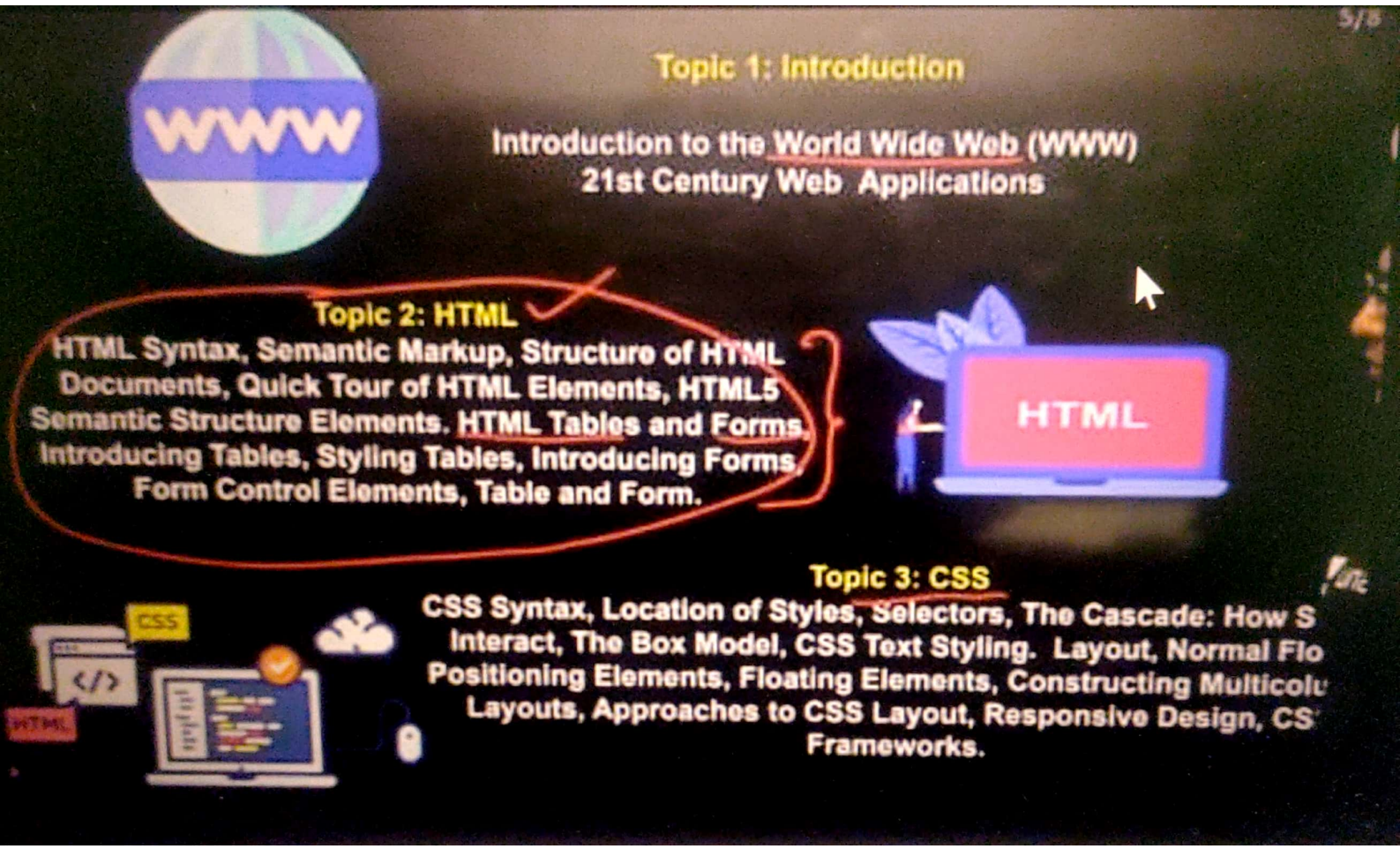
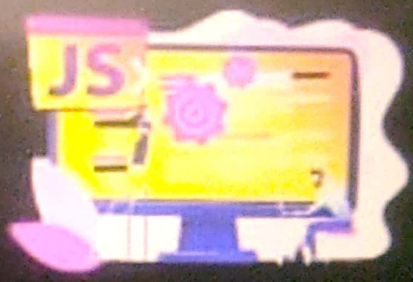


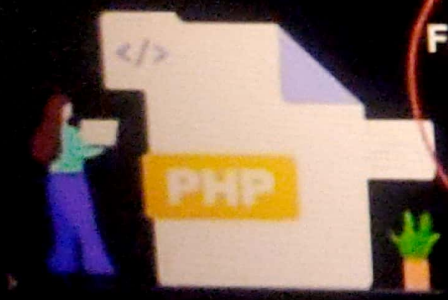
Topic 4: JavaScript

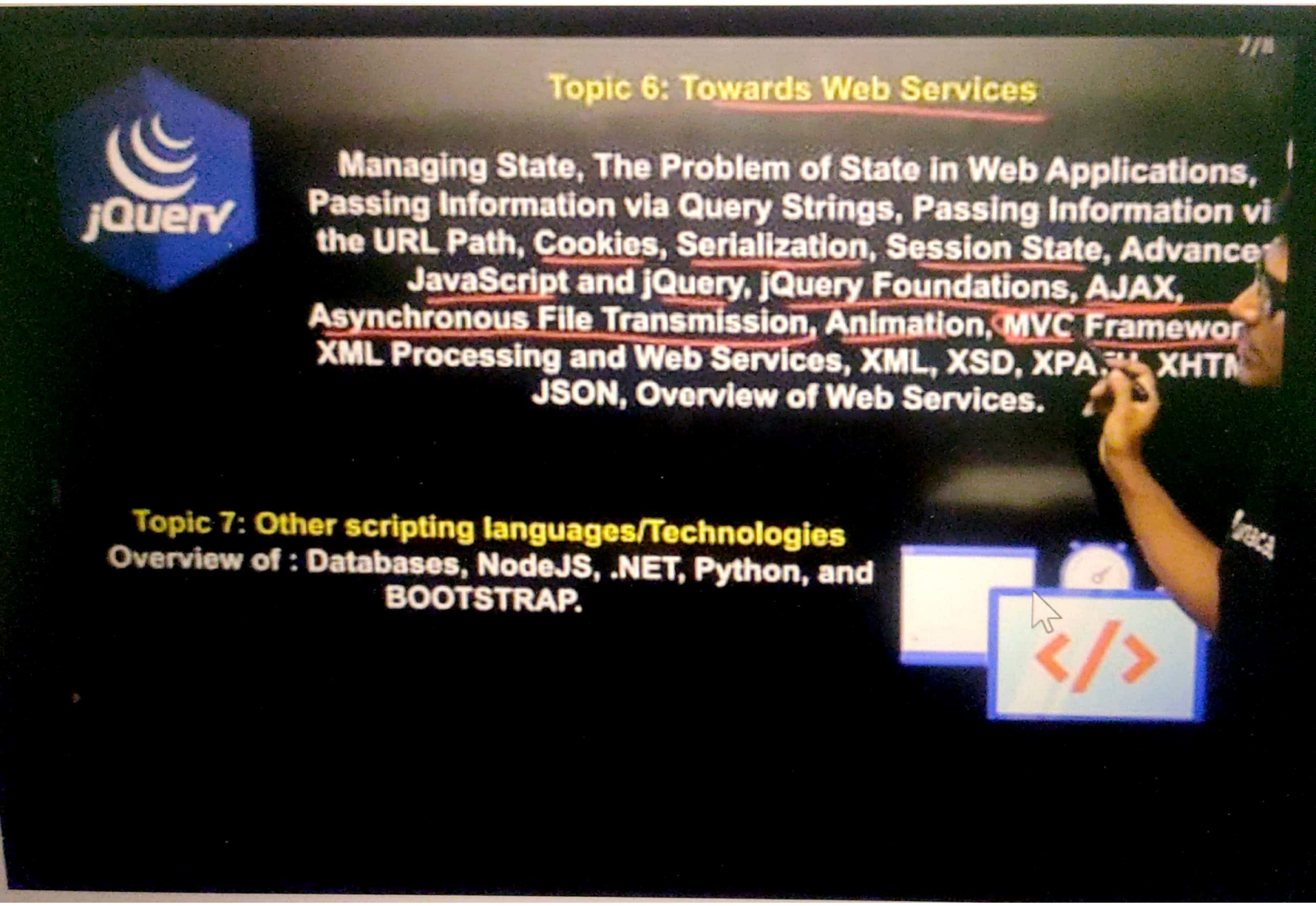
JavaScript Design Principles, Syntax, JavaScript Objects, The Document Object Model (DOM), JavaScript Events, Forms validation, DHTML.



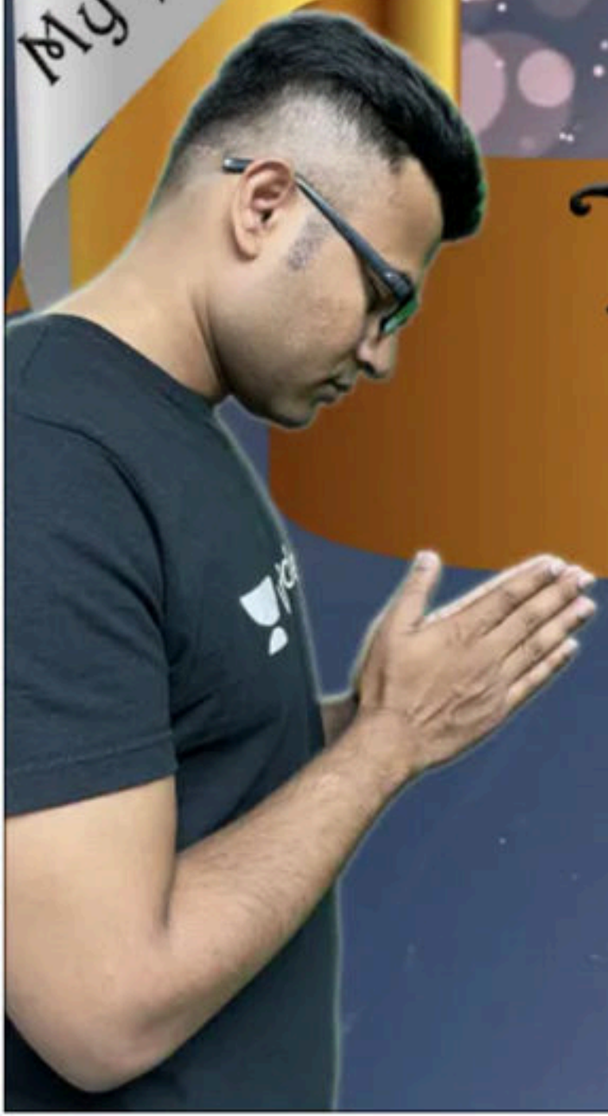
Topic 5: PHP

PHP, Server-Side Development, Program Control, Functions, PHP Arrays and Superglobals, Arrays, $_GET and $_POST Superglobal Arrays, $_SERVER Array, $_Files Array, Reading/Writing Files, PHP Error Reporting, PHP Error and Exception Handling, PHP-MySQL.



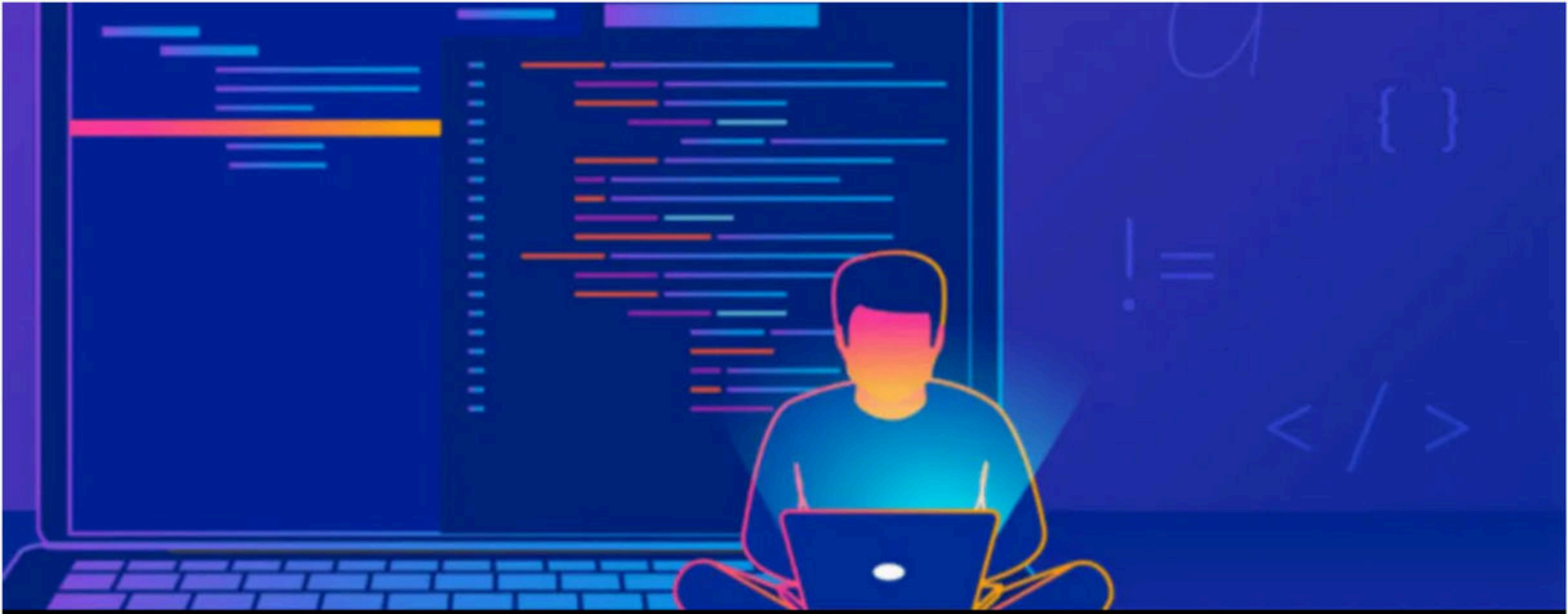


My philosophy

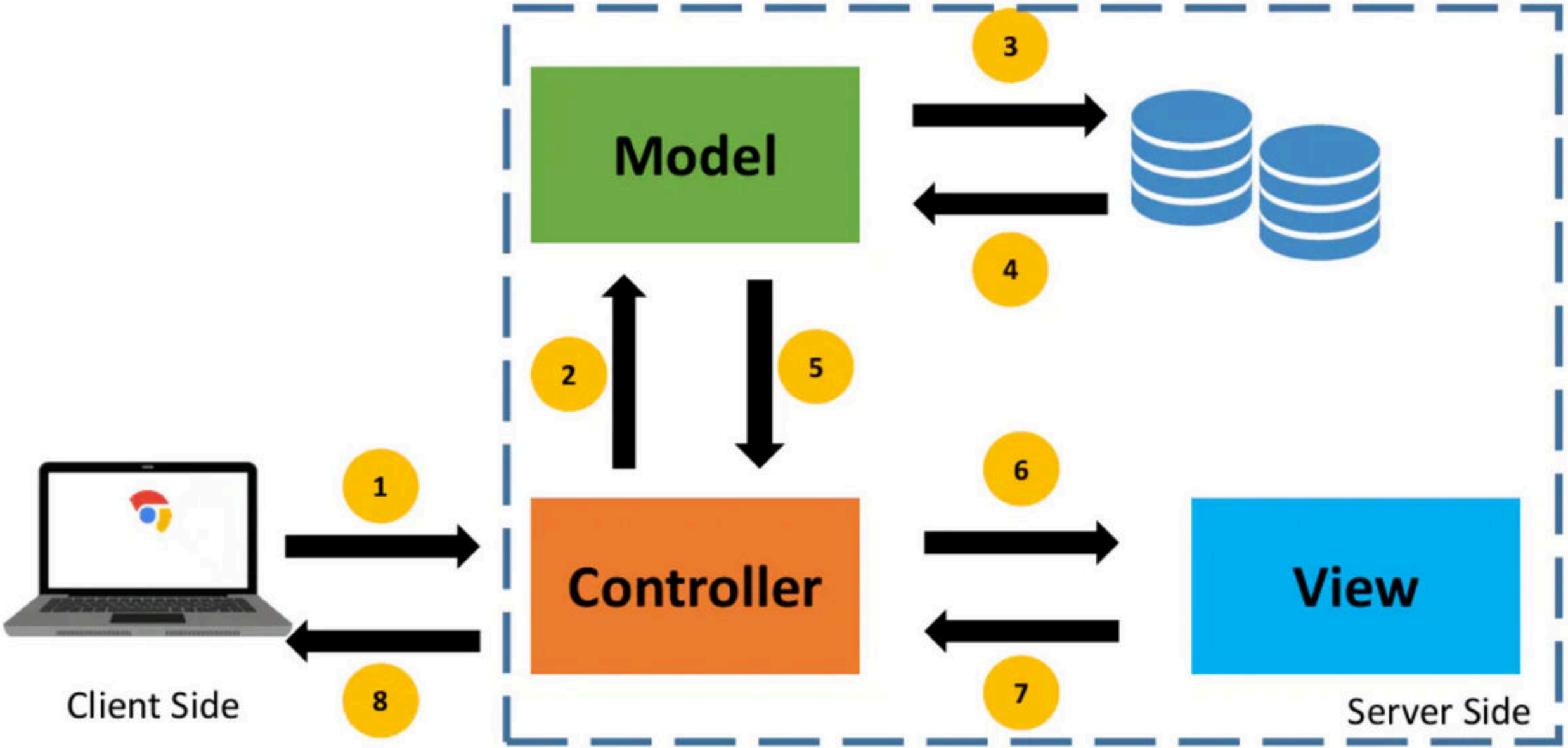


TEACHING IS WORSHIP

STUDENTS ARE GODS



WEB TECHNOLOGIES – Lecture 1
MVC

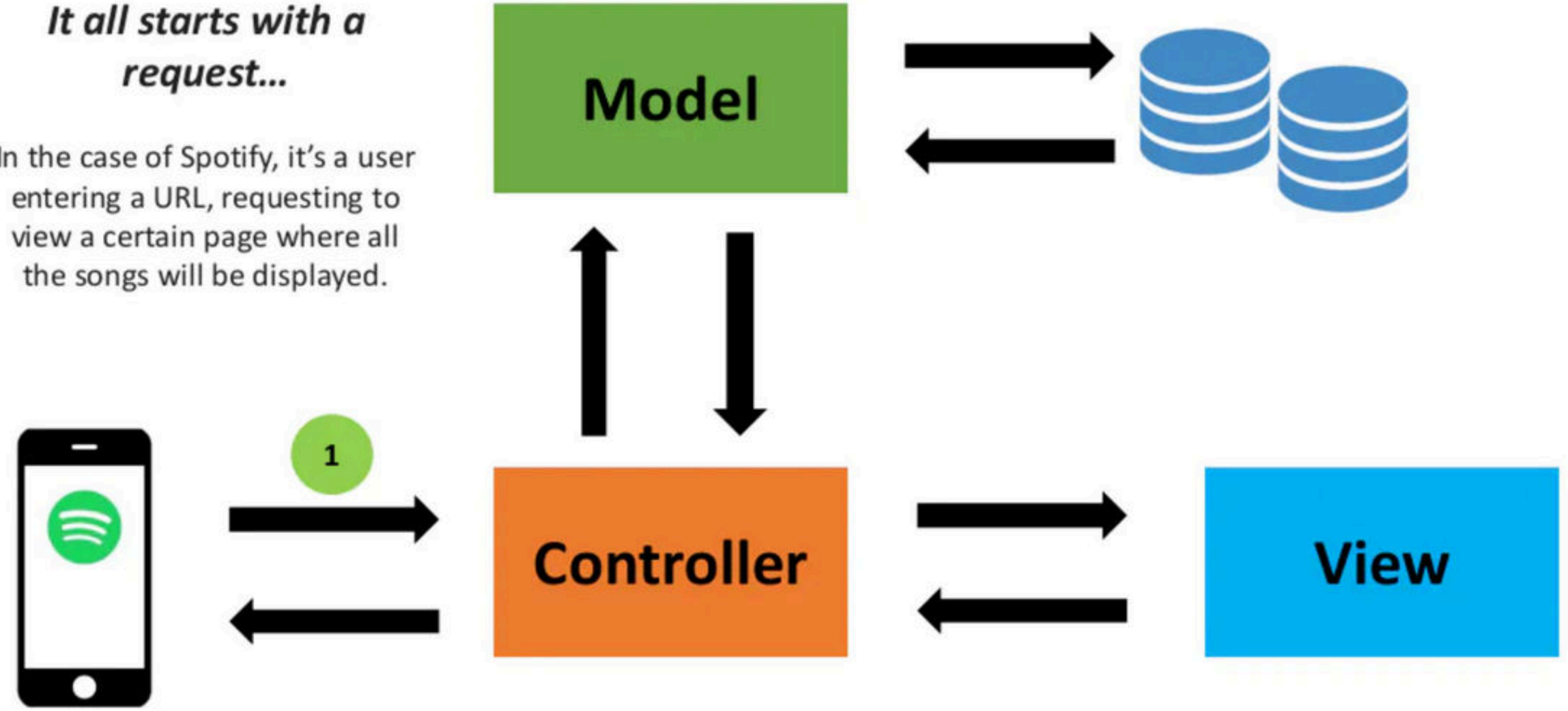

Model
Controller
View
1
2
3
4
5
6
7
8
Client Side
Server Side

Benefits of MVC

- Popular in web applications
- Code is divided based on function to either the model, view, or controller layer.
- The model, view and controller, act independently of each other. Modifications can be done at any layer, without affecting each other.
- Removes unnecessary dependencies
- Reusable without modification
- Easier to maintain or modify
- Supports Multiple views, each part can be tested independently (Model, view, controller).

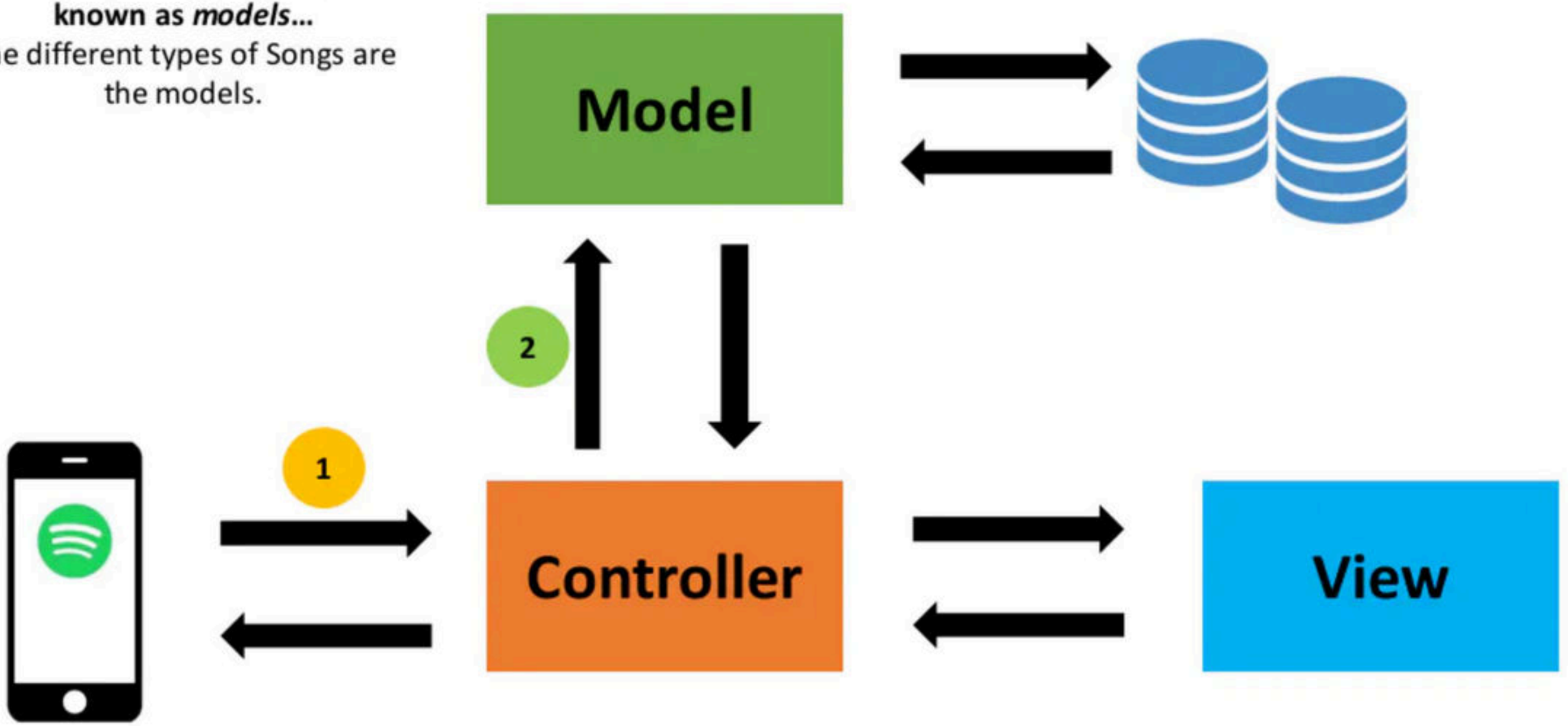
It all starts with a request...

In the case of Spotify, it's a user entering a URL, requesting to view a certain page where all the songs will be displayed.



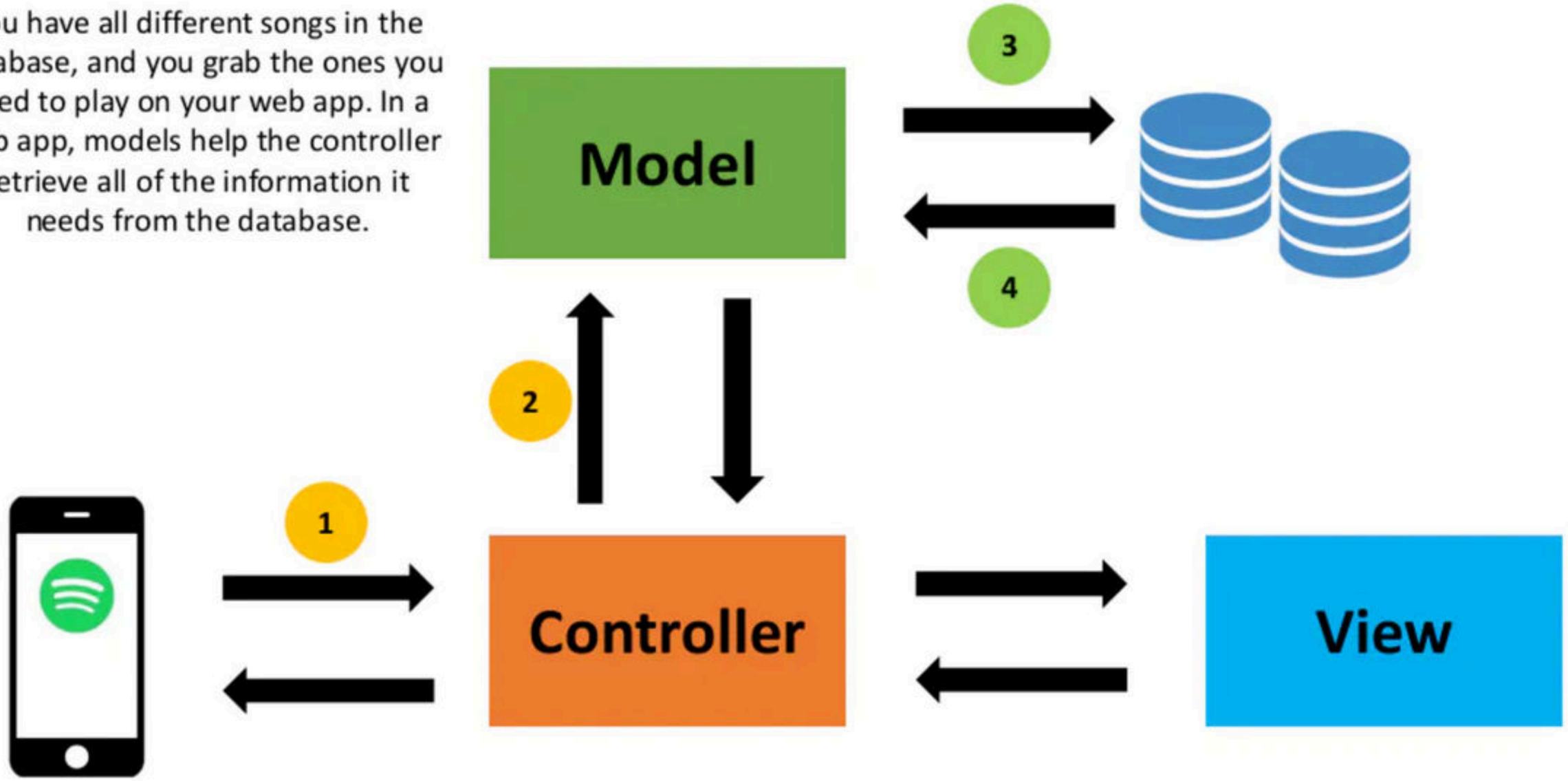
Let's try to visualize all of this by taking an example of your favorite music streaming website such as **Spotify**.

