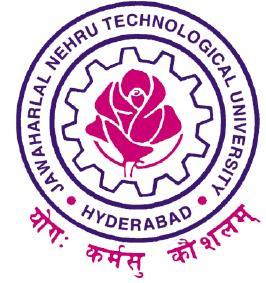
J.N.T.U.H. COLLEGE OF ENGINEERING

**KUKATPALLY, HYDERABAD – 500 085**

**CERTIFICATE**

***This is to certify that B.Kishore Reddy of B. Tech II year II Semester bearing the Hall-Ticket Number 17011A0514 has fulfilled his/her SCRIPTING LANGUAGES record for the academic year 2018-2019.***

***Signature of the Head of the Department                                Signature of the staff member***

***Date of Examination\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Internal Examiner                                                                    External Examiner***

|  |  |
| --- | --- |
| Page No. |  |
| 3 | Perl Introduction |
| 9 | 1 a.PERL program to find largest number |
| 10 | b.PERL program to print multiplication tables |
| 14 | 2.PERL program to implement manipulation functions |
| 15 | 3.PERL script to substitute a word with another |
| 16 | 4.PERL script to validate email ID and IP address |

|  |  |
| --- | --- |
| 17 | PHP Introduction |
| 22 | 1.PHP script to print prime numbers |
| 23 | 2 a.PHP script to find length of string |
| 23 | b.Count number of words |
| 24 | c.Reverse a string |
| 24 | d.Search for a specific string |
| 25 | 3.PHP script to merge 2 arrays and sort |
| 26 | 4.PHP program to copy data from one file to another |

|  |  |
| --- | --- |
| 28 | Python Introduction |
| 33 | 1 a.Python program to solve quadratic equation |
| 33 | b.Python program to find factorial |
| 34 | 2.Python program to generate Fibonacci series |
| 34 | 3.Python program to make simple calculator |
| 36 | 4.Python program to sort words in alphabetical order |
| 36 | 5.Python program to add 2 matrices |

**PERL**

* A program in general is a sequence of instructions written so that a computer can perform certain task.
* A script is a code written in any scripting language. A scripting language is nothing but a type of programming language in which we can write a code to control another software application
* In fact programming languages are of two types:
  + Scripting languages :No need to explicitly invoke code conversion process. It happens automatically in the background when the source code is processed by an interpreter.
  + Compiled languages: A compiled Language is converted into machine code or an intermediate code before the program runs, by a highly specialised piece of software called a compiler.
* Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more.
* Perl is a stable, cross platform programming language
* Though Perl is not officially an acronym but few people used it as **Practical Extraction and Report Language**
* Perl was created by Larry Wall
* Perl Features
* Perl takes the best features from other languages, such as C, awk, sed, sh, and BASIC, among others.
* Perl works with HTML, XML, and other mark-up languages.
* Perl supports Unicode.
* Perl is Y2K compliant.
* Perl supports both procedural and object-oriented programming.
* Perl interfaces with external C/C++ libraries through XS or SWIG.
* Perl is extensible. There are over 20,000 third party modules available from the Comprehensive Perl Archive Network ([CPAN](http://cpan.perl.org/)).
* The Perl interpreter can be embedded into other systems.
* Perl is Interpreted

**PERL V/s Shell Scripting**:

* Even though, shell scripting is available to programmers, they prefer Perl because:
* Programming on Perl does not cause portability issues, which is common when using different shells in shell scripting.
* Error handling is very easy on Perl
* You can write long and complex programs on Perl easily due to its vastness. This is in contrast with Shell that does not support namespaces , modules , object , inheritance etc.
* Shell has fewer reusable libraries available . Nothing compared to Perl's CPAN
* Shell is less secure. Its calls external functions(commands like mv , cp etc depend on the shell being used) . On the contrary PERL does useful work while using internal functions
* First Perl Programming
* Check Perl is installed on your system or not?
  + $perl –v

**Interactive Mode Programming**

* + perl -e 'print "Hello World\n"'
* **Script Mode Programming**
  + **#!/usr/bin/perl**
  + **print("May I take your name please?") ;**

**Perl Variables:**There are Three types of variables

* + Scalars
  + Arrays
  + Hashes

**Scalar Variable**

* + This type of variable holds a single value.
  + Its name begins with a dollar sign and a Perl identifier (it's our name of variable)( character after $ will be either an alphabet or the underscore)
* Two Types of Scalar Data Types
  + Numbers
  + Strings
* Example:
  + $var; $Var32; $vaRRR43; $name\_underscore\_23;

**Arrays**

* @months = ("July", "August", "September");
  + print $months[0]; # This prints "July".
  + $months[2] = "Smarch"; # We just renamed September!
  + $winter\_months[0] = "December"; # This implicitly creates @winter\_months.
* To find the length of an array, use the value $#array\_name.
* If the array just doesn't exist or is empty, $#array\_name is -1.
* To resize an array, just change the value of $#array\_name.
  + @months = ("July", "August", "September");
  + print $#months; # This prints 2.
  + $a1 = $#autumn\_months; # We don't have an autumn\_months, so this is -1.
  + $#months = 0; # Now @months only contains "July".
  + Swaping: @foo[0,1]=@foo[1,0]
  + @months[0..10]=(1,2,3,4..)

**Hashes:**

* *Hashes* are called ``dictionaries'' in some programming languages
* They contains key and value.
* key in a hash has one and only one corresponding value
* The name of a hash begins with a percentage sign, like %parents
* %days\_in\_summer = ( "July" => 31, "August" => 31, "September" => 30 );

**Perl loops:**

* As with all other high-level languages, Perl supports loops. The for and while loops are similar to those in languages like C/C++.
* Loops allow you to run a block of code as many times as you want, as long as some condition evaluates to true. Loops always have some condition attached to them.

**The for loop:**

* The for loop in Perl is similar to that in C/C++ and Java. It consists of three components:  
  for (initial;condition;increment)  
  where initial is the code to run prior to starting the loop, condition is tested prior to each loop and must be true for the loop to continue, and increment is performed after every loop.

**The while loop:**

* The while loop uses a condition to test whether to loop or not. When the condition is true, the loop executes. When the condition is false, the loop terminates. The syntax is:  
  while (condition)  
  {statements…}  
  The condition can be anything that evaluates to true or false.

