)

```
import numpy as np
a = np.array([[1,2,3],
                    [4,5,6],
                    [7,8,9]])
```

```
b = np.array([[2,3,4],
              [5,6,7],
              [8,9,10]])
o = np.matmul(a, b)
print(o)
```

output

```
[[ 36 42  48]
 [ 81  96 111]
 [126 150 174]]
```

MODULE 8

PYTHON STRINGS

STRINGS

Strings are sequence of characters enclosed in single quotes or in double quotes.

String is a datatype in python. String is an array of characters.

Examples

'Welcome to Python' # string in single quotes

" Welcome to Python" # string in double quotes

''' Welcome to Python''' # string in triple quotes

""" Welcome to Python""" # string in triple double-quotes

Positive	0	1	2	3	4	5	index
	P	Y	T	H	O	N	
Negative	-6	-5	-4	-3	-2	-1	index

if you want a string to have a single quote as a part of a string ,then it should be given in double quotes.

```
str="Welcome to 'Python ' strings"
print(str)
```

output

```
Welcome to 'Python ' strings
```

If you want a string literal to have double quotes as part of a string then, it should be given in single quotes.

```
str='Welcome to "Python " strings'
print(str)
```

output

```
Welcome to "Python " strings
```

len( )

len( ) function is used to find the length of a string.

```
str='Python'
print(len(str))
```

```
6
```

replace( )

replace( ) function is used to change all occurrences of a particular character in a string

Sample code

```
str='Python'
print(str.replace('t','a'))
```

```
Pyahon
```

del( )

del( ) function deletes the string

Sample code

```
del str
print(str)
```

```
NameError : name ‘str’ is not defined
```

upper( )

upper( ) function returns the strings in upper case letters

Sample code

```
str='Python'
print(str.upper())
```

```
PYTHON
```

String Operators

Concatenation (+)

Two or more Strings can be joined using '+' operators in python

Sample code

```
str1='Python'
str2=' Programming'
print(str1+str2)
```

output

```
Python Programming
```

String Repetition Operator (*)

The same string can be repeated in python n times using string*n

Sample code

```
str= "Python "
print(str*1)
print(str*2)
print(str*3)
print(str*4)
```

output

```
Python
Python Python
Python Python Python
Python Python Python Python
```

String Slicing

To make substring from the original string.

Sample code

```
str= "Python "
print(str)
```

```
Python
```

Sample code

```
str= "Python "
print(str[:])
```

```
Python
```

 # starts from 0 till end

Sample code

```
str= "Python "
print(str[0:])
```

```
Python
```

#printing characters 0 to 3

Sample code

```
str= "Python "
print(str[:4])
```

```
Pyth
```

 # characters 2,3

Sample code

```
str= "Python "
print(str[2:4])
```

```
th
```

 # characters 2 to end

Sample code

```
str= "Python "
print(str[2:])
```

```
thon
```

#skip alternate characters

Sample code

```
str= "Python "
print(str[::2])
```

```
Pto
```

#skip 2 characters and printing 3rd character

Sample code

```
str= "Python "
print(str[::3])
```

output

```
Ph
```

 # printing in reverse from end

Sample code

```
str= "Python "
print(str[::-2])
```

```
otP
```

Built-in String functions

- **find( )** - returns the index of the first occurence of a substring in the given string (case-sensitive). If the substring is not found it returns -1.
- **index( )** - returns the index of the first occurence of a substring in the given string.
- **len( )** - returns length of the strings
- **capitalize( )** - capitalize the first character of the string
- **isupper( )** - returns true if the string is in upper case
- **islower( )** - returns true if the string is in lower case
- **split( )** - splits the given string to substrings
- **count( )** - returns the number of occurrences in the string
- **strip( )** - removes the white spaces either in the beginning or at the end
- **isdigit( )** - returns true if the string has only numbers
- **isalpha( )** - returns true if the string has only alphabets
- **isalnum( )** - returns true if the string has both alphabets and numbers
- **endswith( )**- returns true if string endwith the substring or character
- **isspace( )** - returns true if string has only white spaces
- **lstrip( )** - removes spaces from left side of the string

❖ **rstrip( )** - removes spaces from right of the string

TRY IT YOURSELF

1. Write a program to reverse a string in python.
2. Write a program in Python to count lower, upper, numeric and special characters in a string.
3. Write a python program to print

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

MODULE 9

PYTHON FUNCTIONS

FUNCTIONS

Functions are the blocks of code that accomplish a single task. Each and every function will have a name and will be called whenever needed. Functions enable the programmers to break down a problem into smaller components.

