

# Web Programming Fundamentals

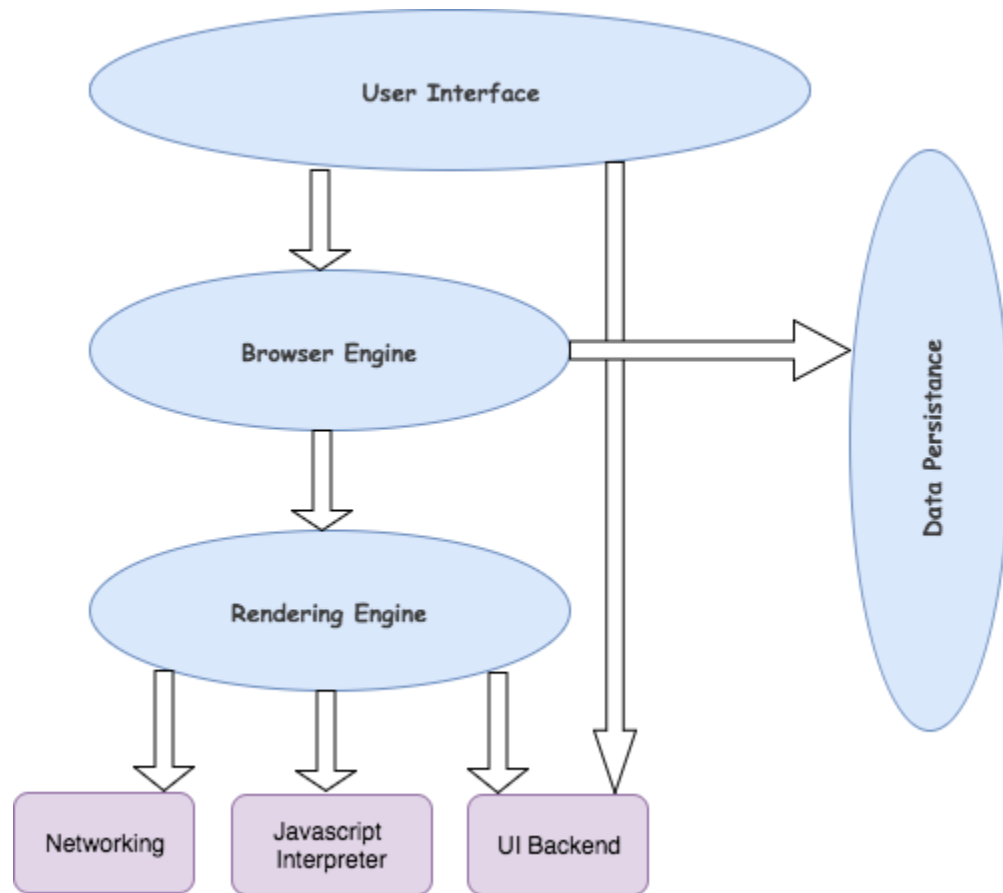
# How does web browser work?

- A browser is a software application used to locate, retrieve and display content on the World Wide Web, including Web pages, images, video and other files.
- As a client/server model, the browser is the client run on a computer that contacts the Web server and requests information.
- The Web server sends the information back to the Web browser which displays the results on the computer or other Internet-enabled device that supports a browser.

# How does web browser work?

- Today's browsers are fully-functional software suites that can interpret and display HTML Web pages, applications, JavaScript, AJAX and other content hosted on Web servers.
- Many browsers offer plug-ins which extend the capabilities of the software so it can display multimedia information (including sound and video)
- The browser can be used to perform tasks such as videoconferencing, to design web pages or add anti-phishing filters and other security features to the browser.

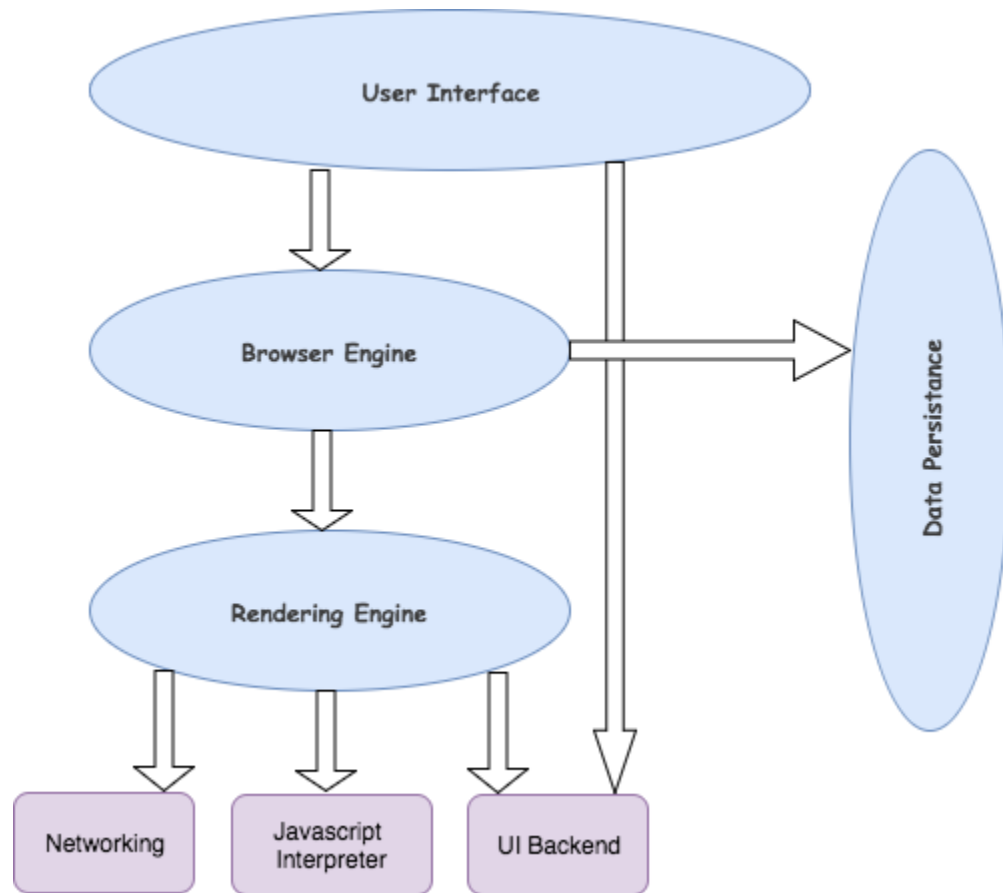
# How does web browser work?



- **The User Interface:**

- The user interface is the space where User interacts with the browser.
- It includes the address bar, back and next buttons, home button, refresh and stop, bookmark option, etc.
- Every other part, except the window where requested web page is displayed, comes under it.

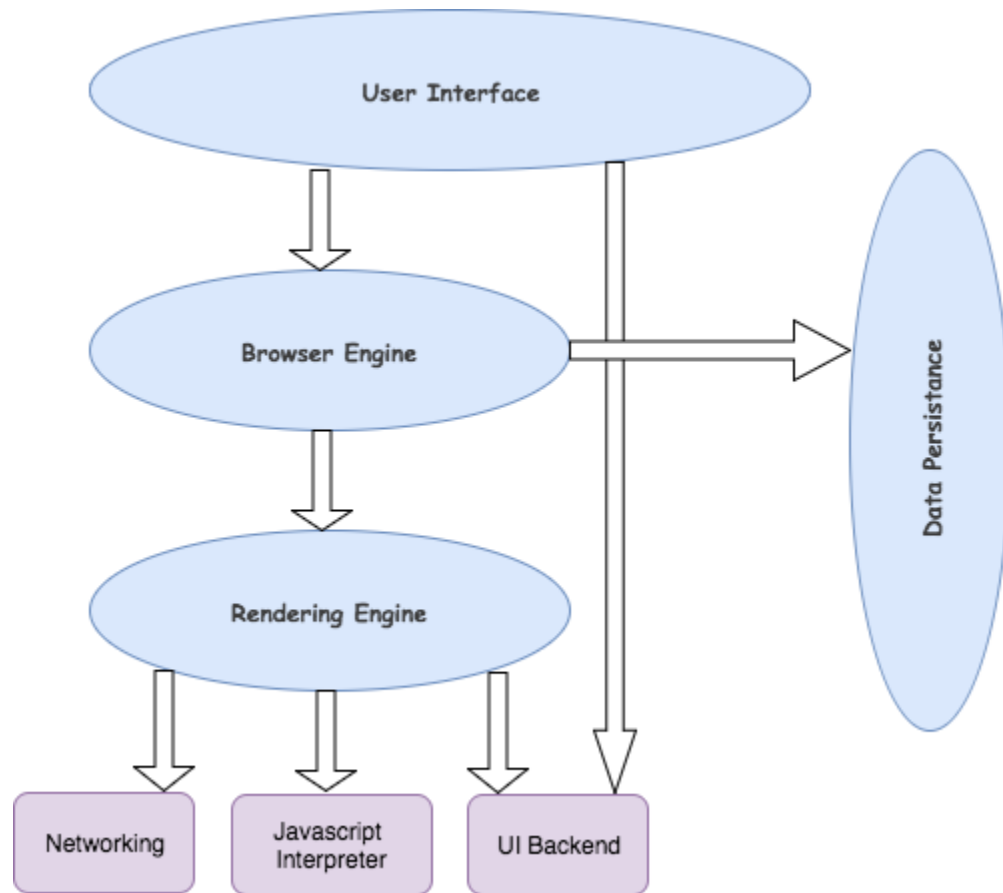
# How does web browser work?



