UNIT – I

1. **What is perl?**

Ans: *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

2. Define Script.

Ans: A scripting or script language is a programming language that supports scripts: programs written for a special run-time environment that automate the execution of tasks that could alternatively be executed one-by-one by a human operator

3. **What are the PERL Data types?**

Ans: We have learnt that Perl has the following three basic data types −

- Scalars
- Arrays
- Hashes

4. What is an array in Perl?

Ans: An array is a variable that stores an ordered list of scalar values. ... In Perl, List and Array terms are often used as if they're interchangeable. But the list is the data, and the array is the variable.

5. What is subroutine ?

Ans: In computer programming, a *subroutine* is a sequence of program instructions that perform a specific task .

6. What is list and give syntax.

In *Perl*, people use term *list* and array interchangeably, however there is a difference. The *list* is the data (ordered collection of scalar values) and the array is a variable that holds the *list*. How to define array? Arrays are prefixed with @ sign. This is how you define an array - @friends = ("Ajeet", "Chaitanya")

7. Write a syntax to add two arrays together in Perl.

Ans:

```
Syntax:@Array1= (a1,a2,a3);
            @array2 = (ba,b2,b3);
@arr1 = ( 1, 0, 0, 0, 1 );
@arr2 = ( 1, 1, 0, 1, 1 );
for ($i=0;$i<scalar @arr1;$i++){
   print $arr[$i] + $arr2[$i] ."\n";
}
```

8. How many types of operators are used in Perl?

Simple answer can be given using the expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. Perl language supports many operator types, but following is a list of important and most frequently used operators −

- Arithmetic Operators
- Equality Operators
- Logical Operators
- Assignment Operators
- Bitwise Operators
- Logical Operators
- Quote-like Operators
- Miscellaneous Operators

Lets have a look at all the operators one by one.

Perl Arithmetic Operators

Assume variable $a holds 10 and variable $b holds 20, then following are the Perl arithmatic operators −

Show Example

Sr.No.	Operator & Description
1	**+ (Addition)** Adds values on either side of the operator **Example** − $a + $b will give 30
2	**- (Subtraction)** Subtracts right hand operand from left hand operand **Example** − $a - $b will give -10
3	*** (Multiplication)** Multiplies values on either side of the operator **Example** − $a * $b will give 200
4	**/ (Division)** Divides left hand operand by right hand operand **Example** − $b / $a will give 2
	% (Modulus) 5. Divides left hand operand by right hand operand and returns remainder **Example** − $b % $a will give 0
	**** (Exponent)**
6	Performs exponential (power) calculation on operators **Example** − $a**$b will give 10 to the power 20
	!= (not equal to) Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.

Example − ($a != $b) is true.

<=>

3 Checks if the value of two operands are equal or not, and returns -1, 0, or 1 depending on whether the left argument is numerically less than, equal to, or greater than the right argument.

Example − ($a <=> $b) returns -1.

> (greater than)

4 Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.

Example − ($a > $b) is not true.

< (less than)

5 Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.

Example − ($a < $b) is true.

>= (greater than or equal to)

6 d is greater than or equal to the value of right operand, if yes then condition becomes

Example − ($a >= $b) is not true.

<= (less than or equal to)

Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

Example − ($a <= $b) is true.

9. List out the numeric and string built in functions available in Perl .

Ans: lc, uc, length

There are a number of simple functions such as **lc** and **uc** to return the lower case and upper case versions of the original string respectively. Then there is **length** to return the number of characters in the given string.

```
use strict;

use warnings;

use 5.010;

my $str = 'HeLlo';
```

```
say lc $str;      # hello

say uc $str;      # HELLO

say length $str;  # 5
```

index

Then there is the **index** function. This function will get two strings and return the location of the second string within the first string.

substr

I think the most interesting function in this article is substr. It is basically the opposite of index(). While index() will tell you **where is a given string**, substr will give you the **substring at a given locations**. Normally substr gets 3 parameters. The first one is the string. The second is a 0-based location, also called the **offset**, and the third is the **length** of the substring we would like to get.

10. What are the applications of Modern Scripting?

Ans: 1.CHARACTERISTICS OF SCRIPTING LANGUAGE • Integrated Compile and Run Scripting Language behave as if they were interpreted. They are often an interactive, experimental activity that doesn't fit well with the "edit-compile-link- run" cycle of conventional programming. A few Scripting Languages like Unix Shell and TCL(Tool Command Language) version 7 are indeed implemented as strict interpreters. However most of the languages in current use opt for hybrid technique which involve compiling to an intermediate form which is then interpreted. • Low Overhead and ease of use Based on the usage variables can be declared and initialized . Later focus was on explicit variable declaration, which requires that you declare every variable before you use it. Eg: Dim sum As Integer (or) sum=10; • Enhanced Functionality Most of the languages provide powerful string manipulation based on the use of regular expression(A regular expression is a string of characters that define the pattern or patterns you are viewing.). Some languages support easy access to API or Object model exported by application. 6/13/2016 1Introduction to scripts and scripting

2.CHARACTERISTICS OF SCRIPTING LANGUAGE • Efficiency is not an issue Ease of use is achieved at the expense of efficiency eg: interpretation rather than compiling Focus is not on high performance but on the speed of development together with ability to make changes to meet new requirement. • Interpreted languages also have a simple syntax which, for the user: makes them easy to learn and use. • assumes minimum programming knowledge or experience. • allows complex tasks to be performed in relatively few steps. • allows simple creation and editing in a variety of text editors. • allows the addition of dynamic and interactive activities to web pages. • Also, interpreted languages are generally portable across various hardware and network platforms and scripts can be embedded in standard text documents for added functionality. 6/13/2016 2Introduction to scripts and scripting

3. Web Scripting Processing Web Forms Dynamic Web pages Dynamically generated Web pages • Processing Web Forms

4. DYNAMIC WEB PAGES • DHTML is about creating web pages that reacts to (user)events. • DHTML is about using JavaScript and the HTML DOM to change the style and positioning of HTML elements.

5.Dynamically generated HTML • It is another form of dynamic web page is one in which some or all of the HTML is generated by scripts executed on the server. • A common application is to construct a page whose content is retrieved from a database. • Examples: 1) Microsoft IIS Web Server which implements Active Server Pages(which has scripts in java script or vb script)

6.UNIVERSE OF SCRIPTING LANGUAGES • There is scripting universe, containing multiple overlapping worlds • The original UNIX world of traditional scripting using Perl and Tcl • The Microsoft world of visualbasic and active x controls. • The world of VBA for scripting compound documents. • The world of client-side and server-side web scripting. • Overlapping of them is very complex, as Web scripting can be done in VBScript,JavaScript,Perl or Tcl. • Perl and Tcl are used to implement complex application for large organization. 1. Eg. Tcl has been used to develop major banking system. 2. Perl is used to implement enterprise wide document management system for leading aerospace company.
7.TYPICAL USES OF SCRIPTING LANGUAGES INCLUDING JAVASCRIPT INCLUDE: Image or text rollovers: If the user rolls the mouse over a graphic or hypertext then a text or graphic box will appear: Creating a pop-up window to display information in a separate window from the Web page that triggered it. This is useful if the user requires to perform a simple calculation or consult a calendar for inputting dates. This is achieved by embedding ActiveX controls or Java applets into the script. Validating the content of fields: When filling in forms, each field, especially required fields denoted by an asterisk, are validated for correct input. If the field is left blank or incorrect information entered then a user message will be generated and you may not continue.

8.USES OF SCRIPTING LANGUAGES • Scripting languages are of two kinds: 1.Traditional Scripting 2.Modern Scripting Traditional Scripting:¬ The activities which require traditional scripting include 1. System administration 2. Controlling Remote Applications 3. System and Application extensions 4. Experimental programming 5. Command Line interface Modern Scripting¬ 1. Visual Scripting 2. Scriptable components 3. Client-side and Server-side Web scripting

UNIT – II

1. **What are Finer points of looping?**

Ans: Perl programming language provides the following types of **loop** to handle the **looping** requirements. Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the **loop** body. Repeats a statement or group of statements until a given condition becomes true.

2. **what are pack and unpack?**

Ans: *pack* and unpack are two functions for transforming data according to a user-defined template, between the guarded way *Perl* stores values and some well-defined representation as might be required in the environment of a *Perl* program.

3. **How do you connect to database in Perl?**

Ans: I won't embarrass the company in question by giving away the name of the database, so let's call this SQL client sqlstar. Very soon I was writing horrendous Perl programs that did things like:

```
my @rows = `sqlstar $database "select * from $table"`;
for (@rows) {
   ...
}
```

Of course, things got terribly confused when we had complex where clauses, and needed to escape metacharacters, and …

```
my @rows = `sqlstar $database "select * from $table where
```

```
$row LIKE \"%pattern%\""`;
```

The code rapidly got ugly, error-prone, and dangerously unsafe. If someone had decided to search for a value with a double quote in it, I don't know where we'd have been. But for the most part it worked, so nobody really worried about it.

4. **What is PHP?**

Ans: *PHP* (recursive acronym for *PHP*: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

5. **How to Authenticate Files in PHP.**

Ans:

```
<?php

$username = $_POST['username'];
$password = $_POST['password'];
$matchFound = false;

$fileArray = array();

$myFile = fopen("customers.txt", 'r');

   while(!feof($myFile) ) //reads  all lines of text file
{
$fileArray[] = fgets($myFile);
}

fclose($myFile);

for($lineCounter = 0; $lineCounter < count($fileArray); $lineCounter++)
   {
    $exploadedLine = explode("\t",$fileArray[$lineCounter]);

     if(trim($exploadedLine[2] == $username && trim($exploadedLine[3])
          == $password))
   {

       echo 'Match Found!';
$matchFound = true;
break;

       }

   elseif($matchFound == false)
   {
echo 'Invalid Login';
   }

 }
```

6. **What are the PHP Encryption Function?**

Ans: The crypt() **function** returns a hashed string using DES, Blowfish, or MD5 algorithms. This **function** behaves different on different operating systems. ... There are some constants that are used together with the crypt() **function**. The value of these constants are set by **PHP** when it is installed.

7. Discuss various File inclusion statements in PHP

Ans: There are two PHP functions which can be used to included one PHP file into another PHP file.

- The include() Function
- The require() Function

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required then instead of changing thousand of files just change included file.

The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **include()** function generates a warning but the script will continue execution.

Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

```
<a href="http://www.tutorialspoint.com/index.htm">Home</a> -
<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -
<a href="http://www.tutorialspoint.com/perl">PERL</a> <br />
```

Now create as many pages as you like and include this file to create header. For example now your test.php file can have following content.

```
<html>
  <body>

    <?php include("menu.php"); ?>
    <p>This is an example to show how to include PHP file!</p>

  </body>
</html>
```

It will produce the following result −

The require() Function

The require() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **require()** function generates a fatal error and halt the execution of the script.

So there is no difference in require() and include() except they handle error conditions. It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

You can try using above example with require() function and it will generate same result. But if you will try following two examples where file does not exist then you will get different results.

```
<html>
  <body>

    <?php include("xxmenu.php"); ?>
    <p>This is an example to show how to include wrong PHP file!</p>

  </body>
</html>
```

This will produce the following result −

This is an example to show how to include wrong PHP file!

Now lets try same example with require() function.

```
<html>
  <body>

     <?php require("xxmenu.php"); ?>
     <p>This is an example to show how to include wrong PHP file!</p>

  </body>
</html>
```

This time file execution halts and nothing is displayed.

8.Expalin about mycrypt packages.

Ans: his is an interface to the mcrypt library, which supports a wide variety of block algorithms such as DES, TripleDES, Blowfish (default), 3-WAY, SAFER-SK64, SAFER-SK128, TWOFISH, TEA, RC2 and GOST in CBC, OFB, CFB and ECB cipher modes. Additionally, it supports RC6 and IDEA which are considered "non-free". CFB/OFB are 8bit by default.

These functions work using » mcrypt. To use it, download *libmcrypt-x.x.tar.gz* from » http://mcrypt.sourceforge.net/ and follow the included installation instructions.

libmcrypt Version 2.5.6 or greater is required.

Windows users will find the library is the PHP 5.2 Windows binaries release. PHP 5.3 Windows binaries uses the static version of the MCrypt library, no DLL are needed.

If you linked against libmcrypt 2.4.x or higher, the following additional block algorithms are supported: CAST, LOKI97, RIJNDAEL, SAFERPLUS, SERPENT and the following stream ciphers: ENIGMA (crypt), PANAMA, RC4 and WAKE. With libmcrypt 2.4.x or higher another cipher mode is also available; nOFB.

Some of functions are:

Mcrypt_cbc

Mcrypt_cfb

Mcrypt_ecb .

9. Explain about various data types in PHP.

Ans: String,Integer,Float (floating point numbers - also called double),Boolean,Array.

Object,NULL,Resource.

10. How to validate a form in PHP? Explain it with an Example.

Ans: **What is Validation ?**

Validation means check the input submitted by the user. There are two types of validation are available in PHP. They are as follows −

- **Client-Side Validation** − Validation is performed on the client machine web browsers.
- **Server Side Validation** − After submitted by data, The data has sent to a server and perform validation checks in server machine.

Some of Validation rules for field

Field	Validation Rules
Name	Should required letters and white-spaces
Email	Should required **@** and **.**
Website	Should required a valid URL
Radio	Must be selectable at least once
Check Box	Must be checkable at least once
Drop Down menu	Must be selectable at least once

Valid URL

Below code shows validation of URL

```
$website = input($_POST["site"]);

if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

Above syntax will verify whether a given URL is valid or not. It should allow some keywords as https, ftp, www, a-z, 0-9,..etc..

Valid Email

Below code shows validation of Email address

```
$email = input($_POST["email"]);

if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid format and please re-enter valid email";
}
```

Above syntax will verify whether given Email address is well-formed or not.if it is not, it will show an error message.

Example

Example below shows the form with required field validation

```
<html>

  <head>
    <style>
      .error {color: #FF0000;}
    </style>
  </head>

  <body>
    <?php
      // define variables and set to empty values
      $nameErr = $emailErr = $genderErr = $websiteErr = "";
      $name = $email = $gender = $comment = $website = "";

      if ($_SERVER["REQUEST_METHOD"] == "POST") {
        if (empty($_POST["name"])) {
          $nameErr = "Name is required";
        }else {
          $name = test_input($_POST["name"]);
        }

        if (empty($_POST["email"])) {
          $emailErr = "Email is required";
        }else {
          $email = test_input($_POST["email"]);

          // check if e-mail address is well-formed
          if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $emailErr = "Invalid email format";
          }
        }

        if (empty($_POST["website"])) {
```

```
      $website = "";
    }else {
      $website = test_input($_POST["website"]);
    }

    if (empty($_POST["comment"])) {
      $comment = "";
    }else {
      $comment = test_input($_POST["comment"]);
    }

    if (empty($_POST["gender"])) {
      $genderErr = "Gender is required";
    }else {
      $gender = test_input($_POST["gender"]);
    }
  }

  function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
  }
?>

<h2>Absolute classes registration</h2>

<p><span class = "error">* required field.</span></p>

<form method = "post" action = "<?php
  echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  <table>
    <tr>
      <td>Name:</td>
      <td><input type = "text" name = "name">
        <span class = "error">* <?php echo $nameErr;?></span>
      </td>
    </tr>

    <tr>
      <td>E-mail: </td>
      <td><input type = "text" name = "email">
        <span class = "error">* <?php echo $emailErr;?></span>
      </td>
    </tr>

    <tr>
      <td>Time:</td>
      <td> <input type = "text" name = "website">
        <span class = "error"><?php echo $websiteErr;?></span>
      </td>
    </tr>
```

```
      <tr>
        <td>Classes:</td>
        <td> <textarea name = "comment" rows = "5" cols = "40"></textarea></td>
      </tr>

      <tr>
        <td>Gender:</td>
        <td>
          <input type = "radio" name = "gender" value = "female">Female
          <input type = "radio" name = "gender" value = "male">Male
          <span class = "error">* <?php echo $genderErr;?></span>
        </td>
      </tr>

      <td>
        <input type = "submit" name = "submit" value = "Submit">
      </td>

    </table>

  </form>

  <?php
    echo "<h2>Your given values are as:</h2>";
    echo $name;
    echo "<br>";

    echo $email;
    echo "<br>";

    echo $website;
    echo "<br>";

    echo $comment;
    echo "<br>";

    echo $gender;
  ?>

 </body>
</html>
```

UNIT III

1. **How to set cookies in PHP?**

Ans: Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users −

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

2. **What are the features of PHP?**

- **Ans:** Features of php. It is most popular and frequently used world wide scripting language, the main reason of popularity is; It is open source and very simple. ...
- Simple. ...
- Interpreted. ...
- Faster. ...
- Open Source. ...
- Platform Independent. ...
- Case Sensitive.

3. **Explain about Arrays in PHP.**

   Ans: An **array** is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an **array** of 100 length. ... Associative **array** − An **array** with strings as index.

4. How to upload error messages in PHP?

   Ans: The move_uploaded_file() function in PHP. I've managed to successfully troubleshoot a problem I was encountering, but I would like to be able to get the actual content of the warnning or error message. According to php.net (http://php.net/manual/en/function.move-uploaded-file.php) the move_uploaded_file() function returns FALSE and a warning on failure. I want the actual content of the warning, such as "failed to open stream: Permission denied..." so that I can record which errors are occurring. Is there a way to do this?

5. **How to Move an uploaded file in PHP?**

   Ans: <form enctype="multipart/form-data" action="*upload.php*" method="POST"> <input type="hidden" name="MAX_FILE_SIZE" value="512000" /> Send this *file*: <input name="userfile" ... If filename is a valid *upload file*, but cannot be *moved* for some reason, no action will occur, and move_uploaded_file() will return FALSE

6. **Discuss about string interpolation in PHP.**

Ans:

We want to include the results of executing a function or expression within a string.

Use the string concatenation operator (.) when the value you want to include can't be inside the string:

```
print 'You have '.($_REQUEST['boys'] + $_REQUEST['girls']).' children.';
print "The word '$word' is ".strlen($word).' characters long.';
print 'You owe '.$amounts['payment'].' immediately';
print "My circle's diameter is ".$circle->getDiameter().' inches.';
```

You can put variables, object properties, and array elements (if the subscript is unquoted) directly in double-quoted strings:

```
print "I have $children children.";
print "You owe $amounts[payment] immediately.";
print "My circle's diameter is $circle->diameter inches.";
```

Direct interpolation or using string concatenation also works with heredocs. Interpolating with string concatenation in heredocs can look a little strange because the heredoc delimiter and the string concatenation operator have to be on separate lines:

```
print <<< END
Right now, the time is
END
. strftime('%c') . <<< END
 but tomorrow it will be
END
. strftime('%c',time() + 86400);
```

Also, if you're interpolating with heredocs, make sure to include appropriate spacing for the whole string to appear properly. In the previous example, "Right now the time" has to include a trailing space, and "but tomorrow it will be" has to include leading and trailing spaces.

7. Explain with an example the creation of a function in PHP.

Ans: PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you −

- Creating a PHP Function
- Calling a PHP Function

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Please refer to PHP Function Reference for a complete set of useful functions.

Creating PHP Function

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below −

```
<html>

   <head>
      <title>Writing PHP Function</title>
   </head>
```

```
   <body>

      <?php
         /* Defining a PHP Function */
         function writeMessage() {
            echo "You are really a nice person, Have a nice time!";
         }

