Web Technologies Notes

Contributor: Abhishek

[KMV (DU)]

Computer Science Notes

HTML :-

- Hyper Text Markup Language.
- It is used to display the document content (text, images ---) in the browser.
- HTML pages can be develop simple text or complex multimedia programs containing images, sounds.
- It is not a programming Language.
- It was developed by Tim Berners Lee in the year of 1991.
- The HTML standards are created by group of organizations called W3C (World Wide Web Consortium)
- A HTML formatting is specified by using Tags.

HTML Tags :-

- The fundamental syntactic units of HTML are called tags.
- Tags are used to specify the categories of content. For each category, browser has default browsers.
- Syntax of Tag is " the tag name surrounded by angler brackets (< >)".
- All the tag names must be written in lowercase.
- Most of the tags appear pairs of opening tag and closing tag.
- The closing tag is name of opening tag with slash attached to the begining.

- the content should enclosed blw tags.
- the container and content together called element.

<P> Hello world </P>

content

Element.

Attributes :-

- Attributes are used to specify the alternate meaning of tag, it can be appear between an opening tag and right angler bracket (>).
- these are in keyword form, followed by equal sign and attribute value.
- Attribute names should be in lowercase and value should enclosed between (" ") double quotes.

<P align = "center"> Hello world </P>

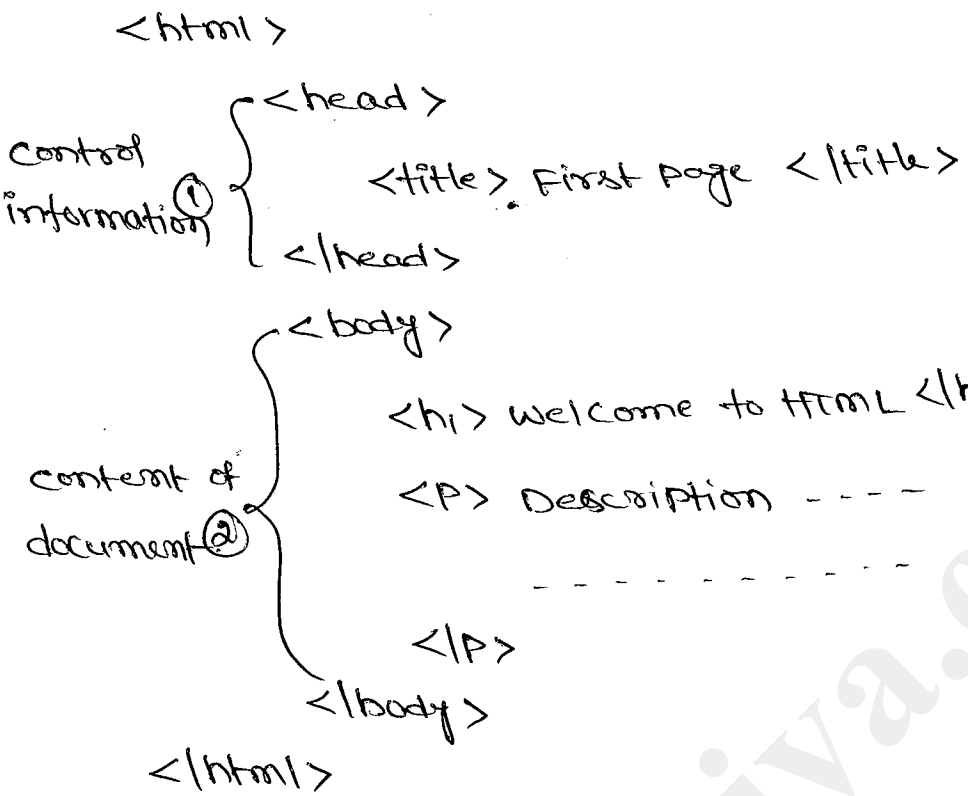
align → attribute name

"center" → attribute value

Comments :-

- comments in a program increase the readability of a program
- Syntax is <!-- -->
- Generally comments are used as follows
  - Name of the application
  - Description of code file
  - Name of author
  - original creation date
  - version number
  - copyright information.

Structure of HTML Document :-



- Every HTML document surrounded by

  <html> ------ </html>

- Head section contains control information used by browser and server.
- Body section contains the content of document that displays on screen and tags.
- Every HTML pages having one title which is displayed at top of the browser window.
- Some of the general tags are

* <P [align="left"|"right"|"center"]> ---- </P>

  Paragraph tag

* Headings tag

  <h1 [align="left"|"right"|"center"]> --- </h1>

  <h2> --- </h2>

  ⋮ ⋮ ⋮

  <h6> --- </h6>

- <h5> & <h6> are used for smaller headings

-
 - breaking ~~(or)~~ the sentence from one line to another line

  * Slash indicates that the tag is both opening & closing.

- Horizontal row

  <hr [align="left"|"right"|"center"]

  [size="n"]

  [width="nn%"] />

- Bold the text

- Italic text

- Highlighting the text or stress the text

- striking the text

  <strike> ---- </strike>

- Superscript the text

  ⁻⁻⁻⁻

- Subscript the text

  ₋₋₋₋

- Large text

  <big> ---- </big>

- Smaller text

  <small> ---- </small>

- Type writter text (or) monospace font style

  <tt> ---- </tt>

** - the html document can save with the extension of .html (or) .htm.

Character Entries :-

Character	Entity	Meaning
&	&amp;	ampresand
<	&lt;	less than
>	&gt;	greater than
"	&quot;	double quote
'	&apos;	single quote (apostrophe)
(space)		non breaking space
©	&copy;	copy write symbol
°	&deg;	degree
1/4	&frac14	one quarter
1/2	&frac12	one half

- Body tag attributes are

```
<body  bgcolor = "#FFFFFF" (or) "color name"
       background = "image location" >
       ---
</body>
```

- Font tag attributes are

```
<font  size = "[+|-]n"
       type = "font type"  like verdana...
       color = "#rrggbb" >  ----  </font>
```

Hyperlinks :-

- Linking the documents together
- Anchor tag

```
<a  href = "target file path"
    name = "string"
    target = "string" | _blank >
    -----
</a>
```

List Tags :-

- Lists are used for effective way of structuring the webpage content
- It provide stright forward index in website.
- There are 3 types of lists
  1. Ordered list (Numbered)
  2. Unordered list (bulletted)
  3. Definition list

1. Order list :-

- The list of items having the sequence of numbering then we are using ordered list
- Syntax is

```
<ol  type = "1"|"a"|"A"|"I"|"i"
     start = "n" >
  ----------
</ol>
```

- the list of items can be enclosed between <li> tag

```
<li> ---- </li>
```

2. Unordered list :-

- Unordered list is used for representing the list of items in the document.
- Syntax is

```
<ul  type = "disc"|"square"|"circle" >
  ----------
</ul>
```

- Lists can be easily embedded in another list to provide complex type of lists.

3. Definition list :-

- Definition list is used for represents the definitions into the webpage.
- Entire definitions are enclosed between

<dl> ------ </dl>

- Definition term

<dt> ------ </dt>

* Making whose definition is providing.

- Definition Data ~~or~~ ~~Data~~

* Definition of the term is enclosed with in the following tags

<dd> ------ </dd>

Table Tags :-

- Tables are providing highly readable way of presenting many types of information.
- Table is used to structure the piece of information and to structure the whole web page.
- A Table is a matrix of rows and columns, in which the intersection of rows and column is called cell.
- A cell containing information or data of the table.
- the information may be text, headings, images or nested tables.
- Syntax is

<table align="center"/"left"/"right"
border ="n"
width = "nn%" >

</table>

- Some more tags in table are

- <caption> ---- </caption>

  * It will give the title or caption of the table.

- <th> ----- </th>

  * It is used for table headings

- <tr> ------ </tr>

  * It is used for creating rows in the table

- <td> ------- </td>

  * It is used for storing data in the cell.

*- Attributes for td are

  * colspan | rowspan
    - Used to merging multiple levels of rows or columns in the table.
  * valign | align
    - Used to placement of the content in the table cell

      align = "center" | "left" | "right"

      valign = "center" | "top" | "bottom"
  * Cell padding and cell spacing
    - Cell padding is the spacing between the content of the cell and edge of cell.
    - Cell spacing is the spacing between the adjacent cell.

- <thead> ---- </thead>

  ↳ It is used for table header

- <tfoot> ------ </tfoot>

  - Used for table footer

- <tbody> ---- </tbody> * content of the table.

Image Tags :-

- The inclusion of images into the document, those images will display on the web browsers.
- GIF (Graphic Interchange format)
  JPEG (Joint photographic Experts group)
  Bmp (Bit Map Image)
  PNG (Portable Network Graphics)
  TIFF (Tag Inder File Format)
- Syntax is

```
<img  src = "location of image"
      alt = "alternate name of image" or "string"
      name = "string"
      height = "nn%."
      width = "nn%."
      align = "left"|"right"|"center" />
```

- Images and text display side by side then we have to use table.
- Images are one of the main aspect of webpages
- loading image into webpage is very slow process, and if too many images are used the document time is become intolerable.

* - Image maps

```
<area  shape = "circle"|"rect"|"poly"|"default"
       href = "URL"
       alt = "string" />
```

Forms :-

- To improve or adding the interactivity of elements in a webpage.
- Generally forms are used to collect the data from the user.
- Syntax

```
<form   action = "URL"
        method = "GET | POST" >
        :
</form>
```

- action attribute : action should perform or target page when the form is submitted.
- method attribute :
  GET (Default) - Users information is visible in the address bar of a web page
  POST - It will hide the information of webpage.

Controls in form :-

- Generally in any application the controls are like text, checkbox, radio button, select box or dropdown list, submit reset buttons and text area field.
- Text : It uses single line of text
  Every component is seperated by name attribute

```
<input type = "text" name = "string" size = "n" />
```

- Password : It is similar to text, but the content is not displayed on the screen.

```
<input type = "password" name = "string" />
```

- Checkbox : It is simply provide checkbox's.
  - Generally for multiple option section, we are using checkbox's

<input type="checkbox" [checked] name="string" />

  - checked attribute is optional, if you write checked then the first or corresponding value is default selected.
- Radio buttons : It will creates a radio button.
  - they are always grouped together with same name but different values
  - From multiple options we are selecting one option.
- Text area : For longer than single sentences we are used.

<textarea cols="n" rows="n" name="string" />

- ~~Submit~~ Select : It will create a button, which displays the value can be select
  - Drop down box which provide options list

<select name="dropdown" >
<option value="maths" selected > maths </option>
<option value="English" > English </option>
</select>

*Size attribute for select - It will used for scrolling the list box or dropdown list.

- Button controls :-

Submit : It will used for sending or storing the users information

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

Reset :- It will used for clears the information in the form

```
<input type="reset" value="clear" names="string"/>
```

Button : It will used for clicking purposes

```
<input type="Button" value="OK" name="string"/>
```

Image button :

```
<input type="image" name="string" src="URL/>
```

- File upload box :- It allows the user to upload a file to the website.

```
<input type="file" name="string"
       accept="image/*"/>
```

* accept attribute indicates the type of file we are uploading.

FRAMES :-

- For displaying more than one document at a time we are using FRAMES
- Here window is divided into rectangular areas, each of which is called a frame.
- Each frame capable of displaying its own document.
- Generally frames are used, the table of content will displayed in one page and part of the main document displayed in another frame.

```
<frameset cols="50%,*" rows="50%,*">
    ⋮  <!-- Frame Details -->
</frameset>
```

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

- No. of frames and its layout of a web browser will decide by rows and cols attribute.

- Frame:
  - the content of the frame is specified
  - It will display the content of frameset.

```
<frame  name = "string"  src = "file name"
        scrolling = "yes"/"auto"/"no"
        frameborder = "0"/"1" />
```

CSS :-

- Acronym of CSS is cascading style sheets.
- HTML is concentrate on the content rather than the presentation of content.
- CSS is a markup language used in web document for presentation purpose.
- Primary intension of CSS was to seperate out the web document content from the presentation
- In the year 1996 – CSS1 by group of organizations (W3C)

  1998 – CSS2

- There are 3 Levels of CSS
  1. Inline style sheet
  2. Document Level style sheet
  3. External Level style sheet.

1. Inline Style Sheet :-
   - It is a kind of style sheet in which the styles can be applied to HTML tags.

Syntax :-

```
<tag-name  style = " property : value ; " >
```

Ex :-
```
<p  style = " font-family : arial ;
            color : red ; " >
        :
</p>
```

Example :-
```
<body>
    <p> This is simple text </p>
    <p style = " font-size = 30pt ; font-family : verdana ">
        Text is different </p>
</body>
```

Drawbacks :-

Adv. By using inline style sheet we can apply uniform styles on tags for whole document.

- Content of web pages is mixed with presentation.

2. Document Level style sheet :-

- It will appears in the head section and in the body section newly defined selector tags are used with actual contents.

Syntax : Style specification format

```
<style  type = " text/css " >
    selector
    {
        property : value ;
        property : value ;
            :
    }
</style>
```

Ex:-

```
<html>
  <head>
    <title> Document Level CSS </title>
    <style type="text/css">
      h1
       {
        font-family : arial;
        color : green;
       }
      h2
       { font-family : bookman old style;
        color : red;
       }
    </style>
  </head>
  <body>
    <center>
      <h1> Text color is Green </h1>
      <h2> Text color is Red </h2>
    </body>
  </html>
```

Drawbacks:-

- When we want to apply the style to more than one document at a time. it is not possible

Advantage:

- It will decide the presentation of webpage when we want to unique style for whole document

3- External Stylesheet:-

- Sometimes we need to apply particular style to more than one web document in such cases external style sheets can be used.

- the desired style sheet is stored in .css file.

- the name of file is mentioned in our webpage then the styles will be applied to our webpage.

Syntax: <link rel="stylesheet" type="text/css" href=".css" />

Ex:-

```
<head>
    <link  rel = "stylesheet"
           type = "text/css"
           href = "ex1.css" />
</head>
```

ex1.css:-

```
h1
{
    font-family : arial;
    color : green;
}
h2
{
    font-family : bookman old style;
    color : red;
}
```

- where we want to link the external style sheet then we want to use <link> tag.

link : tells to the browser some file must be linked to the web page.

rel : tells the browser the linked thing is stylesheet.

type : tells to the browser that the reading is text and which is affected by the css.

href : denote the path of stylesheet file.

Some of CSS categories properties & values :-

- fonts :

font-family : familyname ;

font-style : Normal | italic | Obliqueu ;

font-weight : Normal | bold | bolder | lighter ;

font-sizes : small | smaller | medium | large | larger ;

- Backgrounds and colors :

color : value ;

background-color : value ;

background-image : URL ;

background-repeat : repeat-X | No-repeat ;

background-position : center | top | bottom | left | right ;

- Text :

text-decoration : None | underline | overline | blink |
line-through ;

text-transformation : Uppercase | lowercase | capitalize | None ;

text-align : left | right | center | justify ;

line-height : length | percentage ;

letter-spacing : length | percentage ;

word-spacing : length | percentage ;

margin-right : 200px ;

margin-left : 300px ;

- link

a : link { color : green ; font-size : 18px ; }

a : hover { color : orange ; }

a : visited { color : blue ; }

a : active { color : black ; }

- List :-

list-style-type : roman | lower-greek | upper-latin ;

list-style-image : url("new.gif") ;

- Border :-

border-style : dashed | solid | double | dotted ;

border-color : color ;

border-width : top | bottom | left | right ;

- Table :-

width : length | percentage ;

border : solid | dotted ;

vertical-align : bottom | center ;

Ex :-

```
<style>
    h1, p
      {
        text-align : center ;
        color : red ;
      }
    h2
      {
        background-color : green ;
      }
    body
      {
        background-color : lightblue ;
      }
</style>.
```

Java Script

Introduction to Java Script :-

- Originally it is called as Livescript developed by Netscape in the year of 1995.
- Netscape & Sunmicrosystems jointly called as Javascript
- HTML is used for making static web pages
- A scripting language is used as a interpreted language to respond for user actions and makes web application interactive
- Java script is a scripting language &, it is light weight Programming Language
- Javascript is usually embedded directly into HTML Pages
- Javascript is an interpreted Language
- Java script can be divided into 3 parts
  - *core – Heart of Language (operations, expressions, stmts, subprograms)
  - *Client-side : collection of objects and control of browser interaction.
  - * Server-side : Collection of objects that makes language useful on web servers.
- Client side java script is an HTML embedded scripting language, we refer to every collection of javascript code as script.

Ex:-

```
<html>
  <body>
    <script type="text/javascript">
      document.write("Hello World");
    </script>
  </body>
</html>
```

JavaScript Variables:-

- A variable is a container for storing the information.
  A variable value can change during the script
- variable names are case sensitive.
- JavaScript has four primitive datatypes, they are number, string, boolean and NULL.
- Numbers are always stores floating point values
- Strings may enclosed between single quotes or double quotes
  EX: var name = "venky"
- Booleans are either True or False.
  False - 0, empty string, null, undefined, NAN, false
  True - other values.
- At the time of initializing a variable we can give null values
  EX: var age = null.

Operators :-

- Most of the syntaxs of Javascript borrowed from C.
- Arithematic operators (+, -, *, /, %)
  Comparison operators (<, <=, >, >=, !=, ==)
  Logical operators (&&, ||, !)
  Bitwise operators (&, |, ^, ~, <<, >>)

Javascript POPUP boxes:-

- We are having 3 types of POPUP boxes
  1. Alert box
  2. Confirm box
  3. Prompt box.

1. Alert box :-

- An alert box is used if we want information ~~from~~ to the user (~~reading~~)
- When alert box pops up, the user will have 'OK' to proceed.

Syntax :

```
<script>
    alert("Hello world");
</script>
```

2. Confirm box :-

- A confirm box is used if we want the user to verify or accepting something
- When a confirmation box popup, the user will have to click either OK or cancel to proceed.
- If user click on OK, the box return True. If the user click on Cancel, the box return False.

Syntax: `Confirm("Some thing");`

3. Prompt box :-

- A prompt box is used if we want user input value before entering a page.
- When prompt box popsup, the user will have to click either OK or cancel to proceed after entring an input value.
- If the user clicks on OK, the box returns the input value. If th user clicks Cancel, the box return NULL.

Syntax: `prompt("Sometext","default value");`

Ex :-

```
<script type="text/javascript">
    x = prompt("Enter a Number");
    document.write("Number is" +x);
</script>
```

Conditional Statements :-

- If we want to perform different actions for different decisions, you can use conditional statements.
- In Javascript we are having following conditional statements

* if statement :- If we want to execute some code only if a specific condition is true.
* if...else statement : If we want to execute some code if condition true otherwise execute other code.
* if...else..if...else statement : if we want to select one of many blocks of code.
* switch : If we want to select one of many blocks of codes.

Syntax :-

```
if (condition)
{
  //executed codes
}
```

```
if (condition)
{
  // true condition codes
}
else
{ // false condition codes
}
```

Loop Statements :-

- If we want to execute the codes repeatedly until the condition will false, you can use control statements.

```
while (condition)
{
  statements
}
```

```
do
{ statements
}while(condition);
```

```
for(intilization; condition; incre
{
  statements
}
```

Array Literals :-

- you don't declare the types of variables in javascript.
- Java script has array litterals, written with brackets and commas

EX : color = ["red", "yellow", "green", "blue"]

color[0] → red, yellow, green, blue

- creation of an array

1. var colors = ["red", "green", "blue"];

2. use new Array() to create empty array

```
var colors = new Array()
colors[0] = "red"   colors[1] = "green"
```

3. use new Array(n) to create array of size.

```
var colors = new Array(3);
```

- Array Functions:

* if myarray is an array

1. myarray.sort() - sorts the array alphabetical order
2. myarray.sort(function(a,b){return a-b;}) - Sorts the array numerically
3. myarray.reverse() - reverses the array elements
4. myarray.push(_) - Add a number or string at end of an array and increases array size
5. myarray.pop() - remove an element and returns last element of array and decreases array size
6. myarray.toString() - returns a string containing values of an array elements

Functions:-

- Functions should be defined in the <head> of HTML Page.
- Syntax:

```
function name(arg1, arg2 ---- argN)
{
  statements
}
```

- Function may contain return statement.

Predefined global functions:

1. isFinite() - determines whether a number is finite or not
2. isNAN() - returns true, if it is not a number
3. parseInt() - converts into numeric (int)

4. parseFloat() :- Convert strings to float value.

5. eval() :- for evaluate the expressions

Ex :- eval("2+2") = 4.

