Unit:-5

Internet programming - Hypertext linking, forms, anchors and image, perl programming scalar data, scalar variable strings, operators, arrays hashes and pattern matching, Introduction to CGI using PERL, Overview of JAVA script.

Internet programming

Internet programming involves creating dynamic websites that are interactive and user-friendly. This includes the use of databases, server-side scripting and client-side scripting to create applications that can process data, display content, and interact with users.

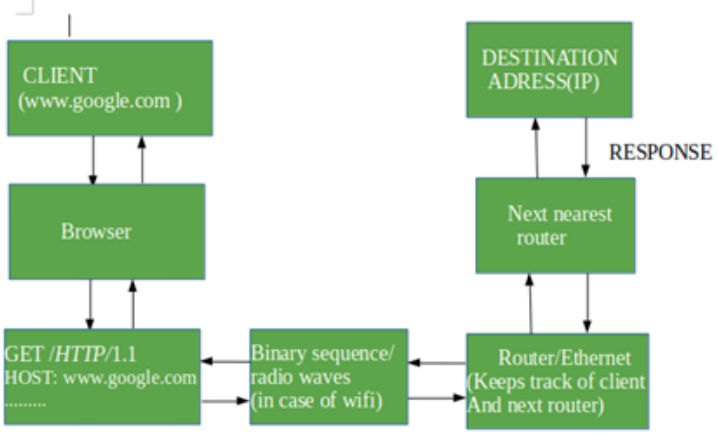
Web programming refers to the development of web applications and websites that are accessed over the Internet. Web programming involves creating web pages, web applications, and other online content that can be displayed in a web browser. Web programming is accomplished using a variety of programming languages including HTML, CSS, JavaScript, PHP, Python, Ruby and Java. Each of these languages has its strengths and weaknesses, and the choice of language depends on the needs of the project.

How Does Internet Programming Work:-

1) **Client-side**:- First, when we type a URL like www.google.com, the browser converts it into a file containing:-

- GET/HTTP/1.1(where GET means we are requesting some data from the server and HTTP refers to a protocol that we are using, 1.1 refers to the version of the HTTP request)
- Host:- www.google.com
- And some other information

Now this file is converted to binary code by the browser and it is sent down the wires if we are connected through Ethernet and if we are using WiFi, first it converts it to a radio signal which is decoded by a router in a very low level. It is converted to binary and then sent to the servers. This information or 'binary codes' go to the destination and responds if it is received by the sender only because of the IP address. One router will send the information to another and this keeps on going until the binary codes reach the destination.

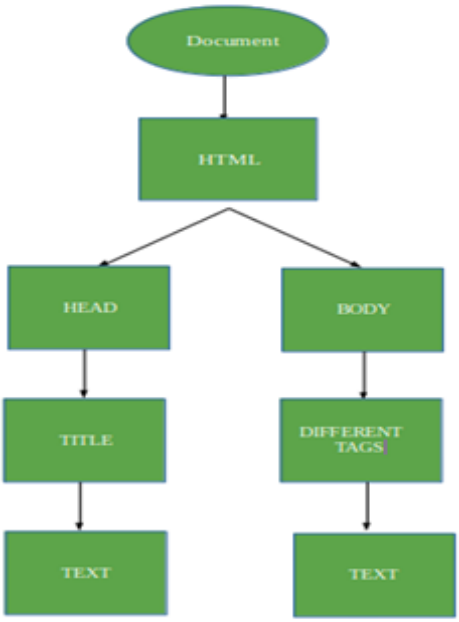


2) **Server-side**:- Now the server receives the binary code and decodes it and sends the response in the following manner:-

- HTTP/1.1 200 ok(where 200 ok is the status)
- Content-type:type/HTML
- Body of page

Now, this is converted back to binary by the server and sent to the IP address that is requesting it. Once the codes are received by the client, the browser again decodes the information in the following way:-

- First, it checks the status
- It starts reading the document from the HTML tag and constructs a Tree-like structure.
- The HTML tree is then converted to corresponding binary code and returned on the screen.
- In the end, we see the website front-end.



3) **Protocols**:- HTTP(Hypertext Transfer Protocol) is the primary protocol used for communication on the web. Clients send HTTP requests to servers, which respond with HTTP responses. HTTPS(HTTP Secure) is a secure version of HTTP that encrypts data transferred between the client and server.

Uses of Internet and Web programming:-

- **Communication**:- The Internet has reformed communication, allowing people to connect with each other through email, social media, video conferencing and instant messaging.
- **Information sharing**:- The Internet has made it possible to access very big amounts of information quickly and easily. Websites like Wikipedia and news sites provide up-to-date information on a wide range of topics.
- **E-commerce**:- The Internet has enabled businesses to sell products and services online, creating new opportunities for entrepreneurs and small businesses.
- **Education**:- The Internet has opened up new opportunities for education, making it possible for people to learn online through webinars and other online courses.
- **Entertainment**:- The Internet has transformed the way we consume entertainment, with streaming services like Netflix and YouTube providing access to movies, TV shows and other content.

Issues in Internet and Web programming:-

- **Security**:- Security is a critical concern in web programming, as hackers can exploit vulnerabilities in web applications to gain unauthorized access to sensitive data or cause damage. Developers need to implement strong security measures to protect against these threats.
- **Compatibility**:- The Internet and web programming involve a wide range of devices, browsers and operating systems. Ensuring compatibility across all of these platforms can be a important challenge for developers.
- **Performance**:- Web applications need to be responsive and perform well, even under heavy loads. This requires careful optimization of code, server infrastructures and other resources.
- **Accessibility**:- Web applications need to be accessible to people with disabilities, including those who use assistive technologies like screen readers or voice recognition software.
- **Privacy**:- As web applications collect and process user data, privacy concerns have become increasingly important. Developers need to implement strong privacy policies and ensure that user data is protected.
- **Usability**:- Web applications need to be easy to use and handling, with simple interfaces that provide a positive user experience.

Security Mechanism Used in Internet and Web Programming:-

- **Encryption**:- This security mechanism deals with hiding and covering of data which helps data to become confidential. It is achieved by applying mathematical algorithms which reconstruct information into not readable form. It is achieved by Cryptography.
- **Access Control**:- It can be achieved by various techniques such as applying passwords, using firewall or just by adding PIN to data.
- **Bit Stuffing**:- This security mechanism is used to add some extra bits into data which is being transmitted. It helps data to be checked at the receiving end and is achieved by Even parity or Odd Parity.
- **Digital Signature**:- This security mechanism is achieved by adding digital data that is not visible. It is form of electronic signature which is added by sender which is checked by receiver electronically. This mechanism is used to protect data which is not more confidential but sender's identity is to be notified.
- **Certification**:- This security mechanism involves use of trusted third party in communication. It acts as intermediate between sender and receiver so that if any chance of conflict is reduced.
- **Authentication**:- This is achieved at the TCP(Transmission Control Protocol)/IP(Internet Protocol) layer where 3-way handshaking mechanism is used to ensure data is sent or not.

Hypertext linking

HTML(Hyper Text Markup Language) Links, also known as Hyperlinks, are used to connect one web page to another, allowing users to move easily between different pages, websites or sections within the same page.

- The <a>(anchor) tag creates hyperlinks, using the href attribute to specify the destination URL.
- It can link text, images or buttons for move.
- Links can open in the same tab or a new tab using the target attribute, and other common attributes include title for additional information.

```
<html>
<head>
  <title>HTML Links</title>
</head>
<body>
  <p>Click on the following link</p>
  <a href="https://www.google.com/">
    Google
  </a>
</body>
</html>
```

Output:-

Click on the following link

Google

By default, links will appear as follows in all browsers:-

- An unvisited link is underlined and blue.
- A visited link is underlined and purple.
- An active link is underlined and red.