**The next statement:**

* The next statement causes execution of a loop to restart. It is similar to the continue statement in some languages. Any statements below the next statement are not executed. The next statement is usually used with a conditional:  
  while ($x < 10)  
  { if ($x == 6) {next;}  
   statements…}

**Labels:**

* Blocks of code and the for and while statements can all be labeled, and identified by that label. To use a label, place it before the statement block followed by a colon:  
  BLOCK1: {statements…}
* The name of the label can be any valid ASCII characters. The convention is to use uppercase for labels so there is no conflict with existing keywords

**The exit statement:**

* The exit statement is used to terminate a Perl script at any time. Whenever the exit statement is encountered, the script is terminated. You can send a return status code back to the calling program with exit, if you want, by appending the return code after the exit statement:  
  if ($val1 == 0) { exit 0;}
* This will exit the program with a return code 0

**Perl – Subroutines:**

* A Perl subroutine or function is a group of statements that together performs a task
* Perl uses the terms subroutine, method and function interchangeably.
* The general form of a subroutine definition in Perl programming language is as follows:

sub subroutine\_name{

body of the subroutine

}

* The typical way of calling that Perl subroutine is as follows
  + subroutine\_name( list of arguments );
* In versions of Perl before 5.0, the syntax for calling subroutines
  + &subroutine\_name( list of arguments );
* Passing parameters
* @\_ is a special array variable which stores the arguments passed to the subroutines.
* The first argument to the function is in $\_[0], the second is in $\_[1], and so on.

**Regular Expression:**

* A regular expression is a string of characters that defines the pattern or patterns you are viewing.
* The basic method for applying a regular expression is to use the pattern binding operators **=~** and **!~**.
* The operators are test and assignment operator.
* There are three regular expression operators within Perl.
* Match Regular Expression - m//
* Substitute Regular Expression - s///
* Transliterate Regular Expression - tr///
* The forward slashes in each case act as delimiters for the regular expression (regex) that you are specifying.

**The Substitution Operator:**

* The substitution operator, s///, is really just an extension of the match operator that allows you to replace the text matched with some new text.
* The basic form of the operator is −
* s/PATTERN/REPLACEMENT/;
* Example:

#!/user/bin/perl

$string = "The cat sat on the mat";

$string =~ s/cat/dog/;

print "$string\n";

* o/p:

The dog sat on the mat

**The Translation Operator:**

* The translation replaces all occurrences of the characters in SEARCHLIST with the corresponding characters in REPLACEMENTLIST
* The translation operators are −
* tr/SEARCHLIST/REPLACEMENTLIST/cds
* y/SEARCHLIST/REPLACEMENTLIST/cds
* Example:

#!/user/bin/perl

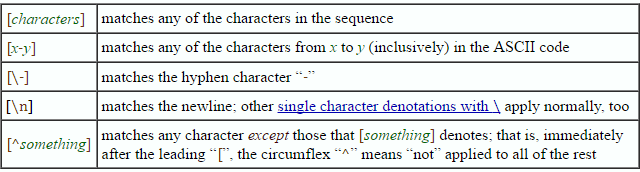
$string = 'The cat sat on the mat';

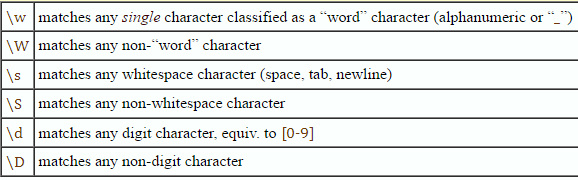
$string =~ tr/a/o/;

print "$string\n";

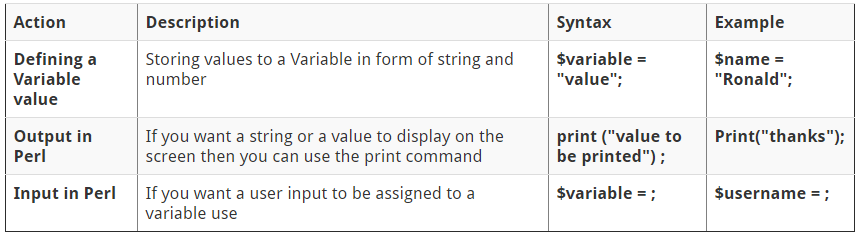
* O/P:The cot sot on the mot.

**Character sets: specialities inside[...]**





**Storing Variables, Input and Output**



* Perl Programs

1. **Aim : To write a PERL script to find largest among 3 numbers.**

**Program:**

print "Largest of three numbers\n";

print "a: ";

$a=<stdin>;

print "b: ";

$b=<stdin>;

print "c: ";

$c=<stdin>;

if($a>$b)

{

if($a>$c)

{

print "a=$a is largest\n";

}

if($a<$c)

{

print "c=$c is largest\n";

}

if($a==$c)

{

print "a,c=$c are largest\n";

}

}

if($b>$a)

{

if($b>$c)

{

print "b=$b is largest\n";

}

if($b<$c)

{

print "c=$c is largest\n";

}

if($b==$c)

{

print "b,c=$c are largest\n";

}

}

if($a eq $b)

{

if($c>$a)

{

print "c=$c is largest\n";

}

if($c<$a)

{

print "a,b=$a are the largest\n";

}

if($c==$a)

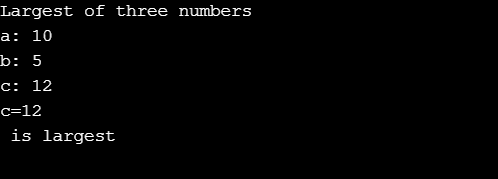
{

print "a,b,c=$a are equal";

}

}

**Output:**



1. **Aim : To write multiplication tables from 1-10 using subroutines**

**Program:**

sub multable

{

for($j=1;$j<11;$j=$j+1)

{

$pro=$\_[0]\*$j;

print "$\_[0] x $j = ($pro)\n";

}

}

print "Multiplication tables\n";

for($i=1;$i<11;$i=$i+1)