String Functions :-

1. match (pattern) - searches for matching pattern in array ~~or string~~.
2. replace (pattern1, pattern2) - search for pattern1, if search successful it replace by pattern2
3. search (pattern) - searches for a pattern in a string, if search successful it will return index position

Objects in javascript :-

- It is not a pure object oriented programming language, it uses the concepts of objects.
- By using new keyword we can create an object, it allocate memory and storage.
- Objects can have functions and variables
- objects have same name in global and local then we can call local objects by using this keyword
- Every object having some properties and methods. So we call properties by using . (dot) operator

objectname . property ;

- We are having some of the objects in javascript, they are

1. Document object
2. Form object
3. Browser object
4. Date object
5. Events
6. Math object
7. String object
8. Regular Expressions.

1. Document Object :-

- Document is a webpage being either created or displayed.
- Document have no. of properties that can be accessed through javascript programs and used to maintain content of webpage.
- write (function or property)
- HTML pages can also be created by using write object or method in document object.

```
document.write("<body>");
document.write("<h1> Hello </h1>");
document.write("</body>");
```

2. Form object :

- For user input data processing purpose we are using form object.

1. document.formName
   - It is name of the formName will displayed
2. document.form.formName
3. document.form[ ]
   - First form of element (in arrays)

3. Browser Object :

- Window objects :-

  alert :- Popup is alert the user

  Confirm :- Popup is confirm the user or not

  Prompt :- User will giving input

  Open :- for opening window

  Close :- for closing window

- History objects :-

  Property : length - displays length of codes in document

  Methods : forward() -> goes to next page

  back() -> goes to previous page

  Go -> goes to next or last page

- Navigator Object :- properties are

Navigator.appName : application name

navigator.appVersion : version of application

navigator.userAgent : Agent of the application

navigator.appCodeName : application ~~Lang~~ CodeName

navigator.language : Language of application

navigator.platform : Plat-form of application

- screen objects :

width

height

availWidth

availHeight

4. Date objects :-

- Information about current date
- Date object can be created naturally by new operator.

```
var today = new date();
```

today.getDate() — Day of (DD) month

today.getMonth() -

today.getFullYear() -

today.getDay() - Day of week

today.getTime()

today.getHours()

today.getMinutes()

today.getSeconds()

today.getMilliseconds()

5. Event objects :-

- To detect certain activities of the browser and user can perform some computational activities with browser
- computations can perform special form of programming language called event-driven Programing

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

- An event is a notification that something specific has occurred with the browser.
- An event handler is a script that is implicitly executed in response to the appearence of an event.
- the process of connecting an event handler to an event is called registration.
- An event handler is never used in document.write().

Events, Tag attributes, Tags :-

Event	Tag attribute	Tags
blur	- onblur	- a, input, textarea, select, button
change	- onchange	- input, textarea, select
click	- onclick	- a, input
focus	- onfocus	- a, input, textarea, select
load	- onload	- body
unload	- onunload	- body (Exit the document)
select	- onselect	- input, text area
submit	- onsubmit	- input, textarea
mousedown	- onmousedown	- (left the click)
mouse move	- onmousemove	(move in elements)
mouse over	- onmouseover	(cursor move over)
mouse out	- on mouseout	(away from element)
mouse up	- onmouseup	(unclick)

Used in any tag in html

6. Math Object:-

- It provides collection of properties and methods of number objects.
- ~~Prop~~ methods are sin(), cos(), floor(), round(), max()

Ex:- math.sin(x)

7. String objects:-

- For performing operations on strings

length - length of string
charAt(2) - At location 2 ~~item~~ char (display)
IndexOf('r') - position of string
substring(2,4) - display substring
toLowerCase(string) - convert to lowercase
toUpperCase(string) - convert to uppercase

8. Regular Expression Object:-

- It is ~~sequence~~ ~~search~~ characters that forms search pattern
- Search pattern used for text search and text replacement operations

Syntax: /pattern/modifiers

modifier	description
i	- Perform case insensitive matching
g	- Perform global match
m	- Perform multiple matching

Patterns:-

[abc] - find any char of a,b,c
[0-9] - find digits
(x|y) - find alternatives.

- We are also having some of the string methods they are

  search()

  replace()

- regular expression methods are

  test() - returns true or false

  exec() - search a string specified pattern, return found text

- ^ - Start of input

  $ - End of input

  * - 0 or more

  + - 1 or more

  ? - 0 or 1

  a/b - match a or b

  {n} - match the string

Dynamic HTML :-

- It is collection of technologies that allows dynamic changes to document display in HTML.

  1. Validation of Data

  2. Rollover Button

1. Data validation :-

- It is the process of ensuring that some data might be correct data for particular application.
- Data validation is the process of ensuring that the user submit only set of characters which you require.
- It is not the process of ensuring that the data is accurate or not.

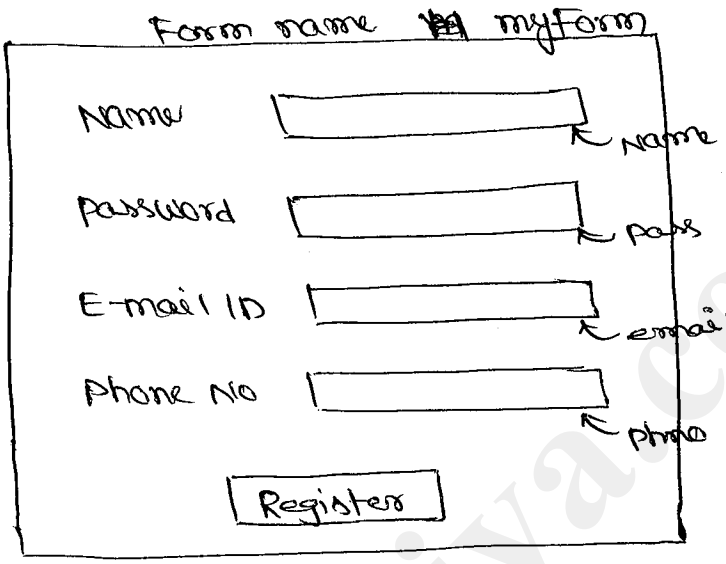
- In the application if you click on submit button then it will automatically calls the function of validate().

```
<input type="submit" value="Register"
    onClick="validate()" />
```

- The following script will validate data

```
<script language="javascript">
  function validate()
  {
    var field = /^[A-Za-z.]+$/;
    if(document.myform.name.value=="" ||
      field.test(document.myForm.name.value))
      {
       alert("Please provide your name");
      }
    if(document.myForm.pass.value=="" ||
      document.myform.pass.value.length<8)
      {
       alert("Please provide password");
      }
    var x= document.myform
    var x=document.forms["myForm"]["email"].value;
    var atpos = x.indexOf("@");
    var dotpos = x.lastIndexOf(".");
    if(atpos<1 || atpos+2>dotpos || dotpos+2>=x.length)
      {
        alert("Invalid email ID");
      }
   if(document.myForm.phno.value=="" ||
      document.myForm.phno.value.length!=10 ||
      isNaN(document.myForm.phno.value))
```

```
        {
            alert("Please provide phone number");
        }
    }
</script>
```



2. Rollover Buttons :-

- It is the technique used to give visual feedback about the location of the mouse cursor by changing the images on browser as the mouse move over them.
- Especially images are used as hyperlinks or image maps
- Let us take two image files, if the image is in inactive, in that mouse is not over else the image is in active, in that mouse is over image.

UNIT - II

Syllabus :-

- Working with XML
- DTD (Document type Definition)
- XML Schemas
- Presenting XML
- Document Object Model
- XML processors (DOM and SAX)

XML :-

- Acronym for XML is Extensible Markup Language.
- XML version 1.0, it was developed by group of Organizations W3C in the year of 1998.
- XML is similar to HTML.
- XML is subset of SGML
- XML is a scripting language means it contains various tags. these tags are not predefined tags but user can define his own tags.
- HTML is designed for representation of data on the webpages.
- XML is designed to store data or processing data.
- XML is used to describing the structure of document not the way of presentation.

Basic XML structure :-

```
<? xml version ="1.0" ?>
<college>
    <stddetails>
        <rollno> 12345 </rollno>
        <name>
            <firstname> abc </firstname>
            <lastname> xyz </lastname>
        </name>
        <branch> CSE </branch>
        <address>
            <street> sattenapalli </street>
            <city> Guntur </city>
            <country> India </country>
        </address>
    </stddetails>
```

- <?xml version = "1.0" ?>

This is called processing instruction, which tells applications how to handle XML because it contains rules of XML version = "1.0".

Valid or well formed :-

- Every document may be valid or wellformed.
- It is conformance between document, DTD and XML standards.
- A well formed document is one which follow all the rules of XML

  like (i) Tags are matched.
  (ii) Don't overlap the tags
  (iii) Empty elements are ended properly
  (iv) XML declaration.
- A valid XML document has its own DTD.
- The document is wellformed but also conforms to the rules set out in DTD.

XML Elements :-

- Every document composed of 3 things
  1. Elements
  2. Control Information
  3. Entities.

1. Elements :-

- Every document has a single root element which Contains all of the other markup.
- Document composed of number of sections, each section is enclosed blw tags.

Nesting Tags:-

- In XML document may have nested tags similar to html tags, the tags must be nested properly and close the tags in reverse of order they have opened.

Case Sensitive:-

- XML markup is case sensitive and it must be in lower case.
- In inside control information may use upper case but not in tags.

Empty Tags:-

- Element may be empty through if formatting data retrived from a database or entered by user.
- When content of element is missing then tag must be like inline tag

<content />

Attributes:-

- Attributes are used to specify the values of elements. these are specified with in double quotes.

2. Control Information:-

- There are 3 control structures

1. comments
2. Processing instructions
3. DTD declaration

comments:-

- XML commands are exactly same as HTML.
- the comments may be several lines or single line of a page

<!-- comment text -->

- The same type of comments used in XML source file and DTD files. ③

Processing Instructions :-

- These are used to control applications

<? xml version = "1.0" ?>

- It tells that the data in file follows the rules of XML version 1.0
- Every XML document should start with processing instruction then only the parser can understand to apply XML rules on file.

DTD Declaration :-

- Each XML document is associated with DTD. DTD is usually held in seperate file so that it can be used with many documents.
- DTD file holds the rules & grammar for a particular XML data structure

<!DOCTYPE college SYSTEM/PUBLIC "college.dtd" >

- It says that particular dtd file is used for College XML file
- Internationaly agreed DTD's are denoted by keyword PUBLI
- If we are developing our own own DTD files then it denoted by the keyword SYSTEM.

3. Entities :-

- Document may have one or more Entities
- An entity is athing which is to be used as part of doc but which is not a simple element.
- It is encrypted signature which is used frequently rather than that create some XML each time, we are creating

Document Type Definition (DTD):-

- DTD is set of structural rules called declarations, it specifies set of elements in the document and how and where these elements are appeared.
- DTD describes the structure of the document.
- The purpose of DTD is to define standard form of XML document.

Declaring Elements:-

- Each element declaration in DTD specifies, one category of an element.
- Declaration provides the name of element along with structure of element.
- An element is a node in a tree, either leaf node or internal node.
- If the element is leaf node then the declaration is

  <! ELEMENT element_name (#PCDATA)>
- If the element is internal node or root node then

  <! ELEMENT ele_name (list of child nodes)>
- In child node if any node having again childs then it will be indicated '+' symbol.
- PCDATA : parsable character Data.

Declaring Attributes:-

- Attributes are declaring seperatly from the element declaration.

  <! ATT LIST ele_name att_name att-type
  [Defaut_value] >
- Att-type is CDATA (string of characters)

- Default values are as follows

A value - Directly gave value like "<4".

#FIXED - Value for every element it can't be change

#REQUIRED - NO default value, must specify the value.

#IMPLIED - NO default value, may or may not specify the value (Optional).

Declaring Entities:-

- Entity is a container which will be filled with some form of content. the content may be include in XML file.

Internal Entity:-

- these are used to create small piece of data. which you want to use repeatedly throughout the document.

<!ENTITY entity_name "entity_value">

External Entity:-

- If the entity is longer than few words then text is defined outside of DTD.

<!ENTITY entity_name SYSTEM "file location">

- SYSTEM defines entity is in seperate file

Internal or External DTD:-

- the DTD can appear in XML insides or outside.
- If DTD is declared insides of XML file then

<!DOCTYPE root_name [- DTD - - -] >

- If DTD is declared outside of XML file i.e external

<!DOCTYPE root_name SYSTEM/ ".DTD file" >
PUBLIC

- DTD Example program

```
<!ELEMENT college (stddetails+)>
<!ELEMENT stddetails (rollno, name+, branch,
                      address+)>
    <!ELEMENT rollno (#PCDATA)>
    <!ELEMENT name (firstname, lastname)>
        <!ELEMENT firstname (#PCDATA)>
        <!ELEMENT last-name (#PCDATA)>
    <!ELEMENT branch (#PCDATA)>
    <!ELEMENT address (street, city, country)>
        <!ELEMENT street (#PCDATA)>
        <!ELEMENT city (#PCDATA)>
        <!ELEMENT country (#PCDATA)>
        <!ATTLIST country name CDATA #REQUIRED>
```

XML Schema :-

- Disadvantages of DTD
  1. The language used in DTD is unrelated to an XML doc. So XML processors are not analyzer the DTD files.
  2. DTD can't define type of data contained in XML doc. DTD can't specify wether the element is string or numeric.
- XML Schema is an XML document, it can be parsed with an XML parser.
- Data type of an element can be require any type (c++).
- There are 2 primary purposes for using schema.
  1. Schema specify the structure of XML doc, it includes elements and attributes in document.
  2. Data type of every Element.

- Including xml schema into XML file

```
<rootelement xmlns="http://www.w3.org/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="filename.xsd">
```

- Defining schema :-
  * schemas are written using collection of names
  * Every schema having schema as its root element.
  * the name of the namespace is
    http://www.w3.org/2001/XMLSchema
  * Namespace can be specified as
    xmlns:xsd = " ____ address ____ "

Data types :-

1. Simple data type
2. Complex data type

- Simple data types can't have attributes or nested elements.
- complex datatype can have attributes and include other datatype elements.
- XML Schema defines 44 datatypes
  - 19 primitive datatypes
    - string, boolean, float, time ----
  - 25 derived datatypes
    - byte, long, decimal, unsigned Int ---

1. Simple Data types :-

- Elements defined in xml schema with element tag which is from xml schema namespace.
- xsd used for names from the namespace.

```
<xsd:element name="name of element"
    type="datatype of element" />
```

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

```
<xsd:simpleType  name="firstname">
    <xsd:restriction  base="xsd:string">
        <xsd:maxLength  value="10" />
    </xsd:restriction>
</xsd:simpleType>
```

2. Complex Datatype :-

- complex types are defined with complexType tag.
- The sequence element is used to contain an ordered group of elements.

```
<xsd:complexType  name="string">
<xsd:sequence>
    <xsd:element/>
    :
</xsd:sequence>
</xsd:complexType>
```

Ex :-

```
<? xml version="1.0" ?>
<xsd:schema  xmlns="___"  version="1.0">
  <xsd:element  name="college">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element  name="stddetails">
        <xsd:complexType>
        <xsd:sequence>
          <xsd:element  name="rollno"  type="xsd:string"/>
          <xsd:element  name="name">
            <xsd:complexType>
            <xsd:sequence>
              <xsd:element  name="firstname"
                            type="xsd:string"/>
              <xsd:element  name="lastname"
                            type="xsd:string"/>
```

```
        </xsd:sequence>
        </xsd:complexType>
      <xsd:element name="branch" type="xsd:string"/>
      <xsd:element name="address">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="street" type="xsd:string"/>
          <xsd:element name="city" type="xsd:string"/>
          <xsd:element name="country" type="xsd:string"/>
        </xsd:sequence>
        </xsd:complexType>
    </xsd:sequence>
    </xsd:complexType>  } <!-- stddetails -->
    </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Namespace :-

- An XML namespace is collection of element names used in XML document.
- Create two different elements with same name but purpose of the element is different then we using name space.

```
<ele_name xmlns[:prefix] = URI >
```

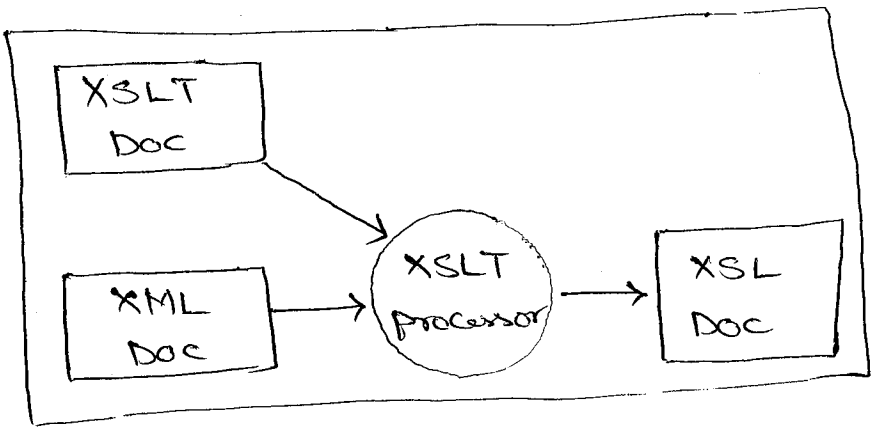
- URI - Uniform Resource Identifier.

Presenting XML or XSLT :-

- XSL (Extensible stylesheet Language) is a family of recommendations for defining XML document transformations and presentations.
- There are 3 standards in XSL
  - * XSLT – XSL Transformations
  - * Xpath – XML Path
  - * XSL-FO – XSL Formatting objects.
- XSLT is transforming XML document into HTML
- Xpath is language for navigation in XML docs we can reach any node of XML doc
- XSL-FO is a language for formatting XML doc for displaying it in desired manner.

Overview of XSLT :-

- The Syntactic structure of XSLT is XML
- XSLT processor takes the input from XML doc and XSLT Document.
- XSLT doc is executed and XML doc is the input data to the program.
- XML doc data is merged with XSLT doc then new doc output called XSL document



- XSLT model of processing XML data is called template-driven model. ⑦

- XSLT doc consists of one or more templates which use Xpath to describe element in XML doc to be processed.

- Each element is associated with it section of XSLT codes, which is executed when matched ~~Content~~ template is found in XML doc.

XSLT for presentation :-

- XSLT stylesheets can be used to control web page layout (or) formatting specifications.

- For including XSLT doc into XML doc, the processing instruction is

  <? xml-stylesheet type="text/xsl" href="file.xsl" ?>

- For creating XSLT doc, the processing instruction is

  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >

- A style document must include atleast one template element.

- A template opening tag contains attribute match to specify Xpath expression to select the node in XML doc.

- The content of template element is specified what is to be placed in output document.

- XSLT doc, template is include to match the root node of XML doc

  1. Xpath expression (/)
  2. Giving root node value.

<xsl:template match="/">

<xsl:template match="college">

- With in template element we can also write html code.
- The content of an element from xml doc is copied to output doc. This is done with value-of element. which uses select attribute specify element of xml doc whose content is copied.
- <xsl:value-of select="rollno">
- Select every xml element of a node set (internal nodes)

<xsl:for-each select="college/stddetails">

* college - it is root node
  stddetails - internal node or element

Example :-

```
<? xml version="1.0" ? >
<xsl:stylesheet xmlns:xsl="-------" >
  <xsl:template match="/" >
    <html>
      <body>
        <h> Student Database </h>
        <xsl:for-each select="college/stddetails" >
          <xsl:value-of select="rollno"/>
          <xsl:for-each select="name">
            <xsl:value-of select="firstname"/>
            <xsl:value-of select="lastname"/>
          </xsl:for-each>
          <xsl:value-of select="branch"/>
          <xsl:for-each select="address">
            <xsl:value-of select="street" />
            <xsl:value-of select="city" />
            <xsl:value-of select="country" />
          </xsl:for-each>
        </xsl:for-each>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

XML processors :-

- Primary goal of XML processor is to parse the given XML document.
- Java having in built API's for parsing given XML doc.
- Given XML data can be parsed in two ways
    1. SAX (Simple API for XML)
    2. DOM (Document object model).
- SAX is used to access the information of XML doc using sequence of events (or) streams of data.
- DOM is used to parse the XML doc using tree structure. We can access the information of XML doc by interacting tree node.

SAX :-

- SAX parsers are used to dealing with streams of data
- Data in XML doc, is passing from one place to other when parser is acting as intermediate-point.
- This model is used when parsing the XML doc across the network between applications and is widely used by Java programmers.
- SAX model is unsuited to websites, where repeated querying and updating of XML doc is required.

DOM :-

- DOM is an API (Application Program Interface) for XML doc.
- An API is set of data items, operations and properties.
- DOM defines a standard for accessing and manipulating Documents.

- DOM is seperated into 3 different levels

  * core DOM - Standard model for any structured doc
  * XML DOM - " for XML doc
  * HTML DOM - " for HTML doc.

