# COLEGIO VOCACIONAL DE ARTES Y OFICIOS DE CARTAGO NOCTURNO

## CURSO LECTIVO 2021

## Primer Periodo

### Enunciado Examen Corto Práctico # 1

**NIVEL: DECIMO**
**SECCIÓN: 10-05**

**Sub-Área: Programación**
**PROFESOR**
**Lic. Alexander Monge Vargas**

**Alumno: Nelson Rodríguez Zúñiga**

**Abril, 2021**

# Contenido



## Examen Corto Practico #1

El presente documento le servirá para entregar el primer examen corto tipo práctico de la materia de programación, el mismo consta de una parte de preguntas y otra de realización práctica.

Las preguntas las contestará en la web que debe realizar, utilizando solo HTML y CSS puro.

### Indicaciones generales

Deberá realizar una web y entregar la misma en la fecha y hora determinada por el profesor en el sitio.

## OBJETIVOS ACADEMICOS:

### General:

Involucrar al alumno en su primera experiencia de realización de una web utilizando HTML, CSS puros.

### Específicos.

| |
|---|
| 1. Crear una web a partir de CSS y HTML |
| 2. Reforzar los conocimientos adquiridos en clases. |
| 3. Conocer más a fondo la teoría y conceptos vistos en clases. |
| 4. Realizar el primer Portafolio digital del curso. |
| 5. Recopilar los conocimientos en un sitio. |

## Distribución de la rúbrica de evaluación (puntos)

- Página a desarrollar 20 pts.
- Investigación a Desarrollar 10 pts
- Presentación del documento de acuerdo a las indicaciones dadas 10 puntos.
- El puntaje de este trabajo es parte del 20% correspondiente a pruebas cortas.

I PARTE

Indicaciones

Realice una página web con los conocimientos vistos en clase, pero no se limite, puede usar y hacer más de lo que acá se solicita. 20pts.

La página web será un tipo de portafolio digital para todas sus subáreas, **deberá tener una página inicial con una bienvenida y explicación de la especialidad que cursa.**

Luego deberá tener una página para cada sub-área, las sub-áreas distintas a programación deberá salir un rotulo de Under Construction.

La página de la sub-área programación tendrá la información que se solicita en las preguntas de la parte #2.

II Parte

Preguntas a contestar.

1. **¿Cuál es la diferencia del Desarrollo Web versus el Diseño Web?**

El diseño web determina la apariencia y la percepción de un sitio web. Comprende el diseño, la navegación y la gama de colores del sitio web. También incluye el diseño gráfico y del logo. El diseño web tiene más que ver con la estética y la experiencia del usuario que con las funciones. Un diseñador web hará que un sitio web sea fácil de usar y que se ajuste al propósito para el que fue creado.

En contraste, el desarrollo web se encarga de las funciones y características de un sitio web. Abarca la programación de la "parte de atrás" de un sitio web, cubriendo funciones como el registro, los sistemas de gestión de contenido, el e-comercio y cualquier aplicación de base de datos. El desarrollo web hace posible que los visitantes de un sitio web puedan llevar a cabo acciones en él.

Estudios demuestran que los visitantes de un sitio web emiten un juicio en tan solo unos segundos. Por lo tanto, un buen diseño web hará que el sitio resulte atractivo y fácil de usar, y un buen desarrollo web lo dotará de características y funciones interesantes. Todo ello dará como resultado que el visitante se sienta atraído por el sitio web y vuelva a visitarlo.

En conclusión, tanto el diseño como el desarrollo web son esenciales para conseguir un sitio web efectivo. El diseño web debe integrarse en el desarrollo web, por lo que lo ideal es que diseñadores y desarrolladores trabajen codo con codo.

## 2. ¿Qué es un modelo cliente servidor?

**Modelo cliente servidor**

La expresión cliente servidor se utiliza en el ámbito de la informática. En dicho contexto, se llama cliente al dispositivo que requiere ciertos servicios a un servidor. La idea de servidor, por su parte, alude al equipo que brinda servicios a las computadoras (ordenadores) que se hallan conectadas con él mediante una red.

El concepto de cliente servidor, o cliente-servidor, refiere por lo tanto a un modelo de comunicación que vincula a varios dispositivos informáticos a través de una red. El cliente, en este marco, realiza peticiones de servicios al servidor, que se encarga de satisfacer dichos requerimientos.

**Importancia del modelo cliente servidor**

La arquitectura cliente servidor tiene dos partes claramente diferenciadas, por un lado, la parte del servidor y por otro la parte de cliente o grupo de clientes donde lo habitual es que un servidor sea una máquina bastante potente con un hardware y software específico que actúa de depósito de datos y funcione como un sistema gestor de base de datos o aplicaciones.

En esta arquitectura el cliente suele ser estaciones de trabajo que solicitan varios servicios al servidor, mientras que un servidor es una máquina que actúa como depósito de datos y funciona como un sistema gestor de base de datos, este se encarga de dar la respuesta demandada por el cliente.

Esta arquitectura se aplica en diferentes modelos informáticos alrededor del mundo donde su propósito es mantener una comunicación de información entre diferentes entidades de una red mediante el uso de protocolos establecidos y el apropiado almacenaje de la misma.

El más claro ejemplo de uso de una arquitectura cliente servidor es la red de Internet donde existen ordenadores de diferentes personas conectadas alrededor del mundo, las cuales se conectan a través de los servidores de su proveedor de Internet por ISP donde son redirigidos a los servidores de las páginas que desean visualizar y de esta manera la información de los servicios requeridos viaja a través de Internet dando respuesta a la solicitud demandada.

La principal importancia de este modelo es que permite conectar a varios clientes a los servicios que provee un servidor y como sabemos hoy en día, la mayoría de las aplicaciones y servicios tienen como gran necesidad que puedan ser consumidos por varios usuarios de forma simultánea.

**Componentes**

Para entender este modelo vamos a nombrar y definir a continuación algunos conceptos básicos que lo conforman.