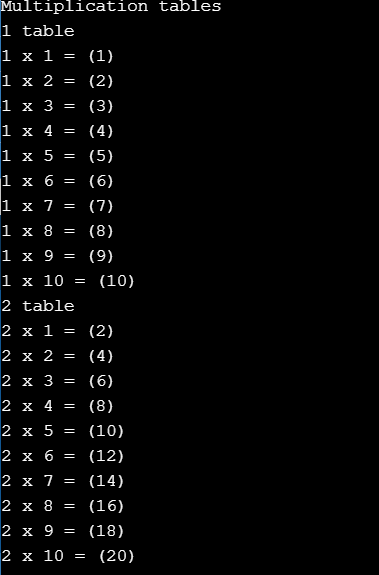
{

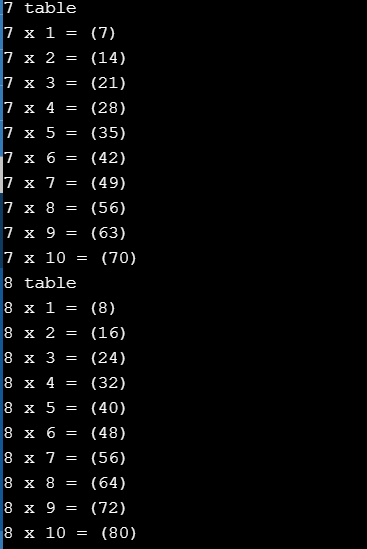
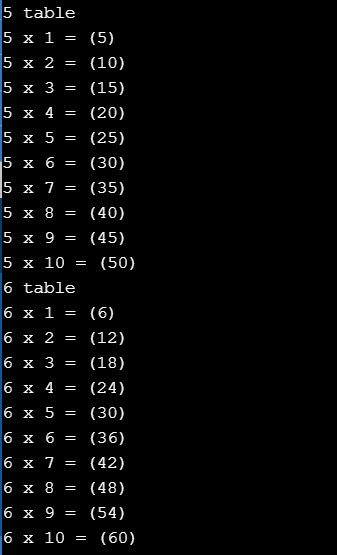
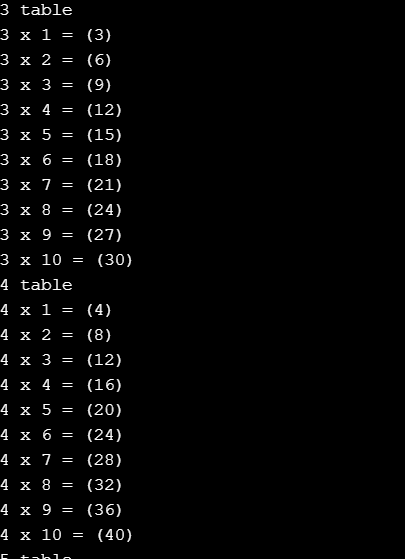
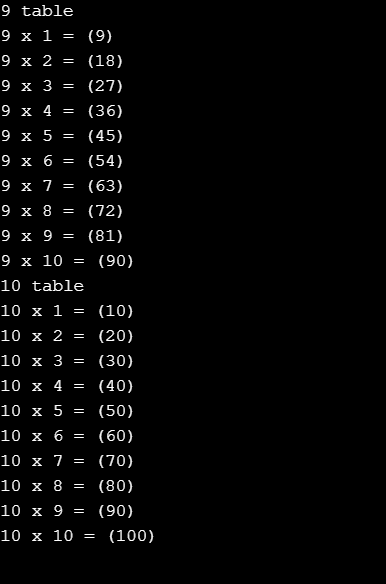
print "$i table\n";

multable($i);

}

**Output:**



1. **Aim : Write a PERL program to implement the following list of manipulating functions.**
2. **Shift**
3. **Unshift**
4. **Push**

**Program:**

print "Shift\n";

@ele=('j','n','t','u','h','c','e','h');

print "@ele\n";

$shiftedele=shift(@ele);

print "Shifted ele= $shiftedele\n";

print "@ele\n";

print "Element inserted at beginning(unshift)\n";

chomp($first=<stdin>);

unshift(@ele,$first);

print "@ele\n";

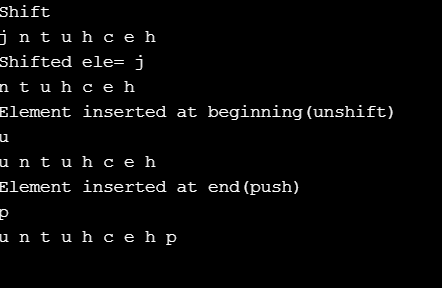
print"Element inserted at end(push)\n";

$last=<stdin>;

push @ele,$last;

print "@ele\n";

**Output:**



1. **Aim : To write a PERL script to substitute a word with another word in a string.**

**Program:**

print("Replace substring\n");

print("Main string\n");

chomp($mains=<stdin>);

print("Word\n");

chomp($w=<stdin>);

print("New word\n");

chomp($new=<stdin>);

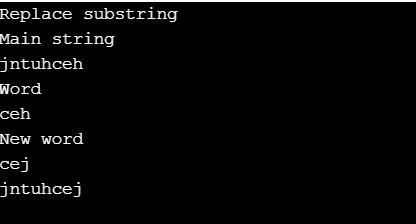
$posw=index($mains,$w);

$lenw=length($w);

substr($mains,$posw,$lenw)=$new;

print($mains);

**Output:**



1. **Aim : To validate IP address and email address**

**Program:**

print("Validate email address\n");

print("Email address\n");

chomp($email=<stdin>);

if($email=~/([a-zA-Z0-9]+)\@([a-zA-z0-9]+).(com|net|org)/)

{ print "Valid";}

else

{ print "Invalid";}

print("Validate IP address\n");

print("IP address\n");

chomp($ip=<stdin>);

if($ip=~/([0-9]{0,3})[.]([0-9]{0,3})[.]([0-9]{0,3})[.]([0-9]{0,3})/)

{

print "Valid";

}

else

{ print "Invalid"; }

**Output:**



**PHP**

* PHP is server side scripting system
* PHP stands for "PHP: Hypertext Preprocessor"
* Syntax based on Perl, Java, and C
* Very good for creating dynamic content
* Powerful, but somewhat risky!
* If you want to focus on one system for dynamic content, this is a good one to choose

**Client-side Environment**

* The client-side environment used to run scripts is usually a browser.
* The processing takes place on the end users computer. The source code is transferred from the web server to the users computer over the internet and run directly in the browser.
* The scripting language needs to be enabled on the client computer.
* Sometimes if a user is conscious of security risks they may switch the scripting facility off.
* When this is the case a message usually pops up to alert the user when script is attempting to run

**Server-side Environment**

* The **server-side environment** that runs a scripting language is a web server.
* A user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages. This HTML is then sent to the client browser.
* It is usually used to provide interactive web sites that interface to databases or other data stores on the server.
* This is different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript.
* The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores

**PHP INSTALLATION**

* To start using PHP:
* Find a web host with PHP and MySQL support
* Install a web server on your own PC, and then install PHP and MySQL

**SYNTAX**

* A PHP script is executed on the server, and the plain HTML result is sent back to the browser.
* A PHP script can be placed anywhere in the document.
* A PHP script starts with **<?php** and ends with **?>**:

**PHP Case Sensitivity**

* In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive

**Variables:**

* Variables are "containers" for storing information

**Creating (Declaring) PHP Variables**

* In PHP, a variable starts with the $ sign, followed by the name of the variable:
* Example
* <?php  
  $txt = "Hello world!";  
  $x = 5;  
  $y = 10.5;  
  ?>

**Rules for PHP variables:**

* A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume).
* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive ($age and $AGE are two different variables)

**phpinfo()**

* The phpinfo() function shows the php environment
* Use this to read system and server variables, setting stored in php.ini, versions, and modules.