XML DOM :-

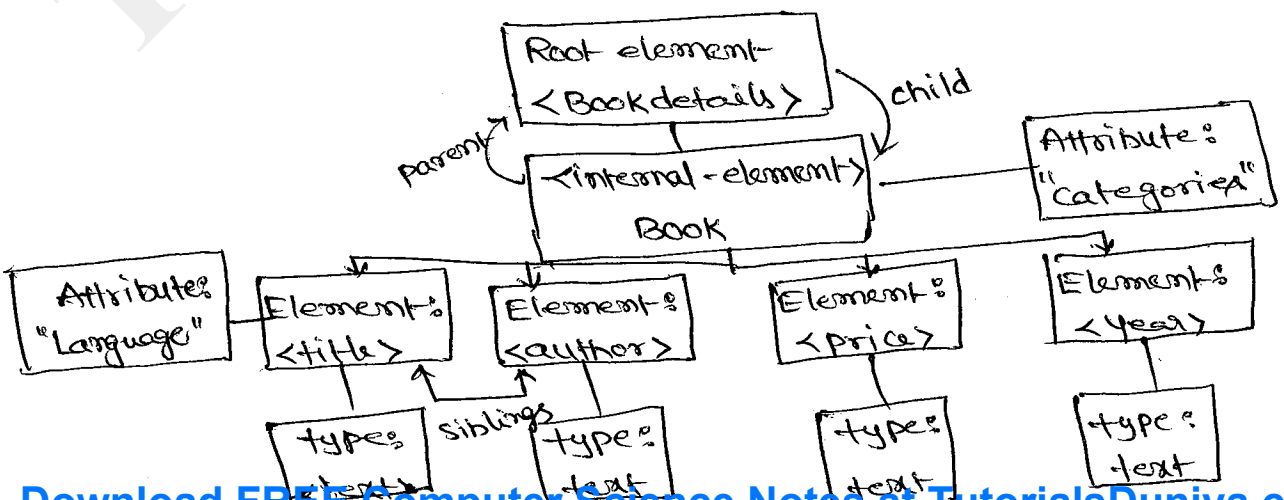
- XML DOM presents XML doc as tree - structure.
- XML DOM defines a standard way for accessing and manipulating XML doc
- All XML elements can be accessed through XML DOM
- XML DOM defines objects, properties and methods of all XML elements.
- XML DOM is platform and language Independent.

Some properties are :

- x.nodeName - Name of x
- x.nodeValue - value of x
- x.parentNode - parent of x
- x.childNodes - childs of x
- x.attributes - attributes of x

Some Methods are :

- x.getElementsByTagName(name)
  - get all elements with specified tag name
- x.appendChild(node)
  - insert a child node for x
- x.removeChild(node)
  - remove a child node from x.



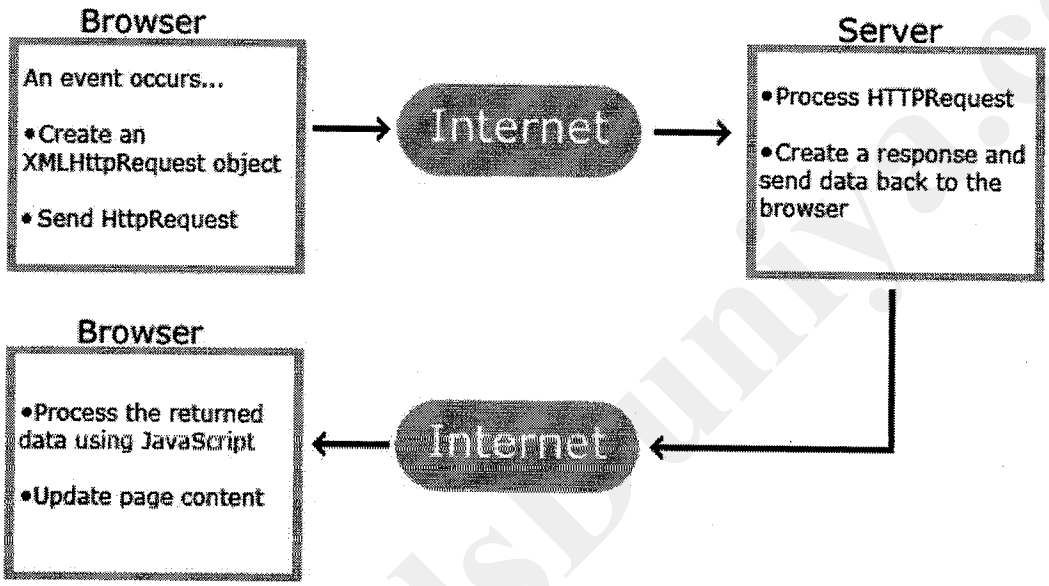
UNIT - III

AJAX

~~UNIT -8~~

What is AJAX?

AJAX = Asynchronous JavaScript and XML.
AJAX is a technique for creating fast and dynamic web pages.
AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.
Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.
How AJAX Works



Technologies Used in AJAX

JavaScript

- Loosely typed scripting language
- JavaScript function is called when an event in a page occurs
- Glue for the whole AJAX operation

DOM

- API for accessing and manipulating structured documents
- Represents the structure of XML and HTML documents

CSS

- Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript

XMLHttpRequest

- JavaScript object that performs asynchrous interaction with the server

Steps of AJAX Operation

1. A client event occurs
2. An XMLHttpRequest object is created
3. The XMLHttpRequest object is configured
4. The XMLHttpRequest object makes an asynchronous request to the Webserver.
5. Webserver returns the result containing XML document.
6. The XMLHttpRequest object calls the callback() function and processes the result.
7. The HTML DOM is updated

Example:

When a user selects a user in the dropdown list above, a function called "showUser()" is executed. The function is triggered by the "onchange" event:

```
<html>
<head>
<script>
function showUser(str)
{
if (str=="")
  {
  document.getElementById("txtHint").innerHTML="";
  return;
  }
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
 {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","getuser.php?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>

<form>
<select name="users" onchange="showUser(this.value)">
<option value="">Select a person:</option>
<option value="1">Peter Griffin</option>
```

```
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>
<br>
<div id="txtHint"><b>Person info will be listed here.</b></div>

</body>
</html>
```

The showUser() function does the following:

- Check if a person is selected
- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a file on the server
- Notice that a parameter (q) is added to the URL (with the content of the dropdown list)

The PHP File

- The page on the server called by the JavaScript above is a PHP file called "getuser.php".
- The source code in "getuser.php" runs a query against a MySQL database, and returns the result in an HTML table:

```
<?php
$q=$_GET["q"];

$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = '".$q."'";

$result = mysql_query($sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";
```

```
while($row = mysql_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "<td>" . $row['Age'] . "</td>";
  echo "<td>" . $row['Hometown'] . "</td>";
  echo "<td>" . $row['Job'] . "</td>";
  echo "</tr>";
  }
echo "</table>";

mysql_close($con);
?>
```

Explanation: When the query is sent from the JavaScript to the PHP file, the following happens:

1. PHP opens a connection to a MySQL server
2. The correct person is found
3. An HTML table is created, filled with data, and sent back to the "txtHint" placeholder

Ajax Security : Server Side

- AJAX-based Web applications use the same serverside security schemes of regular Web applications
- You specify authentication, authorization, and data protection requirements in your web.xml file (declarative) or in your program (programatic)
- AJAX-based Web applications are subject to the same security threats as regular Web applications

Ajax Security : Client Side

- JavaScript code is visible to a user/hacker. Hacker can use the JavaScript code for inferring server side weaknesses
- JavaScript code is downloaded from the server and executed ("eval") at the client and can compromise the client by mal-intended code
- Downloaded JavaScript code is constrained by sand-box security model and can be relaxed for signed JavaScript

Web Services :

web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. Because all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows applications can talk with Unix application

UNIT -8

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

To summarize, a complete web service is, therefore, any service that:

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

How Does it Work?

The steps illustrated above are as follows:

1. The client program bundles the account registration information into a SOAP message.
2. This SOAP message is sent to the Web Service as the body of an HTTP POST request.
3. The Web Service unpacks the SOAP request and converts it into a command that the application can understand. The application processes the information as required and responds with a new unique account number for that customer.
4. Next, the Web Service packages up the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
5. The client program unpacks the SOAP message to obtain the results of the account registration process. For further details regarding the implementation of Web Services technology, read about the Cape Clear product set and review the product components.

Components of Web Services?

The basic Web services platform is XML + HTTP. All the standard Web Services works using following components

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language

SOAP

SOAP is an XML-based protocol for exchanging information between computers.

All statements are TRUE for SOAP

- SOAP is acronym for Simple Object Access Protocol
- SOAP is a communication protocol
- SOAP is designed to communicate via Internet
- SOAP can extend HTTP for XML messaging
- SOAP provides data transport for Web services
- SOAP can exchange complete documents or call a remote procedure
- SOAP can be used for broadcasting a message
- SOAP is platform and language independent
- SOAP is the XML way of defining what information gets sent and how

Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls transported via HTTP.

SOAP enables client applications to easily connect to remote services and invoke remote methods.

Other frameworks, including CORBA, DCOM, and Java RMI, provide similar functionality to SOAP, but SOAP messages are written entirely in XML and are therefore uniquely platform- and language-independent.

A SOAP message is an ordinary XML document containing the following elements.

1. **Envelope:** (Mandatory)
   Defines the start and the end of the message.
2. **Header:** (Optional)
   Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end point.
3. **Body:** (Mandatory)
   Contains the XML data comprising the message being sent.
4. **Fault:** (Optional)
   An optional Fault element that provides information about errors that occurred while processing the message

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<SOAP-ENV:Header>
 ...
 ...
</SOAP-ENV:Header>
<SOAP-ENV:Body>

 ...
 ...
 <SOAP-ENV:Fault>
  ...
```

```
...
 </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

WSDL:

- WSDL stands for Web Services Description Language
- WSDL is an XML based protocol for information exchange in decentralized and distributed environments.
- WSDL is the standard format for describing a web service.
- WSDL definition describes how to access a web service and what operations it will perform.
- WSDL is a language for describing how to interface with XML-based services.
- WSDL is an integral part of UDDI, an XML-based worldwide business registry.
- WSDL is the language that UDDI uses.
- WSDL was developed jointly by Microsoft and IBM.

WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet. A client program connecting to a web service can read the WSDL to determine what functions are available on the server. Any special datatypes used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the functions listed in the WSDL.

Following are the elements of WSDL document. Within these elements are further subelements, or parts:

- **Definition:** element must be the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.
- **Data types:** the data types - in the form of XML schemas or possibly some other mechanism - to be used in the messages
- **Message:** an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.
- **Operation:** the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message
- **Port type :** an abstract set of operations mapped to one or more end points, defining the collection of operations for a binding; the collection of operations, because it is abstract, can be mapped to multiple transports through various bindings.
- **Binding:** the concrete protocol and data formats for the operations and messages defined for a particular port type.
- **Port:** a combination of a binding and a network address, providing the target address of the service communication.
- **Service:** a collection of related end points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.

In addition to these major elements, the WSDL specification also defines the following utility elements:

- **Documentation:** element is used to provide human-readable documentation and can be included inside any other WSDL element.
- **Import:** element is used to import other WSDL documents or XML Schemas.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

The WSDL Document Structure

```
<definitions>
<types>
  definition of types.....
</types>

<message>
  definition of a message....
</message>

<portType>
    <operation>
      definition of a operation.....
    </operation>
</portType>

<binding>
  definition of a binding....
</binding>

<service>
  definition of a service....
</service>
</definitions>
```

Example

```
<message name="getTermRequest">
 <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
 <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
 <operation name="getTerm">
  <input message="getTermRequest"/>
  <output message="getTermResponse"/>
 </operation>
</portType>
```

UDDI:

UDDI is an XML-based standard for describing, publishing, and finding Web services.

- UDDI stands for Universal Description, Discovery and Integration.
- UDDI is a specification for a distributed registry of Web services.
- UDDI is platform independent, open framework.
- UDDI can communicate via SOAP, CORBA, Java RMI Protocol.
- UDDI uses WSDL to describe interfaces to web services.
- UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.
- UDDI is an open industry initiative enabling businesses to discover each other and define how they interact over the Internet.

UDDI has two parts:

- A registry of all a web service's metadata including a pointer to the WSDL description of a service
- A set of WSDL port type definitions for manipulating and searching that registry

A business or company can register three types of information into a UDDI registry. This information is contained into three elements of UDDI.

These three elements are :

(1) White pages:

This category contains:

- Basic information about the Company and its business.
- Basic contact information including business name, address, contact phone number etc.
- A unique identifiers for the company tax IDs. This information allows others to discover your web service based upon your business identification.

(2) Yellow pages:

This category contains:

- This has more details about the company, and includes descriptions of the kind of electronic capabilities the company can offer to anyone who wants to do business with it.
- It uses commonly accepted industrial categorization schemes, industry codes, product codes, business identification codes and the like to make it easier for companies to search through the listings and find exactly what they want.

(3) Green pages:

This category contains technical information about a web service. This is what allows someone to bind to a Web service after it's been found. This includes:

- The various interfaces
- The URL locations
- Discovery information and similar data required to find and run the Web service.

C.S.E - IV Year

UNIT - IV

PHP

Syllabus :-

- Introduction of PHP
  - * Creating PHP script
  - * Running PHP script
- Working with variables and constants
  - * Using variables
  - * Using constants
  - * Data types
  - * Operators
- Controlling program flow
  - * conditional statements
  - * control statements
- Arrays
- Functions
- Working with Forms and Databases
  - * MySQL
  - * Oracle
  - * SQL Server.
- Regular Expressions (out of syllabus)

Introduction of PHP:-

- PHP was developed by Rasmus Lerdorf in the year of 1994.
- Initially PHP was developed by Zend Technologies.
- PHP reference implementations developed by group of organizations called PHP.
- PHP recursive acronym is "Hypertext Preprocessor"
- PHP is Serverside scripting language. It is embedded in HTML.
- PHP scripting language is influenced by C, C++, Java, Perl and TCl.
- It is used to manage dynamic content, database, session tracking and to build e-commerce sites.
- PHP is integrated with no. of databases like MySQL, Oracle, SQL Server, Postgre SQL, Sybase and Informix.
- PHP script is run on your web servers not on the user's browser. So there is no compatability issues.
- PHP file have the extension of ".php", ".php3", and ".phtml".

Common uses of PHP:-

- Handle the forms
- Allows to upload user files
- Querying databases.

- It can encrypt the data
- Restrict users to access some pages of your websites
- Accessing cookie variables and set cookies
- Shopping cart.

Creating and Running PHP Script :-

- To create PHP file simply follows the steps
  1. Create a new file in your favourite editor
  2. Type the PHP code
  3. Save the file with extension of ".php".
  4. Now we can access the php file or running php file by typing the URL on web browser.

http://localhost:portno/appname/path/filename.php.

Ex:- http://localhost:8080/test/sample.php

- The php extension tells the web browser to the server that needs to process the php file.
- If we save file with .html, then server won't understand php code, and browser will give the output like entire code.

Running from command line prompt :-

- We can run PHP script on command prompt
- Let the php program save with test.php.

```
<?php
    echo "Hello Php";
?>
```

- Now in Command prompt move the location of file and follow

```
$> php test.php
```

- It will produce the result as

```
Hello PHP
```

Types of declarations of PHP :-

- there are 4 ways to represent PHP code

1. Canonical php tags

```
<?php ..... ?>
```

2. SGML - Style tags

```
<? ..... ?>
```

3. ASP style tags

```
<% .... %>
```

4. HTML Script style

```
<script language = "php"> ..... </script>
```

- Generally we are using canonical PHP tags.

Comments :-

- Single line comments :-
  - * Generally used for short explanation or notes
  - * Either we can use "#" or "//"
- Multiline comments
  - * Generally used for pseudo code algorithm or explanation when necessary
  - * we can use like in C

```
/* ..... */
```

→ PHP is white space insensitive

EX:- $four = 2+2;

$four = 2+
2;

$four = 2 + 2;

* It will give same result of 4.

→ PHP is case sensitive.

EX:-
```
<?php
    $capital = 67;
    print ("variable is : $capital");
    print ("variable is : $CAPITAL");
?>
```

Result: variable is 67
variable is

→ Every statement and expressions must ends with semicolon (;)

→ Braces makes the blocks

* If we are having sequence of statements anywhere should enclosed between { } (curly braces).

Output functions in PHP:-

- there are 4 types of functions we are used in PHP.

1. print() :- It will display the content of webpage and it returns a boolean value like true (or) false.

EX:-
```
<?php
    print("welcome");
?>
```

2- echo :- echo will print multiple statements atonce, it will not return any value.

- The performance is faster than print().

```
Ex:- <?php
        echo ("Welcome", "PHP");
     ?>
```

3. printf() :- At the time of specifying format specifiers in print statement we are using printf()

- But in real time projects we never used printf().

```
Ex:- <?php
      printf("%.s wants %d apples", "name", 10);
     ?>
```

4. sprint() :- Instead of values displaying on webpage it will stores the return value in a variable

```
Ex:- <?php
      $a = sprintf("%.s wants %d apples", "name", 10);
      echo $a;
     ?>
```

** Variables :-

- Variables are used to store the information in middle of program.
- All PHP variables are preceded by "$" symbol.
- For assigning the values into variables we are using assignment operator (=), it have left variable name and right its value.
- Value of variable is most recent assignment value.
- Variables should be declare before assignment
- PHP variables can be converted automatically from one datatype to another when it is necessary

- PHP variables are like perl variables
- A variable is always start with characters it may be combination of char and digits, one special symbol is _ (underscore).

Types of variables in PHP (or) Scope :-

- the Scope can be defined as the range of availability of variable in program in which it is declared.
- there are 4 types of variables
  1. Local variables
  2. Global variables
  3. Static variables
  4. Function parameters

1- Local variables :-

- A variable declaration with in the function, the scope of availability is the function only.

Ex:-

```
<? php
    $a = 4;
    function $fun1()
    {
        $a = 100;
        echo $a;      // 100
    }
    fun1();
    echo $a;          // 4.
?>
```

Result :-

100

4.

2. Global variables :-

- A variable declared in global location, the scope of availability is entire the script.
- We can't access global variables directly.
- We can access global variables by using $GLOBAL() function, Otherwise we need to redeclare the global variable with in function by GLOBAL keyword.

```
<?php
$x = 4;
function fun1()
{
GLOBAL $x;
$x++;
echo $x;
}
fun1();
?>
```

Result

5

```
<?php
$x = 4;
function fun1()
{
echo $GLOBAL(x);
}
fun1();
?>
```

Result :- 4.

3. Static variables :-

- Static is a keyword (STATIC)
- Generally static variables are used to maintain the state of an application because we can assign the values into static variables only one time.

```
<?php
function fun1()
{
STATIC $count = 0;
$count++;
Print $count;
}
fun1();
fun1();
fun1();
?>
```

Result

1
2
3

4. Function parameters :-

- A function definition can have parameters
- A function parameters are local means those variables can we access with in function.

Ex :-

```
<?php
    function fun1($name)
    {
        print("welcome:$name");
    }
    fun1("Ramana");
?>
```

Result :

Hello Ramana

** Constants :-

- Constant is a name or identifier for simple value.
- A constant value can't be change during the execution of script
- It is case-sensitive, generally we are taking Uppercase.
- A constant name start with char or "_", it may be combination of character and digits
- NO need to mention $ with constant.
- TO define the constant we have to use define(), arguments are variable name and value.
- We can also use constant(), to print or read the constant value dynamically. So constant() return value is constant.
- Only scalar datatype (boolean, int, float, string) values can be stored in constant.

Ex:-

```
<? php
    define ('MINSIZE', 50);
    echo MINSIZE ;
    echo Constant ("MINSIZE");
?>
```

Result

50
50

- Differences between constants and variables :-
  * No need to write $ symbol before constant
  * Constant can't be defined by simple assignment, they are defined using define() function.
  * Constants can be defined and accessing anywhere without regard to variable scope.
  * Once constants have been set, may not be redefined.

- Magical constants :-
  * PHP provides a large no. of predefined constants to any script which it runs
  * these are case-sensitive.
  * there are 8 Magical constants that depending on where they are used.

LINE	- current line no. in a file.
FILE	- Full Path and name of the file
FUNCTION	- name of function
CLASS	- name of class
METHOD	- name of method name
DIR	- name of directory

Datatypes :-

- PHP supports 8 datatypes, it can be categorised into 3 types.

1. Scalar Type : Integer, Floating-point, Strings, Boolean
2. Compound Type : Arrays and objects
3. Special Type : NULL and Resources.