HTML Links - Target Attribute:- The target attribute in the <a> tag specifies where to open the linked document. It controls whether the link opens in the same window, a new window, or a specific frame.

```
<html>
<head>
  <title>Target Attribute Example</title>
</head>
<body>
  <h3>
    Various options available in the Target Attribute
  </h3>
  <p>
    If you set the target attribute to "_blank", the link will open in a new browser window or tab.
```

```
    </p>
    <a href="https://www.google.com/"
       target="_blank">
       Google
    </a>
    <p>
       If you set the target attribute to "_self", the link will open in the same window or tab.
    </p>
    <a href="https://www.google.com/"
       target="_self">
       Google
    </a>
    <p>
       If you set the target attribute to "_top", the link will open in the full body of the window.
    </p>
    <a href="https://www.google.com/"
       target="_top">
       Google
    </a>
    <p>
       If you set the target attribute to "_parent", the link will open in the parent frame.
    </p>
    <a href="https://www.google.com/"
       target="_parent">
       Google
    </a>
</body>
</html>
```

Output:-

Various options available in the Target Attribute

If you set the target attribute to "_blank", the link will open in a new browser window or tab. Google

If you set the target attribute to "_self", the link will open in the same window or tab. Google

If you set the target attribute to "_top", the link will open in the full body of the window. Google

If you set the target attribute to "_parent", the link will open in the parent frame. Google

Attribute	Description

_blank	Opens the linked document in a new window or tab
_self	Opens the linked document in the same frame or window as the link(Default behavior)
_parent	Opens the linked document in the parent frame
_top	Opens the linked document in the full body of the window
Framename	Opens the linked document in a specified frame. The frame's name is specified in the attribute

Linking Different HTML Elements:-

Element to Interlink	Specific Code
Linking to an image	<img src="image.jpg" alt="Image">
Link to an Email Address	Send Email
Phone Number	Call Now
Button	 <button>Visit Example</button>
Link to Download File	Download File
Link Title	Link Text

Forms

An HTML form is used to collect user input. The user input is most often sent to a server for processing. HTML forms, defined using the <form> tag, are essential for collecting user input on web pages. They include interactive controls like text fields, emails, passwords, checkboxes, radios and buttons.

- Widely used, over 85% of websites trust on forms to gather user data.
- They play a crucial role in modern web development by enabling user interaction and data submission.

```
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
      content="width=device-width, initial-scale=1.0">
   <title>HTML</title>
</head>
<body>
   <h2>HTML Forms</h2>
   <form>
      <label for="username">Username:</label><br>
      <input type="text" id="username" name="username"><br><br>
      <label for="password">Password:</label><br>
```

```
    <input type="password" id="password" name="password"><br><br>
    <input type="submit" value="Submit">
  </form>
</body>
```

Output:-

- The code has a basic HTML structure with a title "HTML Forms".
- The <h2> tag displays "HTML Forms" as the main heading on the page.
- The <form> tag defines a form for user input.
- A text input field for the username with a label.
- A password input field and a submit button to send the form data.

The <form> Element:- The HTML <form> element is used to create an HTML form for user input:-

```
<form>
.
form elements
.
</form>
```

The <form> element is a container for different types of input elements, such as text fields, checkboxes, radio buttons, submit buttons etc.

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

Examples:-

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="checkbox">	Displays a checkbox(for selecting zero or more of many choices)
<input type="submit">	Displays a submit button(for submitting the form)
<input type="button">	Displays a clickable button

Text Fields:- The <input type="text"> defines a single-line input field for text input.

Example:-

```
<form>
  <label for="fname">First name:</label><br>
```

```
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

Output:-

First name:

Last name:

The <label> Element:- The <label> tag defines a label for many form elements. The <label> element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focuses on the input element. The <label> element also helps users who have difficulty clicking on very small regions(such as radio buttons or checkboxes) - because when the user clicks the text within the <label> element, it converts the radio button/checkbox. The for attribute of the <label> tag should be equal to the id attribute of the <input> element to bind them together.

Radio Buttons:- The <input type="radio"> defines a radio button. Radio buttons let a user select ONE of a limited number of choices.

Example:-

```
<p>Choose your favorite Web language:</p>
<form>
  <input type="radio" id="html" name="fav_language" value="HTML">
  <label for="html">HTML</label><br>
  <input type="radio" id="css" name="fav_language" value="CSS">
  <label for="css">CSS</label><br>
  <input type="radio" id="javascript" name="fav_language" value="JavaScript">
  <label for="javascript">JavaScript</label>
</form>
```

Output:-

Choose your favorite Web language:

- HTML
- CSS
- Javascript

Checkboxes:- The <input type="checkbox"> defines a checkbox. Checkboxes let a user select ZERO or MORE options of a limited number of choices.

Example:-

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
```

```
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
  <label for="vehicle3"> I have a boat</label>
</form>
```

Output:-

□ I have a bike

□ I have a car

□ I have a boat

The Submit Button:- The <input type="submit"> defines a button for submitting the form data to a form-handler. The form-handler is typically a file on the server with a script for processing input data. The form-handler is specified in the form's action attribute.

Example:-

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="SDCC"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="MZN"><br>
  <input type="submit" value="Submit">
</form>
```

Output:-

First name:

SDCC

Last name:

MZN

Submit

The Name Attribute for <input>:- Notice that each input field must have a name attribute to be submitted. If the name attribute is missed, the value of the input field will not be sent at all.

Example:- This example will not submit the value of the "First name" input field:-

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" value="John"><br><br>
  <input type="submit" value="Submit">
</form>
```

Form Elements:-

Elements	Descriptions
<label>	It defines labels for <form> elements
<input>	It is used to get input data from various types such as text, password, email etc by changing its type
<button>	It defines a clickable button to control other elements or execute a functionality

<select>	It is used to create a drop-down list
<textarea>	It is used to get input long text content
<fieldset>	It is used to draw a box around other form elements and group the related data
<legend>	It defines a caption for fieldset elements
<datalist>	It is used to specify pre-defined list options for input controls
<output>	It displays the output of performed calculations
<option>	It is used to define options in a drop-down list
<optgroup>	It is used to define group-related options in a drop-down list

Input Types in HTML Forms:-

Input Type	Description
<input type="text">	Defines a one-line text input field
<input type="password">	Defines a password field
<input type="submit">	Defines a submit button
<input type="reset">	Defines a reset button
<input type="radio">	Defines a radio button
<input type="email">	Validates that the input is a valid email address
<input type="number">	Allows the user to enter a number. You can specify min, max, and step attributes for range
<input type="checkbox">	Used for checkboxes where the user can select multiple options
<input type="date">	Allows the user to select a date from a calendar
<input type="time">	Allows the user to select a time
<input type="file">	Allows the user to select a file to upload

Anchors and Image

Anchor <a> Tag:- The <a> tag defines a hyperlink that connects one page or resource to another. Its key attribute, href, specifies the destination URL where users are directed upon clicking.

- Can link to web pages, email addresses, phone numbers, or sections within the same page.
- Supports attributes like target, rel, and download for increased functionality.

```
<html>
<body<a href="https://www.sdccmzn.com/Notices/" target="_blank">
   Notices
      </a>
</body>
</html>
```

Syntax:-

```
<a href = "link"> Link Name </a>
```

By default, links appear as follows in all browsers:-

- **Unvisited links**:- underlined and blue.
- **Visited links**:- underlined and purple.
- **Active links**:- underlined and red.