**The PHP echo Statement**

* The echo statement can be used with or without parentheses: echo or echo()

EXAMPLE

<?php  
echo "<h2>PHP is Fun!</h2>";  
echo "Hello world!<br>";  
echo "I'm about to learn PHP!<br>";  
echo "This ", "string ", "was ", "made ", "with multiple parameters.";  
?>

**The PHP print Statement**

* The print statement can be used with or without parentheses: print or print().
* <?php  
  print "<h2>PHP is Fun!</h2>";  
  print "Hello world!<br>";  
  print "I'm about to learn PHP!";  
  ?>

**PHP Data Types**

* Variables can store data of different types, and different data types can do different things.
* PHP supports the following data types:
* String
* Integer
* Float (floating point numbers - also called double)
* Boolean
* Array
* Object
* NULL
* Resource

**PHP String Functions**

**Get The Length of a String**

* The PHP strlen() function returns the length of a string.
* The PHP str\_word\_count() function counts the number of words in a string:
* The PHP strrev() function reverses a string:

**PHP Constants**

* A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
* A valid constant name starts with a letter or underscore (no $ sign before the constant name)
* To create a constant, use the define() function.

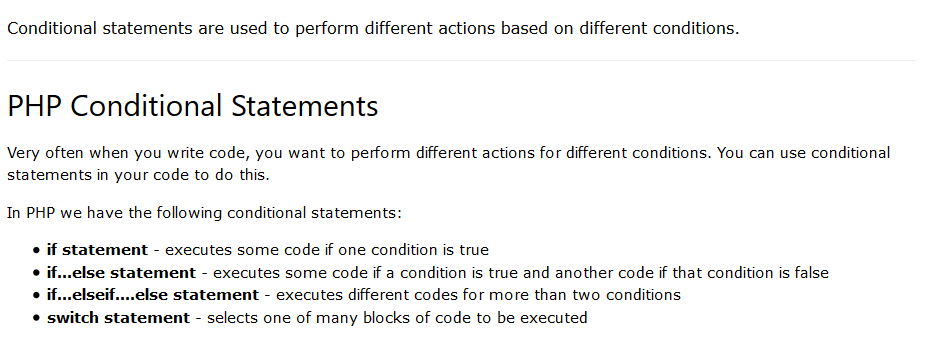
**Syntax**

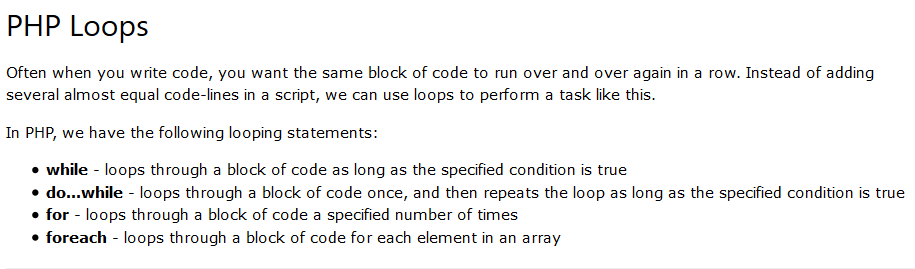
define(*name*, *value*, *case-insensitive*)

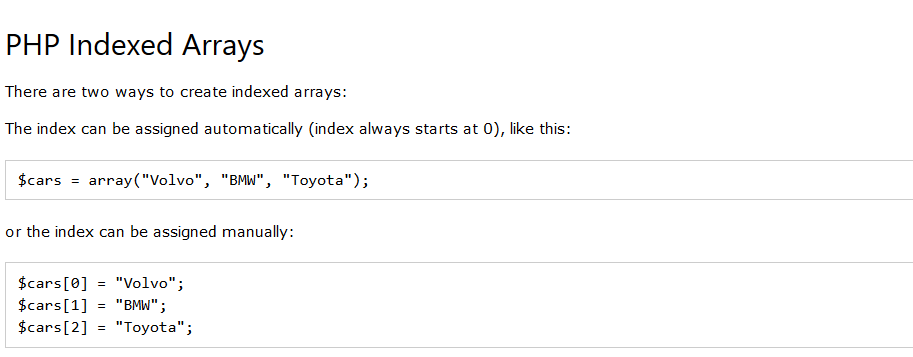
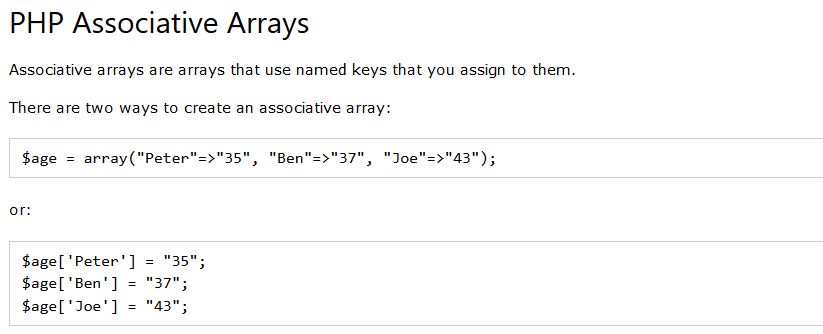
* Parameters:
* *name*: Specifies the name of the constant
* *value*: Specifies the value of the constant
* *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

**OPERATORS**

* Operators are used to perform operations on variables and values.
* PHP divides the operators in the following groups:
* Arithmetic operators
* Assignment operators
* Comparison operators
* Increment/Decrement operators
* Logical operators
* String operators
* Array operators
* Conditional Statements