Red: Una red es un conjunto de clientes, servidores y base de datos unidos de una manera física o no física en el que existen protocolos de transmisión de información establecidos.

Cliente: El concepto de cliente hace referencia a un demandante de servicios, este cliente puede ser un ordenador como también una aplicación de informática, la cual requiere información proveniente de la red para funcionar.

Servidor: Un servidor hace referencia a un proveedor de servicios, este servidor a su vez puede ser un ordenador o una aplicación informática la cual envía información a los demás agentes de la red.

Protocolo: Un protocolo es un conjunto de normas o reglas y pasos establecidos de manera clara y concreta sobre el flujo de información en una red estructurada.

Servicios: Un servicio es un conjunto de información que busca responder las necesidades de un cliente, donde esta información pueden ser mail, música, mensajes simples entre software, videos, etc.

Base de datos: Son bancos de información ordenada, categorizada y clasificada que forman parte de la red, que son sitios de almacenaje para la utilización de los servidores y también directamente de los clientes.

**Diferencia entre cliente y servidor**

Como hemos mencionado anteriormente una máquina cliente como servidor se refieren a computadoras que son usadas para diferentes propósitos.

El cliente es un computador pequeño con una estructura al igual a la que tenemos en nuestras oficinas u hogares la cual accede a un servidor o a los servicios del mismo a través de Internet o una red interna. Un claro ejemplo a este caso es la forma en que trabaja una empresa modelo con diferentes computadores donde cada uno de ellos se conectan a un servidor para poder obtener archivos de una base de datos o servicios ya sea correos electrónicos o aplicaciones.

El servidor al igual que el cliente, es una computadora, pero con diferencia de que tiene una gran capacidad que le permite almacenar gran cantidad de diversos de archivos, o correr varias aplicaciones en simultaneo para así nosotros los clientes poder acceder los servicios.

En la actualidad existen varios tipos de servidores como hablamos anteriormente. Los mismos pueden contener y ejecutar aplicaciones, sitios web, almacenaje de archivos, diversas bases de datos, entre muchos más.

Es importante mencionar que un cliente también puede tener una función de servidor ya que el mismo puede almacenar datos en su disco duro para luego ser usados en vez de estar conectándose al servidor continuamente por una acción que quizás sea muy sencilla.

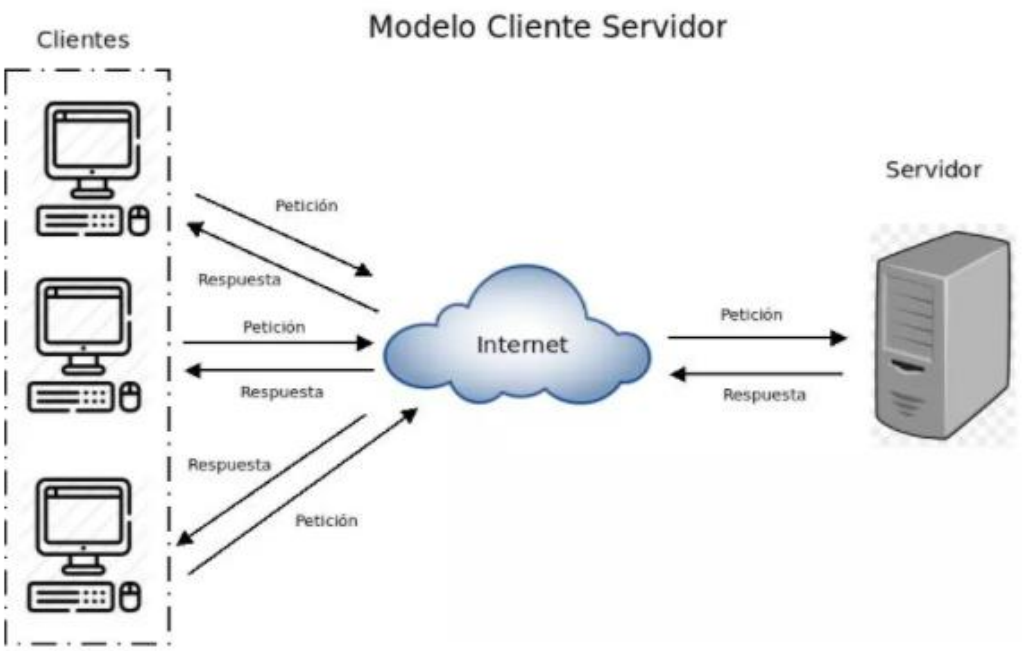


Figura 01. Esquema del Modelo Cliente Servidor

**Tipos de arquitecturas cliente servidor**

Dentro de la arquitectura cliente servidor existen tres tipos en donde hablaremos brevemente de cómo funciona cada uno de ellos.

**Arquitectura de dos capas**

Esta se utiliza para describir los sistemas cliente servidor en donde el cliente solicita recursos y el servidor responde directamente a la solicitud con sus propios recursos. Eso significa que el servidor no requiere de una aplicación extra para proporcionar parte del servicio.

**Arquitectura de tres capas**

En la arquitectura de tres capas existe un nivel intermediario, eso significa que la arquitectura generalmente está compartida por un cliente que como hablamos más arriba es el que solicita los recursos equipados con una interfaz de usuario o mediante un navegador web.

La capa del medio es denominada software intermedio cuya tarea es proporcionar los recursos solicitados pero que requiere de otro servidor para hacerlo. La última capa es el servidor de datos que proporciona al servidor de aplicaciones los datos necesarios para poder procesar y generar el servicio que solicito el cliente en un principio.

**Arquitectura N capas**

En la arquitectura de tres capas, los servidores dos y tres realizaron una tarea específica por lo tanto un servidor web puede usar los servicios de otros servidores para poder proporcionar su propio servicio.

Por consiguiente, la arquitectura en tres niveles es potencialmente una arquitectura en N capas ya que, así como está contemplado en tres niveles como el caso anterior puede estar compuesto por N servidores donde cada uno de ellos brindan su servicio específico.