1) **Opening Links in New Tab**:- To open a link in a new browser Tab, add the target="_blank" attribute:-

   SDCC

2) **Linking to Email Addresses and Phone Numbers**:-

- To link to an email address:-

  Send email

- To link to a phone number:-

  +910000000

3) **Creating Internal Page Anchors**:- To link to another section on the same page:-

   Go to Section 1

4) **Executing JavaScript**:- To trigger JavaScript code:-

   Execute JavaScript

Attributes:-

Attributes	Description
charset	It specifies the character set. It is not supported by HTML 5
download	It is used to specify the target link to download when the user clicks
hreflang	It is used to specify the language of the linked document
media	It is used to specify the linked media
name	It is used to specify the anchor name. It is not supported by HTML 5 you can use the global id attribute instead
rel	It is used to specify the relation between the current document and the linked document
shape	It is used to specify the shape of the link. It is not supported by HTML 5
type	It is used to specify the type of links
target	It specifies the target link
rev	Specifies the relationship between the linked document and the current document. It is not supported by HTML 5

Image <img> Tag:- The HTML <img> tag is used to insert images in a web page. It is an empty or self-closing tag, meaning it doesn't have a closing tag. It allows to display images from various sources, such as files on a website or URLs from other websites.

- Supports local and external image sources.
- Common attributes include src, alt, width and height.
- Does not contain any inner content.

```
<html>
<head>
   <title>SDCC Logo</title>
```

```
</head>
<body style="text-align: center;">
   <h3>SDCC Logo</h3>
   <img
      src=https://media.sdccmzn.com/wp-content/uploads/sdccmzn-13.png
      width="420"
      height="100"
      alt="sdccmzn.com"
   >
</body>
</html>
```

Syntax:-

```
<img src="./image.png" alt="image description" width="200" height="100">
```

HTML image Tag Attributes:- HTML image tag attributes define how an image is sourced, displayed, loaded, and made accessible on a web page, helping improve responsiveness, performance and user experience.

1) **src Attribute:-** The src attribute defines the path or URL of the image that the browser fetches and displays inside the <img> tag. It is a mandatory attribute because, without it, the image cannot be displayed on the web page.

- It can point to a local image file(relative or absolute path) or an external image URL.
- If the src path is incorrect or the image is unavailable, the image will not load.
- Common image formats supported include JPG, PNG, GIF, SVG and WebP.
- It is usually used with the alt attribute to show alternate text when the image fails to load.

2) **alt Attribute**:- The alt attribute provides alternative text for an image, which is displayed when the image cannot be loaded and helps screen readers describe the image to visually impaired users.

- Improves accessibility by allowing screen readers to read the image description.
- Displays text when the image fails to load due to network or path issues.
- Helps search engines understand the content of the image for better SEO.

3) **crossorigin Attribute:-** The crossorigin attribute enables loading images from third-party domains with cross-origin permissions, allowing them to be safely used with the HTML <canvas> element.

- Saves the canvas from becoming ruined when using external images.
- Common values include unnamed and use-credentials.
- Required when performing canvas operations like drawing or exporting images from other domains.

4) **height and width Attributes**:- The height and width attributes define the displayed dimensions of an image on a web page, helping browsers reserve space before the image loads.

- Values are usually specified in pixels(e.g., width="200").

- Help prevent layout shifts while the image is loading.
- Can be used together with CSS(Cascading Style Sheets) for better responsive design control.

5) **ismap Attribute**:- The ismap attribute marks an image as a server-side image map, where click coordinates are sent to the server for processing.
- Used with an <a>(anchor) tag to make different image areas clickable.
- Sends the x and y coordinates of the click to the server as URL parameters.
- Commonly used in older web applications, now largely replaced by usemap.

6) **loading Attribute**:- The loading attribute controls when an image is loaded by the browser, either immediately or only when it is about to enter the viewport.
- Improves page performance by avoiding off-screen images.
- Common values are lazy and impatient.
- Reduces initial page load time.

7) **longdesc Attribute**:- The longdesc attribute provides a URL linking to a detailed description of an image for better accessibility.
- Helpful for complex images like charts or diagrams.
- Supports screen readers and assistive technologies.
- Acts as an extended version of alt text.

8) **referrerpolicy Attribute**:- The referrerpolicy attribute_defines how much advisor information is sent when fetching an image.
- Enhances privacy and security.
- Common values include no-advisor, origin.
- Controls data shared with external servers.

9) **sizes Attribute**:- The sizes attribute specifies how much space an image will take under different layout conditions.
- Used with srcset for responsive images.
- Helps browsers choose the best image size.
- Optimizes bandwidth usage.

10) **srcset Attribute**:- The srcset attribute provides multiple image sources for different screen resolutions or sizes.
- Enables responsive image loading.
- Improves performance on high-DPI(Dots Per Inch) displays.
- Allows browsers to select the most suitable image.

11) **usemap Attribute**:- The usemap attribute defines a client-side image map with clickable areas.
- Works with the <map> and <area> tags.
- Allows multiple interactive regions in one image.
- More flexible than server-side image maps.

Custom styling with image tag:- Custom styling with the <img> tag allows developers to enhance image appearance and behavior using CSS properties like size, borders, shadows, and hover(fly away) effects for better visual presentation.

```
<html>
<head>
  <title>Styled Image Example</title>
  <style>
    img {
      width: 200px;
      height: auto;
      border: 5px solid #4CAF50;
      border-radius: 10px;
      box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
      transition: transform 0.3s ease;
    }
    img:hover {
      transform: scale(1.1);
    }
  </style>
</head>
<body>
  <img
    src=https://media.sdccmzn.com/wp-content/uploads/sdcc-13.png
    alt="GFG Image"
  >
</body>
</html>
```

Perl programming scalar data

A scalar is a single unit of data. That data might be an integer number, floating point, a character, a string, a paragraph or an entire web page.

Example:-

```
#!/usr/bin/perl
$age = 25;              # An integer assignment
$name = "Abc Xyz";      # A string
$salary = 1445.50;      # A floating point
print "Age = $age\n";
print "Name = $name\n";
print "Salary = $salary\n";
```

Output:-

Age = 25

Name = Abc Xyz

Salary = 1445.5

Numeric Scalars:-

Example:−

```
#!/usr/bin/perl
$integer = 200;
$negative = -300;
$floating = 200.340;
$bigfloat = -1.2E-23;
# 377 octal, same as 255 decimal
$octal = 0377;
# FF hex, also 255 decimal
$hexa = 0xff;
print "integer = $integer\n";
print "negative = $negative\n";
print "floating = $floating\n";
print "bigfloat = $bigfloat\n";
print "octal = $octal\n";
print "hexa = $hexa\n";
```

Output:-

```
integer = 200
negative = -300
floating = 200.34
bigfloat = -1.2e-23
octal = 255
hexa = 255
```

Scalar Operations:-

```
#!/usr/bin/perl
$str = "hello" . "world";          # Concatenates strings.
$num = 5 + 10;                     # adds two numbers.
$mul = 4 * 5;                      # multiplies two numbers.
$mix = $str . $num;                # concatenates string and number.
print "str = $str\n";
print "num = $num\n";
print "mul = $mul\n";
print "mix = $mix\n";
```

Output:-

```
str = helloworld
num = 15
```

mul = 20
mix = helloworld15

Scalar variable strings

String Scalars:- Notice the difference between single quoted strings and double quoted strings.