- **The Browser Engine:**

- The browser engine works as a bridge between the User interface and the rendering engine.
- According to the inputs from various user interfaces, it queries and manipulates the rendering engine.

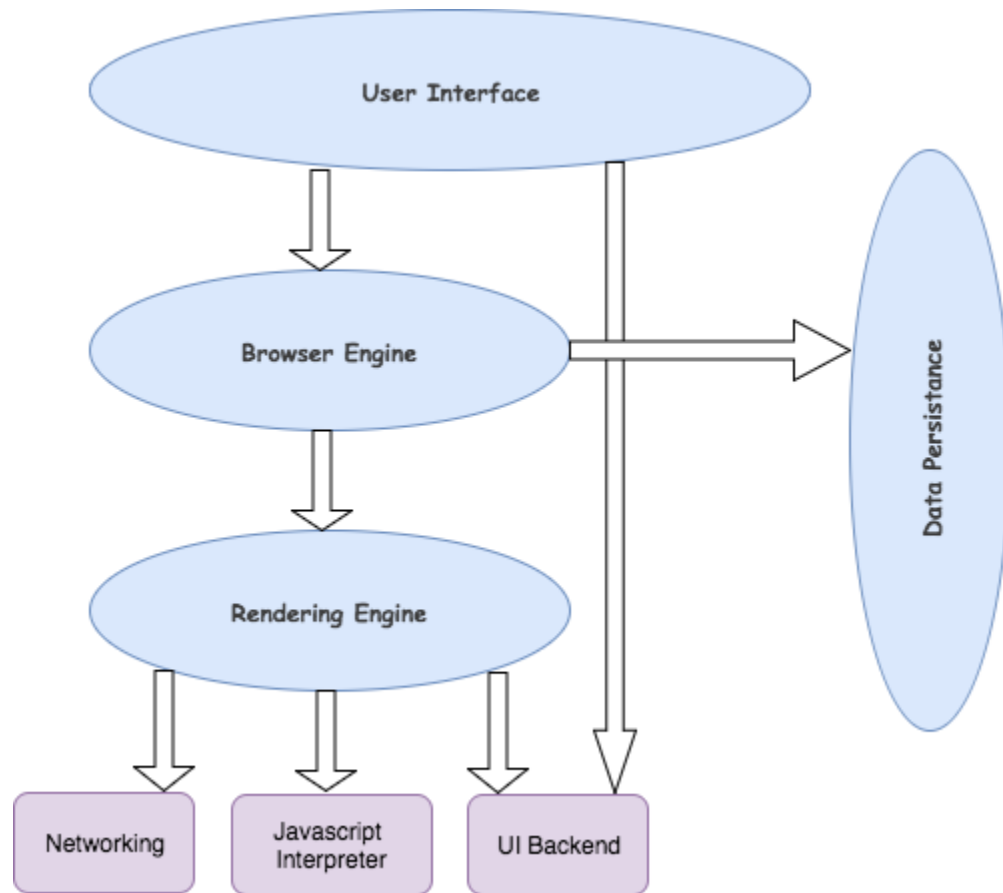
# How does web browser work?



- **The Rendering Engine:**

- The rendering engine, as the name suggests is responsible for rendering the requested web page on the browser screen.
- The rendering engine interprets the HTML, XML documents and images that are formatted using CSS and generates the layout that is displayed in the User Interface.
- However, using plugins or extensions, it can display other types data also.
- Different browsers user different rendering engines:
  - \* Internet Explorer: Trident
  - \* Firefox & other Mozilla browsers: Gecko
  - \* Chrome & Opera 15+: Blink
  - \* Chrome (iPhone) & Safari: Webkit

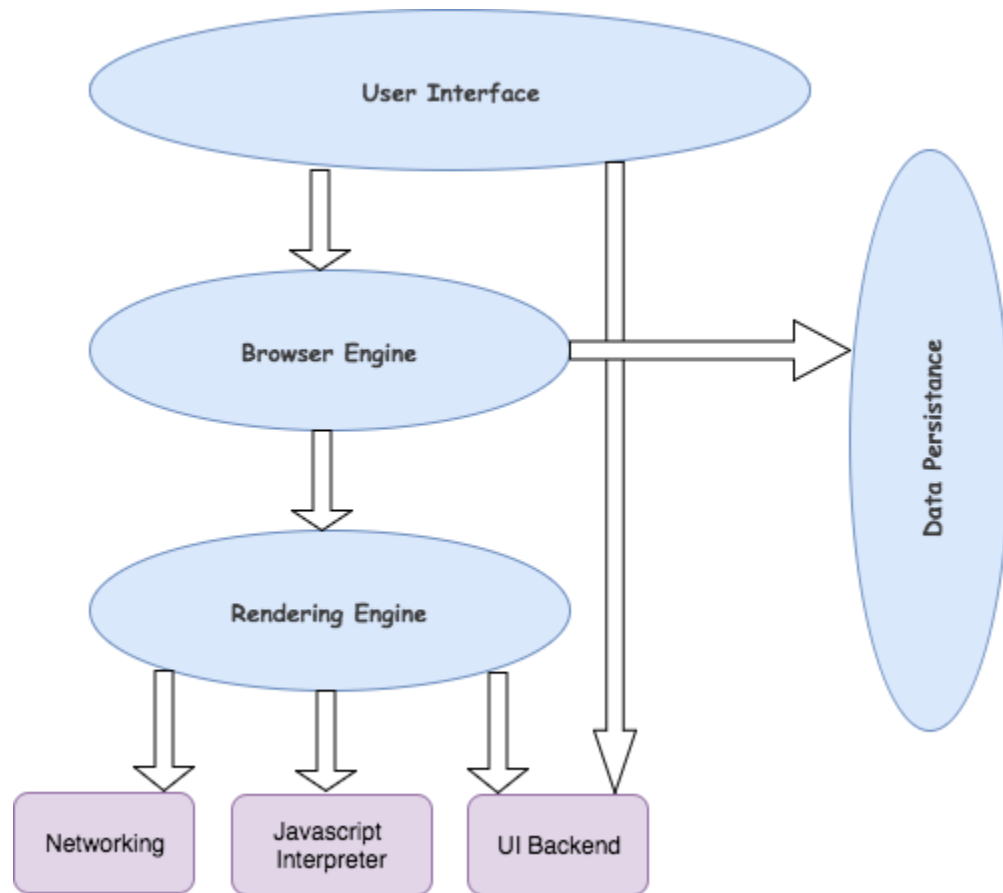
# How does web browser work?



- **Networking:**

- Component of the browser which retrieves the URLs using the common internet protocols of HTTP or FTP.
- The networking component handles all aspects of Internet communication and security.
- The network component may implement a cache of retrieved documents in order to reduce network traffic.