**Ventajas y Desventajas**

Este modelo cliente servidor tiene varias ventajas y desventajas las cuales son importantes mencionar y conocer a la hora de establecer si es lo que estamos necesitando o si se acomoda a lo que estamos buscando.

**Ventajas**

Facilita la integración entre diferentes sistemas y comparte información permitiendo por ejemplo que las máquinas ya existentes puedan ser utilizadas mediante una interfaz más amigable para el usuario. De esta manera podemos integrar varias PCs con sistemas medianos y grandes sin necesidad de que todos tengan que utilizar el mismo sistema operativo.

Al favorecer el uso de la interfaz de gráficas interactivas, los sistemas construidos bajo este esquema tienen una mayor interacción con el usuario.

La estructura modular facilita de más la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional favoreciendo así la estabilidad de las soluciones.

El modelo cliente servidor permite además proporcionar a las diferentes áreas de una empresa generar un orden de trabajo en donde cada sector puede trabajar en su área, pero accediendo al mismo servidor e información que los demás sin generar conflictos. Esto es de gran utilidad ya que si ponemos como ejemplo una empresa con varios empleados al momento de trabajar es importante que todos puedan hacerlo en simultáneo.

**Desventajas**

Requiere habilidad para que un servidor sea reparado. Por ejemplo, si un problema ocurre en la red, se requiere de alguien con un amplio de esta para poder repararla en su totalidad para así dejar que la información y el correcto funcionamiento siga su flujo.

Otro problema es la seguridad, el hecho que se comparte canales de información entre servidores y clientes requieren que estas pasen por procesos de validación, es decir protocolos de seguridad que pueden tener algún tipo de puerta abierta permitiendo que se generen daños físicos, amenazas o ataques de malware.

Este modelo representa una limitación importante en cuanto a los costos económicos debido a que estos servidores son computadoras de alto nivel con un hardware y software específicos para poder dar un correcto funcionamiento a nuestras aplicaciones. Algo importante a destacar es que no solo es caro a la hora de solucionar problemas como mencionamos antes, sino que también tiene un costo elevado para reemplazar componentes que estén averiados.

**Ejemplos de modelo cliente servidor**

Existen muchísimos servicios, protocolos y servidores que trabajan con el mismo modelo que mencionamos antes. Casi todo el Internet funciona de esa forma de hecho.

Algunos ejemplos de la arquitectura cliente servidor pueden ser:

**Navegar una web** funciona basándonos en un cliente web (navegador) y un servidor web como Apache, Nginx o LiteSpeed

**Protocolo FTP**, funciona de idéntica forma, se utiliza un cliente de FTP (como Filezilla) para conectar a un servidor FTP (como Pure-FTPD, Proftpd, etc)

**SSH**: es idéntico también, se utiliza un cliente SSH para conectar al servidor SSH que corre en una red remota.

**Juegos en red**: existen clientes que permiten a jugadores online jugar desde sus casas conectándose a servidores de juegos remotos.

**Sistema DNS**: el famoso servidor DNS interactúa con clientes DNS también, es decir, basa su arquitectura en el modelo cliente servidor

**Servidor de Correo**: donde clientes de correo consultan el correo al servidor de correo remoto, tanto desde móvil o una computadora de escritorio o laptop.

**Conclusión**

Podemos concluir que el sistema cliente servidor es un modelo flexible y adaptable al servicio que se quiere implementar. Este nos permite aumentar el rendimiento, así como también, envolver variadas plataformas, bases de datos, redes y sistemas operativos que pueden ser de diferentes distribuidores con arquitecturas totalmente diferentes y funcionando todos al mismo tiempo.

Además, se puede considerar un sistema ventajoso en cuanto a seguridad, ya que el servidor controla el acceso a sus datos por lo que se necesita que el servidor nos autorice para poder acceder a él.

También es escalable y ante una gran demanda de tráfico se pueden utilizar tecnologías complementarias, por lo que cualquier organización que utilice estos sistemas adquiere ventajas competitivas.

## 3. ¿Qué es un servidor?

En computación, se conoce como servidor (del inglés server) a un computador que forma parte de una red informática y provee determinados servicios al resto de los computadores de la misma, llamados a su vez estaciones o clientes. Dicho computador debe contar con una aplicación específica capaz de atender las peticiones de los distintos clientes y brindarles respuesta oportuna, por lo que en realidad dentro de una misma computadora física (hardware) pueden funcionar varios servidores simultáneos (software), siempre y cuando cuenten con los recursos logísticos necesarios.