Example:-

```
#!/usr/bin/perl
$var = "This is string scalar!";
$quote = 'I m inside single quote - $var';
$double = "This is inside single quote - $var";
$escape = "This example of escape -\tHello, World!";
print "var = $var\n";
print "quote = $quote\n";
print "double = $double\n";
print "escape = $escape\n";
```

Output:-

```
var = This is string scalar!
quote = I m inside single quote - $var
double = This is inside single quote - This is string scalar!
escape = This example of escape -       Hello, World
```

Multiline Strings:- If you want to introduce multiline strings into your programs, you can use the standard single quotes as below:−

```
#!/usr/bin/perl
$string = 'This is
a multiline
string';
print "$string\n";
```

Output:-

```
This is
a multiline
string
```

You can use "here" document syntax as well to store or print multilines as below:−

```
#!/usr/bin/perl
print <<EOF;
This is
a multiline
string
EOF
```

Output:-

This is
a multiline
string

Operators

- Arithmetic Operators
- Equality Operators
- Assignment Operators
- Bitwise Operators
- Logical Operators
- Quote-like Operators
- Miscellaneous Operators
- **Perl Arithmetic Operators**:-

Example:- Assume variable $a holds 10 and variable $b holds 20, then:-

Sr. No.	Operator & Description
1	+(Addition) Adds values on both side of the operator Example:- $a + $b will give 30
2	-(Subtraction) Subtracts right hand operand from left hand operand Example:- $a - $b will give -10
3	*(Multiplication) Multiplies values on both side of the operator Example:- $a * $b will give 200
4	/(Division) Divides left hand operand by right hand operand Example:- $b / $a will give 2
5	%(Modulus) Divides left hand operand by right hand operand and returns remainder Example:- $b % $a will give 0
6	**(Exponent) Performs exponential(power) calculation on operators Example:- $a**$b will give 10 to the power 20

- **Perl Equality Operators**:- These are also called relational operators.

Example:- Assume variable $a holds 10 and variable $b holds 20, then:-

Sr. No.	Operator & Description
1	==(equal to) Checks if the value of two operands are equal or not, if yes then condition becomes true

	Example:- ($a == $b) is not true
2	!=(not equal to) Checks if the value of two operands are equal or not, if values are not equal then condition becomes true Example:- ($a != $b) is true
3	<=> Checks if the value of two operands are equal or not, and returns -1, 0, or 1 depending on whether the left argument is numerically less than, equal to, or greater than the right argument Example:- ($a <=> $b) returns -1
4	>(greater than) Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true Example:- ($a > $b) is not true
5	<(less than) Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true Example:- ($a < $b) is true
6	>=(greater than or equal to) Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true Example:- ($a >= $b) is not true
7	<=(less than or equal to) Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true Example:- ($a <= $b) is true

Below is a list of equity operators.

Example:- Assume variable $a holds "abc" and variable $b holds "xyz", then:-

Sr. No.	Operator & Description
1	lt Returns true if the left argument is stringwise less than the right argument Example:- ($a lt $b) is true
2	gt Returns true if the left argument is stringwise greater than the right argument Example:- ($a gt $b) is false
3	le Returns true if the left argument is stringwise less than or equal to the right argument Example:- ($a le $b) is true
4	ge

	Returns true if the left argument is stringwise greater than or equal to the right argument Example:- ($a ge $b) is false
5	eq Returns true if the left argument is stringwise equal to the right argument Example:- ($a eq $b) is false
6	ne Returns true if the left argument is stringwise not equal to the right argument Example:- ($a ne $b) is true
7	cmp Returns -1, 0, or 1 depending on whether the left argument is stringwise less than, equal to, or greater than the right argument Example:- ($a cmp $b) is -1

- **Perl Assignment Operators**:-

Example:- Assume variable $a holds 10 and variable $b holds 20, then:-

Sr. No.	Operator & Description
1	= Simple assignment operator, Assigns values from right side operands to left side operand Example:- $c = $a + $b will assigned value of $a + $b into $c
2	+= Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand Example:- $c += $a is equivalent to $c = $c + $a
3	-= Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand Example:- $c -= $a is equivalent to $c = $c - $a
4	*= Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand Example:- $c *= $a is equivalent to $c = $c * $a
5	/= Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand Example:- $c /= $a is equivalent to $c = $c / $a
6	%= Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand Example:- $c %= $a is equivalent to $c = $c % a

7	**= Exponent AND assignment operator, Performs exponential (power) calculation on operators and assign value to the left operand Example:- $c **= $a is equivalent to $c = $c ** $a

- **Perl Bitwise Operators**:- Bitwise operator works on bits and perform bit by bit operation.

Example:- Assume if $a = 60; and $b = 13, then:-

$a = 0011 1100

$b = 0000 1101

Sr. No.	Operator & Description
1	& Binary AND Operator copies a bit to the result if it exists in both operands Example:- ($a & $b) will give 12 which is 0000 1100
2	| Binary OR Operator copies a bit if it exists in either operand Example:- ($a | $b) will give 61 which is 0011 1101
3	^ Binary XOR Operator copies the bit if it is set in one operand but not both Example:- ($a ^ $b) will give 49 which is 0011 0001
4	~ Binary Ones Complement Operator is unary and has the effect of 'flipping' bits Example:- (~$a) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number
5	<< Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand Example:- $a << 2 will give 240 which is 1111 0000
6	**>>** Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand Example:- $a **>>** 2 will give 15 which is 0000 1111

- **Perl Logical Operators**:-

Example:- Assume variable $a holds true and variable $b holds false, then:-

Sr. No.	Operator & Description
1	And Called Logical AND operator. If both the operands are true then condition becomes true Example:- ($a and $b) is false
2	&& C-style Logical AND operator copies a bit to the result if it exists in both operands Example:- ($a && $b) is false

3	Or Called Logical OR Operator. If any of the two operands are non zero then condition becomes true Example:- ($a or $b) is true
4	|| C-style Logical OR operator copies a bit if it exists in either operand Example - ($a || $b) is true
5	Not Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false Example:- not($a and $b) is true

- **Quote-like Operators**:-

Example:- In the following table, a {} represents any pair of sequences you choose.

Sr. No.	Operator & Description
1	q{ } Encloses a string with-in single quotes Example:- q{abcd} gives 'abcd'
2	qq{ } Encloses a string with-in double quotes Example:- qq{abcd} gives "abcd"
3	qx{ } Encloses a string with-in invert quotes Example - qx{abcd} gives `abcd`

- **Miscellaneous Operators**:-

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

Sr. No.	Operator & Description
1	. Binary operator dot(.) combines two strings Example:- If $a = "abc", $b = "def" then $a.$b will give "abcdef"
2	X The repetition operator x returns a string consisting of the left operand repeated the number of times specified by the right operand Example:- ('-' x 3) will give ---
3	.. The range operator .. returns a list of values counting (up by ones) from the left value to the right value Example:- (2..5) will give (2, 3, 4, 5)
4	++ Auto Increment operator increases integer value by one Example:- $a++ will give 11

5	-- Auto Decrement operator decreases integer value by one Example:- $a-- will give 9
6	->** The arrow operator is mostly used in retrieving a method or variable from an object or a class name Example:- $obj->$a is an example to access variable $a from object $obj

Arrays hashes and Pattern matching

Arrays:- An array is a variable that stores an ordered list of scalar values. Array variables are preceded by an "at"(@) sign. To refer to a single element of an array, you will use the dollar sign($) with the variable name followed by the index of the element in square brackets.

Example:-

```
#!/usr/bin/perl
&commat;ages = (25, 30, 40);
&commat;names = ("Ramu Garg", "Raju", "Shyam");
print "\$ages[0] = $ages[0]\n";
print "\$ages[1] = $ages[1]\n";
print "\$ages[2] = $ages[2]\n";
print "\$names[0] = $names[0]\n";
print "\$names[1] = $names[1]\n";
print "\$names[2] = $names[2]\n";
```

Here we have used the escape sign (\) before the $ sign just to print it. Other Perl will understand it as a variable and will print its value.