**Sort Functions For Arrays**

* sort() - sort arrays in ascending order
* rsort() - sort arrays in descending order
* asort() - sort associative arrays in ascending order, according to the value
* ksort() - sort associative arrays in ascending order, according to the key
* arsort() - sort associative arrays in descending order, according to the value
* krsort() - sort associative arrays in descending order, according to the key
* PHP Programs

1. **Aim : To write a PHP program to print prime numbers from 1-50**

**Program:**

<?php

for($i=1;$i<51;$i++)

{

$fact=0;

for($j=1;$j<=$i;$j++)

{

if($i%$j==0)

{

$fact++;

}

}

if($fact==2)

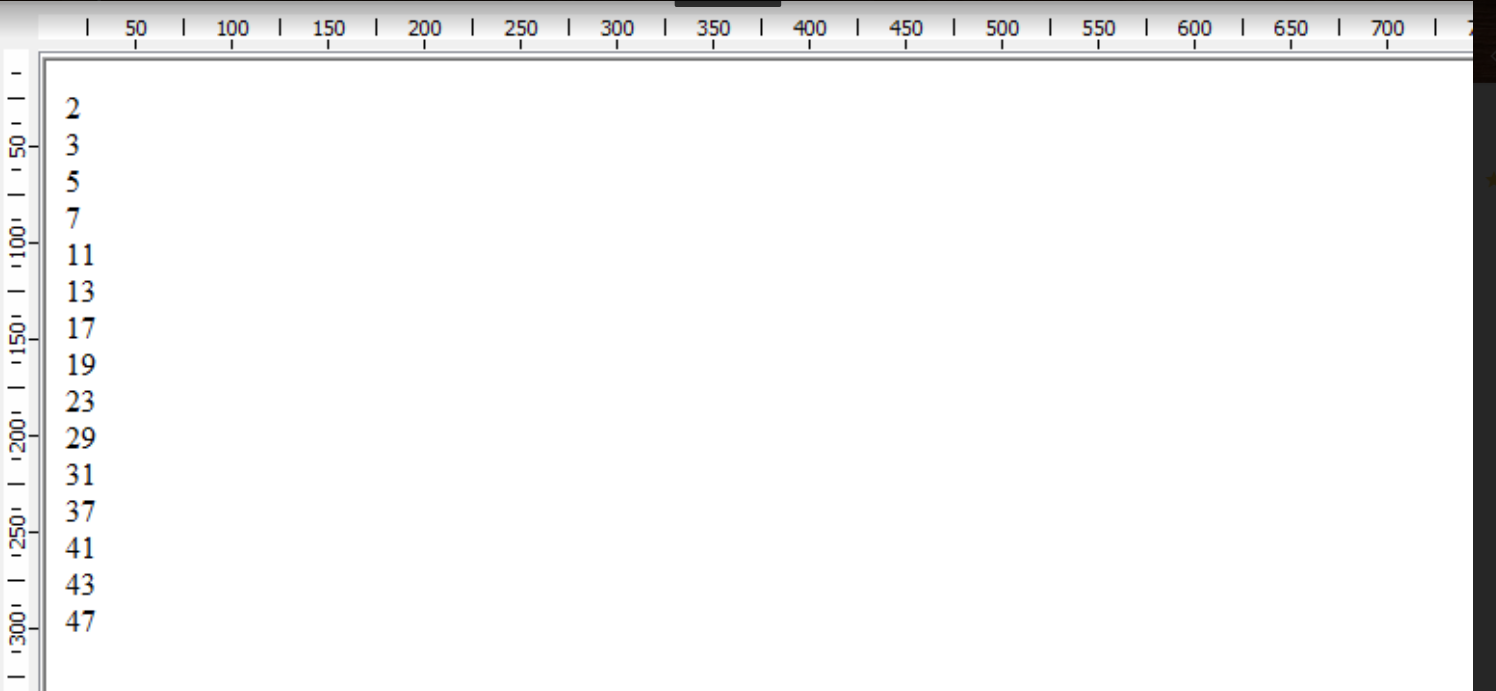
{

echo $i.'<br>';

}

}

**Output :**



1. **a) Aim : To write a PHP program to find length of string**

**Program:**

<?php

$str1="hello ";

$str2="hey , this is me";

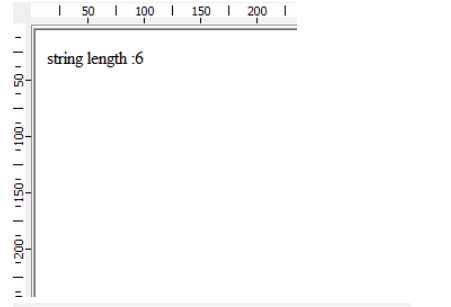
$len=strlen($str1);

echo("string length :");

echo($len."<br>");

?>

**Output:**



**b) Aim : To write a PHP program to count number of words in a given string .**

**program:**

<?php

$str2="hey , this is me";

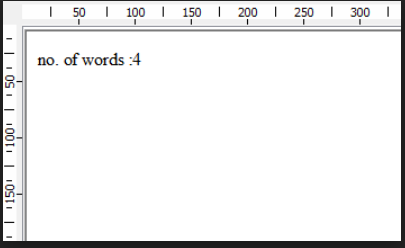
echo("no. of words :");

$w=str\_word\_count($str2);

echo($w);

?>

**Output:**



1. **Aim : To write a PHP program to reverse a string**

**Program:**

<?php

$str1="hello ";

$str2="hey , this is me";

$str1=$str1.$str2;

echo("string is : ");

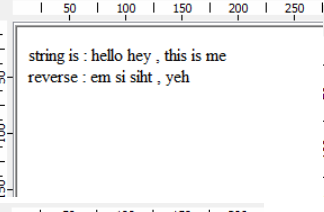
echo($str1."<br>");

echo("reverse : ");

echo(strrev($str2)."<br>");

?>

**Output:**



1. **Aim : To write a PHP program to search for a specific string**

**Program:**

<?php

$str1="hello ";

$str2="hey , this is me";

if(strpos($str2,"welcome")=== false)

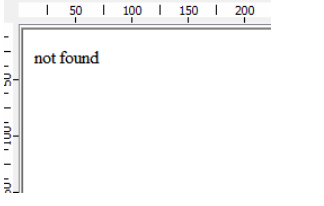
{

echo("not found");

}

?>

**Output:**



1. **Aim : To write a PHP program to merge 2 arrays and sort them as numbers in descending order.**

