UNIVERSIDAD AUTONOMA DEL CARMEN

Facultad de Ciencias de la información
Academia de programación

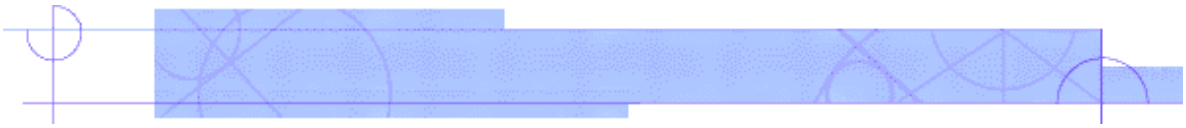
MANUAL DE PRÁCTICAS

“Desarrollo avanzado de Aplicaciones”
“Aplicaciones con bases de datos
Desarrollo de aplicaciones Web”

DESARROLLADO Y APROBADO POR LA ACADEMIA DE PROGRAMACIÓN

noviembre 2023

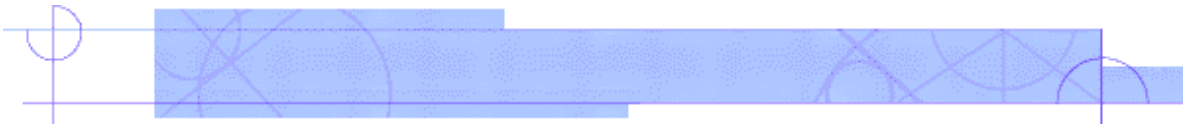
Versión 1.0

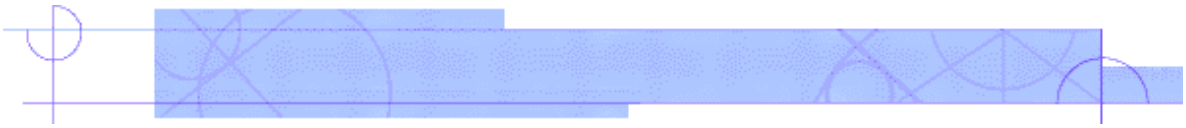


ELABORADO POR

MI. Jesús Alejandro Flores Hernández

Contenido





INTRODUCCIÓN

En este manual de prácticas se realizarán programas que usan técnicas de programación y plataformas web, en la solución de diferentes problemas que permiten al alumno desarrollar sus habilidades como programador WEB.

Este manual esta pensado para el uso de estudiantes de programación avanzada.

El paradigama de programación que se usa en este material esa el de programación estructurada.

Para este material el alumno, deberá tener conocimientos de técnicas, herramientas y lenguajes como los de:

- Programación estructurada
- Programación orientada a objetos
- Diseño de bases de datos
- MySql
- API
- JS
- Curl

PRACTICA 1. Plataforma de trabajo

1.1. Objetivo

El alumno será capaz de instalar la plataforma de trabajo sobre la que construirán su aplicación.

1.2. Equipo

Computadora personal, 3 navegadores web (Chrome, Mozilla, Opera, Safari, Brave).

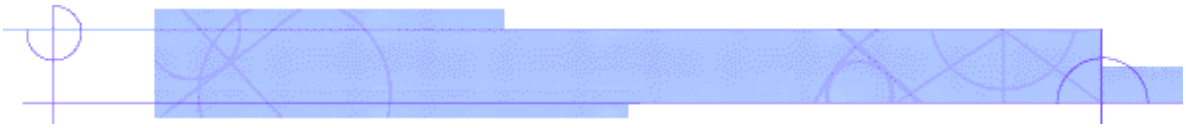
1.3. Materiales

Computadira, internet.

1.4. Descripción

Para el trabajo de este curso necesitaremos:

- Editor de Código



- Gestor de base de datos
- Navegador web
- Herramientas de prueba y desarrollo

1.4.1 Editor

Para el editor de código usaremos el editor de código libre Visual Studio code
Lo puede instalar desde: https://code.visualstudio.com/download

Necesitaremos agregar algunas extensiones a VSC para mejorar la productividad al escribir nuestros proyectos, extensiones:

- ***Thunder Client***. Esta extensión nos permite probar las API REST que vayamos creando. Básicamente permite hacer y enviar solicitudes http a nuestro código, realiza las mismas operaciones que ***curl***, pero en un ambiente más amigable. Será usada en la codificación del backend.

1.4.2 Gestor de base de datos Xamp/Wamp

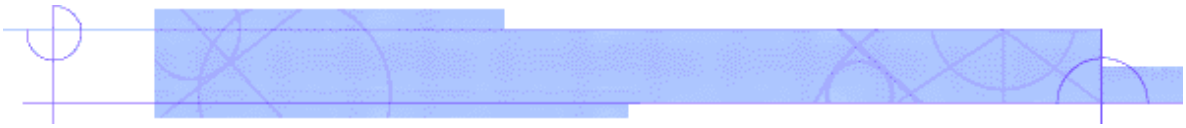
Para gestionar nuestras bases de datos de MYSql usaremos XAMP/WAMP. WAMP es un acrónimo que significa Windows, Apache, MySQL y PHP. Es un stack o conjunto de soluciones de software que significa que cuando instalas WAMP, estás instalando Apache, MySQL y PHP en tu sistema operativo (Windows en el caso de WAMP). Aunque puede instalarlos por separado, por lo general son empaquetados, y también por una buena razón.

WAMP deriva de XAMP (la X significa Linux). La única diferencia entre estos dos es que WAMP se usa para Windows, mientras que XAMP para sistemas operativos basados en Linux.

Puede usar cualquiera de los 2, en este curso asumimos que usara XAMP, lo puede descargar desde: https://www.apachefriends.org/es/download.html

1.4.3 Base de datos SQLite

También usaremos bases de datos de bolsillo SQLite, es una base de datos autocontenida en un archivo que no requiere gestor de base de datos, es muy ligera u útil para aplicaciones con pocos datos.



Un administrador para SQLite lo puede descargar desde:
https://sqliteadmin.orbmu2k.de/

también puede administrar la base de datos de SQlite desde Hidi sql

Heidi sql es una aplicación que permite administrar base de datos en varios esquemas como MySql, SqLite, PosGreSsql, Etc, lo puede descargar desde:
https://www.heidisql.com/

I.5. Procedimiento de la práctica

1. En la introducción se describen una serie de requerimientos para el desarrollo de sus aplicaciones.
2. Lea la descripción en la introducción, ubique los materiales a instalar y proceda a realizar la práctica.

I.6. Práctica

1. Instale el editor VSC y la exptencion Thundel Client, agregue también la extensión: Sqlite Viewer.
2. Instale XAMP en su computadora.
3. Instale Heidi Sql en su computadora desde: (https://www.heidisql.com/) como administrador de Mysql.
4. Descargue/instale SQLiteAdmin.
5. Proponga al profesor la base de datos para su proyecto inicial y una vez aprobada, diseñe sus tablas y consultas.

PRACTICA 2. Base de datos

2.1. Objetivo

El alumno será capaz de implementar sus bases de datos.

2.2. Equipo

Computadora personal, dispositivo de almacenamiento secundario (disquete, memoria USB, etc.), software.

2.3. Materiales

Internet, Computadora, Manuales.

2.4. Introducción

Se asume que tiene conocimientos básicos de sql, en caso contrario, vea en los anexos de este documento una introducción a sql.

2.4.1 MySql

Asumiendo que tiene instalado XAMP Server, arranque el servidor de MySql. Si no tiene instalado XAMP, descárguelo desde: https://www.apachefriends.org/es/download.html

Heidi SQL es un software libre y de código abierto para administrar bases de datos, permite conectarse a servidores MySQL, MariaDB, Microsoft SQL Server, PostgreSQL y SQLite.

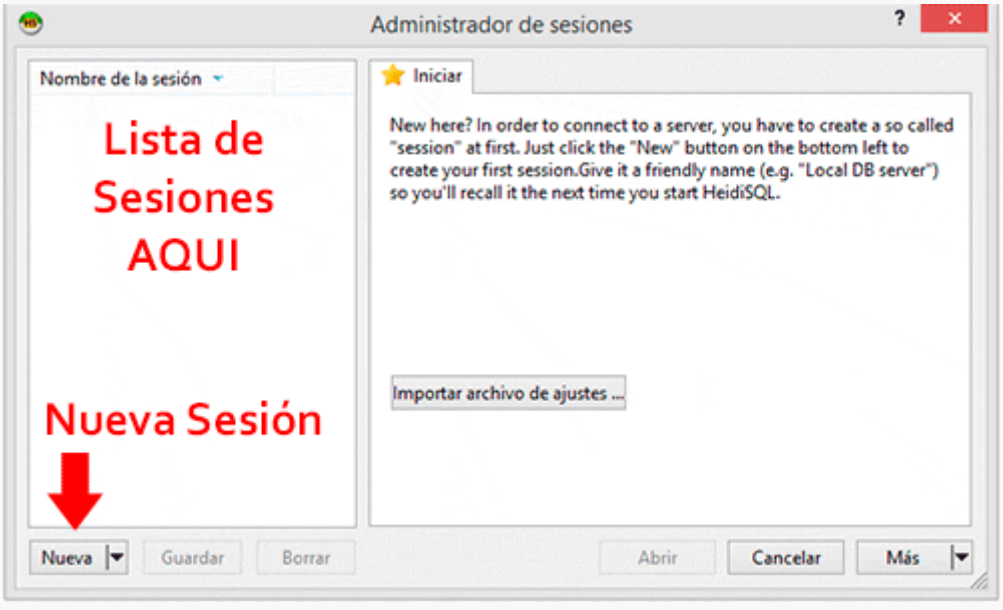
Descargue Heidi desde: sitio: http://www.heidisql.com/download.php e instálelo.

Iniciar MySQL

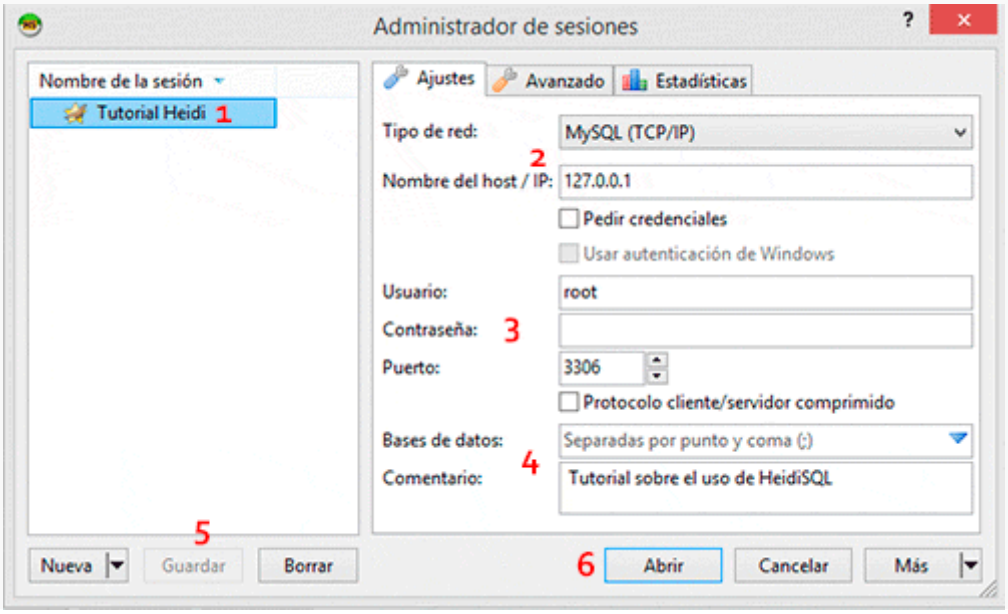
Si no tiene iniciado MySQL, arranque el servidor..

Sesión en Heidi

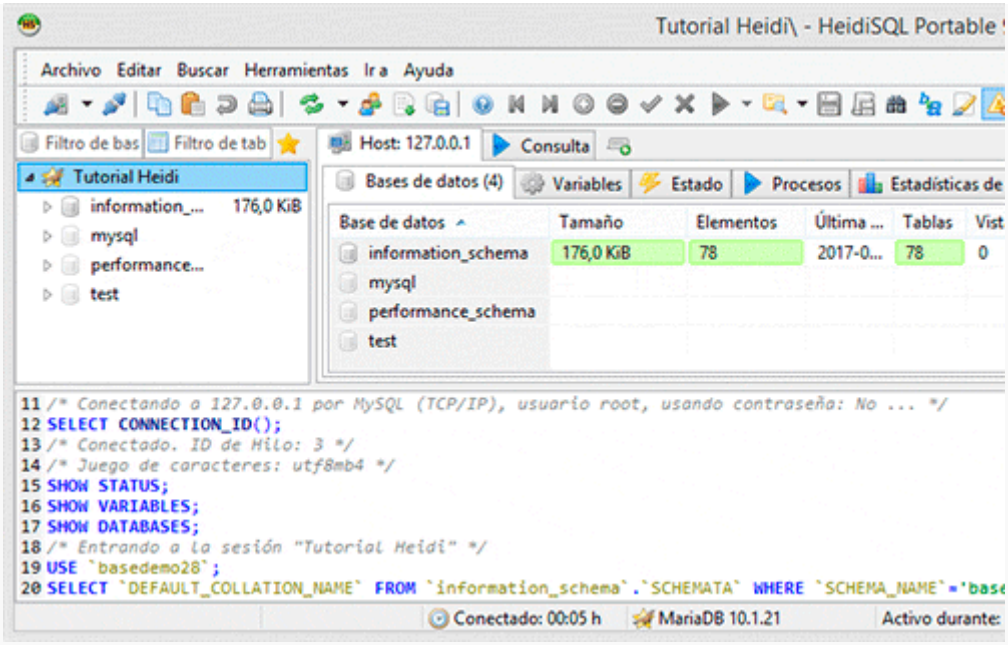
Arranque el Heidi SQL (haciendo doble clic en el ícono del HeidiSql)



Heidi SQL cuenta un sistema de sesiones para gestionar el trabajo. Cuando iniciamos el programa, debemos seleccionar con qué sesión queremos trabajar o en su defecto, crear una nueva sesión presionando el botón «***Nueva***».



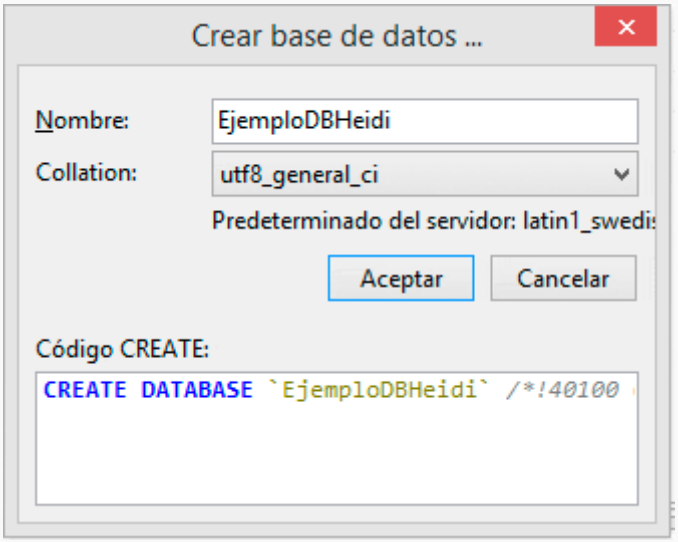
1. Escribimos el nombre de la sesión, por ejemplo «**Tutorial Heidi**»
2. Seleccionamos el Tipo de Red (MySQL TCP/IP) y el nombre del Host lo dejamos en: 127.0.0.1
3. Escribimos el nombre de usuario *(root por defecto en MariaDB)* y contraseña *(si tuviera)* en blanco si no tiene. Podemos indicar también otro puerto de conexión (deje el que trae por defecto).
4. Seleccionamos el o las bases de datos con lo que trabajaremos *(no es obligatorio)*, caso contrario dejamos en blanco.
5. Presionamos «*Guardar*» para grabar los cambios
6. Y finalmente presionamos «*Abrir*»



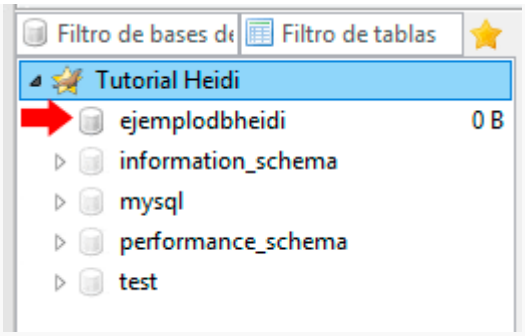
La interfaz de Heidi es intuitiva y además esta en español.

Crear Base de Datos

Hacemos clic derecho sobre el nombre de la sesión, elegimos «*Crear Nuevo*» → «*Base de Datos*«, se nos presenta una ventana de dialogo, donde escribimos el nombre de la base de datos «*EjemploDBHeidi*«, Seleccionamos el «*Collation*» (juego de caracteres asociadas a una base de datos).