Integer :-

- These are whole numbers, without decimal point
- These are simple type, it may be +ve or -ve.
- Integers can be assigned to variable or expressions are also assigned
- Integers can be written in decimals, binary, octal and Hexa decimal numbers.
  * Decimal format is default.
  * Binary numbers are leading by 0b (0,1)
  * Octal numbers are leading by 0 (0-7)
  * Hexa decimal numbers are leading by 0x (0-9, A-F)
- Range of integers is $-(2^{**}31-1)$ and $(2^{**}31-1)$
  $-2,147,483,647$ to $+2,147,483,647$

Floating point :-

- Floating point numbers represent numeric values with decimal digits.
- Range of floating numbers is 1.7E-308 to 1.7E+308 (15 digits)
- PHP recognizes floating point numbers in two ways
  1. Normal format 3.41, 0.017.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

2. Scientific notation

0.314E1 => $0.314 \times 10^1 = 3.14$

$17.0E-3 => 17.0 \times 10^{-3} = 0.017$.

Strings :-

- String is a sequence of characters
- String literals are delimited by single-quoted (' ') or double-quotes (" ").
- Variables are expanded with in double-quotes only.
- Some of the escape sequence characters

\" - double quotes	\$ - dollar sign
\n - newline	\{ - left brace
\t - tabspace	\} - right brace
\\ - back slash	\[- left bracket
	\] - right bracket

Boolean :-

- Boolean have two values either true or false
- PHP provides the constants for boolean TRUE or FALSE

```
if (TRUE)
    { // True stmt }
else { // False stmt }
```

- We can determine truth values
  1. Value of a number, if 0 - False else TRUE
  2. Value of a string, if empty - False else TRUE
  3. Value of Type NULL is always FALSE
  4. Array with zero elements - False else True.

Arrays :-

Array is collection of homogeneous elements.

- Array can be store the values based upon index or associative index.

```
$person[0] = "Rama";
$people['name'] = "Lakshmi";
```

- The array can also create using array() function.

```
$person = array(1,2,3,4);
$people = array("Firstname" => "Ramana",
                "Lastname" => "Lakshmi");
```

Objects :-

- PHP supports object oriented programming (OOP)
- Objects are the run time instances of a class.
- We can create any no. of objects by using class name with keyword of new.
- By using objects we can access properties and methods

Ex:-
```
$ex = new Example;
$ex -> name('Rama');
echo "Hello, {$ex->name}\n";
```

Result
Hello Rama

Special Types :-

⇒ NULL :-

- It have only one value NULL.
- It is case insensitive
- It will represent a variable that has no value
- Ex:- null, NULL, Null all are same

⇒ Resources :-

- Resources are special types
- Resources are used for database connectivity.

** Operators :-

- there are 5 types of operators used in PHP

1. Arithmetic operators
2. comparison operators
3. Logical operators or Relational operators
4. Assignment operators
5. conditional operators. (Ternary)

Arithmetic operators :-

- Arithmetic operators are used to perform arithmetic operations like addition, subtraction, division etc.
- the following arithmetic operations supported by PHP
- Let assume A = 10, B = 20 then

Operator	Example	Description
+	A+B will give 30	Adds two operands
-	A-B will give -10	Subtract B from A
*	A*B will give 200	multiply both operands
/	B/A will give 2	Divide numerator by denominator
%	B%A will give 0	- It will return remainder after division
(+)	+(A) will give +10	- operand multiplid by +1
(-)	-(A) will give -10	- operand multiplied by -1
++	A++ will give 11	- increase operand value by 1
--	A-- will give 9	- decrease operand value by 1

Comparison operators :-

- It will compare the operands, the result is always either true or false.
- Let assume A=10 and B=20

Operator	Example	Description
==	A==B is false	- If both operands are equal, it returns true otherwise false
!= or <>	A!=B is true	- If both operands are not equal, it returns true otherwise false
>	A>B is false	- If left operand is greater than right operand, it returns true else false.
<	A<B is true	- If left operand is less than right operand, it returns true else false.
<=	A<=B is true	- If left operand is less than or equal right operand, it returns true else false
>=	A>=B is false	- If left operand is greater than or equal right operand, it returns true else false

Logical operators :-

- It provides the way to build complex logical expressions
- It takes operands as boolean values and return boolean value
- Let assume A = 10 and B = 20

Operator	Example	Description
&& (and)	A&&B is true	- Logical AND, if both the operands are non zero the condition true
|| (or)	A || B is true	- Logical OR, if any one of operand is non zero the condition will be true
! (not)	!A is false	- Logical NOT, reverse of boolean value already existed.

Assignment operators :-

- Assignment operators store or update values in variables
- The basic assignment operator (=).
- Let assume the following are supported by PHP

Operator	Description
=	- Simple assignment, assign right value to left side Operand.
+=	- It assigns left operand and right operand to the left operand.
-=	- It assigns left operand subtract right operand & result assign to left operand
*=	- It assigns left operand multiplied by right operand and result assign to left operand
/=	- It assigns left operand is divide by right operand and result assign to left operand.
%=	- It assigns left operand is divide by right operand the remaind value assign to left operand

Conditional operators:-

- It also called as ternary operator
- It contains 3 operands
- Syntax:- $result = expr1 ? expr2 : expr3 ;
- If expr1 is true the expr2 value is assign to result variable, else expr3 value is assign to result variable

EX:- $a=10, $b=20

$big = $a > $b ? $a : $b

- $big value is $b = 20.

** Conditional statements :-

- It also called decision making statements
- PHP supports 3 decision statements

1. If ... else statement
2. elseif statement
3. Switch statement.

⇒ If ... else statement :-

- If the condition is true it will execute if block code otherwise execute other set of codes.

Syntax :-

```
if (condition)
  {
    .....
    .....
  }
else{
      ....
      ....
    }
```

Ex :-

```
<? php
$a=10; $b=20;
if ($a > $b)
    echo "a is bigger";
else
    echo "b is bigger";
?>
```

Result :- b is bigger

⇒ If ... elseif ... else :-

- It is used with if ... else statement to execute code if one of several conditions are true

Syntax :-

```
if (condition)
  {
    ....
    ....
  }
elseif(condition)
    {  ....
    }  ....
       ....
else
    {  ....
       ....
    }
```

Ex :-

```
<? php
$a=10; $b=20;
if($a > $b)
  echo "A is bigger";
elseif ($a < $b)
    echo "B is bigger";
else
    echo "A and B equal";
?>
```

Result :- B is bigger.

⇒ Switch :-

- Switch is used if you want to select one of many blocks of code to be executed
- Switch block avoids long blocks of if ... elseif .. else code.

Syntax:-

```
switch (expression)
{
  case label1 :
        ----
        ----
        break;
  case label2 :
        ---
        ---
        break;
    :
  default : -----
}
```

Ex:-

```
<?php
$color = ---;
switch ($color)
{
  case "red":
       echo "red";
       break;
  case "yellow":
       echo "yellow";
       break;
    :
  default : echo "No color";
}
```

**Control Statements:-

- It is also called as looping statements.
- Loops are used to execute the same block of codes for specified number of times
- PHP supports the following four types of loops

  1. for 2. while 3. do...while 4. foreach

- It supports two control statements break and continue.

⇒ For loop:-

- It is used when we know how many times we want to execute statements or block of statements.

Syntax:-

```
for(initialization ; condition ; Inc/Dec)
{
  ---
  ---
  ---
}
```

Ex:-

```
<?php
//$a=0;
//$b=0;
for($i=0; $i<5 ; $i++)
{ echo $i;
  $i++;
}
?>
```

Result:-

0, 1, 2, 3, 4

⇒ While loop :-

- while statement will execute a block of code as long as test the condion.
- If the condition true then the block of code will execute Otherwise it will stop. It will repeate untic condition false.

Syntax :-

```
while (condition)
  {
    - - - -
    - - - -
    - - -
    Inc/Dec;
  }
```

Ex :-

```
<? php
    $i = 5;
    While ( $i)
    {
      echo $i;
      $i--;
    }
?>
```

Result :- 5, 4, 3, 2, 1

⇒ do.... while loop :-

- It will execute the block of codes at least once, then it will repeat the block until the condition will true

Syntax :-

```
do
 {
   - - - - -
   - - - -
   - - -
 } while(condition);
```

Ex :-

```
<? php
    $i = 5;
    ~~while(~~
    do
     {
       echo $i;
       $i--;
     } while ($i);
?>
```

Result :- 5, 4, 3, 2, 1

⇒ foreach statement :-

- It will used for arrays. Every pass it will read the value of an array and assigned to another variable, it will pass until all the elements in array will read.

Syntax :-

```
foreach (array as value)
  {
    - - - -
    - - - -
  }
```

Ex :-

```
<? php
$array = $array(1,2,3,4,5);
foreach ($array as $value)
  {
    echo $value;
  }
```

Result 1, 2, 3, 4, 5

=> Break statement :-

- It is used to terminate the current iteration permanantly.

- Ex:-

```
<?php
    $i = 0;
    while ($i < 10)
    {
        $i++;
        if ($i == 3)
            break;
    }
    echo "Loop stopped at $i";
?>
```

Result :-

Loop stopped at 3

=> Continue statement :-

- It is used to halt the current iteration of loop but it is not ~~iterat~~ terminated.

Ex:-

```
<?php
    $i = 0;
    while ($i < 5)
    {
        $i++;
        if ($i == 3)
            continue;
        echo "value is $i";
    }
?>
```

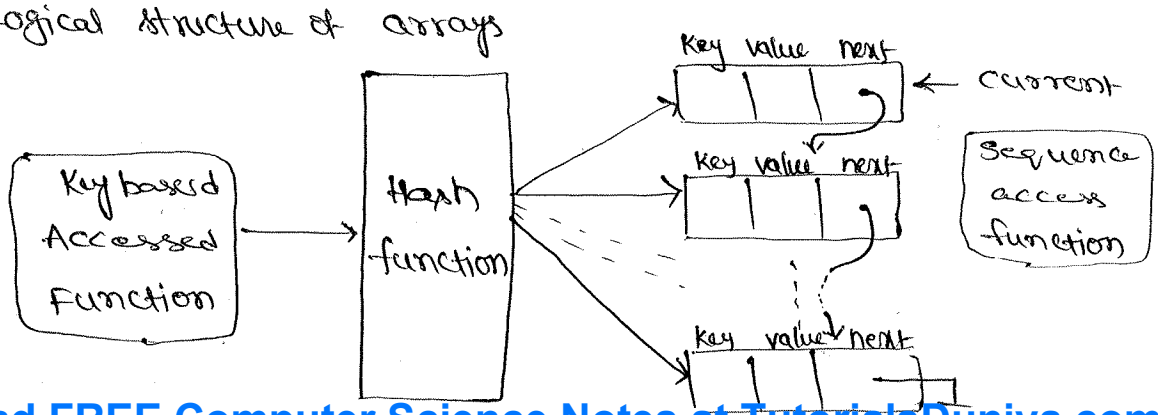
Result :-

value is 1
value is 2
value is 4
value is 5

** Arrays :-

- Array is a datastructure it stores one or more similar types of values in a single variable

-> Logical structure of arrays



- there are 3 kinds of arrays
- Each array value is accessed using ID which is called Index.

1. Numerical arrays:-
   - An array with numeric index, values are stored and accessed in linear fashion.
2. Associative arrays:-
   - An array with string index, it stores the element values associated with key rather than sequential
3. Multidimensional array:-
   - An array containing one or more arrays and values are accessed using multiple indices (index's)

=> Numerical Arrays:-

- It can store numbers, strings and objects but their indexes will be represented by numbers.
- Array index starts from zero
- TO create array we are using function i.e array
- we can access the elements based on index value

Ex:-

```
<? php
```

Method 1

```
$num = array(1,2,3)
foreach ($num as $val)
    echo "value is $val";
```

Result

1
2
3

Method 2

```
$num[0] = "one";
$num[1] = "two";
$num[2] = "three";
echo $num[0];
echo $num[1];
echo $num[2];
?>
```

One
two
three.

⇒ Associative Arrays:-

- Associative arrays are very similar to numeric arrays in terms of functionality but they are different interms of their ~~inex~~ index.
- Associative arrays will have their index as string so that we can establish strong association between key and value.
- Marks of a student (subject, marks)

Ex:-

```
<?php
```

Method 1

```
$num = array("phy" => 50, "maths" => 60);
echo $num['phy'];
echo $num['maths'];
```

Result :-

50
60

Method 2

```
$num['phy'] = 50;
$num['Maths'] = 60;
echo $num['phy'];
echo $num['maths'];
?>
```

50
60.

⇒ Multidimensional Arrays:-

- Each element in main array can also be an array, And each element in sub array can be array and soon.
- values in multidimensional array are accessed using multiple index.

Ex:-

```
<?php
$marks = array("Rama" => array("phy" => 50, "maths" => 60),
               "Lakshmi" => array("phy" => 40, "maths" => 70),
               "venky" => array("phy" => 80, "maths" => 60));
echo $marks['Rama']['phy'];
echo $marks['Lakshmi']['phy'];
echo $marks['venky']['maths'];
?>
```

Result

50
40
60

Functions :-

- PHP functions are similar to other programming language
- A function is a piece of code which take one or more input in the form of parameters and does some processing and return a value.
- Function having 2 parts
  1. Creation of PHP function
  2. Calling a PHP function

User-Defined function :-

- It is very easy to create
- While creating the function it should starts with keyword of function.
- All the sequence of statements can be writtern in block({ })

Ex:-
```
<? php
    function userfun()
    {
        echo "Hello, Php";
    }
    userfun();
    userfun;
?>
```

Result:
```
Hello, Php
Hello, Php
```

Function with parameters :-

- It allows to pass the parameters inside a function
- We can pass as many as parameters.

Ex:-
```
<? php
    function sum ($num1, $num2)
    {
        $sum = $num1 + $num2;
        echo "sum is $sum";
    }
    sum (10,20);
?>
```

Result
```
Sum is 30
```

Function with Return value:-

- A function can return a value using return.
- The return statements stops the execution of the function and sends the value back to calling code.

Ex:-

```
<?php
function sum($num1, $num2)
{
    $sum = $num1 + $num2;
    return $sum;
}
$value = sum(10,20);
print $value;
?>
```

Result
30

Passing arguments by reference:-

- It means that the reference to the variable is manipulated by a function rather than copy the value of variable

Ex:-

```
<?php
function fun1($num)
{
    $num += 5;
}
function fun2(& $num)
{
    $num += 6;
}
$var = 10;
fun1(&$var); # fun1(&$var);
echo "value of var is $var";
fun2($var);
echo "value of var is $var";
?>
```

Result:

value of var is 15
value of var is 21

Setting default values for functions:-

- We can set a parameter as a default value.
- If the function caller pass the values, it will print the passed values otherwise it will prints default values

Ex:-
```
<?php
    function myfun( $str = "Hello")
    {
        print $str;
    }
    myfun("Ramana");
    myfun();
?>
```

Result
Ramana
Hello.

Dynamic function call :-

- Assign the function names as a string into variables and treat those variables are exactly function name.

Ex:-
```
<?php
    function fun1()
    {
        echo "Hello";
    }
    $var = "fun1";
    $var();
?>
```

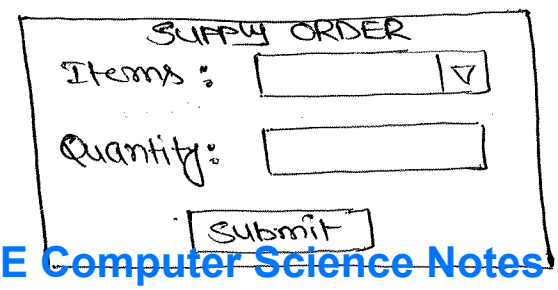
Result
Hello
Hello.

** ## Working with FORMS :-

- One of the most popular ways to make a website interactive is use of forms.
- With forms we can do user registrations and submit any information.

⇒ Create HTML FORM

- We are going to create one form for giving order of items and its quantity.
- If we are click on submit button it will display the order of items and its quantity by using PHP.



TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

order.html

```
<html> <body>
  <h1> Supply ORDER </h1>
  <form action="Procss.php" method="POST">
    Items:
    <select name="item">
      <option> paint </option>
      <option> Brushes </options>
      <option> Erasers </option>
    </select> <br>
    Quantity:
    <input type="text" name="qty"/> <br>
    <input type="submit" value="Submit"/>
  </form>
</body> </html>
```

Procss.php :-

```
<?php
  $qty = $_POST['qty'];
  $item = $_POST['item'];
  echo "your order is $qty with $item";
?>
```

Form Methods :-

- We are having two ways to send the browser information from client to server
  1. GET
  2. POST

GET :-

- The GET method sends the encoded user information appended to the page request
- The page and ~~code~~ encoded information seperated by ?

http://www.test.com/order.html?

Key1 = value1 & key2 = value2

- GET method will sends only 1024 characters
- Never used GET method for sensitive information like passwords
- PHP provides $_GET

POST :-

- The information is encoded as described in GET, and put it into query_string header.
- The information will send via HTTP header so it is secure
- It doesnt have any restriction of size
- To access the values PHP provides $_POST.

Database using MySQL :-

- MySQL is opensource database product and we can download easily from website.
- MySQL is a kind of database in which the records are stored as entities called Tables.
- In tables the data is arranged in the form of rows & columns
- We can query a database to retrive particular information.
- Query is a request for the database.

① Creating databases

$> CREATE database db_name;

- it will create one database on mySQL server.

② Display all databases

$> show databases;

- It will display the all databases are in mySQL server

③ Setting particular database

$> USE db_name;

- We can enter into the database.

④ Create User

$> Create User user_name;

- We can create user in mySQL server

⑤ permissions to user for particular Database

$> Grant all on db_name.* to

'user_name'@'localhost' identified by 'password';

⑥ Create a table

- We can create a table inside a database so before create table we can select any database(USE)

$> Create Table tb_name (

id int(4), name varchar2(20));

⑦ Display a table

- After creating a table using show command we can see all existing tables in current database

$> SHOW TABLES;

⑧ Inserting values into the table

- we can insert only one complete record at a time

$> insert into tb_name values (1,"Rama");

⑨ Display the content of the table

$> Select * from tb_name;

⑩ Display the structure of a table

$> DESC tb_name;

⑪ Updating the record

$> Update tb_name SET name="Lakshmi"

where id=1;

⑫ Deleting a record

$> Delete from tb_name where id=1;

⑬ Truncate a table

$> Truncate table tb_name;

⑭ Deleting table

$> Drop table tb_name;

PHP and MySQL Connectivity :-

- Web database Architecture work as follows

1. User sends HTTP request for particular webpage
2. Web server receives the request and retrive the files and passes it to PHP engine for processing.
3. PHP engine parsing the script, Inside the script is a command to connect database and execute Query.
4. MySQL server receives the database query process it and sends the result, sent ~~as HTML~~ to PHP engine
5. PHP engine finishes the running script, and it will involves in formatting HTML result and it return to webserver.
6. The webserver passes the HTML back to the webbrowser. where the user can see the result.

⇒ ① SETUP connection

- Before accessing the database mySQL ~~for~~ from PHP, it is necessary to establish a connection between them.

mysql_connect($servername, username, password)

EX:-

$conn = mysql_connect("localhost:3306", "test", "test123");

② Close connection

- we can close the connection by using

mysql_close();

③ Create database.

- we can execute the query by using

mysql_query("—", —);

④ Selecting database

- the database can be selected using function

mysql_select_db( );

EX:- mysql_select_db("mydb", $conn);

⑤ Listing the database

- we can display the databases existed in mySQL by using

```
mysql_list_db( );
```

⑥ Listing Tables

```
$tblist = mysql_query("show tables from mydb", $conn)
while( $i = $mysql_fetch_row($tblist))
    {
      echo $i[0];
    }
```

⑦ creating a table

```
$conn = mysql_connect("localhost:3306", "test", "test123");
mysql_select_db("mydb", $conn);
$query = "create table tb_name(id int(4), name varchar(20))";
mysql_query( $query, $conn);
```

⑧ inserting data

```
$query = "insert into tb_name values (1, "Rama")";
mysql_query( $query , $conn);
```

- inserting posted values

```
$query = "insert into tb_name values ('$_POST['myid']',
                                      '$_POST['name']')";
mysql_query ($query, $conn.
```

⑨ Display the table data

```
$result = "select * from tb_name";
$row = mysql_query( $result, $conn);
while( $row1 = mysql_fetch_array ($row))
    {
      echo $row['id'];
      echo $row['name'];
    } echo "<br>";
```

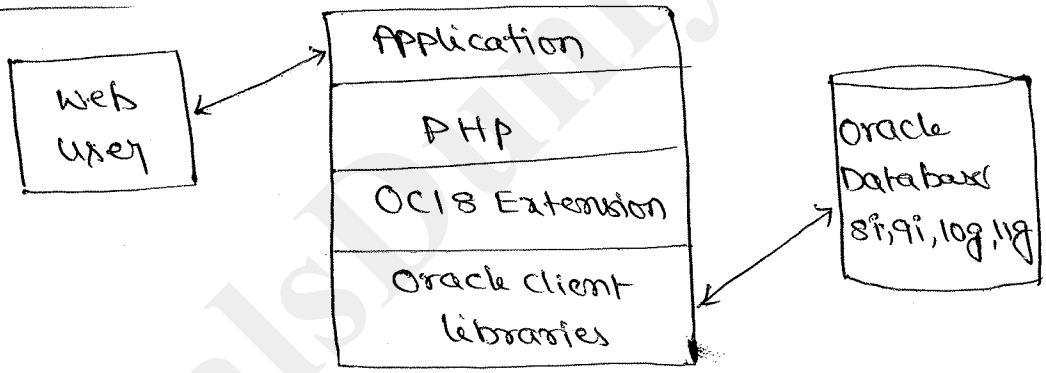
(10) Deleting record from table

```
$query = "Delete from tb_name where id=1";
mysql_query($query, $conn);
```

**PHP with Oracle Database :-

- PHP is server side scripting language designed for web development but also used as general purpose programming language.
- PHP is compatible with most of the databases including Oracle
- Any web based application can be easily developed in PHP

Architecture :-



Mid Tier 9iR2,10g,11g

OCI8 :-

- It have some functions used to access oracle databases 12c, 11g, 10g, 9i and 8i
- It supports SQL and PL/SQL statements
- Basic features including Transaction control, binding PHP variable to oracle placeholders and support for large objects (LOB) types and collections
- Oracle scalability features such as database resident connection pooling (DRCP) and result caching are also supported.