# How does web browser work?



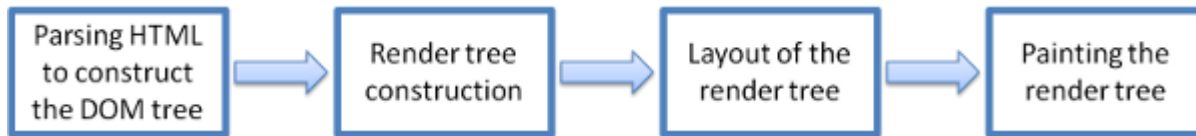
- **Networking:**

- Component of the browser which retrieves the URLs using the common internet protocols of HTTP or FTP.
- The networking component handles all aspects of Internet communication and security.
- The network component may implement a cache of retrieved documents in order to reduce network traffic.



# Rendering Engine:

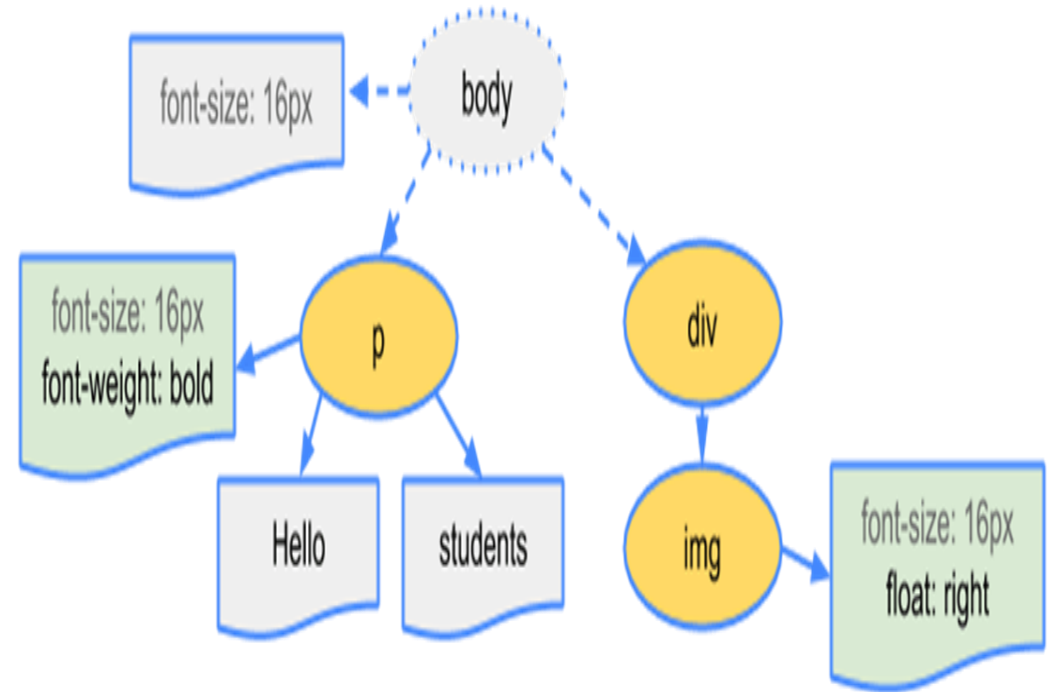
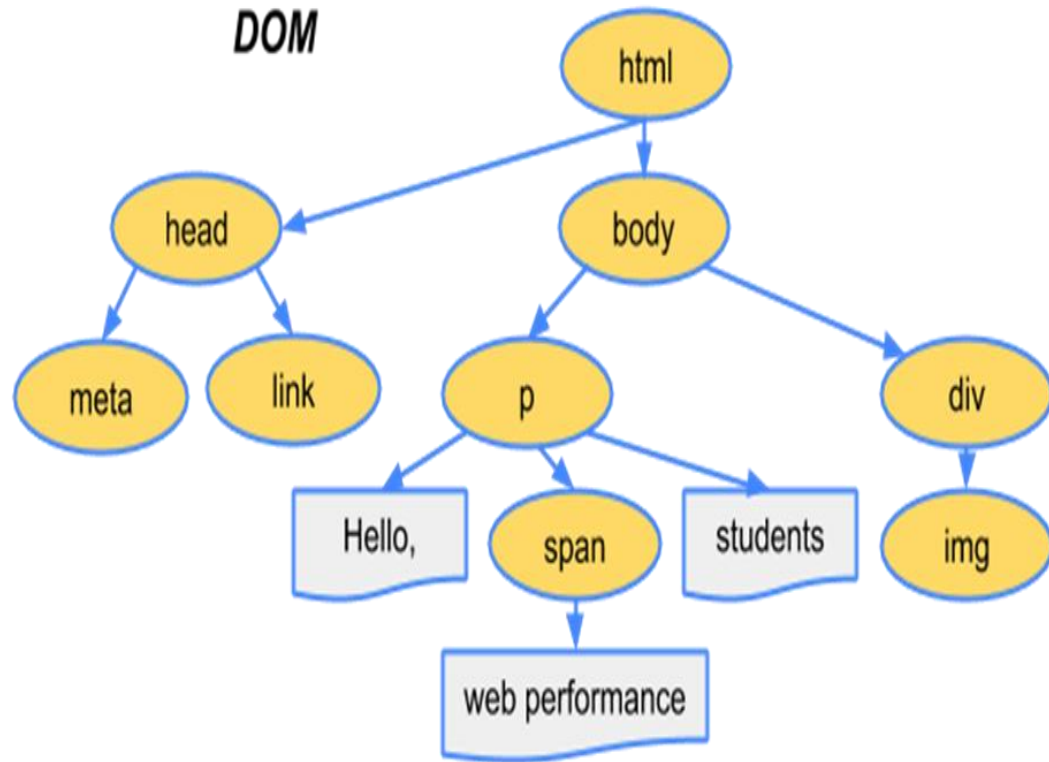
- The networking layer will start sending the contents of the requested documents to the rendering engine in chunks of 8KBs.



- The rendering engine parses the chunks of HTML document and convert the elements to DOM nodes in a tree called the “**content tree**” or the “**DOM tree**”.
- It also parses both the external CSS files as well in style elements.

- While the DOM tree is being constructed, the browser constructs another tree, the **render tree**.
- This tree is of visual elements in the order in which they will be displayed.
- It is the visual representation of the document.
- The purpose of this tree is to enable painting the contents in their correct order.
- Firefox calls the elements in the render tree “frames”.
- WebKit uses the term renderer or render object.