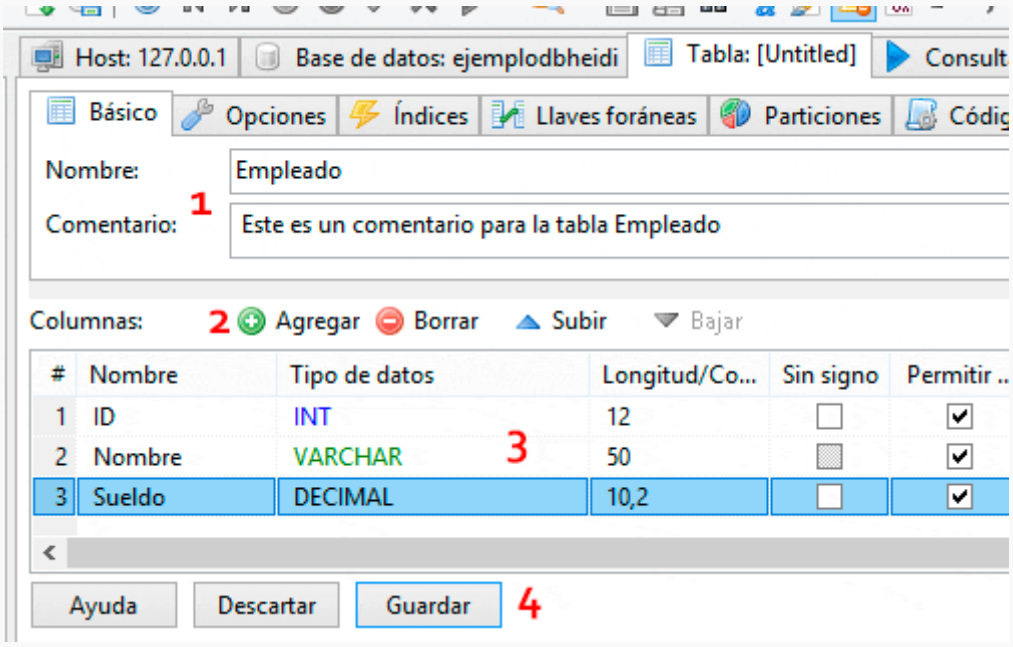


Para terminar la creación, presionamos «*Aceptar*»



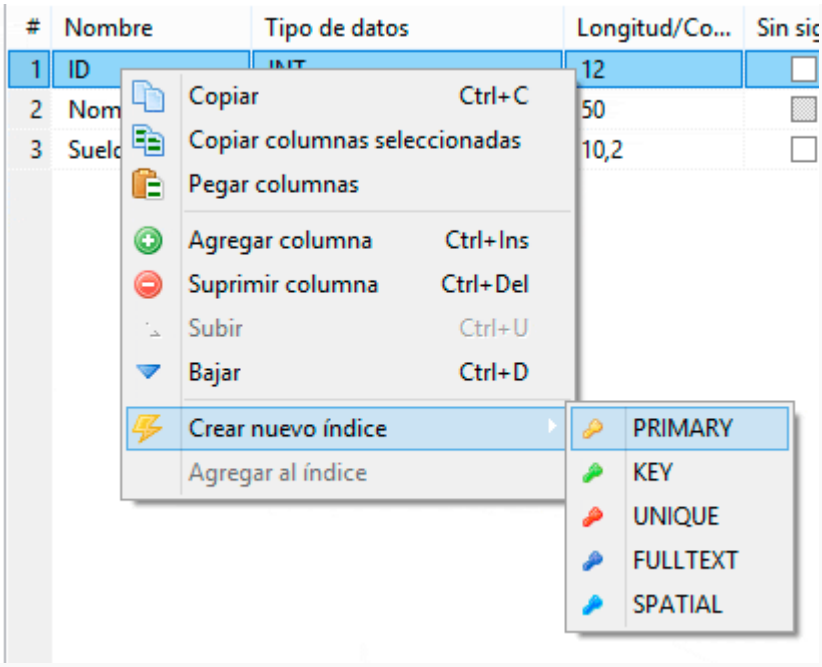
Crear Tabla

Clic derecho sobre la base de datos donde se quiere crear la tabla, seleccionar «*Crear nuevo*» → «*Tabla*«.
1: Escribimos un nombre para la tabla
2: Para crear nuevas columnas (campos) en la tabla, presionamos el botón «*Agregar*»
3: Escribimos nombre de la columna e indicamos sus propiedades
4: Para salvar los cambios, presionamos «*Guardar*»

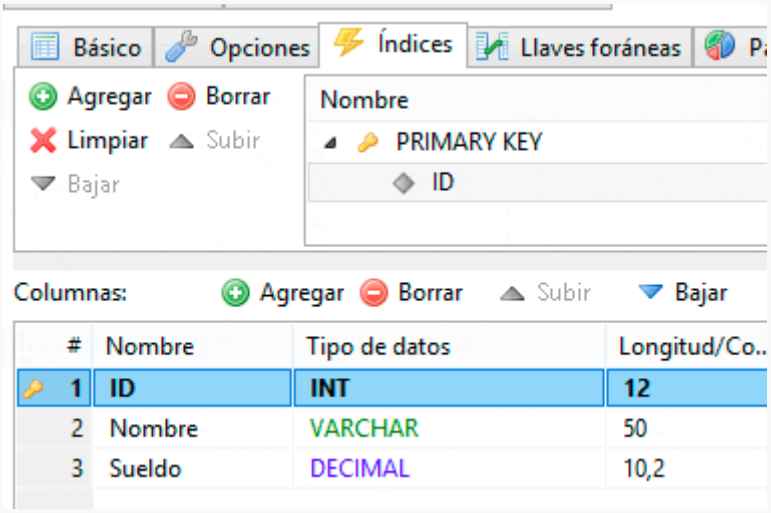


Llave primaria

Clic derecho sobre el campo que queramos definir como Llave Primaria, «*Crear Nuevo Indice*» → **PRIMARY**

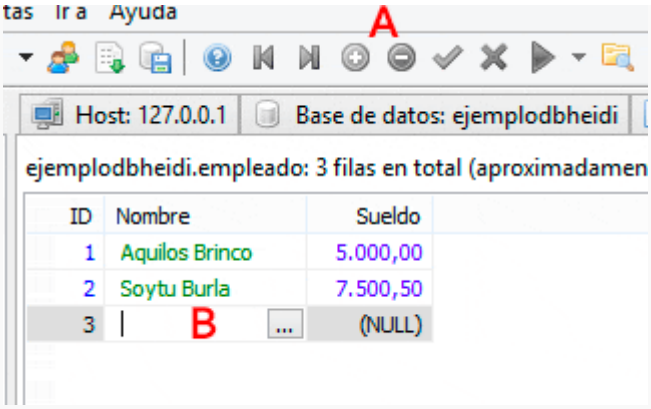


El campo será marcado con una llave para indicar que corresponde a una **PRIMARY KEY**, además en la pestaña «*Indices*» se puede gestionar las llaves.



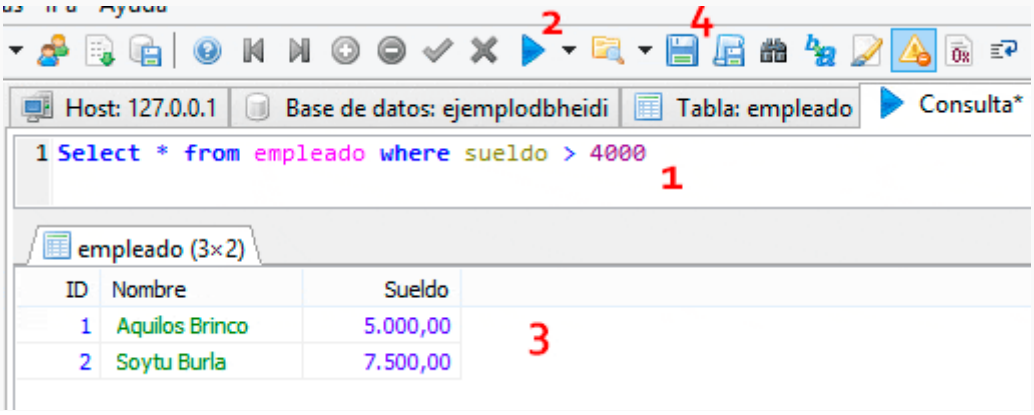
Agregar datos a tabla

Para agregar datos a una tabla, seleccionamos la misma, y abrimos la pestaña «*Datos*»
A: Para agregar o remover filas, utilizamos lo botones (+) y (-)
B: Para agregar datos a cada campo, doble clic y escribir su contenido



Consultas

Para realizar consultas a la base de datos, contamos con la pestaña «> *Consulta*»
1: Área donde escribir SQL
2: Botón para ejecutar la consulta
3: Área de resultados
4: Te permite guardar la consulta en archivo con extensión *.sql



2.4.2 SQLite

SQLite es una biblioteca de funciones que permite implementar una base de datos transaccional auto contenida en un archivo, como tal no se instala pues no existe un DBMS, lo que puede hacer es descargar alguna herramienta que le permita crear una base de datos en SQLite. Algunas formas de crear una base de datos en SQLite:

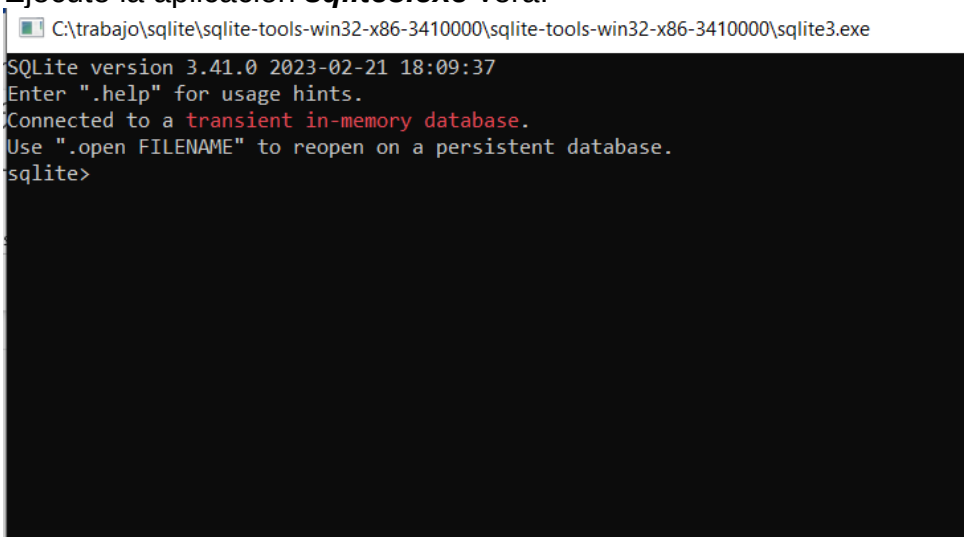
- Mediante código PHP (o algún otro lenguaje que tenga instalado el driver para SQLite)
- Usando el software libre SQLite Admin.
- Usando un complemento pata VS code.

- Usando un complemento para Firefox.
- Usando el Shell sqlite3.
- Usando la aplicación Heidi SQL.

La documentación sobre SQLite la puede encontrar en:
https://www.sqlite.org/docs.html
En la liga de descargas: https://www.sqlite.org/download.htm ,puede descartar el ***Shell comand sqlite3.***

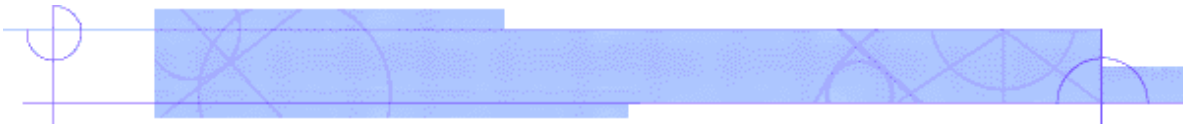
Este Shell viene en el archivo ***sqlitetools: sqlite-tools-win32-x86-3410000.zip*** , que está en la página de descarga de SQLite. Una vez descargado, en su disco vera un archivo sqlite-tools-win32-x86-3410000.zip, descomprímalo. En la carpeta que descomprimió busque sqlite3.exe.

Ejecute la aplicación ***sqlite3.exe*** verá:



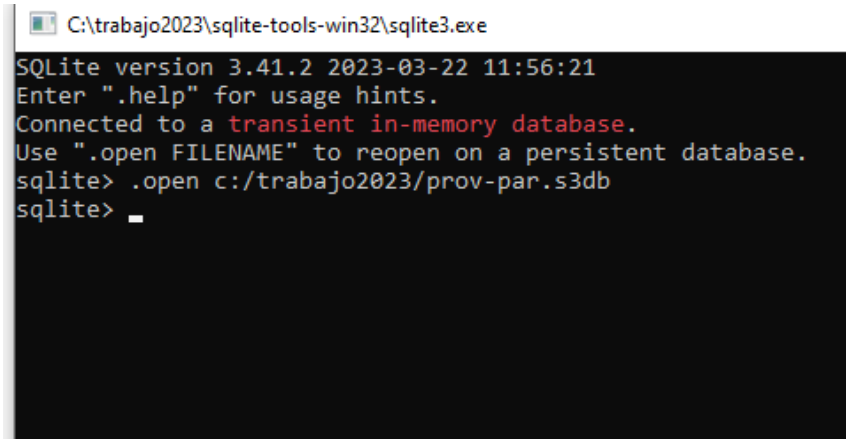
Aquí puede ejecutar sus comandos.
Comandos básicos.

Comandos del shell comand sqlite3
//cambiar al directorio de trabajo (debe escribir el punto) sqlite>.cd c:/trabajo //muestra el directorio de la carpeta corriente sqlite>.shell dir //abrir base de datos que está en el archivo “provpar.s3db” //si la base de datos no existe la crea automáticamente sqlite >.open provpar.s3db

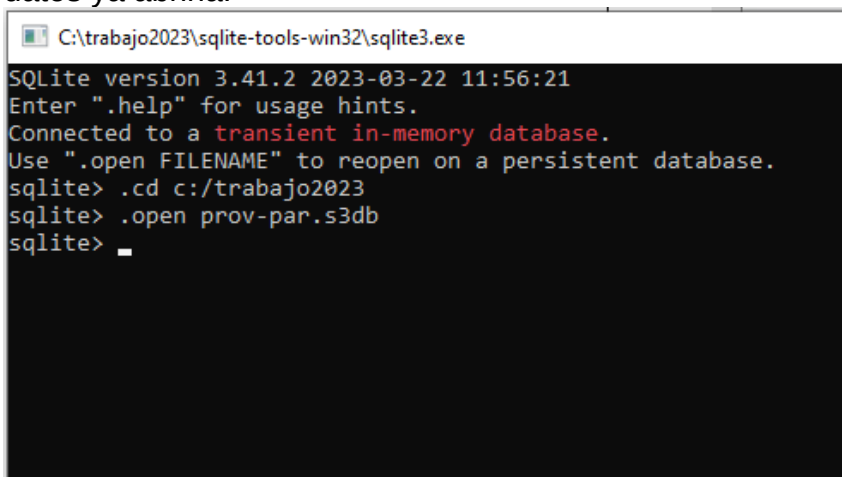


```
mostrar tablas de la base de datos
sqlite >.tables
//mostrar estructura de tabla
sqlite >.schema partes
//crear una tabla
sqlite>sqlite> CREATE TABLE tbl2 (
   ...>   f1 varchar(30) primary key,
   ...>   f2 text,
   ...>   f3 real
   ...> );
sqlite>
ejecutar sentencia sql
sqlite>select * from partes;
//opcionalmente antes de ejecutar sus select puede activar
//un despliegue por columnas de las tablas
Sqlite>.mode column
//otros modos son: table, json, list
//muestra las bases de datos abiertas
sqlite>.databases

//ejecuta el código SQL de un archivo
sqlite>.read archivo.sql

//guarda en un archivo la base de datos que está en memoria
Sqlite>.save archivo.db
salir de sqlite3
>.quit

referencia: https://sqlite.org/cli.html
```

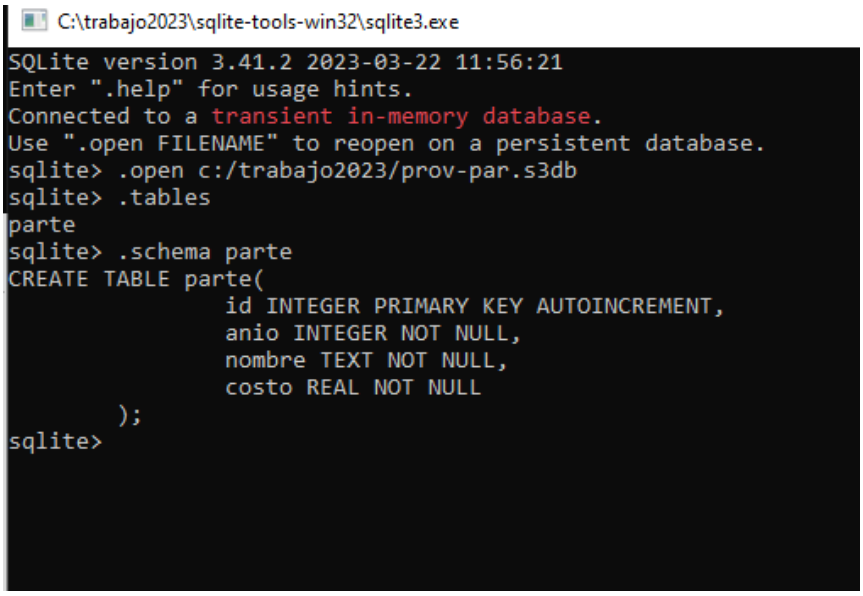
Por ejemplo, si queremos abrir (o crear) una base de datos llamada ‘prov-par.s3db’ hacemos:



Otra forma de hacerlo es primero cambiarnos al directorio donde está la base de datos ya abrirla:



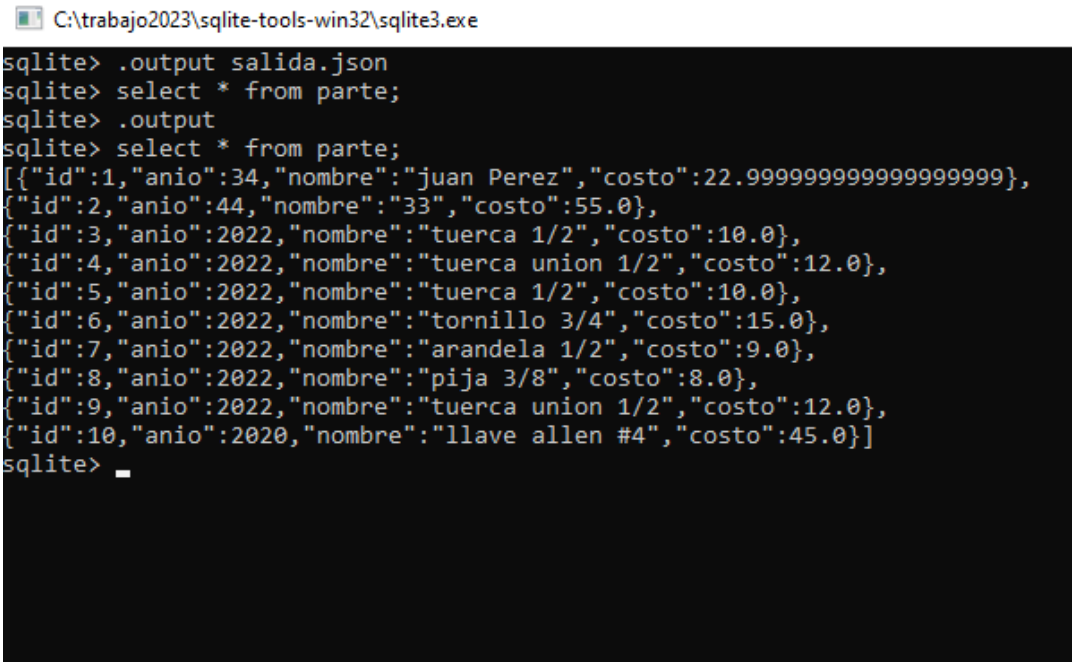
Ahora mostramos las tablas que contiene y el esquema de la tabla parte:



Ejemplo salida en formato json:

Suponga que desea guardar la salida de una sentencia select en formato json en el archivo ‘salida.json’. Necesita:

1. Activar la salida al archivo ‘salida.json’ (.output salida.json)
2. Ejecutar el select (select * from parte;)
3. En este momento ya tiene en su carpeta el archivo salida.json con el resultado de la consulta.
4. Volver a activar la salida a consola (.output)
5. Fin



Ejemplo renderizar SQL.