Connecting to Database :-

- To access the oracle database, first connect the database. Database could reside on local or remote server.
- Before starting the connection we require username, password and database.

Syn :-
```
<?php
$conn = oci_connect('un','pw',"localhost/XE");
```

- SQL Statements for execution

1. Parse : Prepares a statement for execution
   oci_parse()
2. Bind :- Optionally bind variables in where clause or to PL/SQL args
   oci_bind_by_name();
3. Execute : the database executes the statement and buffers the results.
   oci_execute();
4. fetch : Optionally retrives the results from database
   oci_fetch_array();

- Insert :
  - We can create dynamic insert query if the values are taken from the form to be inserted.

```
<?php
$stmt = oci_parse($conn,"insert into tb_name
        values (1,"Rama")");
oci_execute ($stmt);
?>
```

- SELECT
  - For retriving data generally we are using SELECT command.

- For storing the array values we need array and for retrieving arry values we are having

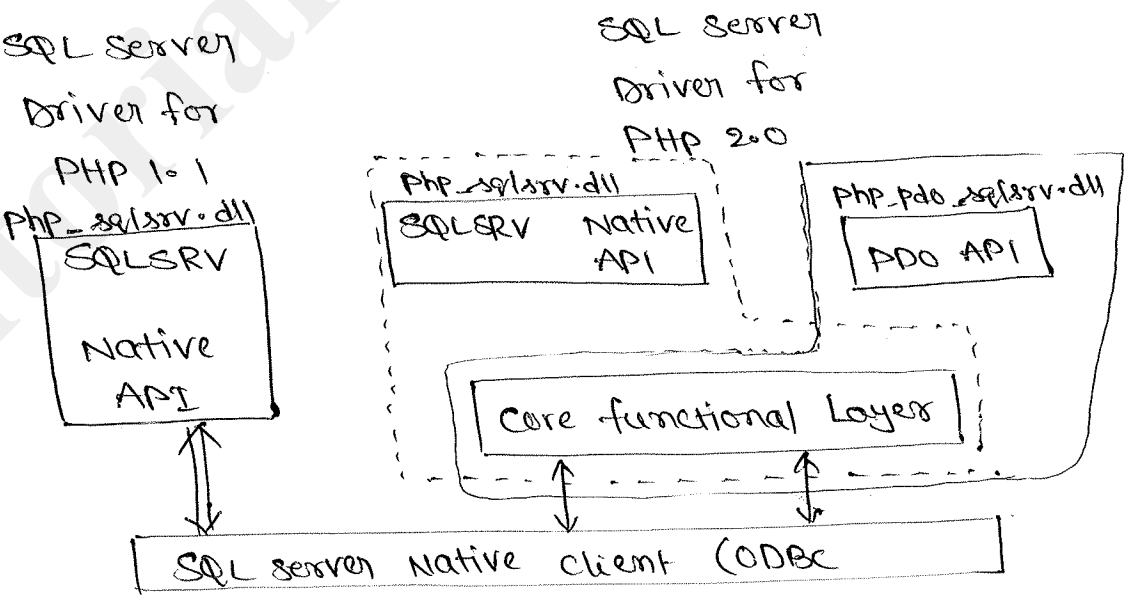
oci_fetch_array()

Ex:-

```
<? Php
    $stmt = oci_parse($conn,"select * from tb_name");
    oci_execute($stmt);
    while( $row = oci_fetch_array($stmt))
        foreach ($row as $val)
            print $val;
?>
```

** PHP with SQL Server:-

- PHP primary task has been taking data from the web and storing it into database. (or) retreiving data from database for displaying on webpage
- SQL server having different drivers for PHP 1.1 and PHP 2.0
- PDO(PHP data objects)



Accessing SQL server from PHP :-

- For accessing SQL server from PHP we need to create username, password and database.
- If we need to set SQL Server security then "SQL Server and windows authentication mode" will accept.
- For creating the connection with database

```
$conn = new PDO ("sqlsrv:server = $servername;
                  database = $database", $un, $pw);
```

- Once we have made our connection querying is an easy.

```
$sql = "select * from db.employee where id < 10";
$conn -> setAttribute (PDO::SQLSRV_ATTR_DIRECT_QUERY
                       => true);
$result = $conn -> query( $sql);
```

- Retriving data

```
$sql = "select id * from db.employee where id < 10 and
                                  firstname = :name";
$query = $conn -> prepare ($sql);
$query -> execute ( array (":name" => 'Rama'));
```

** Regular Expressions :-

- Regular expressions means sequence of characters
- It is used to search a particular string inside another string, we can replace one string by another string and we can split a string into many chunks (substrings)
- PHP supports 2 types of Regular expressions

1. POSIX Regular Expressions
2. perl style Regular Expressions

POSIX Regular Expressions :-

- the structure of a posix regular expression contains typical arithmetic expressions and characters and it form more complex expressions
- the simplest regular expression is one that matches a single character

Ex:- Search for g in bag

⇒ Brackets :-

- Brackets ([ ]) have a special meaning when used in the context of regular expression.

[0-9] - digits from 0 - 9
[a-z] - search a-z
[A-Z] - search A to Z
[a-Z] - search a to z and A to Z

⇒ Quantifiers :-

- the pattern characters are denoted by special characters such as +, *, ?, $, {int, range}

P+ - atleast one P
P* - 0 or more no. of P's
P? - similar to P*
P$ - End with P
^P - not start with P
P{N} - sequence of N-P's
P{2,3} - sequence of 2 or 3 P's
P{2,•} - sequence of atleast 2 P's

POSIX functions :-

- PHP offers some special functions for searching string's using POSIX_style regular expressions

1. ereg() :- search for patterns in a string, if it is existed return true otherwise false
2. ereg_replace() :- search for a pattern and replace the pattern with replacement.
3. Split() :- split strings into various elements
4. eregi()
5. eregi_replace()
6. spliti()

} Similar to previous but these are case insensitive

PERL STYLE :-

- Generally we are using slashes for pattern (/ /)

⇒ Metacharacters :-

- It is simply alphabet character preceded by backslash that acts to give the combination a special meaning

Character	description
.	- single character
\s	- white space (Tab, space, newline)
\S	- non-white space
\d	- digits (0-9)
\D	- Non-digit
\w	- word character
\W	- non-word character
[aeiou]	- matches single char in given set
[^aeiou]	- Outside of set
(foo|bar|baz)	- match for any alternative specified

=> Modifiers :-

i - case insensitive

O - evaluate expressions only once.

m - pattern matched against multiline text, so treat newline as normal character.

Perl style functions :-

1. preg_match() :- search for the string of a pattern
2. preg_match_all() : all occurences of pattern in string
3. preg_replace() : similar to ereg_replace() except regexp used in pattern and replacement input parameter
4. preg_split() : similar to ~~ere~~ split() except regexp used in pattern and replacement input parameters
5. preg_grep() : search all the elements of input_array returning all elements matching in regexp pattern
6. preg_quote() : Quote regexp character.

What is Perl?

Perl is a programming language. Perl stands for Practical Report and Extraction Language. You'll notice people refer to 'perl' and "Perl". "Perl" is the programming language as a whole whereas 'perl' is the name of the core executable. There is no language called "Perl5" -- that just means "Perl version 5". Versions of Perl prior to 5 are very old and very unsupported. Some of Perl's many strengths are:

- **Speed of development.** You edit a text file, and just run it. You can develop programs very quickly like this. No separate compiler needed. I find Perl runs a program quicker than Java, let alone compare the complete modify-compile-run-oh no-forgot-that-semicolon sequence.
- **Power.** Perl's regular expressions are some of the best available. You can work with objects, sockets...everything a systems administrator could want. And that's just the standard distribution. Add the wealth of modules available on CPAN and you have it all. Don't equate scripting languages with toy languages.
- **Usuability.** All that power and capability can be learnt in easy stages. If you can write a batch file you can program Perl. You don't have to learn object oriented programming, but you can write OO programs in Perl. If autoincrementing non-existent variables scares you, make perl refuse to let you.There is always more than one way to do it in Perl. You decide your style of programming, and Perl will accommodate you.
- **Portability.** On the Superhighway to the Portability Panacea, Perl's Porsche powers past Java's jaded jalopy. Many people develop Perl scripts on NT, or Win95, then just FTP them to a Unix server where they run. No modification necessary.
- **Editing tools** You don't need the latest Integrated Development Environment for Perl. You can develop Perl scripts with any text editor. Notepad, vi, MS Word 97, or even direct off the console. Of course, you can make things easy and use one of the many freeware or shareware programmers file editors.
- **Price.** Yes, 0 guilders, pounds, dmarks, dollars or whatever. And the peer to peer support is also free, and often far better than you'd ever get by paying some company to answer the phone and tell you to do what you just tried several times already, then look up the same reference books you already own.

What can I do with Perl ?

Just two popular examples :

Go surf. Notice how many websites have dynamic pages with .pl or similar as the filename extension? That's Perl. It is the most popular language for CGI programming for many reasons, most of which are mentioned above. In fact, there are a great many more dynamic pages written with perl that may not have a *.pl* extension. If you code in Active Server Pages, then you should try using ActiveState's PerlScript. Quite frankly, coding in PerlScript rather than VBScript or JScript is like driving a car as opposed to riding a bicycle. Perl powers a good deal of the Internet.

If you are a Unix sysadmin you'll know about sed, awk and shell scripts. Perl can do everything they can do and far more besides. Furthermore, Perl does it much more efficiently and portably. Don't take my word for it, ask around.

If you are an NT sysadmin, chances are you aren't used to programming. In which case, the advantages of Perl may not be clear. Do you need it? Is it worth it? A few examples of how I use Perl to ease NT sysadmin life:

- **User account creation**. If you have a text file with the user's names in it, that is all you need. Create usernames automatically, generate a unique password for each one

and create the account, plus create and share the home directory, and set the permissions.

- **Event log munging.** NT has great Event Logging. Not so great Event Reading. You can use Perl to create reports on the event logs from multiple NT servers.
- **Anything else** that you would have used a batch file for, or wished that you could automate somehow. Now you *can*.

- single word: atom, chain
- multi word: atomName, centralAtomName
- constants: CALPHA_ATOM_NAME or ATOM_NAME_CALPHA, ATOM_NAME_CBETA

Perl is not type-safe and this can cause confusion and errors. Use a limited prefix notation for such common basic types as array, hash, FileHandle.

- array refs ('a' prefix): aAtoms, aChains
- hash refs ('h' prefix): hNames2Places, hChains
- FileHandle objects ('fh' prefix): fhIn, fhOut, fhPdb
- or ("ist"=input stream, "ost"=output stream): ostPdb, istMsa

Functions

- single word: Trim()
- multi word: OpenFilesForReading()

- single word: Assert
- multi word: FileIoHelper

Classes

As for modules but with 'C' prefix: CStopwatch, CWindowPanel, Pdb::CResidue

Instance methods

- public method: plot(), getColour(), classifyHetGroups()
- private method: _plot(), _getColour(), _classifyHetGroups()
- accessor methods same as JavaBeans: getProperty(), setProperty(), isProperty()

Perl, perl or PeRl?

There is also a certain amount of confusion regarding the capitalization of Perl. Should it be written Perl or perl? Larry Wall now uses —Perl‖ to signify the language proper and —perl‖ to signify the implementation of the language.

Perl History

Version	Date	Version Details
Perl 0		Introduced Perl to Larry Wall's office associates
Perl 1	Jan 1988	Introduced Perl to the world
Perl 2	Jun 1988	Introduced Harry Spencer's regular expression package
Perl 3	Oct 1989	Introduced the ability to handle binary data
Perl 4	Mar 1991	Introduced the first —Camel‖ book (Programming Perl, by Larry Wall, Tom Christiansen, and Randal L Schwartz; O'Reilly & Associates). The book drove the name change, just so it could refer to Perl 4, instead of Perl 3.
Feb 1993	Feb 1993	The last stable release of Perl 4
Perl 5	Oct 1994	The first stable release of Perl 5, which introduced a number of new features and a complete rewrite.
Perl .005_02	Aug 1998	The next major stable release
Perl .005_03	Mar 1999	The last stable release before 5.6
Perl 5.6	Mar 2000	Introduced unified **fork** support, better threading, an updated Perl compiler, and the **our** keyword

Main Perl Features:

1. **Perl Is Free:**
   Perl's source code is open and free anybody can download the C source that constitutes a Perl interpreter. Furthermore, you can easily extend the core functionality of Perl both within the realms of the interpreted language and by modifying the Perl source code.
2. **Perl Is Simple to Learn, Concise, and Easy to Read:**
   It has a syntax similar to C and shell script, among others, but with a less restrictive format. Most programs are quicker to write in Perl because of its use of built-in functions and a huge standard and contributed library. Most programs are also quicker to execute than other languages because of Perl's internal architecture.
3. **Perl Is Fast**
   Compared to most scripting languages, this makes execution almost as fast as compiled C code. But, because the code is still interpreted, there is no compilation process, and applications can be written and edited much faster than with other languages, without any of the performance problems normally associated with an interpreted language.
4. **Perl Is Extensible**
   You can write Perl-based packages and modules that extend the functionality of the language. You can also call external C code directly from Perl to extend the functionality.
5. **Perl Has Flexible Data Types**
   You can create simple variables that contain text or numbers, and Perl will treat the variable data accordingly at the time it is used.
6. **Perl Is Object Oriented**
   Perl supports all of the object-oriented features—inheritance, polymorphism, and encapsulation.
7. **Perl Is Collaborative**
   There is a huge network of Perl programmers worldwide. Most programmers supply, and use, the modules and scripts available via CPAN, the Comprehensive Perl Archive Network

Compiler or Interpreter:

a. **Compiler:** A program that decodes instructions written in a higher order language and produces an assembly language program.
   A compiler that generates machine language for a different type of computer than the one the compiler is running in.

b. **Interpreter:** In computing, an interpreter is a computer program that reads the source code of another compute program and executes that program. *A program that translates and executes source language statements one line at a time.*

c. **Difference between Compiler and Interpreter**
   A compiler first takes in the entire program, checks for errors, compiles it and then executes it. Whereas, an interpreter does this line by line, so it takes one line, checks it for errors and then executes it.
   Example of Compiler – Java
   Example of Interpreter – PHP

d. **Perl is interpreter or compiler?**
   Neither, and both. Perl is a scripting language. There is a tool, called perl, intended to run programs written in the perl language.
   "Compiled" languages are ones like C and C++, where you have to take the source code, compile it into an executable file, and THEN run it.
   "Interpreted" languages, like Perl, PHP, and Ruby, are ones which do NOT require pre-compiling.
   They are generally compiled on-the-fly (which is what the perl command-line tool does) into *opcodes*, and then run. So, Perl is an interpreted language because a tool reads the source code and immediately runs it.

Perl is a compiler because it has to compile that source code before it can be run while it's being interpreted.

Popular "Myth conceptions"

1. **It's only for the Web**
   Probably the most famous of the myths is that Perl is a language used, designed, and created exclusively for developing web-based applications.
2. **It's Not Maintenance Friendly**
   Any good (or bad) programmer will tell you that anybody can write unmaintainable code in any language. Many companies and individuals write maintainable programs using Perl.
3. **It's Only for Hackers**
   Perl is used by a variety of companies, organizations, and individuals. Everybody from programming beginners through ―hackers‖ up to multinational corporations use Perl to solve their problems.
4. **It's a Scripting Language**
   In Perl, there is no difference between a script and program. Many large programs and projects have been written entirely in Perl.
5. **There's No Support**
   The Perl community is one of the largest on the Internet, and you should be able to find someone, somewhere, who can answer your questions or help you with your problems.
6. **All Perl Programs Are Free**
   Although you generally write and use Perl programs in their native source form, this does not mean that everything you write is free. Perl programs are your own intellectual property and can be bought, sold, and licensed just like any other program.
7. **There's No Development Environment**
   Perl programs are text based, you can use any source-code revision-control system. The most popular solution is CVS, or Concurrent Versioning System, which is now supported under Unix, MacOS and Windows.
8. **Perl Is a GNU Project**
   While the GNU project includes Perl in its distributions, there is no such thing as ―GNU Perl.‖ Perl is not produced or maintained by GNU and the Free Software Foundation. Perl is also made available on a much more open license than the GNU Public License.
9. **Perl Is Difficult to Learn**
   Because Perl is similar to a number of different languages, it is not only easy to learn but also easy to continue learning. Its structure and format is very similar to C, **awk**, shell script, and, to a greater or lesser extent, even BASIC.

Perl Overview:

Installing and using Perl

Perl was developed by Larry Wall. It started out as a scripting language to supplement *rn*, the USENET reader. It available on virtually every computer platform.

Perl is an interpreted language that is optimized for string manipulation, I/O, and system tasks. It has built in for most of the functions in section 2 of the UNIX manuals -- very popular with sys administrators. It incorporates syntax elements from the *Bourne shell*,

csh, awk, sed, grep, and C. It provides a quick and effective way to write interactive web applications

Writing a Perl Script

Perl scripts are just text files, so in order to actually "write" the script, all you need to do is create a text file using your favorite text editor. Once you've written the script, you tell Perl to execute the text file you created.

Under Unix, you would use

$ perl myscript.pl

and the same works under Windows:

C:\> perl myscript.pl

Under Mac OS, you need to drag and drop the file onto the *MacPerl* application. Perl scripts have a *.pl* extension, even under Mac OS and Unix.

Perl Under Unix

The easiest way to install Perl modules on Unix is to use the CPAN module. For example:

shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql

The DBD::mysql installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and ODBC on Windows. The default password is "no password.") If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use force install DBD::mysql to ignore the failed tests.

DBI requires the Data::Dumper module. It may be installed; if not, you should install it before installing DBI.

Perl Under Windows

1. Log on to the Web server computer as an administrator.
2. Download the ActivePerl installer from the following ActiveState Web site: http://www.activestate.com/ (http://www.activestate.com/)
3. Double-click the **ActivePerl** installer.
4. After the installer confirms the version of ActivePerl that it is going to be installed, click **Next**.
5. If you agree with the terms of the license agreement, click **I accept the terms in the license agreement**, and then click **Next**. Click **Cancel** if you do not accept the license agreement. If you do so, you cannot continue the installation.
6. To install the whole ActivePerl distribution package (this step is recommended), click **Next** to continue the installation. The software is installed in the default location (typically C:\Perl).
7. To customize the individual components or to change the installation folder, follow the instructions that appears on the screen.
8. When you are prompted to confirm the addition features that you want to configure during the installation, click any of the following settings, and then click **Next**:
    a. **Add Perl to the PATH environment variable**: Click this setting if you want to use Perl in a command prompt without requiring the full path to the Perl interpreter.
    b. **Create Perl file extension association**: Click this setting if you want to allow Perl scripts to be automatically run when you use a file that has the Perl file name extension (.pl) as a command name.
    c. **Create IIS script mapping for Perl**: Click this setting to configure IIS to identify Perl scripts as executable CGI programs according to their file name extension.
    d. **Create IIS script mapping for Perl ISAPI**: Click this setting to use Perl scripts as an ISAPI filter.
9. Click **Install** to start the installation process.
10. After the installation has completed, click **Finish**.

Perl Components:

Variables

Perl Variables with the techniques of handling them are an important part of the Perl language. As a language-type script, Perl was designed to handle huge amounts of data text. Working with variables is fairly straightforward given that it is not necessary to define and allocate them, so no sophisticated techniques for the release of memory occupied by them.

As general information, to note that the names of Perl variables contain alphabetic characters, numbers and the underscore (_) character and are case sensitive.