Those building blocks are known as *models*...
The different types of Songs are the models.



Let's try to visualize all of this by taking an example of your favorite music streaming website such as **Spotify**.

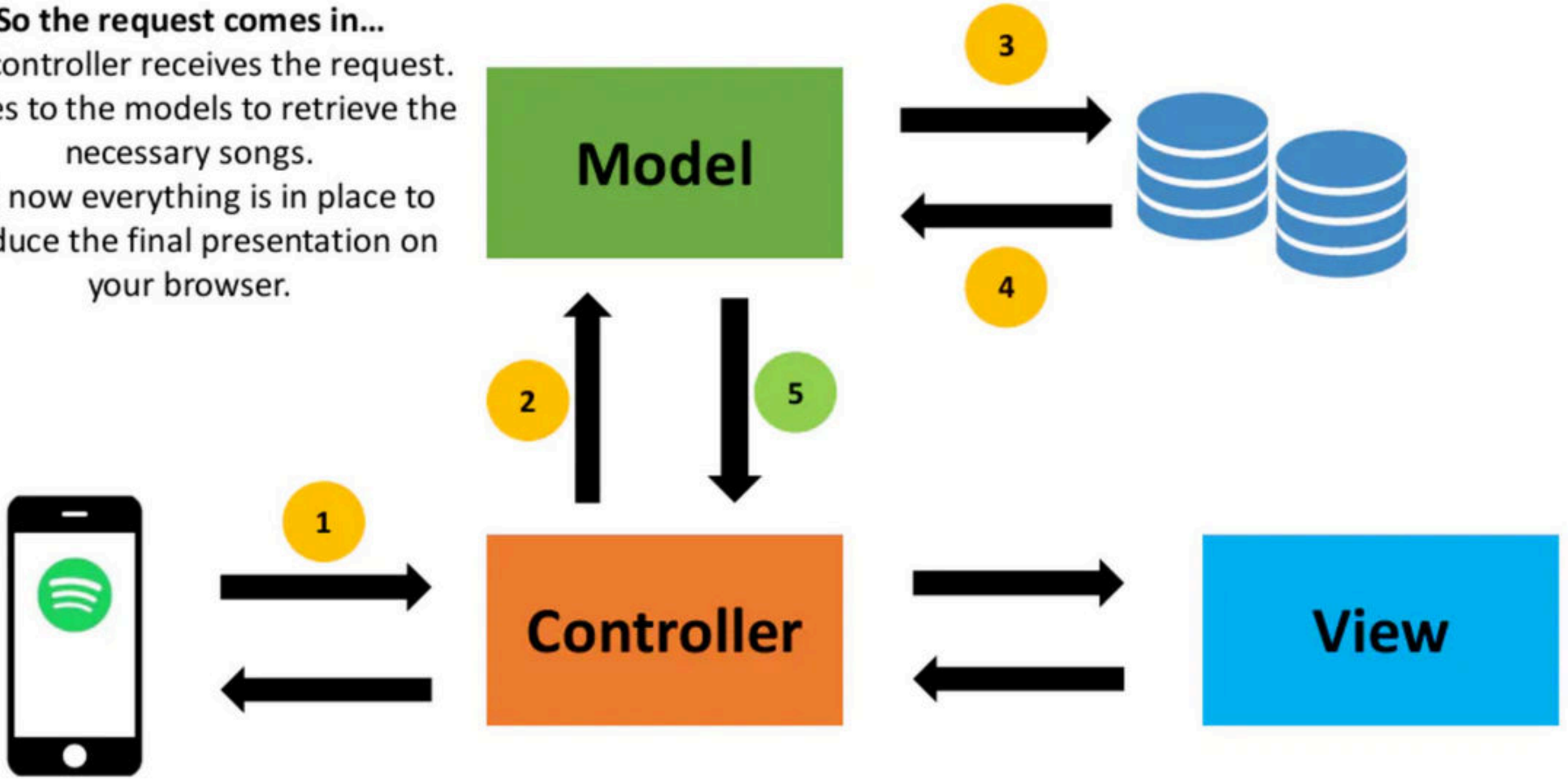
You have all different songs in the database, and you grab the ones you need to play on your web app. In a web app, models help the controller retrieve all of the information it needs from the database.



Let's try to visualize all of this by taking an example of your favorite music streaming website such as **Spotify**.

So the request comes in...

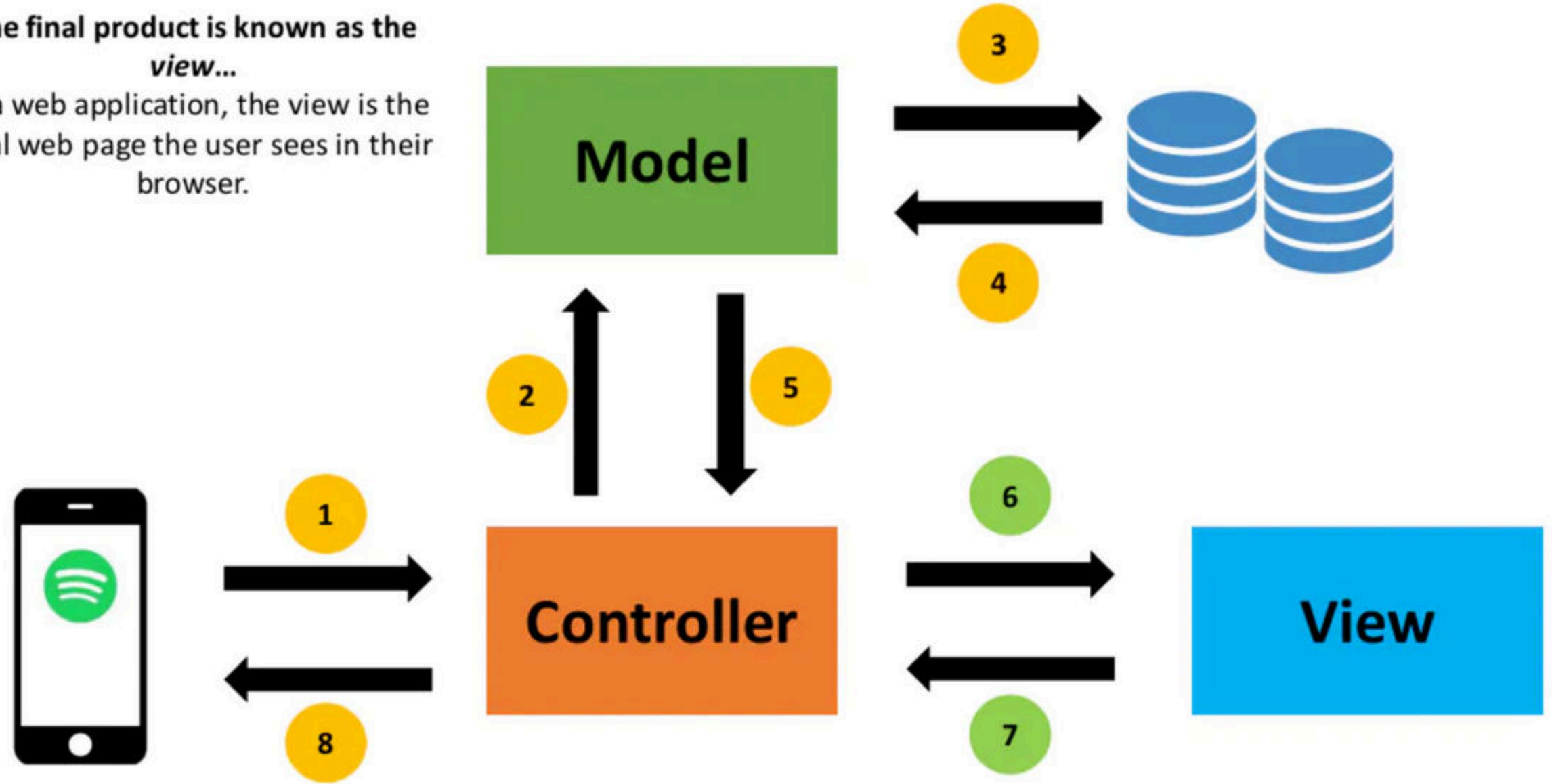
The controller receives the request.
It goes to the models to retrieve the necessary songs.
And now everything is in place to produce the final presentation on your browser.



Let's try to visualize all of this by taking an example of your favorite music streaming website such as **Spotify**.

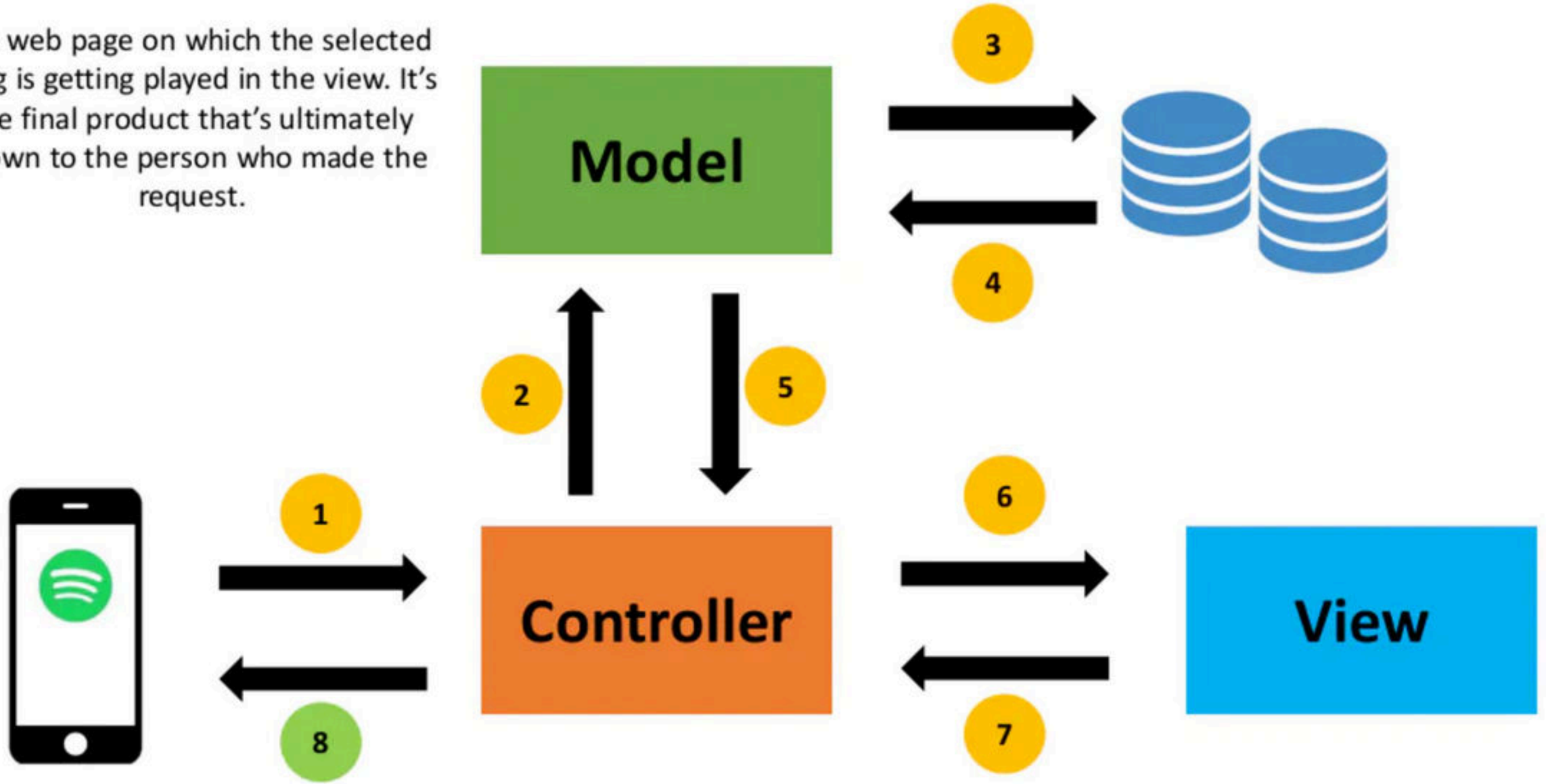
The final product is known as the *view*...

In a web application, the view is the final web page the user sees in their browser.



Let's try to visualize all of this by taking an example of your favorite music streaming website such as **Spotify**.

The web page on which the selected song is getting played in the view. It's the final product that's ultimately shown to the person who made the request.



Let's try to visualize all of this by taking an example of your favorite music streaming website such as **Spotify**.

Small Example in Python Flask to implement a basic MVC :

Models and Controllers

Within the controller action, two main things typically occur: the models are used to retrieve all of the necessary data from a database; and that data is passed to a view, which renders the requested page.

The data retrieved via the models are generally added to a data structure (like a list or dictionary), and that structure is what's sent to the view.

```
@app.route('/')
def main_page():
    """Searches the database for songs, then displays them."""
    model = get_model()
    songs = model.getAllSongs()

    """Data from the model is passed to the view layer"""
    return render_template('index.html', songs)
```

Views

Finally, in the view, that structure of data is accessed and the information contained within is used to render the HTML content of the page the user ultimately sees in their browser.

```
{% for song in songs %}
  <li>
    <h2>{{ song.title }}</h2>
    <div>{{ song.text|safe }}</div>
  </li>
{% else %}
  <li><em>No songs yet. Add some!</em></li>
{% endfor %}
```

This is what is actually returned to the browser. A view is an HTML template that will be binding and displaying HTML controls with data

Popular Frameworks which can be used to implement MVC Pattern in web-apps :
Java: Spring MVC
JavaScript: Angular.js, BackBone.js
Python: Flask, Django

Popular Technology Stacks For Web Application Development

L A M P

Linux

Apache

My SQL

PHP

Pearl

Python

LAMP

LAMP refers to Linux (Kernel), Apache (web server), MySQL (database), PHP (programming language). It is one of the most popular back-end tech stacks for web development.

Full-stack developers prefer LAMP to other tech stacks for its security, flexibility, and customization options.

MEAN STACK

Mango DB

(Database System)

express

Express JS

(Back-end web framework)

Angular JS

(Front-end web framework)

Node JS

(Back-end runtime environment)

MEAN stack

- MEAN stands for MongoDB (database), Express.js (application framework), Angular (front-end framework), and Node.js (runtime environment).
- It is used for full-stack development of web applications.
- It uses JavaScript as the programming language for both front-end and back-end development.
- Leading web development companies in India, USA, and other countries use MEAN for building high performing, scalable web applications.

MVC various technologies at each layer

VIEW	CONTROLLER	MODEL
		

Client Side

URL (Uniform Resource Locator)

Effectively serves as the page's worldwide name.

URLs have three parts:

1. The protocol (also known as the scheme)
2. The DNS name of the machine on which the page is located
3. The path uniquely indicating the specific page (a file to read or program to run on the machine)

https://www.google.com/imghp

This URL consists of three parts:-

- Protocol (*http/https*)
- DNS name of the host (www.google.com)
- Path name (*index.html*).

https://www.google.com/imghp

When a user clicks on a hyperlink, the browser carries out a series of steps in order to fetch the page pointed to.
Let us trace the steps that occur when our example link is selected……

https://www.google.com/imghp

Step 1:
The browser determines the URL (by seeing what was selected).

https://www.google.com/imghp

Step 2:

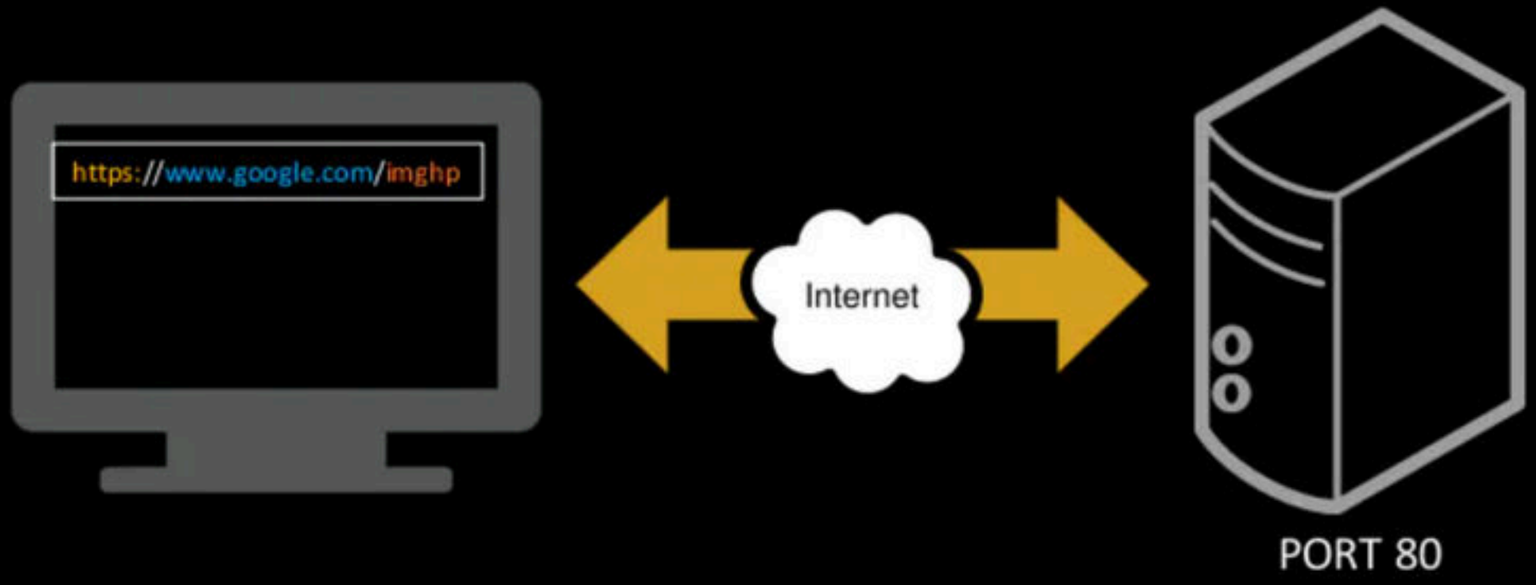
The browser asks DNS for the IP address of the server *https://www.google.com/imghp*.

https://www.google.com/imghp

Step 3:
DNS replies with 216.58.197.46.

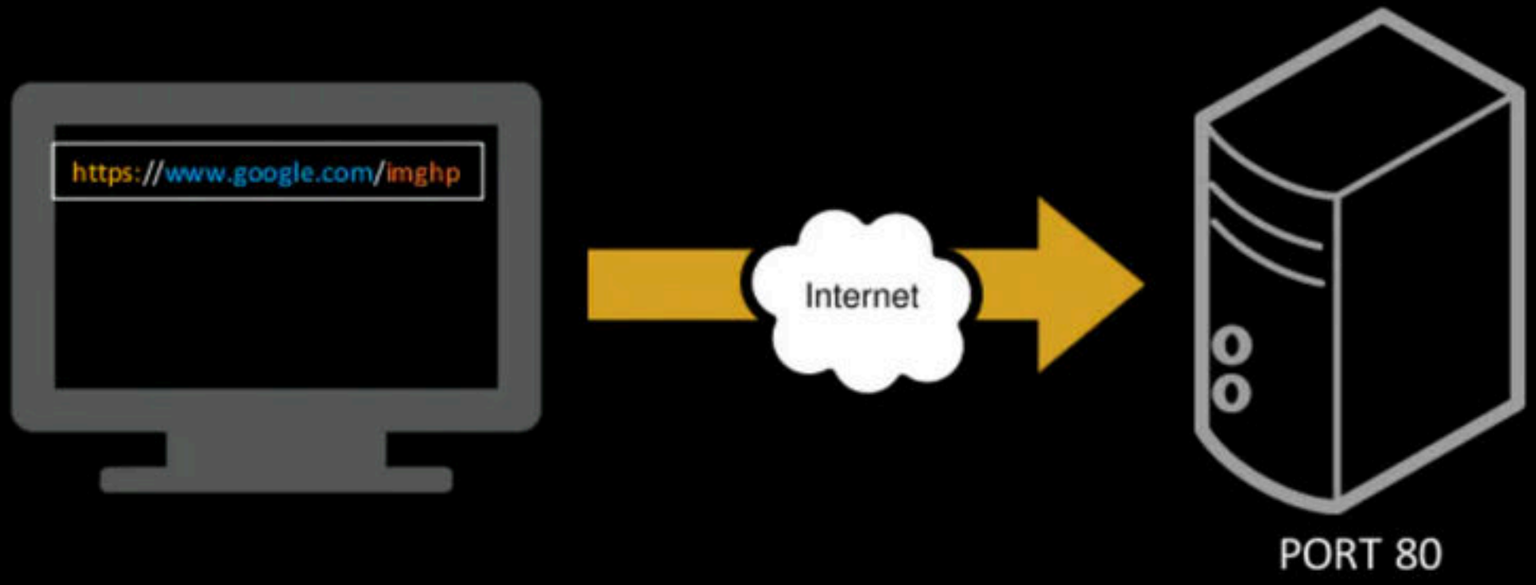
https://www.google.com/imghp

Step 4: The browser makes a TCP connection to 216.58.197.46 on port 80, the well-known port for the HTTP protocol.



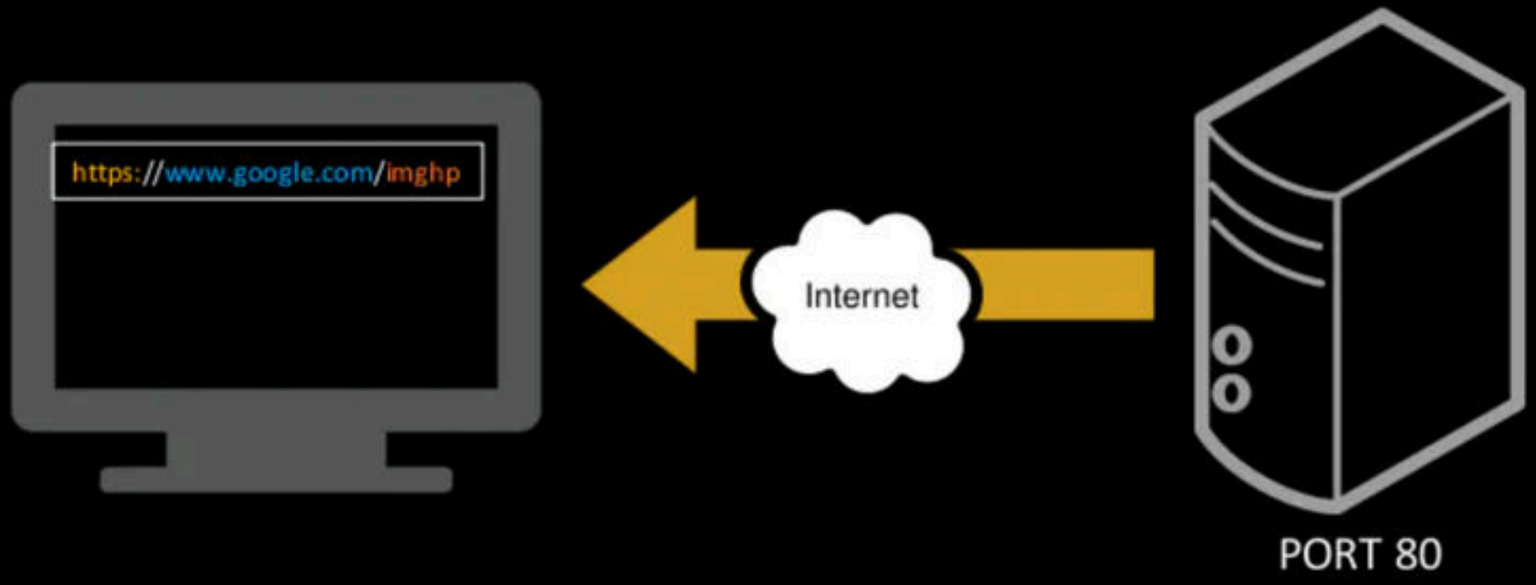
https://www.google.com/imghp

Step 5: It sends over an HTTP request asking for the page /imghp.



https://www.google.com/imghp

Step 6: The www.google.com server sends the page as an HTTP response, for example, by sending the file /imghp.html.

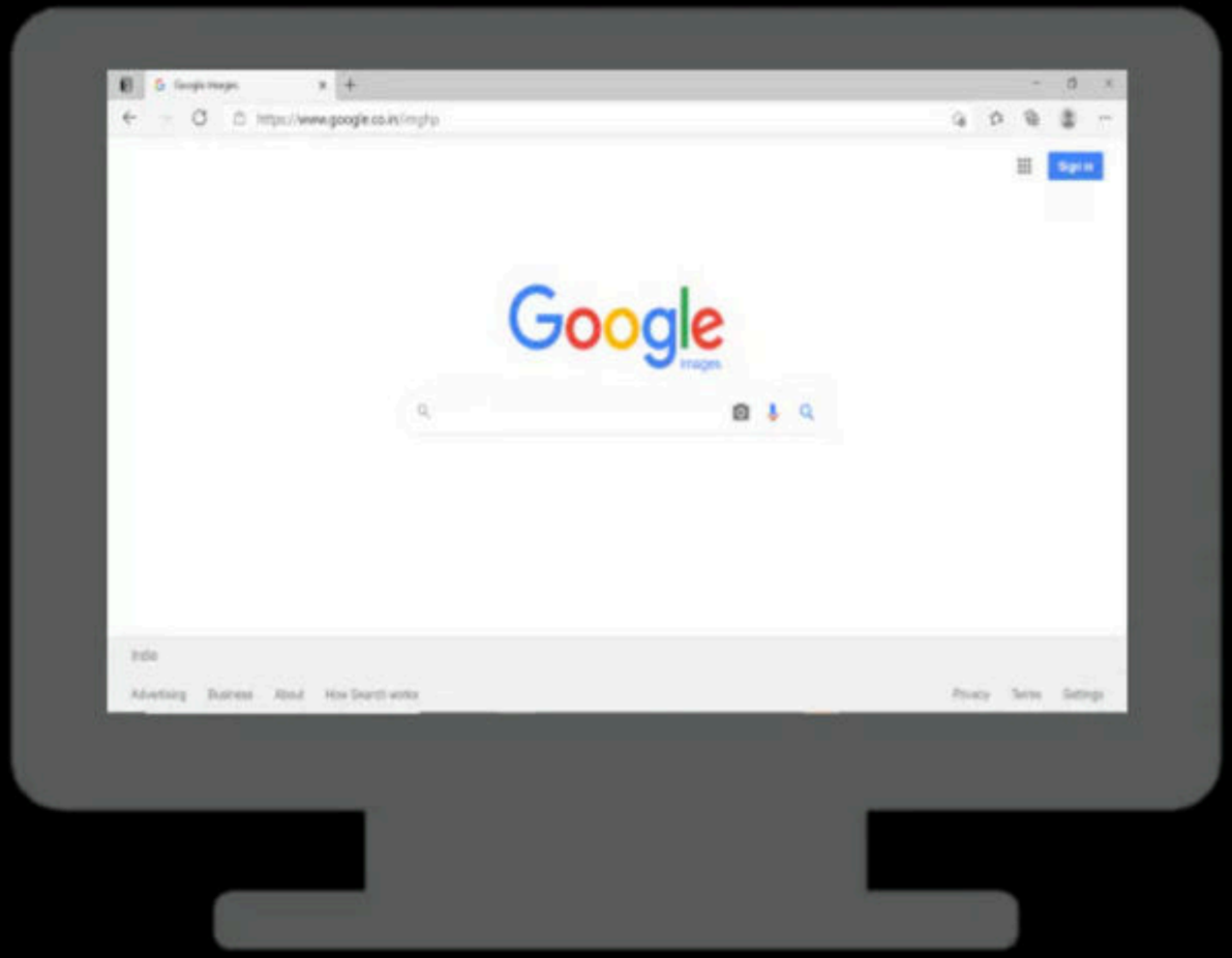


https://www.google.com/imghp

Step 7 : If the page includes URLs that are needed for display, the browser fetches the other URLs using the same process.