Suponga que quiere renderizar a sentencias SQL la base de datos parte y sus tuplas(registros).

C:\trabajo2023\sqlite-tools-win32\sqlite3.exe

```
sqlite> .dump parte
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE parte(
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                anio INTEGER NOT NULL,
                nombre TEXT NOT NULL,
                costo REAL NOT NULL
        );
INSERT INTO parte VALUES(1,34,'juan Perez',23.0);
INSERT INTO parte VALUES(2,44,'33',55.0);
INSERT INTO parte VALUES(3,2022,'tuerca 1/2',10.0);
INSERT INTO parte VALUES(4,2022,'tuerca union 1/2',12.0);
INSERT INTO parte VALUES(5,2022,'tuerca 1/2',10.0);
INSERT INTO parte VALUES(6,2022,'tornillo 3/4',15.0);
INSERT INTO parte VALUES(7,2022,'arandela 1/2',9.0);
INSERT INTO parte VALUES(8,2022,'pija 3/8',8.0);
INSERT INTO parte VALUES(9,2022,'tuerca union 1/2',12.0);
INSERT INTO parte VALUES(10,2020,'llave allen #4',45.0);
COMMIT;
sqlite>
```

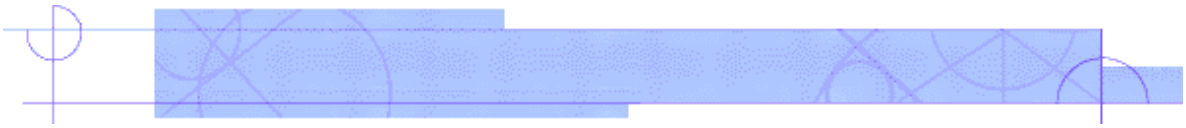
Pruebe los otros comandos.

Ejemplo crear bd a partir de un archivo SQL.

Objetivo: Crear la base de datos carreras con 2 tablas (carreras y alumnos) e insertar 4 carreras y 6 alumnos utilizando el shell sqlite3.exe.

Suponga que después de analizar la información que se requiere, usted diseño las sentencias SQL para crear y poblar las tablas, a continuación, el archivo de texto ‘alumnos.sql’ con las sentencias diseñadas por usted:

Archivo: alumnos.sql
CREATE Table if not exists carreras(id integer PRIMARY KEY autoincrement, alias VARCHAR(10), nombre VARCHAR(30), creditos integer



```
    );

insert into
    carreras(alias, nombre, creditos)
values
(
        'ico',
        'ingenieria en computación',
        100
    );

insert into
    carreras(alias, nombre, creditos)
values
(
        'isc',
        'ingenieria en sistemas computacionales',
        110
    );

insert into
    carreras(alias, nombre, creditos)
values
(
        'itcc',
        'ingenieria en tcnoligias de computo y comunicaciones',
        105
    );
CREATE Table
    if not exists alumnos(
        id integer PRIMARY KEY autoincrement,
        matricula VARCHAR(10),
        nombre VARCHAR(30),
        carrera integer,
        foreign key (carrera) references carreras
    );
    insert into alumnos(matricula, nombre, carrera) values('222120','juan perez',1);
    insert into alumnos(matricula, nombre, carrera) values('232122','ana puc',1);
    insert into alumnos(matricula, nombre, carrera) values('232124','rosa uc',2);
    insert into alumnos(matricula, nombre, carrera) values('222126','luis ruiz',3);
    insert into alumnos(matricula, nombre, carrera) values('232127','paco lin',2);
    insert into alumnos(matricula, nombre, carrera) values('202127','rita xa',3);
```

A continuación, ejecutamos la aplicación sqlite3.exe y ponemos los comandos:

```
.open midb.s3db
.read alumnos.sql
```

Esto crea la base de datos midb.s3db y ejecuta sobre la base de datos los comandos sql del archivo alumnos.sql. A continuación, hagamos unas consultas sobre las tablas creadas

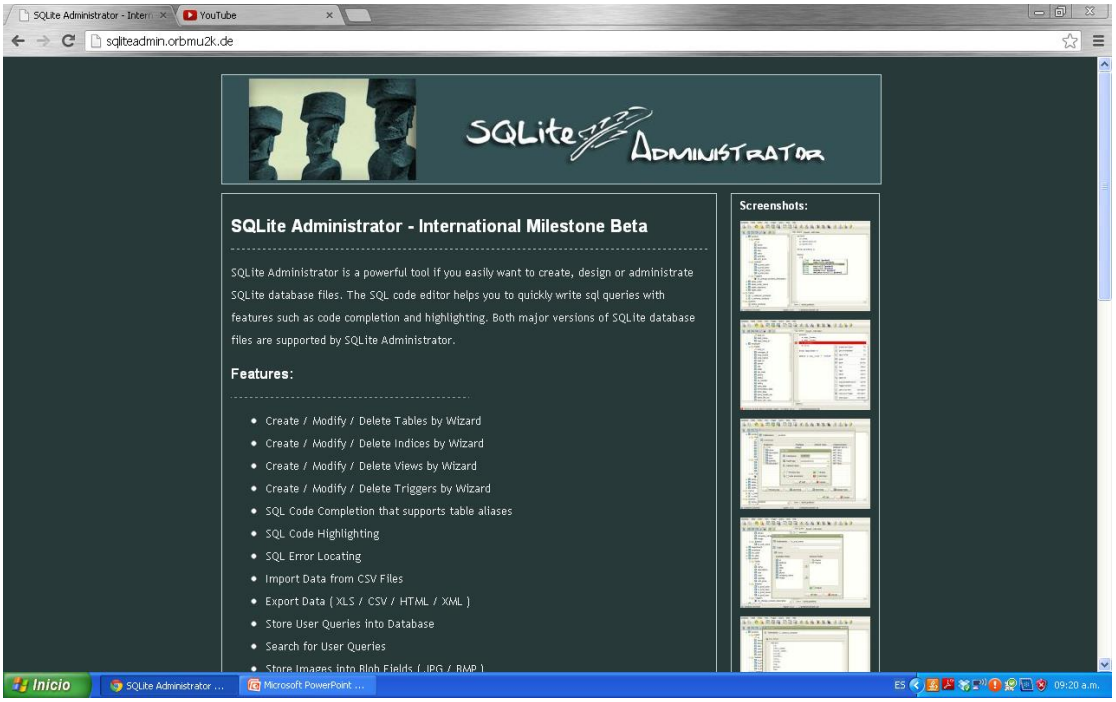
```
.mode list
select * from carreras;
select * from alumnos;
```

Pantalla de ejecución del shell:

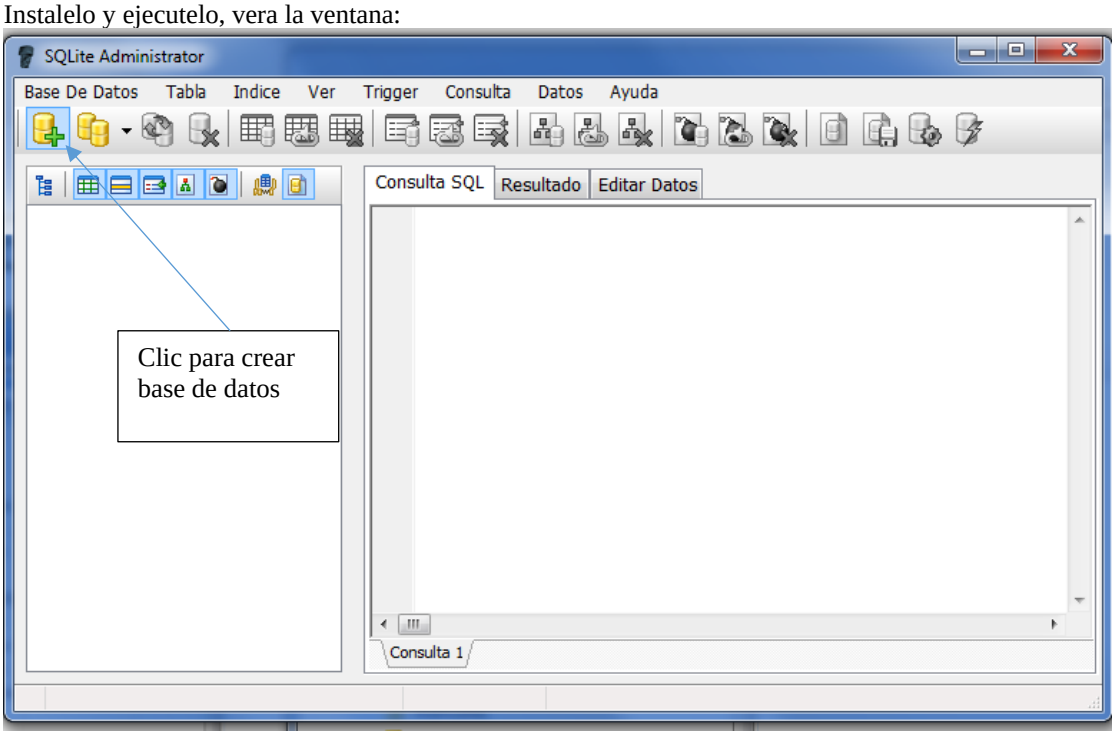
```
SQLite version 3.41.2 2023-03-22 11:56:21
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .read alumnos.sql
sqlite> .mode list
sqlite> select * from carreras;
1|ico|ingenieria en computación|100
2|isc|ingenieria en sistemas computacionales|110
3|itcc|ingenieria en tcnoligias de computo y comunicaciones|105
sqlite> select * from alumnos;
1|222120|juan perez|1
2|232122|ana puc|1
3|232124|rosa uc|2
4|222126|luis ruiz|3
5|232127|paco lin|2
6|202127|rita xa|3
sqlite>
```

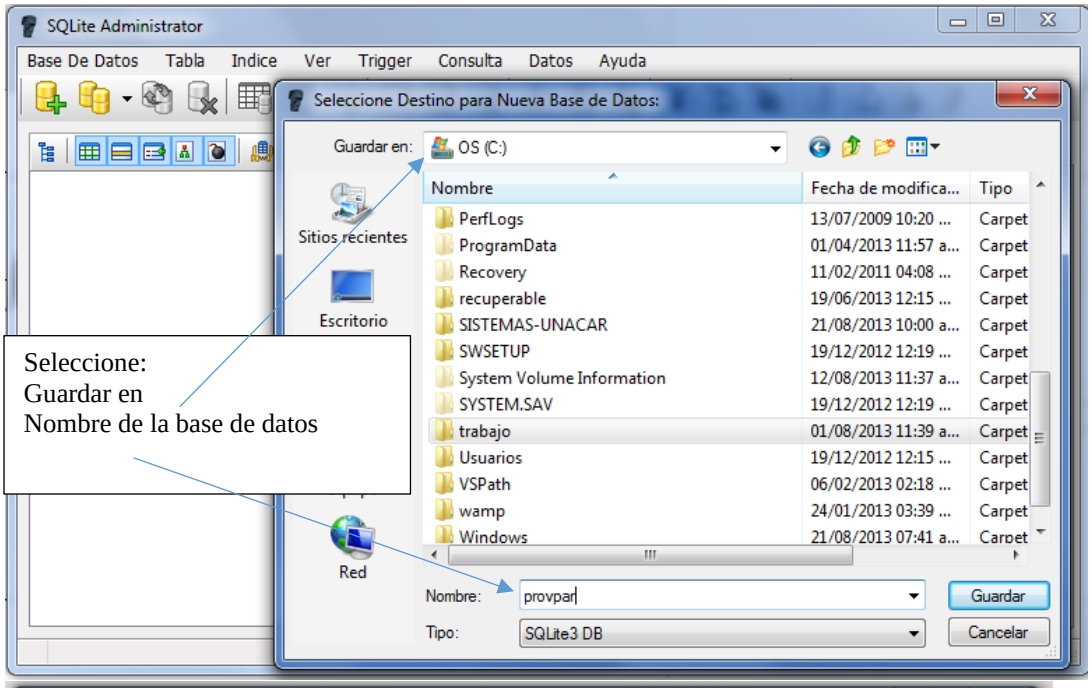
2.4.3 SQLiteAdministrator

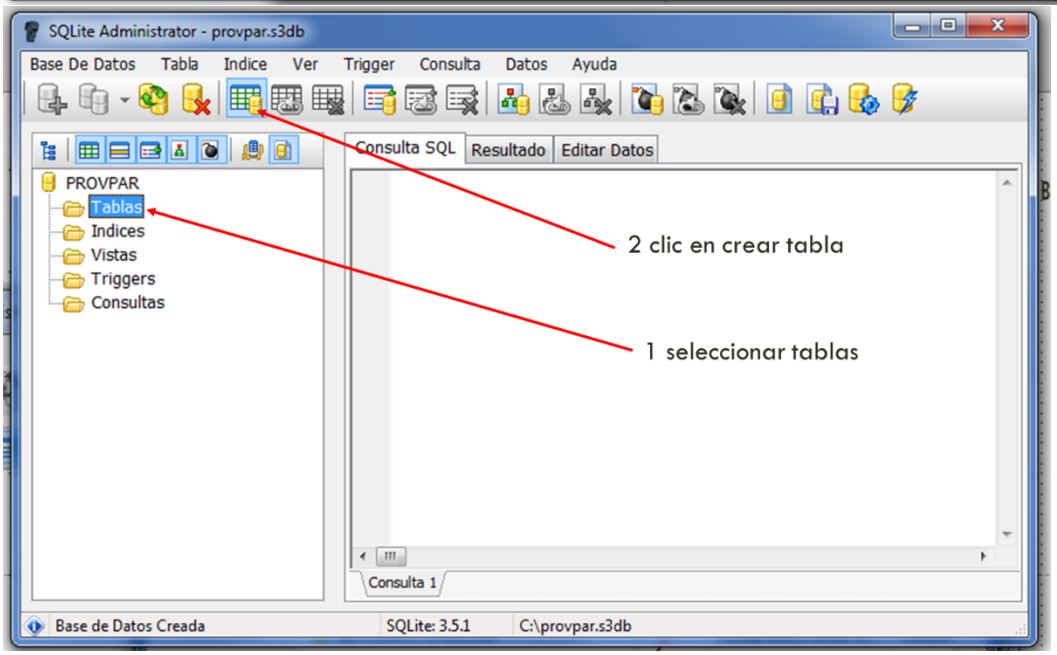
Aunque es útil, puede ser frustrante trabajar con linea de comandos, como solucion usaremos SQLITE-Admin, es una aplicación visual para crear y manipular base de datos de SQLite. Lo puede descargar desde: http://sqliteadmin.orbmu2k.de/

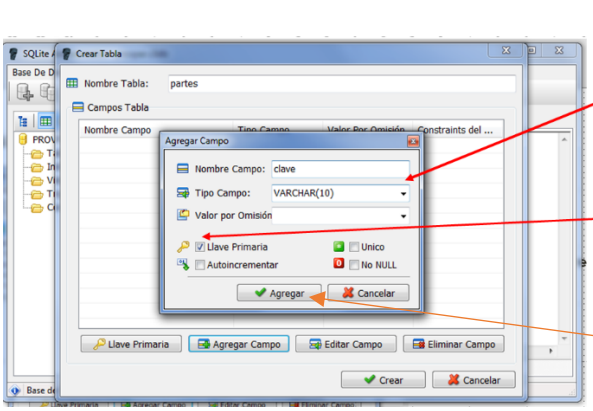


Instalelo y ejecutelo, vera la ventana:




SQLite Administrator
Base De Datos Tabla Indice Ver Trigger Consulta Datos Ayuda
Seleccione Destino para Nueva Base de Datos:
Guardar en: OS (C:)
Sitios recientes
Escritorio
Red
Nombre
Fecha de modifica...
Tipo
PerfLogs 13/07/2009 10:20 ... Carpet
ProgramData 01/04/2013 11:57 a... Carpet
Recovery 11/02/2011 04:08 ... Carpet
recuperable 19/06/2013 12:15 ... Carpet
SISTEMAS-UNACAR 21/08/2013 10:00 a... Carpet
SWSETUP 19/12/2012 12:19 ... Carpet
System Volume Information 12/08/2013 11:37 a... Carpet
SYSTEM.SAV 19/12/2012 12:19 ... Carpet
trabajo 01/08/2013 11:39 a... Carpet
Usuarios 19/12/2012 12:15 ... Carpet
VSPath 06/02/2013 02:18 ... Carpet
wamp 24/01/2013 03:39 ... Carpet
Windows 21/08/2013 07:41 a... Carpet
Nombre: provpar
Tipo: SQLite3 DB
Guardar
Cancelar
Seleccione:
Guardar en
Nombre de la base de datos

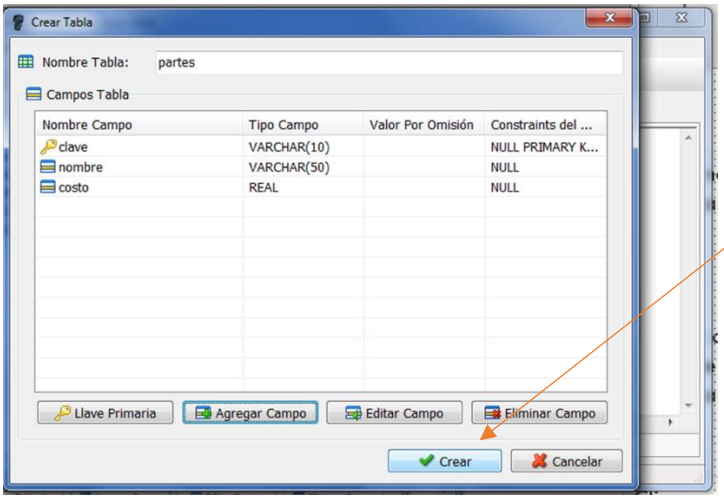

SQLite Administrator - provpar.s3db
Base De Datos Tabla Indice Ver Trigger Consulta Datos Ayuda
PROVPAR
Tablas
Indices
Vistas
Triggers
Consultas
Consulta SQL
Resultado
Editar Datos
Consulta 1
2 clic en crear tabla
1 seleccionar tablas
Base de Datos Creada
SQLite: 3.5.1
C:\provpar.s3db



Dar nombre y tipo de campo

Clic aquí si es llave primaria el campo

Clic en agregar para agregar el campo



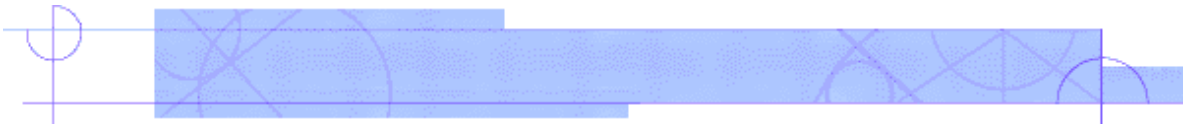
Clic para crear la tabla

2.5. Procedimiento de la práctica

1. Lea la introducción de este tema y Con una base de datos que indique el profesor proceda:

2.6. Prácticas

1. Crear la base de datos e insertar datos de prueba en MySql y en SQlite.
2. Pruebe las consultas que indique el profesor.
3. Valide la base de datos y las consultas con el profesor.
4. Documente en un archivo Word el diseño y construcción de su base de datos.