**Program:**

<?php

$a=array(1,2,3,4);

$b=array(5,6,7,8);

$c=count($b);

$d=count($a);

for($i=0;$i<$c;$i++)

{

$a[$d+$i]=$b[$i];

}

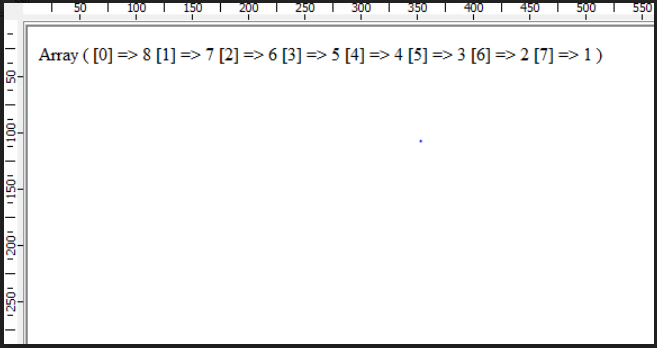
sort($a);

$a=array\_reverse($a);

print\_r($a);

?>

**Output :**



1. **Aim : Write a PHP program that copies contents from one file and writes into another file**

**Program:**

<?php

$fp=fopen("rig.txt","r");

if($fp == false)

{

echo("couldnt open file"."<br>");

exit();

}

$fsize=filesize("file1.txt");

$fdata=fread($fp,$fsize);

$fp2=fopen("file2.txt","w");

if($fp2== false)

{

echo("couldnt create file"."<br>");

exit();

}

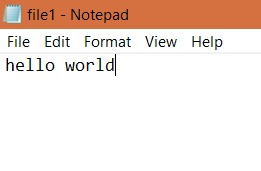
fwrite($fp2,$fdata);

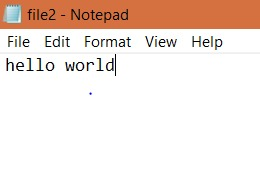
echo("contents written");

fclose($fp2)

?>

**Output:**





**PYTHON**

**CONTEXT**

You have probablry used computers to do all sorts of useful and interesting things. In each application, the computer responds in different ways to your input, from the keyboard, mouse or a file. Still the underlying operations are determined by the design of the program you are given. In this set of tutorials you will learn to write your own computer programs, so you can give the computer instructions to react in the way you want.

**WHY PYTHON….?**

There are many high-level languages. The language you will be learning is Python. Python is one of the easiest languages to learn and use, while at the same time being very powerful: It is used by many of the most highly productive professional programmers. A few of the places that use Python extensively are Google, the New York Stock Exchange, Industrial Light and Magic, .... Also Python is a free language! If you have your own computer, you can download it from the Internet....

**OBTAINING PYTHON FOR YOUR COMPUTER……**

If you are not sure whether your computer already has Python, continue to Section 1.2.2, and give it a try. If it works, you are all set. If you do need a copy of Python, go to the Downloads page linked to http://www.python.org. Be careful to choose the version for your operating system and hardware. Chosse a stable version, 3.1 or later. Do not choose a version 2.X, which is incompatible. (Version 2.6 is described in an older version of this tutorial.)

Windows : You just need to execute the installer, and interact enough to agree to all the default choices. Python works in Windows as well as on Apples and in the free operating system Linux.

OS X : Double-click on the installer. Find and run the MacPython.mpkg that is inside. Follow the defaults for installation.

Linux : Python is generally installed, though Idle is not always installed. Look for something like ’idlepython’ (the name in the Ubuntu distribution).

**LOW LEVEL AND HIGH LEVEL COMPUTER OPERATIONS…….**

First let us place Python programming in the context of the computer hardware. At the most fundamental level in the computer there are instructions built into the hardware. These are very simple instructions, peculiar to the hardware of your particular type of computer. The instructions are designed to be simple for the hardware to execute, not for humans to follow. The earliest programming was done with such instructions. If was difficult and error-prone. A major advance was the development of higher-level languages and translators for them. Higher-level languages allow computer programmers to write instructions in a format that is easier for humans to understand. For example

z = x+y

is an instruction in many high-level languages that means something like:

(1) Access the value stored at a location labeled x

(2) Calculate the sum of this value and the value stored at a location labeled y

(3) Store the result in a location labeled z.

No computer understands the high-level instruction directly; it is not in machine language. A special program must first translate instructions like this one into machine language. This one high-level instruction might be translated into a sequence of three machine language instructions corresponding to the three step description above:

0000010010000001

0000000010000010

0000010110000011

Obviously high-level languages were a great advance in clarity! If you follow a broad introduction to computing, you will learn more about the layers that connect low-level digital computer circuits to high-level languages.

**INTRODUCTION TO TYPES AND FUNCTIONS……**

Python directly recognizes a variety of types of data. Here are a few:

Numbers: 3, 6, -7, 1.25

Character strings: ’hello’, ’The answer is: ’

Lists of objects of any type: [1, 2, 3, 4], [’yes’, ’no’, ’maybe’]

A special datum meaning nothing: None

Python has large collection of built-in functions that operate on different kinds of data to produce all kinds of results. To make a function do its action, parentheses are required. These parentheses surround the parameter or parameters, as in a function in algebra class.

The general syntax to execute a function is functionName ( parameters )

One function is called type, and it returns the type of any object. The Python Shell will evaluate functions. In the Shell the last line should look like >>> Continuing on the same line enter

type(7)

Always remember to end with the Enter key. After the Shell responds, you should see something like >>>

type(7) >>>.

For the rest of this section, at the >>> prompt in the Python Shell, individually enter each line below that is set off in typewriter font. So next enter

type(1.25)

Note the name in the last result is float, not real or decimal, coming from the term “floating point”, for reasons that will be explained later, in Section 1.14.1. Enter

type(’hello’)

In your last result you see another abbreviation: str rather than string. Enter

type([1, 2, 3])

Strings and lists are both sequences of parts (characters or elements). We can find the length of that sequence with another function with the abbreviated name len. Try both of the following, separately, in the Shell:

len([2, 4, 6])

len(’abcd’)

Some functions have no parameters, so nothing goes between the parentheses. For example, some types serve as no-parameter functions to create a simple value of their type. Try

list()

Functions may also take more than one parameter. Try max(5, 11, 2)

Some of the names of types serve as conversion functions (where there is an obvious meaning for the conversion). Try each of the following, one at a time, in the Shell:

str(23)

int(’125’)

An often handy Shell feature: an earlier Shell line may to copied and edited by clicking anywhere in the previously displayed line and then pressing Enter. For instance you should have entered several lines starting with len. click on any one, press Enter, and edit the line for a different test.