         /* Calling a PHP Function */
         writeMessage();
      ?>

   </body>
</html>
```

8. Discuss scalar and compound data types. Discuss File Handling Functions.

Ans:

We are going to be studying abstract data types and linked data structures during the course of this Unit. So what do these terms mean?

Data Type

A data type (in computing terms) is a set of data values having predefined characteristics. You will have encountered data types during computer programming classes. Example data types would be integer, floats, string and characters. Generally in all programming languages we have a limited number of such data types. The range of values that can be stored in each of these data types is defined by the language and the computer hardware that you are using the high level language on.

These basic data types are also known as the primitive data types.

In addition to this all data types can be broken up into:

Scalar Data types

A simple (scalar) data type consists of a collection of ordered values and a set of operations that can be performed on those values. The C, C++ and Java programming languages refer to scalar types as primitive data types.

A scalar variable can hold only one piece of data at a time. So in C, C++ and Java scalar data types include int, char, float and double, along with others. Scalar variables of the same type can be arranged into ascending or descending order based on the value.

Structured data types

Structured data types hold a collection of data values. This collection will generally consist of the primitive data types. Examples of this would include arrays, records (structs), classes and files. These data types, which are created by programmers, are extremely important and are the building block of data structures. So what exactly is a data structure?

9. What type of inheritance that PHP supports?

Ans: Generally, inheritance has three types, *single*, *multiple* and *multi-level* inheritance. But, PHP supports *single* inheritance and multi-level inheritance. That means the subclass will be derived from

a single parent class. Even though PHP is not supporting any multiple inheritance, we can simulate it by using PHP interfaces.

In PHP, inheritance can be done by using *extends* keyword, meaning that, we are extending the derived class with some additional properties and methods of its parent class. The syntax for inheriting a class is as follows.

10. List out the predefined classes in PHP?

Ans: Directory
stdClass
__PHP_Incomplete_Class
exception
php_user_filter

UNIT-IV

TCL stands for ―Tool Command Language‖ and is pronounced ―tickle‖; is a simple scripting language for controlling and extending applications. TCL is a radically simple open-source interpreted programming language that provides common facilities such as variables, procedures, and control structures as well as many useful features that are not found in any other major language. TCL runs on almost all modern operating systems such as Unix, Macintosh, and Windows (including Windows Mobile). While TCL is flexible enough to be used in almost any application imaginable, it does excel in a few key areas, including: automated interaction with external programs, embedding as a library into application programs, language design, and general scripting. However, TCL is a programming language in its own right, which can be roughly described as a cross-breed between

- LISP/Scheme (mainly for its tail-recursion capabilities)
- C (control structure keywords, expr syntax) and
- Unix shells (but with more powerful structuring).

1.Explain the structure of TCL Structure?

The TCL language has a tiny syntax - there is only a single command structure, and a set of rules to determine how to interpret the commands. Other languages have special syntaxes for control structures (if, while, repeat...) - not so in TCL. All such structures are implemented as commands. There is a runtime library of compiled ‘C‘ routines, and the ‘level‘ of the GUI interface is quite high.

Comments: If the first character of a command is #, it is a comment.

TCL commands: TCL commands are just words separated by spaces. Commands return strings, and arguments are just further words.

command argument command

argument

Spaces are important

expr 5*3 has a single

argument expr 5 * 3 has three

arguments

TCL commands are separated by a new line, or a semicolon, and arrays are indexed by text

set a(a\ text\ index) 4

2.write down the Syntax of tcl?

Syntax is just the rules how a language is structured. A simple syntax of English could say(Ignoring punctuation for the moment) A text consists of one or more sentences A sentence consists of one or more words' Simple as this is, it also describes Tcl's syntax very well - if you say "script" for "text", and "command" for "sentence". There's also the difference that a Tcl word can again contain a script or a command.

In Tcl, "everything is a command" - even what in other languages would be called declaration, definition, or control structure. A command can interpret its arguments in any way it wants - in particular, it can implement a different language, like expression. A word is a string that is a simple word, or one that begins with { and ends with the matching } (braces), or one that begins with " and ends with the matching ". Braced words are not evaluated by the parser. In quoted words, substitutions can occur before the command is called: $[A-Za-z0-9_]+ substitutes the value of the given variable. Or, if the variable name contains characters outside that regular expression, another layer of bracing helps the parser to get it right

puts "Guten Morgen, ${Schuler}!"

If the code would say $Schuler, this would be parsed as the value of variable $Sch, immediately followed by the constant string üler.(Part of) a word can be an embedded script: a string in [] brackets whose contents are evaluated as a script (see above) before the current command is called. In short: Scripts and commands contain words. Words can again contain scripts and commands.

3. Explain the about Variables, Data In TCL

As noted above, by default, variables defined inside a procedure are "local" to that procedure. And, the argument variables of the procedure contain local "copies" of the

argument data used to invoke the procedure. These local variables cannot be seen elsewhere in the script, and they only exist while the procedure is being executed. In the "getAvg" procedure above, the local variables created in the procedure are "n" "r" and "avg". TCL provides two commands to change the scope of a variable inside a procedure, the "global" command and the "upvar" command. The "global" command is used to declare that one or more variables are not local to any procedure. The value of a global variable will persist until it is explicitly changed. So, a variable which is declared with the "global" command can be seen and changed from inside any procedure which also declares that variable with the "global" command.

Variables which are defined outside of any procedure are automatically global by default. The TCL "global" command declares that references to a given variable should be global rather than local. However, the "global" command does not create or set the variable … this must be done by other means, most commonly by the TCL "set" command.

4.Explain Control Flow in Tcl ?

In Tcl language there are several commands that are used to alter the flow of a program. When a program is run, its commands are executed from the top of the source file to the bottom. One by one. This flow can be altered by specific commands. Commands can be executed multiple times. Some commands are conditional. They are executed only if a specific condition is met.

The if command

The if command has the following general form:

if expr1 ?then? body1 elseif expr2 ?then? body2 elseif ... ?else? ?bodyN?

The if command is used to check if an expression is true. If it is true, a body of command(s) is then executed. The body is enclosed by curly brackets.

The if command evaluates an expression. The expression must return a boolean value. In Tcl, 1, yes, true mean true and 0, no, false mean false.

```
!/usr/bin/tclsh
if yes {
   puts "This message is always shown"
}
```

In the above example, the body enclosed by { } characters is always executed.

```
#!/usr/bin/tclsh
if true then {
   puts "This message is always shown"
}
```

The then command is optional. We can use it if we think, it will make the code more clear. We can use the else command to create a simple branch. If the expression inside the square brackets following the if command evaluates to false, the command following the else command is automatically executed.

```
#!/usr/bin/tclsh
set sex female
if {$sex == "male"}
 {
   puts "It is a boy"
} else
 {
   puts "It is a girl"
}
```

We have a sex variable. It has "female" string. The Boolean expression evaluates to false and we get "It is a girl" in the console.

```
$ ./girlboy.tcl
```

```
It is a girl
```

We can create multiple branches using the elseif command. The elseif command tests for another condition, if and only if the previous condition was not met. Note that we can use multiple elseif commands in our tests.

```
#!/usr/bin/tclsh
# nums.tcl
puts -nonewline "Enter a number: "
flush stdout
set a [gets stdin]
if {$a < 0}
{
   puts "the number is negative"
```

```
} elseif { $a == 0 }
 {
   puts "the numer is zero"
} else
{
   puts "the number is positive"
}
```

In the above script we have a prompt to enter a value. We test the value if it is a negative number or positive or if it equals to zero. If the first expression evaluates to false, the second expression is evaluated. If the previous conditions were not met, then the body following the else commands would be executed.

$ ./nums.tcl

Enter a number: 2
the number is positive
$ ./nums.tcl
Enter a number: 0
the numer is zero
$ ./nums.tcl
Enter a number: -3
the number is negative
Running the example multiple times.

5. Explain control statements in TCL?

Switch command

The switch command matches its string argument against each of the pattern arguments in order. As soon as it finds a pattern that matches the string it evaluates the following body argument by passing it recursively to the Tcl interpreter and returns the result of that evaluation. If the last pattern argument is default then it matches anything. If no pattern argument matches string and no default is given, then the switch command returns an empty string.

```
#!/usr/bin/tclsh
# switch_cmd.tcl
```

```
puts -nonewline "Select a top level domain name:"
flush stdout
gets stdin domain
switch $domain
{
   us { puts "United States" }
   de { puts Germany }
   sk { puts Slovakia }
   hu { puts Hungary }
   default { puts "unknown" }
}
```

In our script, we prompt for a domain name. There are several options. If the value equals for example to us the "United States" string is printed to the console. If the value does not match to any given value, the default body is executed and unknown is printed to the console.

```
$ ./switch_cmd.tcl
Select a top level domain name:sk
Slovakia
```

We have entered sk string to the console and the program responded with Slovakia.

While command: The while command is a control flow command that allows code to be executed repeatedly based on a given Boolean condition. The while command executes the commands inside the block enclosed by curly brackets. The commands are executed each time the expression is evaluated to true.

```
#!/usr/bin/tclsh
# whileloop.tcl
set i 0
set sum 0
while { $i < 10 }
 {
   incr i
```

```
    incr sum $i
}
puts $sum
```

In the code example, we calculate the sum of values from a range of numbers. The while loop has three parts: initialization, testing, and updating. Each execution of the command is called a cycle.

```
set i 0
```

We initiate the i variable. It is used as a counter.

```
while { $i < 10 }
{
...
}
```

The expression inside the curly brackets following the while command is the second phase, the testing. The commands in the body are executed, until the expression is evaluated to false.

```
incr i
```

The last, third phase of the while loop is the updating. The counter is incremented. Note that improper handling of the while loops may lead to endless cycles.

FOR command When the number of cycles is know before the loop is initiated, we can use the for command. In this construct we declare a counter variable, which is automatically increased or decreased in value during each repetition of the loop.

```
#!/usr/bin/tclsh
for {set i 0} {$i < 10} {incr i}
 {
   puts $i
}
```

In this example, we print numbers 0..9 to the console.

```
for {set i 0} {$i < 10} {incr i}
 {
   puts $i
}
```

There are three phases. First, we initiate the counter i to zero. This phase is done only once. Next comes the condition. If the condition is met, the command inside the for block is executed. Then comes the third phase; the counter is increased. Now we repeat phases 2 and 3 until the condition is not met and the for loop is left. In our case, when the counter i is equal to 10, the for loop stops executing.$ ./forloop.tcl

0123456789

Here we see the output of the forloop.tcl script.

The foreach command: The foreach command simplifies traversing over collections of data. It has no explicit counter. It goes through a list element by element and the current value is copied to a variable defined in the construct.

```
#!/usr/bin/tclsh
set planets { Mercury Venus Earth Mars Jupiter Saturn     Uranus Neptune
} foreach planet $planets
 {
   puts $planet
}
```

In this example, we use the foreach command to go through a list of planets.

```
foreach planet $planets
 {
   puts $planet
}
```

The usage of the foreach command is straightforward. The planets is the list that we iterate

through. The planet is the temporary variable that has the current value from the list. The for each command goes through all the planets and prints them to the console.