PRACTICA 3. HTML

3.1. Objetivo

El alumno será capaz de elaborar interfaces de programas que den solución a problemas de gestión de datos, en la parte de interfaz, usando html.

3.2. Equipo

Computadora personal, dispositivo de almacenamiento secundario (disquete, memoria USB, etc.), navegador web.

3.3. Materiales

Editor de código, navegadores, manuales.

3.4. Descripción

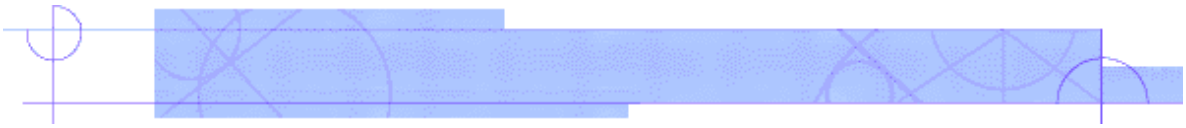
HTML5 Elementos

HTML, que significa Lenguaje de Marcado de Hipertextos (HyperText Markup Language), es la pieza más básica para la construcción de la web y se usa para definir el sentido y estructura del contenido en una página web. Otras tecnologías además de HTML son usadas generalmente para describir la apariencia/presentación de una página web (CSS) o su funcionalidad (JavaScript).

"Hipertexto" se refiere a los enlaces que conectan las páginas web entre sí, ya sea dentro de un mismo sitio web o entre diferentes sitios web. los vínculos son un aspecto fundamental de la web. Al subir contenido a Internet y vincularlo a páginas creadas por otras personas, te haces participante activo en la red mundial (World Wide Web).

HTML usa "marcado" (markup en inglés) para anotar textos, imágenes y otro contenido para ser mostrado en un navegador web. El marcado en HTML incluye "elementos" especiales tales como <head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img>, <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, <ul>, <ol>, <li>, y muchos otros más.

Un elemento HTML se separa de otro texto en un documento por medio de "etiquetas", las cuales consisten en elementos rodeados por "<" y ">". El nombre de un elemento dentro de una etiqueta no es sensible a mayúsculas. Esto es, puede ser escrito en mayúsculas, minúsculas o una combinación. Por ejemplo, la etiqueta <title> puede ser escrita como <Title>, <TITLE> o de cualquier otra forma.



3.4.1 Tags HTML

El principio esencial del lenguaje HTML es el uso de las etiquetas (tags). Funcionan de la siguiente manera:

<XXX> Este es el inicio de una etiqueta.
</XXX> Este es el cierre de una etiqueta.

Las letras de la etiqueta pueden estar en mayúsculas o minúsculas, indiferentemente. Lo que haya entre ambas etiquetas estará influenciada por ellas. Por ejemplo, todo el documento HTML debe estar entre las etiquetas <HTML> y </HTML>:

<HTML> [Todo el documento] </HTML>

Un documento HTML en sí está dividido en dos zonas principales:

- El encabezado, comprendido entre las etiquetas <HEAD> y </HEAD>
- El cuerpo, comprendido entre las etiquetas <BODY> y </BODY>
- El lenguaje de marcas de hipertexto (HTML) funciona por medio de marcas o tags y casi todos tienen inicio y final.

3.4.2 Encabezado

En el encabezado podemos poner el título del documento, comprendido entre las etiquetas <TITLE> y </TITLE>

Ejemplo:

Archivo HTML: titulo.html

```
<!DOCTYPE html>
<html>

<head>
    <title>mi pagina web</title>
    <meta charset="UTF-8"/>
</head>

<body>
    <p>cuerpo de la página</p>
    <p>Este es un ejemplo de acentos á é í ó ú ñññ ÑÑÑ</p>
</body>
</html>
```

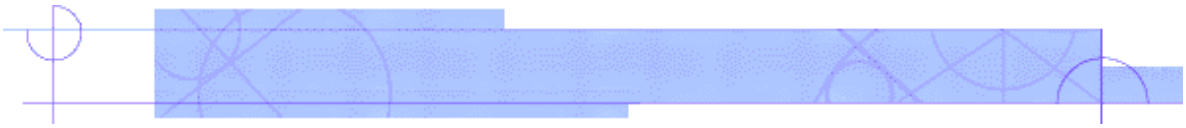


Tabla 1:

Nota: la primera línea: ***<!DOCTYPE html>*** Indica que estamos usando la versión 5 de HTML.

Ejecúte así su archivo HTML:

1. Cree una carpeta de trabajo, recomiendo que la carpeta no esté en ***mis documentos***, créela en el disco c por ejemplo así: ***c:trabajo/html***
2. Ponga el código de la tabla 1 (lo puede copiar desde su carpeta de ejemplos “códigos/html” con el nombre “titulo.html” que proporcionó el profesor) edítelo con Visual Studio Code (VSC),. Se debe ver como en la tabla 1. Guárdelo en: ***c:trabajo/html/titulo.html.***
3. En su carpeta ***trabajo/html*** de clic derecho en el archivo “titulo.html” y elija abrir con. Ahí selecciónе su navegador (alguno de estos: EDGE, Mozilla, Google Chrome, Opera) Chrome por ejemplo, como resultado verá el contenido de su página en la ventana en su navegador.

Código:

La marca: ***<!DOCTYPE html>*** indica que estamos usando HTML 5
<head> </head> definen el encabezado de la página
<body> </body> define el cuerpo de la página.

Dentro de <head> </head> también podemos ver la marca

<title> </title> que definen el título de la página.

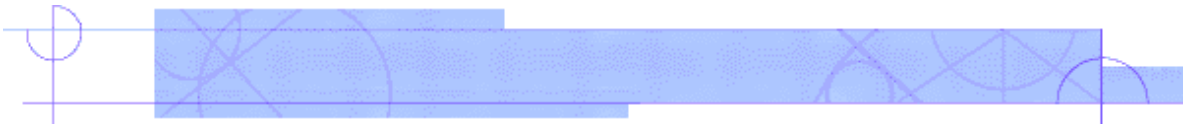
otros elementos que pueden ponerse en el <head> son:

<meta> para describir características de los datos, por ejemplo, el conjunto de caracteres a usar en la página:

<meta charset="UTF-8"/>

3.4.3 Cuerpo <body>

Aquí se ponen todos los elementos que definen la página, como cuadros de texto, combos, formularios, contenedores etcétera.



P. ej. Cuando queremos poner un texto sin ninguna característica especial, lo ponemos directamente poniendo el texto entre los tag que definen un párrafo :

*** mi parrafo ***

Ejemplo:
https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_paragraphs1

Otra etiqueta o tag son las h1, h2, hasta h6 que sirven para definir encabezados de diferentes tamaños, ejemplo:
https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_headers

Si queremos separar los párrafos, o cualquier otra cosa y ponerlo en una nueva, usamos una etiqueta
 (break, o romper). Este tag no tiene etiqueta de cierre.
Para destacar alguna parte del texto se pueden usar:

<B> y </B> para poner algo en negrita (bold).
<I> y </I> para poner algo en cursiva (italic).

Otra etiqueta interesante es <PRE> y </PRE>. El texto que se encuentre entre estas etiquetas estará preformateado, es decir que aparecerá como si hubiera sido escrito con una máquina de escribir, con una fuente de espaciado fijo (tipo Courier). Además, se respetarán los espacios en blanco y retornos del carro, tal como estaban en nuestro documento HTML. Es muy apropiada para confeccionar tablas y otros documentos similares. Ejemplo:
https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_pre

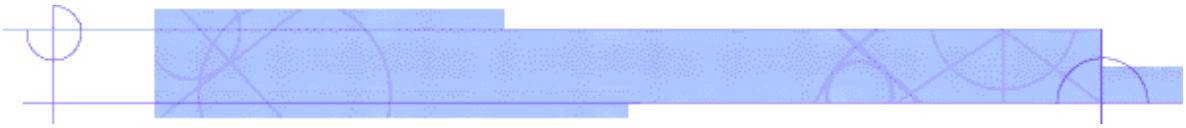
En las fórmulas matemáticas puede interesar poder escribir índices y subíndices, que se consiguen con las etiquetas y respectivamente. Ejemplo:
https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_sup

3.4.4 Atributos

Los elementos de HTML pueden tener atributos, por ejemplo en el tag ancla <a> , este te lleva a otro documento html o otra parte del mismo documento, para que el ancla sepa donde debe llevarte usa el atributo href="":

3.4.4.1 Anclas

Esta es mi liga



Este elemento <a> permite al usuario seguir la liga al sitio indicado en el atributo href.

Un ejemplo práctico:

a.html

```
<!DOCTYPE html>
<html>
    <body>
        <a href="https://www.w3schools.com/html/">Esta es mi liga</a>
    </body>
</html>
```

Tabla 2

Pruebe el código de la tabla 2 y de clic en el enlace, al hacer clic debe llevarlo a la nueva página.
Otros elementos pueden tener atributos:

```
<meta charset="UTF-8">
```

El elemento <meta> cuando se usa con el atributo: charset=”UTF-8” define el conjunto de caracteres a usar en la página.

3.4.4.2 Atributos id, name y class

Los tags pueden tener varios atributos entre ellos el atributo id que identifica de manera única al elemento en la página. El atributo id define un identificados únicono que no debe repetirse en la página.

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

Otro atributo es el atributo name que identifica de manera única a un tag dentro de un formulario:

```
<input type="text" id="fname" name="nombre">
```

El atributo name no debe repetirse en el formulario.

En estos ejemplos también se está usando el atributo type=”” que define el tipo de input que se desea, más adelante se explicará el input.

Otro atributo es el atributo class que identifica a un grupo de tags, este atributo puede repetirse y permite hacer referencia a un conjunto de controles o tags.

```
<input type="text" placeholder="celular" class="form-control" />
<button class=" form-control">nuevo</button>
```

Aquí además se muestra el atributo placeholder, se trata de un texto que aparece dentro de la caja de entrada de texto y que se usa para describir brevemente el propósito de la caja de texto.

ver ejemplo en: https://www.w3schools.com/tags/tryit.asp?filename=tryhtml5_input_placeholder

Para mayor referencia de id y class vea:

1. https://developer.mozilla.org/es/docs/Web/HTML/Atributos_Globales/class#:~:text=El%20atributo%20global%20class%20es,getElementsByClassName%20del%20DOM.
2. https://www.w3schools.com/html/html_id.asp

Otro atributo puede ser:

```
style="color:green"
```

Que permite poner color al texto.

2.4.5 Agrupadores

Contenedor de bloque es un contenedor que agrupa un conjunto de controles y pone una línea en blanco de separación al final.

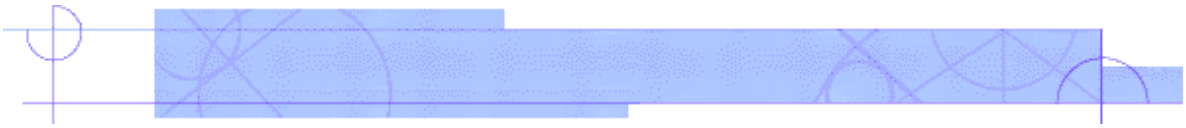
Un contenedor de línea agrupa uno o más controles sin moverlos del lugar donde se encuentran.

Los agrupadores sirven para agrupar a un conjunto de controles, para ponerles formato o enviar sus datos a aun servidor, por ejemplo.

2.4.5.1

Es un contenedor de bloque, permite agrupar un conjunto de controles para por ejemplo aplicarles un estilo en particular, o hacer referencia a ellos en forma grupal.

Ejemplo:
https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_div_test



2.4.5.2

Es un contenedor de línea, permite agrupar un conjunto de controles para por ejemplo aplicarles un estilo en particular, o hacer referencia a ellos en forma grupal.

Ejemplo:
https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_span

2.4.6 <form>

En un contenedor de bloque que permite agrupar otros elementos y se usa para crear formularios en html, por ejemplo:

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_submit

En los formularios usaremos tags o controles como:
Cuadros de texto que son los <input type=”text”>
Botones que son los <input type=”button”>
Check box que son los <input type="checkbox">
Radio button que son los <input type="radio">
Y otros.

2.4.6.1 listas

Podemos escoger entre tres tipos distintos:

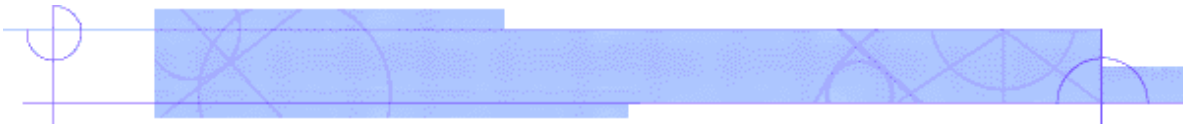
- Listas desordenadas (no numeradas)
- Listas ordenadas (numeradas)
- Listas de definición.

Las listas desordenadas (Unordered Lists) sirven para presentar cosas que, por no tener un orden determinado, no necesitan ir precedidas por un número. Su estructura es la siguiente:

<UL>

<LI> Un elemento

<LI> Otro elemento



<LI> Otro más

<LI> etc.

</UL>

Es decir, toda la lista está dentro de la etiqueta <UL> y </UL>, y luego cada elemento va precedido de la etiqueta <LI> (list ítem). El resultado de lo anterior es el siguiente:

Se puede anidar una lista dentro de otra. Por ejemplo:

<UL>

<LI> Mamíferos

<LI> Peces

<UL>

<LI> Sardina

<LI> Bacalao

</UL>

<LI> Aves

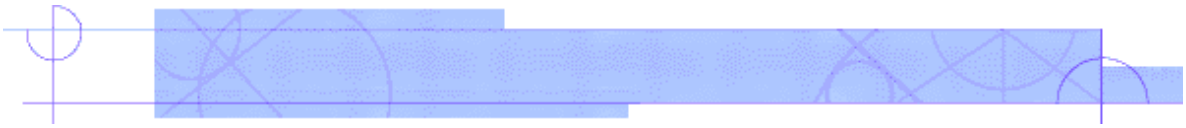
</UL>

Las listas ordenadas (Ordered Lists) sirven para presentar elementos en un orden determinado. Su estructura es muy similar a la anterior. La diferencia estriba en que en el resultado aparecerá automáticamente un número correlativo para cada elemento.

<OL>

<LI> Primer Elemento

<LI> Segundo Elemento



<LI> Tercer Elemento

<LI> etc.

</OL>

Al igual que las listas desordenadas, también se pueden anidar las listas ordenadas.

El tercer tipo lo forman las listas de definición. Como su nombre indica, son apropiadas para glosarios (o definiciones de términos). Toda la lista debe ir englobada entre las etiquetas <DL> y </DL>. Y a diferencia de las dos que hemos visto, cada renglón de la lista tiene dos partes:

- El nombre de la cosa a definir, que se consigue con la etiqueta <DT> (definition term).
- La definición de dicha cosa, que se consigue con la etiqueta <DD> (definition definition).

<DL>

<DT> Una cosa a definir

<DD> La definición de esta cosa

<DT> Otra cosa a definir

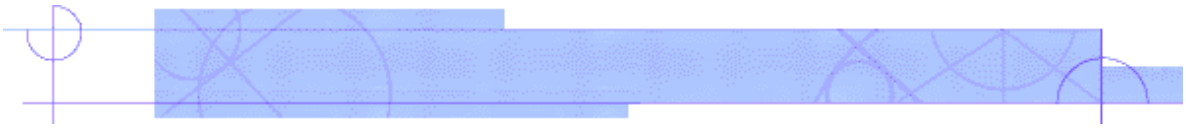
<DD> La definición de esta otra cosa

</DL>

Ejemplo listas: https://www.w3schools.com/html/tryit.asp?filename=tryhtml_lists_intro

2.4.6.2 tablas

Las tablas son un modo sencillo de disponer el texto en columnas, añadir un título a una ilustración, pero hay modos de sacar un gran partido de una característica aparentemente sencilla. La etiqueta <TABLE> puede ser una poderosa herramienta de formato. Se puede hacer, por ejemplo, no mostrar el borde de una tabla en absoluto. También se puede hacer uso de la etiqueta <TABLE> para ubicar texto e imágenes con precisión, en prácticamente casi cualquier lugar de una página.



Estructura de una tabla

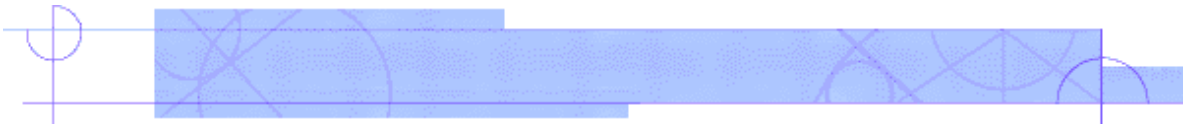
Vamos a ver ordenadamente (de fuera hacia dentro) las etiquetas necesarias para confeccionar las tablas.

<TABLE> [resto de las etiquetas] </TABLE>	**Es la etiqueta general, que engloba a todas las demás.**
<TABLE BORDER=n> [resto de las etiquetas] </TABLE>	**Presenta los datos tabulados con un borde, haciendo las tablas más atractivas, y el grosor es de n pixeles.**
<TR> [etiquetas de las distintas celdas de la primera fila] </TR>	**Permite formar cada fila de la tabla. Hay que repetirla tantas veces como filas queremos que tenga la tabla.**
<TD> [contenido de cada celda (imágenes, texto, etc.)] </TD>	**Permite formar las distintas celdas que contendrá cada fila de la tabla. Hay que repetirlas tantas veces como celdas queramos que tenga la fila.**
<TH> [encabezamiento de tabla] </TH>	**Es utilizada para colocar encabezamientos en negrita sobre las columnas**

Tabla 1.2-3

Ejemplo:

Tabla en HTML
<!DOCTYPE html> **<html>** **<body>** **<table>**



```
  <thead>
    <tr>
      <th>Month</th>
      <th>Savings</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>Sum</td>
      <td>$180</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>January</td>
      <td>$100</td>
    </tr>
    <tr>
      <td>February</td>
      <td>$80</td>
    </tr>
  </tbody>
</table>
</body>
</html>
```

Otro ejemplo de tabla en:
https://www.w3schools.com/tags/tryit.asp?filename=tryhtml_table_test

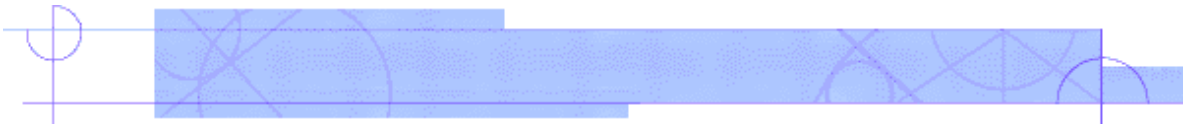
2.4.6.3 Input

Los campos de texto <input> son los controles de formulario básicos. Son un modo muy cómodo de permitir al usuario introducir cualquier tipo de datos.