Output:-

```
$ages[0] = 25
$ages[1] = 30
$ages[2] = 40
$names[0] = Ramu Garg
$names[1] = Raju
$names[2] = Shyam
```

In Perl, List and Array terms are often used as if they're interchangeable. But the list is the data, and the array is the variable.

Array Creation:- Array variables are prefixed with the @ sign and are populated using either parentheses or the qw operator.

Example:-

```
&commat;array = (1, 2, 'Hello');
&commat;array = qw/This is an array/;
```

The second line uses the qw// operator, which returns a list of strings, separating the bounded string by white space. In this example, this leads to a four-element array; the first element is 'this' and last (fourth) is 'array'.

@days = qw/Monday

Tuesday

...

Sunday/;

You can also populate an array by assigning each value individually.

$array[0] = 'Monday';

...

$array[6] = 'Sunday';

Accessing Array Elements:- When accessing individual elements from an array, you must prefix the variable with a dollar sign ($) and then link the element index within the square brackets after the name of the variable.

Example:-

```
#!/usr/bin/perl
@days = qw/Mon Tue Wed Thu Fri Sat Sun/;
print "$days[0]\n";
print "$days[1]\n";
print "$days[2]\n";
print "$days[6]\n";
print "$days[-1]\n";
print "$days[-7]\n";
```

Output:-

```
Mon
Tue
Wed
Sun
Sun
Mon
```

Array indexes start from zero, so to access the first element you need to give 0 as indices. You can also give a negative index, in which case you select the element from the end, rather than the beginning, of the array.

```
print $days[-1]; # outputs Sun
print $days[-7]; # outputs Mon
```

Sequential Number Arrays:- Perl offers a shortcut for sequential numbers and letters. Rather than typing out each element when counting to 100.

Example:-

```
#!/usr/bin/perl
@var_10 = (1..10);
```

```
@var_20 = (10..20);
@var_abc = (a..z);
print "@var_10\n";        # Prints number from 1 to 10
print "@var_20\n";        # Prints number from 10 to 20
print "@var_abc\n";       # Prints number from a to z
```

Here double dot (..) is called range operator.

Output:-

```
1 2 3 4 5 6 7 8 9 10
10 11 12 13 14 15 16 17 18 19 20
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Array Size:- The size of an array can be determined using the scalar context on the array - the returned value will be the number of elements in the array.

```
&commat;array = (1,2,3);
print "Size: ",scalar &commat;array,"\n";
```

The value returned will always be the physical size of the array, not the number of valid elements. You can demonstrate this, and the difference between scalar @array and $#array, using this fragment.

```
#!/usr/bin/perl
@array = (1,2,3);
$array[50] = 4;
$size = @array;
$max_index = $#array;
print "Size:  $size\n";
print "Max Index: $max_index\n";
```

Output:-

```
Size: 51
Max Index: 50
```

There are only four elements in the array that contains information, but the array is 51 elements long, with a highest index of 50.

Adding and Removing Elements in Array:-

Sr. No.	Types & Description
1	**push @ARRAY, LIST** Adds the values of the list onto the end of the array
2	**pop @ARRAY** Pops off and returns the last value of the array
3	**shift @ARRAY** Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down
4	**unshift @ARRAY, LIST** Prefixes list to the front of the array, and returns the number of elements in the

	new array

```
#!/usr/bin/perl
# create a simple array
@coins = ("Quarter","Dime","Nickel");
print "1. \&commat;coins  = &commat;coins\n";
# add one element at the end of the array
push(&commat;coins, "Penny");
print "2. \&commat;coins  = &commat;coins\n";
# add one element at the beginning of the array
unshift(&commat;coins, "Dollar");
print "3. \&commat;coins  = &commat;coins\n";
# remove one element from the last of the array.
pop(&commat;coins);
print "4. \&commat;coins  = &commat;coins\n";
# remove one element from the beginning of the array.
shift(&commat;coins);
print "5. \&commat;coins  = &commat;coins\n";
```

Output:-

```
1. &commat;coins = Quarter Dime Nickel
2. &commat;coins = Quarter Dime Nickel Penny
3. &commat;coins = Dollar Quarter Dime Nickel Penny
4. &commat;coins = Dollar Quarter Dime Nickel
5. &commat;coins = Quarter Dime Nickel
```

Slicing Array Elements:- You can also extract a "slice" from an array - that is, you can select more than one item from an array in order to produce another array.

```
#!/usr/bin/perl
&commat;days = qw/Mon Tue Wed Thu Fri Sat Sun/;
&commat;weekdays = &commat;days[3,4,5];
print "&commat;weekdays\n";
```

Output:-

```
Thu Fri Sat
```

The specification for a slice must have a list of valid indices, either positive or negative, each separated by a comma. For speed, you can also use the .. range operator.

```
#!/usr/bin/perl
&commat;days = qw/Mon Tue Wed Thu Fri Sat Sun/;
&commat;weekdays = &commat;days[3..5];
print "&commat;weekdays\n";
```

Output:-

```
Thu Fri Sat
```

Replacing Array Elements:-

Syntax:-

splice @ARRAY, OFFSET [, LENGTH [, LIST]]

This function will remove the elements of @ARRAY designated by OFFSET and LENGTH, and replaces them with LIST, if specified. Finally, it returns the elements removed from the array.

Example:-

```
#!/usr/bin/perl
&commat;nums = (1..20);
print "Before - &commat;nums\n";
splice(&commat;nums, 5, 5, 21..25);
print "After - &commat;nums\n";
```

Output:-

```
Before - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
After - 1 2 3 4 5 21 22 23 24 25 11 12 13 14 15 16 17 18 19 20
```

Here, the actual replacement begins with the 6th number after that five elements are then replaced from 6 to 10 with the numbers 21, 22, 23, 24 and 25.

Transform Strings to Arrays:-

Syntax:-

split [PATTERN [, EXPR [, LIMIT]]]

This function splits a string into an array of strings, and returns it. If LIMIT is specified, splits into at most that number of fields. If PATTERN is omitted, splits on empty space.

Example:-

```
#!/usr/bin/perl
# define Strings
$var_string = "Rain-Drops-On-Roses-And-Whiskers-On-Kittens";
$var_names = "Larry,David,Roger,Ken,Michael,Tom";
# transform above strings into arrays.
@string = split('-', $var_string);
@names  = split(',', $var_names);
print "$string[3]\n";  # This will print Roses
print "$names[4]\n";    # This will print Michael
```

Output:-

```
Roses
Michael
```

Transform Arrays to Strings:- We can use the join() function to rejoin the array elements and form one long scalar string.

Syntax:-

join EXPR, LIST

This function joins the separate strings of LIST into a single string with fields separated by the value of EXPR, and returns the string.

Example:-

```
#!/usr/bin/perl
# define Strings
$var_string = "Rain-Drops-On-Roses-And-Whiskers-On-Kittens";
$var_names = "Larry,David,Roger,Ken,Michael,Tom";
# transform above strings into arrays.
&commat;string = split('-', $var_string);
&commat;names  = split(',', $var_names);
$string1 = join( '-', &commat;string );
$string2 = join( ',', &commat;names );
print "$string1\n";
print "$string2\n";
```

Output:-

```
Rain-Drops-On-Roses-And-Whiskers-On-Kittens
Larry,David,Roger,Ken,Michael,Tom
```

Sorting Arrays:- The sort() function sorts each element of an array according to the ASCII Numeric standards.

Syntax:−

```
sort [ SUBROUTINE ] LIST
```

This function sorts the LIST and returns the sorted array value. If SUBROUTINE is specified then specified logic inside the SUBTROUTINE is applied while sorting the elements.