A specific language feature is that variables have a non-alphabetical prefix that fashion somewhat cryptic the language.

a. *scalar variables – starting with $*
b. *array variables – starting with @*
c. *hashes or associative arrays indicated by %*

The $, @ and % characters actually predefine the variable type in Perl. Perl language also offers some built-in predefined variables that facilitate and shorten the programming code.

Operators

The operators work with numbers and strings and manipulate data objects called operands. We found the operators in expressions which we need to evaluate.

Statements

The statements are one of the most important topics in the Perl language, actually for any programming language. We use statements in order to process or evaluate the expressions. Perl uses the values returned by statements to evaluate or process other statements.

A Perl statement ends with the semicolon character (;) which is used to tell interpreter that the statement was complete.

Subroutines (Functions)

Definition: *Subroutine* is a block of source code which does one or some tasks with specified purpose.

Advantages:

1. It reduces the Complexity in a program by reducing the code.
2. It also reduces the Time to run a program.In other way,It's directly proportional to Complexity.
3. It's easy to find-out the errors due to the blocks made as function definition outside the main function.

Modules:

A ***Perl module*** is a discrete component of software for the Perl programming language. Technically, it is a particular set of conventions for using Perl's package mechanism that has become universally adopted.

A module defines its source code to be in a *package* (much like a Java package), the Perl mechanism for defining namespaces, e.g. *CGI* or *Net::FTP* or *XML::Parser*; the file structure mirrors the namespace structure (e.g. the source code for *Net::FTP* is in *Net/FTP.pm*). A collection of modules, with accompanying documentation, build scripts, and usually a test suite, compose a ***distribution***.

Perl Parsing Rules

The Execution Process:

The execution process of *perl* contains the following steps

- It takes raw input,
- Parses each statement and converts it into a series of opcodes,
- Builds a suitable opcode tree,
- Executes the opcodes within a Perl "virtual machine."

It classifies only two stages

- The parsing stage and the
- Execution or run-time stage

The Perl parser thinks about all of the following when it looks at a source line:

- **Basic syntax** The core layout, line termination, and so on
- **Comments** If a comment is included, ignore it
- **Component identity** Individual terms (variables, functions and numerical and textual constants) are identified
- **Bare words** Character strings that are not identified as valid terms

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

- **Precedence** Once the individual items are identified, the parser processes the statements according to the precedence rules, which apply to all operators and terms
- **Context** What is the context of the statement, are we expecting a list or scalar, a number or a string, and so on. This actually happens during the evaluation of individual elements of a line, which is why we can nest functions such as sort, reverse, and keys into a single statement line
- **Logic Syntax** For logic operations, the parser must treat different values, whether constant- or variable-based, as true or false values

All of these present some fairly basic and fundamental rules about how Perl looks at an entire script.

The basic rules govern such things as line termination and the treatment of white space. These basic rules are

- Lines must start with a token that does not expect a left operand
- Lines must be terminated with a semicolon, except when it's the last line of a block, where the semicolon can be omitted.
- White space is only required between tokens that would otherwise be confusing, so spaces, tabs, newlines, and comments (which Perl treats as white space) are ignored. The line sub menu *{print"menu"}* works as it would if it were more neatly spaced.
- Lines may be split at any point, providing the split is logically between two tokens.

Component Identity

When Perl fails to identify an item as one of the predefined operators, it treats the character sequence as a "term." Terms are core parts of the Perl language and include variables, functions, and quotes. The term-recognition system uses these rules:

- Variables can start with a letter, number, or underscore, providing they follow a suitable variable character, such as $, @, or %.
- Variables that start with a letter or underscore can contain any further combination of letters, numbers, and underscore characters.
- Variables that start with a number can only consist of further numbers—be wary of using variable names starting with digits. The variables such as $0 through to $9 are used for group matches in regular expressions.
- Subroutines can only start with an underscore or letter, but can then contain any combination of letters, numbers, and underscore characters.
- Case is significant—$VAR, $Var, and $var are all different variables.
- Each of the three main variable types have their own name space—$var, @var, and %var are all separate variables.
- File handles should use all uppercase characters—this is only a convention, not a rule, but it is useful for identification purposes.

Operators and Precedence

a) Arithmetic Operators:

The following are the arithmetic operators in Perl.

Operator	Description
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Modulus operator
**	Exponentiation operator

The operators +, -, *, / take two operands and return the sum, difference, product and quotient respectively. Perl does a floating point division not an integral division. To get the integral quotient one has to use *int()* function. Say if you divide *"int(5/2)"* the result will be 2, to get the exact result use the code below.

b) Assignment Operators

Operator	Description
=	Normal Assignment
+=	Add and Assign
-=	Subtract and Assign
*=	Multiply and Assign
/=	Divide and Assign
%=	Modulus and Assign
**=	Exponent and Assign

Everyone knows how to use the assignment operator (=). There are other operators, when used with "=" gives a different result.

c) Increment/Decrement Operators

The following are the auto increment, decrement operators in Perl.

Operator	Description
++	Auto-increment operator
--	Auto-decrement operator

The usage of auto increment operators are same as in C Language. In prefix decrement / increment first thevalue is increased or decreased then the new value is returned eg: "++$a", "--$a".

The vice versa of the above, is post decrement/increment operators. First the old value is returned then incremented or decremented to give the result. eg: "$a++", "$a--"

d) Comparison Operators

Operator		Function
=	eq	Equal to Operator
!=	ne	Not Equal to Operator
<	lt	Less than Operator
>	gt	Greater than Operator
<=	le	Less than or Equal to Operator
>=	ge	Greater than or Equal to operator

Interface with CGI

What is CGI ?

The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script.

The CGI specs are currently maintained by the NCSA and NCSA defines CGI is as follows −

The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.

The current version is CGI/1.1 and CGI/1.2 is under progress.

Web Browsing

To understand the concept of CGI, lets see what happens when we click a hyper link to browse a particular web page or URL.

- Your browser contacts the HTTP web server and demand for the URL ie. filename.
- Web Server will parse the URL and will look for the filename in if it finds that file then sends back to the browser otherwise sends an error message indicating that you have requested a wrong file.
- Web browser takes response from web server and displays either the received file or error message.

However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your browser to display. This function is called the Common Gateway Interface or CGI and the programs are called CGI scripts. These CGI programs can be a PERL Script, Shell Script, C or C++ program etc.

CGI Architecture Diagram

Web Server Support & Configuration

Before you proceed with CGI Programming, make sure that your Web Server supports CGI and it is configured to handle CGI Programs. All the CGI Programs be executed by the HTTP server are kept in a pre-configured directory. This directory is called CGI Directory and by convention it is named as /cgi-bin. By convention PERL CGI files will have extention as **.cgi**.

First CGI Program

```
#!/usr/bin/perl

print "Content-type:text/html\r\n\r\n";
print '<html>';
print '<head>';
print '<title>Hello Word - First CGI Program</title>';
print '</head>';
print '<body>';
print '<h2>Hello Word! This is my first CGI program</h2>';
print '</body>';
print '</html>';

1;
```

Output

```
Hello Word! This is my first CGI program
```

HTTP Header

The line **Content-type:text/html\r\n\r\n** is part of HTTP header which is sent to the browser to understand the content. All the HTTP header will be in the following form

```
HTTP Field Name: Field Content
```

For Example

Content-type:text/html\r\n\r\n

There are few other important HTTP headers which you will use frequently in your CGI Programming.

S.No.	Header & Description
1	**Content-type: String** A MIME string defining the format of the file being returned. Example is Content-type:text/html
2	**Expires: Date String** The date the information becomes invalid. This should be used by the browser to decide when a page needs to be refreshed. A valid date string should be in the format

01 Jan 1998 12:00:00 GMT.

Location: URL String

3 The URL that should be returned instead of the URL requested. You can use this filed to redirect a request to any file.

Last-modified: String

4 The date of last modification of the resource.

Content-length: String

5 The length, in bytes, of the data being returned. The browser uses this value to report the estimated download time for a file.

Set-Cookie: String

6 Set the cookie passed through the *string*

CGI Environment Variables

All the CGI program will have access to the following environment variables. These variables play an important role while writing any CGI program.

S.No.	Variable Name & Description
1	**CONTENT_TYPE** The data type of the content. Used when the client is sending attached content to the server. For example file upload etc.
2	**CONTENT_LENGTH** The length of the query information. It's available only for POST requests.
3	**HTTP_COOKIE** Return the set cookies in the form of key & value pair.
4	**HTTP_USER_AGENT** The User-Agent request-header field contains information about the user agent originating the request. Its name of the web browser.
5	**PATH_INFO** The path for the CGI script.
6	**QUERY_STRING** The URL-encoded information that is sent with GET method request.
7	**REMOTE_ADDR** The IP address of the remote host making the request. This can be useful for logging or for authentication purpose.

REMOTE_HOST

8 The fully qualified name of the host making the request. If this information is not available then REMOTE_ADDR can be used to get IR address.

REQUEST_METHOD

9 The method used to make the request. The most common methods are GET and POST.

SCRIPT_FILENAME

10 The full path to the CGI script.

SCRIPT_NAME

11 The name of the CGI script.

SERVER_NAME

12 The server's hostname or IP Address.

SERVER_SOFTWARE

13 The name and version of the software the server is running.

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";
print "<font size=+1>Environment</font>\n";

foreach (sort keys %ENV) {
   print "<b>$_</b>: $ENV{$_}<br>\n";
}

1;
```

Output

```
Environment CONTEXT_DOCUMENT_ROOT:
CONTEXT_PREFIX:
DOCUMENT_ROOT:
GATEWAY_INTERFACE:
GEOIP_ADDR:
GEOIP_CONTINENT_CODE:
GEOIP_COUNTRY_CODE:
GEOIP_COUNTRY_NAME:
HTTP_ACCEPT:
HTTP_ACCEPT_ENCODING:
HTTP_ACCEPT_LANGUAGE:
HTTP_COOKIE:
HTTP_HOST:
HTTP_UPGRADE_INSECURE_REQUESTS:
HTTP_USER_AGENT:
HTTP_VIA:
HTTP_X_FORWARDED_FOR:
HTTP_X_FORWARDED_PROTO:
HTTP_X_HOST:
PATH:
QUERY_STRING:
REMOTE_ADDR:
```

```
REMOTE_PORT:
REQUEST_METHOD:
REQUEST_SCHEME:
REQUEST_URI:
SCRIPT_FILENAME:
SCRIPT_NAME:
SCRIPT_URI:
SCRIPT_URL:
SERVER_ADDR:
SERVER_ADMIN:
SERVER_NAME:
SERVER_PORT:
SERVER_PROTOCOL:
SERVER_SIGNATURE:
SERVER_SOFTWARE:
UNIQUE_ID:
```

How To Raise a "File Download" Dialog Box ?

Sometime it is desired that you want to give option where a use will click a link and it will pop up a "File Download" dialogue box to the user in stead of displaying actual content. This is very easy and will be achived through HTTP header.

This HTTP header will be different from the header mentioned in previous section.

For example,if you want make a **FileName** file downloadable from a given link then its syntax will be as follows.

```
#!/usr/bin/perl

# HTTP Header
print "Content-Type:application/octet-stream; name=\"FileName\"\r\n";
print "Content-Disposition: attachment; filename=\"FileName\"\r\n\n";

# Actual File Content will go hear.
open( FILE, "<FileName" );
while(read(FILE, $buffer, 100) ){
   print("$buffer");
}
```

GET and POST Methods

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your CGI Program. Most frequently browser uses two methods two pass this information to web server. These methods are GET Method and POST Method.

Passing Information using GET method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows −
http://www.test.com/cgi-bin/hello.cgi?key1=value1&key2=value2

The GET method is the defualt method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limtation: only 1024 characters can be in a request string.

This information is passed using QUERY_STRING header and will be accessible in your CGI Program through QUERY_STRING environment variable.

You can pass information by simply concatenating key and value pairs along with any URL or you can use HTML <FORM> tags to pass information using GET method.

Simple URL Example : Get Method

Here is a simple URL which will pass two values to hello_get.cgi program using GET method.

Below is hello_get.cgi script to handle input given by web browser.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "GET") {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}

$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Hello - Second CGI Program</title>";
print "</head>";
print "<body>";
print "<h2>Hello $first_name $last_name - Second CGI Program</h2>";
print "</body>";
print "</html>";

1;
```

Output

```
Hello ZARA ALI .....
```

Simple FORM Example: GET Method

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same CGI script hello_get.cgi to handle this input.

```
<FORM action = "/cgi-bin/hello_get.cgi" method = "GET">
   First Name: <input type = "text" name = "first_name">  <br>

   Last Name: <input type = "text" name = "last_name">
   <input type = "submit" value = "Submit">
</FORM>
```

Here is the actual output of the above form, You enter First and Last Name and then click submit button to see the result.

First Name:

Last Name:

Passing Information using POST method

A generally more reliable method of passing information to a CGI program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes into the CGI script in the form of the standard input.

Below is hello_post.cgi script to handle input given by web browser. This script will handle GET as well as POST method.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST"){
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
```

```
}

$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Hello - Second CGI Program</title>";
print "</head>";
print "<body>";
print "<h2>Hello $first_name $last_name - Second CGI Program</h2>";
print "</body>";
print "</html>";

1;
```

Let us take again same example as above, which passes two values using HTML FORM and submit button. We are going to use CGI script hello_post.cgi to handle this input.

```
<FORM action = "/cgi-bin/hello_post.cgi" method="POST">
   First Name: <input type="text" name="first_name">  <br>

   Last Name: <input type="text" name="last_name">

   <input type="submit" value="Submit">
</FORM>
```

Here is the actual output of the above form, You enter First and Last Name and then click submit button to see the result.

First Name:

Last Name:

Passing Checkbox Data to CGI Program

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code for a form with two checkboxes

```
<form action = "/cgi-bin/checkbox.cgi" method = "POST" target = "_blank">
   <input type = "checkbox" name = "maths" value = "on"> Maths
   <input type = "checkbox" name = "physics" value = "on"> Physics
   <input type = "submit" value = "Select Subject">
</form>
```

The result of this code is the following form

- Maths
- Physics

Below is checkbox.cgi script to handle input given by web browser for radio button.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST"){
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}

if( $FORM{maths} ){
   $maths_flag ="ON";
}else{
   $maths_flag ="OFF";
}

if( $FORM{physics} ){
   $physics_flag ="ON";
}else{
   $physics_flag ="OFF";
}

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Checkbox - Third CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> CheckBox Maths is : $maths_flag</h2>";
print "<h2> CheckBox Physics is : $physics_flag</h2>";
print "</body>";
print "</html>";

1;
```

Passing Radio Button Data to CGI Program

Radio Buttons are used when only one option is required to be selected.

Here is example HTML code for a form with two radio button −

```
<form action = "/cgi-bin/radiobutton.cgi" method = "POST" target =
"_blank">
   <input type = "radio" name = "subject" value = "maths"> Maths
   <input type = "radio" name = "subject" value = "physics"> Physics
   <input type = "submit" value = "Select Subject">
</form>
```

The result of this code is the following form −

Maths

Physics

Below is radiobutton.cgi script to handle input given by web browser for radio button.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$subject = $FORM{subject};
print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Radio - Fourth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Selected Subject is $subject</h2>";
print "</body>";
print "</html>";

1;
```

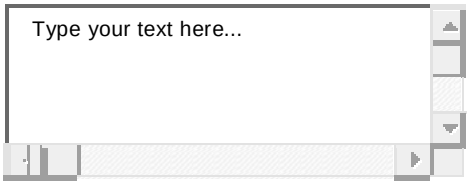
Passing Text Area Data to CGI Program

TEXTAREA element is used when multiline text has to be passed to the CGI Program.

Here is example HTML code for a form with a TEXTAREA box −

```
<form action = "/cgi-bin/textarea.cgi" method = "POST" target = "_blank">
   <textarea name = "textcontent" cols = 40 rows = 4>
      Type your text here...
   </textarea>
   <input type = "submit" value = "Submit">
</form>
```

The result of this code is the following form −



Below is textarea.cgi script to handle input given by web browser.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}

$text_content = $FORM{textcontent};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Text Area - Fifth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Entered Text Content is $text_content</h2>";
print "</body>";
print "</html>";

1;
```

Passing Drop Down Box Data to CGI Program

Drop Down Box is used when we have many options available but only one or two will be selected.

Here is example HTML code for a form with one drop down box

```
<form action = "/cgi-bin/dropdown.cgi" method = "POST" target = "_blank">
   <select name = "dropdown">
      <option value = "Maths" selected>Maths</option>
      <option value = "Physics">Physics</option>
   </select>
   <input type = "submit" value = "Submit">
</form>
```

The result of this code is the following form −

Below is dropdown.cgi script to handle input given by web browser.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}

$subject = $FORM{dropdown};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Dropdown Box - Sixth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Selected Subject is $subject</h2>";
print "</body>";
print "</html>";

1;
```

Using Cookies in CGI

HTTP protocol is a stateless protocol. But for a commercial website it is required to maintain session information among different pages. For example one user registration ends after completing many pages. But how to maintain user's session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

How It Works

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the cookie is available for retrieval. Once retrieved, your server knows/remembers what was stored.

Cookies are a plain text data record of 5 variable-length fields −

- **Expires** − The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain** − The domain name of your site.
- **Path** − The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure** − If this field contains the word "secure" then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value** − Cookies are set and retrviewed in the form of key and value pairs.

Setting up Cookies

This is very easy to send cookies to browser. These cookies will be sent along with HTTP Header. Assuming you want to set UserID and Password as cookies. So it will be done as follows −

```
#!/usr/bin/perl

print "Set-Cookie:UserID=XYZ;\n";
print "Set-Cookie:Password=XYZ123;\n";
print "Set-Cookie:Expires=Tuesday, 31-Dec-2007 23:12:40 GMT";\n";
print "Set-Cookie:Domain=www.tutorialspoint.com;\n";
print "Set-Cookie:Path=/perl;\n";
print "Content-type:text/html\r\n\r\n";
.....Rest of the HTML Content....
```

From this example you must have understood how to set cookies. We use **Set-Cookie** HTTP header to set cookies.

Here it is optional to set cookies attributes like Expires, Domain, and Path. It is notable that cookies are set before sending magic line **"Content-type:text/html\r\n\r\n**.

Retrieving Cookies

This is very easy to retrieve all the set cookies. Cookies are stored in CGI environment variable HTTP_COOKIE and they will have following form.

```
key1=value1;key2=value2;key3=value3....
```

Here is an example of how to retrieving cookies.