## DOM



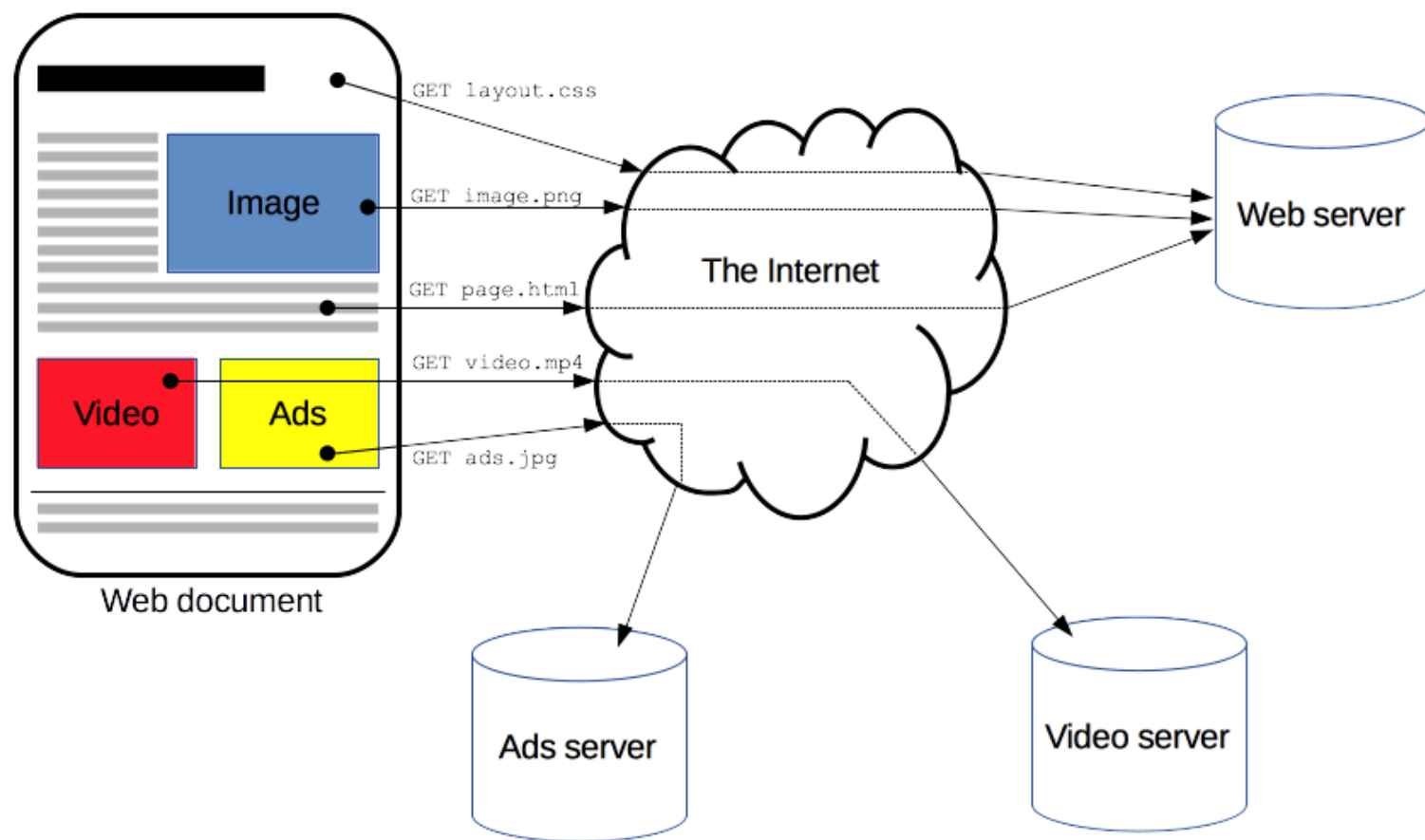
- After the construction of the render tree, it goes through a “**layout process**” of the render tree.
- When the renderer is created and added to the tree, it does not have a position and size.
- The process of calculating these values is called layout or reflow.
- This means giving each node the exact coordinates where it should appear on the screen.
- The position of the root renderer is 0,0 and its dimensions are the viewport—the visible part of the browser window.
- All renderers have a “layout” or “reflow” method, each renderer invokes the layout method of its children that need layout.

- The next stage is **painting**.
- In the painting stage, the render tree is traversed and the renderer's "paint()" method is called to display content on the screen.
- Painting uses the UI backend layer.

# HTTP

- **HTTP** is a protocol which allows the fetching of resources, such as HTML documents.
- It is the foundation of any data exchange on the Web.
- It is a client-server protocol.
- which means requests are initiated by the recipient, usually the Web browser.
- A complete document is reconstructed from the different sub-documents fetched, for instance text, layout description, images, videos, scripts, and more.

# HTTP



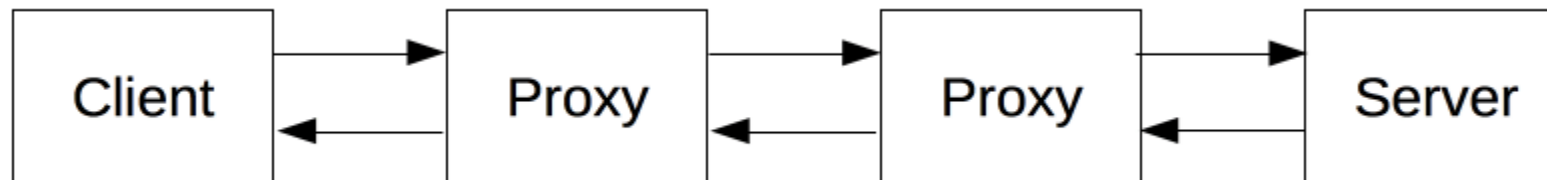


# HTTP

- Clients and servers communicate by exchanging individual messages.
- The messages sent by the client, usually a Web browser, are called requests.
- The messages sent by the server as an answer are called responses.

# Components of HTTP:

- HTTP is a client-server protocol:
- Requests are sent by one entity, the user-agent (or a proxy on behalf of it).
- Each individual request is sent to a server, which handles it and provides an answer, called the *response*.
- Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches.



# Components of HTTP:

- In reality, there are more computers between a browser and the server handling the request: there are routers, modems, and more.

# Client: the user-agent

- The user-agent is any tool that acts on the behalf of the user.
- This role is primarily performed by the Web browser.
- The browser is always the entity initiating the request.
- To present a Web page, the browser sends an original request to fetch the HTML document that represents the page.
- It then parses this file, making additional requests corresponding to execution scripts, layout information (CSS) to display, and sub-resources contained within the page (usually images and videos).
- The Web browser then mixes these resources to present to the user a complete document, the Web page.

- Scripts executed by the browser can fetch more resources in later phases and the browser updates the Web page accordingly.
- A Web page is a hypertext document.
- This means some parts of displayed text are links which can be activated (usually by a click of the mouse) to fetch a new Web page, allowing the user to direct their user-agent and navigate through the Web.
- The browser translates these directions in HTTP requests, and further interprets the HTTP responses to present the user with a clear response.

# The Web Server

- The Server, which serves the document as requested by the client.
- A server appears as only a single machine virtually: this is because it may actually be a collection of servers, sharing the load (load balancing) or a complex piece of software interrogating other computers (like cache, a DB server, or e-commerce servers), totally or partially generating the document on demand.
- A server is not necessarily a single machine, but several server software instances can be hosted on the same machine.

# Proxies

- Between the Web browser and the server, numerous computers and machines relay the HTTP messages.
- Due to the layered structure of the Web stack, most of these operate at the transport, network or physical levels, becoming transparent at the HTTP layer and potentially making a significant impact on performance.
- Those operating at the application layers are generally called **proxies**.
- These can be transparent, forwarding on the requests they receive without altering them in any way,
- or non-transparent, in which case they will change the request in some way before passing it along to the server.

# Proxies

- Proxies may perform numerous functions:
  - caching (the cache can be public or private, like the browser cache)
  - filtering (like an antivirus scan or parental controls)
  - load balancing (to allow multiple servers to serve the different requests)
  - authentication (to control access to different resources)
  - logging (allowing the storage of historical information)



# Characteristics/ aspects of HTTP

- HTTP is simple.
  - HTTP is generally designed to be simple and human readable.
  - HTTP messages can be read and understood by humans, providing easier testing for developers, and reduced complexity for newcomers.
- HTTP is Extensible.
  - HTTP headers make this protocol easy to extend and experiment with.
  - New functionality can even be introduced by a simple agreement between a client and a server about a new header's semantics.

- HTTP is stateless but not sessionless.
  - HTTP is stateless.
  - There is no link between two requests being successively carried out on the same connection.

# Common Features controllable with HTTP

- Caching:
  - How documents are cached can be controlled by HTTP.
  - The server can instruct proxies and clients, about what to cache and for how long.
- Authentication:
  - Some pages may be protected so that only specific users can access them.
  - Basic authentication may be provided by HTTP, either using the www-Authenticate or by setting a specific session during HTTP cookies.