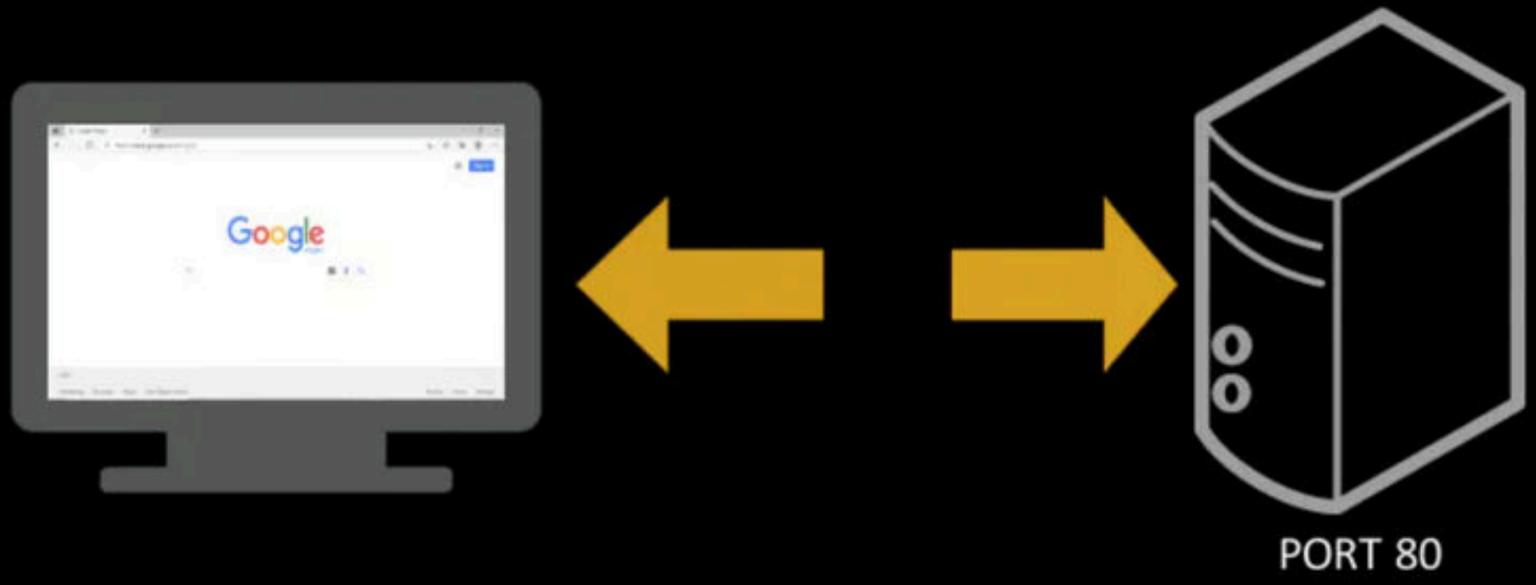
https://www.google.com/imghp

Step 8 : The browser displays the page /imghp.html



https://www.google.com/imghp

Step 9 : The TCP connections are released if there are no other requests to the same servers for a short period.



Many browsers display which step they are currently executing in a status line at the bottom of the screen. In this way, when the performance is poor, the user can see if it is due to DNS not responding, a server not responding, or simply page transmission over a slow or congested network

MIME

IANA is responsible for MIME types.

To be able to display the new page (or any page), the browser has to understand its format. To allow all browsers to understand all Web pages, Web pages are written in a standardized language called HTML

Browser is basically an HTML interpreter
A page may consist of a video in MPEG format, a document in PDF format, a photograph in JPEG format, a song in MP3 format, or any one of hundreds of other file types.
Since standard HTML pages may link to any of these, the browser has a problem when it hits a page it does not know how to interpret.

Rather than making the browsers larger and larger by building in interpreters for a rapidly growing collection of file types, most browsers have chosen a more general solution.

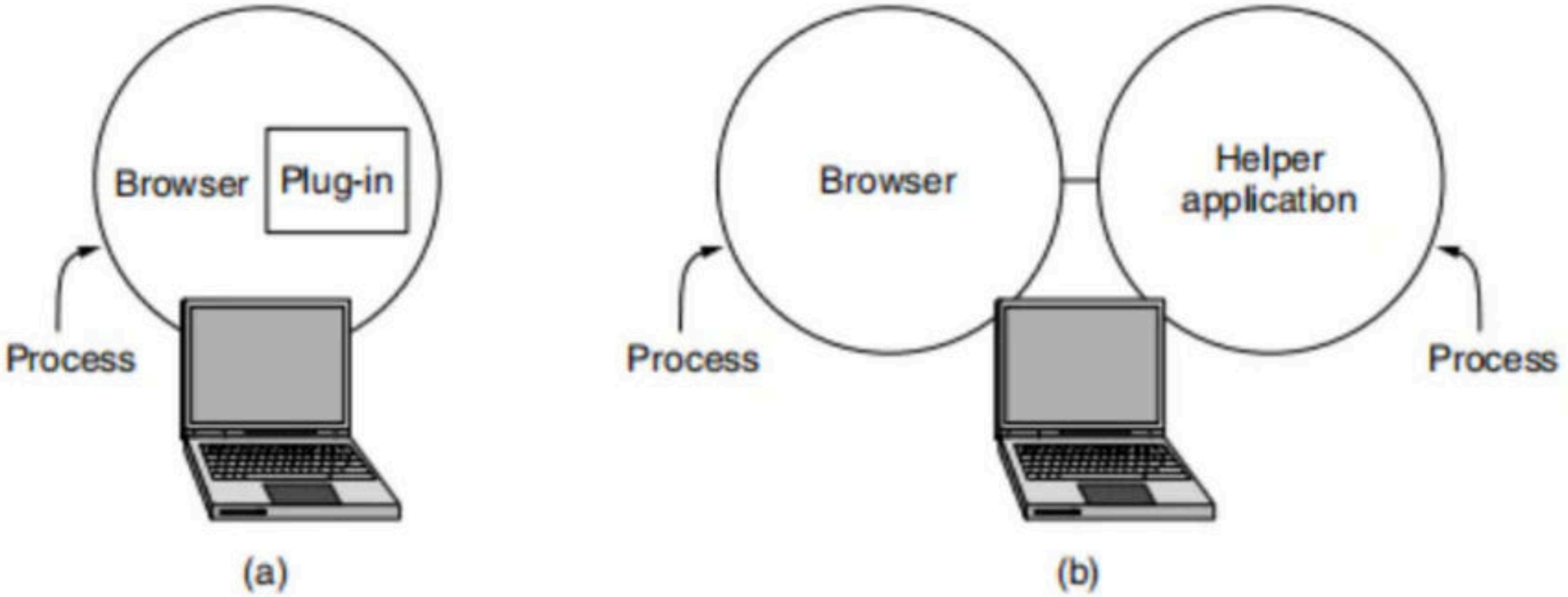
When a server returns a page, it also returns some additional information about the page. This information includes the MIME type of the page.

Pages of type text/html are just displayed directly, as are pages in a few other built-in types. If the MIME type is not one of the built-in ones, the browser consults its table of MIME types to determine how to display the page. This table associates MIME types with viewers.

There are two possibilities:-
plug-ins and helper applications.
A plug-in is a third-party code module that is installed as an extension to the browser.

Common examples are plug-ins for PDF, Flash, and Quicktime to render documents and play audio and video.

Because plug-ins run inside the browser, they have access to the current page and can modify its appearance.


Browser
Plug-in
Process
(a)
Browser
Helper application
Process
Process
(b)

Each browser has a set of procedures that all plug-ins must implement so the browser can call the plug-ins.

For example, there is typically a procedure the browser's base code calls to supply the plug-in with data to display.

This set of procedures is the plug-in's interface and is browser specific.

Before a plug-in can be used, it must be installed.

The usual installation procedure is for the user to go to the plug-in's Web site and download an installation file.

Executing the installation file unpacks the plug-in and makes the appropriate calls to register the plug-in's MIME type with the browser and associate the plug-in with it.

Browsers usually come preloaded with popular plug-ins

The other way to extend a browser is make use of a helper application.
This is a complete program, running as a separate process.
Since the helper is a separate program, the interface is at arm's length from the browser.
It usually just accepts the name of a scratch file where the content file has been stored, opens the file, and displays the contents.

Typically, helpers are large programs that exist independently of the browser,
for example, Microsoft Word or PowerPoint.

Many helper applications use the MIME type application.
As a consequence, a considerable number of subtypes have been defined for them to use, for example, application/vnd.ms-powerpoint for PowerPoint files. vnd denotes vendor-specific formats.

In this way, a URL can point directly to a PowerPoint file, and when the user clicks on it, PowerPoint is automatically started and handed the content to be displayed.
Adobe Photoshop uses image/x-photoshop, for example

A source of conflicts is that multiple plug-ins and helper applications are available for some subtypes, such as *video/mpeg*.

What happens is that the last one to register overwrites the existing association with the MIME type, capturing the type for itself. As a consequence, installing a new program may change the way a browser handles existing types.

Virus

All a malicious Web site has to do is produce a Web page with pictures of, say, movie stars or sports heroes, all of which are linked to a virus. A single click on a picture then causes an unknown and potentially hostile executable program to be fetched and run on the user's machine.

To prevent unwanted guests like this, Firefox and other browsers come configured to be cautious about running unknown programs automatically, but not all users understand what choices are safe rather than convenient.

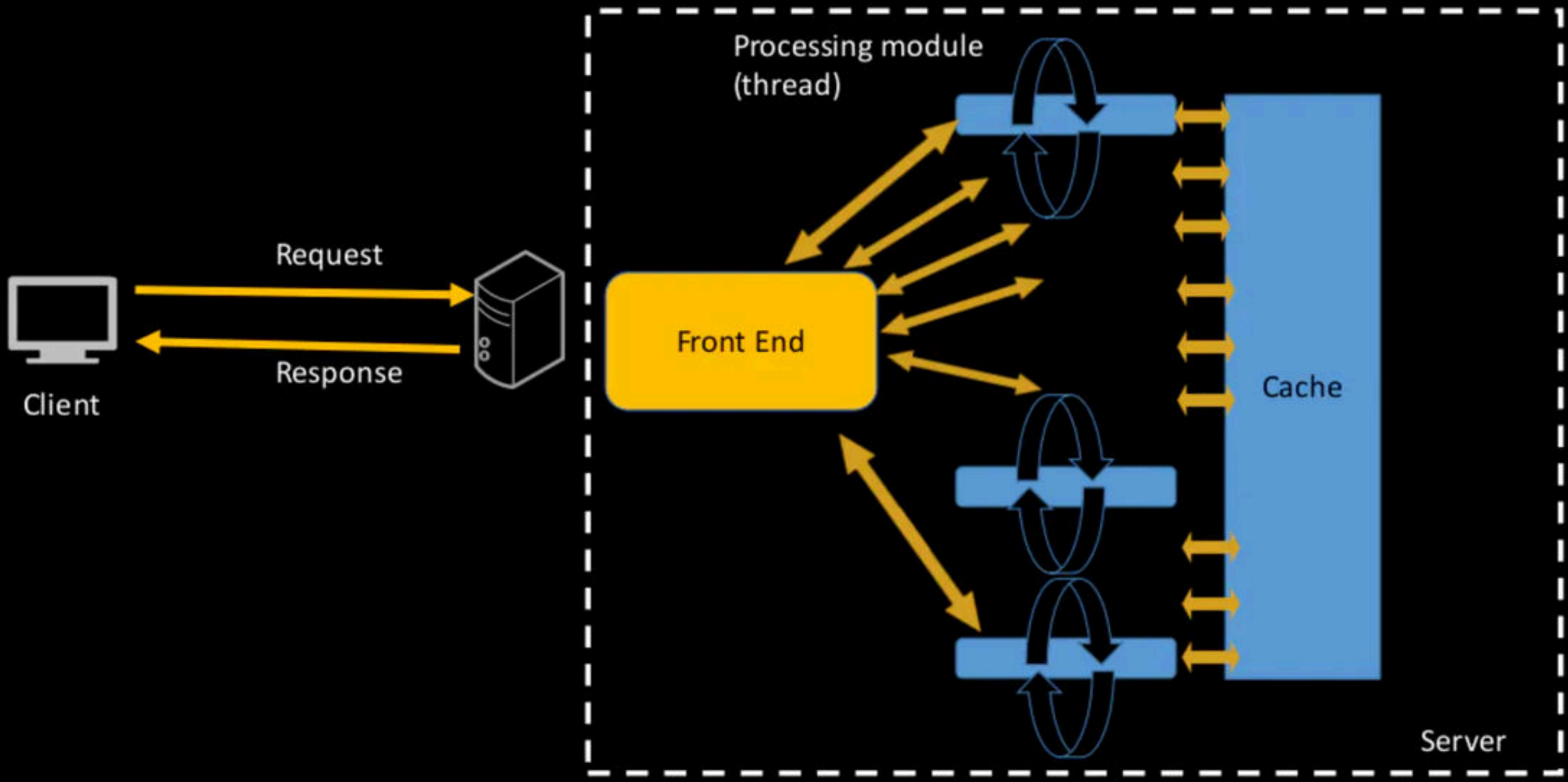
Server Side

1. When the user types in a URL or clicks on a line of hypertext, the browser parses the URL and interprets the part between http:// and the next slash as a DNS name to look up.
2. Armed with the IP address of the server, the browser establishes a TCP connection to port 80 on that server.
3. Then it sends over a command containing the rest of the URL, which is the path to the page on that server.
4. The server then returns the page for the browser to display.

The steps that the server performs in its main loop are:

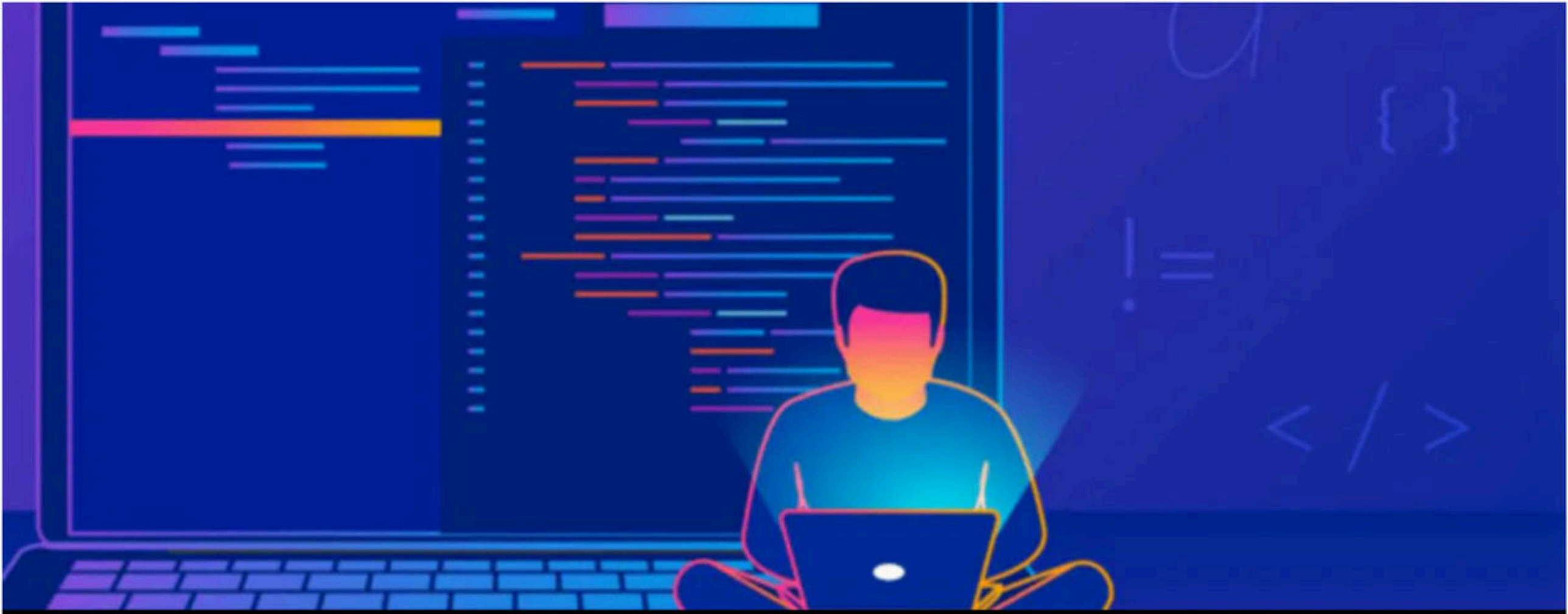
1. Accept a TCP connection from a client (a browser).
2. Get the path to the page, which is the name of the file requested.
3. Get the file (from disk).
4. Send the contents of the file to the client.
5. Release the TCP connection

For dynamic content, the third step may be replaced by the execution of a program (determined from the path) that returns the contents.



These steps occur after the TCP connection and any secure transport mechanism have been established.

1. Resolve the name of the Web page requested.
2. Perform access control on the Web page.
3. Check the cache.
4. Fetch the requested page from disk or run a program to build it.
5. Determine the rest of the response (e.g., the MIME type to send).
6. Return the response to the client.
7. Make an entry in the server log



WEB TECHNOLOGIES

Cookies,Web Tracking, Google ADS

Problem:

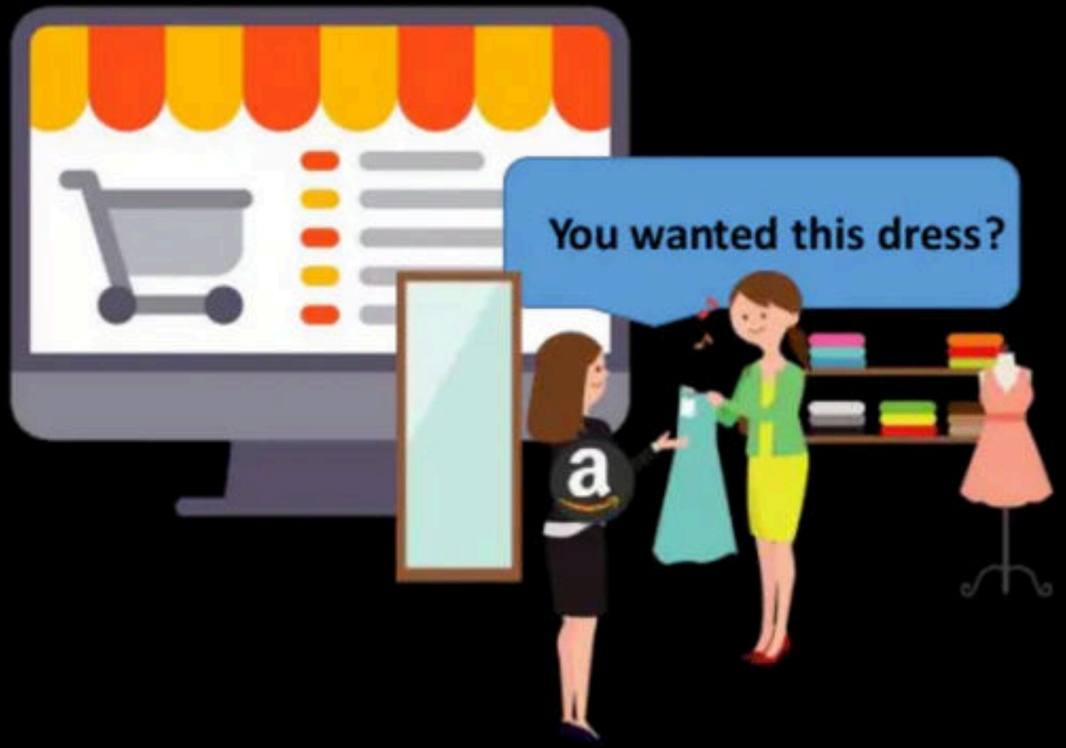
WWW was originally designed as stateless entity.

Client sends a request and server responds and their relationship is over!

But now many websites want to know about the history of the client even without having a login session because they want to send the web pages based on user.

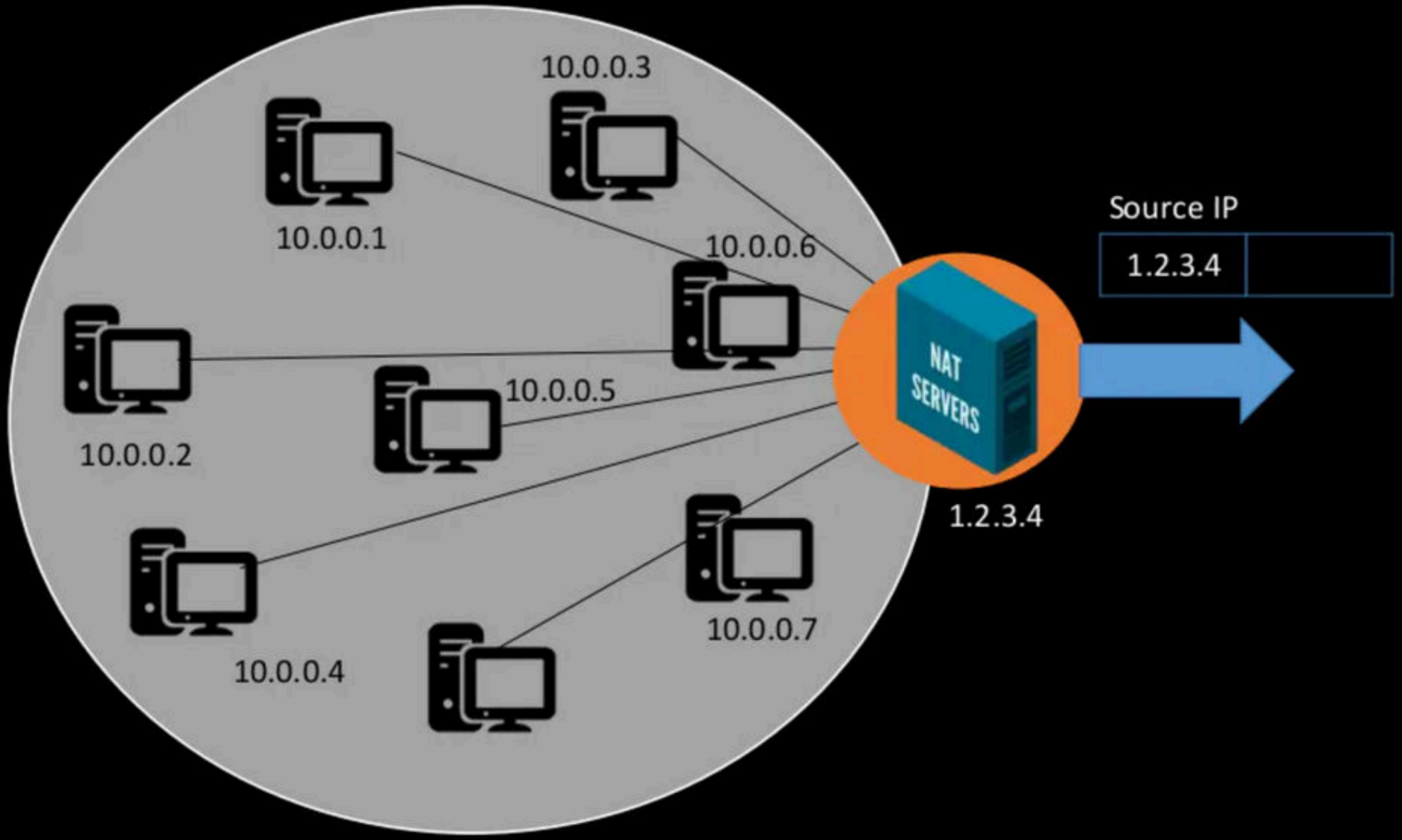
Ex. E-commerce websites

Without maintaining the state,
how can a web browser know the history?



Solution 1 : Use IP address

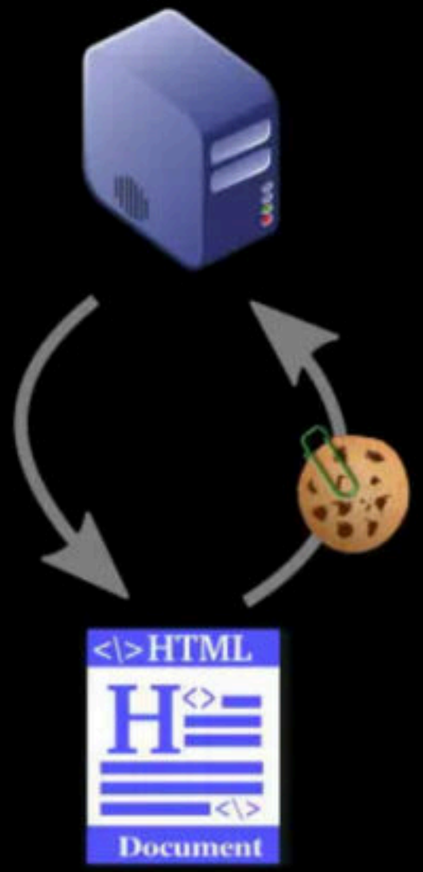
NAT – Network Address Translation



Solution 2 : Cookies

What is a Cookie?

- When a client requests a Web page, the server can supply additional information in the form of a cookie along with the requested page.
- The cookie is a rather small, named string (of at most 4 KB) that the server can associate with a browser.



A cookie may contain up to five fields:

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Yes
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	No
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-12-20 23:59	No
sneaky.com	/	UserID=4627239101	31-12-19 23:59	No

Domain

- The Domain tells where the cookie came from.
- Browsers are supposed to check that servers are not lying about their domain.
- Each domain should store no more than 20 cookies per client.

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Yes
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	No
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-12-20 23:59	No
sneaky.com	/	UserID=4627239101	31-12-19 23:59	No

Path

- The Path is a path in the server's directory structure that identifies which parts of the server's file tree may use the cookie.
- It is often /, which means the whole tree.

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Yes
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	No
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-12-20 23:59	No
sneaky.com	/	UserID=4627239101	31-12-19 23:59	No

Content

- The Content field takes the form name = value.
- Both name and value can be anything the server wants. This field is where the cookie's content is stored.

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Yes
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	No
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-12-20 23:59	No
sneaky.com	/	UserID=4627239101	31-12-19 23:59	No

Expires - Persistent and Non Persistent Cookies

The Expires field specifies when the cookie expires.
If this field is absent, the browser discards the cookie when it exits.
Such a cookie is called a **non persistent cookie**.

If a time and date are supplied, the cookie is said to be a **persistent cookie** and is kept until it expires.

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Yes
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	No
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-12-20 23:59	No
sneaky.com	/	UserID=4627239101	31-12-19 23:59	No

How Server Removes a Cookie???

Expiration times are given in Greenwich Mean Time. To remove a cookie from a client’s hard disk, a server just sends it again, but with an expiration time in the past.

Are Cookies Encrypted during transmission???

Secure

The Secure field can be set to indicate that the browser may only return the cookie to a server using a secure transport, namely SSL/TLS. This feature is used for e-commerce, banking, and other secure applications.

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Yes
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	No
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-12-20 23:59	No
sneaky.com	/	UserID=4627239101	31-12-19 23:59	No

How are cookies used??

Just before a browser sends a request for a page to some Web site, it checks its cookie directory to see if any cookies there were placed by the domain the request is going to.
If so, all the cookies placed by that domain, and only that domain, are included in the request message.
When the server gets them, it can interpret them any way it wants to.

There is some controversial use of cookie, What is that?