```
$ ./planets.tcl
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
```

6.What is the Data Structures used in Tcl?

The list is the basic Tcl data structure. A list is simply an ordered collection of stuff; numbers, words, strings, or other lists. Even commands in Tcl are just lists in which the first list entry is the name of a proc, and subsequent members of the list are the arguments to the proc. Lists can be created in several way by setting a variable to be a list of values set lst {{item 1} {item 2} {item 3}} with the split command set lst [split "item 1.item 2.item 3" "."] with the list command. set lst [list "item 1" "item 2" "item 3"] An individual list member can be accessed with the index command. The brief description of these commands is list **?arg1? ?arg2? ... ?argN?**

makes a list of the arguments

split **string ?splitChars?**

Splits the *string* into a list of items wherever the *splitChars* occur in the code. *SplitChars* defaults to being whitespace. Note that if there are two or more *splitChars* then each one will be used individually to split the string. In other words: split "1234567" "36" would return the following list: {12 45 7}.lindex **list index**

Returns the *index*'th item from the list.

Note: lists start from 0, not 1, so the first item is at index 0, the second item is at index 1, and so on.llength **list.**Returns the number of elements in a list.The items in list can be iterated through using the foreach command foreach **varname list body** The foreach command will execute the *body* code one time for each list item in *list*. On each pass, *varname* will contain the value of the next *list* item.In reality, the above form of foreach is the simple form, but the command is quite powerful. It will allow you to take more than one variable at a time from the list: foreach {a b} $listofpairs { ... }. You can

even take a variable at a time from multiple lists! For xample: foreach a $listOfA b $listOfB { ... }

Input / Output Tcl uses objects called channels to read and write data. The channels can be created using the open or socket command. There are three standard channels available to Tcl scripts without explicitly creating them. They are automatically opened by the OS for each new application. They are stdin, stdout and stderr. The standard input, stdin, is used by the scripts to read data. The standard output, stdout, is used by scripts to write data. The standard error, stderr, is used by scripts to write error messages. In the first example, we will work with the puts command. It has the following synopsis:

puts ?-nonewline? ?channelId? string

The channelId is the channel where we want to write text. The channelId is optional. If not specified, the default stdout is assumed.

```
#!/usr/bin/tclsh
puts "Message 1"
puts stdout "Message 2"
puts stderr "Message 3"
```

The puts command writes text to the channel.

```
puts "Message 1"

puts stdout "Message 2"

puts stderr "Message 3"
```

If we do not specify the channelId, we write to stdout by default. This line does the same thing as the previous one. We only have explicitly specified the channelId.

We write to the standard error channel. The error messages go to the terminal by default.

```
$ ./printing.tcl
Message 1
Message 2
Message 3
```

Example output.

Read command: The read command is used to read data from a channel. The optional argument specifies the number of characters to read. If omitted, the command reads all of the data from the channel up to the end.

```
#!/usr/bin/tclsh
set c [read stdin
1] while {$c !=
"q"}
 {
   puts -nonewline "$c"
   set c [read stdin 1]
}
```

The script reads a character from the standard input channel and then writes it to the standard output until it encounters the q character.

set c [read stdin 1]

We read one character from the standard input channel (stdin).

```
while {$c != "q"}
```

We continue reading characters until the q is pressed.

Gets command

The gets command reads the next line from the channel, returns everything in the line up to (but not including) the end-of-line character.

```
#!/usr/bin/tclsh
puts -nonewline "Enter your name: "
flush stdout
set name [gets stdin]
puts "Hello $name"
```

The script asks for input from the user and then prints a message. The puts command is used to print messages to the terminal. The -no newline option suppresses the new line character. Tcl buffers output internally, so characters written with puts may not appear immediately on

the output file or device. The flush command forces the output to appear immediately.

```
puts -no newline "Enter your name: "
flush stdout
set name [gets stdin]
```

The gets command reads a line from a channel.

```
$ ./hello.tcl
Enter your name: Jan
Hello Jan
```

Sample output of the script.

Procedures

A procedure is a code block containing a series of commands. Procedures are called functions in many programming languages. Procedures bring modularity to programs. The proper use of procedures brings the following advantages

- Reducing duplication of code
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code
- Information hiding

There are two basic types of procedures: built-in procedures and user defined ones. The built-in procedures are part of the Tcl core language. For instance, the rand(), sin() and exp() are built-in procedures. The user defined procedures are procedures created with the proc keyword.The proc keyword is used to create new Tcl commands. The term procedures and commands are often used interchangeably. We start with a simple example.

```
#!/usr/bin/tclsh
proc tclver {}
{
   set v [info tclversion]
   puts "This is Tcl version $v"
}
```

```
tclver
```

In this script, we create a simple tclver procedure. The procedure prints the version of Tcl language. proc tclver {}

{

The new procedure is created with the proc command. The {} characters reveal that the procedure takes no arguments.

{

```
 set v [info tclversion]
 puts "This is Tcl version $v"
 }
tclver
```

This is the body of the tclver procedure. It is executed when we execute the tclver command. The body of the command lies between the curly brackets.The procedure is called by specifying its name.

$ ./version.tcl
This is Tcl version 8.6

Sample output.

Procedure arguments: An argument is a value passed to the procedure. Procedures can take one or more arguments. If procedures work with data, we must pass the data to the procedures. In the following example, we have a procedure which takes one argument.

```
#!/usr/bin/tclsh
proc ftc {f}
{
   return [expr $f * 9 / 5 + 32]
}
puts [ftc 100]
puts [ftc 0]
puts [ftc 30]
```

We create a ftc procedure which transforms Fahrenheit temperature to Celsius temperature. The procedure takes one parameter. Its name f will be used in the body of the procedure.

```
proc ftc {f} {
return [expr $f * 9 / 5 + 32]
puts [ftc 100]
```

We compute the value of the Celsius temperature. The return command returns the value to the caller. If the procedure does not execute an explicit return, then its return value is the value of the last command executed in the procedure's body.The ftc procedure is executed. It takes 100 as a parameter. It is the temperature in Fahrenheit. The returned value is used by the puts command, which prints it to the console. Output of the example.

```
$ ./fahrenheit.tcl
212
32
86
```

Next we will have a procedure which takes two arguments.

```
#!/usr/bin/tclsh
proc maximum {x y}
{
      if {$x > $y}
 {
      return $x
   }
 else
 {
          return $y
   }
}
set a 23
set b 32
set val [maximum $a $b]
puts "The max of $a, $b is $val"
```

The maximum procedure returns the maximum of two values. The method takes two arguments.

```
proc maximum {x y}
{
if {$x > $y}
 {
   return $x
}
 else
{
       return $y
}
```

Here we compute which number is greater. We define two variables which are to be compared.

```
set a 23
set b 32
set val [maximum $a $b]
```

We calculate the maximum of the two variables. This is the output of the maximum.tcl script.

$ ./maximum.tcl
The max of 23, 32 is 32

Variable number of arguments A procedure can take and process variable number of arguments. For this we use the special arguments and parameter.

```
#!/usr/bin/tclsh

proc sum {args} {

    set s 0

    foreach arg $args {

        incr s $arg
    }

    return $s
}

puts [sum 1 2 3 4]
puts [sum 1 2]
```

```
puts [sum 4]
```

We define a sum procedure which adds up all its arguments. The sum procedure has a special args argument. It has a list of all values passed to the procedure.

```
proc sum {args}
{
foreach arg $args
{
   incr s $arg
}
We go through the list and calculate the sum.
puts [sum 1 2 3 4]
puts [sum 1 2]
puts [sum 4]
```

We call the sum procedure three times. In the first case, it takes 4 arguments, in the second case 2, in the last case one. Output of the variable tcl script

$ ./variable.tcl

10

3

4

Implicit arguments

The arguments in Tcl procedures may have implicit values. An implicit value is used if no explicit value is provided.

```
#!/usr/bin/tclsh
proc power {a {b 2}}
{
   if {$b == 2}
{
      return [expr $a * $a]
   }
   set value 1
   for {set i 0} {$i<$b} {incr i}
 {
      set value [expr $value * $a]
   }
   return $value
}
```

```
set v1 [power 5]
set v2 [power 5 4]
puts "5^2 is $v1"
puts "5^4 is $v2"
```

Here we create a power procedure. The procedure has one argument with an implicit value. We can call the procedure with one and two arguments.

```
proc power {a {b 2}} {
set v1 [power 5]
set v2 [power 5 4]
```

The second argument b, has an implicit value 2. If we provide only one argument, the power procedure then returns the value of a to the power 2.We call the power procedure with one and two arguments. The first line computes the value of 5 to the power 2. The second line value of 5 to the power 4. Output of the example.

```
$ ./implicit.tcl
5^2 is 25
5^4 is 625
```

Returning multiple values

The return command passes one value to the caller. There is often a need to return multiple values. In such cases, we can return a list.

```
#!/usr/bin/tclsh
proc tworandoms {}
{
   set r1 [expr
   round(rand()*10)] set r2
   [expr round(rand()*10)]
   return [list $r1 $r2]
}
puts [two randoms]
puts [two randoms]
```

```
puts [two randoms]
puts [two randoms]
```

We have a two randoms procedure. It returns two random integers between 1 and 10. A random integer is computed and set to the r1 variable.

```
set r1 [expr round(rand()*10)]
return [list $r1 $r2]
```

Two values are returned with the help of the list command. A sample output.

```
$ ./tworandoms.tcl

3 7
1 3
8 7
9 9
```

Files

The file command manipulates file names and attributes. It has plenty of options.

```
#!/usr/bin/tclsh

puts [file volumes]
[file mkdir new]
```

The script prints the system's mounted values and creates a new directory. The file volumes command returns the absolute paths to the volumes mounted on the system.

```
puts [file volumes]
 [file mkdir new]
```

The file mkdir creates a directory called new.

```
$ ./voldir.tcl
/
```

```
$ ls -d */
doc/ new/  tmp/
```

On a Linux system, there is one mounted volume—the root directory. The `ls` command confirms the creation of the `new` directory. In the following code example, we are going to check if a file name is a regular file or a directory.

```
#!/usr/bin/tclsh
set files [glob *]
foreach fl $files
 {
   if {[file isfile $fl]}
 {
            puts "$fl is a file"
   }
elseif
 { [file isdirectory $fl]}
  {
    puts "$fl is a directory"
   }
}
```

We go through all file names in the current working directory and print whether it is a file or a directory.

Using the glob command we create a list of file and directory names of a current directory. We execute the body of the if command if the file name in question is a file.