**ARITHEMETIC OPERATIONS……**

**Addition and Subtraction:**

We start with the integers and integer arithmetic, not because arithmetic is exciting, but because the symbolism should be mostly familiar. Of course arithmetic is important in many cases, but Python is probably more often used to manipulate text and other sorts of data, as in the sample program

Python understands numbers and standard arithmetic. For the whole section on integer arithmetic, where you see a set-off line in typewriter font, type individual lines at the >>> prompt in the Python Shell. Press Enter after each line to get Python to respond:

77 2 + 3 5 - 7

Python should evaluate and print back the value of each expression. Of course the first one does not require any calculation. It appears the shell just echoes back what you printed. Do note that the line with the value produced by the shell does not start with >>> and appears at the left margin. Hence you can distinguish what you type (after the “>>>” prompt) from what the computer responds.

The Python Shell is an interactive interpreter. As you can see, after you press Enter, it is evaluating the expression you typed in, and then printing the result automatically. This is a very handy environment to check out simple Python syntax and get instant feedback. For more elaborate programs that you want to save, we will switch to an Editor Window later

**Multiplication, Parentheses, and Precedence:**

2 x 3

You should get your first syntax error. The ’x’ should have become highlighted, indicating the location where the Python interpreter discovered that it cannot understand you: Python does not use x for multiplication as you may have done in grade school. The x can be confused with the use of x as a variable (more on that later). Instead the symbol for multiplication is an asterisk ’\*’. Enter each of the following. You may include spaces or not. The Python interpreter can figure out what you mean either way.

2\*5

2 + 3 \* 4

If you expected the last answer to be 20, think again: Python uses the normal precedence of arithmetic operations: Multiplications and divisions are done before addition and subtraction, unless there are parentheses.

(2+3)\*4

2 \* (4 - 1)

**Division and Remainders:**

If you think about it, you learned several ways to do division. Eventually you learned how to do division resulting is a decimal.

5/2

14/4

As you saw in the previous section, numbers with decimal points in them are of type float in Python. They are discussed more

In the earliest grades you would say “14 divided by 4 is 3 with a remainder of 2”. The problem here is that the answer is in two parts, the integer quotient 3 and the remainder 2, and neither of these results is the same as the decimal result. Python has separate operations to generate each part. Python uses the doubled division symbol // for the operation that produces just the integer quotient, and introduces the symbol % for the operation of finding the remainder. Try each in the Shell

14/4

14//4

14%4

**STRINGS……**

**String Delimiters:**

A string in Python is a sequence of characters. For Python to recognize a sequence of characters, like hello, as a string, it must be enclosed in quotes to delimit the string. For this whole section on strings, continue trying each set-off line of code in the Shell. Try

"hello"

**String Concatenation:**

Strings also have operation symbols. Try in the Shell (noting the space after very):

’very ’ + ’hot’

The plus operation with strings means concatenate the strings. Python looks at the type of operands before deciding what operation is associated with the +. Think of the relation of addition and multiplication of integers, and then guess the meaning of

3\*’very ’ + ’hot’

Were you right? The ability to repeat yourself easily can be handy.

* Python Programs

**1 a) Aim : To write a program to find roots of a quadratic equation.**

**Program:**

import cmath

print("ax^2+bx+c")

a=int(input("Enter a :"))

b=int(input("Enter b :"))

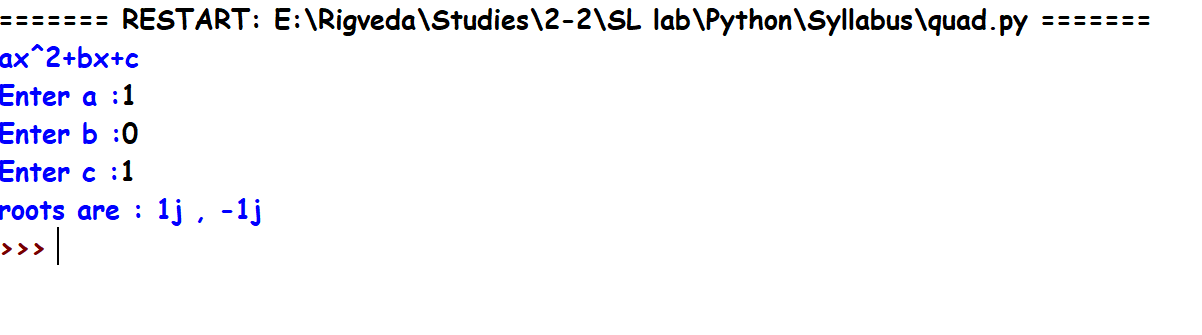
c=int(input("Enter c :"))

root1=(-1\*b + cmath.sqrt(b\*\*2-4\*a\*c))/2\*a

root2=(-1\*b - cmath.sqrt(b\*\*2-4\*a\*c))/2\*a

print("roots are :",root1,',',root2)

**Output:**



1 b)**Aim : To find factorial of a given number.**

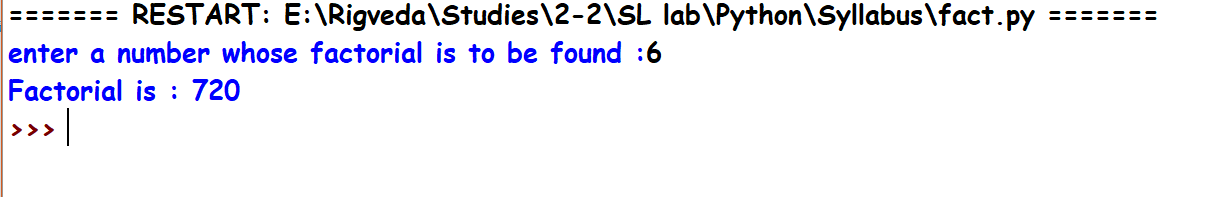
**Program:** n=int(input("enter a number whose factorial is to be found :"))

fact=1

for i in range(1,n+1):

fact=fact\*I

print("Factorial is :",fact)

Output:

**2) Aim : To generate Fibonacci series.**

**Program:**

n=int(input("enter number of terms to be printed in fibonacci series :"))

print(0,1,sep=(','),end=(','))

pres=1;

prev=0;

temp=0

i=0

while(i<n-2):

temp=pres

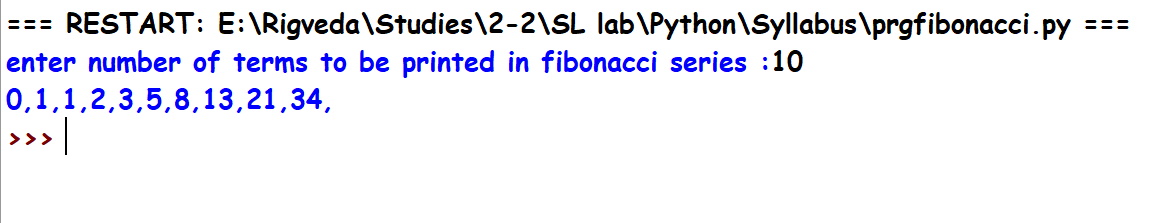
pres=pres+prev

prev=temp

print(pres,end=',')

i+=1

**Output:**



1. **Aim : To make a simple calculator.**

**Program:**

op=['+','-','\*','/']

op=['+','-','\*','/']

while(True):

a=float(input("Enter first number :"))

b=float(input("Enter second number :"))

o=input("enter operation to be performed :")

if o not in op:

print("invalid operation")

break

elif o==op[0]:

print("Sum is :",a+b)

elif o==op[1]:

print("Difference is :",a-b)

elif o==op[2]:

print("Product is :",a\*b)

else:

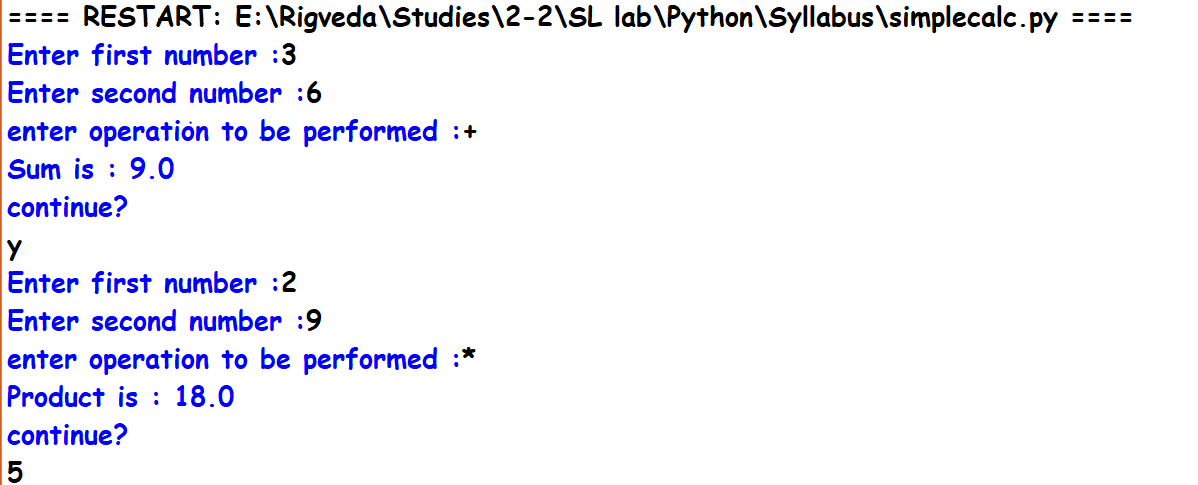
print("Quotient is :",a/b)

print("continue?")

if(input()!="y"):

break

**Output:**



1. **Aim : To sort words in alphabetical order.**

**Program:**

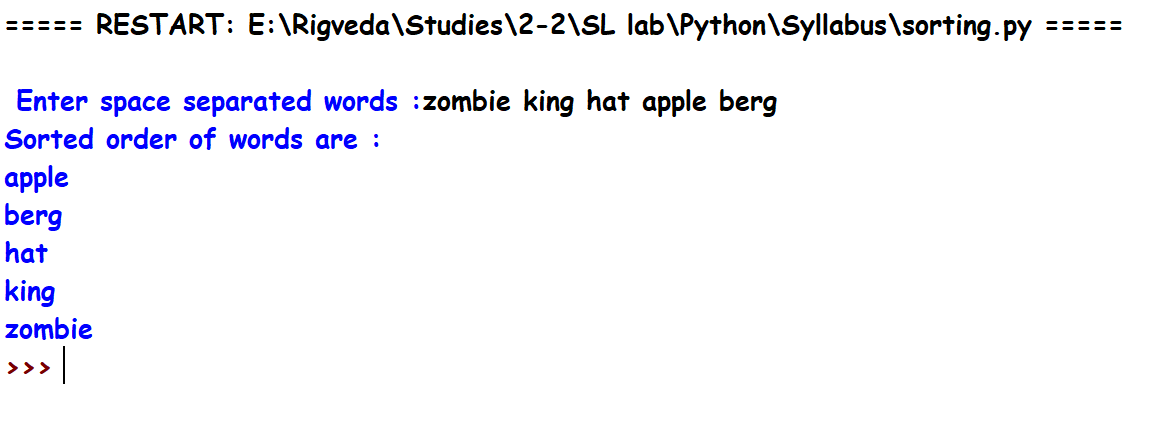
lst=sorted(input("\n Enter space separated words :").split())

print("Sorted order of words are :")

for i in lst:

print (i)

**Output:**



1. **Aim : To add 2 matrices.**

**Program:**

m=input("enter number of rows of matrix 1 :")

n=input("enter number of columns of matrix 1 :")

p=input("enter number of rows of matrix 2 :")

q=input("enter number of columns of matrix 2 :")

if m!=p or n!=q:

print("cannot perform addition")

exit

else:

rows=int(m)

col=int(n)

m1=[]

m2=[]

m3=[]

print('taking values of first matrix :')

for i in range(0,rows):

print('enter elements of row',i+1,':')

lst=[]

for j in range(0,col):

print("element",j+1)

num=int(input())

lst.append(num)

m1.append(lst)

print(m1)

print('taking values of second matrix :')

for i in range(0,rows):

print('enter elements of row',i+1,':')

lst=[]

for j in range(0,col):

print("element",j+1)

num=int(input())

lst.append(num)

m2.append(lst)

print(m2)

for i in range(0,rows):

lst=[]

for j in range(0,col):

lst.append((m1[i][j])+(m2[i][j]))

m3.append(lst)

print('sum is ',m3)

**Output:**