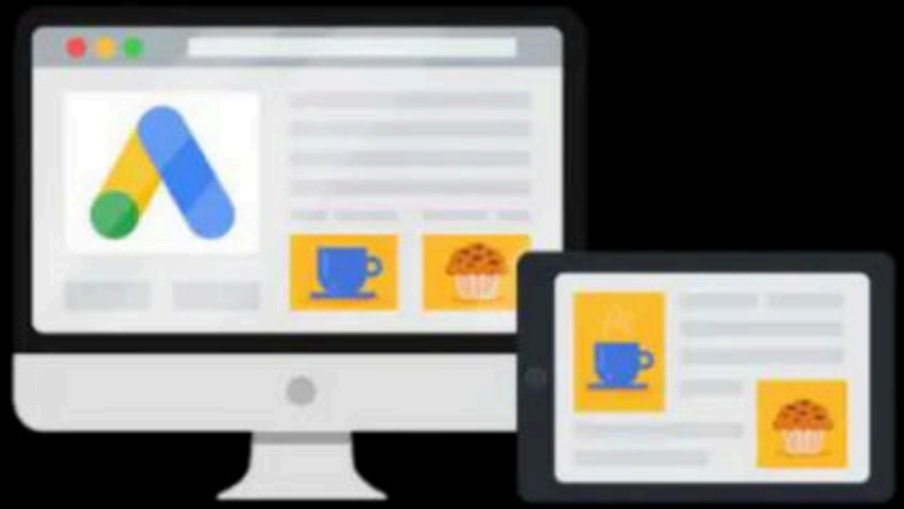
Web Tracking

Controversial use of cookies is to track the online behavior of users.
This lets Web site operators understand how users navigate their sites, and advertisers build up profiles of the ads or sites a particular user has viewed.
The controversy is that users are typically unaware that their activity is being tracked, even with detailed profiles and across seemingly unrelated Web sites. Nonetheless, Web tracking is big business.

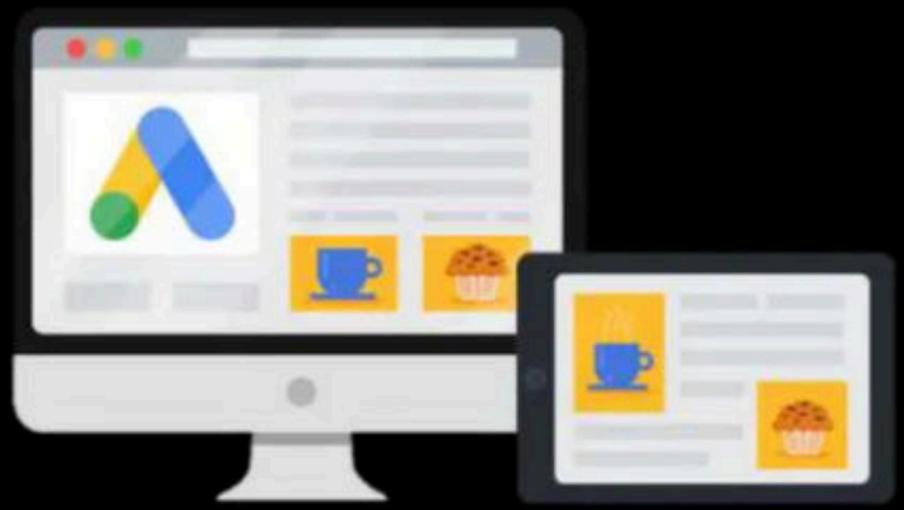
How Google Ads work?

An advertising agency, say, Sneaky Ads, contacts major Web sites and places ads for its clients' products on their pages, for which it pays the site owners a fee.

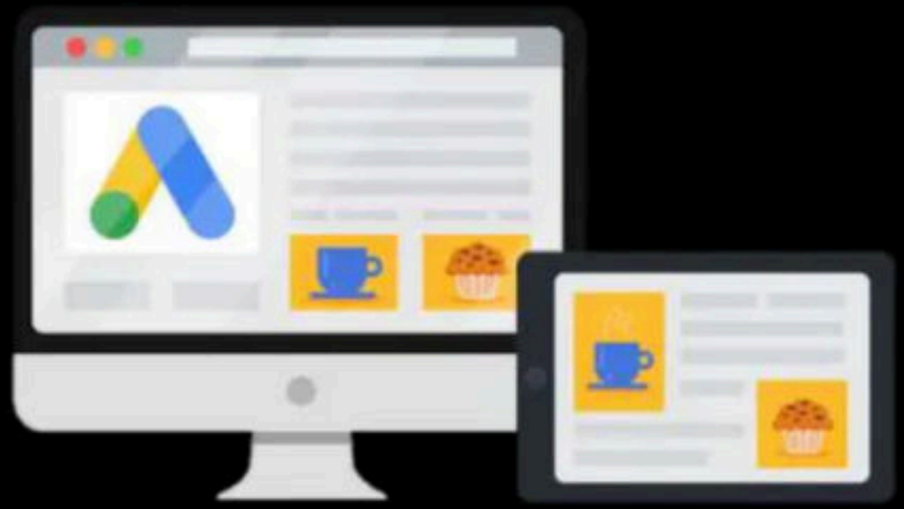
Instead, of giving the sites the ad as a GIF file to place on each page, it gives them a URL to add to each page.



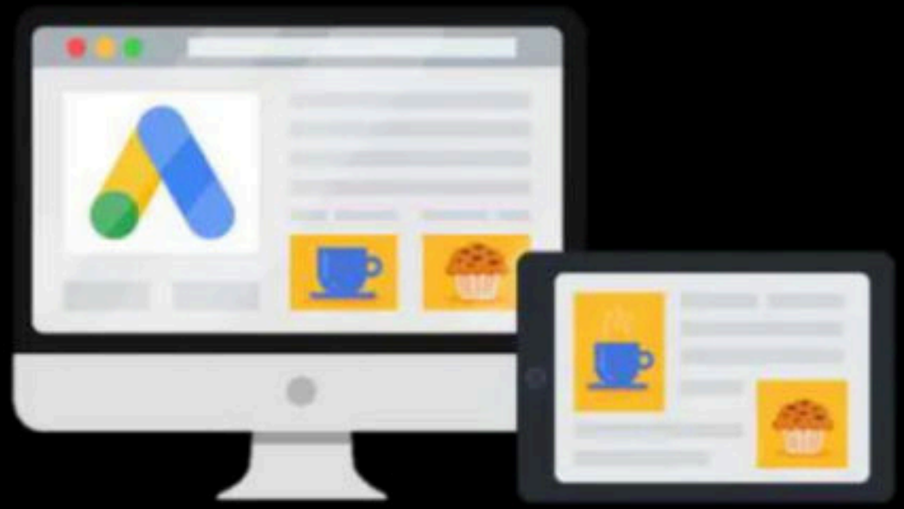
Each URL it hands out contains a unique number in the path, such as
http://www.sneaky.com/382674902342.gif



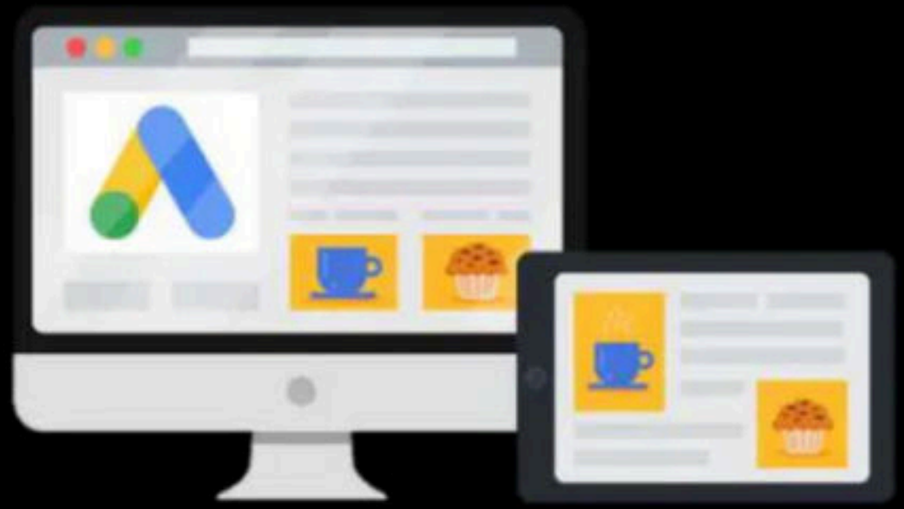
When a user first visits a page, *P*, containing such an ad, the browser fetches the HTML file. Then the browser inspects the HTML file and sees the link to the image file at *www.sneaky.com*, so it sends a request there for the image.
A GIF file containing an ad is returned, along with a cookie containing a unique user ID, 4627239101 (table we saw before) .



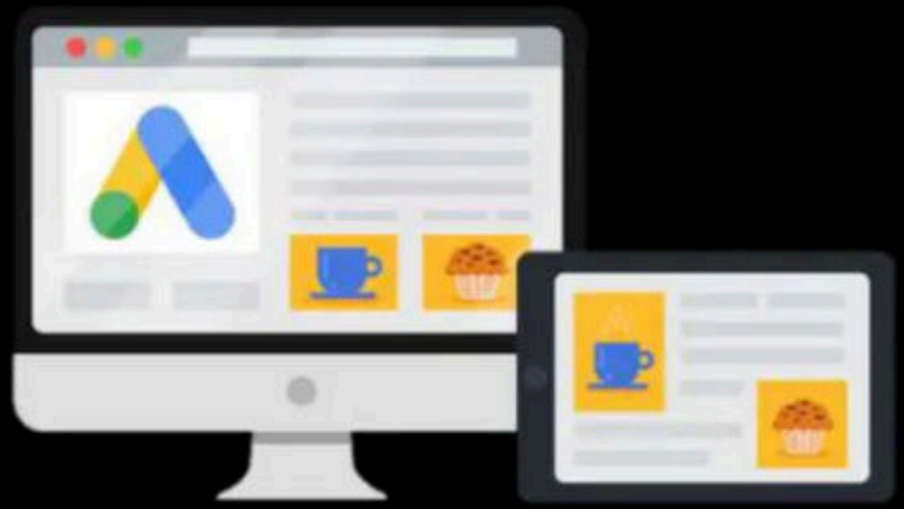
Sneaky records the fact that the user with this ID visited page *P*. This is easy to do since the path requested (*382674902342.gif*) is referenced only on page *P*. Of course, the actual ad may appear on thousands of pages, but each time with a different name. Sneaky probably collects a fraction of a penny from the product manufacturer each time it ships out the ad.



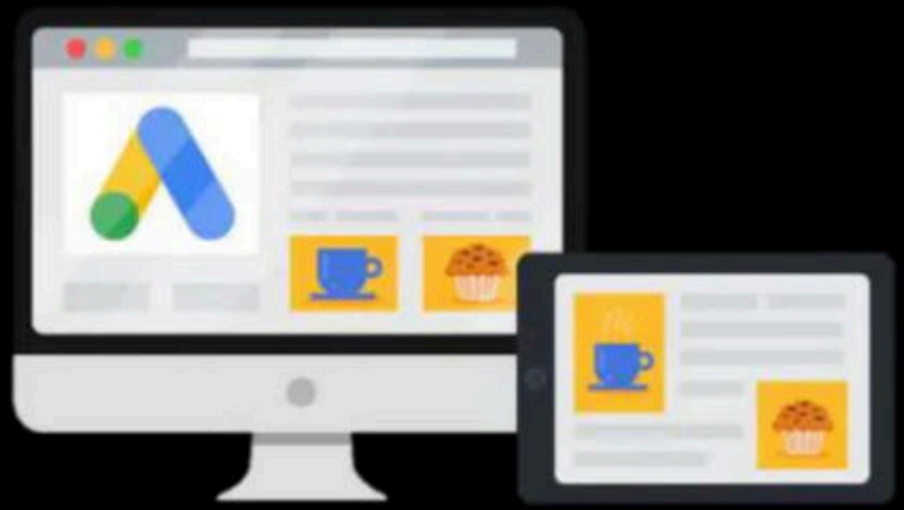
Later, when the user visits another Web page containing any of Sneaky's ads, the browser first fetches the HTML file from the server. Then it sees the link to, say, *http://www.sneaky.com/193654919923.gif* on the page and requests that file. Since it already has a cookie from the domain *sneaky.com*, the browser includes Sneaky's cookie containing the user's ID.
Sneaky now knows a second page the user has visited.



In due course, Sneaky can build up a detailed profile of the user's browsing habits, even though the user has never clicked on any of the ads. Of course, it does not yet have the user's name (although it does have his IP address, which may be enough to deduce the name from other databases).

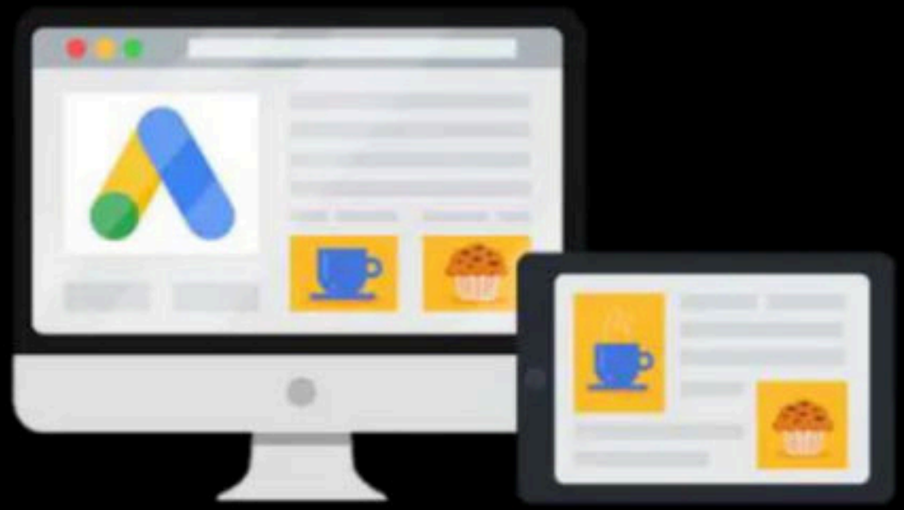


However, if the user ever supplies his name to any site cooperating with Sneaky, a complete profile along with a name will be available for sale to anyone who wants to buy it. The sale of this information may be profitable enough for Sneaky to place more ads on more Web sites and thus collect more information.



And if Sneaky wants to be supersneaky, the ad need not be a classical banner ad.

An ‘‘ad’’ consisting of a single pixel in the background color (and thus invisible) has exactly the same effect as a banner ad: it requires the browser to go fetch
the 1 × 1-pixel GIF image and send it all cookies originating at the pixel’s domain.



Spyware

Cookies have become a focal point for the debate over online privacy because of tracking behavior like the above. The most insidious part of the whole business is that many users are completely unaware of this information collection and may even think they are safe because they do not click on any of the ads. For this reason, cookies that track users across sites are considered by many to be **spyware**.

How to count, how many times
Visitor has visited a page ?

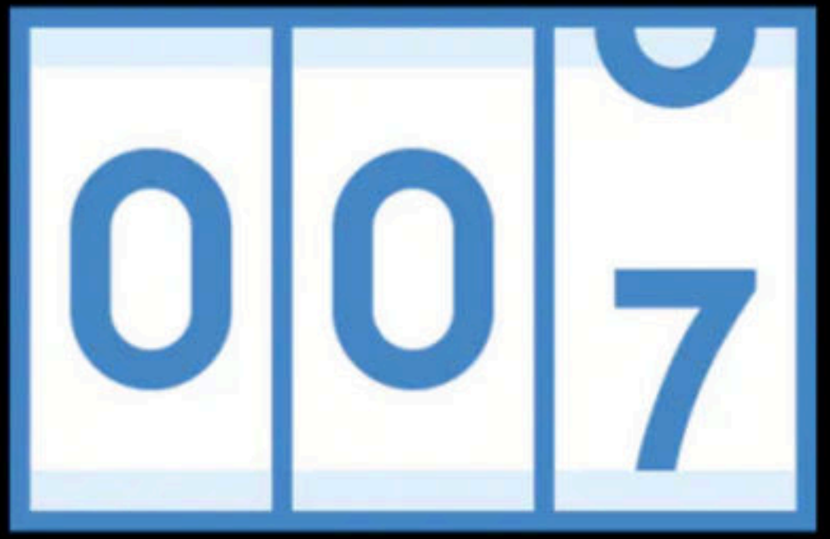
It is easy for a server to track user activity with cookies.

Suppose a server wants to keep track of how many unique visitors it has had and how many pages each visitor looked at before leaving the site.

When the first request comes in, there will be no accompanying cookie, so the server sends back a cookie containing *Counter = 1*.

Subsequent page views on that site will send the cookie back to the server. Each time the counter is incremented and sent back to the client.

By keeping track of the counters, the server can see how many people give up after seeing the first page, how many look at two pages, and so on.



How to Deal with Cookies?

- To maintain a semblance of privacy, some users configure their browsers to reject all cookies. However, this can cause problems because many Web sites will not work properly without cookies.

- Alternatively, most browsers let users block third-party cookies.

- A third-party cookie is one from a different site than the main page that is being fetched, for example, the sneaky.com cookie that is used when interacting with page P on a completely different Web site.

- Blocking these cookies helps to prevent tracking across Web sites.

WEB TECHNOLOGIES
HTML and CSS

HTML is a markup language, or language for describing how documents are to be formatted.

Writing a browser is then straightforward: the browser simply has to understand the markup commands and apply them to the content.

Embedding all the markup commands within each HTML file and standardizing them makes it possible for any Web browser to read and reformat any Web page.

That is crucial because a page may have been produced in a 1600 × 1200 window with 24-bit color on a high-end computer but may
have to be displayed in a 640 × 320 window on a mobile phone

- Tags can be in either lowercase or uppercase.

Thus, <head> and <HEAD> mean the same thing, but lower case is best for compatibility

Attributes

Some tags have (named) parameters, called **attributes**.

For example, the <img> tag is used for including an image inline with the text. It has two attributes, src and alt. The first attribute gives the URL for the image

<title>

The main item in the head is the title, delimited by <title> and </title>.
Certain kinds of meta information may also be present, though none are present in our example.
The title itself is not displayed on the page.
Some browsers use it to label the page's window.
Several headings are used in Each heading is generated by an <h*n*>
tag, where *n* is a digit in the range 1 to 6. Thus, <h1> is the most important head
ing; <h6> is the least important one. I

The tags <b> and <i> are used to enter boldface and italics mode, respectively.
The <hr> tag forces a break and draws a horizontal line across the display.
The <p> tag starts a paragraph

Lists

- HTML provides various mechanisms for making lists, including nested lists.
- Unordered lists, started with <ul>, with <li> used to mark the start of items. There is also an <ol> tag to starts an ordered list. The individual items in unordered lists often appear with bullets ( ) in front of them.
- Items in ordered lists are numbered by the browser.

<a>

- The <a> tag has various parameters, the most important of which is *href* the linked URL. The text between the <a> and is displayed.
- If it is selected, the hyperlink is followed to a new page. It is also permitted to link other elements.
- For example, an image can be given between the <a> and tags using <img>. In this case, the image is displayed and clicking on it activates the hyperlink.

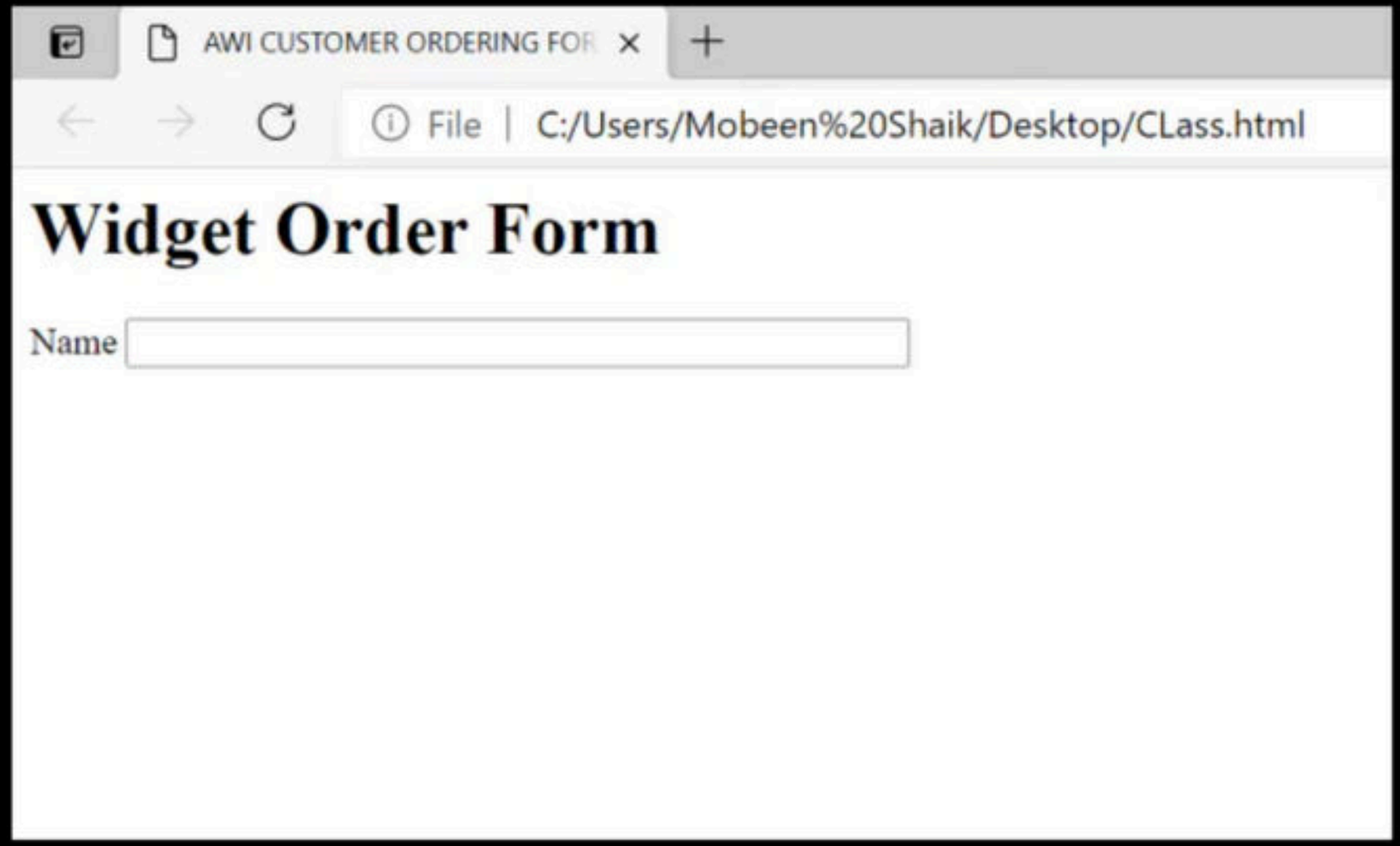
Forms

- **Forms** were included with this functionality in HTML 2.0.
- Forms contain boxes or buttons that allow users to fill in information or make choices and then send the information back to the page's owner. Note that forms are still static content.
- Like all forms, this one is enclosed between the <form> and </form> tags. The attributes of this tag tell what to do with the data that are input, in this case using the *POST* method to send the data to the specified URL.

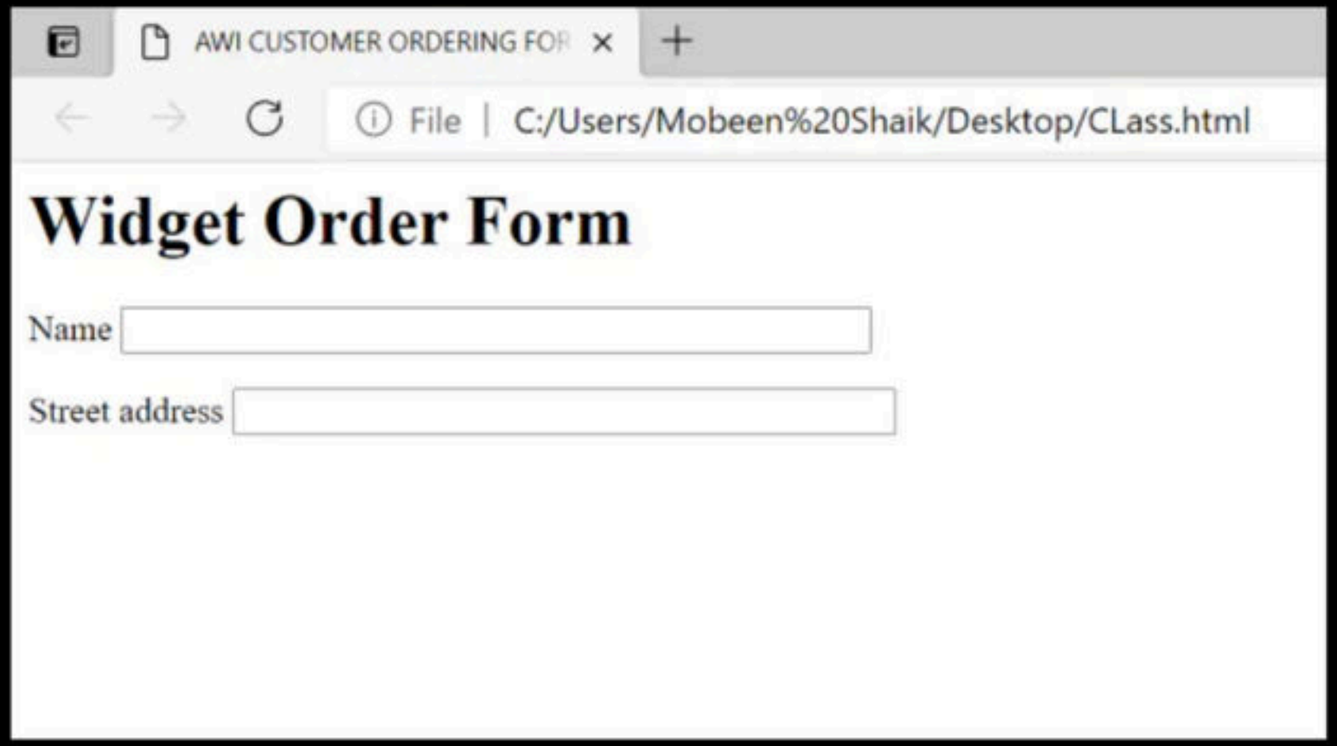
- Three kinds of input boxes are used in this form, each of which uses the <input> tag.
- It has a variety of parameters for determining the size, nature, and usage of the box displayed.
- The most common forms are blank fields for accepting user text, boxes that can be checked, and *submit* buttons that cause the data to be returned to the server.

```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
</form>
</body>
</html>
```

```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title>
</head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi"
method=POST>
<p> Name <input name="customer" size=46> </p>
</form>
</body>
</html>
```



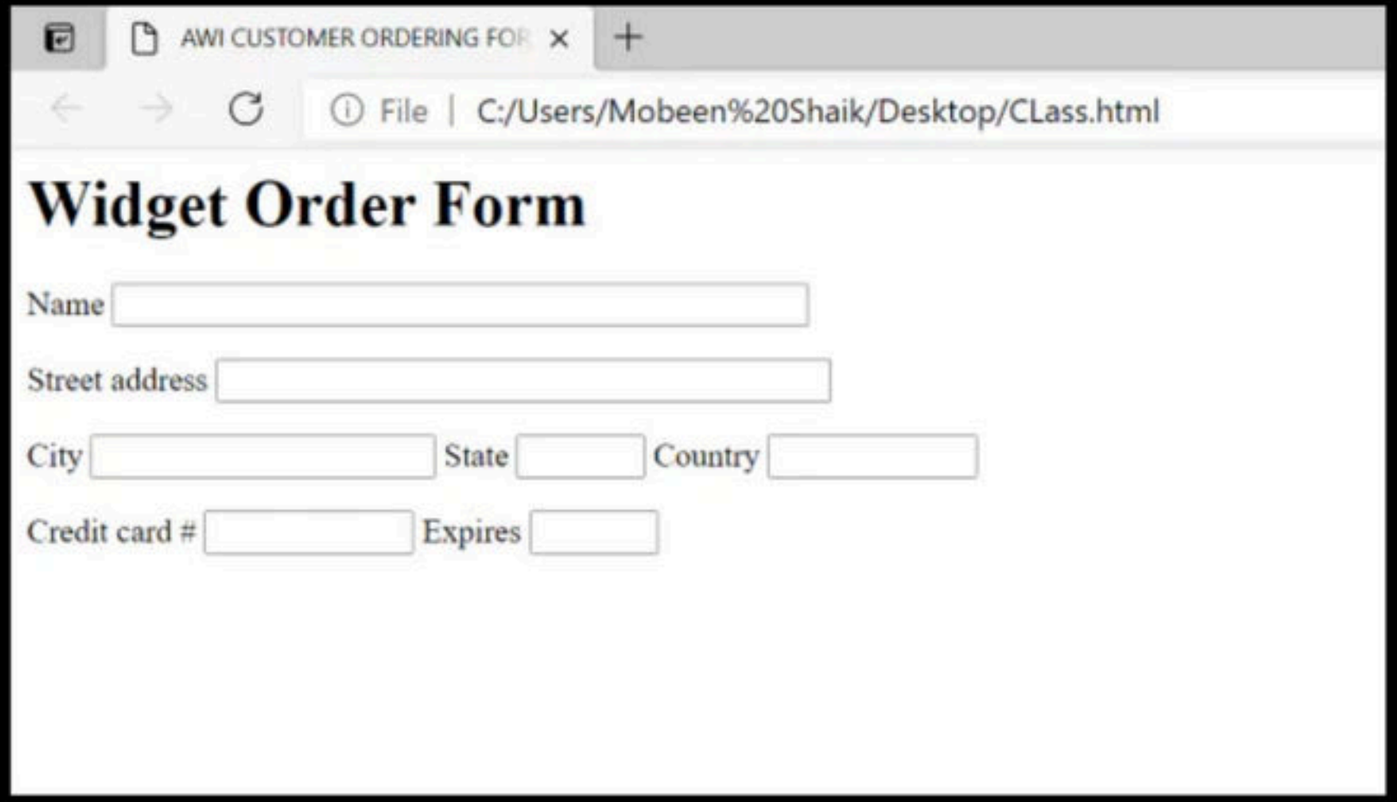
```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street address <input name="address" size=40> </p>
</form>
</body>
</html>
```