```
set files [glob *]
if {[file isfile $fl]}
 {
} elseif { [file isdirectory $fl]} {
```

The file is directory command determines, whether a file name is a directory. Note that on Unix, a directory is a special case of a file. The puts command can be used to write to files.

```
#!/usr/bin/tclsh
set fp [open days w]
set days {Monday Tuesday Wednesday Thursday Friday Saturday Sunday}
puts $fp $days
close $fp
```

In the script, we open a file for writing. We write days of a week to a file. We open a file named days for writing. The open command returns a channel id. This data is going to be written to the file. We used the channel id returned by the open command to write to the file.

```
set fp [open days w]
set days {Monday Tuesday Wednesday Thursday Friday Saturday Sunday}
puts $fp $days
close $fp
```

We close the opened channel.

```
$ ./write2file.tcl
$ cat days
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

We run the script and check the contents of the days file. In the following script, we are going to read data from a file.

```
$ cat languages
Python
Tcl
Visual Basic
Perl
Java
C
```

C#

Ruby

Scheme

Eval

One difference between Tcl and most other compilers is that Tcl will allow an executing program to create new commands and execute them while running. A tcl command is defined as a list of strings in which the first string is a command or proc. Any string or list which meets this criteria can be evaluated and executed. The **eval** command will evaluate a list of strings as though they were commands typed at the **%** prompt or sourced from a file. The **eval** command normally returns the final value of the commands being evaluated. If the commands being evaluated throw an error (for example, if there is a syntax error in one of the strings), then **eval** will will throw an error.Note that either **concat** or **list** may be used to create the command string, but that these two commands will create slightly different command strings.

eval ***arg1 ??arg2?? ... ??argn??***

Evaluates *arg1 - argn* as one or more Tcl commands. The *args* are concatenated into a string, and passed to **tcl_Eval** to evaluate and execute.

Eval returns the result (or error code) of that evaluation.

Source

This command takes the contents of the specified file or resource and passes it to the Tcl interpreter as a text script. The return value from source is the return value of the last command executed in the script. If an error occurs in evaluating the contents of the script then the source command will return that error. If a return command is invoked from within the script then the remainder of the file will be skipped and the source command will return normally with the result from the return command.The -rsrc and -rsrcid forms of this command are only available on Macintosh computers. These versions of the command allow you to source a script from a TEXT resource. You may specify what TEXT resource to source by either name or id. By default Tcl searches all open resource files, which include the current application and any loaded C extensions. Alternatively, you may specify the fileName where the TEXT resource can be found.

source fileName

source -rsrc resourceName ?fileName?

source -rsrcid resourceId ?fileName?

Tk

Tk refers to Toolkit and it provides cross platform GUI widgets, which helps you in building a Graphical User Interface. It was developed as an extension to Tcl scripting language by John Ousterhout. Tk remained in development independently from Tcl with version being different to each other, before, it was made in sync with Tcl in v8.0.

Features

It is cross platform with support for Linux, Mac OS, Unix, and Microsoft Windows operating systems.

- It is an open source.
- It provides high level of extendibility.
- It is customizable.
- It is configurable.

Applications Built in Tk

Large successful applications have been built in Tcl/Tk.

- Dashboard Soft User Interface
- Forms GUI for Relational DB
- Ad Hoc GUI for Relational DB
- Software/Hardware System Design
- Xtask - Task Management

Tk canvas

The Tk canvas widget allows you to draw items on a pane of the application. Items may be tagged when created, and then these tagged items may be bound to events, which may be used to manipulate the items at a later stage.This process is described in detail in Robert Biddle‘s –Using the Tk Canvas Facility‖, a copy of which is found at ~cs3283/ftp/CS-TR-94-

5.pdf.Note also the use of dynamically created variable names (node$nodes).

7.Explain about Events and Bindings in Tcl?

As was mentioned earlier, a Tkinter application spends most of its time inside an event loop (entered via the mainloop method). Events can come from various sources, including key presses and mouse operations by the user, and redraw events from the window manager (indirectly caused by the user, in many cases).Tkinter provides a powerful mechanism to let you deal with events yourself. For each widget, you can bind Python functions and methods to events.widget.bind(event, handler)If an event matching the event description occurs in the widget, the given handler is called with an object describing the event.

Here‘s a simple example:

Capturing clicks in a window

```
from Tkinter import *

root = Tk()

def callback(event):

 print "clicked at", event.x, event.y

frame = Frame(root, width=100, height=100)

frame.bind("<Button-1>", callback)

frame.pack()

root.mainloop()
```

In this example, we use the bind method of the frame widget to bind a callback function to an event called <Button-1>. Run this program and click in the window that appears. Each time you click, a message like ‒clicked at 44 63‖ is printed to the console window.

Perl/Tk

Perl/Tk (also known as pTk) is a collection of modules and code that attempts to wed the easily configured Tk 8 widget toolkit to the powerful lexigraphic, dynamic memory, I/O, and object-oriented capabilities of Perl 5. In other words, it is an interpreted scripting language for making widgets and programs with **Graphical User Interfaces (GUI)**.**Perl** or *Practical Extraction and Report Language* is described by Larry Wall, Perl's author, as follows: "Perl is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language

for any system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal)." The perlintro man page has this to say.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.

Tk, the extension(or module) that makes GUI programming in perl possible, is taken from Tcl/Tk. Tcl(Tool Command Language) and Tk(ToolKit) was created by Professor John Ousterhout of the University of California, Berkeley. Tcl is a scripting language that runs on Windows, UNIX and Macintosh platforms. Tk is a standard add-on to Tcl that provides commands to quickly and easily create user interfaces. Later on Tk was used by a lot of other scripting languages like Perl, Python, Ruby etc.

Applications

Perl has been used since the early days of the web to write CGI scripts, and is now a component of the popular LAMP (Linux/Apache/MySQL/Perl) platform for web development. Perl has been called "the glue that holds the web together". Large systems written in Perl include Slashdot, and early implementations of Wikipedia and PHP.Perl finds many applications as a glue language, tying together systems and interfaces that were not specifically designed to interoperate. Systems administrators use Perl as an all-purpose tool; short Perl programs can be entered and run on a single command line.

UNIT-V

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages. Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP. Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs. Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects. Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

1. What is the History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

Python's features include:

Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

Easy-to-read: Python code is more clearly defined and visible to the eyes.

Easy-to-maintain: Python's source code is fairly easy-to-maintain. A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Databases: Python provides interfaces to all major commercial databases.

GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below: It supports functional and structured programming methods as well as OOP. It can be used as a scripting language or can be compiled to byte-code for building large applications. It provides very high-level dynamic data types and supports dynamic type checking. It supports automatic garbage collection. It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

2. Explain the Basic Syntax Python

The Python language has many similarities to Perl, C, and Java. However, there are some definite differences between the languages.

First Python Program

First Python Program

Let us execute programs in different modes of programming. Interactive Mode Programming: Invoking the interpreter without passing a script file as a parameter brings up the following prompt:

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
 [GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
 Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type the following text at the Python prompt and press the Enter:

```
>>> print "Hello, Python!";
```

If you are running new version of Python, then you need to use print statement with parenthesis as in print ("Hello, Python!");. However in Python version 2.4.3, this produces the following result:

Hello, Python!

Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active. Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file:

```
print "Hello, Python!";
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows:

```
$ python test.py
```

This produces the following result:

```
Hello, Python!
```

1. What are the Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z, or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9). Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python. Here are naming conventions for Python identifiers:

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

2. What are the Python Keywords?

The following list shows the Python keywords. These are reserved words and you cannot use them

as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

And	Exec	Not	del	import
Assert	finally	or	while	try
Break	For	pass	with	elif
Class	from	print	in	yield
Continue	global	raise	else	is
def	if	return	lambda	except

Lines and Indentation:

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example:

if True:

print "True"

else:

print "False"

However, the following block generates an error:

if True:

print "Answer" print "True"

else:

print "Answer" print "False"

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks.

Note: Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example:

```
total = item_one + \
        item_two + \
        item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example:

```
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

Quotation in Python

Python accepts single ('), double (") and triple ('" or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal:

```
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is made up of multiple lines and sentences."""
```

Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

```
#!/usr/bin/python

 # First comment print "Hello, Python!"; # second comment
```

This produces the following result:

Hello, Python!

You can type a comment on the same line after a statement or expression:

name = "Madisetti" # This is again comment

You can comment multiple lines as follows:

This is a comment.

This is a comment, too.

This is a comment, too.

I said that already.

Variable Types

Variables are nothing but reserved memory locations to store values. This means when you create a variable, you reserve some space in memory. Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables. The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example:

```
#!/usr/bin/python
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point name =
"John"  # A string
print counter
print miles
print name
```

Here, 100, 1000.0, and "John" are the values assigned to counter, miles, and name variables respectively. This produces the following result:

100

1000.0

John

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example:

a = b = c = 1

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example:

a, b, c = 1, 2, "john"

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types:

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example:

var1 = 1

var2 = 10

You can also delete the reference to a number object by using the del statement. The syntax of the

del statement is:

del var1[,var2[,var3[....,varN]]]]

You can delete a single object or multiple objects by using the del statement. For example:

del var

del var_a, var_b

Python supports four different numerical types:

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Examples

Here are some examples of numbers:

int long Float complex

10 51924361L 0.0 3.14j

100 -0x19323L 15.20 45.j

-786 0122L -21.9 9.322e-36j

080 0xDEFABCECBDAECBFBAEl 32.3+e18 .876j

-0490 535633629843L -90. -.6545+0J

-0x260 -052318172735L -32.54e100 3e+26J

0x69 -4721885298529L 70.2-E12 4.53e-7j

•Python allows you to use a lowercase L with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.

•A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x is the real part and b is the imaginary part of the complex number.

3. Define Python Strings?

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example:

```
#!/usr/bin/python

str = 'Hello World!'

print str          # Prints complete string

print str[0]  # Prints first character of the string print str[2:5]  # Prints characters starting from 3rd to 5th print str[2:]  # Prints string starting from 3rd character print str * 2  # Prints string two times

print str + "TEST" # Prints concatenated string
```

This will produce the following result:

```
Hello World! H

llo

llo World!