```
#!/usr/bin/perl
$rcvd_cookies = $ENV{'HTTP_COOKIE'};
@cookies = split /;/, $rcvd_cookies;

foreach $cookie ( @cookies ){
   ($key, $val) = split(/=/, $cookie); # splits on the first =.
   $key =~ s/^\s+//;
   $val =~ s/^\s+//;
   $key =~ s/\s+$//;
   $val =~ s/\s+$//;

   if( $key eq "UserID" ){
      $user_id = $val;
   }elsif($key eq "Password"){
      $password = $val;
   }
}

print "User ID  = $user_id\n";
print "Password = $password\n";

This will produce following result
User ID = XYZ
Password = XYZ123
```

A form to mail Program:

Create the web form

First we need to create a simple HTML form, to start with we'll keep the form simple by just asking for the users email address and comments. Here is our HTML form:

```
<html>
<head>
<title>Simple Feedback Form</title>
<style>label{display:block;}</style>
</head>
<body>

<form action="/cgi-bin/feedback_form.cgi" method="post">

<label>Email Address</label>
<input type="text" name="email_address" size="40">

<label>Your Feedback</label>
<textarea name="feedback" cols="50" rows="10"></textarea>
```

```
<input type="submit" name="send" value="Submit">

</form>

</body>
</html>
```

This form will send two parameters to our cgi script, *email_address* and *feedback*. Save this file as *feedback_form.html* and upload it to the **web** folder on your hosting.

Create the form script

We're going to use the **CGI.pm Perl module** to help make writing our cgi script easier. At the top of the script we start with the location of the perl interpretor, then we tell Perl we want to use the CGI.pm module and create a new cgi object:

```
#!/usr/bin/perl

use CGI;

my $cgi = new CGI;
```

The CGI.pm module is object-orientated, this means all of the CGI.pm functions and data are accessed through an instance of CGI.pm, in our script this instance is called *$cgi*.

Lets use our CGI object to retrieve the information from the form the user filled in. To access the form parameters we can use the CGI objects *param* function:

```
my $email_address = $cgi->param('email_address');
my $feedback = $cgi->param('feedback');
```

We store the form data in two local Perl variables, *$email_address* and *$feedback*.

Filtering user submitted data

Whenever you write a cgi script that receives data from an unknown source you should always filter the data to make sure it doesn't contain anything harmful. For example, if we don't filter the data in our form it would be quite easy for a Hacker to use our cgi script to send out spam to thousands of people. The golden rule is never trust any data you haven't created or don't control.

To filter our user data we're going to create two filter functions:

```
sub filter_email_header
{
  my $form_field = shift;
  $form_field = filter_form_data($form_field);
  $form_field  =~ s/[\0\n\r\|\!\/\<\>\^\$\%\*\&]+/ /g;

  return $form_field;
}

sub filter_form_data
```

```
{
  my $form_field = shift;
  $form_field  =~ s/From://gi;
  $form_field  =~ s/To://gi;
  $form_field  =~ s/BCC://gi;
  $form_field  =~ s/CC://gi;
  $form_field  =~ s/Subject://gi;
  $form_field  =~ s/Content-Type://gi;

  return $form_field;
}
```

The first filter function removes special characters which could be used to trick our script into sending spam and is applied to the *$email_address* data.The second filter function removes common email headers from the data the user submitted and can be applied to both *$email_address* and *$feedback*. We'll place the two functions at the bottom of our script.

Now we'll call the two filter functions to clean up our user submitted data:

```
$email_address  = filter_email_header($email_address);
$feedback = filter_form_data($feedback);
```

Emailing the feedback

Once we have the filtered data we need to email it back to you. Our web hosting servers run a local mail server (sendmail) that your cgi script can use to send email. To send the email our cgi script opens a communication channel to the sendmail program using the pipe (|) symbol. It then prints all the information necessary to send an email across that channel:

```
open ( MAIL, "| /usr/lib/sendmail -t" );
print MAIL "From: $email_address\n";
print MAIL "To: you\@domain.com\n";
print MAIL "Subject: Feedback Form Submission\n\n";
print MAIL "$feedback\n";
print MAIL "\n.\n";
close ( MAIL );
```

Make sure you set your email address on line 3, you'll need to escape the @ symbol by putting a backslash (\) before it because Perl uses the @ symbol to denote a special type of variable. The two newline characters (\n\n) at the end of line 4 are used to mark the end of the email headers ready for the content. The \n.\n on line 6 prints a dot (.) on its own line to tell sendmail that we've finished printing the message.

Thank the user for their feedback

Finally, when a user submits your form, let's show a page thanking them for their feedback:

```
print $cgi->header(-type => 'text/html');

print <<HTML_PAGE;
 <html>
<head>
<title>Thank You</title>
</head>
<body>
```

```
<h1>Thank You</h1>
<p>Thank you for your feedback.</p>
</body>
</html>
HTML_PAGE
```

The first thing we do is print back the HTTP header, using the CGI header function, to let the web browser know what type of content to expect. Then we print out the HTML page.

The final script

This example script shows a very basic way to get form contents emailed to you, it doesn't however have the refinements of a professional script, e.g. input validation. Below is the finished script. We've added some comments (lines beginning with #) to help make it clearer.

```
#!/usr/bin/perl

use CGI;

# Create a CGI.pm object
my $cgi = new CGI;

# Get the form data
my $email_address = $cgi->param('email_address');
my $feedback = $cgi->param('feedback');

# Filter the form data
$email_address  = filter_email_header($email_address);
$feedback = filter_form_data($feedback);

# Email the form data
open ( MAIL, "| /usr/lib/sendmail -t" );
print MAIL "From: $email_address\n";
print MAIL "To: you\@domain.com\n";
print MAIL "Subject: Feedback Form Submission\n\n";
print MAIL "$feedback\n";
print MAIL "\n.\n";
close ( MAIL );

# Print the HTTP header
print $cgi->header(-type => 'text/html');

# Print the HTML thank you page
print <<HTML_PAGE;
 <html>
<head>
<title>Thank You</title>
</head>
<body>
<h1>Thank You</h1>
<p>Thank you for your feedback.</p>
</body>
</html>
HTML_PAGE

# Functions to filter the form data
```

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

```
sub filter_email_header
{
  my $form_field = shift;
  $form_field = filter_form_data($form_field);
  $form_field  =~ s/[\0\n\r\|\!\/\<\>\^\$\%\*\&]+/ /g;

  return $form_field ;
}

sub filter_form_data
{
  my $form_field = shift;
  $form_field  =~ s/From://gi;
  $form_field  =~ s/To://gi;
  $form_field  =~ s/BCC://gi;
  $form_field  =~ s/CC://gi;
  $form_field  =~ s/Subject://gi;
  $form_field  =~ s/Content-Type://gi;

  return $form_field ;
}
```

Simple Page Search:

```
#!/usr/bin/env perl

use strict;
use warnings;

use HTML::Parser;
use LWP::UserAgent;

my $ua = LWP::UserAgent->new;
my $response = $ua->get('http://search.cpan.org/');
if ( !$response->is_success ) {
    print "No matches\n";
    exit 1;
}

my $parser = HTML::Parser->new( 'text_h' => [ \&text_handler, 'dtext' ] );
$parser->parse( $response->decoded_content );

sub text_handler {
    chomp( my $text = shift );

    if ( $text =~ /language/i ) {
        print "Matched: $text\n";
    }
}
```

UNIT - VI

RUBY

Syllabus:-

- Introduction to Ruby
- variables
- Data types
- Simple I/O
- control statements
- Arrays
- Hashes
- Methods
- classes
- Iterators
- Pattern Matching
- Practical Web applications.

** Introduction to RUBY :-

- Ruby is a dynamic programming Language with complex but it have expressive grammar and core class libraries with a rich and powerful API.
- Ruby draws inspiration from Lisp, small talk, perl but uses a grammar i.e easy for c & Java programming.
- Ruby is a pure objectoriented language but it is also suitable for procedural and functional programming style
- Ruby was developed by Yukihiro Matsumoto in the year of 1995.

Features of Ruby :-

- Ruby is open-source and it is freely available on web
- Ruby is general purpose interpreted programming language
- Ruby is pure object oriented programming Language.
- Ruby is server-side scripting language similar to Phython and perl.
- Ruby is used to write CGI scripts
- Ruby can be embedded into HTML, it has similar syntax to that of C++ and perl.
- Ruby can be used to develop Internet & Intranet Applications
- Ruby can be installed on windows or POSIX Environment
- Ruby supports many GUI tools like Tcl/tk, GTK and OpenGL.
- Ruby can easily connected to DB2, MySQL, Oracle and Sybase databases
- Ruby has rich set of built in functions, which can be used directly into ruby scripts.

Syntax's:-

- The extension of ruby file is .rb or .rbw.

Prg:-

```
#!/usr/bin/ruby -w

puts "Hello Ruby";
```

- /usr/bin is the directory, ruby interpreter is available.
- $ ruby text.rb
  * Running of ruby program involves ruby followed by filename with extension.
- Result :- Hello Ruby

** Language properties:-

→ White spaces in Ruby:

- spaces and tab spaces are generally ignored in ruby code, except when they are appear in the string.

  Ex:-
  a+b is interpreted as a+b (local variables)
  a +b is interpreted as a(+b) (method call).

→ Line ending in Ruby program:

- Ruby interprets semicolons and newline characters as the ending of a statement.
- If operators encountered such as +, -, &, \ ... at end of a line, they indicates that continuation of a statement.

→ Ruby identifiers:

- Identifiers are the names of variables, constants and methods
- Ruby identifiers are case sensitive

  Ex:- RAM and ram (Different)
- Ruby identifier may contain alphanumeric characters and the underscore (_) character.

→ Reserved Words :

- the reserved words may not be used as constant or variable names.
- Total 39 reserved words in Ruby.

→ Here Document in Ruby

- Here document refers to build strings from multiple lines
- Following << we can specify a string or an identifier to terminate the string literals and all lines following current line upto the terminator are the value of string.
- there is no space between << and terminator, it should be in double quotes in general.

Ex:-

```
#! /usr/bin/ruby -w
print <<EOF
    echo "Welcome";
    echo "Ruby";
EOF
Print <<"END"
    echo "Welcome";
    echo "Programming";
END
```

Result

```
Welcome Ruby
Welcome programming.
```

→ Ruby BEGIN & END Statement

- BEGIN declares the codes to be called before run the program.

Syn:-
```
BEGIN {
    = } code
}
```

- END declares the code to be called at end of the program.

Syn:-
```
END {
    = } Code
}
```

Ex:-

```
PUts "Main program"
END { PUts "terminating program" }
BEGIN { PUts "Starting program" }
```

Result :-

```
Starting program
Main program
terminating program
```

→ Ruby comments :-

- A comment hides the line or part of a line or several lines from ruby interpreter
- Use hash (#) at beginning of a line
- A block of comments cancels several lines from interpreter with =begin / =end.

Syn :- =begin
=
=
=end.

Interactive Ruby (IRb) :-

- It provides shell for experimentation with in the IRb shell, we can see immediatly expression results line by line.
- This tool comes along with Ruby installation so we have nothing to do extra for working IRb
- Just type ~~IRb~~ irb at your command prompt and the session will start
- Every statement have its return value for every method.
- In ruby null values are represented as NIL.

Ex :- Simple I/O :-

$ irb ↵

=> a=10
10 → print (~~return value~~)

=> "your ~~number~~ is 10"
"your number is 10" → print

=> "your number is #{a}"
"your number is 10" → print

=> "your number is #{a+2}"
"your number is 12" → print

=> puts "hi"
hi → print
NIL → return value.

=> puts "the number is #{a}"

the number is 10

NIL

=> puts "abcde" + "fghij"

abcdefghij

NIL

=> gets

hai

"hai\n" [return value] \n (enter char)

=> b = gets

hello

"hello\n"

=> b

"hello\n"

=> b.chomp

↳ remove \n, At printing time

hello

=> b

hello\n

=> b.chomp!

↳ remove \n, at printing time and from the variable

hello

=> b

hello

=> puts "Enter your name"

Enter your name

NIL

=> name = gets.chomp

Rama

"Rama"

=> puts "Hello #{name}"

Hello Rama

NIL

** Classes and Objects:-

- Ruby is perfect object oriented programming language. The features of object-oriented programming language includes
  - * Data Abstraction
  - * Data Encapsulation
  - * Inheritance
  - * polymorphism
- An object oriented program involves classes & objects.
- A class is a blueprint, from which individual objects are created.
- A class is collection of data members and member function (or) characteristics and functionality.

Syntax:- class Customer

end

- A class in ruby always start with class followed by name of class. The class name always Initial capitals.
- terminating class by the keyword end.
- The data members and member functions we have to write between class definition and end keywords.

=> Variables in Ruby class:-

- * Local variables
- * Instance variables
- * class variables
- * Global variables

Local variables:-

- These are the variables that are defined in method. Local variables are not available outside of method.
- Local variables begins with lowercase or _

Instance variables :-

- Instance variables are available across the methods for any particular instance or object. i.e instance variables changes from object to object.
- Instance variables preceded by @ followed by variable name

Class variables :-

- class variables are available across different objects. A class variable belongs to the class and is a characteristic of a class.
- class variables preceded by @@ followed by variable name.

Global variables :-

- Global variables are available across different classes
- Global variables preceded by $ followed by variable name.

⇒ creating objects using new method :-

- Objects are instances of the class. We can create objects in ruby using new method.

```
cust1 = Customer.new
cust2 = Customer.new
```

- the method new is a unique type of method which is predefined in the ruby library
- objectname followed by = sign after which the classname follow dot operator and keyword new will follow.

⇒ Custom methods to create Ruby objects :-

- We can pass parameters to new method and those parameters are used to initialize class variables.
- When we plan to declare new method with parameters, we need to declare the method initialize at the time of class creation.

- The initialize method is special kind of method which will be executed when new method of a class is called with parameters.

Ex:-
```
class Customer
    @@no_of_cust = 0
    def initialize(id, name, addr)
        @cust_id = id
        @cust_name = name
        @cust_addr = addr
    end
end
cust1 = Customer.new(1, "Rama", "Ongole");
```

- For defining ruby method we are using def and end.

⇒ Member functions in Ruby class

- In ruby functions are called methods. Each method in class starts with keyword def followed by method name.
- The method name is always lowercase.
- Ending the method by the keyword end.

Ex:-
```
class Sample
    def hello
        puts "Hello ruby"
    end
end
obj = Sample.new
obj.hello   # calling method
```

Result
Hello Ruby

** preceded the variable name by # in puts statement. The text and instance variable along with hash symbol should be enclosed in double quotes.

Ex:-
```
puts "Printing variable value #@cust_id"
```

Variables & constants :-

- Variables are the memory locations, which hold any data to be used by a program.
- There are 5 Types of variables in ruby

1. Local variables
2. Global variables
3. Instance variable
4. class variables
5. Constants

1. Local variables :-

- Local variables are begins with lowercase or _(underscore).
- Scope of local variable is, ranges from class, def, method or do the corresponding end (or) from block of opening brace to its closed brace
- when an uninitialized variable passed through the arguments of a method then it will treated as no arguments.
- Assignment to uninitialized local variables also serves as variable declaration.
- To print the variable value with in double quotes, the variable is enclosed in #{var}.

2. Global variables :-

- Global variables are begin with $.
- Uninitialized Global variables have the value NIL.
- The scope of global variable is entire the program.
- To print the varible value with in double quotes, the variable is preceded by #

EX:- print "#$var";

Ex:-

```
$global_var = 10

class Class1
    def print_global
        puts "Global variable value is #$global_var"
    end
end
class Class2
    def print_global
        puts "Global variable in Class2 is #$global_var"
    end
end
ob1 = Class1.new
ob1.print_global
ob2 = Class2.new
ob2.print_global
```

Result

- Global variable value in class1 is 10
- Global variable value in class 2 is 10.

3. Instance variables :-

- Instance variables begins with @.
- Uninitialized variable having the value NIL.
- The scope of the variable is across the methods.
- To print the variable value with in double quotes, the variable is preceded by #

Ex:- puts "#@ins_var"

```
class Customer                                  -> local variables
    def initialize(id, name, addr)
        @cust_id = id.
        @cust_name = name
        @cust_addr = addr
    end.
    def display()
        puts "Custom id #@cust_id";
        puts "Customer name #@cust_name"
        puts "Customer address #@cust_addr"
    end
end
```

(instance variables)

```
cust1 = Customer.new("1", "Rama", "Guntur, AP");
cust2 = Customer.new("2", "Lakshmi", "SAP, AP");
cust1.display()
cust2.display()
```

Result:-

```
Customer id 1
Customer name Rama
Customer address Guntur, AP
Customer id 2
Customer name Lakshmi
Customer address SAP, AP
```

4. Class variables:-

- class variable starts with @@ and it must be initialized before used by a method.
- Uninitialized class variables produce error.
- The scope of class variables is across methods with in the class
- Overriding class variables produce error with -W option.
- To print the variable value with in double quotes, it must be preceded by #

```
Puts "#@@class_var"
```

Ex:-

```
class Customer
  @@no_of_cust = 0
  def initialize(id, name, addr)
    @cust_id = id
    @cust_name = name
    @cust_addr = addr
  end
  def total_cust()
    @@no_of_cust += 1
  puts "Total customers #@@no_of_cust"
  end
```

```
    def display()
        puts "customer id #@cust_id"
        puts "customer name #@cust_name"
        puts "customer address #@cust_addr"
    end
end
cust1 = Customer.new("1","Rama","Guntur")
cust2 = Customer.new("2","Lakshmi","SAP")
cust1.total_cust();
cust2.total_cust();
```

Result:-

Total no. of customers 1

Total no. of customers 2

5. Ruby Constants:-

- Constants begin with an Upper case Letters
- The constants defined with in a class can be accessed from with in class otherwise if we defined outside the class those can be accessed Globally.
- constants may not be defined in Methods.
- Referencing an uninitialized constant produce an error.
- To print the value of a variable with in double quotes, them "#{var}"

Ex:-

```
class Sample
    var1 = 100
    var2 = 200
    def show()
        puts "value of var1 is #{var1}"
        puts "value of var2 is #{var2}"
    end
end
ob = Sample.new
ob.show()
```

Result

value of var1 is 100

value of var2 is 200

Ruby pseudo variables :-

- These are special variables that have the appearance of local variables but behave like constants.
- We can't assign any value to these variables

* self : the receiver object of current method.
* true : value representing true
* false : value representing false
* nil : value representing undefined
* __FILE__ : the name of current source file.
* __LINE__ : the current line number in source file

** Datatypes :-

1. Ruby basic Literals
2. Ruby Arrays
3. Ruby Hashes
4. Ruby Ranges

1. Ruby basic literals :-

- The rules for ruby literals are very simple
- There are two types of literals
  - * Numbers → Integer Literals, Floating Literals
  - * Text → String Literals, character Literals

Integer Literals :-

- Integers with in the range are stored in class of FIXNUM.
- Integers outside the range are stored in class of BIGNUM.
- The range of integers -2^{30} to $2^{30}-1$ or -2^{62} to $2^{62}-1$
- The integers uses optional leading sign.
- Optional base indicator

number starts with	
0 (zero)	Octal (0-7)
0x	Hexa (0-9, A-F)
0b	Binary (0,1)

- Underscore characters are ignored in digit string
- We can also get the integer value corresponding to an ASCII character or escape sequence preceding it with question mark.

Ex:-

```
123          # fixed num decimal
1_234        # fixed num decimal with underscore
-500         # negative fixednum
0377         # Octal number
0xff         # Hexa decimal number
0b0101       # binary number
?a           # ASCII value for character 'a'
?\n          # ASCII value for newline (0x0a)
12345678901234567890 # Big num
```

Floating Numbers:-

- Floating point numbers are stored in objects of class Float.
- the integer number with decimals

Ex:-

```
123.4        # floating point value
1.0e6        # scientific notation
4E20         # dot not required
4e+20        # sign before exponential
```

Character literals:-

- Single characters can be included literaly in ruby by preceding the character ?.

Ex:- ?a
?\n

String Literals

* Single quote strings
* Double quote strings

- Ruby strings are simply sequence of 8-bit and they are stored in objects of class String.
- Double quoted strings allow substitution and backslash notation
- Single-quoted strings doesn't allow substitution and backslash notation only \\ and \'.
- This substitution is called String interpolation.
- we can substitute the value of any ruby expression into a string using sequence #{expr}

EX:- puts "multiplication is #{5*6*7}"

output : multiplication is 210

EX:- puts 'escape using "\\"';
puts 'that\'s right';

output : escape using "\"
that's right.

⇒ Backslash Notations:-

\n - newline
\r - carriage return
\b - backspace
\a - Bell
\e - escape
\s - space
\nnn - octal notation (n from 0-7)
\xnn - Hexadecimal (0-9, A-F)

⇒ 2. Ruby Arrays:-

- Literals of ruby array are created by placing a comma seperated series of object references between square brackets.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Ex:- array = ["Rama", "Lakshmi", "Venky"]

```
array.each do |i|
    puts i
end
```

Result
Rama
Lakshmi
Venky.

⇒ 3. Ruby Hashes:-

- Literal ruby hash is created by placing list of key/value pairs between braces with either comma or a sequence (=>) between key and value.

Ex:- hsh = {"Rama" => 50, "Lakshmi" => 70, "Venky" => 60}

Result
Rama is 50
Lakshmi is 70
Venky is 60.

```
hsh.each do |key, value|
    print key, " is " value, "\n"
end
```

⇒ 4. Ruby Ranges:-

- Range represents an interval set of values with start and an end.
- It may constructed using s..e (include)
  s...e (exclude)

1..5 => 1,2,3,4,5
1...5 => 1,2,3,4

Ex:- (10..15).each do |i|
    print i
end

Result
10, 11, 12, 13, 14, 15

** Operators:-

- It represents an operation to be performed on one or more operands
- Ruby supports rich set of operators as you had expect from modern language.
- Most of the operators are method calls.

⇒ Arithmetic operators :-

let a=10, b=20

+ - a+b ⇒ 30

- - a-b ⇒ -10

* - a*b ⇒ 200

/ - b/a ⇒ 2

% - b%a ⇒ 0

** - a**b ⇒ a^b ⇒ 10^{20}

⇒ Comparison operators :-

== - checks equal or not

!= - checks not equal or not

> - left operand is greater than right operand or not

< - left operand is less than right operand or not

>= - left operand is greater than or equal to right operand

<= - left operand is less than or equal to right operand

<=> - Combined comparison operator

a=b ⇒ 0, a<b ⇒ -1, a>b ⇒ 1

=== - Used to test equality with in clause

• eql? - receiver and argument have same type and equal value.

1==1.0 (True) 1.eql?(1.0) - false

• equal? - true if receiver object have same object id.

Ex:- bobj = aobj ;

aobj === bobj ⇒ True

a.equals(bobj) - false

b.equals(aobj) - true

⇒ Assignment operators :-

=, +=, *=, -=, /=, **=, %=

⇒ parallel assignment :-

- multiple variables initialized with single line ruby code.