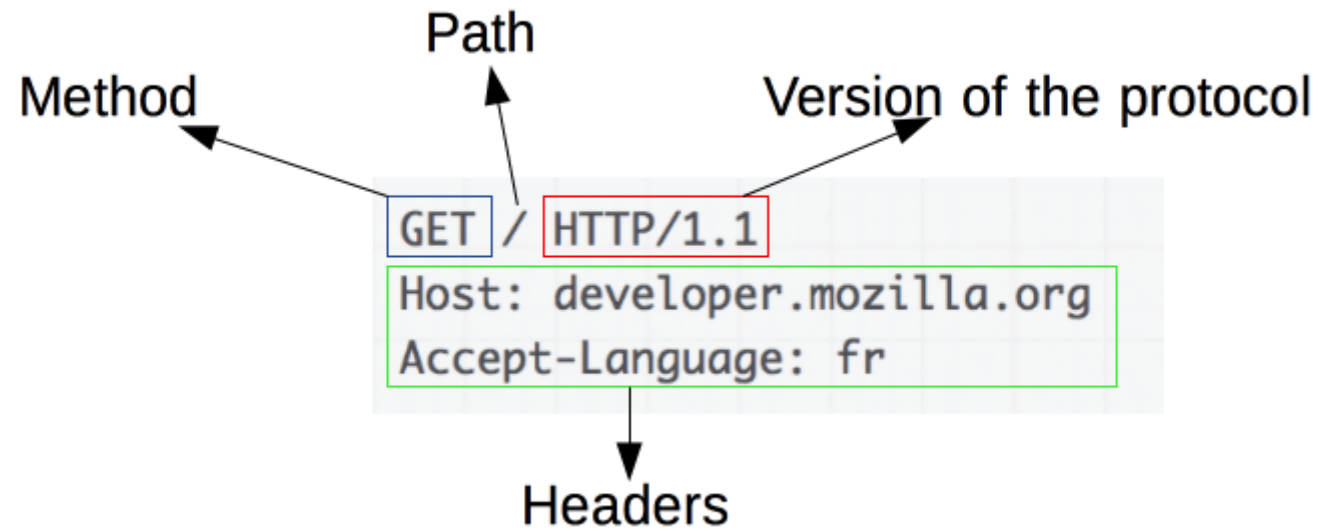
- HTTP Cookies:

- HTTP cookies allows you to link requests with the state of the server.
- This creates sessions, despite basic HTTP being a state-less protocol.
- This is useful not only for e-commerce shopping baskets, but also for any site allowing user configuration of the output.

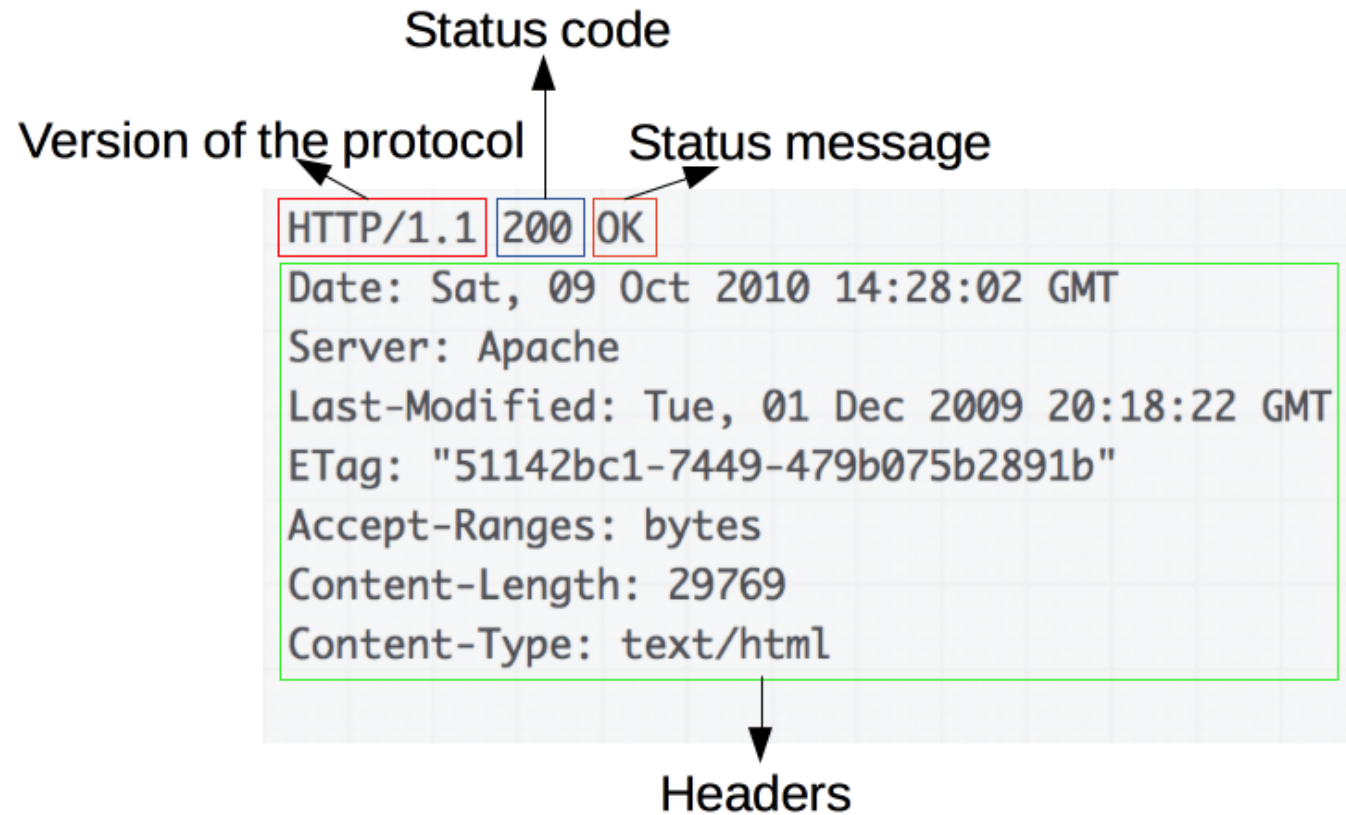
# HTTP Messages:

- HTTP messages (HTTP/1.1 and earlier) are human-readable.
- In HTTP/2, these messages are embedded into a binary structure, a frame, allowing optimizations like compression of headers and multiplexing.
- There are two types of HTTP messages, requests and responses, each with its own format.

# Request:



# Response:



# HTTPS

- HTTPS (Hypertext Transfer Protocol Secure) is a secure version of the HTTP protocol that uses the TLS protocol for encryption and authentication.
- The HTTPS protocol makes it possible for website users to transmit sensitive data such as credit card numbers, banking information, and login credentials securely over the internet.
- HTTPS is especially important for securing online activities such as shopping, banking, and remote work.



# How is HTTPS different from HTTP?

- HTTPS adds **encryption**, **authentication**, and **integrity** to the HTTP protocol:
- Encryption:
  - Because HTTP was originally designed as a clear text protocol, it is vulnerable to eavesdropping and man in the middle attacks.
  - By including TLS encryption, HTTPS prevents data sent over the internet from being intercepted and read by a third party.
- Authentication:
  - HTTPS includes robust authentication via TLS protocol.

- Integrity:
  - Each document (such as a web page, image, or JavaScript file) sent to a browser by an HTTPS web server includes a digital signature that a web browser can use to determine that the document has not been altered by a third party or otherwise corrupted while in transit.
  - The server calculates a cryptographic hash of the document's contents, included with its digital certificate, which the browser can independently calculate to prove that the document's integrity is intact.
- **HTTPS is a much safer protocol for browsing and conducting business on the web than HTTP.**

# Working:

- HTTPS adds encryption to the HTTP protocol by wrapping HTTP inside the TLS protocol so that all messages are encrypted in both directions between two networked computers (e.g. a client and web server).
- Although an eavesdropper can still potentially access IP addresses, port numbers, domain names, the amount of information exchanged, and the duration of a session, all of the actual data exchanged are securely encrypted by TLS, including:
  - Request URL (which web page was requested by the client)
  - Website content
  - Query parameters
  - Headers
  - Cookies

# DNS:

- All computers on the Internet that serve content for massive retail websites, find and communicate with one another by using numbers.
- These numbers are known as IP addresses.
- When you open a web browser and go to a website, you don't have to remember and enter a long number.
- Instead, you can enter a domain name like abc.com.

- A DNS service that translates human readable names like `www.abc.com` into the numeric IP addresses like `192.1.2.1` that computers use to connect to each other.
- The Internet's DNS system works much like a phone book by managing the mapping between names and numbers. DNS servers translate requests for names into IP addresses, controlling which server an end user will reach when they type a domain name into their web browser.
- These requests are called queries.

# Types of DNS:

- Authoritative DNS:
- An authoritative DNS service provides an update mechanism that developers use to manage their public DNS names.
- It then answers DNS queries, translating domain names into IP address so computers can communicate with each other.
- Authoritative DNS has the final authority over a domain and is responsible for providing answers to recursive DNS servers with the IP address information.