Hello World!Hello World! Hello World!TEST
```

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example:

```
#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ] tinylist = [123, 'john']

print list          # Prints complete list
```

print list[0] # Prints first element of the list

print list[1:3] # Prints elements starting from 2nd till 3rd print list[2:] # Prints elements starting from 3rd element print tinylist * 2 # Prints list two times

print list + tinylist # Prints concatenated lists

This produces the following result:

['abcd', 786, 2.23, 'john', 70.200000000000003]

abcd

[786, 2.23]

[2.23, 'john', 70.200000000000003]

[123, 'john', 123, 'john']

['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read- only lists. For example:

#!/usr/bin/python

tuple = ('abcd', 786 , 2.23, 'john', 70.2) tinytuple = (123, 'john')

print tuple # Prints complete list

print tuple[0] # Prints first element of the list

print tuple[1:3] # Prints elements starting from 2nd till 3rd print tuple[2:] # Prints elements starting from 3rd element print tinytuple * 2 # Prints list two times

print tuple + tinytuple # Prints concatenated lists

This produces the following result:

('abcd', 786, 2.23, 'john', 70.200000000000003)

abcd

(786, 2.23)

(2.23, 'john', 70.200000000000003)

(123, 'john', 123, 'john')

('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists:

#!/usr/bin/python

tuple = ('abcd', 786 , 2.23, 'john', 70.2)

list = ['abcd', 786 , 2.23, 'john', 70.2]

tuple[2] = 1000 # Invalid syntax with tuple list[2] = 1000 # Valid syntax with list

Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

For example:

#!/usr/bin/python

dict = {}

dict['one'] = "This is one" dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'} print dict['one'] # Prints value for 'one' key print dict[2] # Prints value for 2 key

print tinydict # Prints complete dictionary print tinydict.keys() # Prints all the keys

print tinydict.values() # Prints all the values

This produces the following result:

This is one This is two

{'dept': 'sales', 'code': 6734, 'name': 'john'} ['dept', 'code', 'name']

['sales', 6734, 'john']

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

4. Explain Type Conversions in python

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.

oct(x)	Converts an integer to an octal string.

5.Explain Operators in python?

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left handoperand by right hand operand	b / a = 2
% Modulus	Divides left operand by right operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10to the power 20

// Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.

9//2 = 4 and 9.0//2.0 = 4.0

Example

Assume variable a holds 10 and variable b holds 20, then:

```
#!/usr/bin/python

a = 21

b = 10

c = 0

c = a + b

print "Line 1 - Value of c is ", c c = a - b

print "Line 2 - Value of c is ", c c = a * b

print "Line 3 - Value of c is ", c c = a / b

print "Line 4 - Value of c is ", c

c = a % b

print "Line 5 - Value of c is ", c

a = 2

b = 3

c = a**b

print "Line 6 - Value of c is ", c

a = 10

b = 5

c = a//b

print "Line 7 - Value of c is ", c
```

When you execute the above program, it produces the following result:

Line 1 - Value of c is 31 Line 2 - Value of c is 11 Line 3 - Value of c is 210 Line 4 - Value of c is 2 Line 5 - Value of c is 1 Line 6 - Value of c is 8

Line 7 - Value of c is 2

Python Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true. This is similar to != operator.	(a <> b) is true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Example

Assume variable a holds 10 and variable b holds 20, then:

```
#!/usr/bin/python

a = 21

b = 10

c = 0

if ( a == b ):

print "Line 1 - a is equal to b" else:

print "Line 1 - a is not equal to b"

if ( a != b ):

print "Line 2 - a is not equal to b" else:

print "Line 2 - a is equal to b"
```

```
if ( a <> b ):
print "Line 3 - a is not equal to b" else:
print "Line 3 - a is equal to b"
if ( a < b ):
print "Line 4 - a is less than b" else:
print "Line 4 - a is not less than b"
if ( a > b ):
print "Line 5 - a is greater than b" else:
print "Line 5 - a is not greater than b"
a = 5;
b = 20;
if ( a <= b ):
print "Line 6 - a is either less than or  equal to          b" else:
print "Line 6 - a is neither less than nor  equal to       b"
if ( b >= a ):
print "Line 7 - b is either  greater than          or equal to b" else:
print "Line 7 - b is neither  greater than          nor equal to b"
```

When you execute the above program it produces the following result:

Line 1 - a is not equal to b Line 2 - a is not equal to b Line 3 - a is not equal to b Line 4 - a is not less than b Line 5 - a is greater than b

Line 6 - a is either less than or equal to b

Line 7 - b is either greater than or equal to b

Python Assignment Operators

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description

= Assigns values from right side operands to left side operand

+= It adds right operand to the left operand and assign the result to left operand

-= It subtracts right operand from the left operand and assign the result to left operand

*= It multiplies right operand with the left operand and assign the result to left operand

/= It divides left operand with the right operand and assign the result to left operand

%= It takes modulus using two operands and assign the result to left operand

**= Performs exponential (power) calculation on operators and assign value to the left operand ** a

//= It performs floor division on operators and assign value to the left operand // a

Example

Assume variable a holds 10 and variable b holds 20, then:

```
#!/usr/bin/python

a = 21

b = 10

c = 0

c = a + b

print "Line 1 - Value of c is ", c

c += a

print "Line 2 - Value of c is ", c

c *= a

print "Line 3 - Value of c is ", c

c /= a

print "Line 4 - Value of c is ", c

c = 2 c %= a

print "Line 5 - Value of c is ", c
```

```
c **= a
print "Line 6 - Value of c is ", c
c //= a
print "Line 7 - Value of c is ", c
```

When you execute the above program, it produces the following result:

```
Line 1 - Value of c is 31
Line 2 - Value of c is 52
Line 3 - Value of c is 1092
Line 4 - Value of c is 52
Line 5 - Value of c is 2
Line 6 - Value of c is 2097152
Line 7 - Value of c is 99864
```

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in binary format they will be as follows:

```
a = 0011 1100
b = 0000 1101
----------------- a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011
```

There are following Bitwise operators supported by Python language.

Operator	Description
&	Operator copies a bit to the result if it exists in both operands.

| It copies a bit if it exists in either operand.

^ It copies the bit if it is set in one operand but not both.

~ It is unary and has the effect of 'flipping' bits.

<< The left operands value is moved left by the number of bits specified by the right operand.

>> The left operands value is moved right by the number of bits specified by the right operand.

Example

```
#!/usr/bin/python

a = 60 # 60 = 0011 1100

b = 13# 13 = 0000 1101

c = 0

c = a & b;      # 12 = 0000 1100

print "Line 1 - Value of c is ", c

c = a | b;      # 61 = 0011 1101

print "Line 2 - Value of c is ", c

c = a ^ b;      # 49 = 0011 0001

print "Line 3 - Value of c is ", c

c = ~a;         # -61 = 1100 0011

print "Line 4 - Value of c is ", c

c = a << 2;     # 240 = 1111 0000

print "Line 5 - Value of c is ", c

c = a >> 2;     # 15 = 0000 1111

print "Line 6 - Value of c is ", c
```

When you execute the above program it produces the following result:

Line 1 - Value of c is 12

Line 2 - Value of c is 61

Line 3 - Value of c is 49

Line 4 - Value of c is -61

Line 5 - Value of c is 240

Line 6 - Value of c is 15

Python Logical Operators

There are following logical operators supported by Python language. Assume variable

a holds 10 and variable b holds 20 then:

Operator	Description
and Logical AND	If both the operands are true then condition becomes true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.
not Logical NOT	Used to reverse the logical state of its operand.

Python Membership Operators

Python‘s membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below:

Operator	Description
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.

x in y, here in results in a 1 if x is a member of sequence y.

not in Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. x not in y, here not in results in a 1 if x is not a member of sequence y.

Example

```
#!/usr/bin/python

a = 10

b = 20
```

```
list = [1, 2, 3, 4, 5 ];

if ( a in list ):

print "Line 1 - a is available in the given list" else:

print "Line 1 - a is not available in the given list"

if ( b not in list ):

print "Line 2 - b is not available in the given list" else:

print "Line 2 - b is available in the given list"

a = 2

if ( a in list ):

print "Line 3 - a is available in the given list" else:

print "Line 3 - a is not available in the given list"
```

When you execute the above program it produces the following result:

Line 1 - a is not available in the given list Line 2 - b is not available in the given list

Line 3 - a is available in the given list

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators as explained below:

Operator	Description
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.

Example

```
#!/usr/bin/python

a = 20

b = 20
```

```
if ( a is b ):
print "Line 1 - a and b have same identity" else:
print "Line 1 - a and b do not have same identity"
if ( id(a) == id(b) ):
print "Line 2 - a and b have same identity" else:
print "Line 2 - a and b do not have same identity"
b = 30
if ( a is b ):
print "Line 3 - a and b have same identity" else:
print "Line 3 - a and b do not have same identity"
if ( a is not b ):
print "Line 4 - a and b do not have same identity" else:
print "Line 4 - a and b have same identity"
```

When you execute the above program it produces the following result:

```
Line 1 - a and b have same identity Line 2 - a and b have same identity
Line 3 - a and b do not have same identity
Line 4 - a and b do not have same identity
```

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift

&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= +=*= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Operator precedence affects how an expression is evaluated.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first multiplies 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Example