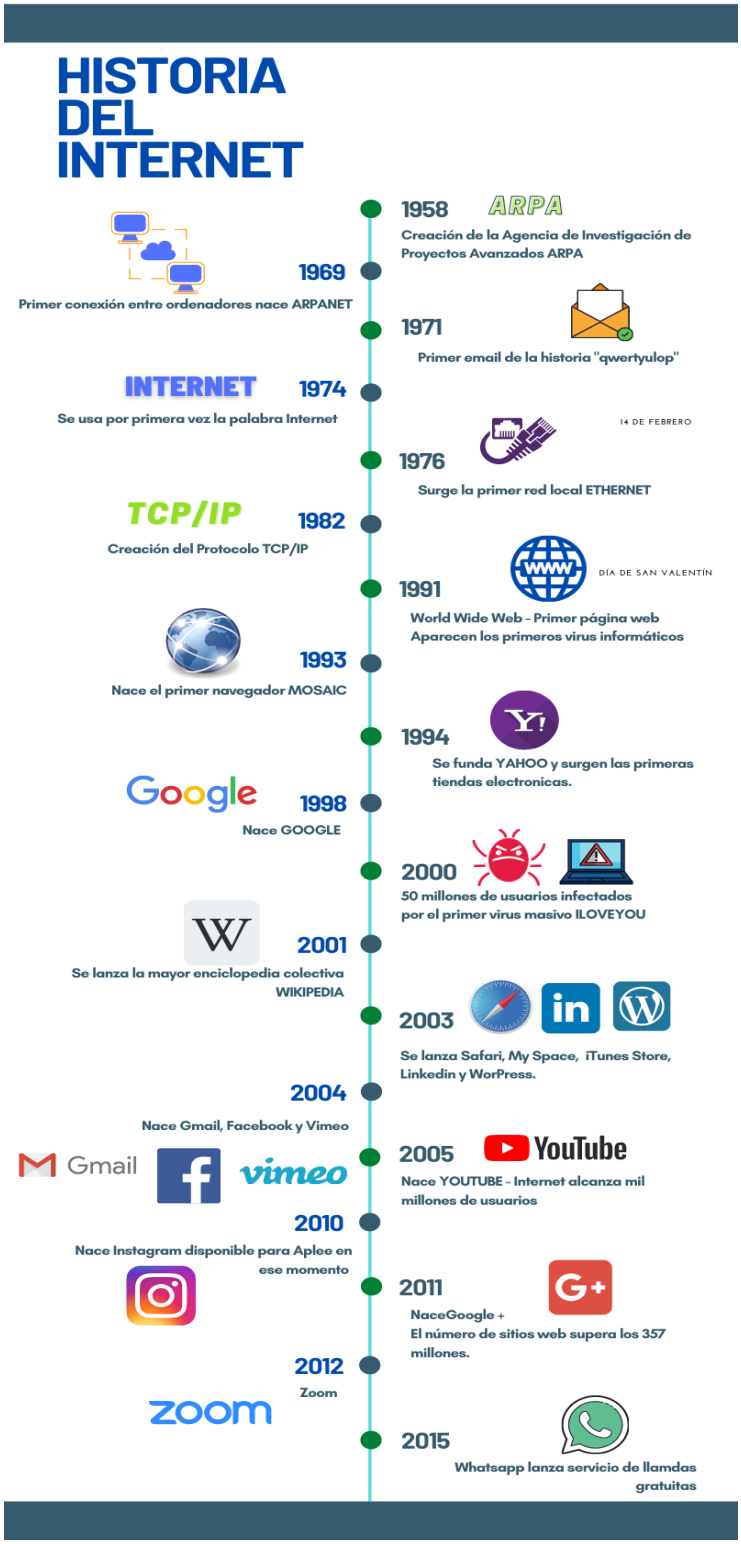
Se denomina **servidor dedicado**, aquel que dedica todos sus recursos a atender solicitudes de los equipos cliente.

Sin embargo, **un servidor compartido** es aquel que no dedica todos sus recursos a servir las peticiones de los clientes, sino que también es utilizado por un usuario para trabajar de forma local.

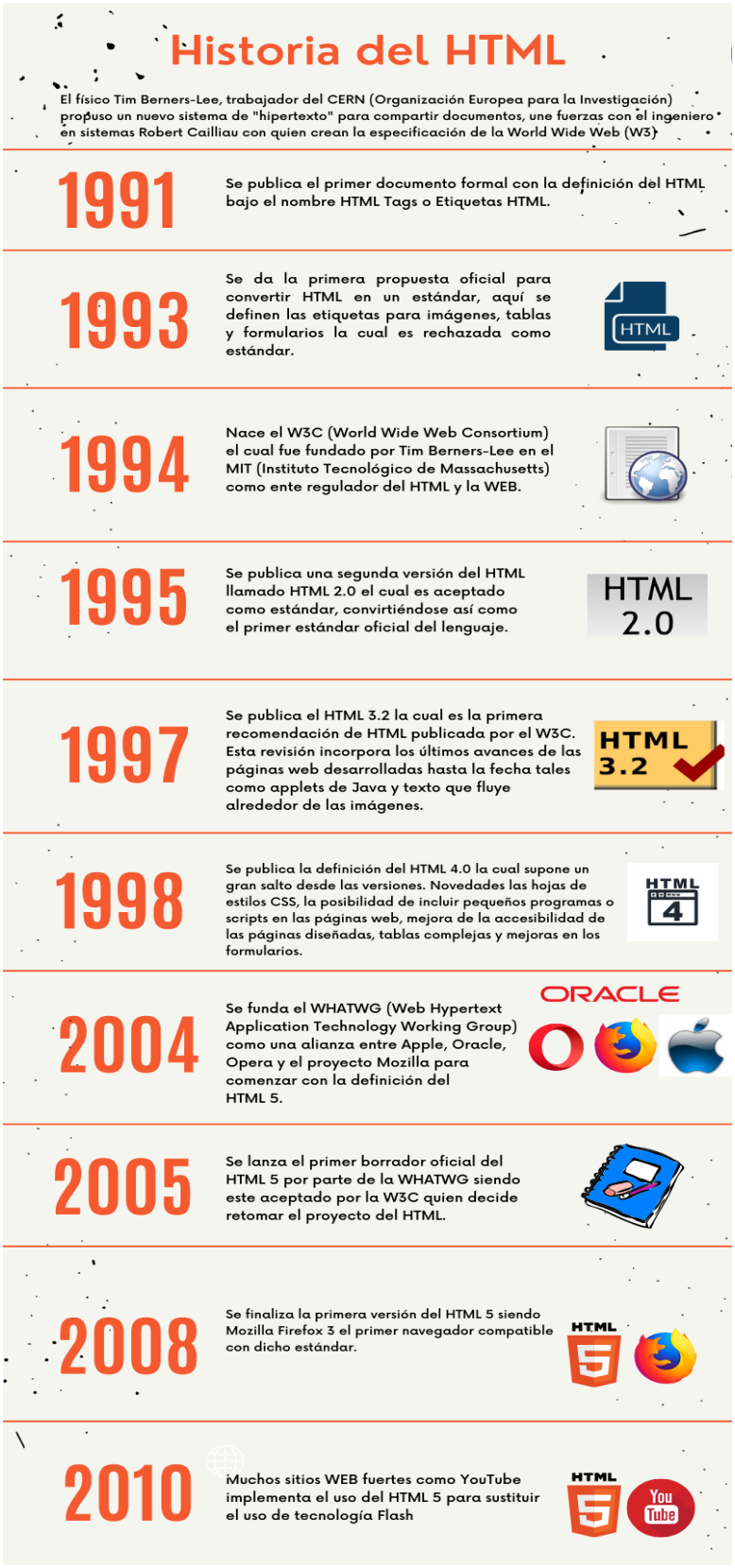
Existen gran cantidad de tipos de servidores o roles que estos pueden desempeñar. En la siguiente relación enumeramos algunos de los más comunes.

