CHAPTER - 1

REVIEW OF WEB TECHNOLOGIES I

Basic Internet Protocols:

Network protocols are the set of rules that governs data communication. Types of protocols are explained below:

1. **TCP (Transmission Control Protocol):**

When information is sent over the Internet, it is generally broken up into smaller pieces or "packets". The use of packets facilitates speedy transmission since different parts of a message can be sent by different routes and then reassembled at the destination. It is also a safety measure to minimize the chances of losing information in the transmission process. TCP is the means for creating the packets, putting them back together in the correct order at the end, and checking to make sure that no packets got lost in transmission. If necessary, TCP will request that a packet be resent.

2. **IP (Internet Protocol):**

Internet Protocol (IP) is the method used to route information to the proper address. Every computer on the Internet has to have its own unique address known as the IP address. Every packet sent will contain an IP address showing where it is supposed to go. A packet may go through a number of computer routers before arriving at its final destination and IP controls the process of getting everything to the designated computer. Note that IP does not make physical connections between computers but relies on TCP for this function. IP is also used in conjunction with other protocols that create connections.

3. **HTTP (Hyper Text Transfer Protocol):**

HTTP is a communication protocol. It defines mechanism for communication between browser and the web server. It is also called request and response protocol because the communication between browser and server takes place in request and response pairs. It uses port 80.

4. **HTTPS(Hypertext Transfer Protocol over Secure Socket Layer):**

HTTPS was first introduced by Netscape. HTTPS takes care of secure communication between a web server and web browser. HTTPS typically handles credit card transaction and other sensitive data. A Web page using this protocol will have https: at the front of its URL. It uses port 443.

5. **FTP (File Transfer Protocol):**

File Transport Protocol is a program that allows users and computers to send and receive large portions of data through a private or public network. It can also be used to send configuration files and software updates to network devices, such as switches and routers. FTP uses ports for communications and also uses encryption to protect the information being received and sent. It uses port 20 for data transfer and port 21 for connection.

6. **SMTP (Simple Mail Transfer Protocol):**

SMTP is a TCP/IP protocol used in sending and receiving e-mail. However, since it is limited in its ability to queue massages at the receiving end, it is usually used with one of two other protocols POP3 or IMAP that let the user save the messages in a server mailbox and download them periodically from the server. In other words, users typically use a program that uses SMTP for sending e-mail and either POP3 or IMAP for receiving e-mail. On UNIX based systems, send mail is the most widely used SMTP server for e-mail. It uses port 25.

7. **POP (Post Office Protocol):**

POP is one of the most commonly used protocols used to receive e-mail on many e-mail clients. There are two different versions of POP: POP2 and POP3. POP2 was an early standard of POP that was only capable of receiving e-mail and required SMTP to send e-mail. POP3 is the latest standard and can send and receive e-mail only using POP, but can also be used to receive e-mail and then use SMTP to send e-mail.

POP3 is designed to delete mail on the server as soon as the user has downloaded it. However, some implementations allow users or an administrator to specify that mail be saved for some period of time. POP can be thought of as a "store-and-forward" service. It uses port 110.

8. **IMAP (Internet Mail Access Protocol):**

IMAP (Internet Message Access Protocol) is a standard email protocol that stores email messages on a mail server, but allows the end user to view and manipulate the messages as though they were stored locally on the end user's computing device(s). This allows users to organize messages into folders, have multiple client applications know which messages have been read, flag messages for urgency or follow-up and save draft messages on the server.

IMAP can be contrasted with another client/server email protocol, Post Office Protocol 3 (POP3). With POP3, mail is saved for the end user in a single mailbox on the server and moved to the end user's device when the mail client opens. While POP3 can be thought of as a "store-and-forward" service, IMAP can be thought of as a remote file server. It uses port 143.

9. **UDP (User Datagram Protocol):**

The User Datagram Protocol is one of the core members of the internet protocol suite. UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network and so there is no guarantee of delivery, ordering, or duplicate protection. If error correction facilities are needed at the network interface level, an application may use the TCP or SCTP (Stream Control Transmission Protocol) which are design for this purpose.