```
#!/usr/bin/python

a = 20

b = 10

c = 15

d = 5

e = 0

e = (a + b) * c / d       #( 30 * 15 ) / 5 print "Value of (a + b) * c / d is ",      e

e = ((a + b) * c) / d  # (30 * 15 ) / 5 print "Value of ((a + b) * c) / d is ",    e

e = (a + b) * (c / d); # (30) * (15/5) print "Value of (a + b) * (c / d) is ", e

e = a + (b * c) / d;      #          20 + (150/5) print "Value of a + (b * c) / d is ",e
```

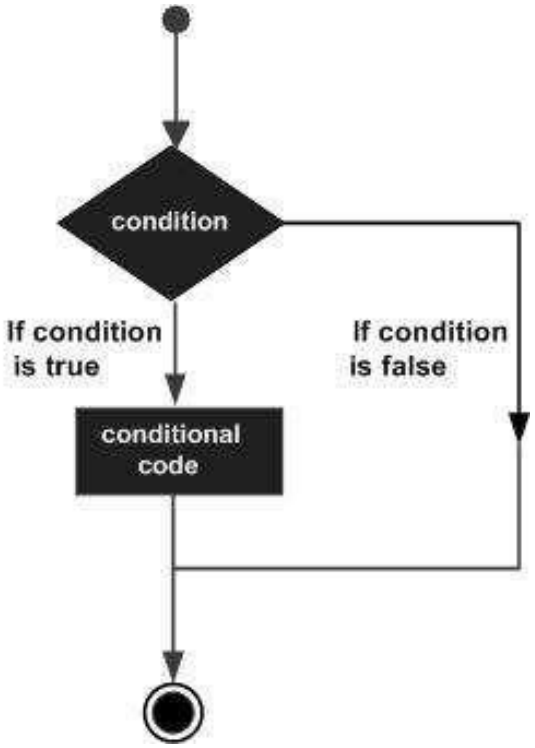
When you execute the above program, it produces the following result:

Value of (a + b) * c / d is 90 Value of ((a + b) * c) / d is 90 Value of (a + b) * (c / d) is 90

Value of a + (b * c) / d is 50

6.Explain control Statements in python?

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions. Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise. Following is the general form of a typical decision making structure found in most of the programming languages:



Python programming language assumes any non-zero and non-null values as TRUE, and if it is either zero or null, then it is assumed as FALSE value. Python programming language provides following types of decision making statements. Click the following links to check their detail.

if statements if statement consists of a Boolean expression followed by one or more statements.

if...else statements if statement can be followed by an optional else statement, which executes when the Boolean expression is FALSE. Nested if statements .You can use one if or else if

statement inside another if or else if statement(s).

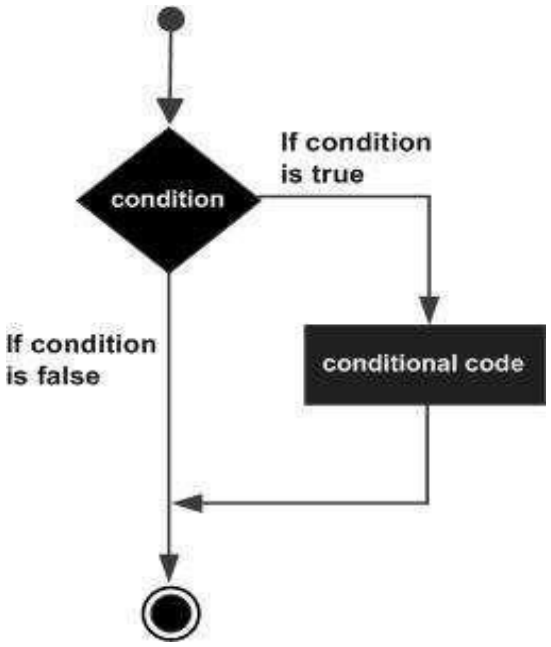
If Statement

It is similar to that of other languages. The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.

Syntax

if expression: statement(s)

If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed. If boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.



If…else Statement

An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

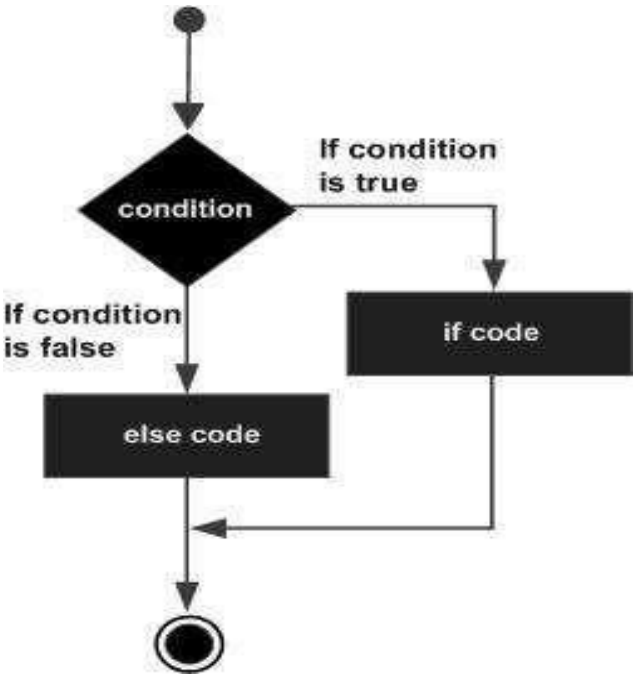
The else statement is an optional statement and there could be at most only one else statement following if.

Syntax

The syntax of the if...else statement is:

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

Flow Diagram:



The elif Statement

The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.Similar to the else, the elif statement is optional.

However, unlike else, for which there can be at most one statement, there can be an arbitrary number of elif statements following an if.

Syntax

if expression1: statement(s)

elif expression2: statement(s)

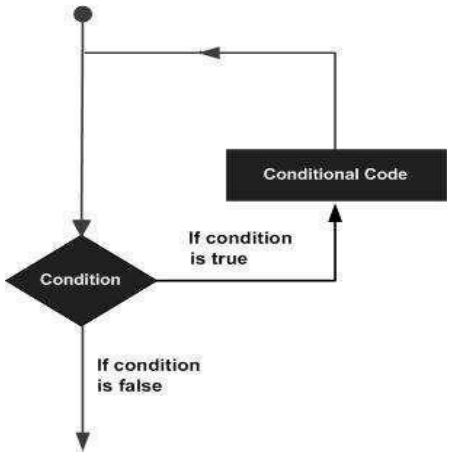
elif expression3: statement(s)

else:

statement(s)

LOOPS

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times .Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement:



Python programming language provides following types of loops to handle looping requirements.

While

A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

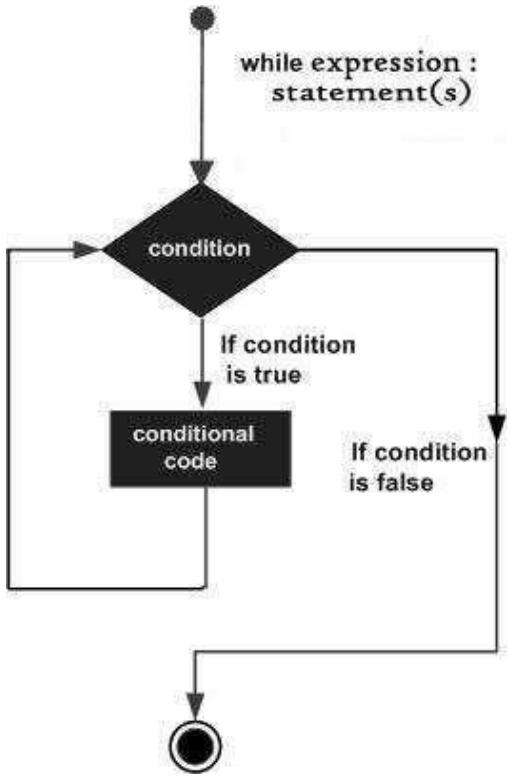
Syntax

The syntax of a while loop in Python programming language is:

while expression: statement(s)

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true .When the condition becomes false, program control passes to the line immediately following the loop. In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

Flow Diagram



Here, key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Using else Statement with Loops

Python supports to have an else statement associated with a loop statement. If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

For Loop

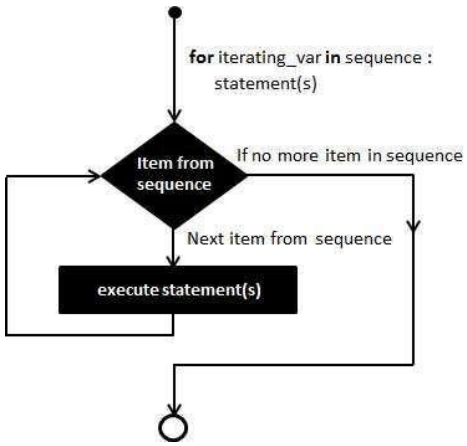
It has the ability to iterate over the items of any sequence, such as a list or a string.

Syntax

for iterating_var in sequence: statements(s)

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statements block is executed. Each item in the list is assigned to iterating_var, and the statement(s) block is executed until the entire sequence is exhausted.

Flow Diagram



Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements. Click the following links to check their detail.

Control Statement Description

break statement

Terminates the loop statement and transfers execution to the statement immediately following the loop.

continue statement

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

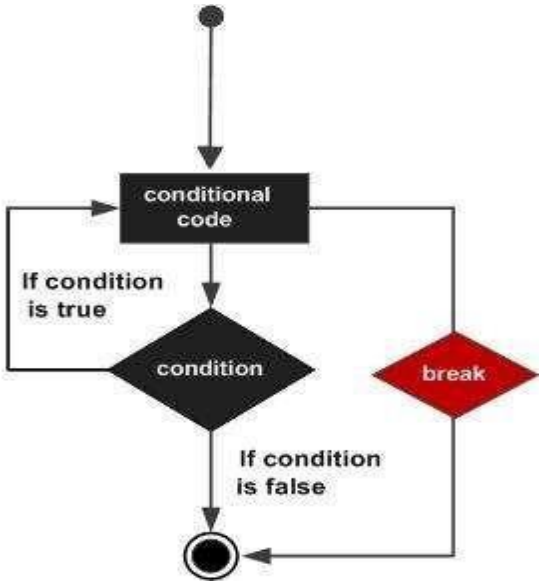
pass statement

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

Break Statement

It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.The syntax for a break statement in Python is as follows:

break



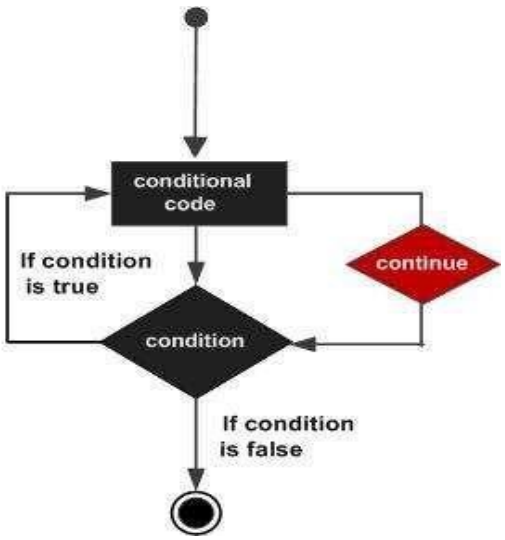
Continue Statement

It returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The continue statement can be used in both while and for loops.

Syntax

continue

Flow Diagram



Pass Statement: It 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 (e.g., in stubs for example):

Syntax

```
pass
```

Functions

It is possible, and very useful, to define our own functions in Python. Generally speaking, if you need to do a calculation only once, then use the interpreter. But when you or others have need to perform a certain type of calculation many times, then define a function. For a simple example, the compound command