```
#!/usr/bin/perl
# define an array
&commat;foods = qw(pizza steak chicken burgers);
print "Before: &commat;foods\n";
# sort this array
@foods = sort(&commat;foods);
print "After: &commat;foods\n";
```

Output:-

```
Before: pizza steak chicken burgers
After: burgers chicken pizza steak
```

Please note that sorting is performed based on ASCII Numeric value of the words. So the best option is to first transform every element of the array into lowercase letters and then perform the sort function.

Merging Arrays:-

```
#!/usr/bin/perl
&commat;numbers = (1,3,(4,5,6));
print "numbers = &commat;numbers\n";
```

Output:-

```
numbers = 1 3 4 5 6
```

The embedded arrays just become a part of the main array.

```
#!/usr/bin/perl
&commat;odd = (1,3,5);
&commat;even = (2, 4, 6);
&commat;numbers = (&commat;odd, &commat;even);
print "numbers = &commat;numbers\n";
```

Output:-

```
numbers = 1 3 5 2 4 6
```

Selecting Elements from Lists:- The list notation is identical to that for arrays. You can extract an element from an array by appending square brackets to the list and giving one or more indices.

```
#!/usr/bin/perl
$var = (5,4,3,2,1)[4];
print "value of var = $var\n"
```

Output:-

```
value of var = 1
```

Similarly, we can extract slices, although without the requirement for a leading @ character.

```
#!/usr/bin/perl
&commat;list = (5,4,3,2,1)[1..3];
print "Value of list = &commat;list\n";
```

Output:-

```
Value of list = 4 3 2
```

Hashes:- A hash is a set of key/value pairs. Hash variables are preceded by a percent (%) sign. To refer to a single element of a hash, you will use the hash variable name preceded by a "$" sign and followed by the "key" associated with the value in curly brackets.

Example:-

```
#!/usr/bin/perl
%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);
print "\$data{'John Paul'} = $data{'John Paul'}\n";
print "\$data{'Lisa'} = $data{'Lisa'}\n";
print "\$data{'Kumar'} = $data{'Kumar'}\n";
```

Output:-

```
$data{'John Paul'} = 45
$data{'Lisa'} = 30
$data{'Kumar'} = 40
```

Creating Hashes:- In the first method, you assign a value to a named key on a one-by-one basis.

```
$data{'John Paul'} = 45;
```

$data{'Lisa'} = 30;
$data{'Kumar'} = 40;

In the second case, you use a list, which is converted by taking individual pairs from the list. The first element of the pair is used as the key, and the second, as the value.

Example:-

%data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);

For clarity, you can use => as an alias for , to indicate the key/value pairs.

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

Here is one more variant of the above form, have a look at it, here all the keys have been preceded by hyphen (-) and no quotation is required around them.

%data = (-JohnPaul => 45, -Lisa => 30, -Kumar => 40);

But it is important to note that there is a single word, i.e., without spaces keys have been used in this form of hash formation and if you build-up your hash this way then keys will be accessed using hyphen only.

$val = %data{-JohnPaul}
$val = %data{-Lisa}

Accessing Hash Elements:- When accessing individual elements from a hash, you must prefix the variable with a dollar sign ($) and then append the element key within curly brackets after the name of the variable.

Example:-

```
#!/usr/bin/perl
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
print "$data{'John Paul'}\n";
print "$data{'Lisa'}\n";
print "$data{'Kumar'}\n";
```

Output:-

45
30
40

Extracting Slices:- You will need to use @ prefix for the variable to store the returned value because they will be a list of values.

```
#!/uer/bin/perl
%data = (-JohnPaul => 45, -Lisa => 30, -Kumar => 40);
&amp;commat;array = &amp;commat;data{-JohnPaul, -Lisa};
print "Array : &amp;commat;array\n";
```

Output:-

Array : 45 30

Extracting Keys and Values:- You can get a list of all of the keys from a hash by using keys function.

Syntax:−

keys %HASH

This function returns an array of all the keys of the named hash.

```
#!/usr/bin/perl
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
&commat;names = keys %data;
print "$names[0]\n";
print "$names[1]\n";
print "$names[2]\n";
```

Output:-

```
Lisa
John Paul
Kumar
```

Similarly, you can use values function to get a list of all the values.

Syntax:−

```
values %HASH
```

This function returns a normal array consisting of all the values of the named hash.

```
#!/usr/bin/perl
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
&commat;ages = values %data;
print "$ages[0]\n";
print "$ages[1]\n";
print "$ages[2]\n";
```

Output:-

```
30
45
40
```

Checking for Existence:- If you try to access a key/value pair from a hash that doesn't exist, you'll normally get the undefined value, and if you have warnings switched on, then you'll get a warning generated at run time. You can get safe from this by using the exists function, which returns true if the named key exists, regardless of what its value might be.

```
#!/usr/bin/perl
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
if( exists($data{'Lisa'} ) ) {
   print "Lisa is $data{'Lisa'} years old\n";
} else {
   print "I don't know age of Lisa\n";
}
```

Here we have introduced the IF...ELSE statement. If(condition) part will be executed only when the given condition is true otherwise else part will be executed.

Output:-

Lisa is 30 years old

Getting Hash Size:- You can get the size - that is, the number of elements from a hash by using the scalar context on either keys or values. Simply saying first you have to get an array of either the keys or values and then you can get the size of array.

```
#!/usr/bin/perl
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
&commat;keys = keys %data;
$size = &commat;keys;
print "1 - Hash size:  is $size\n";
@values = values %data;
$size = &commat;values;
print "2 - Hash size:  is $size\n";
```

Output:-

```
1 - Hash size: is 3
2 - Hash size: is 3
```

Add and Remove Elements in Hashes:- Adding a new key/value pair can be done with one line of code using simple assignment operator. But to remove an element from the hash you need to use delete function.

```
#!/usr/bin/perl
%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
&commat;keys = keys %data;
$size = &commat;keys;
print "1 - Hash size:  is $size\n";
# adding an element to the hash;
$data{'Ali'} = 55;
&commat;keys = keys %data;
$size = &commat;keys;
print "2 - Hash size:  is $size\n";
# delete the same element from the hash;
delete $data{'Ali'};
&commat;keys = keys %data;
$size = &commat;keys;
print "3 - Hash size:  is $size\n";
```

Output:-

```
1 - Hash size: is 3
2 - Hash size: is 4
3 - Hash size: is 3
```

Pattern matching:- m operator in Perl is used to match a pattern within the given text. The string passed to m operator can be enclosed within any character which will be used as a

seperator to regular expressions. To print this matched pattern and the remaining string, m operator provides various operators which include $, which contains whatever the last grouping match matched. $& - contains the entire matched string $` - contains everything before the matched string $' - contains everything after the matched string.

Syntax:-

m/String/

Return:-

0 on failure and 1 on success

Example:-

```
!/usr/bin/perl -w
# Text String
$string = "Welcome to SDCCMZN";
# Let us use m operator to search
# "to SD"
$string =~ m/to SD/;
# Printing the String
print "Before: $`\n";
print "Matched: $&\n";
print "After: $'\n";
```

Output:-