- **Servidor de archivos**: es aquel que almacena y sirve ficheros a equipos de una red.
- **Servidor de Directorio Activo/Dominio**: es el que mantiene la información sobre los usuarios, equipos y grupos de una red.
- **Servidor de Impresión**: se encarga de servir impresoras a los equipos cliente y poner en la cola los trabajos de impresión que estos generan.
- **Servidor de Correo**: se encarga de gestionar el flujo de correo electrónico de los usuarios, envía, recibe y almacena los correos de una organización.
- **Servidor de Fax**: gestiona el envío, recepción y almacenamiento de los faxes.
- **Servidor Proxy**: su principal función es guardar en memoria caché las páginas web a las que acceden los usuarios de la red durante un cierto tiempo, de esta forma las siguientes veces que estos acceden al mismo contenido, la respuesta es más rápida.
- **Servidor Web**: Almacena contenido web y lo pone al servicio de aquellos usuarios que lo solicitan.
- **Servidor de Base de Datos**: es aquel que provee servicios de base de datos a otros programas o equipos cliente.
- **Servidor DNS**: permite establecer la relación entre los nombres de dominio y las direcciones IP de los equipos de una red.
- **Servidor DHCP**: este dispone de un rango de direcciones con el cual, asigna automáticamente los parámetros de configuración de red IP a las maquinas cliente cuando estas realizan una solicitud.
- **Servidor FTP**: su función es permitir el intercambio de ficheros entre equipos, normalmente su aplicación va muy ligada a los servidores Web.

4. **Realice un infográfico (Para ello puede usar el siguiente enlace https://www.canva.com/es_es/ de la historia del internet.**



## 5. Realice un Infográfico de la historia de HTML



## 6. ¿Cuál es la diferencia entre un lenguaje compilado y uno interpretado?

De forma general, la diferencia entre los lenguajes compilados e interpretados es que los primeros usan un compilador para poder traducirlo y ejecutar el programa, mientras que los segundos requieren de un intérprete que traduzca el código al momento de la ejecución.

Para entender la diferencia entre los lenguajes compilados y los interpretados, primero se debe entender la definición de cada uno.

**Lenguajes compilados**

Son aquellos lenguajes de alto nivel que como su nombre lo sugiere, requieren de un compilador (programa que traduce un lenguaje de alto nivel en código máquina o lenguaje máquina) para traducirlo y crear la parte ejecutable.

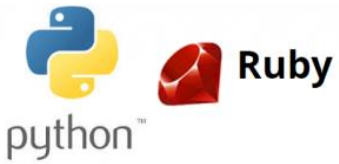
Los lenguajes de alto nivel permiten escribir instrucciones en un idioma muy parecido al inglés, así como hacer uso de notaciones matemáticas comunes.

Ejemplos: C y C++

**Lenguajes interpretados**

Los lenguajes interpretados son aquellos lenguajes de programación también de alto nivel en donde el código fuente debe ser traducido a un lenguaje que la máquina entienda y pueda realizar, no crean un archivo externo.

Ejemplos: Python y Ruby



**Diferencia entre lenguajes compilados e interpretados**

Los lenguajes interpretados son multiplataformas, recordemos que una plataforma es un sistema operativo que ejecuta programas, por lo que son más flexibles, pero se requiere de un intérprete (programa informático que analiza y ejecuta otros programas) para traducirlo y que la máquina lo reconozca.

Todo esto sucede al instante ya que no se guarda la traducción del intérprete, es decir, se traduce al momento cada línea de código de forma ordenada cada vez que se ejecuta el programa por lo que lo hace un poco más lento a comparación de los compilados debido a la sobrecarga al momento de la ejecución. A pesar de esta diferencia de tiempo, los lenguajes interpretados son más populares en el desarrollo de programas que se modifican constantemente, ya sea para agregar o corregir una parte del código. Si bien el aspecto de parecer más lentos puede ser algo negativo, también tiene su punto a favor ya que, a diferencia de los lenguajes compilados que requieren de compilar y esperar a que el programa se haya terminado para saber si existen errores, en los lenguajes interpretados se sabe de inmediato si existe un error y su localización ya que se ejecuta línea por línea, así se puede corregir y modificar el error.

Por otro lado, los lenguajes compilados están preparados para ejecutarse inmediatamente ya que durante la compilación se tradujo todo a un lenguaje que la máquina entendiera (lenguaje máquina) y por ello suelen ser más rápidos.

De modo que, gracias al compilador se traduce el código fuente, se crea un archivo ejecutable y de este no se puede acceder a su código fuente (a menos que sea desde la máquina de origen) dependiendo del caso esto podría ser o no algo positivo. Esto se debe a que una vez compilado el programa ya no es necesario el código fuente para poder ejecutarlo. Al obtener el archivo objeto puedes ejecutar el programa sin tener que compilar cada vez que lo ejecutes. Otra diferencia radica en que estos no son multiplataformas (solo sirve para una plataforma en específico) y de querer ejecutarlo en otra plataforma o computadora este se debe compilar de nueva cuenta lo que le suma un paso extra y tiempo de más. En caso de existir un error en el código, el compilador lo hará saber, una vez depurado el problema se requiere compilar otra vez el programa.