```
>>> def f(x):
... return x* x
...
```

defines the squaring function f (x) = x2, a popular example used in elementary math courses. In the definition, the first line is the function header where the name, f, of the function is specified. Subsequent lines give the body of the function, where the output value is calculated. Note that the final step is to return the answer; without it we would never see any results. Continuing the example, we can use the function to calculate the square of any given input:

```
>>> f(2) 4
>>> f(2.5)
6.25
```

The name of a function is purely arbitrary. We could have defined the same function as above, but with the name square instead of f; then to use it we use the new function name instead of the old:

```
>>> def square(x):
... return x* x
...
>>> square (3)
9
```

>>> square (2.5)

6.25

Actually, a function name is not completely arbitrary, since we are not allowed to use a reserved word as a function name. Python‘s reserved words are: and, def, del, for, is, raise, assert, elif, from, lambda, return, break, else, global, not, try, class, except, if, or, while, continue, exec, import, pass, yield.

By the way, Python also allows us to define functions using a format similar to the Lambda Calculus in mathematical logic. For instance, the above function could alternatively be defined in the following way:

>>> square = lambda x: x* x

Here lambda x: x*x is known as a lambda expression. Lambda expressions are useful when you need to define a function in just one line; they are also useful in situations where you need a function but don‘t want to name it.

Usually function definitions will be stored in a module (file) for later use. These are indistinguishable from Python‘s Library modules from the user‘s perspective.

Explain Files in python?

Python allows us to store our code in files (also called modules). This is very useful for more serious programming, where we do not want to retype a long function definition from the very beginning just to change one mistake. In doing this, we are essentially defining our own modules, just like the modules defined already in the Python library. For example, to store our squaring function example in a file, we can use any text editor3 to type the code into a file, such as

```
def
square(x):
return x* x
```

Notice that we omit the prompt symbols >>>, ... when typing the code into a file, but the indentation is still important. Let‘s save this file under the name ―SquaringFunction.py‖ and then open a terminal in order to run it:

```
doty@ brauer:~ python
Python 2.5.2 ( r252 :60911 , Apr 21 2008 , 11 :12 :42 )
```

[GCC 4.2.3 (Ubuntu 4.2.3 -2 ubuntu 7)] on
linux2 Type " help", " copyright", " credits" or " license"
for more inform ation.
>>> from Squaring Function import square
>>> square (1.5)
2.25

Notice that I had to import the function from the file before I could use it. Importing a command from a file works exactly the same as for library modules. (In fact, some people refer to Python files as –modules‖ because of this analogy.) Also notice that the file‘s extension (.py) is omitted in the import command.

Testing code

As indicated above, code is usually developed in a file using an editor. To test the code, import it into a Python session and try to run it. Usually there is an error, so you go back to the file, make a correction, and test again. This process is repeated until you are satisfied that the code works. The entire process is known as the development cycle. There are two types of errors that you will encounter. Syntax errors occur when the form of some command is invalid. This happens when you make typing errors such as misspellings, or call something by the wrong name, and for many other reasons. Python will always give an error message for a syntax error.

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order. Some of them are

abs(x)

Return the absolute value of a number. The argument may be an integer or a floating point number. If the argument is a complex number, its magnitude is returned.

all(iterable)

Return True if all elements of the iterable are true.

ascii(object)

As repr(), return a string containing a printable representation of an object, but escape the non-ASCII characters in the string returned by repr() using \x, \u or \U escapes. This generates a string similar to that returned by repr() in Python 2.

bin(x)

Convert an integer number to a binary string. The result is a valid Python expression. If x is not a Python int object, it has to define an_index_() method that returns an integer.

bool([x])

Convert a value to a Boolean, using the standard truth testing procedure. If x is false or omitted, this returns False; otherwise it returns True. bool is also a class, which is a subclass of int. Class bool cannot be subclassed further. Its only instances are False and True.

chr(i)

Return the string of one character whose Unicode codepoint is the integer i. For example, chr(97) returns the string 'a'.

dir([object])

Without arguments, return the list of names in the current local scope. With an argument, attempt to return a list of valid attributes for that object.

float([x])

Convert a string or a number to floating point.

hex(x)

Convert an integer number to a hexadecimal string.

pow(x, y[, z])

Return x to the power y; if z is present, return x to the power y, modulo z (computed more efficiently than pow(x, y) % z). The two-argument form pow(x, y) is equivalent to using the power operator: x**y.

Python Methods

Python has some list methods that you can use to perform frequency occurring task (related to list) with ease. For example, if you want to add element to a list, you can use append() method.The page contains all methods of list objects. Also, the page includes built-in functions that can take list as a parameter and perform some task. For example, all() function returns True if all elements of an list (iterable) is true. If not, it returns False.

Method	Description
Python List append()	Add Single Element to The List
Python List extend()	Add Elements of a List to Another List
Python List insert()	Inserts Element to The List
Python List remove()	Removes Element from the List
Python List index()	returns smallest index of element in list
Python List count()	returns occurrences of element in a list
Python List pop()	Removes Element at Given Index
Python List reverse()	Reverses a List
Python List sort()	sorts elements of a list
Python List copy()	Returns Shallow Copy of a List
Python List clear()	Removes all Items from the List
Python any()	Checks if any Element of an Iterable is True
Python all()	returns true when all elements in iterable is true
Python ascii()	Returns String Containing Printable Representation
Python bool()	Coverts a Value to Boolean
Python enumerate()	Returns an Enumerate Object
Python filter()	constructs iterator from elements which are true
Python iter()	returns iterator for an object
Python list() Function	creates list in Python
Python len()	Returns Length of an Object
Python max()	returns largest element
Python min()	returns smallest element
Python map()	Applies Function and Returns a List
Python reversed()	returns reversed iterator of a sequence
Python slice()	creates a slice object specified by range()
Python sorted()	returns sorted list from a given iterable
Python sum()	Add items of an Iterable
Python zip()	Returns an Iterator of Tuples

Modules in python

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Example

The Python code for a module named aname normally resides in a file namedaname.py.
Here is an example of a simple module, support.pydef
print_func(par):
print "Hello : ", par
return

Import Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The import has the following syntax
import module1[, module2[,... moduleN]

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module.

From...import Statement

Python's from statement lets you import specific attributes from a module into the current namespace. The from...import has the following syntax.
Syntax

From modname import name1[, name2[, ... nameN]]

From...import * Statement

It is also possible to import all the names from a module into the current namespace by using the following import statement

From modname import *

This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.

Exception Handling

Python provides very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them-

Exception

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error. When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an Exception

If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.

Syntax

Here is simple syntax of try....except...else blocks

blockstry:

You do your operations here

.....

except ExceptionI:

If there is ExceptionI, then execute this block.

except ExceptionII:

If there is ExceptionII, then execute this block.

.....

else:

If there is no exception then execute this block.

Here are few important points about the above-mentioned syntax-

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the elseblock executes if the code in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection.

Except Clause with No Exceptions

You can also use the except statement with no exceptions defined as follows

try:

You do your operations here

.....

except:

If there is any exception, then execute this block.

.....

else:

If there is no exception then execute this block. This kind of a try-except statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

Try-finally Clause

You can use a finally: block along with a try: block. The finally: block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this

try:

You do your operations here;

.....

Due to any exception, this may be skipped.

finally:

This would always be executed.

.....

Note: You can provide except clause(s), or a finally clause, but not both. You cannot use else clause as well along with a finally clause.

Raising an Exception

You can raise exceptions in several ways by using the raise statement. The general syntax for the raise statement is as follows

Syntax

raise [Exception [, args [, traceback]]]

Here, Exception is the type of exception (for example, NameError) and argument is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.The final argument, traceback, is also optional (and rarely used in practice), and if present,is the traceback object used for the exception.

9.Explain about Web Application Framework

A web framework is nothing but a collection of packages and modules that allows easier development of websites. It handles all low-level communication within the system and hides it from you to make no issues for performing common tasks for the development.

Why choose a Python web framework?

You can always create a web application from scratch, but there are various reasons you would not want to. Below are the top five reasons why you should choose Python framework over your own.

Used and trusted by many large companies

Popular Python frameworks like Pyramid and Django are used by companies like Bitbucket, Pinterest, Instagram and Dropbox in their web application development. So, it is safe to say these frameworks are able to handle almost everything you throw at them.

Hides complicated low-level details

As stated above, web frameworks are meant to hide and handle all low-level details so that you as a developer, do not have to dig deep into how everything works when you are developing a web-enabled application.

Saves time

The frameworks have been built investing thousands of developer and testing hours. Building on top of it saves your valuable time. Especially, when you are developing a simple prototype of a website or are closer to a deadline, using web frameworks can turn out to be a life safer.

Security

One of the most important advantages of using a web framework as opposed to building something on your own is handling the security of your website. Since web frameworks have been used and backed by thousands, it inherently handles security, preventing any misuse of the web application.

Efficient and scalable system

Good frameworks are built ensuring scalability from the very beginning of the development process. So, whenever you are planning to scale your website by adding a new component or using a new database, web frameworks are more likely to scale better than what you come up with when building from scratch.

10.Which framework to choose?

There are tons of Python web frameworks, and every framework has their own strengths and weaknesses. Thus, it is necessary to evaluate your project requirements and pick one the best one from the collection.Below are the three most popular web frameworks in Python.

Django

Django is probably the most popular Python web framework and is aimed mostly at larger applications. It takes a ‑batteries-included‖ approach and contains everything needed for web development bundled with the framework itself. So, you do not have to handle things like database administration, templating, routing, authentication and so on. With fairly less code, you can create great applications with Django.If you are building a mid-high ranged web applicants and are quite comfortable with Python, you should go for Django.

Pyramid

The Pyramid is the most flexible Python web framework and just like Django, it is aimed at mid-high scale applications. If you think Django brings too much bloat to your web application, use Pyramid. It does not force you to use a single solution for a task, but rather gives you a pluggable system to plug-in according to your project requirements. You do have the basic web development capabilities like routing and authentication, but that is about it. So, if you want to connect to a database for storage, you ought to do that yourself using external libraries.

Flask

Flask is the new kid in town. Unlike Pyramid and Django, Flask is a micro-framework and is best suited for small-scale applications.Even if it is new, Flask has integrated great features of other frameworks. It includes features like unit testing and built-in development server that enable you to create reliable and efficient web applications.