```
a,b,c = 10,20,30;
a,b = b,c    => swap
```

⇒ Bitwise operator :-

- perform bit operations based on operations

```
a=60 = 0011 1100
b=13 = 0000 1101

a&b  = 12 = 0000 1100
a|b  = 61 = 0011 1101
a^b  = 49 = 0011 0001
~a   = 61 = 1100 0011

~~a<<2 = 0000~~
a<<2 = 0011 1100 << 2
     = 11110000 = 240
a>>2 = 0000 1111 = 15
```

⇒ logical operators :-

Let a=10, b=20

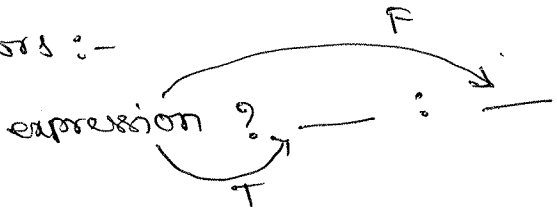
(&&) and (a and b) => false

(||) or (a or b) => true

(!) not !(a && b) => true

⇒ Ternary operators :-

?:



⇒ Range operators

.. - 1..10 - prints 1 to 10

... - 1...10 - prints 1 to 9

**Conditional statements :-

- Executes the block of code based on condition or expression.
- We are having following conditional statements in ruby
  * if ... else statements
  * if modifier
  * unless statement
  * unless modifier
  * case

⇒ if ... else statements :

- If the condition is true it will executes if block code otherwise it will execute else block code

Syntax :-

```
if condition [then]
    code ...
[elsif condition [then]
    code ... ]...
[else
    code ... ]
end
```

Example :-

```
x=1
if x>2
  puts "x is greaterthan 2"
elsif x<=2 and x!=0
  put "x is 1"
else
  puts "we cant guess"
end
```

Result :- x is 1.

- if expressions are used to conditional execution
- elsif is not else if .
- if condition will false then executes else block.

⇒ if Modifier :

- It will executes the code if condition is true

Syntax :-

```
Code ... if condition
```

Ex :-

```
$debug=1
print "debug" if $debug
```

⇒ Unless statement :-

- Executes the code if the condition is false otherwise executes else block code
- It is simply reverse of if ... else statement

Syntax :-

```
unless condition [then]
    code...
[else
    code... ]
end
```

Ex:-
```
x=1
unless x>2   (false)
  puts "x is less than 2"
else
  puts "x is greater than 2"
end
```
Result :- x is less than 2

⇒ Unless modifier :-

- It will executes the code if the condition will be false.

Syntax :-

```
code... unless condition
```

Ex:-
```
$var = 1   $var = false
print "1... value" if $var
print "2... value" unless $var
```
Result :-
1... value ✓
2... value

⇒ case statement :-

- compares the expression specified by case and execute the code when it matches

Syntax :-

```
case expression
[when expression(,expression...)[then]
    code]...
else
    code... ]
end
```

Ex:-
```
$age = 5
case $age
when 0..2
  puts "baby"
when 3..6
  puts "little child"
when 7.. 12
  puts "child"
when 13.. 18
  puts "youth"
else
  puts "adult"
end
```
Result :-
Little child.

** Control Statements :-

- Executes same block of code with specified no. of times
- the following control statements ruby can supports

  * While statement
  * While modifier
  * Until statement
  * Until Modifier
  * For statement
  * Break statement
  * Next statement
  * Redo statement
  * Retry statement

While statement :-

- Executes the while block of code if the condition is true

Syntax :-

```
while condition [do]
    Code...
end
```

Ex:-

```
$i=0
$num=5
while $i<$num do
  puts "i value is #$i"
  $i = $i+1
end
```

Result

```
i value is 0
i value is 1
i value is 2
i value is 3
i value is 4
```

While Modifier :-

- It is also similar to while statement but we are writting condition at end of block but no change logically.

Syntax :-

```
code while condition
    (or)
begin
    code
end while condition
```

Ex:-

```
$i=0
$num=5
begin
  puts"i value is #$i"
  $i+=1
end while $i<$num
```

Result

```
i value is 0
i value is 1
i value is 2
i value is 3
i value is 4
```

Until Statement :-

- It will executes the statements when the condition false.
- It is reverse of while statement

Syntax :-

```
until condition [do]
    code
end
```

Ex :-

```
$i = 0
$num = 5
until $i > $num do
    puts "value of i is #$i"
    $i += 1
end
```

→ False execute stmts

Result

```
value of i is 0
value of i is 1
value of i is 2
value of i is 3
value of i is 4
```

Until Modifier :-

- It is similar functionality of until statement but we are writing the condition at end of statements block.

Syntax :-

```
code until condition
    (or)
begin
    code
end until condition
```

Ex :-

```
$i = 0
$num = 5
begin
  puts "value of i is #$i"
  $i += 1
end until $i > $num
```

Result

```
value of i is 0
value of i is 1
value of i is 2
value of i is 3
value of i is 4
```

For Statement :-

- Executes code once for each element in expression.

Syntax :-

```
for variable[,variable...] in expression [do]
    code
end
```

Ex :-

```
for i in 0..5
    puts "value of i. is #{i}"
end
```

Result :-

```
value of i is 0
value of i is 1
value of i is 2
value of i is 3
value of i is 4
```

- We are also having one more way.

```
(expression).each do |variable[,variable ...]|
    code
end
```

Ex:-
```
(0..5).each do |i|
    puts "value of i is #{i}"
end
```

Break statement :-

- It will terminates most inner loop
- Terminates a method with an associated block if it is called with in the block

Ex:-
```
for i in 0..5
    if i>2 then
        break
    end
    puts "value is #{i}"
end
```

Result
value is 0
value is 1
value is 2

Next statement :-

- It will Jumps to the next iteration of most inner loop. terminates execution of a block if called with in block.

Ex:-
```
for i in 0..5
    if i<2 then
        next
    end
    puts "value is #{i}"
end
```

Result
value is 2
value is 3
value is 4
value is 5

Redo statement :-

- Restart the ~~most inner~~ iteration of the most inner loop, with out checking loop condition.

Ex:-
```
for i in 0..5
    if i<2 then
        puts "value is #{i}"
        redo
    end
end
```

Result
value is 0
value is 0
.
.
.

Retry Statement :-

- If retry appears in iterator then the block of code is restarted.
- It means if the condition ~~false~~ true it will start it again from starting

Ex:-
```
for i in 0..5
    retry if i > 2
    puts "value is #{i}"
end
```

Result
```
value is 0
value is 1
value is 2
value is 0
value is 1
value is 2
.
.
.
```

** Methods :-

- Methods are very similar to functions in any other programming language.
- Methods are used to bundle one or more repeatable statements into single unit
- Method name should start with lowercase letter.
- If we start with uppercase then ruby will treats that constants.
- Methods should be defined before calling otherwise rise the exception undefined method invoking
- Methods are starts with def.

Syntax :-
```
def method_name [ ( args [=default] ... [, *arg [, &expr]] ) ]
    expression
    statements
end
```

⇒ Simple Method : When we call simple method we can write only method name

⇒ Accepts parameters : when we call method with parameters, we can write method name along with parameters

```
def method_name
    expr.....
end
```

```
def meth_name(var1, var2)
    expr....
end
```

method_name 20,30

⇒ Set default values:

- We need to assign the values for the formal arguments in method.

```
def method_name( var1 = v1 , var2 = v2 )
    expr ....
end
```

Ex:-

```
def test (a1 = "Ruby" , a2 = "perl")
    puts "#{a1}"
    puts "#{a2}"
end
test "C", "C++"
test
```

Result

C, C++

Ruby, perl

⇒ Return value from Method

- Every method in ruby returns a value by default. this returned value will be the value of last statement.

Ex:-

```
def test
    i = 100
    j = 10
    k = 0
end
```

** It returns last declared variable k.

⇒ Return statement:

- Return statement in ruby used to return one or more values from method.

Syntax:- return [expr [, expr ....]]

- If more than two expressions are return then the values stored in array variable
- If no expression return then it will return NIL

Ex:-

```
def test
    i = 100
    j = 200
    k = 300
    return i, j, k
end
val = test
puts val
```

acts as array variable

Result

100
200
300.

=> Variable no. of parameters :-

- Suppose we declare a method that takes two parameters when ever we call this method, we need to pass two parameters along with it.
- Ruby allows we to declare methods that works with variable no. of parameters

Ex:-
```
def Sample (*text)
    puts "no. of param #{text.length}"
    for i in 0...text.length
        puts "#{text[i]}"
    end
end
Sample "1", "Rama"
Sample "2", "Lakshmi", "Ongole"
```

Result :-
```
no. of param 2
1
Rama
no. of param 3
2
Lakshmi
ongole
```

=> Ruby undef statement :-

- cancels method definition, it can't be appear in method body

Syntax :- Undef method_name

** Arrays :-

- To store more than one value into single variable
- Array index will starts from 0 to n-1
- In arrays no negative index

=> Creation of Arrays

- we are having so many ways to create arrays

① By using new method.

```
names = array.new
```

② By using new method with size of array

```
names = array.new(20)
```

- At index location we can insert the values

③ new method with values

names = array.new(4, "Rama", "Lakshmi", "Venky", "Nag")

- Here first argument represents the size of array and followed by array elements

④ new method with size and with values

values = array.new(10) { |x| x = x * 2 }

- It will take values from 0 to 9 values and its squares are inserted into array variable

⑤ By using Builtinmethod

names = Array.[](1,2,3,4)

(or)

names = Array[1,2,3,4]

(or)

names = array(1..4)

- All are for storing array values into variable from 1 to 4.

⇒ Functions to be perform on arrays :-

① first ⇒ array.first

- It will give the first element in array

② last ⇒ array.last

- It will give the last element in array

③ length ⇒ array.length

- It will give the size of array

④ size ⇒ array.size

- It will give the size of array

⑤ at(index) ⇒ array.at(index)

- It will give at particular index location value

⑥ delete_at(index) ⇒ array.delete_at(index)

- It will delete at particular index location value from array

(7) clear => array.clear

- It will clear all the values of array

(8) concat(array1) => array.concat(array1)

- It will combine two array values into single array

(9) push => array.push(—)

- It will insert the element into array

(10) pop => array.pop()

- It will remove last element from array

(11) reverse => array.reverse

- It will reverse the array elements

(12) sort => array.sort

- It will sort the array elements based on alphabet

(13) empty? => array.empty?

- It will check wether array is empty or not

(14) eql?() => array.eql?(array1)

- It will check both arrays elements are same or not.

** Hashes :-

- It is also called as associative arrays.
- It is collection of key-value pairs like "employee" => "salary"
- It is similar to arrays, except the indexing is done via arbitary key of an object type not an integer index
- Traverse the elements in hash is either key or value not an insertion order.
- If we are search for key if it is not exists then it will return NIL.

=> Creation of Hashes :-

1. We can create an empty hash by using new method.

```
months = Hash.new
```

- Giving default values for Hashes.

```
months = Hash.new("month")
(or)
months = Hash.new "month"
```

- Now, we can access any key in hash that has default value. If key or value doesn't exist, accessing the hash will return default values.

```
puts "#{months[0]}"      => month
puts "#{months[72]}"     => month.
```

2. We can also create hashes as follows key/value pair

```
H = Hash["a" => 10, "b" => 20]
(or)
H = Hash[:a => 10, :b => 20]
(or)
H = {:a, 10, :b, 20}
(or)
H = {a: 10, b: 20}
```

Ex:-

```
months = Hash.new("month")
months = {"a" => 10, "b" => 20}
key = months.keys
puts "#{key}"
```

Result
a
b.

=> Function to be perform on Hashes :-

① hash.keys
- It will gives the keys which are existed in hash

② hash.values
- It will gives the values which are existed in hash

③ hash.clear
- It will clear all the elements in hash.

④ hash.delete(key)

- It will remove key/value pair from hash

⑤ hash.has_key?(value)

- It will check key is existed in hash or not

⑥ hash.has_value?(value)

- It will check value is existed in hash or not

⑦ hash.empty?

- It will check hash is empty or not

⑧ hash.length

- It will return length of hash

⑨ hash.sort

- It will sort the elements in hashes based on key

⑩ hash.each {|key, value| block}

- It is used to print all the key/value pairs present in hash

⑪ hash.each_key {|value| block}

- It will retrive the values based on keys in hash

**Iterators :-

- Iterators are also equalent to loops.
- We are having 3 types of iterators in ruby

1. Numeric iterators
2. Enumerable objects
3. Custom iterators

1. Numeric iterators :-

- Integer class defines the following iterators
  - * upto
  - * times
- floating point numbers iterator is
  - * step.

Upto :- It will prints upto the number.

Syn :- integer.upto(integer)

Ex:- 4.upto(6) {|x| print x}

⇒ 4, 5, 6

Times :- It will repeatedly prints statement given no. of statements

Syn :- integer.times

Ex:- 3.times {"Hello"}

⇒ Hello Hello Hello.

Step :- It will increase the value step by step upto the value

0.step(Math::PI, 0.1) {|x| puts x}

- It prints from 0 to 3.14 each step increasement by 0.1

2. Enumerable objects :-

- Generally used for array, hash and ranges

each :- It will take the values every time upto taking NIL.

Ex:- [1,2,3].each {|x| print x}

(1..3).each {|x| print x}

- It is equalent to 1.upto(3)

- It is also used for IO classes

File.open(filename) do |f|

f.each {|line| print line}

end

→ It print each line in file

each_with_index

↳ print line along with index

each_char

↳ each time it will take only one character.

3. Custom iterators:-

- We can create our own iterators

Ex:-
```
def twice
  yeild
  yeild
end.
```

```
2.twice { "Hello" }
HelloHello
```

- yeild is used to call the blocks which are existed in ruby program.

** Pattern Matching (or) Regular Expressions:-

- A regular expression is a special sequence of characters that helps you match or find other strings or set of strings using specialized syntax followed by pattern.
- A regular expression literal is a pattern between slashes (or) delimited followed by %r

Syntax:- /pattern/

/pattern/im

%r| _ |

⇒ Regexp literals:-

- closing slash is not a true delimiter because it contains optional flag, that contains additional information

Modifier	Description
i -	ignore case when match text
m -	pattern to be matched against multiline text, so treat new line as ordinary char, it allows . to match newline
x -	extended sign allow white spaces and comments in regular expression.

EX:-

/Ruby?/ => match text followed Rub optional y

/ruby?/i => match text followed by RUB or rub optional y

/./mu => matches unicode char in multiline mode

%r|/| => matches single slash

%r[<|(.*)>]i => flag char allowed with syntax

/\(1)/ => matches open & close parantheses

/\\/ => matches single back slash

=> Regexp factory Method :-

- We can also create regexp with

  regexp.new or Regexp.compile

EX:-

1. Regexp.new ("Ruby?") => equivalent to /Ruby?/
2. Regexp.new ("ruby?", Regexp::IGNORECASE) => /ruby?/i
3. Regexp.compile(".", Regexp::MULTILINE, 'u") => /./mu.
4. Regexp.Union ("Ruby", "perl", "Phython", /java(script)?/)
   - ↳ It will search for more than one patterns at a time
   - ↳ It will search for 5 words

     Ruby, perl, phython, java, java script.

=> Regexp Syntax :-

1. Literal chars : direct chars writtern between slashes

   /ruby/ => matches ruby

2. char classes

   /[Rr]uby/ => Ruby or ruby

   /rub[ye]/ => ruby or rube

   /[aeiou]/ => matches lower case vowels

   /[^aeiou]/ => matches other than lowercase vowels.

/[0-9]/ - matches the digits 0 to 9

/[a-z]/ - matches lower case letters

/[A-Z]/ - matches Upper case letters

/[a-zA-Z0-9]/ - matches for characters and numbers

3. Special character classes

/./ => matches except newline

/./m => matches multiline mode.

/\d/ => matches the digits

/\D/ => matches except digits

/\w/ => matches word characters

/\W/ => matches non word character.

4. Repetition :-

/ruby?/ => y is optional

/ruby*/ => 0 or more y's

/ruby+/ => 1 or more y's

/\d{3}/ => exactly 3 digits

/\d{3,}/ => 3 or more digits

/\d{3,5}/ => 3 to 5 digits.

=> pattern matching with Regexp

- =~ is ruby pattern matching operator, one regexp operand and other string it contains.
- If a match is found, the operator returns string index at which the first match begin otherwise NIL

Ex:- pattern = /Ruby?/i

1. pattern =~ "backrub" => 4th location matched.

2. pattern =~ "rail" ⇒ NIL
not matched

3. "ruby rub" =~ pattern ⇒ 0
first match at 0th location.

**practical Web applications :-

- Ruby is a general purpose language, it can't properly called Web programming Language, but also most common use of ruby is creating web applications and working on web tools.
- Generally ruby used for * SMTP server, FTP server and web server and also used for CGI Programming.

⇒ writing CGI scripts

Let text.cgi

```
puts "HTTP/1.0 200 OK"
puts "content-type: text/HTML \n"
puts "<html><body> This is test </body></html>"
```

- the file should upload in Unix-based web hosting provider with proper file permissions and also call the script test.cgi then it is used as CGI script.

Ex:- we are having website http://www.example.com hosted with a linux web hosting provider and test.cgi script is uploaded into main directory and give execute permissions.

- Now visit http://www.example.com/test.cgi, it will return HTML page saying that This is test.
- Test.cgi is requested from web browser, the web server looks for test.cgi and execute it as ruby interpreter. the ruby interpreter returns basic HTTP header and

⇒ Using cgi.rb :-

- Ruby having special library called CGI.

Ex:-
```
require "cgi"
cgi = CGI.new
puts cgi.header
puts "<html><body>This is text</body></html>"
```

- we are creating cgi object and used to print the header.

⇒ FORM processing :-

- Using CGI class, we can give HTML query parameters in two ways
- Let us assume URL is

http://www.example.com/cgi-bin/test.cgi?
firstname = Rama & lastname = Lakshmi

1. we can directly access the parameters using CGI as follows

```
require "cgi"
cgi = CGI.new
cgi['firstname']      # Rama
cgi['lastname']       # Lakshmi
```

2. Another way to access the variable
- the code gives all keys and values of hash.

```
require "cgi"
cgi = CGI.new
h = cgi.params      # {"firstname"=>"Rama",
                    #  "lastname"=>"Lakshmi"}
h['firstname']      # Rama
h['lastname']       # Lakshmi
```

- We can retrieve all the keys

```
require "cgi"
cgi = CGI.new
cgi.keys        # firstname, lastname
```

- If the form contains multiple fields for same name then the values will be return as an array. the result to get index of first, by using params retrieve all

- Let form having 3 names "Nag", "venky", "padma"

```
require "cgi"
cgi = CGI.new
cgi['name']          # Nag
cgi.params['name']   # Nag, venky, padma
cgi.keys             # name
cgi.params           # name => Nag, venky, padma
```

Note:- Ruby will take care of GET and POST methods automatically.

HTML FORM :-

```
<html>
  <body>
    <form method="POST" action="./cgi-bin/test.cgi">
      Firstname: <input type="text" name="Firstname" /> <br>
      Lastname : <input type="text" name="Lastname" /> <br>
      <input type="submit" value="Submit" />
    </form>
  </body>
</html>
```

⇒ Creating Forms and HTML :-

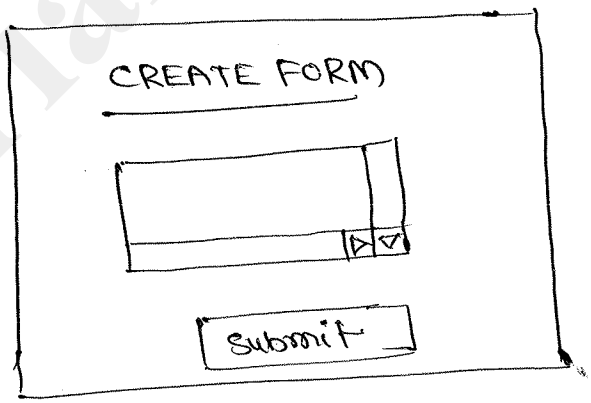
- CGI contains huge no. of methods to create HTML, we are having one method for each tag. Inorder to enable these methods we need to create CGI object by using CGI.new.

- For our understanding each method content is written with in blocks, the code block returns string.

```
require "cgi"
cgi = CGI.new("Html4")
cgi.out
  {
    cgi.html
      {
        cgi.head
          {
            cgi.title {"test"}
          } +
        cgi.body
          { "\n" +
            cgi.form
              { "\n" +
                cgi.h1 { "CREATE FORM" } + "\n" +
                cgi.hr +
                cgi.textarea ("get_text") + "\n" +
                cgi.br +
                cgi.submit
              }
          }
      }
  }
```

- If we write same code in HTML

```
Content-type : text/html
Content-length : 400
<html>
  <head>
    <title> Test </title>
  </head>
  <body>
    <form method = "post" >
      <h1> CREATE FORM </h1>
      <hr>
      <text area cols = "70" rows = "10"
          name = "get_text" > </text area>
      <br>
      <input type = "submit" value = "submit" />
    </form>
  </body>
</html>
```



TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well