**Acerca de los intérpretes y compiladores**

**Intérprete**

El punto clave de los lenguajes interpretados es, como su nombre lo sugiere, el intérprete que requieren para ejecutar el programa. Un intérprete es un programa que interpreta y ejecuta otros programas. La traducción se realiza línea por línea y no guarda la traducción.

El proceso para ejecutar un programa en los lenguajes interpretados se lleva a cabo cuando el intérprete obtiene la instrucción del código fuente (recordemos que el código fuente son líneas de texto con instrucciones escritas en un lenguaje de programación diferente al lenguaje máquina por lo que es necesario traducirlos y que la máquina lo entienda) para realizarla de forma inmediata. Primero se ejecutan en secuencia parte del código máquina que radican en el intérprete, éstas pasan a la unidad central de procesamiento (C.P.U). Cuando se finaliza dicha secuencia el resultado realizado por la C.P.U era lo que las líneas del código fuente dictaban o expresaban. Esto se realiza línea por línea hasta que se finaliza todo lo escrito por el programador.

Existen diferentes tipos de intérpretes y se clasifican según la estructura interna del intérprete. Algunos de ellos utilizan procesos propios de los compiladores y de los intérpretes, por los que se les considera híbridos. Algunos ejemplos de otros tipos de intérpretes son:

Interpretes avanzados: permite realizar un análisis más a fondo del código fuente e incorporan un paso previo de análisis de todo el programa que se está creando.

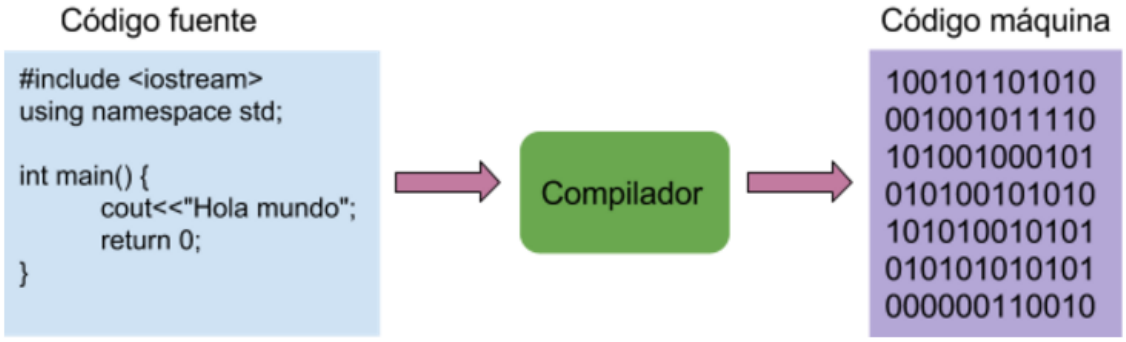
Intérpretes incrementares: utilizados comúnmente en sistemas interactivos, donde existen módulos compilados y módulos que se pueden modificar.

**Compilador**

Por otro lado, el punto clave de los lenguajes compilados es la necesidad de un compilador, como ya se mencionó anteriormente, el compilador es el programa que se encarga de traducir los programas que escribimos en un lenguaje de alto nivel (parecido al inglés y que nosotros podemos entender) a código máquina (lenguaje de máquina, binario), se producirá la compilación si el programa no tiene ningún error de sintaxis. De existir un error en el código el compilador genera una lista con los errores existentes para poder depurarlos.

La secuencia para la compilación inicia cuando creamos el código fuente y lo compilamos. De no haber error se obtiene un programa fuente que el compilador tradujo a código máquina. El programa fuente pasa a un linker o enlazador que es un programa que se encarga de insertar el código máquina de las librerías que decidiste usar en tu programa (las librerías son colecciones de funciones ya traducidas a código máquina que facilitan al programador crear su programa). Después de realizar ese proceso, el linker crea el programa ejecutable. El programa ejecutable es la traducción ya terminada a código máquina.

Para ejecutar el programa una parte del sistema operativo de la computadora, conocido como cargador, inserta el programa ejecutable a la memoria principal de la máquina y lo prepara para la ejecución. El control pasa a la unidad central de procesamiento (C.P.U) donde primero se lleva a cabo una fase llamada “captación de instrucciones” en la cual las instrucciones que se ejecutarán se leen desde la memoria. La fase siguiente, denominada “ejecución de la instrucción”, interpreta las instrucciones que se leyeron la fase anterior y envía unas señales a los controles que tengan que intervenir en la ejecución, después de esa ejecución se establece la siguiente instrucción a realizar.




## 7. ¿Qué es HTML?

HTML no es un lenguaje de programación, HTML es un lenguaje de marcado de hipertexto o "HyperText Markup Language" por el desarrollo de sus iniciales en inglés, básicamente este lenguaje se escribe en su totalidad con elementos, estos elementos están constituidos por etiquetas, contenido y atributos.

HTML es un lenguaje que interpreta el navegador web para mostrar los sitios o aplicaciones web tal y como estamos acostumbrados.

## 8. ¿Qué es CSS?

CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados "documentos semánticos"). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.

Al crear una página web, se utiliza en primer lugar el lenguaje HTML/XHTML para marcar los contenidos, es decir, para designar la función de cada elemento dentro de la página: párrafo, titular, texto destacado, tabla, lista de elementos, etc.

Una vez creados los contenidos, se utiliza el lenguaje CSS para definir el aspecto de cada elemento: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc.

## 9. ¿Cuál es la estructura básica de una página en HTML, explique cada parte?

Estructura básica en HTML

Cada página HTML debe constar de dos secciones:

1. **El HEAD** o cabecera, donde está la información sobre la página web, como el título, una breve descripción y algunas palabras clave. También puede contener información de estilos (CSS) y librerías JavaScript.
2. **El BODY** o cuerpo, donde está todo el contenido que el usuario va a ver en su navegador.

Las siguientes etiquetas HTML se utilizan para construir la estructura básica de una web:

<html> significa el inicio (y final) del documento.