- Recursive DNS:
  - Clients typically do not make queries directly to authoritative DNS services.
  - Instead, they generally connect to another type of DNS service known as a resolver, or a recursive DNS service.

# TLS(Transport Layer Security)

- Transport Layer Security is a security protocol.
- It is designed to facilitate privacy and data security for communications over the Internet.
- A primary use case of TLS is encrypting the communication between web applications and servers, such as web browsers loading a website.
- TLS can also be used to encrypt other communications such as email, messaging, and voice over IP (VoIP).



# TLS(Transport Layer Security)

- HTTPS is an implementation of TLS encryption on top of the HTTP protocol, which is used by all websites as well as some other web services.
- Any website that uses HTTPS is therefore employing TLS encryption.
- TLS encryption can help protect web applications from data breaches and other attacks.
- Additionally, TLS-protected HTTPS is quickly becoming a standard practice for websites.

# TLS(Transport Layer Security)

- **Encryption:** hides the data being transferred from third parties.
- **Authentication:** ensures that the parties exchanging information are who they claim to be.
- **Integrity:** verifies that the data has not been forged or tampered with.

# Working:

- For a website or application to use TLS, it must have a TLS certificate installed on its origin server.
- A TLS certificate is issued by a certificate authority to the person or business that owns a domain.
- The certificate contains important information about who owns the domain, along with the server's public key

# Working:

- A TLS connection is initiated using a sequence known as the TLS handshake.
- When a user navigates to a website that uses TLS, the TLS handshake begins between the user's device and the web server.
- During the TLS handshake, the user's device and the web server:
  - Specify which version of TLS (TLS 1.0, 1.2, 1.3, etc.) they will use
  - Decide on which cipher suites they will use
  - Authenticate the identity of the server using the server's TLS certificate
  - Generate session keys for encrypting messages between them after the handshake is complete

- Because of the complex process involved in setting up a TLS connection, some load time and computational power must be expended.
- The client and server must communicate back and forth several times before any data is transmitted.
- It eats up precious milliseconds of load times for web applications, as well as some memory for both the client and the server.

# XML:

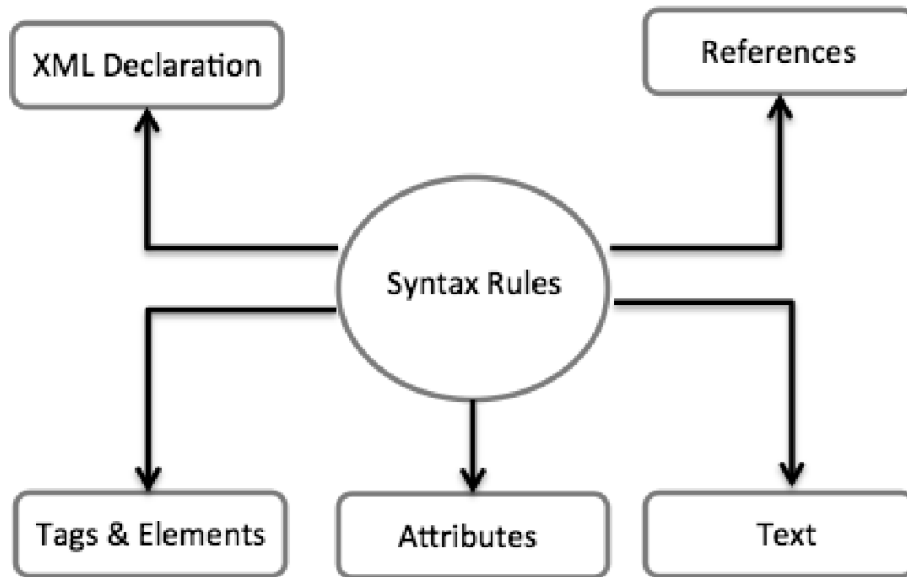
- XML stands for Extensible Markup Language.
- XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data.
- There are three important characteristics of XML
  - XML is extensible: XML allows you to create your own self-descriptive tags or language, that suits your application.
  - XML carries the data, does not present it: XML allows you to store the data irrespective of how it will be presented.
  - XML is a public standard: XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard

# XML Syntax:

- There are two kinds of information in this example:
  - Markup, like<contact-info>
  - The text, or the character data,

```
<?xml version="1.0"?>  
<contact-info>  
  <name> Ayushi shah</name>  
  <college>TSEC</college>  
  <phone>022-123456</phone>  
</contact-info>
```

# XML:



- Components of XML:

1. XML Declaration
2. References
3. Tags
4. Attributes
5. Text



# Components of XML: XML Declaration

- The XML document can optionally have an XML declaration.
- It is written as follows:
- `<?xml version="1.0" encoding="UTF-8"?>`
- Where version is the XML version and encoding specifies the character encoding used in the document.

# Components of XML: Tags and Elements

- An XML file is structured by several XML-elements, also called XML-nodes or XML-tags.
- The names of XML-elements are enclosed in triangular brackets < > as shown below:
- <contact-info>, <phone>
- Every time we should close XML element.
- Each XML-element needs to be closed either with start or with end elements.
- An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap

# Components of XML: Tags and Elements

- `<?xml version="1.0"?>`
- `<contact-info>`
- `<college>TSEC`
- `</contact-info>`
- `</college>`

- `<?xml version="1.0"?>`
- `<contact-info>`
- `<college>TSEC</college>`
- `</contact-info>`

# Components of XML: Tags and Elements

- Root Element:
- An XML document can have only one root element.

```
<x>...</x>  
<y>...</y>
```

```
<root>  
<x>...</x>  
<y>...</y>  
</root>
```

- The names of XML-elements are case-sensitive.
- That means the name of the start and the end elements need to be exactly in the same case.
- For example, **<contact-info>** is different from **<Contact-Info>**.

# Components of XML: Attributes

- An **attribute** specifies a single property for the element, using a name/value pair.
- An XML-element can have one or more attributes.
- Attribute names in XML (unlike HTML) are case sensitive.
- HREF and href are considered two different XML attributes.
- Same attribute cannot have two values in a syntax.
- incorrect syntax because the attribute **b** is specified twice:
- Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks

```
<a b="x" c="y" b="z">....</a>
```

```
<a b=x>....</a>
```

# Components of XML: References

- A **reference** allows you to include additional text or markup in an XML document.
- References always begin with the character “&” (which is specially reserved) and end with the character “;”.
- XML has two kinds of references:
  1. Entity Reference:
    - An entity reference, like “&”, contains a name (in this case, “amp”) between the start and end delimiters.
    - The name refers to a predefined string of text and/or markup

## 2. Character References:

- A character references, like “&#38;”, contains a hash mark (“#”) followed by a number.
- The number always refers to the Unicode code for a single character, such as 65 for the letter “A”.
- XML also provides five pre-declared entities that you can use to escape special characters in an XML document:

Character	Predeclared Entity
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;



# Components of XML: Text

- The names of XML-elements and XML-attributes are case-sensitive, which means the name of start and end elements need to be written in the same case.
- To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files.
- Whitespace characters like blanks, tabs and line-breaks between XML-elements and between the XML-attributes will be ignored.
- Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly.
- To use them, some replacement-entities are used,

# Difference between XML and HTML

HTML	XML
It is used to display the data and control how data is displayed.	XML is used to describe the data and focus on what data is.
HTML tags are predefined	XML tags are not predefined
HTML tags are not case sensitive	Tags are case sensitive.
It is not mandatory to close the tag.	It is mandatory to close the tag.
Stylesheets for HTML are optional.	Stylesheet for XML called XSL are compulsory for formatting of data
HTML is presentation language.	It is neither programming nor presentation language.

# XML Document:

- **Document Prolog** comes at the top of the document, before the root element.
- This section contains:
  - XML declaration
  - Document type declaration

```
<?xml version="1.0"?>
```

```
<contact-info>
```

```
  <name>Tanmay Patil</name>
```

```
  <company>TutorialsPoint</company>
```

```
  <phone>(011) 123-4567</phone>
```

```
</contact-info>
```

Document Prolog

Document Elements

- **XML declaration** contains details that prepare an XML processor to parse the XML document.
- It is optional, but when used, it must appear in the first line of the XML document.

```
<?xml  
version="version_number"  
encoding="encoding_declaration"  
standalone="standalone_status"  
?>
```

Parameter	Parameter value	Parameter Description
Version	1.0	Specifies the version of the XML standard
Encoding	UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift_JIS, EUC-JP	It defines the character encoding used in the document. UTF-8 is the default encoding used.
standalone	Yes/ No	It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to no. Setting it to yes tells the processor there are no external declarations required for parsing the document.