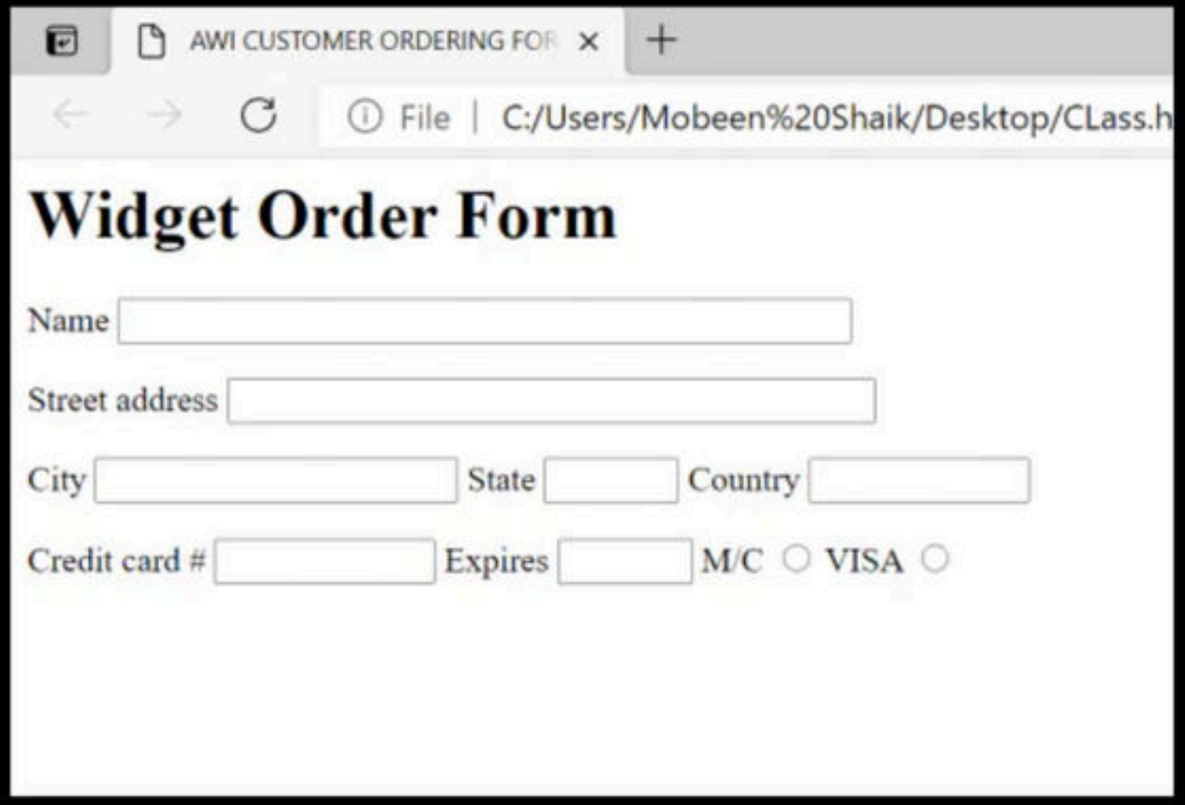
```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
</form>
</body>
</html>
```

```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
Country <input name="country" size=10> </p>
</form>
</body>
</html>
```

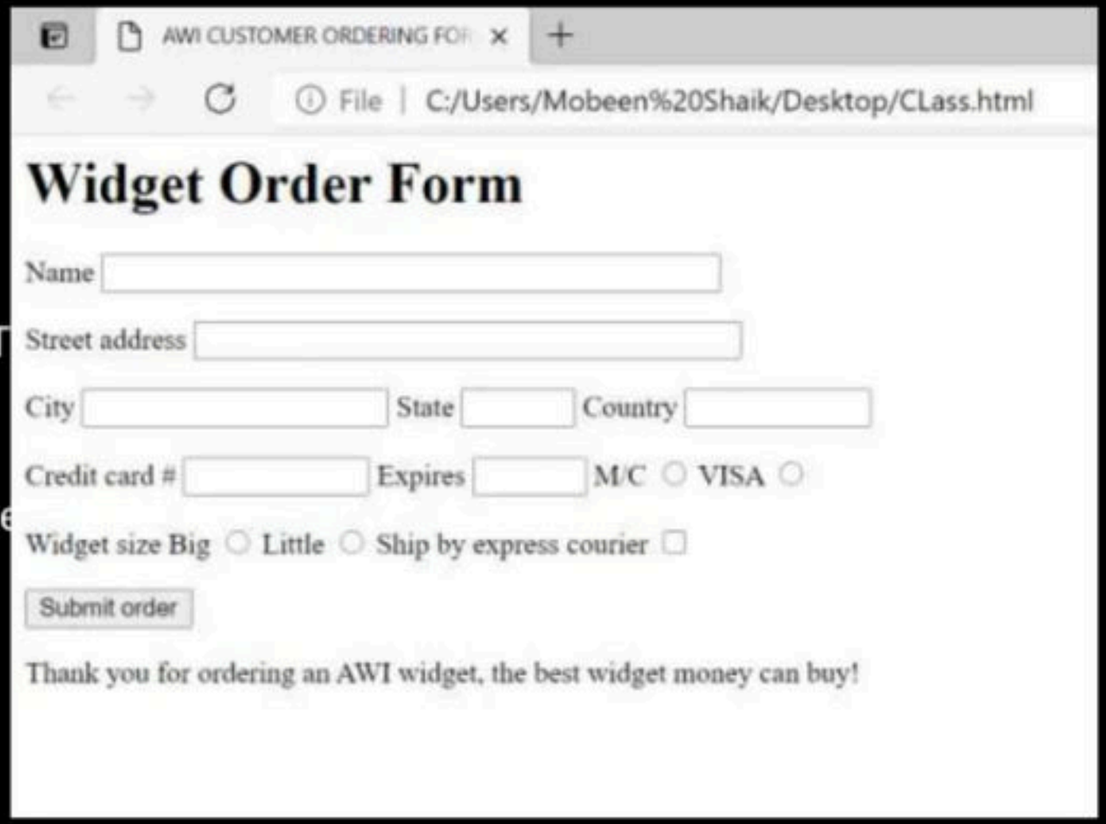
```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
</form>
</body>
</html>
```



```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
</form>
</body>
</html>
```



```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST
<p> Name <input name="customer" size=46> </p>
<p> Street address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive">
Little <input name="product" type=radio value="cheap">
Ship by express courier <input name="express" type=checkbox> </p>
<p><input type=submit value="Submit order"> </p>
Thank you for ordering an AWI widget, the best widget money can buy!
</form>
</body>
</html>
```



CSS—Cascading Style Sheets

The original goal of HTML was to specify the *structure* of the document, not its *appearance*.
For example,

```
<h1> Deborah's Photos </h1>
```

instructs the browser to emphasize the heading, but does not say anything about the typeface, point size, or color. That is left up to the browser, which knows the properties of the display (e.g., how many pixels it has). However, many Web page designers wanted absolute control over how their pages appeared, so new tags were added to HTML to control appearance, such as

```
<font face="helvetica" size="24" color="red"> Deborah's Photos </font>
```

CSS (**Cascading Style Sheets**) introduced style sheets to the Web with HTML 4.0, though widespread use and browser support did not take off until 2000. CSS defines a simple language for describing rules that control the appearance of tagged content. Let us look at an example. Suppose that AWI wants snazzy Web pages with navy text in the Arial font on an off-white background,
and level headings that are an extra 100% and 50% larger than the text for each level, respectively.

```
body {background-color:linen; color:navy; font-family:Arial;}
h1 {font-size:200%;}
h2 {font-size:150%;}
```

CSS example

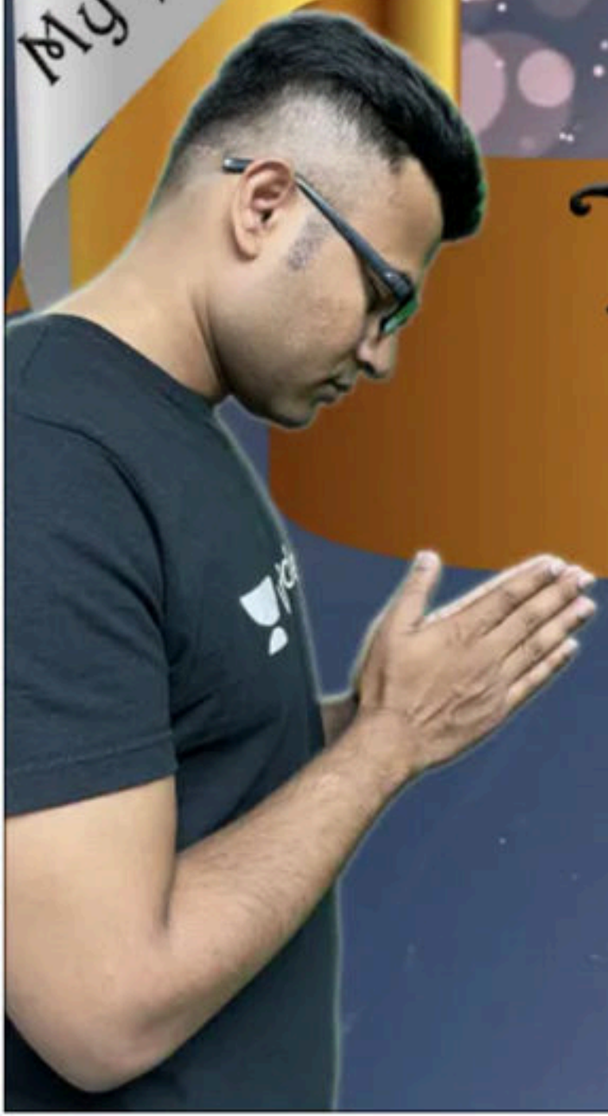
- As can be seen, the style definitions can be compact.
- Each line selects an element to which it applies and gives the values of properties. The properties of an element apply as defaults to all other HTML elements that it contains.
- Thus, the style for `body` sets the style for paragraphs of text in the body.
- There are also convenient short hands for color names (e.g., `red`). Any style parameters that are not defined are filled with defaults by the browser. This behavior makes style sheet definitions optional; some reasonable presentation will occur without them.

Style sheets can be placed in an HTML file (e.g., using the <style> tag), but it is more common to place them in a separate file and reference them.
For example,
the <head> tag of the AWI page can be modified to refer to a style sheet in the file *awistyle.css*.
The example also shows the MIME type of CSS files to be *text/css*.

```
<head>
<title> AMALGAMATED WIDGET, INC. </title>
<link rel="stylesheet" type="text/css" href="awistyle.css" />
</head>
```

Including a CSS style sheet.

My philosophy



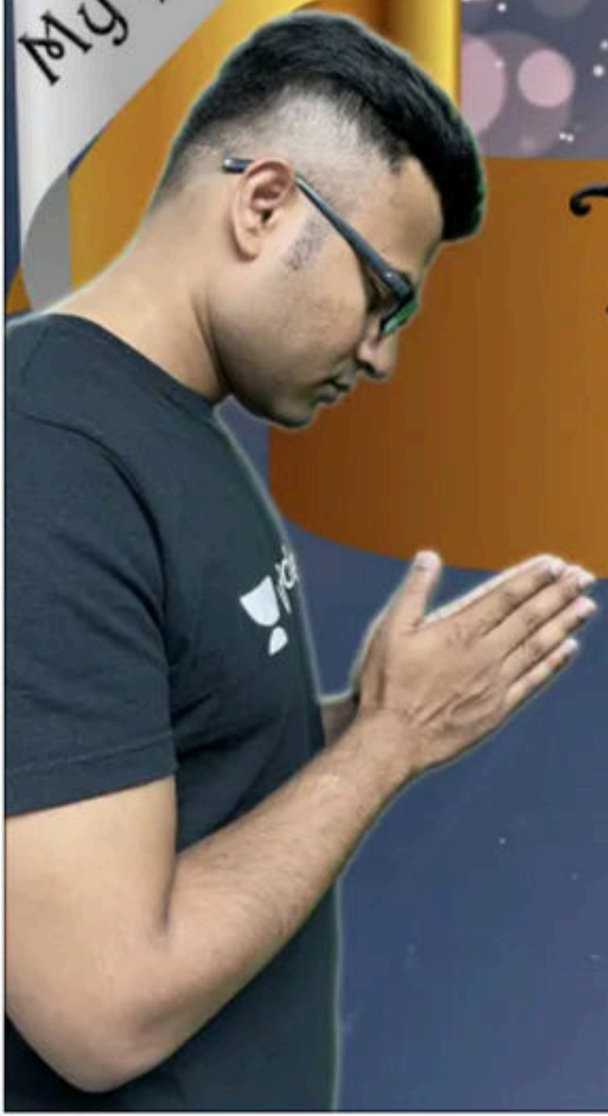
TEACHING IS WORSHIP

STUDENTS ARE GODS

My philosophy

TEACHING IS WORSHIP

STUDENTS ARE GODS



INTRODUCTION

- HTTP is an acronym for Hypertext Transfer Protocol . It is an application layer protocol in the TCP/IP protocol suite.
- WWW is repository of resources (web pages) stored in different computers all over the world. The primary purpose of HTTP is to transfer web pages from one computer (web server) to another computer(web client).
- HTTP can be used to access virtually all types of resources on the web. It allows us to transfer a wide variety of data such as text ,image ,audio , video and even the result of a query.

WEB SERVERS AND CLIENTS

- HTTP protocol is basically a request- response protocol between clients and servers.
- According to this protocol , a process is run which creates and stores resources such as HTML files, images ,etc. This process provides the resources on request and is called a web server or HTTP server.

To access resources stored on the web server, a process is designed to communicate with the server. The process is called web client or user agent.

- Web browsers are nothing but those web clients . They use HTTP protocol to communicate with web server to access resources specified by the URL at the address bar of the web browsers.

- Web servers typically runs on port 80 though any available port number may be used.

In TCP/IP protocol suite, a process in a machine is assigned a locally unique positive integer called port number.

- IP address together with the port number uniquely identifies a process all over the world.

So, IP address together with the port number uniquely identifies a process all over the world. This (IP address , port) pair is called a socket address of the process . To communicate with a web server , web clients need to specify the socket address of the web server.

URL AND ITS ANATOMY

A resource on the web is identified by an address called Uniform Resource Locator (URL).

An HTTP URL has the following form:

protocol : //host : [port]/[path[? params] [#anchor]]

protocol : //host : [port]/[path[? params] [#anchor]]

It indicates the protocol to be used for this URL. For HTTP URL , the protocol is http. Other possible protocols are ftp, gopher, mailto ,news ,nntp , telnet , wais ,file and prospero.

protocol : //host : [port]/[path[? params] [#anchor]]

This is the Fully Qualified Domain Name (FQDN) or the IP address of the computer where the web server runs.

Domain names are case –insensitive .So, www.google.com and WWW.GOOGLE.COM refer to the same host.

protocol : //host : [port]/[path[? params] [#anchor]]

Web servers typically run on port 80, if no port is specified ,port 80 is assumed.

protocol : //host : [port]/[path[? params] [#anchor]]

- This is the location of a file or a program (CGI ,Perl ,PHP ,JSP , etc) on the server relative to a document root specified by the web server. The document root is a directory where resources are stored.
- In the Apache web server ,the document root is usually set to /var /www/html. However , it can be configured.

protocol : //host : [port]/[path[? params] [#anchor]]

- This portion of the URL contains the parameters to be passed to web applications such as CGI, PERL, PHP or JSP.

- The path and params are separated by ? character. The params consists of a (name=value) pair separated by an ampersand(&).

For example, login= ukr&sid=145321& page=inbox specifies three parameters , login, sid and page whose values are “ukr ”, “145321” and “inbox” respectively.

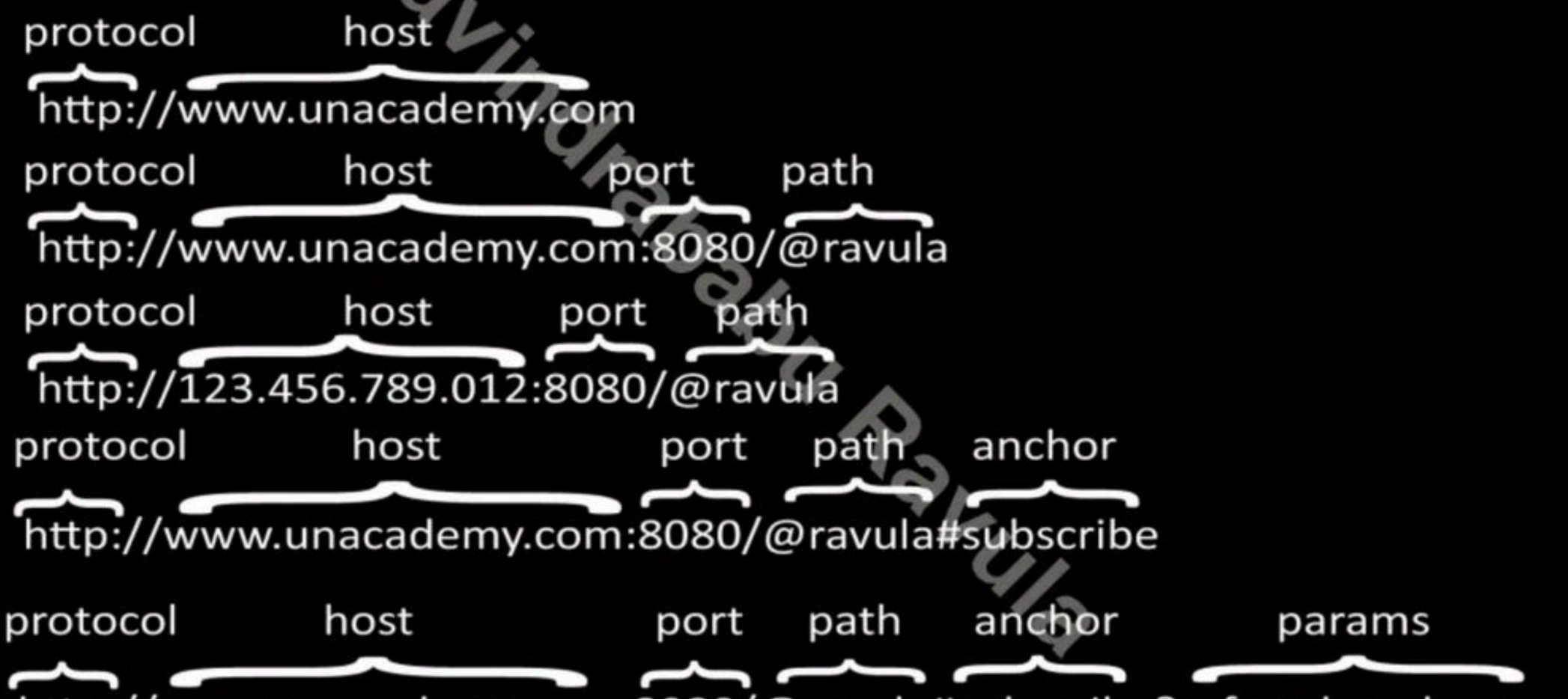
- The server - side program typically extracts this information from the URL for processing.

protocol : //host : [port]/[path[? params] [#anchor]]

This part indicates a specific location in the web page. For example, #appendix specifies the appendix section of the web page. This location is created using <a> tag as follows:

 Appendix
The named section can then be referred to as :
http:// www.it.jusl.ac.in/httphelp.htlm#appendix

URL EXAMPLE



MESSAGE FORMAT

It specifies a set of rules that clients and servers use to communicate:

- An HTTP server process is created on a port (usually 80) , which waits for clients to establish a TCP connection.

- An HTTP client initiates a TCP connection with the HTTP server (process) at the designated port.

- The HTTP server accepts this connection.

- The HTTP client then sends a request for a resource to the server.

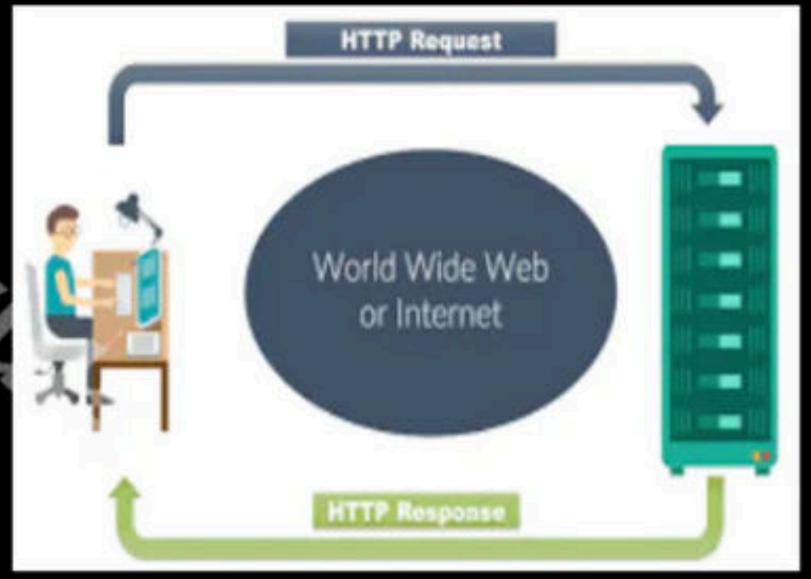
- Upon receiving the request , the server processes the request ,performs the desired task, and sends a response back to the client.

- The HTTP server closes the TCP connection.

- The HTTP client receives the response containing information and processes it.

Note that every time the HTTP client wants to get resource from the server, it has to follow these steps .

This makes HTTP stateless .This means that web server treats every request as a new request . There is no way to specify that some requests are related.



REQUEST MESSAGE

A request message is sent by a web client to the web server . It consists of the following parts:

- A request line.
- A header.
- An empty line.
- An optional body.

Request line
Header
Empty Line
Body(available for some messages)

HTTP request message format

REQUEST LINE

A request line consists of three parts : request type, URL, and HTTP version . Two consecutive parts are separated by a space.

Request Type		URL		HTTP Version

REQUEST TYPE (METHOD)

- It indicates the type of request , a client wants to send. They are also called methods.
- A method makes a message either a request or a command to the server.
- Request messages are used to retrieve data from the server whereas a command tells the server to do a specific task.

GET

- This is the most frequently used method in the WWW. It is specified when client wants to retrieve (GET) a resource(document) from the server.
- The URL in the request line identifies the resource . If the URL specified is a valid one , the server reads the content of the resource and sends the content back to the client; otherwise an error message is sent back to the client.
- If the resource being requested is a server-side program such as CGI script or ASP or JSP , the result generated by the program is returned instead of the content of the resource.
- The message body is empty for the GET method.
- The GET method may be a “conditional GET” method. In such a case, the request message includes an “If-Modified-Since” header that specifies a date . This header specifies that the identified resource has to be transferred only if it is modified after the specified date. So users can avoid downloading resources that were downloaded on some earlier date and have not been modified since.
- The conditional GET method effectively reduces network bandwidth and helps increasing network performance.

HEAD

- It is used when the client wants to know the header information (meta-information) about a resource but not the resource content.

POST

- It is used when a client wants to send (POST) some information (possibly large) to the server.
- The actual information is included in the body part of the request message instead of appending it to the URL as done in the GET method.
- The headers describe the message body such as content type and content length.
- The commonest form of the POST method is to submit an HTML form to the server.

PUT

- It is used to upload a new resource or replace an existing document. The actual document is specified in the body part.
- As the PUT method can modify or replace an existing document, it is vulnerable and is not permitted by most of the web servers.
- However, it is not recommended to configure web servers in such a way without any valid reasons.

PATCH

- This is similar to PUT method except that it specifies a list of differences that must be applied on the existing file.

COPY

- The HTTP protocol may be used to copy a file from one location to another.
- The method COPY is used for this purpose. The URL specified in the request line specifies the location of the source file.
- The location of the target file is specified in the entity header.
- Note that the target web server must be configured properly to accept the COPY method.
- This method is also vulnerable.

MOVE

- It is similar to the COPY method except that it deletes the source file.
- The location of the source file is specified by the URL in the request line.
- The entity header specifies the location of the target file.
- Note that the target web server must be configured properly to accept the MOVE method.
- This method is also vulnerable.

DELETE

- This method is used to remove a document from the server.
- The location of the document to be deleted is specified by the URL in the request line.
- This method is vulnerable.

LINK

- This is used to create a link or links from one document to another.
- The URL in the request line specifies the location of the source file and the entity header specifies the location of the target document.

UNLINK

- It is used to remove a link or links created by the LINK method.

OPTIONS

- It is used to retrieve the set of methods supported by the server.
- It is used to check whether a server is functioning properly before performing other tasks.
- It is used to check whether the web server really supports a method before actually using that method.

CONNECT

- It is used to convert a request connection into the transparent TCP/IP tunnel.
- It is usually done to facilitate Secured Socket Layer(SSL) encrypted communication(eg HTTPS) through an unencrypted HTTP proxy server.

TRACE

- It is used to instruct the web server to echo the request back to the client.
- The client can then see what additions or change are done by the immediate servers.

SAFE AND UNSAFE METHODS

- Among the methods discussed ,some methods such as GET ,HEAD, OPTIONS and TRACE are used only to retrieve information from the server . Such methods are called safe. They cannot change the state of the server . This means, they do not have any harmful side effects such as caching ,logging ,etc .
- On the other hand, methods such as DELETE,MOVE,UNLINK take actions that may change the state of the server.

These methods have harmful side effects and hence are vulnerable.

Sensitive web servers are usually not configured to accept these methods.

HTTP VERSIONS

This field specifies the version of the HTTP protocol being used. It can have the following values: HTTP/1.0 and HTTP/1.1 .

The current version is HTTP/1.1.

RESPONSE MESSAGE

In response to the request message , a response message is sent by a server to the client. It consists of the following parts:

- A status line
- A header
- An empty line
- An optional body

Status line
Header
Empty Line
Body(available for some messages)

HTTP response message format

STATUS LINE

Status line consists of three parts: HTTP version ,Status code and Status phrase. Two consecutive parts are separated by a space.

HTTP Version		Status Code		Status Phrase

STATUS CODE

It is a three –digit code that indicates the status of the response. The status codes are classified with respect to their functionality into five groups:

- 1 xx series (Informational) – This class of status codes represents provisional responses.
- 2 xx series (Success) – This class of status codes indicates that the client's requests are received, understood and accepted successfully.
- 3xx series(Redirectional) – These status codes indicates that additional actions must be taken by the client to complete the request. The user agent may take further actions in order to fulfill the request automatically provided that it uses either the HEAD or the GET method.
- 4xx series(Client error) – These status codes are used to indicate that the client request had an error and therefore it cannot be fulfilled. Except for the HEAD method, the body of the response message contains the explanation that caused the error. The user agent should display the error message to inform the user.
- 5xx series(Server Error) – This set of status code indicates that the server encountered some problem and hence the request cannot be satisfied at this time. The reason of the failure is embedded in the message body . It also indicates whether the failure is temporary or permanent.