10. **Telnet:**

Telnet (short for TErminal NETwork) is a network protocol used to provide a command line interface for communicating with a device. Telnet is used most often for remote management but also sometimes for the initial setup for some devices, especially network hardware like switches, access points, etc. Managing files on a website is also something Telnet is sometimes

used for. Telnet is sometimes written in uppercase as TELNET and may also be misspelled as Telenet. It uses port 23.

11. DHCP (Dynamic Host Configuration Protocol):

DHCP (Dynamic Host Configuration Protocol) is a protocol used to provide quick, automatic, and central management for the distribution of IP addresses within a network. DHCP is also used to configure the proper subnet mask, default gateway, and DNS server information on the device. It uses port 68.

HTTP - Overview:

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This is the foundation for data communication for the World Wide Web (i.e. internet) since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.

Basically, HTTP is a TCP/IP based communication protocol, that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.

BASIC FEATURES:

There are three basic features that make HTTP a simple but powerful protocol:

- **HTTP Is Connectionless:**

The HTTP client, i.e., a browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send a response back.

- **HTTP Is Media Independent:**

It means, any type of data can be sent by HTTP as long as both the client and the server know how to handle the data content. It is required for the client as well as the server to specify the content type using appropriate MIME *(Multipurpose Internet Mail Extensions)* type.

- **HTTP Is Stateless:**

As mentioned above, HTTP is connectionless and it is a direct result of HTTP being a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.

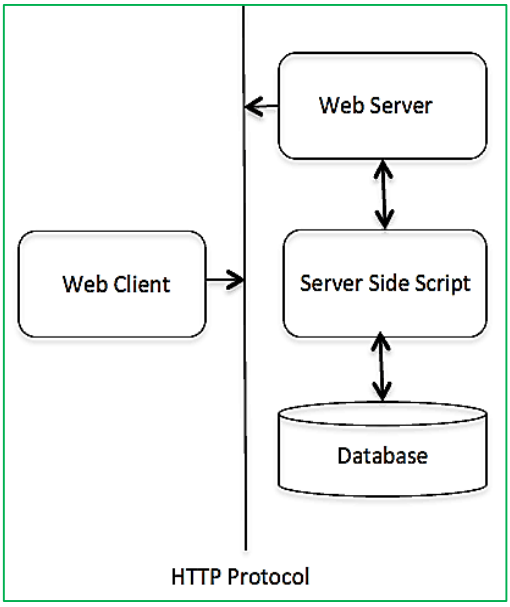
BASIC ARCHITECTURE:

The following diagram shows a very basic architecture of a web application and depicts where HTTP sits:

The HTTP protocol is a request/response protocol based on the client/server based architecture where web browsers, robots and search engines, etc. act like HTTP clients, and the Web server acts as a server.

- **Client:**

The HTTP client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME *(Multipurpose Internet Mail Extensions)* like message containing request modifiers, client information, and possible body content over a TCP/IP connection.



- **Server:**

The HTTP server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME *(Multipurpose Internet Mail Extensions)* like message containing server information, entity Meta information, and possible entity-body content.

HTTP - Messages:

HTTP is based on the client-server architecture model and a stateless request/response protocol that operates by exchanging messages across a reliable TCP/IP connection.

An HTTP "client" is a program (Web browser or any other client) that establishes a connection to a server for the purpose of sending one or more HTTP request messages. An HTTP "server" is a program (generally a web server like Apache Web Server or Internet Information Services IIS, etc.) that accepts connections in order to serve HTTP requests by sending HTTP response messages.

HTTP makes use of the Uniform Resource Identifier (URI) to identify a given resource and to establish a connection. Once the connection is established, **HTTP messages** are passed in a format similar to that used by the Internet mail [RFC *(Remote Function Call)* 5322] and the Multipurpose Internet Mail Extensions (MIME) [RFC2045]. These messages include **requests** from client to server and **responses** from server to client which will have the following format: ***HTTP-message = <Request> | <Response>; HTTP/1.1 messages***

HTTP requests and HTTP responses use a generic message format of RFC 822 for transferring the required data. This generic message format consists of the following four items.