```
Before: Welcome
Matched: to SD
After: CCMZN
```

Introduction to CGI using PERL

A Common Gateway Interface(CGI) is a set of standards that defines how information is exchanged between the web server and a custom script. The CGI specs are currently maintained by the NCSA(National Center for Supercomputing Applications).

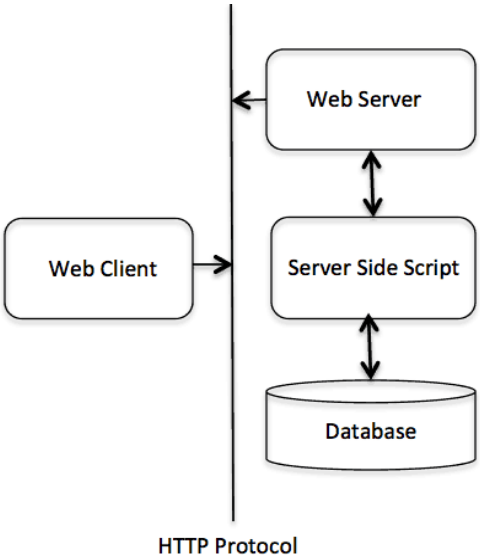
The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.

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

Web Browsing:-

- Your browser contacts web server using HTTP protocol and demands for the URL i.e. web page filename.
- Web Server will check the URL and will look for the filename requested. If web server finds that file then it sends the file back to the browser without any further execution otherwise sends an error message indicating that you have requested a wrong file.
- Web browser takes response from web server and displays either the received file content or an error message.

However, it is possible to set up HTTP server in such a way so that whenever a file in a certain directory is requested that file is not sent back, instead it is executed as a program, and whatever that program outputs as a result, that is sent back for your browser to display. This can be done by using a special functionality available in the web server and it is called CGI and such programs which are executed by the server to produce final result are called CGI scripts. These CGI programs can be a PERL Script, Shell Script, C or C++ program etc.



Web Server Support and Configuration:- Before you proceed with CGI Programming, make sure that your Web Server supports CGI functionality and it is configured to handle CGI programs. All the CGI programs to be executed by the web server are kept in a pre-configured directory. This directory is called CGI directory and generally it is named as /cgi-bin. In Perl CGI files will have extention as .cgi.

First CGI Program:- Here is a simple link which is linked to a CGI script called hello.cgi. This file has been kept in /cgi-bin/ directory and it has the following content. Before running your CGI program, make sure you have change mode of file using chmod 755 hello.cgi UNIX command.

```
#!/usr/bin/perl
print "Content-type:text/html\r\n\r\n";
print '<html>';
print '<head>';
print '<title>Hello Word - First CGI Program</title>';
print '</head>';
print '<body>';
print '<h2>Hello Word! This is my first CGI program</h2>';
print '</body>';
print '</html>';
1;
```

Now if you click hello.cgi link then request goes to web server who search for hello.cgi in /cgi-bin directory, execute it and whatever result got generated, web server sends that result back to the web browser.

Output:-

Hello Word! This is my first CGI program

This hello.cgi script is a simple Perl script which is writing its output on STDOUT file i.e. screen. There is one important and extra feature available which is first line to be printed Content-type:text/html\r\n\r\n. This line is sent back to the browser and specifies the content type to be displayed on the browser screen. This script can interact with any other external system also to exchange information such as a database, web services, or any other complex interfaces.

Understanding HTTP Header:- The very first line Content-type:text/html\r\n\r\n is a part of HTTP header, which is sent to the browser so that browser can understand the incoming content from server side. All the HTTP header will be in the following form:-

HTTP Field Name: Field Content

Example:-

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

HTTP Headers:-

Sr. No.	Header & Description
1	**Content-type:-String** A MIME(Multipurpose Internet Mail Extensions) string defining the format of the content being returned Example:- Content-type:text/html
2	**Expires:-Date String** The date when the information becomes invalid. This should be used by the browser to decide when a page needs to be refreshed. A valid date string should be in the format 01 Jan 1998 12:00:00 GMT
3	**Location:-URL String** The URL that should be returned instead of the URL requested. You can use this filed to redirect a request to any other location
4	**Last-modified:-String** The date of last modification of the file
5	**Content-length:-String** The length, in bytes, of the data being returned. The browser uses this value to report the estimated download time for a file
6	**Set-Cookie:-String** Set the cookie passed through the string

CGI Environment Variables:-

Sr. No.	Variables Names & Description
1	**CONTENT_TYPE** The data type of the content. Used when the client is sending attached content to the server. For example file upload etc
2	**CONTENT_LENGTH**

	The length of the query information. It's available only for POST requests
3	**HTTP_COOKIE** Returns the set cookies in the form of key & value pair
4	**HTTP_USER_AGENT** The User-Agent request-header field contains information about the user agent originating the request. Its name of the web browser
5	**PATH_INFO** The path for the CGI script
6	**QUERY_STRING** The URL-encoded information that is sent with GET method request
7	**REMOTE_ADDR** The IP address of the remote host making the request. This can be useful for logging or for authentication purpose
8	**REMOTE_HOST** The fully qualified name of the host making the request. If this information is not available then REMOTE_ADDR can be used to get IR(Internet Remote) address
9	**REQUEST_METHOD** The method used to make the request. The most common methods are GET and POST
10	**SCRIPT_FILENAME** The full path to the CGI script
11	**SCRIPT_NAME** The name of the CGI script
12	**SERVER_NAME** The server's hostname or IP Address
13	**SERVER_SOFTWARE** The name and version of the software the server is running

Example:-

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "<font size=+1>Environment</font>\n";
foreach (sort keys %ENV) {
print "<b>$_</b>: $ENV{$_}<br>\n";
}
1;
```

Raise a "File Download" Dialog Box:- Sometime it is desired that you want to give option where a user will click a link and it will pop up a "File Download" dialogue box to the user instead of displaying actual content. This is very easy and will be achived through HTTP header. This HTTP header will be different from the header mentioned in previous section.

Example:-

```
#!/usr/bin/perl
# HTTP Header
print "Content-Type:application/octet-stream; name = \"FileName\"\r\n";
print "Content-Disposition: attachment; filename = \"FileName\"\r\n\n";
# Actual File Content will go hear.
open( FILE, "<FileName" );
while(read(FILE, $buffer, 100) ) {
   print("$buffer");
}
```

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

Passing Information using GET Method:- The GET method sends the encoded user information appended to the page URL itself. The page and the encoded information are separated by the ? character.

http://www.test.com/cgi-bin/hello.cgi?key1=value1&key2=value2

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

This information is passed using QUERY_STRING header and will be accessible in your CGI Program through QUERY_STRING environment variable which you can analyze and use in your CGI program.

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

Simple URL Example:-

Get Method:-

http://www.sdccmzn.com/cgi-bin/hello_get.cgi?first_name=ZARA&last_name=ALI

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

```
#!/usr/bin/perl
local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "GET") {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
```

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

Simple FORM Example:-

GET Method:-

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

Output:-

First Name:

Last Name: Submit

Passing Information using POST Method:- This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL, it sends it as a separate message as a part of HTTP header. Web server provides this message to the CGI script in the form of the standard input.

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

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

```
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};
print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Hello - Second CGI Program</title>";
print "</head>";
print "<body>";
print "<h2>Hello $first_name $last_name - Second CGI Program</h2>";
print "</body>";
print "</html>";
1;
```

We are going to use CGI script hello_post.cgi to handle input.