# XML Document:

- **Document Elements** are the building blocks of XML.
- These divide the document into a hierarchy of sections, each serving a specific purpose.
- You can separate a document into multiple sections so that they can be rendered differently, or used by a search engine.
- The elements can be containers, with a combination of text and other elements.

# XML Declaration:

- XML declaration with no parameters:

```
<?xml>
```

- XML declaration with version definition:

```
<?xml version="1.0">
```

- XML declaration with all parameters defined:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

- XML declaration with all parameters defined in single quotes:

```
<?xml version='1.0' encoding='iso-8859-1' standalone='no' ?>
```

# Document Type Documentation:

- The purpose of a DTD is to define the structure of an XML document.
- It defines the structure with a list of legal elements:
- DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.
- An XML DTD can be either specified inside the document, or it can be kept in a separate document.



# Syntax:

- <!DOCTYPE element DTD identifier [
  - declaration1
  - declaration2
  - .....
- ]>

```
|<?xml version="1.0"?>
|<!DOCTYPE note [
|<!ELEMENT note (to,from,heading,body)>
|<!ELEMENT to (#PCDATA)>
|<!ELEMENT from (#PCDATA)>
|<!ELEMENT heading (#PCDATA)>
|<!ELEMENT body (#PCDATA)> ]>
|<note>
|<to>Bhushan</to>
|<from>TSECIT</from>
|<heading>Reminder</heading>
|<body>Lecture on Friday </body>
|</note>
```

# Internal DTD

- A DTD is referred to as an internal DTD if elements are declared within the XML files.
- To refer it as internal DTD, standalone attribute in XML declaration must be set to yes.
- This means, the declaration works independent of an external source.

| Syntax of Internal DTD:

<!DOCTYPE root-element [element-declarations]>

```
|<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
|<!DOCTYPE address
|[
|  <!ELEMENT address (name,company,phone)>
|  <!ELEMENT name (#PCDATA)>
|  <!ELEMENT company (#PCDATA)>
|  <!ELEMENT phone (#PCDATA)>
|]>
|<address>
|  <name>Sujata P.</name>
|  <company>ABC Corp</company>
|  <phone>(011) 123-4567</phone>
|</address>
```

## | Rules

- | The document type declaration must appear at the start of the document (preceded only by the XML header) — it is not permitted anywhere else within the document.
- | Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- | The Name in the document type declaration must match the element type of the root element.

# External DTD

- | In external DTD elements are declared outside the XML file.
- | They are accessed by specifying the system attributes which may be either the legal .dtd file or a valid URL.
- | To refer it as external DTD, standalone attribute in the XML declaration must be set as no.
- | This means, declaration includes information from the external source.

| Syntax;

|       <!DOCTYPE root-element SYSTEM "file-name">  
| where file-name is the file with .dtd extension.

```
|<?xml version="1.0" encoding="UTF-8" standalone="no" ?>  
|<!DOCTYPE address SYSTEM "address.dtd">  
|<address>  
|  <name>Sujata P.</name>  
|  <company>ABC Corp</company>  
|  <phone>(011) 123-4567</phone>  
|</address>
```



| The content of the DTD file address.dtd

```
| <!ELEMENT address (name,company,phone)>  
| <!ELEMENT name (#PCDATA)>  
| <!ELEMENT company (#PCDATA)>  
| <!ELEMENT phone (#PCDATA)>
```

# XSL:

- XSL stands for Extensible Style Sheet Language.
- It is a styling language for XML just like CSS is a styling language for HTML.
- An XSL document specifies how a browser should render an XML document.

## Main parts of XSL Document

- **XSLT:** It is a language for transforming XML documents into various other types of documents.
- **XPath:** It is a language for navigating in XML documents.
- **XQuery:** It is a language for querying XML documents.
- **XSL-FO:** It is a language for formatting XML documents.

# JSON:

- JSON stands for JavaScript Object Notation
- JSON is a lightweight data-interchange format
- JSON is language independent.
- JSON is "self-describing" and easy to understand.
- JSON uses JavaScript syntax, but the JSON format is text only, just like XML.
- Text can be read and used as a data format by any programming language.

# Syntax:

- The JSON syntax is a subset of the JavaScript syntax.
- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:
- Example: "firstname": "Shreya"

# JSON Data Types

- JSON values can be:
  - A number (integer or floating point)
  - E.g. `{"age":20}`
  - A string (in double quotes)
  - E.g. `{"name":"Shreya"}`
  - A Boolean (true or false)
  - E.g. `{"exam":true}`
  - An array (in square brackets)
  - E.g. `{"students":["Aayushi","Mahima","Aashna"]}`
  - An object (in curly braces)
  - E.g. `{"student":{"Rno":1,"Name":"abc","Marks":67}}`
  - Null
  - E.g. `{"middlename":null}`

# Json Objects

- JSON objects are written inside curly braces.
- Just like JavaScript, JSON objects can contain multiple
- name/values pairs:
- e.g.
- `{"firstname":"Shreya","lastname":"Sharma"}`

- e.g.
- {
- "id": 1,
- "name": "A green door",
- "price": 12.50,
- "tags": ["home", "green"]
- }



# Storing JSON Data:

- `var jason = {`
- `"age" : 24,`
- `"hometown" : "Mumbai",`
- `"gender" : "male"`
- `};`

- This creates an object that we access using the variable `jason`.
- By enclosing the variable's value in curly braces, we're indicating that the value is an object.
- Inside the object, we can declare any number of properties using a `"name": "value"` pairing, separated by commas.
- To access the information stored in `jason`, we can simply refer to the name of the property we need.

# JSON Arrays:

- JSON arrays are written inside square brackets.
- Just like JavaScript, a JSON array can contain multiple objects:

- <students>
- <student>
- <firstName>Saurabh</firstName>
- <lastName>Shah</lastName>
- </student>
- <student>
- <firstName>Shreya</firstName>
- <lastName>Sharma</lastName>
- </student>
- <student>
- <firstName>Aanchal</firstName>
- <lastName>Mehta</lastName>
- </student>
- </students>

- The object “students” is an array containing three objects.
- Each object is a record of a student (with a first name and a last name).

```
{“students”:[  
  {“firstName”:“Saurabh”, “lastName”:“Shah”},  
  {“firstName”:“Shreya”, “lastName”:“Sahrma”},  
  {“firstName”:“Aanchal”, “lastName”:“Mehta”}  
]}
```

- The first entry in the JavaScript object array can be accessed like this:
- `students[0].firstName + " " + students[0].lastName;`
- The returned content will be:
- Saurabh Shah
- And the data can be modified like this:
- `students[0].firstName = "Anil";`

- The file type for JSON files is ".json"
- • The MIME type for JSON text is "application/json"

- A common use of JSON is to read data from a web server, and display the data in a web page.
- JSON syntax is a subset of JavaScript syntax.
- The JavaScript function `JSON.parse(text)` can be used to convert a JSON text into a JavaScript object:
- `var obj = JSON.parse(text);`
- It is safer to use a JSON parser to convert a JSON text to a JavaScript object.
- A JSON parser will recognize only JSON text and will not compile scripts.



## JSON.stringify()

- When sending the data to a server the data has to be a string.
- We can convert a JavaScript object into string with JSON.stringify()

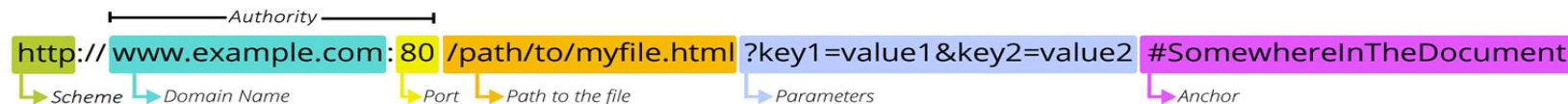
# URL(Uniform Resource Locator)

- URL stands for Uniform Resource Locator.
- A URL is nothing more than the address of a given unique resource on the Web.