HEADERS

- HTTP headers are very important part of both request message and response message.
- They collectively specify the characteristics of the resource requested and the data that are provided.
- For Example, a client may want to accept image files only in specified format.
- The headers are separated by an empty line from the request and response body.

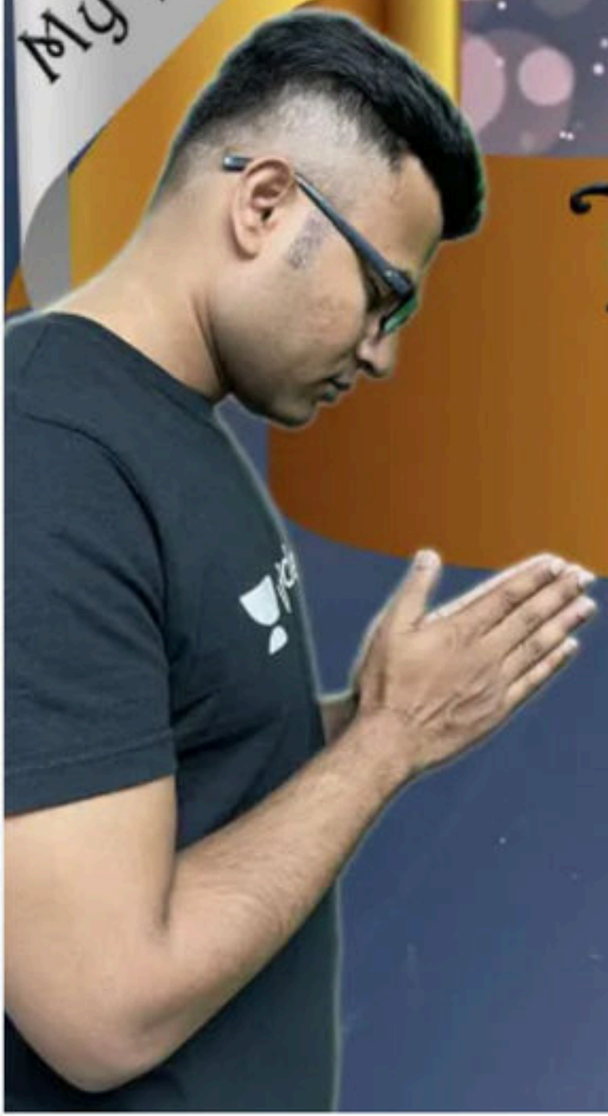
General Header
Request Header
Entity Header

HTTP Request Header Format

General Header
Response Header
Entity Header

HTTP Response Header Format

My philosophy



TEACHING IS WORSHIP

STUDENTS ARE GODS

<html> : root element of an HTML page.
<head> : element contains meta information about the HTML page
<title> : specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
<body> : defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
<! --> : Comments
The ** ** tag defines a division or a section in an HTML document.
The tag is used as a container for HTML elements - which is then styled with CSS or manipulated with JavaScript.
The tag is easily styled by using the class or id attribute. Any sort of content can be put inside the tag.

The <form> tag:

The <form> tag is used to create an HTML form for user input.

Attributes used in our form are:

Attribute	Value	Description
action	URL to server side script.	Specifies where to send the form-data when a form is submitted
method	GET Or POST	Specifies the HTTP method to use when sending form-data
name	text	Specifies the name of a form

The <form> element contains following:

<input>

<select>

<option>

The <select> element is used to create a drop-down list. The <select> element is most often used in a form, to collect user input.
The <option> tag defines an option in a select list. <option> elements go inside a <select>
The **<table>** tag defines an HTML table.
An HTML table consists of one <table> element and one or more <tr> and <td> elements.
The <tr> element defines a table row, and the <td> element defines a table cell.

•The HTML <input> element is the most used form element. An <input> element can be displayed in many ways, depending on the type attribute.

Type	Description
<input type="text">	Displays a single-line text input field
<input type="radio">	Displays a radio button (for selecting one of many choices)
<input type="password">	Displays a single-line text input field by masking the characters.
<input type="submit">	Displays a submit button (for submitting the form)

The page (body) is structured as follows:

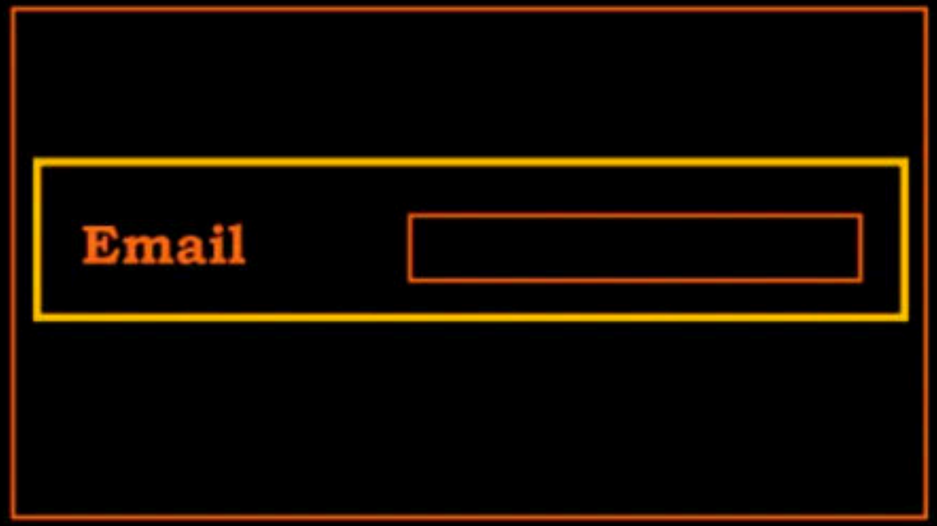
- A div is created (id=”div1”), which is split horizontally to contain div2 and div3.
- The second division container (id=”div2”) to display the link for login page (login.html)
- The third division container (id=”div3”) to display the HTML form (name="sign-up").
- All the form elements are aligned properly by creating a table with two columns and 9 rows.

```
<form name="sign-up" action="" method="">
<table>
<tr>
<td colspan=2 align="center">
<b>Registration</b>
</td>
</tr>
</table>
</form>
```

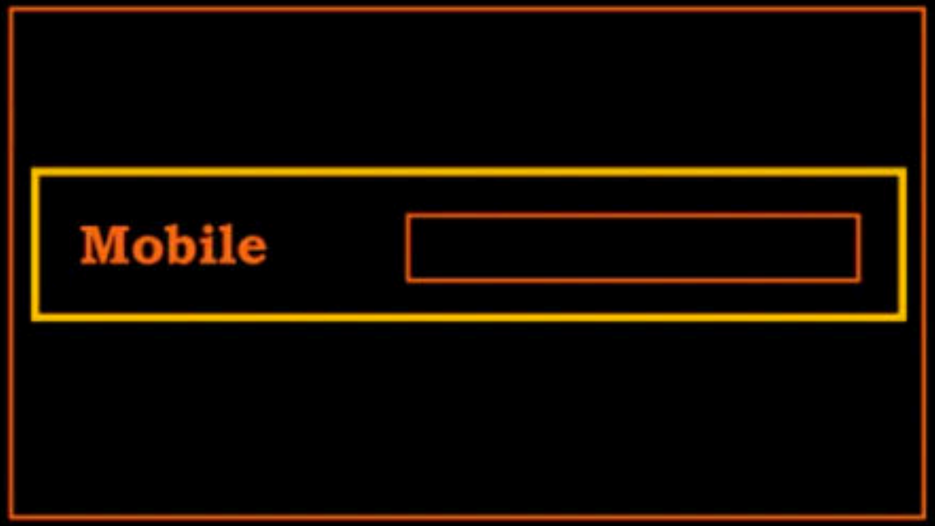
```
<form name="sign-up" action="" method="">
<table>
<tr>
<td colspan=2 align="center">
<b>Registration</b>
</td>
</tr>
</table>
</form>
```

```
<form name="sign-up" action="" method="">
<table>
<tr>
<td>Username </td>
<td><input  type="text" name="username“
 id="username"></td>
</tr>
</table>
</form>
```

```
<form name="sign-up" action="" method="">
<table>
<tr>
<td>Email </td>
<td><input  type="text" name="email"
id="email"></td>
</tr>

</table>
</form>
```

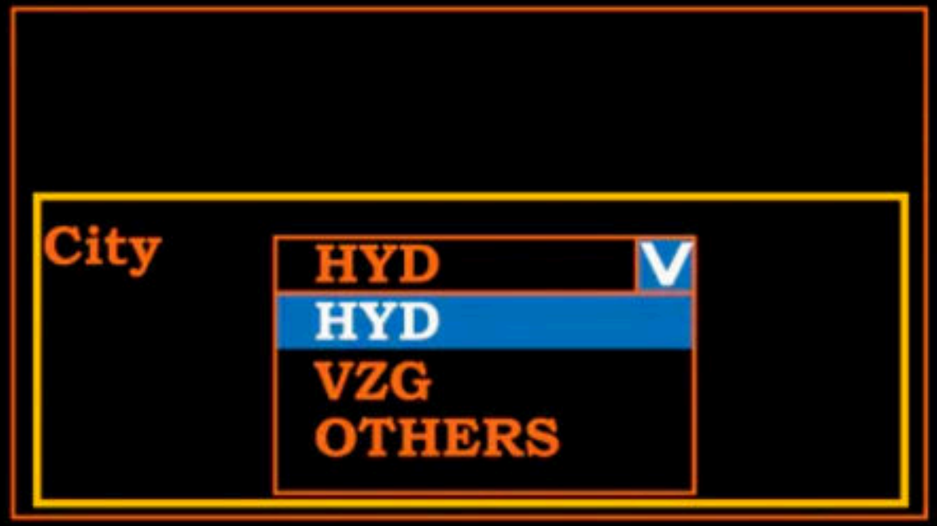


```
<form name="sign-up" action="" method="">
<table>
<tr>
<td>Mobile </td>
<td><input  type="text" name="mobile"
id="mobile"></td>
</tr>

</table>
</form>
```



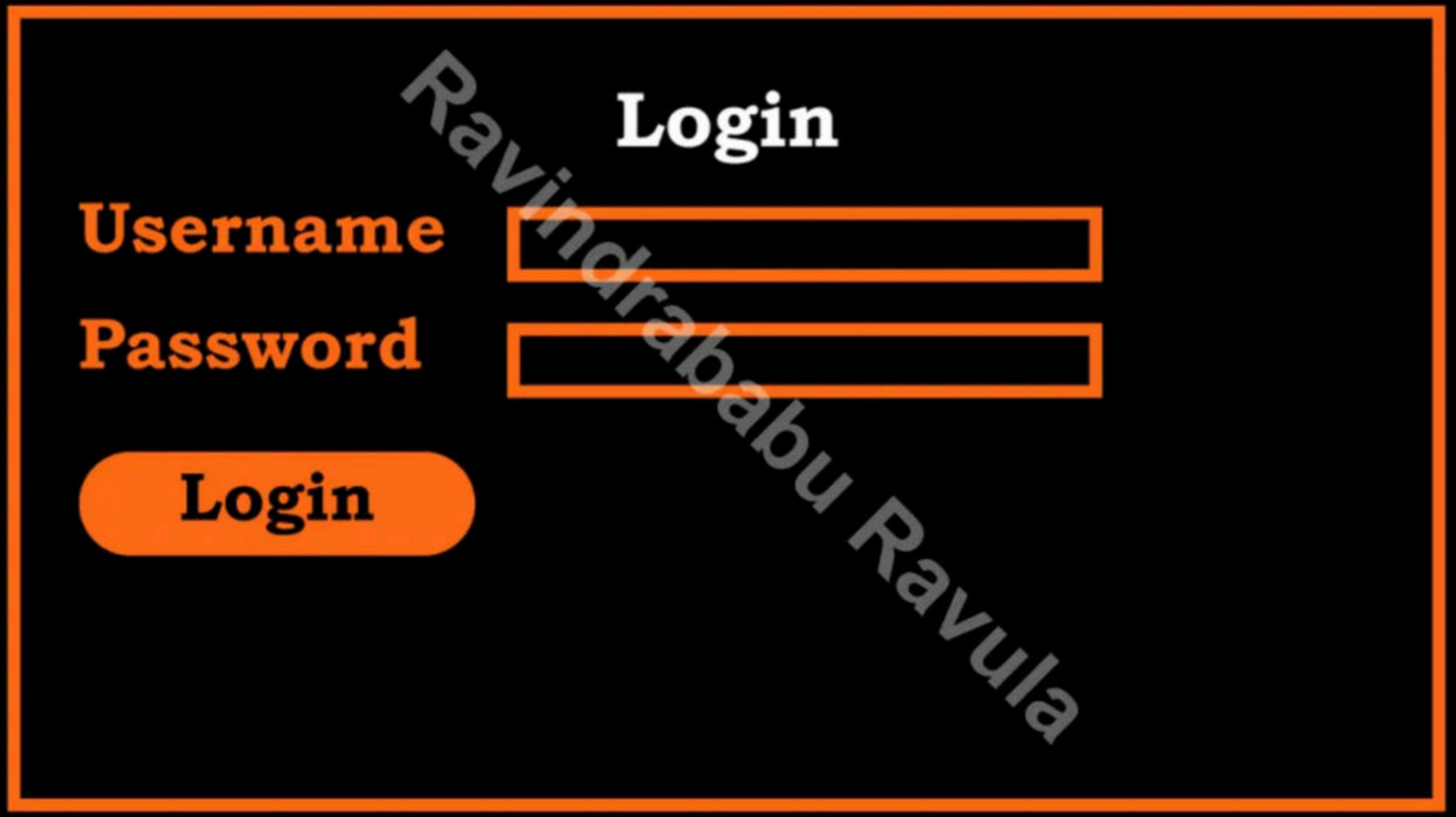
```
<form name="sign-up" action="" method="">
<table>
<tr>
<td>Gender </td>
<td>
<input type="radio" name="gender"> Male
<input type="radio" name="gender"> Female
</td>
</tr>

</table>
</form>
```

```
<form name="sign-up" action="" method="">
<table>
<tr>
<td>City </td>
<td>
<select>
<option value="HYD" name="city">HYD</option>
<option value="VZG" name="city" >VZG</option>
<option value="OTHER"name="city">OTHER
</option>
</select>
</td>
</tr>

</table>
</form>
```

```
<form name="sign-up" action="" method="">
<table>
<tr>
<td>Password </td>
<td><input  type="password" name="signup-
password" id="signup-password"></td>
</tr>

</table>
</form>
```

```
<form name="sign-up" action="" method="">
<table>
<tr>
<td colspan=2 align="center"><input class=
"btn" type="submit" name="signup-
btn“ id="signup-btn" value="Sign up">
</td>
</tr>
</table>
</table>
</form>
```



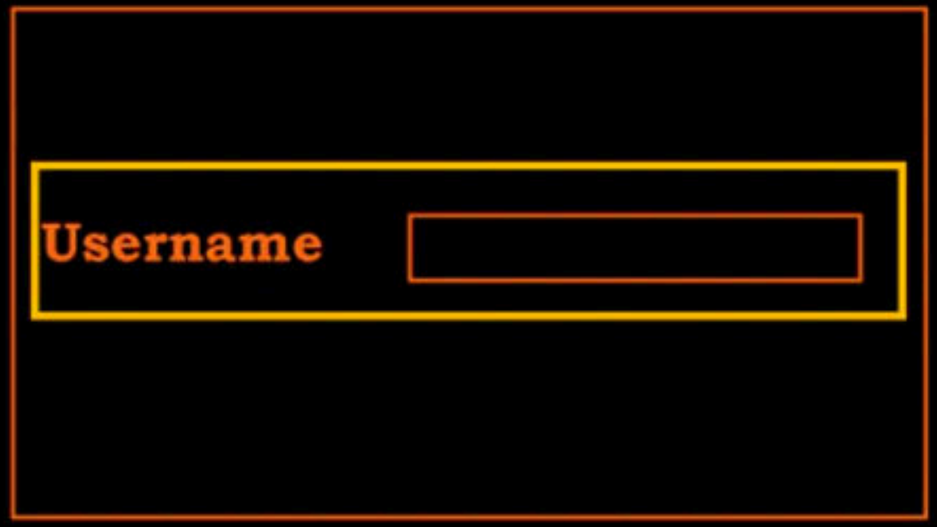
login.html

The page (body) is structured as follows:

- A div is created (id=”div1”), which is split horizontally to contain div2 and div3.
- The second division container (id=”div2”) to display the link for the signup page.
- The third division container (id=”div3”) to display the HTML form (name="login").
- All the form elements are aligned properly by creating a table with two columns and 4 rows.

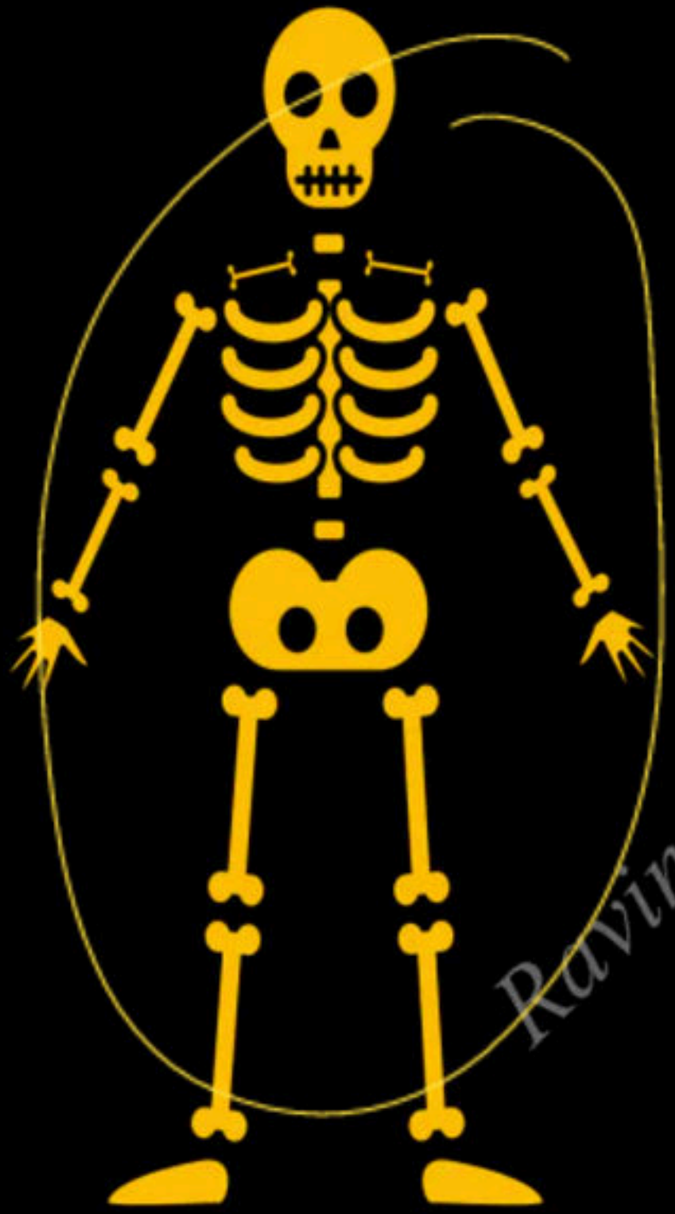
```
form name="sign-up" action="" method="">
<table>
<tr>
<td colspan=2 align="center">
<b>Login</b>
</td>
</tr>
</table>
</form>
```


```
<form name="sign-up" action="" method="">
<table>
<tr>
<td>Username </td>
<td><input  type="text" name="username“
 id="username"></td>
</tr>

</table>
</form>
```



```
<form name="sign-up" action="" method="">
<table>
<tr>
<td>Password </td>
<td><input  type="password" name=
"signup-password" id="signup-
password"></td>
</tr>

</table>
</form>
```

```
<form name="sign-up" action="" method="">
<table>
<tr>
<td><input type="submit" name="login-
btn" id="login-btn" value="Login"></td>
</tr>
</table>
</table>
</form>
```



HTML

HTML + CSS

What is CSS ?

- CSS : Cascading Style Sheets are used inside HTML to style the content of your HTML page.
- CSS is used to define how that content is to be displayed with colors, borders, fonts, backgrounds etc.
- Throughout our web technology course we use CSS 3.0 or version 3.

Applying CSS

There are three ways to apply CSS to HTML: *Inline*, *internal*, and *external*.

Inline styles are plonked straight into the HTML tags using the style attribute.

/* Inline Example */

```
<p style="color: red">RBR</p>
```

Embedded, or internal, styles are used for the whole page. Inside the head element, the style tags surround all of the styles for the page.

/* Internal style example */

```
<head>
  <style>
    p {
      color: red;
    }
  </style>
</head>
```

External styles are used for the whole, multiple-page website. There is a separate CSS file.

Applying CSS

/*If this file is saved as “style.css” in the same directory as your HTML page then it can be linked to in the HTML like this: */

```
<head>
   <link rel="stylesheet" href="style.css">
</head>
```

CSS Properties and Rules

A CSS property styles an aspect of an HTML element.
A CSS property declaration consists of a property name and a property value.

property-name : property-value

A CSS rule is a grouping of one or more CSS properties which are to be applied to one or more target HTML elements.

```
div {
    border    : 1px solid black;
    font-size : 18px;
 }
```

CSS Selectors

CSS selectors are the part of CSS rules that determine what HTML elements that are affected by the CSS rule.

```
div {
    border: 1px solid black;
}
```

 is the CSS selector part of the above CSS rule.

Different types of selectors

Universal Selector

The universal CSS selector is used to select all elements. It is marked with a *.

```
* {
    font-size: 18px;
}
```

Element Selector :

The element selector is the most basic CSS selector.

It selects all the HTML elements of the same type.

```
div {
    border: 1px solid black;
}
p {
    font-size: 18px;
}
```

Class Selector

The class selector selects all HTML elements which have the given CSS class set on them. You set a CSS class on an HTML element by giving the HTML element a class attribute.

```
<div class="red"> RBR... </div>

<p class="red"> My Paragraph... </p>

/* CSS class code */

.red {
    border: 1px solid red;
}
```

ID Selector

The ID of an HTML element is set via the id attribute.

```
<div id="mydiv"> Content </div>
```

You can select this HTML element using an ID selector, by prefixing the CSS selector with a # and then write the ID of the HTML element.

```
#mydiv {
    border: 1px solid blue;
}
```

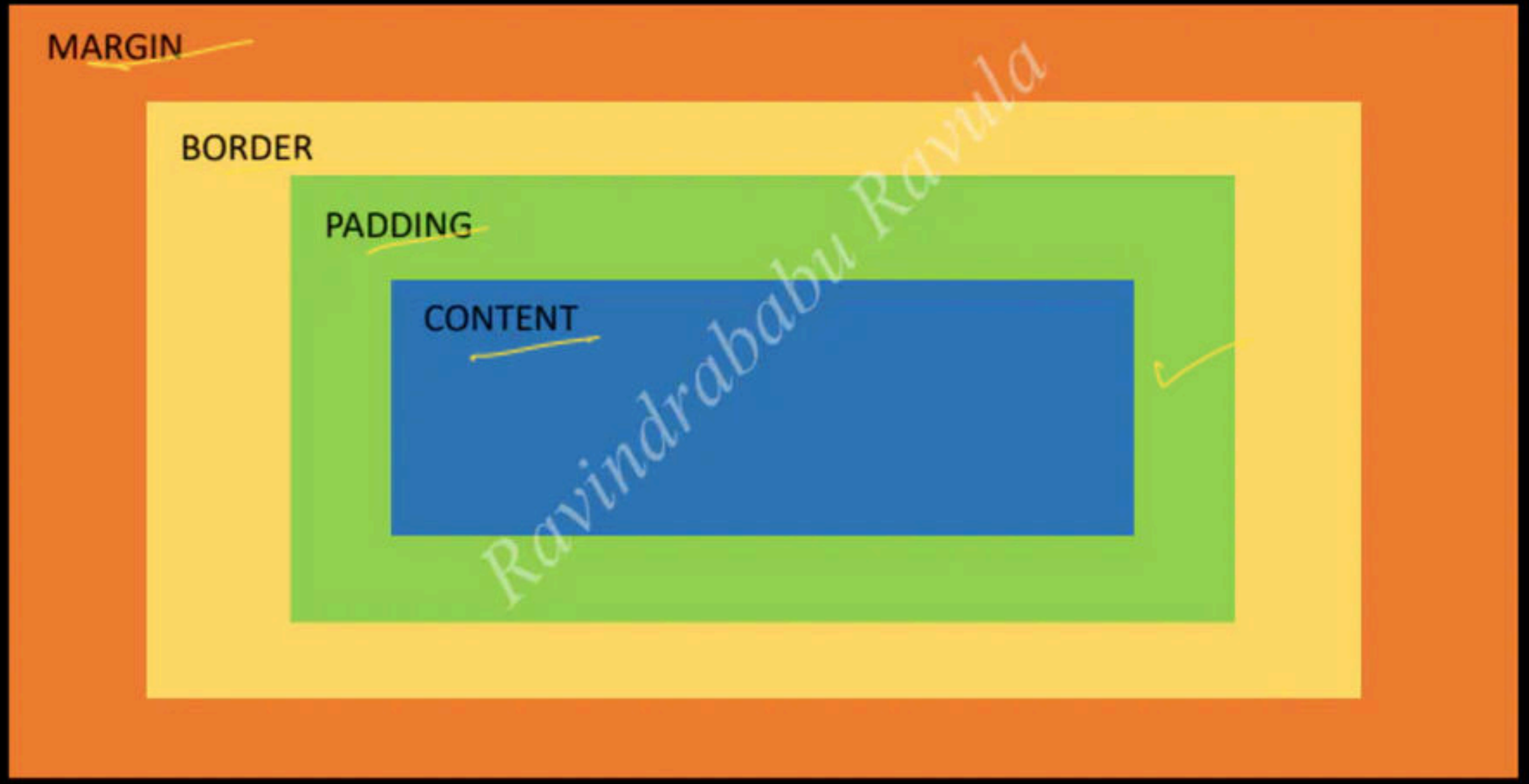
Group Selector

- The CSS group selector is used to group together multiple selectors into one, big CSS selector.
- That means that all elements targeted by any of the selectors will be affected by the CSS rule.

```
div, p {
   font-size : 18px;
}
```

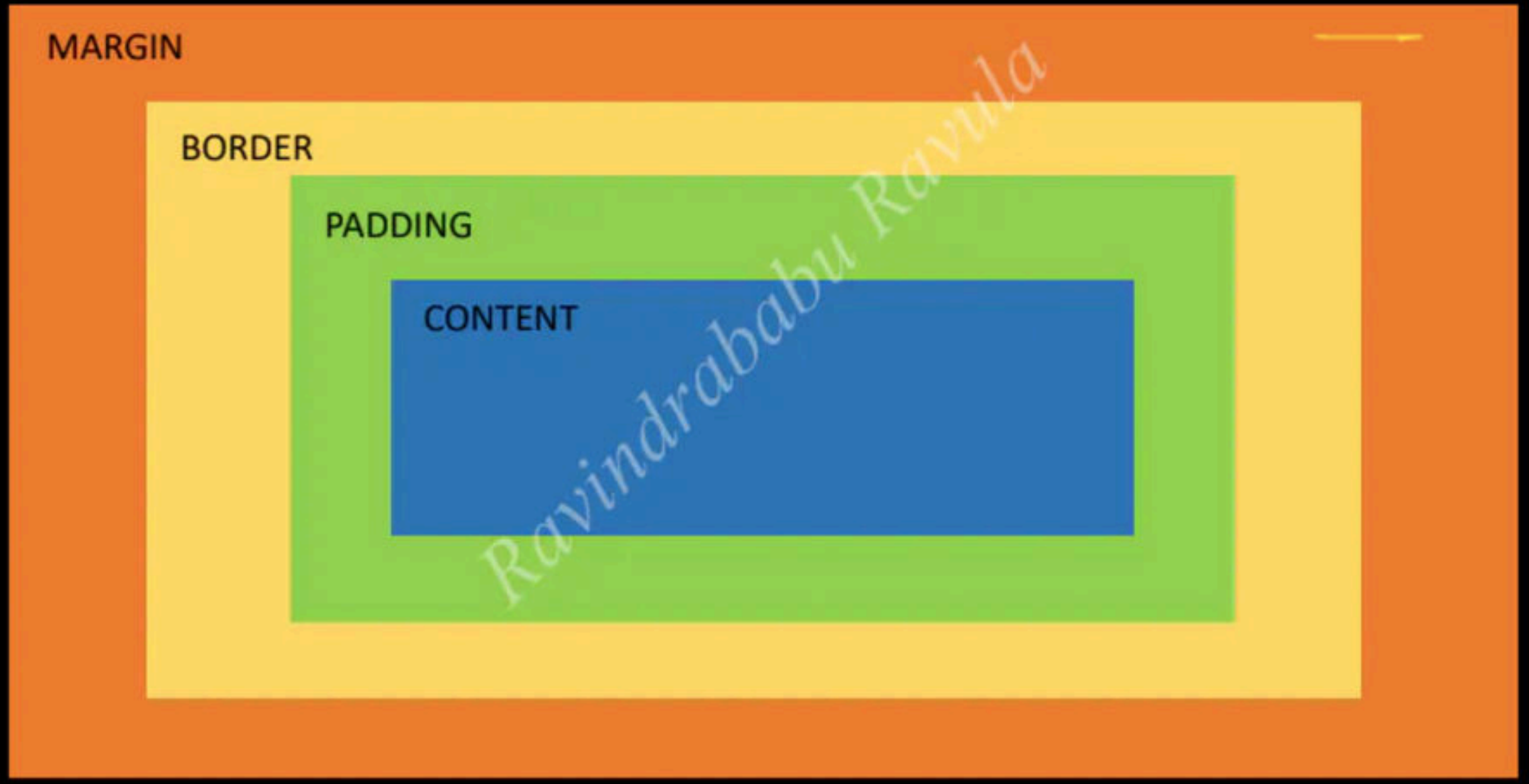
CSS Box Model

Each HTML element rendered is considered to be a box.
The box has four layers.