- A Start-line
- Zero or more header fields followed by CRLF *(Carriage Return, Line Feed)*
- An empty line (i.e. a line with nothing preceding the CRLF) indicating the end of the header fields
- Optionally a message body

1. **Message Start-Line:**

A start-line will have the following generic syntax: ***start-line = Request-Line | Status-Line***

Example:

GET /hello.htm HTTP/1.1 (This is Request-Line sent by the client)
HTTP/1.1 200 OK (This is Status-Line sent by the server)

2. **Header Fields:**

HTTP header fields provide required information about the request or response, or about the object sent in the message body. There are four types of HTTP message headers:

- **General-header:** These header fields have general applicability for both request and response messages.
- **Request-header:** These header fields have applicability only for request messages.
- **Response-header:** These header fields have applicability only for response messages.
- **Entity-header:** These header fields define Meta information about the entity-body or, if nobody is present, about the resource identified by the request.

All the above mentioned headers follow the same generic format and each of the header field consists of a name followed by a colon (**:**) and the field value as follows:

message-header = field-name ":" [field-value]

Following are the examples of various header fields:

User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain

3. **Message Body:**

The message body part is optional for an HTTP message but if it is available, then it is used to carry the entity-body associated with the request or response. If entity body is associated, then usually **Content-Type** and **Content-Length** headers lines specify the nature of the body associated.

A message body is the one which carries the actual HTTP request data (including form data and uploaded, etc.) and HTTP response data from the server (including files, images, etc.). Shown below is the simple content of a message body:

```
<html>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

HTTP - Requests:

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

A Request-line
Zero or more header (General|Request|Entity) fields followed by CRLF
An empty line (i.e. a line with nothing preceding the CRLF) indicating the end of the header fields
Optionally a message-body

The following sections explain each of the entities used in an HTTP request message.

- **Request-Line:**

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by space SP characters.

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Let's discuss each of the parts mentioned in the Request-Line.

- **Request Method:**

The request **method** indicates the method to be performed on the resource identified by the given **Request-URI**. The method is case-sensitive and should always be mentioned in uppercase. The following table lists all the supported methods in HTTP/1.1.

S.N.	Method and Description
1.	**GET** The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2.	**HEAD** Same as GET, but it transfers the status line and the header section only.
3.	**POST** A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4.	**PUT** Replaces all the current representations of the target resource with the uploaded content.
5.	**DELETE** Removes all the current representations of the target resource given by URI.
6.	**CONNECT** Establishes a tunnel to the server identified by a given URI.
7.	**OPTIONS** Describe the communication options for the target resource.

8.	**TRACE** Performs a message loop back test along with the path to the target resource.

Request-URI:

The Request-URI is a Uniform Resource Identifier and identifies the resource upon which to apply the request. Following are the most commonly used forms to specify an URI:

Request-URI = "*" | absoluteURI | abs_path | authority

S.N.	Method and Description
1	The asterisk * is used when an HTTP request does not apply to a particular resource, but to the server itself, and is only allowed when the method used does not necessarily apply to a resource. For example: **OPTIONS * HTTP/1.1**
2	The **absoluteURI** is used when an HTTP request is being made to a proxy. The proxy is requested to forward the request or service from a valid cache, and return the response. For example: **GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1**
3	The most common form of Request-URI is that used to identify a resource on an origin server or gateway. For example, a client wishing to retrieve a resource directly from the origin server would create a TCP connection to port 80 of the host "www.w3.org" and send the following lines: **GET /pub/WWW/TheProject.html HTTP/1.1** **Host: www.w3.org** Note that the absolute path cannot be empty; if none is present in the original URI, it MUST be given as "/" (the server root).

Request Header Fields:

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers. Here is a list of some important Request-header fields that can be used based on the requirement:

- ✓ Accept-Charset
- ✓ Accept-Encoding
- ✓ Accept-Language
- ✓ Authorization
- ✓ Expect
- ✓ From
- ✓ Host
- ✓ If-Match
- ✓ If-Modified-Since
- ✓ If-None-Match
- ✓ If-Range
- ✓ If-Unmodified-Since
- ✓ Max-Forwards

- ✓ Proxy-Authorization
- ✓ Range
- ✓ Referer
- ✓ TE
- ✓ User-Agent

Examples of Request Message

Now let's put it all together to form an HTTP request to fetch **hello.htm** page from the web server running on tutorialspoint.com

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Here we are not sending any request data to the server because we are fetching a plain HTML page from the server. Connection is a general-header, and the rest of the headers are request headers. The following example shows how to send form data to the server using request message body:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
licenseID=string&content=string&/paramsXML=string
```