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

Output:-

First Name:

Last Name: Submit

Passing Checkbox Data to CGI Program:-

Example:-

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

```
<input type = "submit" value = "Select Subject">
</form>
```

Output:-

☐ Maths ☐ Physics Select Subject

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

```
#!/usr/bin/perl
local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
if( $FORM{maths} ) {
   $maths_flag ="ON";
} else {
   $maths_flag ="OFF";
}
if( $FORM{physics} ) {
   $physics_flag ="ON";
} else {
   $physics_flag ="OFF";
}
print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Checkbox - Third CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> CheckBox Maths is : $maths_flag</h2>";
print "<h2> CheckBox Physics is : $physics_flag</h2>";
```

```
print "</body>";
print "</html>";
1;
```

Passing Radio Button Data to CGI Program:-

Example:-

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

Output:-

Maths Physics Select Subject

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

```
#!/usr/bin/perl
local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$subject = $FORM{subject};
print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Radio - Fourth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Selected Subject is $subject</h2>";
print "</body>";
print "</html>";
```

1;

Passing Text Area Data to CGI Program:-

Example:-

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

Output:-

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

```
#!/usr/bin/perl
local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$text_content = $FORM{textcontent};
print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Text Area - Fifth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Entered Text Content is $text_content</h2>";
print "</body>";
```

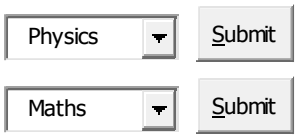
```
print "</html>";
1;
```

Passing Drop Down Box Data to CGI Program:-

Example:-

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

Output:-



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

```
#!/usr/bin/perl
local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
} else {
   $buffer = $ENV{'QUERY_STRING'};
}
# Split information into name/value pairs
@pairs = split(/&/, $buffer);
foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$subject = $FORM{dropdown};
print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Dropdown Box - Sixth CGI Program</title>";
print "</head>";
print "<body>";
```

```
print "<h2> Selected Subject is $subject</h2>";
print "</body>";
print "</html>";
1;
```

Using Cookies in CGI:- HTTP protocol is a stateless protocol. But for a commercial website it is required to maintain session information among different pages. For example one user registration ends after transactions which spans through many pages. But how to maintain user's session information across all the web pages?

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

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

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

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

Setting up Cookies:- These cookies will be sent along with the HTTP Header. Assuming you want to set UserID and Password as cookies.

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

Here we used Set-Cookie HTTP header to set cookies. It is optional to set cookies attributes like Expires, Domain and Path. It is important to note that cookies are set before sending magic line "Content-type:text/html\r\n\r\n.

Retrieving Cookies:- Cookies are stored in CGI environment variable HTTP_COOKIE and they will have following form:-

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

Example:-

```
#!/usr/bin/perl
$rcvd_cookies = $ENV{'HTTP_COOKIE'};
@cookies = split /;/, $rcvd_cookies;
foreach $cookie ( @cookies ) {
   ($key, $val) = split(/=/, $cookie); # splits on the first =.
   $key =~ s/^\s+//;
   $val =~ s/^\s+//;
   $key =~ s/\s+$//;
   $val =~ s/\s+$//;
   if( $key eq "UserID" ) {
      $user_id = $val;
   } elsif($key eq "Password") {
      $password = $val;
   }
}
print "User ID  = $user_id\n";
print "Password = $password\n";
```

This will produce the following result, provided above cookies have been set before calling retrieval cookies script.

```
User ID = XYZ
Password = XYZ123
```

Overview of JAVA script

Introduction:- JavaScript is a versatile, dynamically typed programming language that brings life to web pages by making them interactive. It is used for building interactive web applications, supports both client-side and server-side development, and merges smoothly with HTML, CSS, and a rich standard library.

- JavaScript is a single-threaded language that executes one task at a time.
- It is an interpreted language which means it executes the code line by line.
- The data type of the variable is decided at run-time in JavaScript, which is why it is called dynamically typed.

"Hello, World!" Program in Browser Console:-

```
<html>
<head></head>
<body>
   <h1>Check the console for the message!</h1>
   <script>
      // This is our first JavaScript program
      console.log("Hello, World!");
```

```
    </script>
</body>
</html>
```

- The <script> tag is used to include JavaScript code inside an HTML document.
- console.log() prints messages to the browser's developer console. Open the browser console to see the "Hello, World!" message.

"Hello World" Program in Server Console:- We can also print the "Hello World" program directly into the console terminal without embedded it into HTML. Create an index.js file and add the code to it.

```
// This is a comment
console.log("Hello, World!");
```

Run it in your terminal with:-

```
node hello.js
```

Comments in JavaScript:- Comments are notes in your code that the JavaScript interpreter ignores. They're great for explaining what your code does or for testing purposes.

- Single-line comment: Starts with //

  ```
  // This is a single-line comment
  ```

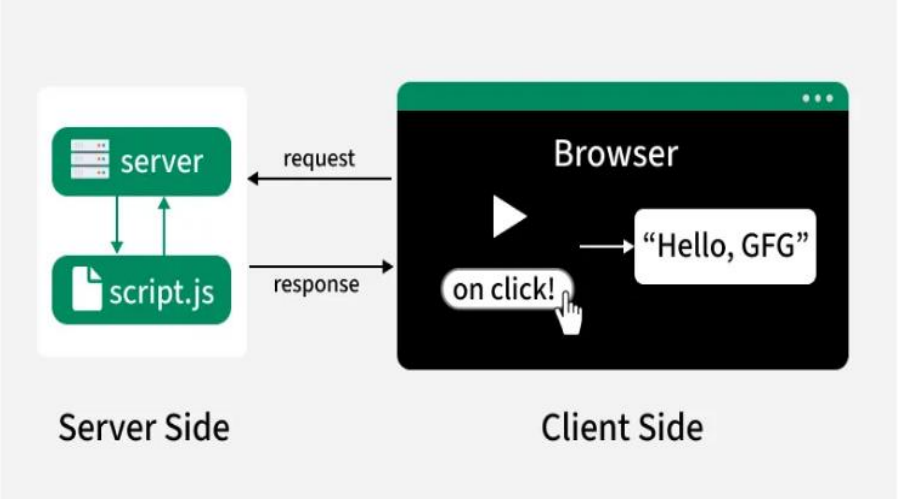
- Multi-line comment: Enclosed in /* */

  ```
  /* This is a multi-line comment
  spanning multiple lines */
  ```

Features of JavaScript:-

- **Client-Side Scripting**:- JavaScript runs on the user's browser, so has a faster response time without needing to communicate with the server.
- **Versatile**:- Can be used for a wide range of tasks, from simple calculations to complex server-side applications.
- **Event-Driven**:- Responds to user actions (clicks, keystrokes) in real-time.
- **Asynchronous**:- It can handle tasks like fetching data from servers without freezing the user interface.
- **Rich Ecosystem**:- There are numerous libraries and frameworks built on JavaScript, such as React, Angular and Vue.js, which make development faster and more efficient.

Client Side and Server Side nature of JavaScript:-



JavaScript's flexibility extends to both the client-side and server-side, allowing developers to create complete web applications.

Client-Side:-

- Involves controlling the browser and its DOM (Document Object Model).
- Handles user events like clicks and form inputs.
- Common libraries include AngularJS, ReactJS and VueJS.

Server-Side:-

- Involves interacting with databases, manipulating files, and generating responses.
- Node.js and frameworks like Express.js are widely used for server-side JavaScript, enabling full-stack development.

Imperative vs. Declarative JavaScript:- The imperative and declarative is a programming paradigm, and JavaScript follows both.

- **Imperative JavaScript**:- In imperative JavaScript, we write code in the manner that the code describes the steps to get the output. So, we are concerned about the code execution flow and output both. For example, to sum all array elements, if we write code for loop, it explains each step to get the sum.
- **Declarative JavaScript**:- In declarative JavaScript, we don't need to worry about execution flow, but we should get the correct output at the end. For example, we use a built-in array.reduce() method to get a sum of array elements. Here, we dont concern about how reduce() method is implemented in the library.

Limitations of JavaScript:-

- **Security Risks**:- Can be used for attacks like Cross-Site Scripting (XSS), where malicious scripts are injected into a website to steal data by exploiting elements like <img**>**, <object**>**, or <script**>** tags.
- **Performance**:- Slower than traditional languages for complex tasks, but for simple tasks in a browser, performance is usually not a major issue.
- **Complexity**:- To write advanced JavaScript, programmers need to understand core programming concepts, objects, and both client- and server-side scripting, which can be challenging.