El primero es la entrada de texto vea el ejemplo:

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_input_text
Tiene otros tipos como radio del que puede ver un ejemplo aquí:
https://www.w3schools.com/html/tryit.asp?filename=tryhtml_input_radio
el tipo check:
https://www.w3schools.com/html/tryit.asp?filename=tryhtml_input_checkbox
el tipo date:
https://www.w3schools.com/html/tryit.asp?filename=tryhtml_input_date
tipo number:
https://www.w3schools.com/html/tryit.asp?filename=tryhtml_input_number
el tipo button:
https://www.w3schools.com/html/tryit.asp?filename=tryhtml_input_button

Vea una referencia completa de las características del input en:
https://www.w3schools.com/html/html_form_input_types.asp



Realice las siguientes prácticas, no se preocupe por el formato.

3.5. Procedimiento

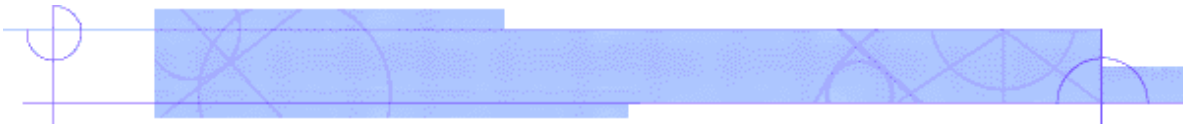
1. En la sección de códigos/html puede ver ejemplos de páginas html, úselos como referencia.
2. Abra su editor de código Visual Studio Code.
3. Codifique las prácticas en lenguaje html.
4. Guardar el programa y probarlo con 3 diferentes navegadores.

3.6. Prácticas

1. Construya una página web sin formatos, que muestre su curriculum.
2. Construir un menú basado en una lista (ver 3.3.1), cada elemento de la lista debe enviarlos a otra página en blanco (ver 3.4.4.1 referencias) páginas: altas, mostrar, buscar, bajas.
3. En la página de altas agregue el código html para capturar los datos de un alumno, datos: matricula, nombre, carrera, sexo, semestre, lenguajes que maneja (opciones c, vb, java, js, python), el alumno puede manejar mas de un lenguaje.
4. En la página de buscar agregue los controles para capturar la matricula del alumno y un botón para buscar.
5. En la página mostrar construya una tabla con datos de ejemplo.
6. En su carpeta de ejercicios “codigos\html\interfaz\ejemplo-0” pruebe el ejemplo menu.html. Compare con su ejercicio 2.
7. En su carpeta de ejercicios “codigos\html\interfaz\ejemplo-0” pruebe el ejemplo alta_persona.html. Compare con su ejercicio 3.
8. En su carpeta de ejercicios “codigos\html\interfaz\ejemplo-0” pruebe el ejemplo mostrar_personas.html. Compare con su ejercicio 5.

Nota. En estos ejercicios solo debe mostrarse la interfaz html, no deberá agregar funcionalidad ni estilos.

PRACTICA 4. CSS



4.1. Objetivo

El alumno será capaz de dar estilos a programas que se desplegarán e, un navegador.

4.2. Equipo

Computadora personal, dispositivo de almacenamiento secundario (disquete, memoria USB, etc.), navegadores, editor de código.

4.3. Materiales

Hojas de papel bond o cuaderno, lápiz o boligrafo.

4.4. Descripción

4.4.1 Sintaxis CSS

Para dar formato a una página web se utilizan las hojas de estilo CSS. Antes de empezar repase los conceptos de id, class, y los agrupadores, div, span, form, que se explicaron en la parte de html. El formato a una página web significa aplicar estilos a elementos web, estos elementos son los tags de html y el estilo es la forma en que se visualiza el tag, el estilo aplica bordes, colores, espaciado, dirección en que se despliega el tag etcétera.

Aunque no es la mejor práctica, se puede usar reglas de html en línea, esto consiste en aplicar estilos agregando atributos a sus elementos html.

ejemplo:

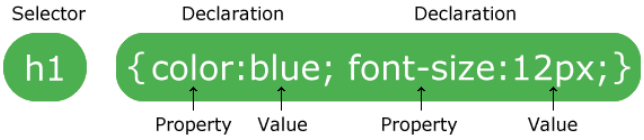
css-inline-01.html
<!DOCTYPE HTML> <html> <body>   letras grandes color azul   este es un texto con estilo </body> </html>

Tabla 4.1 ejemplo uso de css en línea.

En este ejemplo los estilos se aplican directamente en los tags. Pruebe el ejemplo de la tabla 4.1.

Una segunda y mejor opción, es agrupar todos los estilos entre las etiquetas: ***<style></style>*** en el head del documento. En este caso habrá que especificar a qué elementos se aplica el estilo.

Para este caso necesitamos definir una regla o conjunto de reglas CSS. Una regla consiste en un selector y un bloque de declaraciones:



El selector indica a que elementos se aplica el estilo. La regla anterior indica que los elementos HTML del tipo h1 tienen su propiedad *color* en azul y su tamaño de fuente en 12. El selector indica el elemento HTML al que se desea dar estilo. El bloque de declaraciones contiene una o más declaraciones de estilo separadas por punto y coma.

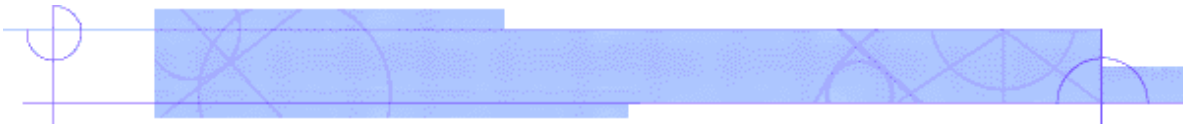
ejcss01.html

```
<!DOCTYPE html>
<html>
   <head>
      <style>
         p {
            color: red;
            text-align: center;
         }
      </style>
   </head>
   <body>

      <p>Hola mundo!</p>
      <p>Este parrafo tiene un estilo con CSS.</p>

   </body>
</html>
```

Tabla 4.2

La regla de la tabla 2:

```
p {
   color: red;
   text-align: center;
}
```



indica que los elementos HTML del tipo <p> (párrafo), deberán ir centrados y en color rojo. Como resultado deberá ver en su navegador dos líneas de texto centradas (text-align: center) y en color rojo:

Hola mundo

Este parrafo tiene un estilo con CSS.

4.4.2 Almacenamiento de reglas

Las reglas en CSS, se pueden almacenar de manera interna en la página HTML como en el ejemplo de la tabla 4.2, veamos ahora el almacenamiento de la misma regla en un archivo externo con extensión .css:

Archivo: ejcss03.html

```
<!DOCTYPE html>
<html>
   <head>
      <link rel="stylesheet" href="miEstilo.css">
   </head>
   <body>
      <p>Hola mundo!</p>
      <p>Este parrafo tiene un estilo con CSS.</p>
   </body>
</html>
```

Archivo: miEstilo.css

```
p {
    color: red;
    text-align: center;
}
```

Tabla 4.3 Estilos en un archivo css

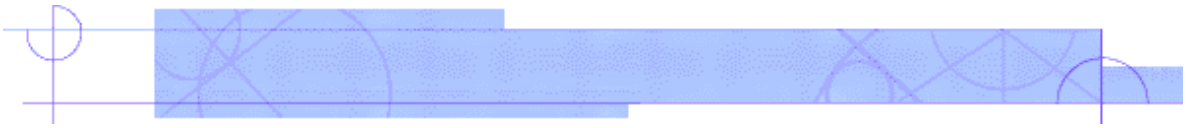
En su editor de textos escriba los dos archivos de la tabla 2 y guárdelos con los nombres indicados y en la misma carpeta, pruebe el archivo: ejcss03.html en su navegador.

4.4.3 Definiendo reglas

Las reglas se definen para tipos de elementos HTML, ejemplo:

Ejemplo

```
<!DOCTYPE html>
<html>
   <head>
      <style>
         body {
            background-color: linen;
```



```
        }
        h1 {
            color: maroon;
            margin-left: 40px;
        }
    </style>
</head>
<body>

    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>

</body>
</html>
```

Tabla 4 Reglas para body y p

Explicación:

La regla:

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

Define que al elemento body se le aplicara el color de fondo linen.

La regla:

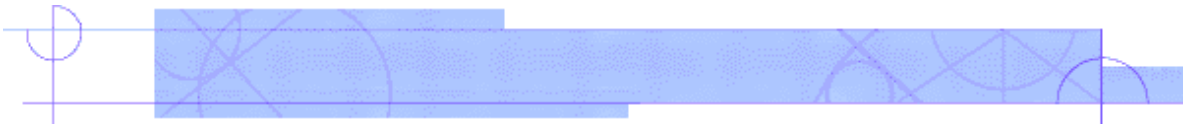
```
h1 {
    color: maroon;
    margin-left: 40px;
}
```

Indica que a los elementos h1 se les aplicará un color marrón y margen izquierdo de 40 pixels.

4.4.4 Selectores

Los selectores son empleados para encontrar o seleccionar elementos HTML por medio de su nombre de elemento, clase, id, atributo y más.

4.1 Selector de elemento por su tipo



Este selecciona los elementos basados en su tipo de elemento.

ejemplo:

```
p {
   text-align: center;
   color: red;
}
```

Tabla 4.5

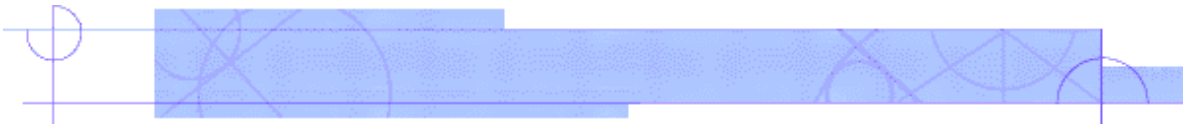
Selecciona todos los elementos <p> en la página, con esta regla todos los elementos <p> quedaran centrados y en color rojo.

Ejemplo de uso:

```
<!DOCTYPE html>
<html>
<head>
   <style>
      p {
         text-align: center;
         color: red;
        }
   </style>
</head>
<body>

   <p>Every paragraph will be affected by the style.</p>
   <p id="para1">Me too!</p>
   <p>And me!</p>

</body>
</html>
```

Tabla 4.6

3.4.5 Selector id



Este usa el atributo id de los elementos HTML para aplicar la regla, el id de un elemento HTML debe ser único dentro de la página. La siguiente regla se aplica al elemento con id="para1":

```
#para1 {
   text-align: center;
   color: red;
}
```

Tabla 4.7

Ejemplo de uso:

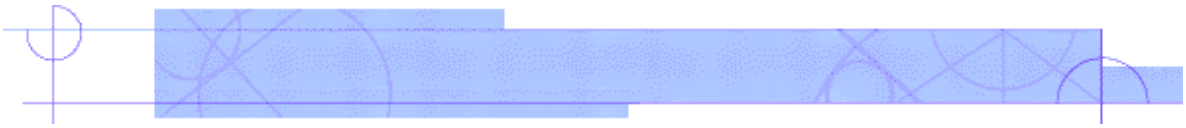
```
<!DOCTYPE html>
<html>
<head>
   <style>
      #para1 {
        text-align: center;
        color: red;
      }
   </style>
</head>
<body>

   <p id="para1">Hola mundo!</p>
   <p>Este párrafo no es afectado por el estilo.</p>

</body>
</html>
```

Tabla 4.8

3.4.6 Selector de clase

Este selector, selecciona elementos con un específico nombre de clase, varios elementos pueden tener la misma clase, incluso si no son del mismo tipo, el siguiente selector, solo los elementos <p> con clase "center" serán centrados:

```
.center {
   text-align: center;
```



```
   color: red;
}
```

Ejemplo de uso:

```
<!DOCTYPE html>
<html>
<head>
   <style>
      .center {
        text-align: center;
        color: red;
      }
   </style>
</head>
<body>

   <h1 class="center">Red and center-aligned heading</h1>
   <p class="center">Red and center-aligned paragraph.</p>

</body>
</html>
```

Tabla 4.9

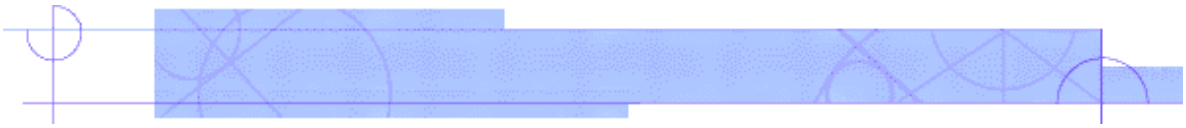
Usted puede indicar que solo un tipo específico de elementos HTML sea afectado por la clase:

```
p.center {
   text-align: center;
   color: red;
}
```

Esta regla selecciona los elementos p con clase center.

Ejemplo de uso:

```
<!DOCTYPE html>
<html>
<head>
```



```
   <style>
      p.center {
        text-align: center;
        color: red;
      }
</style>
</head>
<body>

   <h1 class="center">This heading will not be affected</h1>
   <p class="center">This paragraph will be red and center-aligned.</p>

</body>
</html>
```

Tabla 4.10

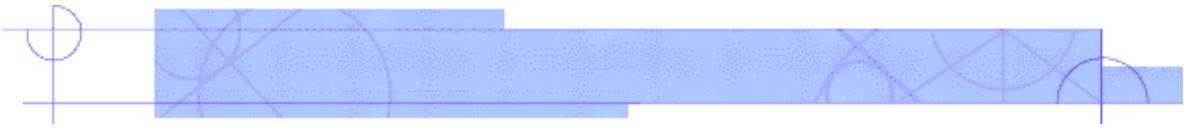
3.4.7 Mas sobre selector por tipo de elemento

Puede seleccionar todos los párrafos, o todas las tablas o todos los encabezados del tipo h1.

```
h1 {
   text-align: center;
   color: red;
}

h2 {
   text-align: center;
   color: red;
}

p {
   text-align: center;
   color: red;
}
```

Tabla 4.11

El ejemplo de selección anterior se puede también escribir como sigue:



```
h1, h2, p {
   text-align: center;
   color: red;
}
```

Tabla 12

Este último aplica el estilo a los elementos del tipo h1, h2 y p.

Ejemplo de uso:

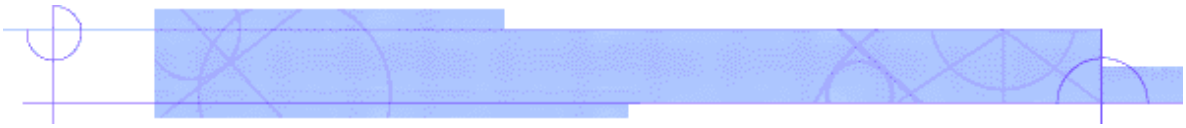
```
<!DOCTYPE html>
<html>
<head>
   <style>
      h1, h2, p {
         text-align: center;
         color: red;
      }
</style>
</head>
<body>

<h1>Hello World!</h1>
   <h2>Smaller heading!</h2>
   <p>This is a paragraph.</p>

</body>
</html>
```

Tabla 4.13

3.4.8 Resumen selectores

Selector	Ejemplo	Descripción
.class	.intro	Selecciona todos los elementos con class="intro"
#id	#firstname	Selecciona los elementos con id="firstname"
*	*	Selecciona todos los elementos
element	p	Selecciona todos los elementos <p>

element,element	div, p	Selecciona todos los elementos y todos los elementos <p>
element element	div p	Selecciona todos los elementos <p> dentro de los elementos
element>element	div > p	Selecciona todos los elementos <p> que tienen como padre a un elemento
element+element	div + p	Selecciona todos los <p> que se encuentran inmediatamente después de un elemento
element1~element2	p ~ ul	Selecciona cada elemento que esta precedido por un elemento <p>
[*attribute*]	[target]	Selecciona todos los elementos con atributo target
[*attribute=value*]	[target=_blank]	Selecciona todos los elementos con target="_blank"
[*attribute~=value*]	[title~=flower]	Selecciona todos los elementos con atributo title que contenga la palabra "flower"
[*attribute|=value*]	[lang|=en]	Selecciona todos los elementos con atributo lang que comienzan con "en"
[*attribute^=value*]	a[href^="https"]	Selecciona todos los elementos <a> cuyo valor de atributo href inicia con "https"
[*attribute$=value*]	a[href$=".pdf"]	Selecciona todos los elementos <a> cuyo valor de atributo href termina con ".pdf"
[*attribute*=value*]	a[href*="w3schools"]	Selecciona todos los elementos <a> cuyo valor de atributo href contiene la subcadena "w3schools"
:active	a:active	Selecciona los link activos
::after	p::after	Inserta algo después del contenido de cada <p>
::before	p::before	Inserta algo antes del contenido de <p>
:checked	input:checked	Selecciona cada elemento <input> seleccionado
:disabled	input:disabled	Selecciona todos los elementos <input> no seleccionados
:empty	p:empty	Selecciona todos los elementos <p> sin hijos (incluyendo nodos de texto)
:enabled	input:enabled	Selecciona todos los elementos <input> activos



:first-child	p:first-child	Selecciona cada elemento <p> que es el primer hijo de su padre
::first-letter	p::first-letter	Selecciona la primera letra de cada elemento <p>
::first-line	p::first-line	Selecciona la primera línea de cada elemento <p>
:first-of-type	p:first-of-type	Selecciona cada elemento<p> que es el primer elemento <p> de su padre
:focus	input:focus	Selecciona el elemento input que tiene el foco
:hover	a:hover	Selecciona los link con el ratón sobre ellos
:in-range	input:in-range	Selecciona los elementos input con un valor dentro de un rango específico
:invalid	input:invalid	Selecciona a todos los elementos input con un valor invalido
:lang(*language*)	p:lang(it)	Selecciona cada elemento <p> con un atributo lang igual a "it" (Italian)
:last-child	p:last-child	Selecciona cada elemento <p> que es el último hijo de su padre
:last-of-type	p:last-of-type	Selecciona cada elemento <p> que es el último elemento <p> de su padre
:link	a:link	Selecciona todos los links no visitados
:not(*selector*)	:not(p)	Selecciona cada elemento que no es un elemento <p>
:nth-child(*n*)	p:nth-child(2)	Selecciona cada elemento <p> que es el segundo hijo de su padre
:nth-last-child(*n*)	p:nth-last-child(2)	Selecciona cada elemento <p> que es el segundo hijo de su padre contado desde el último
:nth-last-of-type(*n*)	p:nth-last-of-type(2)	Selecciona cada elemento <p> que es el segundo elemento <p> de su padre, contado desde el último
:nth-of-type(*n*)	p:nth-of-type(2)	Selecciona cada elemento <p> que es el segundo elemento <p> de su padre
:only-of-type	p:only-of-type	Selecciona cada elemento <p> que es el único elemento <p> de su padre