Here the given URL */cgi-bin/process.cgi* will be used to process the passed data and accordingly, a response will be returned. Here **content-type** tells the server that the passed data is a simple web form data and **length** will be the actual length of the data put in the message body. The following example shows how we can pass plain XML to our web server:

```
POST /cgi-bin/process.cgi HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="http://clearforest.com/">string</string>
```

HTTP - Responses:

After receiving and interpreting a request message, a server responds with an HTTP response message:

A Status-line
Zero or more header (General|Response|Entity) fields followed by CRLF
An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
Optionally a message-body

The following sections explain each of the entities used in an HTTP response message.

Message Status-Line:

A Status-Line consists of the protocol version followed by a numeric status code and its associated textual phrase. The elements are separated by space SP characters.

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

HTTP Version:

A server supporting HTTP version 1.1 will return the following version information:

HTTP-Version = HTTP/1.1

Status Code:

The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

S.N.	Code and Description
1	**1xx: Informational** It means the request was received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all registered status codes.

- **Response Header Fields:**

The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status- Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

- ✓ Accept-Ranges
- ✓ Age
- ✓ ETag
- ✓ Location
- ✓ Proxy-Authenticate
- ✓ Retry-After
- ✓ Server
- ✓ Vary
- ✓ WWW-Authenticate

Examples of Response Message:

Now let's put it all together to form an HTTP response for a request to fetch the **hello.htm** page from the web server running on tutorialspoint.com

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
    <body>
        <h1>Hello, World!</h1>
    </body>
</html>
```

The following example shows an HTTP response message displaying error condition when the web server could not find the requested page:

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Connection: Closed
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
    <html>
        <head>
            <title>404 Not Found</title>
```

```
    </head>
    <body>
      <h1>Not Found</h1>
      <p>The requested URL /t.html was not found on this server.</p>
    </body>
</html>
```

Following is an example of HTTP response message showing error condition when the web server encountered a wrong HTTP version in the given HTTP request:

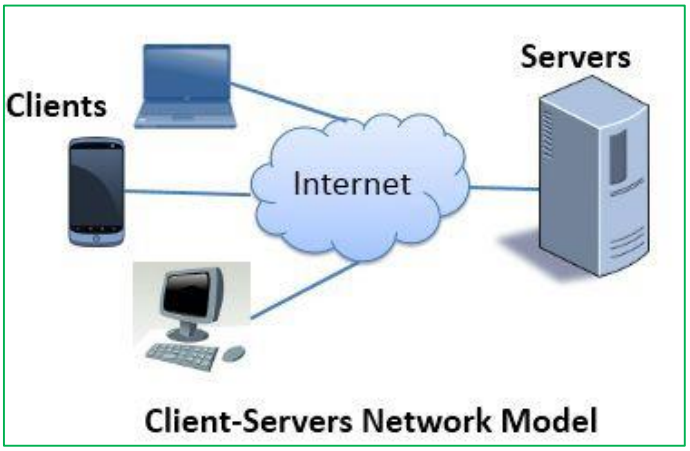
```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2012 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 230
Content-Type: text/html; charset=iso-8859-1
Connection: Closed

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
    <html>
        <head>
          <title>400 Bad Request</title>
        </head>
        <body>
          <h1>Bad Request</h1>
          <p>Your browser sent a request that this server could not understand.</p>
          <p>The request line contained invalid characters following the protocol string.</p>
        </body>
    </html>
```

Different Architectures of Connection

1. Client-Server Architecture:

The Client-Server network model is widely used network model. Here, **Server** is a powerful system that stores the data or information in it. On the other hands, the **Client** is the machine which let the users access the data on the remote server.