- <https://tsec.edu/>
- <https://tsec.edu/it-about-department/it-time-table/>
- [https://tsec.edu/wp-content/uploads/2021/07/S1\\_IT.pdf](https://tsec.edu/wp-content/uploads/2021/07/S1_IT.pdf)

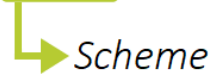
# URL(Uniform Resource Locator)

- Any of those URLs can be typed into browser's address bar to tell it to load the associated page (resource).
- A URL is composed of different parts, some mandatory and others optional.
- The most important parts are highlighted on the URL



# Scheme:


http://www.example.com:80/path/



- The first part of the URL is the scheme, which indicates the protocol that the browser must use to request the resource
- Usually for websites the protocol is HTTPS or HTTP

# Authority:

http://www.example.com:80/path/to/my



- Next follows Authority which is separated from the scheme by the character pattern ://
- If present the authority includes both the domain (e.g. www.example.com) and the port 80 separated by colon.
- The domain indicates which Web server is being requested.
- Usually this is a domain name, but an IP address may also be used
- The port indicates the technical "gate" used to access the resources on the web server.
- It is usually omitted if the web server uses the standard ports of the HTTP protocol (80 for HTTP and 443 for HTTPS) to grant access to its resources.
- Otherwise it is mandatory.

# Path to resources:

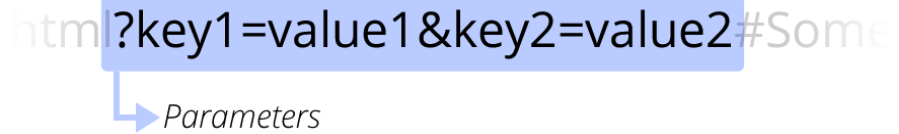
http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2

→ *Path to resource*

- /path/to/myfile.html is the path to the resource on the Web server.
- Nowadays, it is mostly an abstraction handled by Web servers without any physical reality.

# Parameters:

html?key1=value1&key2=value2#Some



Parameters

- ?key1=value1&key2=value2 are extra parameters provided to the web servers.
- Those parameters are a list of key/value pairs separated with the & symbol.
- The Web server can use those parameters to do extra stuff before returning the resource.

# Anchor:

blue2#SomewhereInTheDocument



- is an anchor to another part of the resource itself.
- An anchor represents a sort of "bookmark" inside the resource, giving the browser the directions to show the content located at that "bookmarked" spot.
- On an HTML document, for example, the browser will scroll to the point where the anchor is defined
- On a video or audio document, the browser will try to go to the time the anchor represents.
- It is worth noting that the part after the #, also known as the fragment identifier, is never sent to the server with the request.

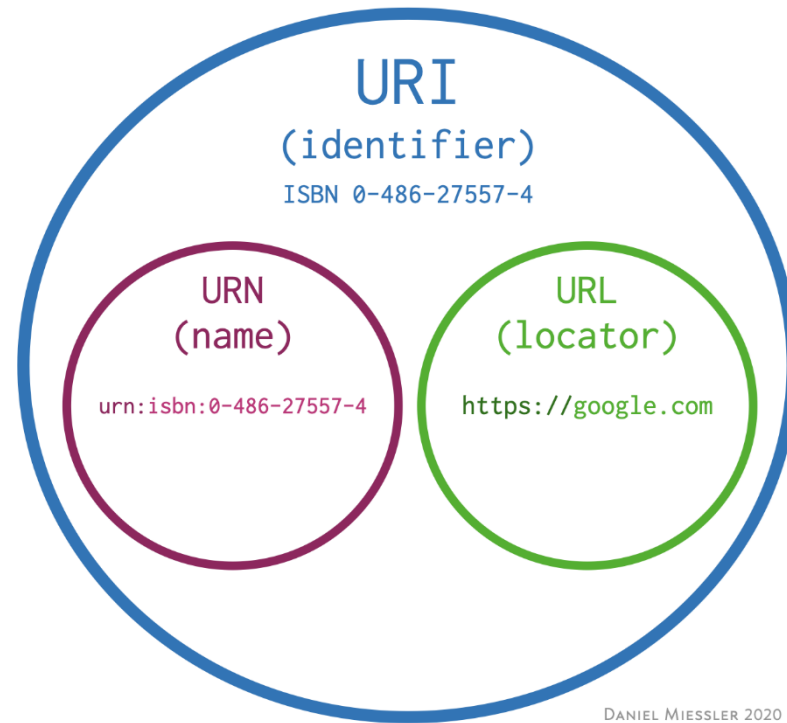


# URI:

- A **Uniform Resource Identifier (URI)** is a unique sequence of characters that identifies a logical or physical resource used by web technologies.
- URIs may be used to identify anything, including real-world objects, such as people and places, concepts, or information resources such as web pages and books.
- Some URIs provide a means of locating and retrieving information resources on a network (either on the Internet or on another private network, such as a computer filesystem or an Intranet);.
- These are Uniform Resource Locators (URLs).

- A URL provides the location of the resource.
- A URI identifies the resource by name at the specified location or URL.
- Other URIs provide only a unique name, without a means of locating or retrieving the resource or information about it, these are Uniform Resource Names (URNs).
- The web technologies that use URIs are not limited to web browsers.

# Difference between URL and URI



- The terms “URI” and “URL” are often used interchangeably, but they are not exactly the same.
- A **URI** is an **identifier** of a specific resource.
- Like a page, or book, or a document.
- A URL is a special type of identifier that also tell you how to access it such as HTTPs and FTP like <https://www.google.com>.
- All URLs are URIs but not all URIs are URLs

# API:

- An API (Application Programming Interface) is a set of features and rules that exist inside a software program (the application) enabling interaction with it through software.
- In Web development, an API is generally a set of code features (e.g. methods, properties, events, and URLs) that a developer can use in their apps for interacting with components of a user's web browser, or other software/hardware on the user's computer, or third party websites and services.

- The Geolocation API can be used to retrieve location information from whatever service the user has available on their device (e.g. GPS), which can then be used in conjunction with the Google Maps APIs.

# REST API:

- A REST API is an API that conforms to the design principles of the REST, or representational state transfer architectural style.
- For this reason, REST APIs are sometimes referred to RESTful APIs.

# Design Principles:

- **Uniform interface.**
- All API requests for the same resource should look the same, no matter where the request comes from.
- The REST API should ensure that the same piece of data, such as the name or email address of a user, belongs to only one uniform resource identifier (URI).
- Resources shouldn't be too large but should contain every piece of information that the client might need.



- **Client-server decoupling.**
- In REST API design, client and server applications must be completely independent of each other.
- The only information the client application should know is the URI of the requested resource; it can't interact with the server application in any other ways.
- Similarly, a server application shouldn't modify the client application other than passing it to the requested data via HTTP.

- **Statelessness.**
- REST APIs are stateless, meaning that each request needs to include all the information necessary for processing it.
- REST APIs do not require any server-side sessions.
- Server applications aren't allowed to store any data related to a client request.

- **Cacheability.**

- When possible, resources should be cacheable on the client or server side.
- Server responses also need to contain information about whether caching is allowed for the delivered resource.
- The goal is to improve performance on the client side, while increasing scalability on the server side.

- **Layered system architecture.**
- In REST APIs, the calls and responses go through different layers.
- There may be a number of different intermediaries in the communication loop.
- REST APIs need to be designed so that neither the client nor the server can tell whether it communicates with the end application or an intermediary.

- **Code on demand (optional).**
- REST APIs usually send static resources, but in certain cases, responses can also contain executable code (such as Java applets).
- In these cases, the code should only run on-demand.

- REST APIs communicate via HTTP requests to perform standard database functions like CRUD within a resource.
- For example, a REST API would use a GET request to retrieve a record, a POST request to create one, a PUT request to update a record, and a DELETE request to delete one.
- All HTTP methods can be used in API calls.
- A well-designed REST API is similar to a website running in a web browser with built-in HTTP functionality.
- The state of a resource at any particular instant, or timestamp, is known as the resource representation. This information can be delivered to a client in virtually any format including JavaScript Object Notation (JSON), HTML, XLT, Python, PHP, or plain text.