:only-child	p:only-child	Selecciona cada elemento <p> que es el único hijo de su padre
:optional	input:optional	Selecciona todos los elementos input sin atributo "required"
:out-of-range	input:out-of-range	Selecciona los elementos input con un valor fuera del rango especificado
:read-only	input:read-only	Selecciona los elementos input con atributo "readonly" especificado
:read-write	input:read-write	Selecciona los elementos input con atributo "readonly" NO especificado
:required	input:required	Selecciona los elementos input con atributo "required" especificado
:root	:root	Selecciona el elemento raiz del documento
::selection	::selection	Selecciona la porción de un elemento que esta seleccionado or el usuario
:target	#news:target	Selecciona elemento corriente activo #news (cuando se hace clic en el URL que contiene su ancla)
:valid	input:valid	Selecciona todos los elementos input con un valor valido
:visited	a:visited	Selecciona todos loe elementos link visitados

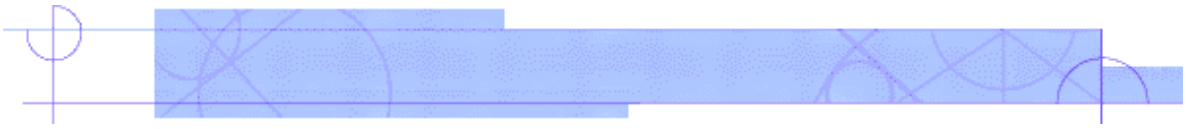
Tabla 4.14

Ejercicios:
Usar directivas css para dar formato a un menú

Archivo: menu01css.html
En la carpeta css

```
<!DOCTYPE html>
<html>
<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></s
cript>
    <style>
        ul {
            list-style-type: none; /*sin la marca de la lista (el punto)*/
```

```
            margin: 0;
            padding: 0;
            overflow: hidden; /* no mostrar barra de desplazamiento*/
            border: 1px solid #e7e7e7;
            background-color: #f3f3f3;
        }

        li {
            float: left;
        }

        li a {
            display: block;
            color: #666;
            text-align: center;
            padding: 14px 16px;
            text-decoration: none;
        }

        li a:hover:not(.active) {
            background-color: #ddd;
        }

        li a.active {
            color: white;
            background-color: #4CAF50;
        }
    </style>
</head>
<body>

<ul id="mi_menu">
  <li><a id="inicio" class="active" href="#">inicio</a></li>
  <li><a id="pagina1" href="#">altas</a></li>
  <li><a id="pagina2" href="#">bajas</a></li>
  <li><a id="pagina3" href="#">mostrar</a></li>
</ul>
<div id="area_trabajo"></div>

</body>
```



```
</html>
```

Tabla 4.15

Otros ejemplos de menú:

1. https://www.w3schools.com/howto/tryit.asp?filename=tryhow_css_vertical_menu
2. https://www.w3schools.com/howto/tryit.asp?filename=tryhow_css_menu_hor_scroll

Ejemplos de formularios con CSS:

1. https://www.w3schools.com/css/tryit.asp?filename=trycss_forms

Referencias css:

2. https://www.w3schools.com/css/css_float.asp

4.5. Procedimiento

1. En la sección de códigos/css/intefaz se describen una serie de ejercicios en los que donde se aplica css.
2. Probar los códigos.

4.6. Prácticas

1. A su ejercicio de curriculum, agregue formatos de manera que se vea profesional, agregue colores, imágenes, tablas.
2. Modificar su menú para que se muestre horizontal en la parte superior y tenga colores de fonto y de texto.
3. Elija algún diseño css de los vistos en esta descripción y aplíquelo a su ventana de altas y buscar.
4. Elija un formato de tabla y aplíquelo a su página de mostrar datos
5. Construya un formulario en html que capture los datos de un alumno: matricula, nombre, sexo, estado de origen y edad, apliquele un estilo en css.
6. Programe una interfaz para alta, modificar y mostrar artículos de un almacen los datos del artículo son: [id, descripcion, existencia, stock, costoCompra, costoVenta], hágalo en html usando css.

PRACTICA 5. W3CSS

5.1. Objetivo

El alumno será capaz de utilizar la biblioteca W3CSS para dar formato a sus códigos HTML.

5.2. Equipo

Computadora personal, internet, editor, manuales.

5.3. Materiales

Internet, navegador, manuales.

5.4. Descripción

W3CSS es un framework CSS responsivo y ligero y es una alternativa para Bootstrap.
Referencia: https://www.w3schools.com/w3css

Inserte este tag en el <head> de su archivo html para usar W3CSS
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

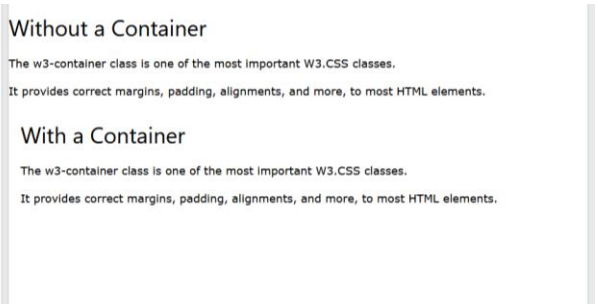
Container

La clase contenedor w3 agrega un margen izquierdo y derecho de 16 píxeles a cualquier elemento HTML. La clase w3-container es la clase perfecta para usar con todos los elementos contenedor HTML como:

, <article>, <section>, <header>, <footer>, <form>

Ejemplo:

```
<!DOCTYPE html>
<html>
<title>W3.CSS</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
<body>

<div>
  <h2>Without a Container</h2>
  <p>The w3-container class is one of the most important W3.CSS classes.</p>
  <p>It provides correct margins, padding, alignments, and more, to most HTML elements.</p>
</div>

<div class="w3-container">
  <h2>With a Container</h2>
  <p>The w3-container class is one of the most important W3.CSS classes.</p>
  <p>It provides correct margins, padding, alignments, and more, to most HTML elements.</p>
</div>

</body>
</html>
```



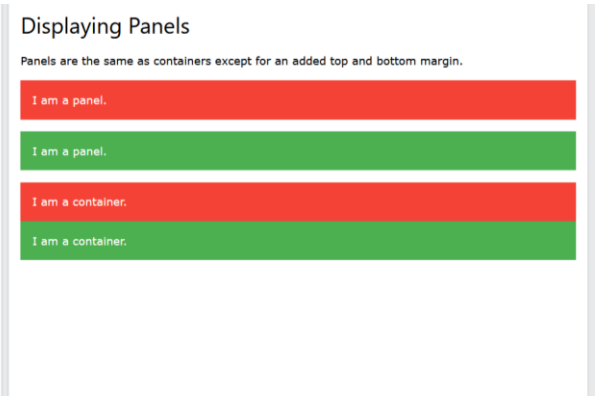
Panel

Un panel es una clase de W3CSS que agrega un margen de 16px arriba, abajo, izquierda y derecha.

Panel
  I am a panel.
I am a panel.

Ejemplo:

```
<!DOCTYPE html>
<html>
<title>W3.CSS</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
<body>

<div class="w3-container">
  <h2>Displaying Panels</h2>
  <p>Panels are the same as containers except for an added top and bottom margin.</p>

  <div class="w3-panel w3-red">
    <p>I am a panel.</p>
  </div>
  <div class="w3-panel w3-green">
    <p>I am a panel.</p>
  </div>

  <div class="w3-container w3-red">
    <p>I am a container.</p>
  </div>
  <div class="w3-container w3-green">
    <p>I am a container.</p>
  </div>
</div>

</body>
</html>
```

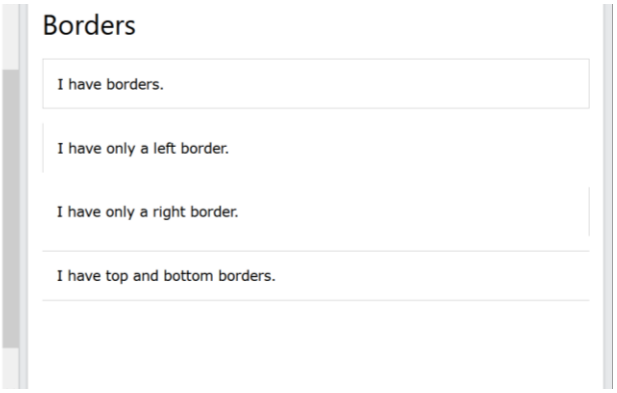


Bordes
Puede poner bordes a un panel, agregando las clases:

w3-border	Adds borders (top, right, bottom, left) to an element
w3-border-top	Adds a top border to an element
w3-border-right	Adds a right border to an element
w3-border-bottom	Adds a bottom border to an element
w3-border-left	Adds a left border to an element
w3-border-0	Removes all borders
w3-border-*color*	Displays the border in a specified color (like red, blue, etc)
w3-hover-border-*color*	Adds a hoverable border color
w3-bottombar	Adds a thick bottom border to an element
w3-leftbar	Adds a thick left border to an element
w3-rightbar	Adds a thick right border to an element
w3-topbar	Adds a thick top border to an element

Ejemplo:

```
<body>

<div class="w3-container">
  <h2>Borders</h2>

  <div class="w3-panel w3-border">
    <p>I have borders.</p>
  </div>

  <div class="w3-panel w3-border-left">
    <p>I have only a left border.</p>
  </div>

  <div class="w3-panel w3-border-right">
    <p>I have only a right border.</p>
  </div>

  <div class="w3-panel w3-border-top w3-border-bottom">
    <p>I have top and bottom borders.</p>
  </div>

</div>
```



Grid

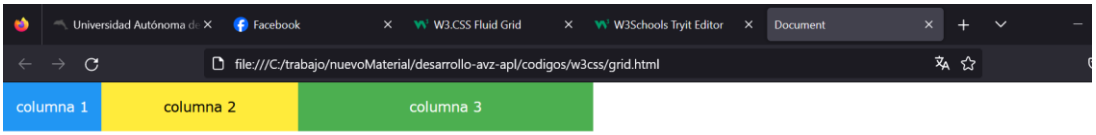
Podemos dividir un contenedor hasta en 12 columnas que serán responsivas, para diferentes tipos de pantalla se manejas columnas tipo: s, m, l small, médium,large). Ejemplo poner un contendor div con 3 columnas que ocupen 1 2 y 3 columnas respectivamente, para una pantalla mediana:

Vea este ejemplo en: códigos/w3css/grid.html

```
<div class="w3-row">
    <div class="w3-col m1 w3-blue">
        <p class="w3-center">columna 1</p>
    </div>
    <div class="w3-col m2 w3-yellow">
```

```
        <p class="w3-center">columna 2</p>
      </div>
      <div class="w3-col m3 w3-green">
        <p class="w3-center">columna 3</p>
      </div>
    </div>
```

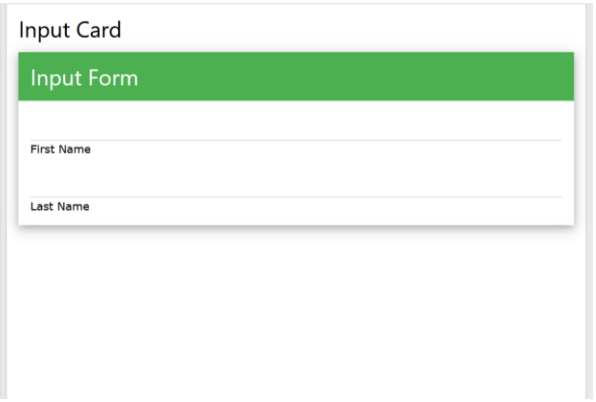
Resultado:



Formularios

Ejemplo:

```
<!DOCTYPE html>
<html>
<title>W3.CSS</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
<body>

<div class="w3-container">
  <h2>Input Card</h2>

  <div class="w3-card-4">
    <div class="w3-container w3-green">
      <h2>Input Form</h2>
    </div>

    <form class="w3-container">
      <p>
      <input class="w3-input" type="text">
      <label>First Name</label></p>
      <p>
      <input class="w3-input" type="text">
      <label>Last Name</label></p>
    </form>
  </div>
</div>

</body>
</html>
```



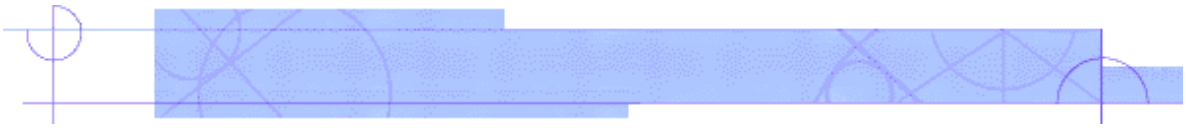
Ejemplo de formulario

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <title>mi pagina</title>
   <link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

</head>
<body>
   <div class="div_main">
      <form class="w3-panel w3-pale-green" id="frm_alta">
         <fieldset class="w3-panel w3-pale-blue w3-padding-16">
            <legend>id</legend>
```

```
            <label for="txt_id">id</label>
            <input type="text" id="txt_id" name="id">
        </fieldset>
        <fieldset class="w3-panel w3-pale-blue w3-padding-16">
            <legend>sexo</legend>
            <label for="rb_h">hombre</label>
            <input type="radio" id="rb_h" name="sexo" value="h">
            <label for="rb_m">mujer</label>
            <input type="radio" id="rb_m" name="sexo" value="m" checked>
        </fieldset>
        <fieldset class="w3-panel w3-pale-blue w3-padding-16">
            <legend>estado</legend>
            <label for="cmb_estado"></label>
            <select id="cmb_estado">
                <option value="PU">Puebla</option>
                <option value="MX">Mexico city</option>
                <option value="CC">Ciudad del Carmen</option>
                <option value="CA">Campeche</option>
                <option value="VH">Villa Hermosa</option>
            </select>
        </fieldset>
        <fieldset class="w3-panel w3-pale-blue w3-padding-16">
            <label for="_semana"></label>
            <input type="week" id="_semana">
        </fieldset>
    </form>
    <input type="button" id="btn_aceptar" value="enviar">
    <input type="button" id="btn_cancelar" value="cancelar">
  </div>
  <script src="./js/main.js"></script>
</body>
</html>
```

Archivo: min.js

```
window.onload=function(){

  function eventoClic(){
    alert("evento clic 2")
  }
  function traeFuncion(){
    return function(){
      const sexo=document.forms.frm_alta.elements.sexo.value;
      const estado=document.forms["frm_alta"].elements.cmb_estado.value;
      alert("sexo:"+sexo+" estado: "+estado);
    };
  }
  document.querySelector("#btn_aceptar").addEventListener('click',traeFuncion());
  document.querySelector("#btn_cancelar").addEventListener('click',()=>{
    alert("evento clic desde cancelar")
  });
```



```
}
```

Ejemplo de formulario

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
   <title>Document</title>
</head>
<body>
   <div class="w3-container w3-row">
      <div class="w3-col l3">
         <p></p>
      </div>
      <div id="div_form" class="w3-card w3-col l5 w3-padding-16">
         <h2 class="w3-padding">Datos de la pregunta</h2>
         <div class="w3-container" id="mensaje">
         </div>
         <form id="miformulario" class="w3-contaier w3-padding">
            <p id="p-profesor">
               <label for="id">Profesor</label>
               <select class="w3-input w3-border-top w3-hover-sepia" name="select-
profesor" id="sp">
                  <option value="1">juan 1</option>
                  <option value="2" selected>luis 2</option>
                  <option value="3">paco 3</option>
               </select>
            </p>
            <p id="p-materia">
               <label for="id">materia</label>
               <select class="w3-input w3-border-top w3-hover-sepia" name="select-materia"
id="sm">
                  <option value="1">materia 1</option>
                  <option value="2" selected>materia 2</option>
                  <option value="3">materia 3</option>
               </select>
            </p>
            <p id="p-tema">
               <label for="tema">tema</label>
               <select class="w3-input w3-border-top w3-hover-sepia" name="select-tema"
id="st">
                  <option value="1">tema 1</option>
                  <option value="2" selected>tema 2</option>
                  <option value="3">tema 3</option>
               </select>
            </p>
            <p>
               <label for="id">Su id</label>
```

```
            <input readonly class="w3-input w3-border-top w3-hover-sepia" type="text"
id="id" name="id"
                placeholder="no capture el id" />
            </p>
            <p>
              <label for="pregunta">Su pregunta</label>
              <textarea name='pregunta' type='text' id='pregunta'
                value='' rows='auto' cols='40'>su pregunta?
              </textarea>
            </p>

            <p>
              <label for="d1">distractor 1</label>
              <input class="w3-input w3-border-top w3-hover-sepia" type="text" id="d1"
name="d1"
                placeholder="d1" />
            </p>
            <p>
              <label for="d2">distractor 2</label>
              <input class="w3-input w3-border-top w3-hover-sepia" type="text" id="d2"
name="d2"
                placeholder="d2" />
            </p>
            <p>
              <label for="d3">distractor 3</label>
              <input class="w3-input w3-border-top w3-hover-sepia" type="text" id="d3"
name="d3"
                placeholder="d3" />
            </p>
            <p>
              <label for="d4">distractor 4</label>
              <input class="w3-input w3-border-top w3-hover-sepia" type="text" id="d4"
name="d4"
                placeholder="d4" />
            </p>
            <p>
              <label for="d4">Opcion correcta (entre 1 y 4)</label>
              <input class="w3-input w3-border-top w3-hover-sepia" type="number"
id="correcta" name="correcta"
                placeholder="Opcion correcta" min="1" max="4" required/>
            </p>
            <p>
              <input type="button" id="btn-actualizar" class="w3-input w3-border-top w3-
hover-sepia"
              value="Actualizar pregunta" onclick="actualizaForm(event)"
              style="visibility:hidden">
            </p>
            <p>
              <input type="button" id="btn-insertar" class="w3-input w3-border-top w3-
hover-sepia"
```

```
            value="Insertar nueva pregunta" onclick="insertaForm(event)"
            style="visibility:hidden">
          </p>
          <p>
            <input type="button" id="btn-cancelar" class="w3-input w3-border-top w3-
hover-sepia"
            value="Cancelar" onclick="cancelar(event)">
          </p>
        </form>
      </div>

    </div>
</body>
</html>
```

5.5. Procedimiento

1. En la sección de ejercicios se describen una serie de problemas en los que es necesario aplicar estilos para mejorar la visibilidad de la página.
2. Elaborar el análisis del problema en base al tipo de usuario y proponer un diseño.
3. Codificar usando la biblioteca Bootstrap.
4. Probar en tres diferentes navegadores.