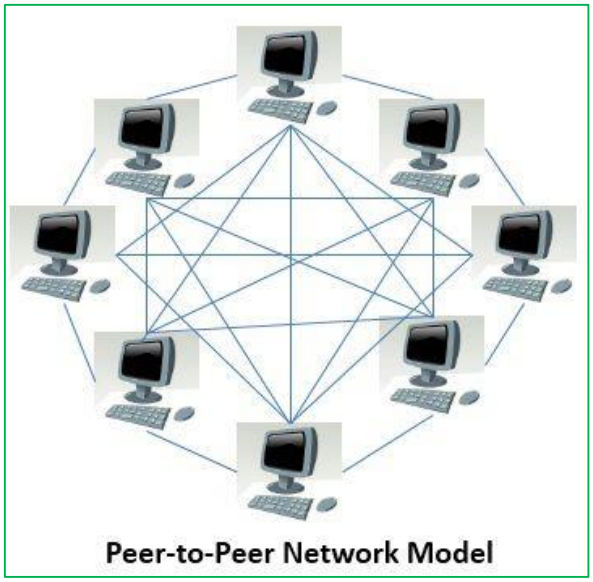
The **system administrator** manages the data on the server. The client machines and the server are connected through a **network**. It allows the clients to access data even if the client machine and server are far apart from each other.

In Client-Server model, the client process on the client machine sends the **request** to the server process on the server machine. When the server receives the client request, it lookouts for the requested data and **send** it back with the reply.

As all the services are provided by a centralized server, there may be chances of server getting **bottlenecked**, slowing down the efficiency of the system.

2. Peer-to-Peer Architecture:

Unlike Client-Server, the Peer-to-Peer model does not distinguish between client and server instead each **node** can either be a client or a server depending on the whether the node is **requesting** or **providing** the services. Each node is considered as a **peer**.



Peer-to-Peer Network Model

To become a part of peer-to-peer, a node must initially **join** the network. After joining it must start to provide services to and must request the services from other nodes in the peer-to-peer system. There are **two ways** to know which node provides which services; they are as follow:

- When a node enters the peer-to-peer system, it must **register** the services it will be providing, into a **centralized lookup service** on the network. When a node desires for any specific service it must contact centralized lookup services to check out which node will provide the desired services. Rest of the communication is done by the desiring node and the service providing node.
- A node desiring for the specific services must **broadcast** the request for services to all other nodes in the peer-to-peer system. The node providing the requested service will **respond** to the node making the request.

Peer-to-Peer network has the advantage over client-server that the server is **not bottlenecked** as the services are provided by the several nodes distributed in a peer-to-peer system.

Basis	Client-Server	Peer-To-Peer:
Basic	There is a specific server and specific clients connected to the server.	Clients and server are not distinguished; each node act as client and server.
Service	The client request for service and server respond with the service.	Each node can request for services and can also provide the services.
Focus	Sharing the information.	Connectivity.
Data	The data is stored in a centralized server.	Each peer has its own data.
Server	When several clients request for the services simultaneously, a server can get bottlenecked.	As the services are provided by several servers distributed in the peer-to-peer system, a server in not bottlenecked.
Expense	The client-server are expensive to implement.	Peer-to-peer are less expensive to implement.
Stability	Client-Server is more stable and scalable.	Peer to Peer suffers if the number of peers increases in the system.

Differentiate between Client Side and Server Side Scripting Language:

Client Side Scripting Language	Server Side Scripting Language
Client side scripting is used when the user's browser already has all the code and the page is altered on the basis of the users input.	Server side scripting is used to create dynamic pages based a number of conditions when the users browser makes a request to the server.
The Web Browser executes the client side scripting that resides at the user's computer.	The Web Server executes the server side scripting that produces the page to be sent to the browser.
The browser receives the page sent by the server and executes the client-side scripts.	Server executes server-side scripts to send out a page but it does not execute client-side scripts.
Client side scripting cannot be used to connect to the databases on the web server.	Server side scripting is used to connect to the databases that reside on the web server.
Client side scripting can't access the file system that resides at the web server.	Server side scripting can access the file system residing at the web server.
The files and settings that are local at the user's computer can be accessed using Client side scripting.	The settings that belong to Web server can be accessed using Server side scripting.
Client side scripting is possible to be blocked by the user.	Server side scripting can't be blocked by the user.
Response from a client-side script is faster as compared to a server-side script because the scripts are processed on the local computer.	Response from a server-side script is slower as compared to a client-side script because the scripts are processed on the remote computer.
Examples of Client side scripting languages : JavaScript, VB script, etc.	Examples of Server side scripting languages: PHP, JSP, ASP, ASP.Net, Ruby, Perl and many more.