ADVANTAGES OF FUNCTIONS

- Reusability of Code
- Code Sharing

TYPES OF FUNCTIONS

Functions are classified as

- User defined functions
- Lambda functions
- Recursive functions
- Built-in Functions

USER DEFINED FUNCTIONS

A function that you can define on your own as per your need is known as user defined function.

DEFINING FUNCTIONS

def keyword is used to define a function followed by the name of a functions .

SYNTAX

```
def functionname :
    block of code
    return
```

Sample code

```
def sum(a, b):
        return (a + b)
a = int(input('Enter 1st number: '))
b = int(input('Enter 2nd number: '))
# f is format for output
print(f'Sum of {a} and {b} is {sum(a, b)}')
```

output

```
Enter 1st number:
12
Enter 2nd number:
22
Sum of 12 and 22 is 34
```

Here in this example , a and b are called parameters through which the values are passed to the functions. Name of the function is sum. The function sum is called from the print function in the main program

RETURN STATEMENT

The Python return keyword exits a function and instructs Python to continue executing the main program. The return keyword can send a value back to the main program.

LAMBDA FUNCTIONS

Lambda functions are also called as anonymous functions. These function are defined without a name. Lambda functions are defined with the keyword lambda. Lambda functions are used for creating small and one time anonymous functions. Lambda functions can take any number of arguments and must return one value.

Sample code

```
cube = lambda a: a*a*a
print(cube(10))
```

output

```
1000
```

RECURSIVE FUNCTIONS

A recursive function is a function that calls itself during its execution. The process may repeat several times. It calls itself over and over again until a base condition is met that breaks the loop.

Why recursion?

Recursion is made for solving problems that can be broken down into smaller, repetitive problems. It is especially good for working on things that have many possible branches and are too complex for an iterative approach .

Sample code

Finding factorial

```
def rec_fact(n):
        if n == 1:
                return n
        else:
                return n*rec_fact(n-1)     #function calling itself
#taking input from the user
n = int(input("Enter any number : "))
print("The factorial of", n, "is", rec_fact(n))
```

output

```
Enter any number :
5
The factorial of 5 is 120
```

Built-in Functions

A built-in function is a function that is already available in a programming language, application, or another tool that can be accessed by end users.

Some of the built-in functions are given below

len()

list()

max()

min()

next()

object()

oct()

open()

pow()

print()

range()

round()

set()

str()

sum()

tuple()

type()

abs()

ascii()

bin()

bool()

program for practice

1. write a program to find area of a circle using functions

MODULE 10

PYTHON LISTS

INTRODUCTION TO LISTS

List is a data type used in python for storing multiple items of different or same data type in a single variable. Some popular languages that include list comprehension are JavaScript, Perl 6, C# and Python.

Example :

Num=[12,13,15]

Sample code

```
#list sample code
name=['Hari','Booshan','Charu','Shami','Varshini','Ashwin']
print(name)
```

output

```
['Hari', 'Booshan', 'Charu', 'Shami', 'Varshini', 'Ashwin']
```

List elements are mutable which means list contents can be changed.

List also has index value like arrays. In python , index value is an integer number which can be positive or negative.

 # to create empty list

```
List=[ ]
```

 # list of same data type

```
List=[1,2,3,4]
```

 # list of different items

```
List=[12,'python',3.6]
```

 # list of lists

```
List=[[11,22],[33,44,55]]
```

Sample code

 # same value repeatedly

```
list = [3]* 5
print(list)
```

output

```
[3, 3, 3, 3, 3]
```

List Traversal

Each element in a list can be accessed using its index in square brackets. The list elements can be accessed using loops using positive or negative index.

Sample code

```
weight=[22.5,66,45.5,34.5]
for x in weight:
        print(x)
```

output

```
22.5
66
45.5
34.5
```

List methods in Python

Method	Description
clear( )	Removes all the elements from the list
remove( )	to delete a particular element
del list	to delete entire list
copy( )	Returns a copy of the list
count( )	Returns the number of elements with the specified value
append( )	Adds single element to the existing list
extend( )	Add more than one element to the existing list
insert( )	to include a element at desired position
len( )	Returns the number of element in the list .
list( )	Creates a list out of an iterable.
range( )	Returns an iterator of integers from start to stop, with an increment of step.

Sample code

```
square=[a**2 for a in range(1,10)]
    print(square)
```

output

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Why 100 not printed ?

sum( ) Adds all items of an iterable.

Sample code

```
print(sum([19,3,2,5,1,-9]))
```

```
21
```

min( ) Gets the smallest item in a sequence.