5.6. Prácticas

1. Diseñe un menú para las necesidades de una tienda pequeña usando w3css.
2. Diseñe un menú para la aplicación de su proyecto.
3. Diseñe un formulario de captura de datos para su proyecto, usando elementos w3css.
4. Diseñe una página con una tabla que muestre datos usando elementos w3css.
5. Diseñe una página de búsqueda de datos usando elementos de w3css.
6. A las ventanas de su proyecto aplique estilos CSS + w3css

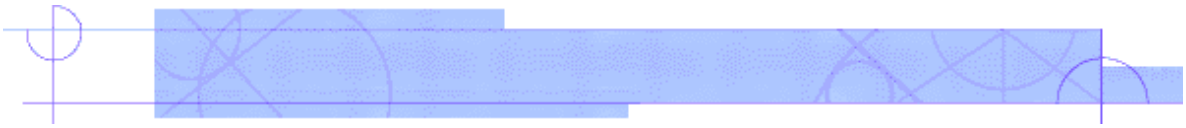
PRACTICA 6. JS

6.1. Objetivo

El alumno será capaz de elaborar aplicaciones computacionales que den solución a problemas que requieran páginas web dinámicas usando lenguaje JS.

6.2. Equipo

Computadora personal, dispositivo de almacenamiento secundario (disquete, memoria USB, etc.), lenguaje de programación visual basic, c.



6.3. Materiales

Hojas de papel bond o cuaderno, lápiz o boligrafo.

6.4. Descripción

Introducción:

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico, es soportado por virtualmente todos los navegadores modernos y nos permite agregar funcionalidad del lado del cliente a las páginas WEB.

Aquí veremos una introducción a el uso de este lenguaje.

1 Java Script y HTML.

Con Js puede cambiar el contenido del HTML. En un archivo con extensión html introduce el código de la tabla 6.1 y ábrelo con el navegador Google Chrome o Mozilla, sobre el archivo da clic derecho y luego abrir con, y elige en navegador.

Cambiando contenido en HTML, archivo: holamundo.html

```
<!DOCTYPE html>
<html>
    <body>
        <h1>Que puedo hacer con JS?</h1>
        <p id="demo">JavaScript puede cambiar el contenido de HTML.</p>
        <button type="button" onclick="document.getElementById('demo').innerHTML = 'Hola Mundo!'">Haz clic aqui!</button>
    </body>
</html>
```

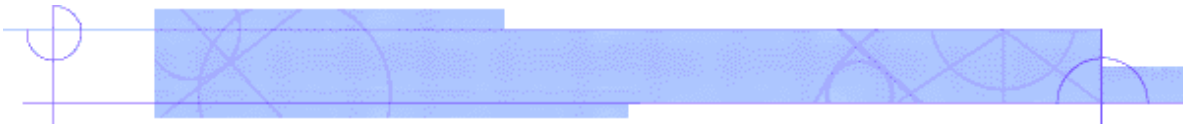
Tabla 6.1.

Aquí el trabajo lo hace la función onclick que esta como atributo del botón:

***onclick*="document.getElementById('demo').innerHTML = 'Hola Mundo!'"**

Onclick ejecuta una sentencia cuando se hace clic sobre el botón, en este caso la sentencia es:

document.getElementById('demo').innerHTML = 'Hola Mundo!'



En la sentencia; ***document*** se refiere a todo el documento, es decir la página web, ***getElementById***, indica, que queremos hacer referencia a un elemento por su nombre, como se le pasa como parámetro ***('demo'),*** document.getElementById('demo') quiere decir que dentro del documento debemos seleccionar al elemento cuyo id sea ***demo*** y en este caso es el elemento:

*** JavaScript puede cambiar el contenido de HTML. ***

Que estamos seleccionando. El resto de la sentencia:

.innerHTML = 'Hola Mundo!'"

¡Indica que al elemento seleccionado dentro de su atributo innerHTML, debemos asignar la cadena de texto 'Hola Mundo!', el atributo ***innerHTML*** hace referencia al contenido que visualiza el control, sería como el atributo text de un control en visual Basic.

La función getElementById('demo') que obtiene la referencia a un elemento de html mediante su nombre puede ser sustituida por la sentencia:

querySelector('#demo'), de manera que la sentencia completa quedaría así:

onclick="document. querySelector('#demo')***.innerHTML = 'Hola Mundo!'"***

También puede cambiar los atributos del HTML

Cambiado atributos, archivo: atributo.html

```
<!DOCTYPE html>
<html>
   <body>
      <h1>Que puedo hacer con JS?</h1>
      <p id="demo">JavaScript puede cambiar el estilo de un elemento HTML.</p>
      <button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Haga
clic!</button>
   </body>
</html>
```

Tabla 5.2

Esta sentencia indica que, al hacer clic en el botón, el estilo del texto del párrafo deberá cambiarse a una fuente de tamaño 35 pixels.

2 Donde codificar el JS.

En los ejemplos anteriores, se vio como codificar JS incrustandolo en los tag de HTML, una mejor forma de hacerlo es dentro de un área epecial dentro del archivo HTML (ver tabla 5.3) en la tabla se muestra el uso de una fucnión de JS asociada a un Button, donde la función está definida en el área <script>...</script>

Ejemplo JS

```
<!DOCTYPE html>
<html>
   <head>
      <script>
         function myFunction() {
            document.getElementById("demo").innerHTML = "Parrafo cambiado.";
         }
      </script>
   </head>
   <body>
      <h1>JavaScript in Head</h1>
      <p id="demo">Mi parrafo.</p>
      <button type="button" onclick="myFunction()">clic aqui</button>
   </body>
</html>
```

Tabla 5.3 Archivo: js03.html

Pruebe el ejemplo de la tabla 5.3.

Otra forma de hacerlo (seguramente la mas 'limpia' desde el punto de vista de un programador organizado) es poner el código JS en un archivo aparte, como se muestra en la tabla 5.4.

Archivo: js04.html

```
<!DOCTYPE html>
<html>
   <head>
      <script src="myscript.js"></script>
   </head>
   <body>
      <h1>JavaScript in Head</h1>
      <p id="demo">Mi parrafo.</p>
      <button type="button" onclick="myFunction()">clic aqui</button>
   </body>
</html>
```

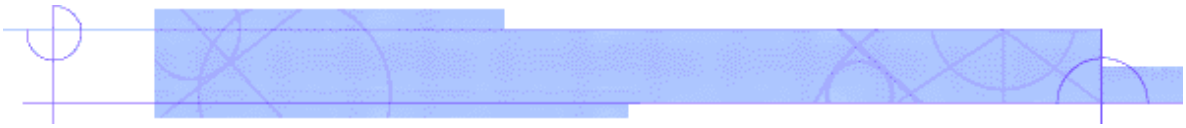
Archivo: myscript.js

```
function myFunction() {
   document.getElementById("demo").innerHTML = "Parrafo cambiado.";
}
```

Tabla 5.4

El ejemplo de la tabla 5.4 consta de 2 archivos, lo puede encontrar en la carpeta: \ejemplos\js-puro\basicos, que puede descargar desde la carpeta de ejemplos.

Pruebe los ejemplos de la tabla 5.3 y 5.4 Comente sus dudas con el profesor.



3 Entrada Salida

En JS no existe E/S como tal, no puede como en c usar printf, en su lugar el JS se comunica con su página HTML

Ejemplo mostrar una ventana Alert:

Mostrar una ventana emergente

```
<!DOCTYPE html>
<html>
<body>
   <h1>Ejemplo de alert</h1>
   <p>mi parrafo.</p>
   <script>
      window.alert(5 + 6);
   </script>
</body>
</html>
```

Tabla 5.5 Alert en JS

También puede escribir al documento HTML

Escribir en el documento HTML

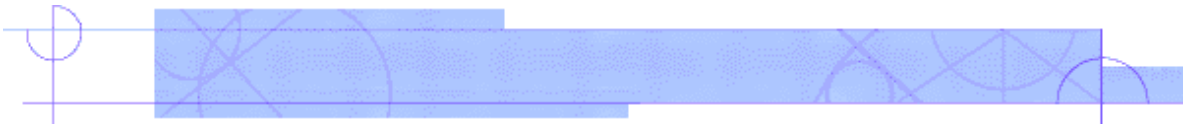
```
<!DOCTYPE html>
<html>
   <body>
      <h1>mi Web Page</h1>
      <p>mi parrafo.</p>
      <script>
         document.write(5 + 6);
      </script>
   </body>
</html>
```

Tabla 5.6 Escribir al documento.

Si usa document.write despues de que el HTML es cargado totalmente, elimina el contenido anterior del HTML, vea este ejemplo

```
<!DOCTYPE html>
<html>
   <body>
      <h1>mi Web Page</h1>
      <p>mi parrafo.</p>
      <button onclick="document.write(5 + 6)">click aqui</button>
   </body>
</html>
```

Tabla 5.7.



Puede también escribir dentro de un elemento HTML

Escribiendo el el elemento HTML con id = "demo"

```
<!DOCTYPE html>
<html>
   <body>
      <h1>My First Web Page</h1>
      <p>My First Paragraph</p>
      <p id="demo"></p>
      <script>
         document.getElementById("demo").innerHTML = 5 + 6;
      </script>
   </body>
</html>
```

Tabla 5.8

Finalmente, y este es un método usado por desarrolladores, se puede escribir a la consola del navegador (puede visualizar la consola de su navegador haciendo clic en la tecla f12, y en la ventana que aparece seleccione la pestaña consola), esto es muy útil para depurar su código. Normalmente el usuario no visualiza la consola, así que use este método para dar seguimiento a su código mientras está desarrollando.

Escribiendo a la consola

```
<!DOCTYPE html>
<html>
   <body>
      <h1>My First Web Page</h1>
      <p>My first paragraph.</p>
      <script>
         console.log(5 + 6);
      </script>
   </body>
</html>
```

Tabla 5.9

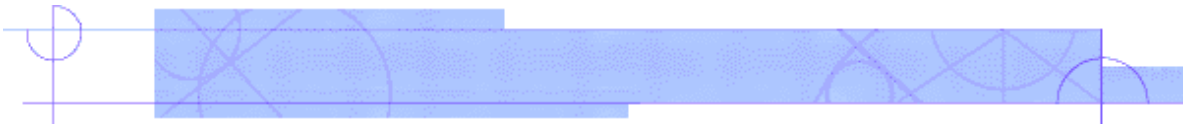
3 Entrada JS

La entrada en JS tampoco existe, JS tomara sus datos de los tag de HTML

4 Variables, tipos y expresiones.

Variables

En JS las variables se declaran así:



var mivariable;

var miotravariable=23;

Si son variables globales

O así si son locales:

let mivariable;

let miotravariable=23;

o como constante:

const miotravariable=23;

No se declarar el tipo, el tipo de una variable se asocias al asignarle un dato, dependiendo del, dato una variable puede ser: string, number, boolean, array y object.

Las cadenas en JS se representan entre comillas o entre apostrofes.

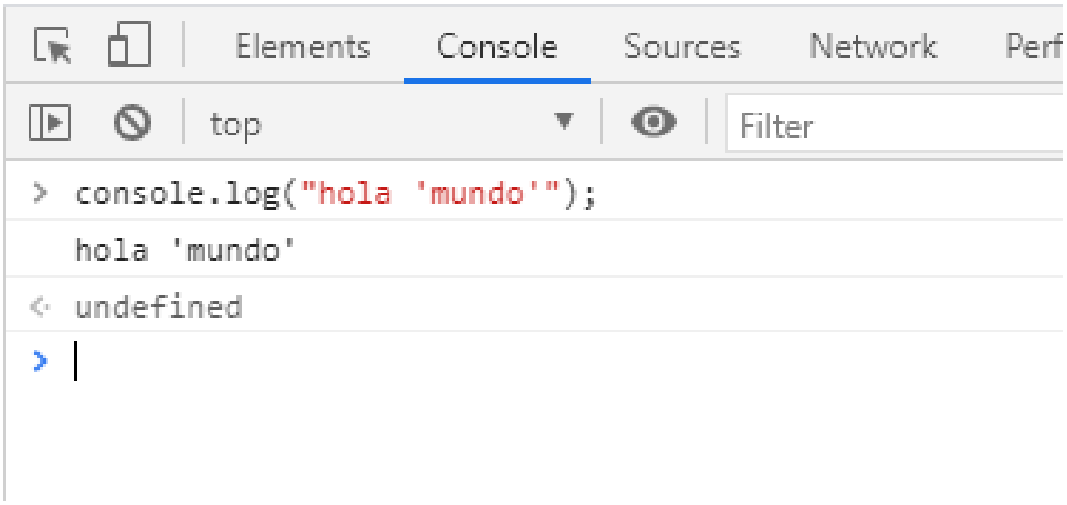
"esta es una cadena"

'esta tambien es una cadena'

var nombre='Andrea Flores';

Las cadenas se pueden anidar así:

Pruebe este código escribiéndolo en la consola de su navegador así:



Observe que los apostrofes dentro de la cadena se toman como parte de la cadena.

Tipos

Como dijimos el tipo de la variable se define cuando se le carga un dato ejemplo:

```
var length = 16;                       // Number
var lastName = "Johnson";              // String
var cars = ["Saab", "Volvo", "BMW"];   // Array
var x = {firstName:"John", lastName:"Doe"};    // JSON
```

pruebe este código en la consola del navegador:

```
let persona={nombre:'juan', apellido:'perez'};
console.log(persona.nombre);
```

Concatenación de cadenas

Las cadenas se concatenan con +

var nombre= "Alex" +" "+"Flores";

Operadores

Operador	Descripción
+	Adición
-	Substracción
*	Multiplicación
/	División
%	Modulo
++	Incremento
--	Decremento

igualdad	Comprueba si dos valores son iguales entre sí, y devuelve un valor de `true`/`false`(booleano).	==
identidad	Comprueba si dos valores son identicos entre sí, y devuelve un valor de `true`/`false`(booleano).	===
Negación, distinto (no igual)	En ocasiones utilizado con el operador de identidad, la negación es en JS el equivalente al operador lógico NOT — cambia `true` por `false` y viceversa.	!, !==

Tabla 5.10

Ejemplos:

```
1 == 1 // true
"1" == 1 // true
1 == '1' // true
0 == false // true
1 != 2 // true
1 != "1" // false
1 != '1' // false
1 != true // false
0 != false // false
3 === 3 // true
3 === '3' // false
3 !== '3' // true
4 !== 3 // true
```

5 Funciones.

Una función en JS se declara de forma similar que, en C, ejemplo:

```
//definición de función

function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit-32);
}

//llamado a la función
document.getElementById("demo").innerHTML = toCelsius(9);
```

Una función puede asignarse a una variable:

Función a una variable

```
<!DOCTYPE html>
<html>
   <body>
      <p id="demo"></p>
      <script>
         var x= function celsius(fahrenheit) {
            return (5/9) * (fahrenheit-32);
         }
         document.getElementById("demo").innerHTML ="la temperatura es " + x(77) + " Celsius";
   </script>
</body>
</html>
```

Tabla 5.11

En este caso ya que la función esta referenciada por la variable x podemos omitir el nombre de la función a lo que se llama función anónima.

Función anónima

```
<!DOCTYPE html>
<html>
   <body>
      <p id="demo"></p>
      <script>
         var x= function(fahrenheit) {
            return (5/9) * (fahrenheit-32);
         }
         document.getElementById("demo").innerHTML ="la temperatura es " + x(77) + " Celsius";
   </script>
</body>
</html>
```

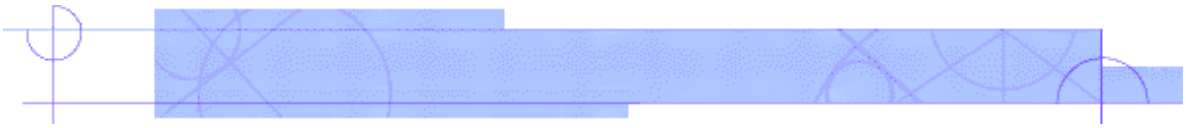


Tabla 5.12

5.1 Funciones anónimas

Una función anónima es una función que no tiene nombre se declaran así:

```
function () {
  console.log('Esta función no tiene un nombre')
}
```

No parece tener sentido ya que al no tener nombre no puedes llamarla, pero se usan cuando se quiere una función autoinvocada o enviar una función como parámetro.

5.2 Funciones autoinvocadas.

Una función anónima autoinvocada se ejecuta de forma automática, su sintaxis es:

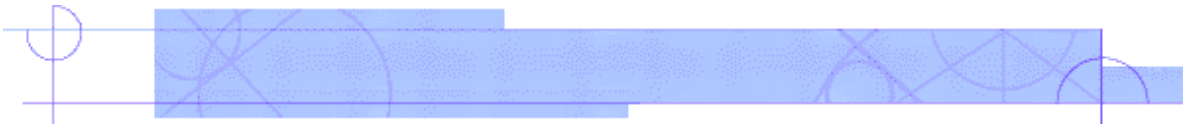
```
(function(){

   // Código

})();
```

Su contenido se ejecuta automáticamente, ejemplo pruebe el ejemplo de la siguiente tabla:

Ejemplo función anónima autoinvocada

```
<!DOCTYPE html>
<html>
   <script>
      (function() {
         var primera = 'Hola',
         segunda = 'Don Pepito';

         function saludo() {
            return [primera, segunda].join(';');
      }
      document.write(saludo());
   })();
</script>
</html>
```

Tabla 5.13



Para el resultado del ejemplo bastaba hacer:

document.write(primera + segunda);

en lugar de llamar a una función sin embargo quisimos hacerlo así para mostrar que una función puede contener a otra función. La función join() por otra parte une los elementos de una arreglo en una cadena separados por el separador enviado como parámetro a la función join.

6 Eventos.

En JS se pueden manejar varios eventos ejemplo:

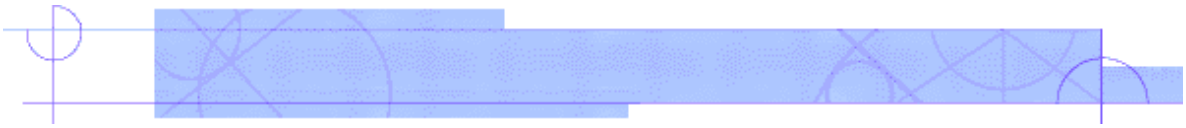
```
<!DOCTYPE html>
<html>
   <body>
      <p id="demo" onmouseover="saluda()">hola</p>
         <button onclick="displayDate()">The time is?</button>
   </body>
   <script>
      function displayDate() {
         document.getElementById("demo").innerHTML = Date();
         document.getElementById("demo").style.fontSize='12px';
      }
      function saluda() {
         document.getElementById("demo").style.fontSize='35px';
      }

</script>
</html>
```

Tabla 6.1

En la tabla 6.1 el evento onmouseover se dispara cuando pasa el cursor sobre el párrafo, el otro es un evento clic.