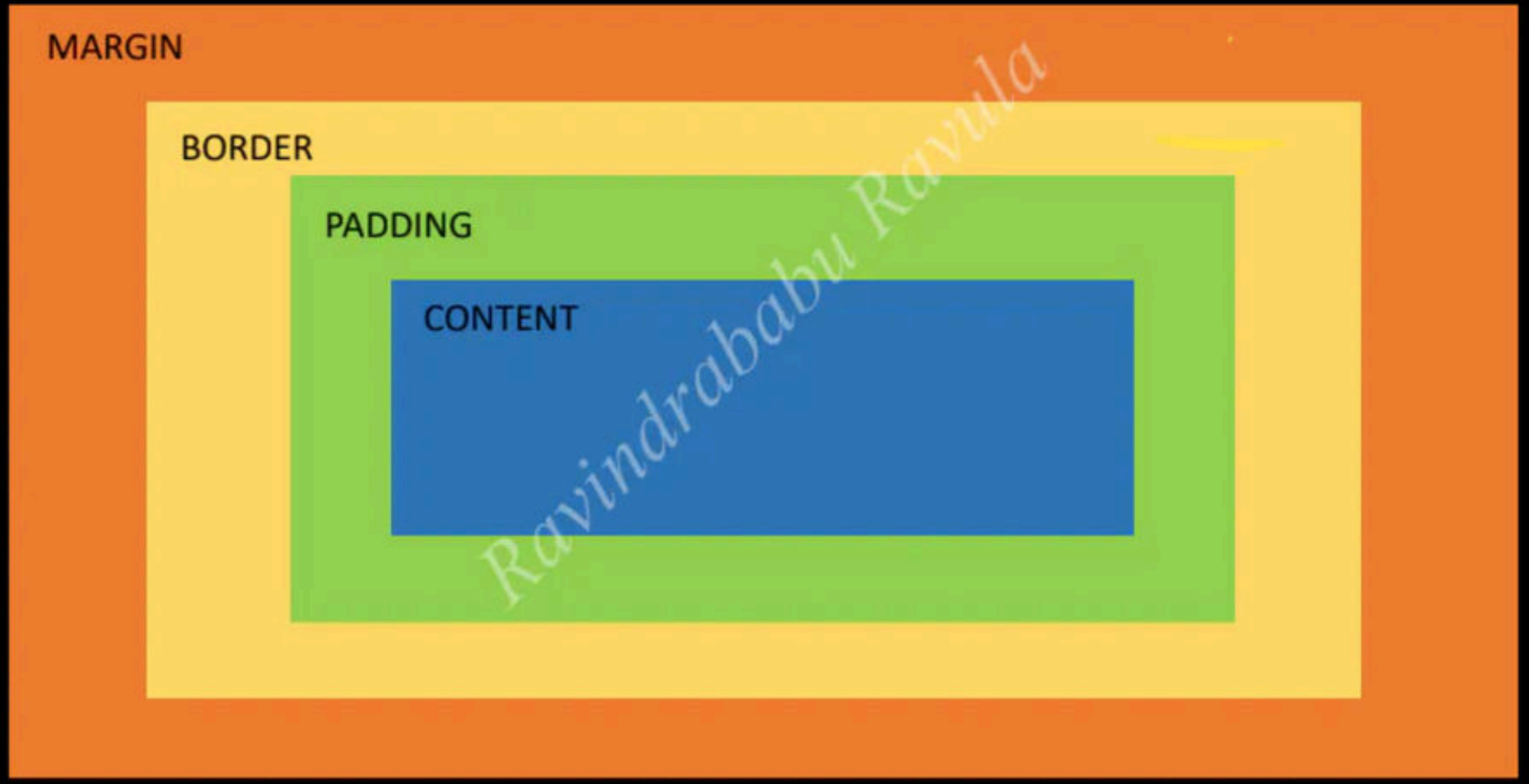
Margin:

The outermost part is the margin between this HTML element to other HTML elements.

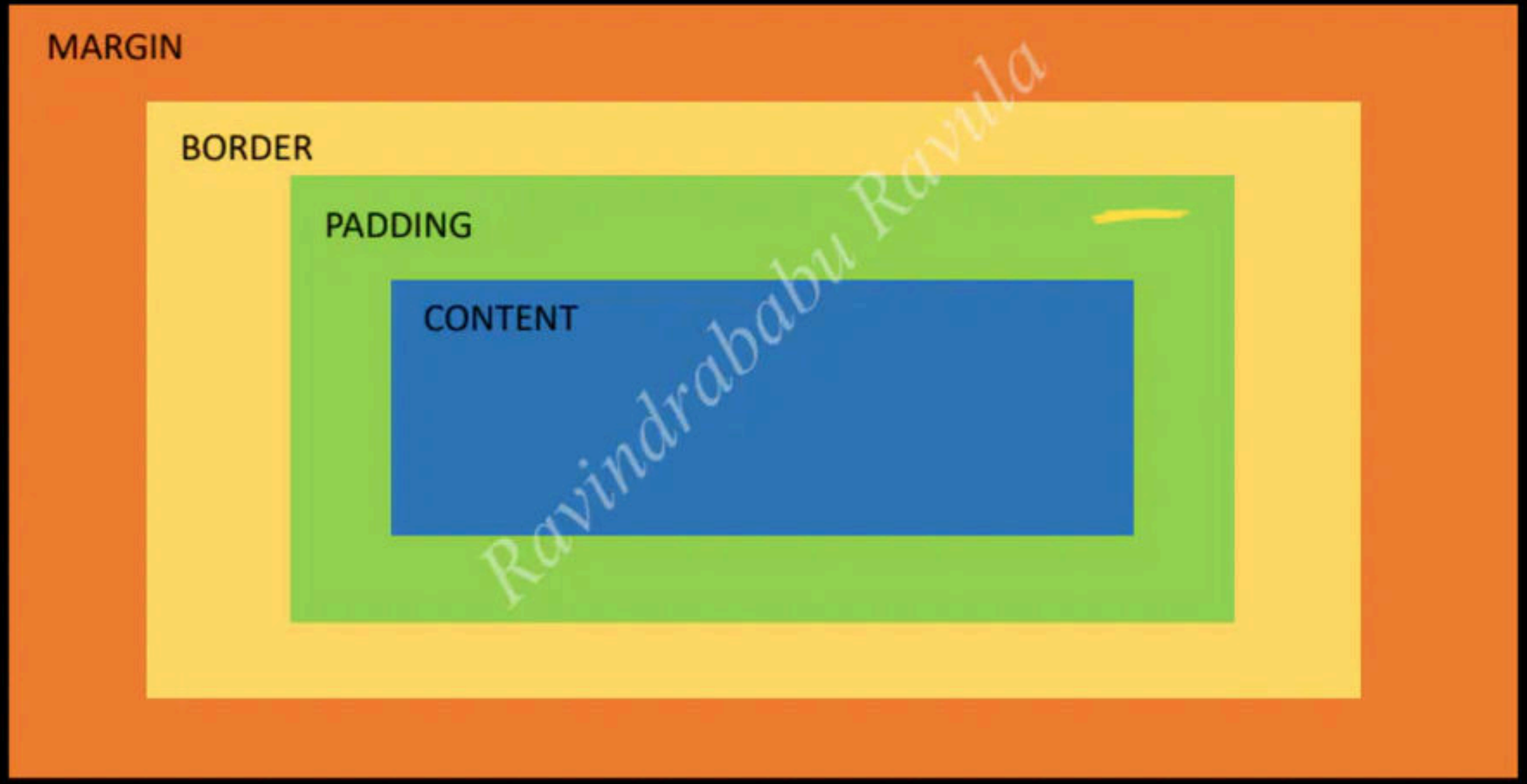


Border: The second part is the border.

The border sits inside the margin, and surrounds the padding and content of the HTML element.

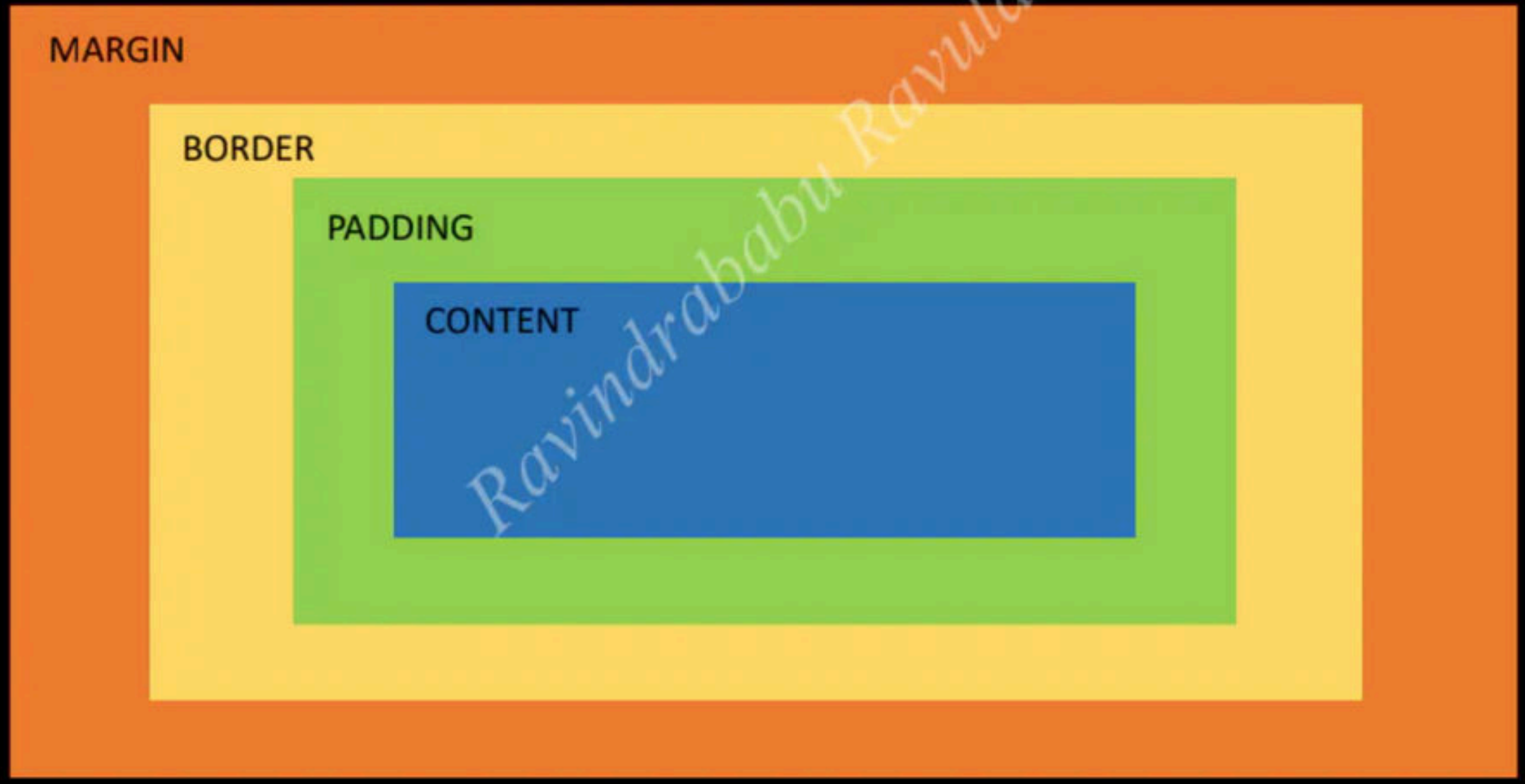


Padding: The third part is the padding. The padding sits inside the border and surrounds the content of the HTML element.



Content: The fourth part is the content. The content of an HTML element is whatever is displayed inside the HTML element. This is typically a combination of text, images and other HTML elements.

The margin, border and padding can be controlled via CSS properties.



CSS Margins:

```
#div1 {
    margin-top    : 20px;
    margin-right  : 10px;
    margin-bottom : 20px;
    margin-left   : 10px;
}
```

CSS Padding:

```
#div2 {
    padding-top    : 20px;
    padding-right  : 10px;
    padding-bottom : 20px;
    padding-left   : 10px;
}
```

CSS Border:

The border CSS property sets all four borders (top, right, bottom and left). The border CSS property value consists of three parts:

border width
border style
border color
border radius

```
#mydiv {
    border : 1px solid #000000;
}
```

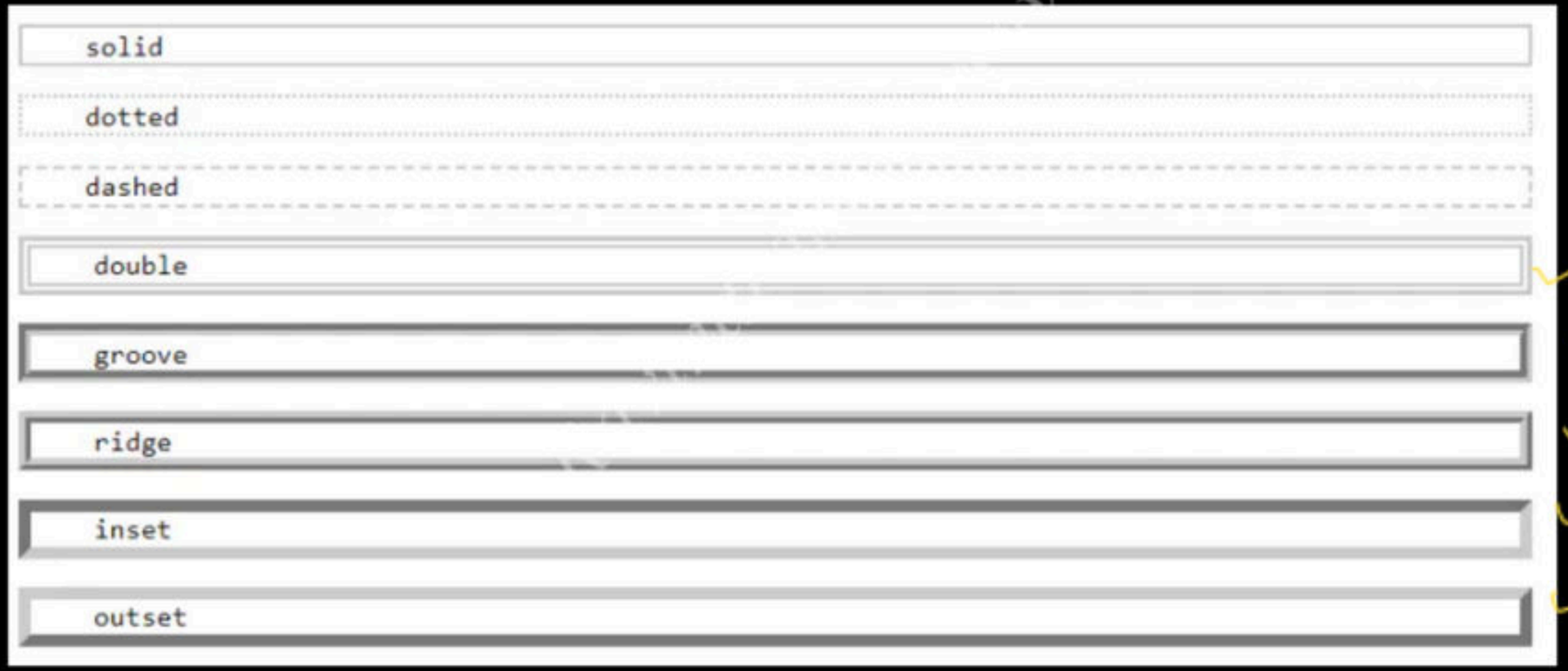
The first value is 1px which is the border width.

The second value is solid which is the border style.

The third value is #000000 which is the border color. This value sets the border color to black. Border colors are specified using the CSS colors formats.

This value sets the border to be solid, meaning a simple line. There are many styles :

Solid, dotted, dashed, double, groove, ridge, inset, outset, none, hidden



The border-radius CSS property is used to create rounded corners of borders on HTML elements.

```
#mydiv {
   border        : 1px solid #cccccc;
   border-radius : 10px;
   padding       : 10px;
}
```

This `div` element has rounded corners.

The content box can have its width and height set.
You control the width and height of an HTML element box using the width and height CSS properties. Width and height can be specified using any of the standard CSS units.

```
#theDiv {
   width  : 300px;
   height : 200px;
}
```

CSS Units

Unit	Description
px	Pixels

Unit	Description
px	Pixels
%	Percentage of parent element's dimension (or browser window dimension for body element), e.g. 25%

Unit	Description
px	Pixels
%	Percentage of parent element's dimension (or browser window dimension for body element), e.g. 25%
em	Dimension is based on standard font size in the browser. 1.0em = standard size.

Unit	Description
px	Pixels
%	Percentage of parent element's dimension (or browser window dimension for body element), e.g. 25%
em	Dimension is based on standard font size in the browser. 1.0em = standard size.
cm	Centimeters

Unit	Description
px	Pixels
%	Percentage of parent element's dimension (or browser window dimension for body element), e.g. 25%
em	Dimension is based on standard font size in the browser. 1.0em = standard size.
cm	Centimeters
mm	Millimeters

Unit	Description
px	Pixels
%	Percentage of parent element's dimension (or browser window dimension for body element), e.g. 25%
em	Dimension is based on standard font size in the browser. 1.0em = standard size.
cm	Centimeters
mm	Millimeters
in	Inches

Unit	Description
px	Pixels
%	Percentage of parent element's dimension (or browser window dimension for body element), e.g. 25%
em	Dimension is based on standard font size in the browser. 1.0em = standard size.
cm	Centimeters
mm	Millimeters
in	Inches
pt	Point - equal to 1/72 of an inch (because of the previous standard for computer monitors was 72 dpi).

Unit	Description
px	Pixels
%	Percentage of parent element's dimension (or browser window dimension for body element), e.g. 25%
em	Dimension is based on standard font size in the browser. 1.0em = standard size.
cm	Centimeters
mm	Millimeters
in	Inches
pt	Point - equal to 1/72 of an inch (because of the previous standard for computer monitors was 72 dpi).
pc	Pica - equal to 12 points.

CSS Text Styling

Font-family:

The font-family CSS property is used to specify the font family used by an HTML element containing text.

```
p {
    font-family: Arial;
}
```

Font-size: The font-size CSS property sets the size of the rendered text.

```
p {
    font-size: 20px;
}
```

Color: The color CSS property sets the color of the text rendered inside an HTML element.

```
p {
    color : #333333;
}
```

Font-style: The font-style CSS property can set the style of a font to one of three different values: **normal, italic, oblique.**

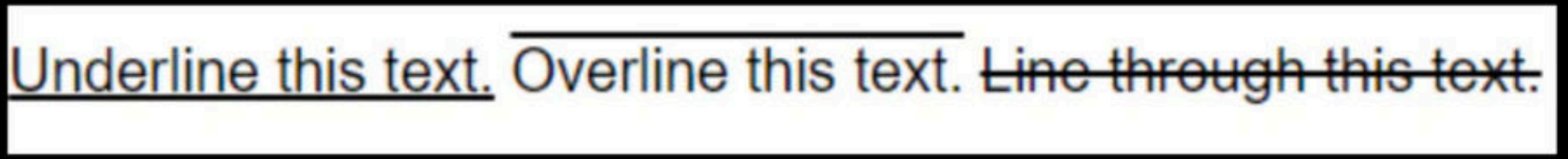
```
p {
    font-style: normal;
}

p#p1 {
    font-style: italic;
}
p#p2 {
    font-style: oblique;
}
```

Font-weight: The font-weight CSS property sets the thickness of text rendered inside an HTML element.
The font-weight CSS property can take the following values: normal, bold, bolder, lighter.

```
div#divid1 {
    font-weight: bold;
}
```

Text-decoration: The text-decoration CSS property can be used to underline, overline or strike-through text etc.

```
div#id1 {
   text-decoration: none;
}
div#id2 {
   text-decoration: underline;
}
div#id3 {
   text-decoration: overline;
}
div#id4 {
   text-decoration: overline;
}
```

Text-align: The text-align CSS property can align the text inside an HTML element to the left, right, center or justify. (Same as microsoft word

```
p#alignleft {
    text-align: left;
}
p#alignright {
    text-align: right;
}
p#aligncenter {
    text-align: center;
}

p#alignjustify {
    text-align: justify;
}
```

CSS Link Styling

Text links can be styled to set the font-family, font-size, font-weight, color, text-decoration etc.

```
a {
    font-family: Helvetica;
    font-weight: bold;
    color      : #000099;
}
```

Other properties:

```
a:link    { color: #00ff00; } // Link color
a:visited { color: #009900; } // Visited link color
a:hover   { color: #66ff66; } // Oh mouse over color
a:active  { color: #ffff00; } // Active link color
```

You can style a link to look like a button. You do so by setting the border, background-color, color and padding CSS properties, in addition to the text properties.

```
a.greenButton {
    border: 2px solid #006600;
    background-color: #009900;
    color: #ffffff;
    text-decoration: none;
    margin : 20px;
    padding: 10px 20px 10px 20px;
    display: inline-block;
}
a.greenButton:hover {
    border: 2px solid #009900;
    background-color: #00cc00;
}
```

CSS Form Styling

Text Fields

Text fields are input elements with the type attribute set to text.

```
<input type="text">
```

To style all text fields (<input type="text" >) you need to use a combination of the element and attribute CSS selectors.For example:

```
input[type='text'] {
    /* set CSS properties here*/
}
```

Text Field Borders

It is possible to set the border style of a text field.

```
input[type='text'] {
    border : 1px solid #cccccc;
}
```

Text Field Background Color

You can also set the background-color of a text field.

```
input[type='text'] {
    border           : 1px solid #cccccc;
    background-color : #e0e0e0;
}
```

Text Field Width and Height

You can also set the width and height of a text field using the width and height CSS properties.

```
input[type='text'] {
    width: 50px;
    height: 10px;
}
input[type='text'] {
    width: 100px;
    height: 15px;
}
input[type='text'] {
    width: 150px;
    height: 20px;
}
```

Text Field Font

It is possible to set the font family and font size used by a text field.:

```
input[type='text'] {
    font-family: Times;
    font-size  : 24px;
}
```

Buttons

Buttons are input elements with the type set to button or submit, or button elements.

```
<input type="button" value="A Button">
<input type="submit" value="A Submit Button">
<button>A Button</button>
```

Button CSS Selectors

```
input[type='button'] {  }
input[type='submit'] {  }
button {  }
```

Button Styles

You can set borders, background colors, fonts and much more on buttons.

```
button {
    font-family      : Helvetica;
    font-size        : 24px;
    color            : #ffffff;
    background-color : #009900;
    border           : 3px solid #006600;
    padding          : 10px;
}
```

Labels

You can also style the label element, which normally contains the label for a given input field. A common way to style label elements is to give them all the same width. That way labels and fields are displayed nicely in a two-column layout. For example:

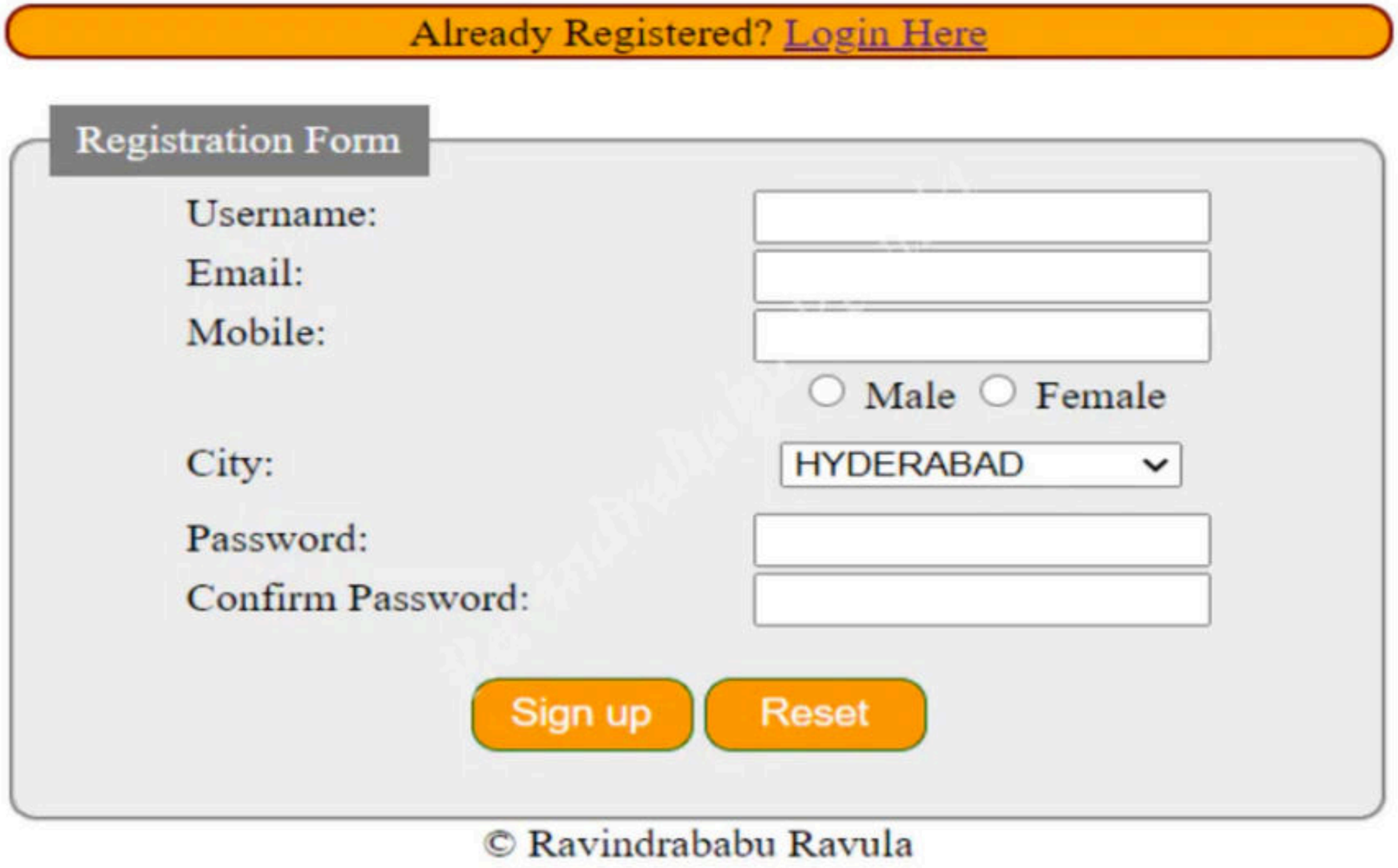
```
<style>
    label {
        display: inline-block;
        width  : 200px;
    }
</style>

<label>Name</label> <input type="text"> <br>
<label>Address</label> <input type="text">
```

fieldset and legend:

The fieldset and legend HTML elements are used to draw a box around a form, and write a legend in the top of the box (inside the border). Here is a code example:

```
<fieldset>
   <legend>MY  Form</legend>

   <label style="display: inline-block; width: 150px;">First name</label>
   <input type="text"><br/>
   <label style="display: inline-block; width: 150px;">Last name</label>
   <input type="text"><br/>

</fieldset>
```

User-registration.html



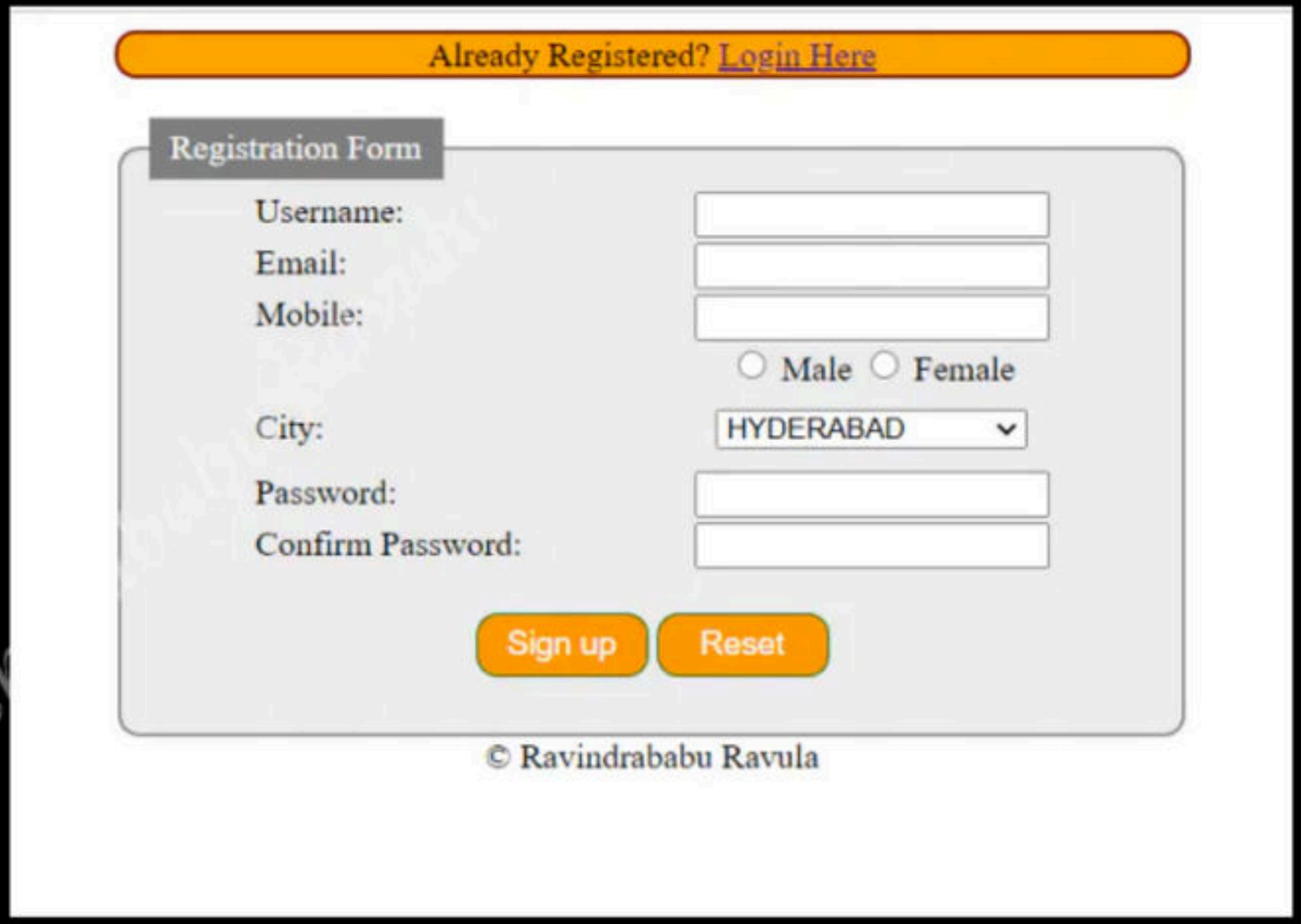
HTML Code:

```
<BODY>
    <center>
```

Align everything in the center of page.

CSS code:

```
body {
   background-color: white;
}
```



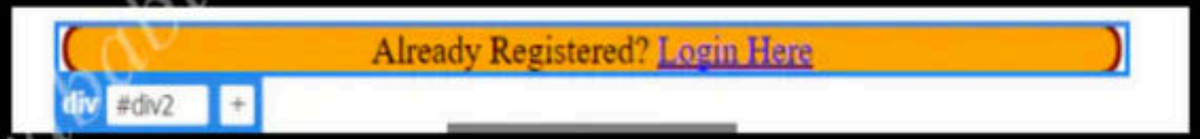
HTML Code:

```
<div id="div1">
   <div id="div2">
      Already Registered?
         <a href="login.html">Login Here</a>
    </div>
    <br>
<div id="div3">
```

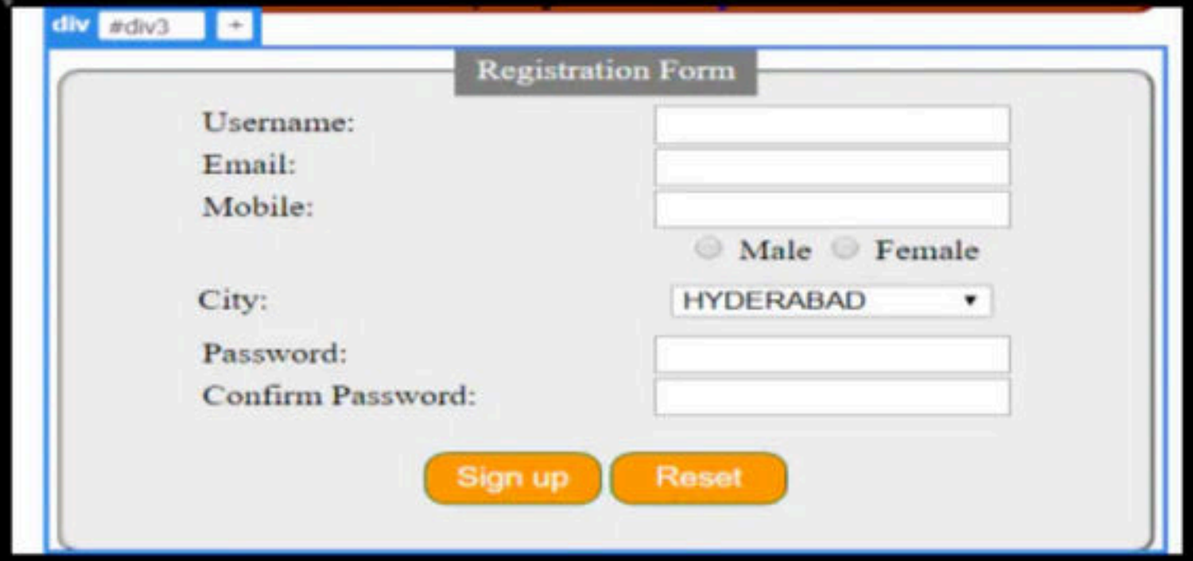
CSS Code:

```
#div1 {
   width: 500px;
}
#div2 {
   border: 2px solid maroon;
   border-radius : 10px;
   background-color: orange;
}
#div3 {
   border-radius : 10px;
   background-color: white;
}
```

Div1



Div2



Div3

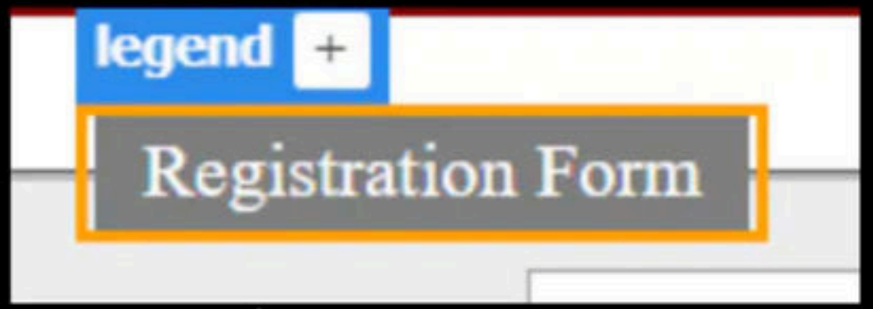
HTML Code:

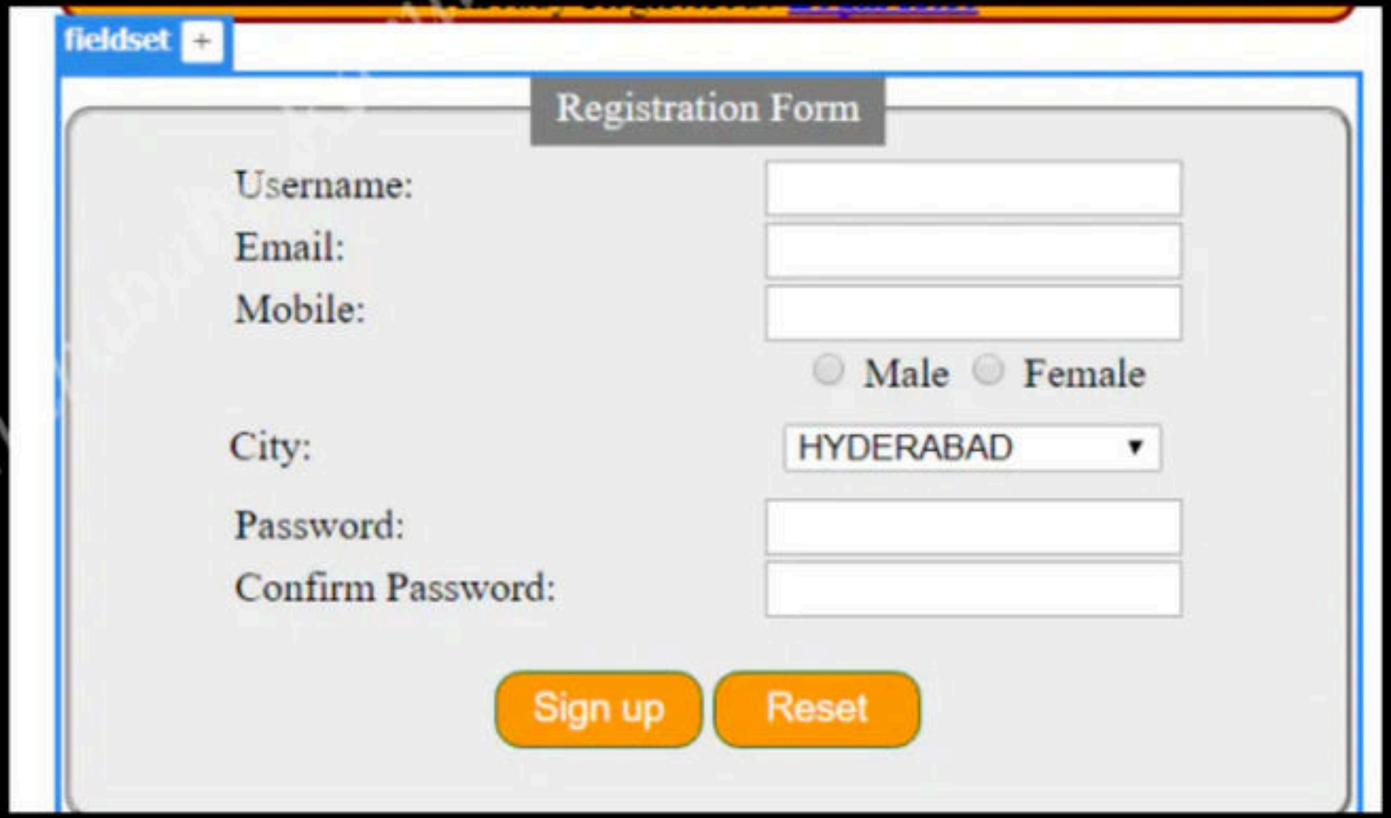
```
<fieldset>
    <legend>Registration Form</legend>
```

```
fieldset {
  background-color: #eeeeee;
  border-radius : 10px;

}
```

CSS Code:

```
legend {
  background-color: gray;
  color: white;
  padding: 5px 10px;
}
```



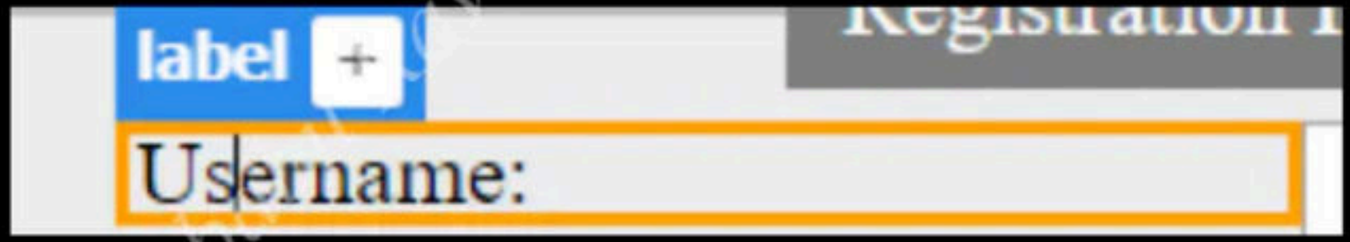


HTML Code:

```
<label>Username: </label>
   <input  type="text" name="username"
id="username"><br/>
```

CSS Code:

```
legend {
 background-color: gray;
 color: white;
 padding: 5px 10px;
}
```

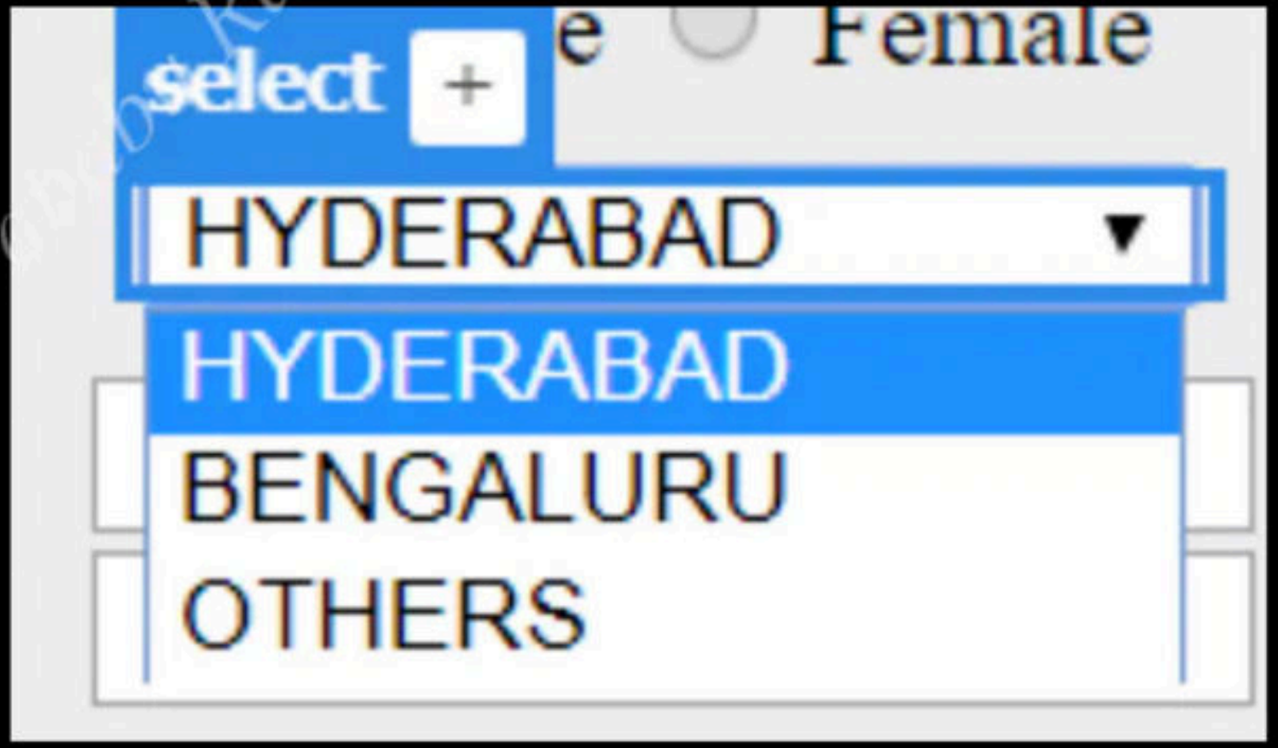


HTML Code:

```
<select>
<option value="HYD">HYDERABAD</option>
<option value="BLR">BENGALURU</option>
<option value="OTHER">OTHERS</option>
    </select><br/>
```

CSS Code:

```
select {
        width: 145px;
        margin: 10px;
}
```

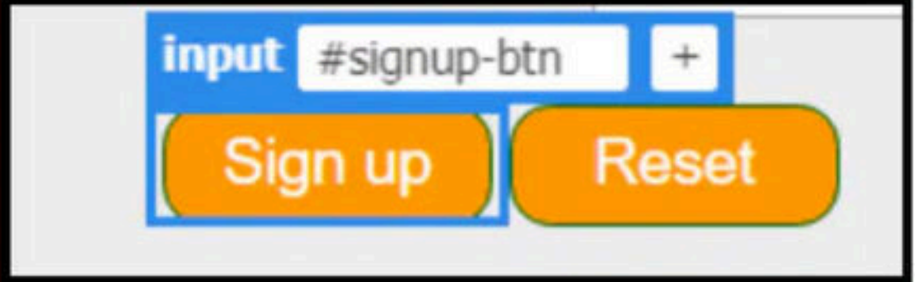
HTML Code:

```
<input type="submit" name="signup-btn"id="signup-btn" value="Sign up">
    <input type="reset" name="reset-btn"id="reset-btn" value="Reset">
```

CSS Code:

```
input[type='submit']{
    font-family      : Helvetica;
    font-size        : 15px;
    color            : #ffffff;
    background-color : #FF9900;
    border           : 1px solid #006600;
    padding          : 5px;
    border-radius : 10px;
    width: 80px;
}
```

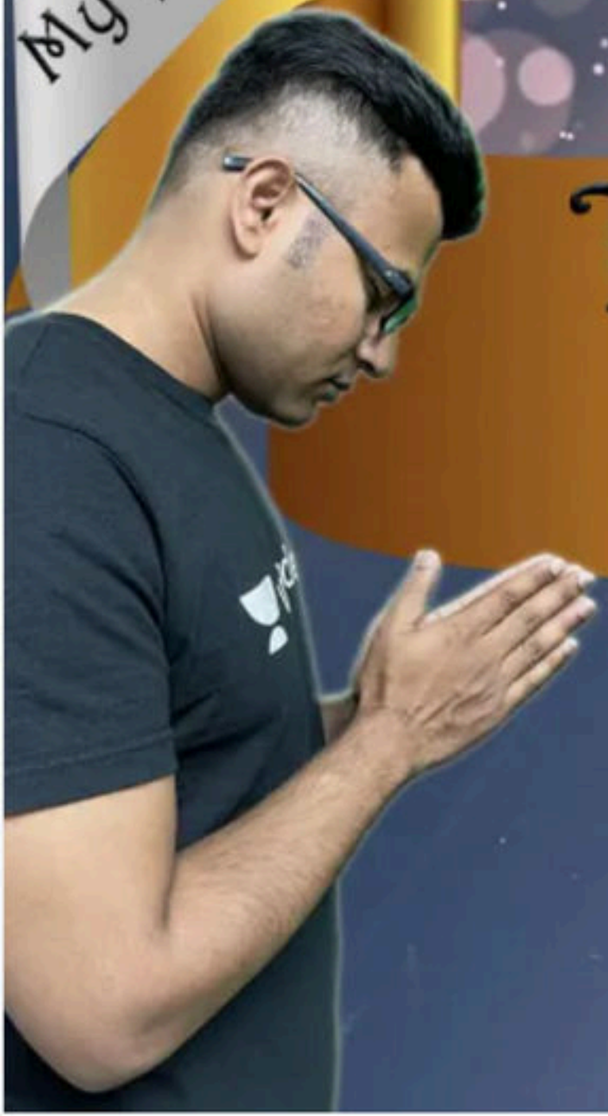
```
input[type='reset']{
    font-family      : Helvetica;
    font-size        : 15px;
    color            : #ffffff;
    background-color : #FF9900;
    border           : 1px solid #006600;
    padding          : 5px;
    border-radius : 10px;
    width: 80px;
}
```

My philosophy

TEACHING IS WORSHIP

STUDENTS ARE GODS



Introduction

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document.

You can either include the JavaScript code internally within your HTML document itself
Or in a separate external file and then point to that file from your HTML document.

Example: (Internal)

```
<html>
<head>
  <title>JavaScript code</title>
  <script type="text/javascript">
    // Create a Date Object
    var day = new Date();
  </script>
</head>
<body>

</body>
</html>
```

Example: (External)

HTML

```
<html>
<head>
  <title>My External JavaScript Code!!!
        </title>
<script type="text/javascript"
src="myjavascript.js">
  </script>
</head>
<body>
</body>
</html>
```

JS (myjavascript.js)

```
// JS code

var currentDate = new Date();
var day =
currentDate.getDate();
```

Basics

Comments: // is single line comment. /* */ is a multiline comment.

Variables:

Keyword var is used to declare a variable. JS is a loosely typed language. For example, a variable can be used to represent string as well as integer.

```
var x = 10;
var y = 20;
var z=x+y;
var z = “Hello”;
```

Operators:

Javascript supports all the operators used in C programming language.
The syntax is also the same.

IF, ELSE, SWITCH, FOR and WHILE loop:

The syntax of these constructs is the same as C programming language.

Arrays:

var arrayname=[value1,value2.....valueN];

```
<script>
var emp=["item1","item2","item3"];
for (i=0;i<emp.length;i++){
document.write(emp[i] +"<br/>");
}
</script>
```

Dynamic array:

var arrayname=new Array();

Example:

```
<script>
var i;
var emp = new Array();
emp[0] = "item1";
emp[1] = "item2";
emp[2] = "item3";

for (i=0;i<emp.length;i++){
document.write(emp[i] + "<br>");
}
</script>
```

Functions:

The function keyword is used to define the function.

function fun_name(parameters...) { }

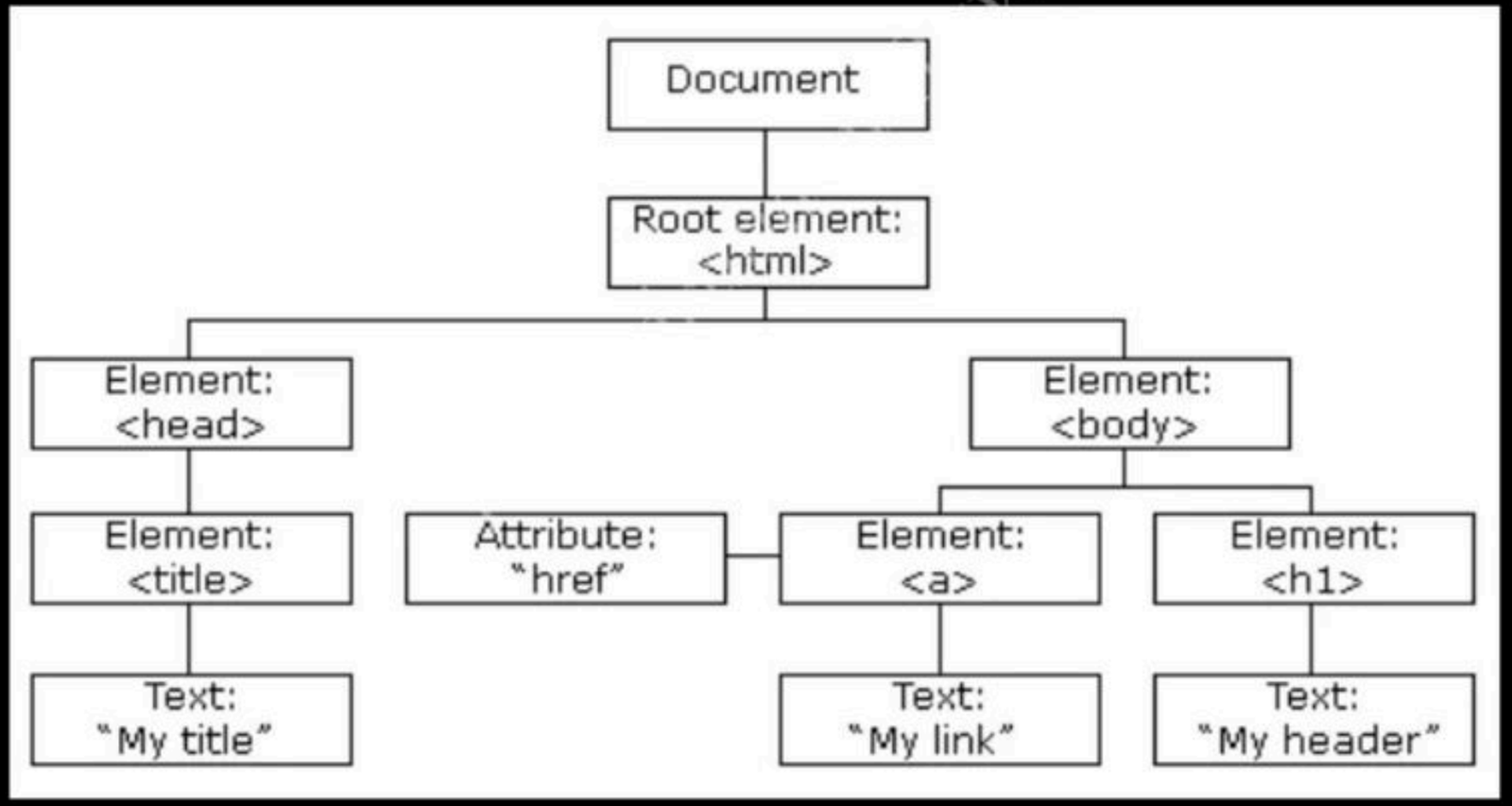
Full example1:

```
<html>
<body>
<script>
function msg(){
//alert is pop-up to display a message.
alert("hello World");
}
</script>
//On click function msg is called.
<input type="button" onclick="msg()"
value="Click"/>
</body>
</html>
```

JavaScript DOM

When a web page is loaded, the browser creates a Document Object Model of the page.

The HTML DOM model is constructed as a tree of Objects:



With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes
- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page.

The value of an individual form field can be accessed and retrieved using the syntax

document.formName.fieldName.value
or document.getElementsByName(name).value

We can access and change the contents of document by its methods.:

Method	Description
write("string")	writes the given string on the document.
writeln("string")	writes the given string on the document with a newline character at the end.
getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByClassName()	returns all the elements having the given class name.

Full Example2:

```
<html>
<head>
<script type="text/javascript">
function printname(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>
</head>
<body>
<form name="form1">
Enter Your Name:<input type="text" name="name"/>
<input type="button" onclick="printname()" value="Click here"/>
</form>
</body>
```


Note: value is the property that returns the value of the input text.

Full Example3:

```
<html>
<head>
<script type="text/javascript">
function square(){
var number=document.getElementById("number").value;
alert("The Square is: "+number*number);
}
</script>
<body>
<form>
Enter a Number:<input type="text" id="number"
name="number"/><br/>
<input type="button" value="Get Square"
onclick="square()"/>
</form>
</body>
```

Full Example4: (changing CSS)

```
<html>
<head>
<script>
function changeColor(){
document.getElementById("p2").style.color = "blue";
}
</script>
</head>
<body>

<p id="p2">Hello World!</p>
<form>
<input type="button" value="Change Color"
onclick="changeColor()" >
</form>

</body>
</html>
```

JavaScript Events

The change in the state of an object is known as an Event.
Various events which represent that some activity is performed by the user or by the browser.
When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling.
JavaScript handles the HTML events via Event Handlers.

Some of the common events are :

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Full Example5:

```
<html>
<body>

<button
onclick="document.getElementById('demo').innerHTML=Date()">
Get Time!</button>

<p id="demo"></p>

</body>
</html>
```