Sample code

```
print(min([19,3,2,5,1,-9]))
```

```
-9
```

max( ) Gets the largest item in a sequence.

Sample code

```
print(max([19,3,2,5,1,-9]))
```

```
19
```

sort( ) Returns a new list of sorted items in iterable.

Sample code

```
name=['Hari','Booshan','Charu','Shami','Varshini','Ashwin']
name.sort(reverse=True)
print(name)
```

output

```
['Varshini', 'Shami', 'Hari', 'Charu', 'Booshan', 'Ashwin']
```

map( ) maps a function to each item of iterables and returns an

iterator.

Syntax: map(function, iterable,...]

This function is mostly used when we want to apply a function on each item of iterables but we don't want to use the traditional for loop.

Sample code

prg to Add 2 to each item of the list

```
>>> l1 = [6,4,8,9,2,3,6]
>>> list(map(lambda x: x+2, l1))
[8, 6, 10, 11, 4, 5, 8]
```

In the example above, we used lambda expressions to add 2 to each item and we used the Python list() function to create a list from the iterator returned by the map() function.

Nested list:

A list having one or more sub-lists is called a nested list.

One of the big advantages of list comprehension is that they allow developers to write less code that is often easier to understand.

List comprehension is usually faster.

Program

1. Write a program in Python to remove duplicate items from a list.

MODULE 11

PYTHON TUPLES

TUPLES

Tuples are collection of objects which are ordered and immutable

(unchangeable which means we cannot change the elements of a tuple once

assigned.). Tuples are used to store multiple items in a single variable.

A tuple is one of the four inbuilt data types used to store collections in Python. .

Tuples are sequences, just like lists.

Tuples Vs Lists

1. The elements of a list are mutable whereas the elements of a tuple are immutable.
2. Iterating over the elements of a tuple is faster compared to iterating over a list.

To create a tuple in Python, place all the elements in a ( ) parenthesis, separated by commas. A tuple can have heterogeneous data items, a tuple can have string and list as data items as well.

 # empty tuple

```
tup = ( )
```

 # Tuple with only single element:

To create a tuple with only one element, just put a comma after the element.

```
tup = (12,)
```

if comma is not given, it is considered as integer.

Sample code

#creating tuples of similar data type

```
Tup=( 1,2,3,4,5 )
print(Tup)
```

output

```
(1, 2, 3, 4, 5)
```

Sample code

#creating tuples of different data types

```
Tup=(6,'hari',19)
print(Tup)
```

Sample code

#tuples of tuples

```
Tup=( (1,2),(3,4),(5,6))
print(Tup)
```

output

```
((1, 2), (3, 4), (5, 6))
```

Sample code

 # tuple of lists

```
Tup=( [1,2],[3,4],[5,6,7])
print(Tup)
```

output

```
([1, 2], [3, 4], [5, 6, 7])
```

Sample code

 # tuple of tuples,lists,strings

```
Tup=( (1,2),[3,4],'hari')
print(Tup)
```

output

```
((1, 2), [3, 4], 'hari')
```

Accessing Tuple Elements

Tuple elements can be accessed similar to arrays and lists by indexing. To access the values or elements of tuples, we can use square brackets and follow positive or negative indexing.

index

0	1	2	3	4	5
Mango	Orange	Grapes	Apple	Guava	Banana
-6	-5	-4	-3	-2	-1

tup=(‘Mango’,’Orange’,’Grapes’,’Apple’,’Guava’,’Banana’)

Pack Unpack

In packing, we place value into a new tuple while in unpacking we extract those values back into variables.

If we talk of packing in literal terms, Just like we pack certain items into a box in the real world. In python we pack certain variables in a single iterable.

Unpacking a tuple means splitting the tuple's elements into individual variables. For example: x, y = (1, 2)

Sample code

```
x = ("HARI", 100, "CS")    # tuple packing
(NAME, MARKS, SUBJECT) = x    # tuple unpacking
print(NAME)
print(MARKS)
print(SUBJECT)
```

Output

```
HARI
100
CS
```

Delete

To delete an entire tuple, the del command is used.

Syntax :

del tuple_name

Example

```
del tup
```

After deleting tuple, if you try to print the tuple, python will show name error

Slicing

A slice is a substring of the main string. A substring can be taken from the original string by using [ ] operator and index or subscript values. Thus, [ ] is also known as the slicing operator. Using the slice operator, we can slice one or more substrings from the main string.