<head> significa el inicio (y final) de la sección de cabecera del documento.

<title> Es el título del documento. Esto se muestra por la mayoría de los navegadores como título de la ventana, cuando agregas una web a favoritos, etc. Los motores de búsqueda como Google suelen utilizarlo también como título en los resultados de búsqueda, por lo que es muy importante para la optimización de la página web y el SEO en general. Por todo esto, debes escoger muy bien el título de cada página de tu sitio web y procurar que todos sean únicos y descriptivos.

<body> significa el inicio (y final) del contenido visible del documento (lo que ve el usuario final).

## 10. ¿Cuál es la estructura de un documento en CSS?

Al igual que los documentos HTML, los documentos CSS son archivos de texto donde se escribe una serie de órdenes y el cliente (navegador) las interpreta y aplica a los documentos HTML asociados.

**Sintaxis básica**

La estructura CSS se basa en reglas que tienen el siguiente formato:


1. **Selecto**r: El selector es el elemento HTML que vamos a seleccionar del documento para aplicarle un estilo concreto. Por ejemplo, con p seleccionaríamos todas las etiquetas <p> del HTML. Más adelante veremos que esto puede ser mucho más complejo, y dedicaremos una serie de capítulos exclusivamente a este tema.
2. **Propiedad**: La propiedad es una de las diferentes características que brinda el lenguaje CSS y que aplicaremos al selector para darle estilo.
3. **Valor**: Cada propiedad CSS tiene una serie de valores concretos a que se le pueden asignar, con los que tendrá uno u otro comportamiento.

Con todo esto le iremos indicamos al navegador que, para cada etiqueta (selector especificado) debe aplicar las reglas (propiedad y valor) indicadas.

Vamos a verlo con un ejemplo para afianzar conceptos. Tenemos el siguiente código HTML:

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Título de página</title>
    <link rel="stylesheet" href="index.css" />
  </head>
  <body>
    <div id="first">
      <p>Párrafo</p>
    </div>
    <div id="second">
      <span>Capa</span>
    </div>
  </body>
</html>
```

Además, por otro lado, en el archivo index.css indicado en la etiqueta <link> tenemos el siguiente código CSS que vemos a continuación:

```css
p {
  color: red; /* Color de texto */
}
```

En este caso, estamos seleccionando todas las etiquetas <p> del documento HTML (en este ejemplo es una sola, pero si existieran más se aplicaría a todas), y les aplicaremos el estilo indicado: color de texto rojo.

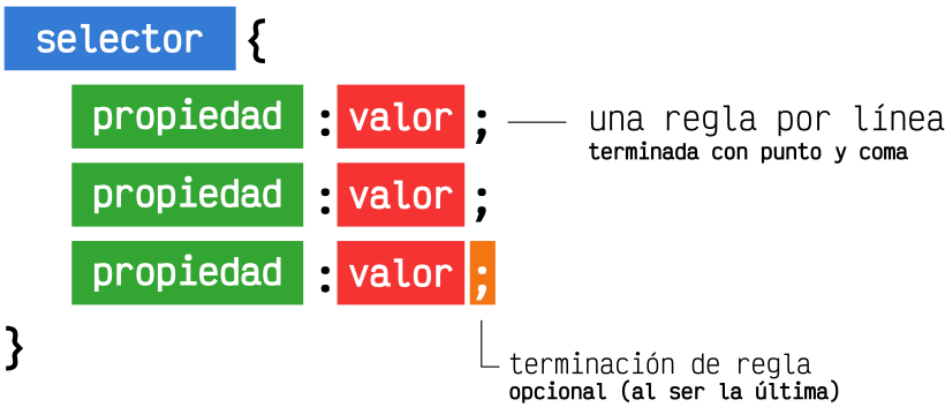
Ojo: Se pueden incluir comentarios entre los caracteres /* y */, los cuales serán ignorados por el navegador. Estos suelen servir para añadir notas o aclaraciones dirigidas a humanos.

Sin embargo, esto es sólo un ejemplo muy sencillo. Se pueden aplicar muchas más reglas (no sólo una, como el color del ejemplo), consiguiendo así un conjunto de estilos para la etiqueta indicada en el selector.

Cada una de estas reglas se terminará con el carácter punto y coma (;), seguido de la siguiente regla. El último punto y coma es opcional y se puede omitir si se desea:


Sin embargo, el código CSS se va haciendo más grande a medida que escribimos, por lo que colocar las reglas una detrás de otra sería muy poco legible. Por esa misma razón, se suele indentar el código. Esto no es más que utilizar una determinada estructura visual (utilizando saltos de línea, es decir, ENTER) de modo que en cada línea sólo haya una regla como máximo.

Esto se considera una buena práctica, indispensable a la larga, que nos facilitará la lectura, escritura y comprensión del código rápidamente:



Esto mejora sustancialmente la legibilidad del código y se considera un convenio que debe utilizarse para ayudar a entender más rápidamente el código ajeno (o incluso el nuestro) sin necesidad de necesitar mucho tiempo.

## 11. ¿Cómo se realizan las referencias dentro del CSS a clases o IDs dentro de HTML?

Clases (class) e Identificadores (id), ¿Cuál es mejor usar? Esta es una duda muy habitual entre cualquiera que se inicie en el apasionante mundo de CSS (Cascading Style Sheets).

La cuestión, a la hora de maquetar el HTML de una página web con CSS, es cómo se decide qué propiedades se le aplican a cada elemento y en qué orden.

Para especificar esto se usan los selectores, que, como su propio nombre indica, sirven para seleccionar los elementos a los que vamos a aplicar las propiedades que declararemos en nuestro código CSS.

La forma más sencilla es simplemente usar selectores de elementos. Por ejemplo:

```
1  body {color:#000;}
```

Esta regla haría que todo el texto dentro de la etiqueta body sea de color negro.

Pero a lo mejor queremos que el texto de los párrafos (identificados con el elemento <p>) tenga un color diferente, digamos, gris oscuro. Pues definimos el color gris oscuro para los elementos de párrafo.