Sample code

```
tup = (11, 22, 33, 44, 55, 66, 77, 88, 99)
print(tup)

# prints (33, 44, 55)
print(tup[2:5])

# prints (11, 22, 33, 44)
print(tup[:4])

# prints (55, 66, 77, 88, 99)
print(my_data[4:])

# prints (55, 66, 77, 88)
print(my_data[4:-1])

# displaying entire tuple
print(my_data[:])
```

Program

1.write a program to swap two values using tuple assignment.

MODULE 12

PYTHON SETS AND DICTIONARY

SETS

Set is an unordered (it will be displayed in random order) and unindexed (no indexing for set elements) collection of items in Python. The elements of a set are defined inside curly brackets and are separated by commas. Each element in the set must be unique (duplicates not allowed), immutable (unchangeable which means we cannot change the elements)

set( ) function is used to create sets in python.

For example –

Set1 = {1, 2, 3, "welcome"}

Set operations

The major set operations are

- union
- intersection
- difference
- symmetric difference

Union( )

In python , use union( ) or | operator function to combine all elements from two or more sets

Method 1: mySet1.union(mySet2)

Method 2: mySet1 | mySet2

intersection( )

In python, use intersection( ) function or & operator to have common elements in both the sets.

Method 1: mySet1.intersect(mySet2)

Method 2: mySet1 & mySet2

difference( )

The difference( ) functions or minus- operator is used to list the elements in the first and not in second.

Method 1: mySet1.difference(mySet2)

Method 2: mySet1 - mySet2

symmetric difference( )

The symmetric difference( ) function or caret operator is used to list all the elements in 2 sets and (removing common elements) not the elements in both.

Method 1: mySet1.symmetric_difference(mySet2)

Method 2: mySet1 ^ mySet2

DICTIONARY – KEY VALUE PAIRS

Dictionary is a mixed collection of elements. Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is ordered*, changeable and do not allow duplicates. It is represented by curly {} brackets. While creating a dictionary, we should always remember that keys should always be unique.

Inside the curly braces you have a key-value pair. Keys are separated from their associated values with colon.

Dict={'name': 'ram'}

What are the advantages of dictionary in Python?

> It improves the readability of your code. Writing out Python dictionary keys along with values adds a layer of documentation to the code. If the code is more streamlined, it is a lot easier to debug

to create empty dictionary

```
Dict={ }
```

Get the list of keys

```
Dict.keys()
```

#Delete element with key “-k” from the dictionary

```
del Dict["k"]
```

Delete all the elements in the dictionary

```
Dict.clear()
```

Sample code

```
dict={' maths': 100, 'science': 100, 'computer': 100}
print(dict)
```

output

```
{' maths': 100, 'science': 100, 'computer': 100}
```

Sample code

to display keys

```
dict={'maths': 100, 'science': 100, 'computer': 100}
print(dict.keys())
```

output

```
dict_keys(['maths', 'science', 'computer'])
```

Sample code

```
#dictionary sample code
sub=('maths','science','computer')
centum=100
dict1 = dict.fromkeys(sub,centum)
print(dict1)
```

#output

```
{'maths': 100, 'science': 100, 'computer': 100}
```

MODULE 13

PYTHON CLASSES AND OBJECTS

CLASSES AND OBJECTS

Python is an object-oriented language and almost every entity in Python is an object, which means that programmers extensively use classes and objects while coding in Python.

Encapsulation which is one of the important features of OOP's in Python describes the concept of bundling data and methods within a single unit . Objects in Python are basically an encapsulation of Python variables and functions that they get from classes.

Data Abstraction which is another important feature of OOP's is implemented to hide unnecessary data and withdrawing relevant data. These mechanism hides the code and the data together from the outside world or misuse. It focuses on the inner details of how the object works. Modifications can be done later to the settings.

Creating classes

Python class simply behaves as a template to create objects. Python class is simply an entity. The **class** keyword is used for creating a class. Constructor is a special function used to initialize the class variables. Constructor function will be automatically executed when an object is created.

In Python, Constructor is created by a special function called "**init**". It starts and ends with double underscore symbol.

The self is used to represent the instance of the class. With this keyword, you can access the attributes and methods of the class in python. It binds the attributes with the given arguments.

Variables of class

The class variables are called as class objects. The class variables are accessed using dot operator. A Python class variable is shared by all object instances of a class. Class variables are declared when a class is being constructed.

Public scope :

The class variables are public by default. These variables can be accessed anywhere in the program.

Private scope :

A variable prefixed with double underscore is a private variable. These variables can be accessed only within the class.

Methods of a class

Functions of the class are called as Methods.

Sample code

```
class student:
    id = 10
    name = "Hari"
    def display (self):
        print(self.id,self.name)
```

Here, the self is used as a reference variable, which refers to the current class object. It is always the first argument in the function definition. However, using self is optional in the function call.

Sample code

```
#class name student
class student:
    RollNo = 1   # class variables
    name = "Hari"
    def display (self):  #method display
        print("RollNo: %d \nName: %s"%(self.RollNo,self.name))
# Creating a instance of class
stu=student()
stu.display() #accessing class function using dot operator
```

output

```
RollNo: 1
Name: Hari
```

Try it yourself

1.write a Python program using class to find area of a rectangle.

MODULE 14

14 PYTHON MODULES AND PACKAGES

MODULES IN PYTHON

Already we have learnt about functions in Python. Consider, if we are using the same function in different programs, we have to define it again and again so many times which is not that much easy . That is time consuming also. We can just create a code or a script containing that function and save it as a file with a name and import that script in every program that makes use of that function. These scripts are called modules in Python.

The module is a simple Python file that can contain (Python functions, python variables, python classes). In Python, modules are used to divide the code into smaller parts. In this, we can group similar data which makes the program easier to understand.

Module is a file which contains python functions , global variables etc.

A module is a file containing Python definitions and statements.

It is a . py file which has python executable code / statement.

PACKAGES

There are more than 200,000 Python packages in the world. Python Modules are grouped as Packages. Already in previous MODULEs, we discussed about the most important packages numpy.

Example of Python Package:

```
import math
print("math package")
```

Package is namespace which contains multiple package/modules.

LIBRARY

It is a collection of modules. (Library either contains built in modules (written

in C) + modules written in python)

Example min( ) , max( ), sum( )

Import Module in Python

To import the module in Python, the import keyword is used.

Few important packages are given below.

- **Pandas** is an open source Python package that is most widely used for data science/data analysis and machine learning tasks.

- **Scikit-learn (Sklearn)** is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

- **NumPy** is used for:
  Advanced array operations (e.g. add, multiply, slice, reshape, index), Comprehensive mathematical functions, Random number generation, Linear algebra routines, Fourier transforms.

- **Matplotlib** is the most common data exploration and visualization library. You can use it to create basic graphs like line plots, histograms, scatter plots, bar charts, and pie charts. You can also create animated and interactive visualizations with this library. Matplotlib is the foundation of every other visualization library.
  The library offers a great deal of flexibility with regards to formatting and styling plots. You can freely choose how to display labels, grids, legends, etc.

- **Pillow**

If you work with image data, make sure to check out the Pillow package. It is a fork of PIL (Python Image Library) that developed into an easy-to-use and efficient tool for image manipulation in Python.

Pillow is used for:
Open and save images of different file types (JPEG, PNG, GIF, PDF, etc.).

Create thumbnails for images.
Use a collection of image filters (e.g. SMOOTH, BLUR, SHARPEN).

This is a great image manipulation tool for beginners, and it has fairly powerful image processing capabilities.

- **pytest**

  This package provides a variety of modules for testing new code,

  including small unit tests and complex functional tests for applications

  and libraries.

MODULE 15

PYTHON DATE TIME MODULE

Date time module

The datetime module supplies classes for manipulating dates and times.

The datetime module exports the following constants:

datetime.MINYEAR

The smallest year number allowed in a date or datetime object.

MINYEAR is 1.

datetime.MAXYEAR

The largest year number allowed in a date or datetime object.

MAXYEAR is 9999.

date Objects

A date object represents a date (year, month and day) in an idealized calendar,

the current Gregorian calendar indefinitely extended in both directions.

January 1 of year 1 is called day number 1, January 2 of year 1 is called day number 2, and so on. 2

class datetime.date(year, month, day)

The datetime module has many methods to return information about the date

object.

All arguments are required. Arguments must be integers, in the following ranges:

MINYEAR <= year <= MAXYEAR

1 <= month <= 12

1 <= day <= number of days in the given month and year

If an argument outside those ranges is given, ValueError is raised.

classmethod date.today()

Return the current local date.

Sample code

```
import datetime
x = datetime.datetime.now()
print(x)
```

output

```
2022-06-29 05:48:16.129384
```

The date contains year, month, day, hour, minute, second, and microsecond.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

Sample code

#program to return the year and name of weekday:

```
import datetime
x = datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
```

output

```
2022
Wednesday
```

Various date time formats

Directive	Description	
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6,	0 is Sunday
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year,short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%C	Century	20
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00

%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

Sample code

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%p"))
```

output

```
AM
```

Sample code

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%B"))
```

OUTPUT

```
June
```

Sample code

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%j"))
```

output

```
180
```

Sample code

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%G"))
```

output

2022