```
1  p {color:#333;}
```

Claro que ahora quizá nos interesa destacar alguno de los párrafos de manera especial. Por ejemplo, poner un párrafo importante de color naranja o un párrafo que es resumen de todo el contenido en color púrpura y letra itálica (para gustos, colores...). ¿Cómo le decimos al navegador esto? En este caso podemos hacer uso de clases e Ids para especificar estas propiedades.

**Nota**: En realidad existen muchos más tipos de selectores y pseudo-selectores para identificar elementos, aquí nos centraremos en las clases e identificadores.

Así, al siguiente párrafo le podemos añadir la clase “destacado”:

```
1  <p class="destacado">Aquí va el texto que quiero destacar.</p>
```

Y en CSS lo ponemos en naranja a través de la propiedad “color”. Para marcar que es una clase anteponemos un punto al nombre de la clase:

```
.destacado {color:orange;}
```

Y obtendríamos:

> Aquí va texto que quiero destacar.

Pero es que si lo hacemos con un ID:

```
<p id="destacado">Aquí va el texto que quiero destacar.</p>
```

En el código CSS, para indicar que es un ID, anteponemos el símbolo almohadilla (#) en vez del punto:

```
#destacado {color:red;}
```

En este caso obtendríamos exactamente el mismo resultado:

> Aquí va texto que quiero destacar.

A primera vista, clases e IDs parece que sirven para lo mismo y que se emplearán de la misma manera: podemos aplicar propiedades CSS a elementos HTML a través de ambos y así modificar el aspecto visual y posicionamiento de estos al renderizarse en el navegador.

### 12. ¿Cuándo hay que usar clases y cuándo IDs?

> La regla más importante a la hora de elegir entre clases e ID's es que **un ID sólo debe ser usado una vez en el documento**.

- Es decir, una vez que asignamos un ID a un elemento no se puede volver a asignar a otro elemento de la misma página. ¡Ojo! Cuando hablamos de página nos referimos a la página que se está cargando actualmente en el navegador, no al sitio completo.
- Las clases, en cambio, las podemos usar las veces que queramos dentro del mismo documento. Así es que, si hay una serie de propiedades comunes entre una serie de elementos de la misma página lo apropiado será usar clases. Si dentro de esos elementos hay uno que queremos destacar de una manera especial una única vez en la página, entonces usaremos un ID.

En nuestro pequeño ejemplo, usaríamos clases para los párrafos "destacados" porque lo más probable es que haya más de un párrafo que queramos destacar y así podemos controlar la

apariencia de todos los párrafos destacados desde un único lugar. En cambio, también lo más probable es que sólo haya un párrafo que sea el resumen de todos los demás. En este caso usaríamos un ID, y nuestro código quedaría así:

```
<p id="resumen">Este es el resumen del texto y quiero que vaya en itálica.</p>
<p>Esto sería un párrafo de texto normal.</p>
<p>Esto sería otro párrafo de texto normal.</p>
<p>Un nuevo párrafo de texto normal.</p>
<p class="destacado">Aquí va texto que quiero destacar.</p>
<p>Esto sería otro párrafo de texto normal.</p>
<p>Más texto normal.</p>
<p class="destacado">Aquí iría más texto que quiero destacar.</p>
<p>Esto sería otro párrafo de texto normal.</p>
<p>Esto sería otro párrafo de texto normal.</p>
```

Y empleando **clases** sería:

```
body {color:#000;}
p {color:#333;}
#resumen {color:green;font-style:italic;}
.destacado {color:red;}
Y se vería así:
```

*Este es el resumen del texto y quiero que vaya en itálica.*
Esto sería un párrafo de texto normal.
Esto sería otro párrafo de texto normal.
Un nuevo párrafo de texto normal.
Aquí va texto que quiero destacar.
Esto sería otro párrafo de texto normal.
Más texto normal.
Aquí iría más texto que quiero destacar.
Esto sería otro párrafo de texto normal.
Esto sería otro párrafo de texto normal.

## 13. Explique al menos 6 atributos que usted puede cambiar a un elemento desde CSS.

1. **Font-size**:

Define el tamaño de la fuente y el valor se puede escribir en pixels o en ems. En este momento se recomienda usar ems. Los dos son valores relativos, el pixel es un valor relativo a la resolución de la pantalla, pero el em es relativo al tamaño de la fuente definida por el usuario. Si el usuario no cambió la configuración, el valor por defecto de los textos en todos los navegadores es de 16px. Entonces 1em = 16px.

2. **Color**: Define el color de la tipografía. Los colores se pueden escribir de 3 formas distintas: con sistema hexadecimal, por ejemplo: #FF0000 (es rojo). Con los nombres de los colores (más limitado) por ejemplo: black, red, green. O usando RGB, esta paleta permite agregar el canal alfa para hacer transparencias.
3. **Width**: Define el ancho de un elemento, el valor se puede escribir en pixels, ems o porcentaje.
4. **Max-width o min-width**: Definen el ancho máximo o mínimo de un elemento. Muy importante en sitios adaptables
5. **Height**: Define el alto de un elemento, el valor se puede escribir en pixels, ems o porcentaje.
6. **Max-height o min-height**: Definen el alto máximo o mínimo de un elemento. Muy importante en sitios adaptables
7. **Paddin**g: Es la distancia desde el borde de un elemento hasta su contenido.
8. **Margin**: Es la distancia entre un elemento y otro (desde el borde de un elemento hacia afuera)
9. **Border**: Define el borde de un elemento, su color, su estilo y grosor.

10. **Background**: Define los fondos de un objeto. El fondo puede ser una imagen o un color. El color puede ser pleno o degradado. La imagen se puede repetir formando una trama (es lo que ocurre por defecto) o se puede especificar que no repita y que se coloque en determinada posición.