Otros eventos son:

Evento	Descripción
onchange	Cuando un elemento HTML cambió
onclick	Cuando el usuario hizo clic sobre un elemnto HTML



onmouseover	El usuario mueve el mouse sobre un elemento HTML
onmouseout	El usuario mueve el mouse fuera de un elemento HTML
onkeydown	El usuario oprime una tecla
onload	El navegador termina de cargar la página

Tabla 6.2

6.1 Donde asignar eventos

Se puede asignar un evento en forma estática como en la tabla 6.1 en la línea:

<button onclick="displayDate()">The time is?</button>

Tambien se pueden asiganar eventos en forma deinámica dentro del código html:

Archivo:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">

        <style>
            table{
                width:300px;
            }
            th{
                background-color:#008080;
                color:#ffffff;
            }
            tr:nth-child(even){background-color:#f2f2f2;}
            tr:nth-child(odd){background-color:#C0C0C0;}
            th, td{
                text-align:center;
                padding:10px;
            }
            td:hover{
                background-color:#424242;
                color:#ffffff;
            }
            #miBoton {
                background-color: #008CBA;
                border: none;
```

```
                color: white;
                padding: 10px 25px;
                text-align: center;
                text-decoration: none;
                display: inline-block;
                font-size: 16px;
            }
        </style>
    </head>
    <body>
        <div id="area_trabajo">

        </div>
        <input type="button" id="miBoton" name="llenar" value="llenar tabla">

    </body>
    <script>

        window.onload=function(){
            document.getElementById("miBoton").addEventListener("click", function(){
                document.getElementById("area_trabajo").innerHTML="Hola mundo";
            });
        }
    </script>
</html>
```

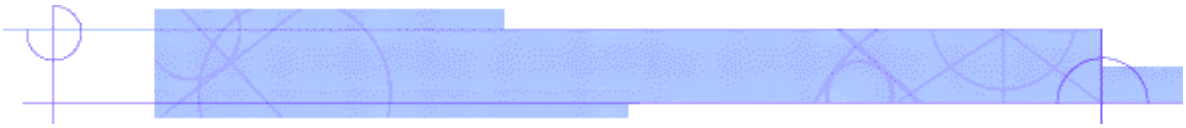
Tabla 6.3

Explicación:
Aquí obrevamos dos cosas:

```
window.onload=function(){


}
```

Una que estamos usando la función Windows.onload, es recomendable usarla para encapsular su código JS, esta función ejecuta su código después de que la página se cargo totalmente, esto evita errores. De esta forma estamos seguros que nuestro código se ejecuta cuando todos los elementos de la página están ya disponibles.

La otra parte del código:



document.getElementById("miBoton").addEventListener(evento , manejador);

Agrega un evento al ejlemento del documento cuyo id es igual a “miboton”, los parámetros evento es una cadena que indica el evento que agregamos y el parámetro manejador es una función que maneja el evento.

Los parametyros que se ponen en la sentencia son:

```
"click"
```

Para agregar el envento clic y el otro parámetro es la función anónima:

```
function(){
      document.getElementById("area_trabajo").innerHTML="Hola mundo";
}
```

Que se ejecutará cuando se dispare el evento.

Ejemplo de evento clic en menú

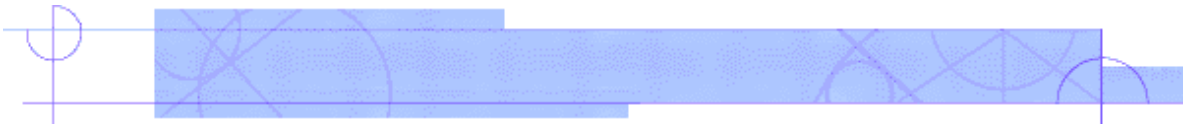
Veamos ahora algoi un poco mas complejo, mostraremos un menú y asignaremos clic a las opciones del menú de manera que al hacer clci en alguna opción, se muestre en la consola, un mensaje que indique a que opción se hizo clic:
Este ejemplo lo puede ver en la carpeta de ejercicios: ***ejemplos\js-puro\eventos\event-mnu-0.html***

Y en la tabla siguiente:

Archivo: event-mnu-0.html

```
<!DOCTYPE html>
<html>
        <style>
                ul {
                    list-style-type: none; /*sin la marca de la lista (el punto)*/
                    margin: 0;
                    padding: 0;
                    overflow: hidden; /* no mostrar barra de desplazamiento*/
                    border: 1px solid #e7e7e7;
                    background-color: #f3f3f3;
                }
                li {
```

```
                    float: left; /* los muestra en la misma linea */
                }
                li a {
                    display: block; /*despliega la opcion en modo de una caja ocupando
todo su espacio*/
                    color: #666;
                    text-align: center;
                    padding: 14px 16px;
                    text-decoration: none; /*sin subraya*/
                }
                li a:hover:not(.active) { /*cuando el cursos pasa sobre las opciones no
activas*/
                    background-color: #ddd;
                }
                li a.active { /*la opcion activa*/
                    color: white;
                    background-color: #4CAF50;
                }
        </style>
<head>

</head>
<body>

    <ul>
        <li><a id="op_1" class="op_menu active" href="#">Alta alumno</a></li>
        <li><a id="op_2" class="op_menu" href="#">Buscar Alumno</a></li>
        <li><a id="op_3" class="op_menu" href="#">Mostrar alumnos</a></li>
        <li><a id="op_4" class="op_menu" href="#">Acerca de</a></li>
    </ul>

</body>
<script>
    function click_menu(e){
        let quien=e.target; //a quien se le hizo clic
        console.log("clic en "+quien.getAttribute("id"));
    }
    window.onload=function(){
        let ops=document.getElementsByClassName("op_menu");
        //agregamos el listener a las opciones del menu
```



```
        for(i=0;i<ops.length;i++){
            ops[i].addEventListener('click',click_menu);
        }
    };
</script>
</html>
```

Ejecutelo dando clic derecho en el archivo y abrir con el navegador de su preferencia. Al hacer clci en alguna opción en la consola se verá un mensaje.

Tabla 6.4

Explicación:

La sentencia

```
let ops=document.getElementsByClassName("op_menu");
```

Obtiene una lista de elementos, en este caso la lista de elementos cuto nombre de clase es “op_menu”.

La sentencia:

```
        for(i=0;i<ops.length;i++){
            ops[i].addEventListener('click',click_menu);
        }
```

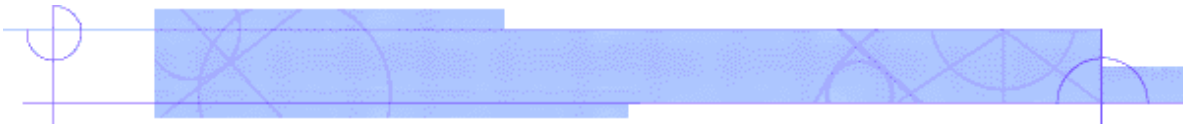
Recorre la lista ops y a cada elemento le asocia el evento clic e indica que el manejador de evento es la función click_menu.

```
function click_menu(e){
        let quien=e.target; //a quien se le hizo clic
        console.log("clic en "+quien.getAttribute("id"));
    }
```

La función clic menú recibe como arguimento (de manera automática, usted no tiene que enviarlo) on objeto que tiene la información del evento, en este caso se recibe en la variable local e.

e.target;

nos devuelve el nombre del control que generó el evento, es decir a quien le hicimos clic.
Finalmente escribimos en la consola el atributo id del control que nos generó el evento.



7 Objetos (opcional).

Los objetos en JS son como variables, pero pueden contener mas de un valor.

var car = {type:"Fiat", model:"500", color:"white"};

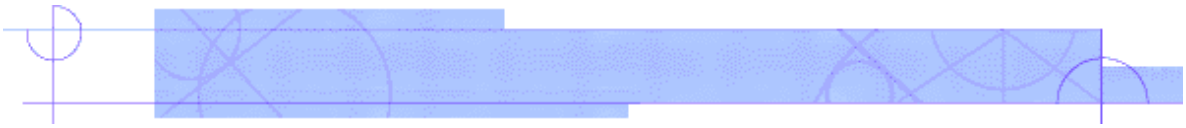
Para acceder a las propiedades de u objeto se usa:

```
<!DOCTYPE html>
<html>
   <body>
      <p id="demo"></p>
      <p id="otro"></p>
      <script>
         var car = {tipo:"Fiat", modelo:"500", color:"white"};
         document.getElementById("demo").innerHTML ="el auto es un " + car.tipo;
         document.getElementById("otro").innerHTML =" otra forma de acceder a las
propiedades es " + car["tipo"];
      </script>
</body>
</html>
```

Tabla 7.1

Los objetos también pueden tener operaciones:

```
<!DOCTYPE html>
<html>
   <body>
      <p id="demo"></p>
      <script>
         var car = {
            tipo:"Fiat",
            modelo:"500",
            color:"white",
            muestra:function(){
               return [this.tipo, this.modelo, this.color].join(',');
            }
         };
```



```
        document.getElementById("demo").innerHTML ="el auto es un " + car.muestra();
    </script>
  </body>
</html>
```

Tabla 7.2

8 prototipos de objetos (opcional).

Como dijimos los objetos son variables, los valores en los objetos se escriben como nombre:valor, los objetos en JS son contenedores para los pares nombre:valor. Los pares nombre:valor en los objetos son llamados propiedades, Los métodos son acciones que pueden ser realizadas en objetos, puede acceder a un método de un objeto con la sintaxis:

objectName.methodName()

8.1 Creando Objetos

En JS puede:

- Definir y crear un objeto simple usando un objeto literal.
- Definir y crear un objeto simple con new.
- Definir un constructor de objetos (object constructor)y crear objetos del tipo construido.

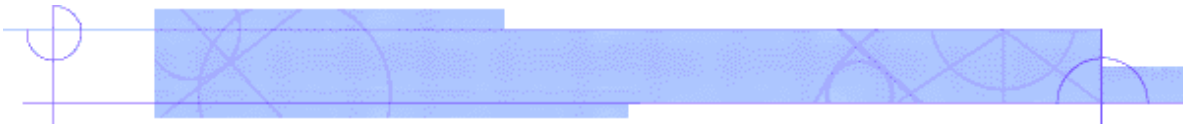
Forma 1: Objeto literal:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Espacios y CR no importan en la definición:

```
var person = {
   firstName:"John",
   lastName:"Doe",
   age:50,
   eyeColor:"blue"
};
```

Forma 2: Otra forma de tener el mismo resultado (el objeto de arriba), es:



```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Esta forma también crea un objeto persona con 4 propiedades, por rapidez de ejecución y legibilidad es recomendable la primera forma.

Forma 3: Constructor de objetos, también llamado prototipo de objetos:

```
function person(first, last, age, eye) {
   this.firstName = first;
   this.lastName = last;
   this.age = age;
   this.eyeColor = eye;
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
```

JS tiene un conjunto de objetos preconstruidos:

```
var x1 = new Object();    // A new Object object
var x2 = new String();    // A new String object
var x3 = new Number();    // A new Number object
var x4 = new Boolean();   // A new Boolean object
var x5 = new Array();     // A new Array object
var x6 = new RegExp();    // A new RegExp object
var x7 = new Function();  // A new Function object
var x8 = new Date();      // A new Date object
```

Otro objeto es Math(), sin embargo este es un objeto global y de este no puede usar new.

8.2 Agregar propiedades y métodos a objetos.

Se puede agregar propiedades al objeto o al prototipo, veamos primero como agregar una propiedad a un objeto:

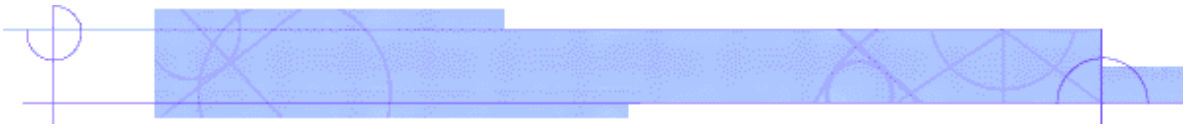
```
<!DOCTYPE html>
<html>
   <body>

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

      <script>
         function Person(first, last, age, eye) {
            this.firstName = first;
            this.lastName = last;
            this.age = age;
            this.eyeColor = eye;
         }

         var myFather = new Person("John", "Doe", 50, "blue");
         var myMother = new Person("Sally", "Rally", 48, "green");

         myFather.nationality = "English";

         myFather.name = function() {
            return this.firstName + " " + this.lastName;
         };

         document.getElementById("demo").innerHTML ="My father is " +
myFather.nationality;
      </script>

   </body>
</html>
```

Tabla 7.3

En este ejemplo se agregó la propiedad nationality y el método name al objeto myFather y no al objeto myMother.

8.3 Agregando propiedades y métodos al prototipo

```
<!DOCTYPE html>
<html>
   <body>
```



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

    <script>
      function Person(first, last, age, eye) {
        this.firstName = first;
        this.lastName = last;
        this.age = age;
        this.eyeColor = eye;

        this.nationality = "English";
        this.name = function() {
          return this.firstName + " " + this.lastName
        };
      }

      var myFather = new Person("John", "Doe", 50, "blue");
      document.getElementById("demo").innerHTML ="My father is " + myFather.name();
    </script>

  </body>
</html>
```

Tabla 7.4

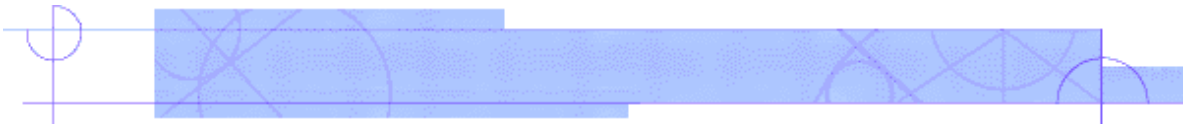
En color azul se muestra la agregación de una propiedad y un método al prototipo.

9 Formato JSON

JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

En la liga: http://www.json.org/json-es.html vea un diagrama para escritura de objetos JSON

En la liga:http://jsonlint.com/ puede verificar la sintaxis de sus objetos JSON.



Ejemplo de una cadena json:

let personas=[{"id" : 01,"nombre": "ana"},{"id": 02,"nombre": "bety"},{"id : 03,"nombre" : "ceci"}];

Observe que la variable se esta declarando con la sentencia let y no con var, let se utiliza para declarar variables locales y var se usa para declarar variables globales.

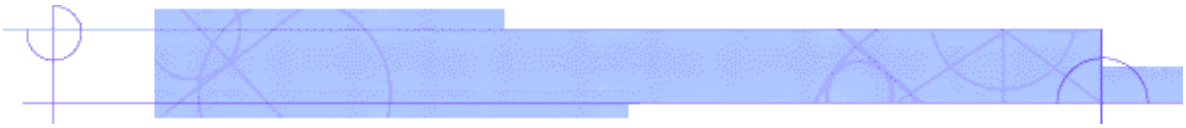
Para recorrer un elemento json podemos usar:

```
<script>
        let personas=[
                {
                        "id" : 01,
                        "nombre": "ana"
                },
                {
                        "id": 02,
                        "nombre": "bety"
                },
                {
                        "id" : 03,
                        "nombre" : "ceci"
                }
        ];
        personas.forEach(function recorre(item, index){
        console.log("index=" + index + " dato: " + item.nombre);
        });
</script>
```

Muestra todos los elementos del json en la consola.

Explicación:
Personas es el objeto josn, y los objetos json tienes asociada cienta funcionalidad en JS.
personas.forEach indica que queremos recorrer cada elemento del json, al forEach le podemos dar como parámetro una función que se encargará de hacer el recorrido:

persona.forEach(function recorre(parametros){//codigo });



los parámetros de la función son ítem que contiene el elemento corriente an cada iteración del for e index que contiene el índice o posición del elemento dentro del json, console.log ya sabemos que escribe a cosnola. Pruebe el ejercicio.

Dentro de la pestaña de ejercicios en la carpeta: ejemplos\js\js02, pruebe los ejercicios:
recorrer_js.html
recorrer_js2.html
recorrer_js3.html

enseguida dos ejemplos mas de json:

10 Recorre JSON

json-avanzado0.html

```
<!DOCTYPE html>
<html>
<head>

</head>
<body>

</body>
<script>

	let personas=[
		{
			"nombre" : "anita pacheco",
			"edad" : 22
		},
		{
			"nombre" : "bety",
			"edad" : 33
		},
		{
			"nombre" : "ceci",
			"edad" : 44
		}
	];
	for(var {"nombre": n, "edad": e} of personas){
		console.log(n, e);
	}
</script>
</html>
```

json-avanzado.html

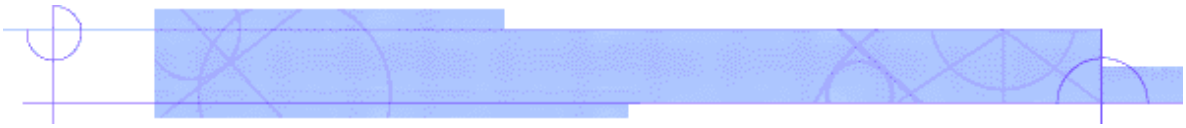
```
<!DOCTYPE html>
<html>
<head>

</head>
<body>

</body>
<script>

	let personas=[
		{
			nombre : "anita pacheco",
			familia : {
				padre : "anastacio",
				madre : "nastasia"
			},
			edad : 22
		},
		{
			nombre : "bety",
			familia : {
				padre : "beto",
				madre : "beta"
			},
			edad : 33
		},
		{
			nombre : "ceci",
			familia : {
				padre : "cesilio",
				madre : "cecilia"
			},
			edad : 44
		}
	];
	for(var {nombre: n, familia:{padre: p}} of personas){
	console.log(n, p);
	}
</script>
</html>
```

Ejemplo que llena una tabla html con datos de un arreglo JSON:

Archivo: ejemplos\js-puro\tablas

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <style>
            table{
                width:300px;
            }
            th{
                background-color:#008080;
                color:#ffffff;
            }
            tr:nth-child(even){background-color:#f2f2f2;}
            tr:nth-child(odd){background-color:#C0C0C0;}
            th, td{
                text-align:center;
                padding:10px;
            }
            td:hover{
                background-color:#424242;
                color:#ffffff;
            }
            #btnLlenar {
                background-color: #008CBA;
                border: none;
                color: white;
                padding: 10px 25px;
                text-align: center;
                text-decoration: none;
                display: inline-block;
                font-size: 16px;
            }
        </style>
    </head>
    <body>
        <div id="area_trabajo">
            <table>
                <thead>
                    <tr>
                        <th>id</th>
```

```
                    <th>nombre</th>
                </tr>
            </thead>
            <tbody id="datosTabla">

            </tbody>
        </table>
    </div>
    <input type="button" id="btnLlenar" name="llenar" value="llenar tabla">

</body>
<script>
    var personas=[{
            "id":"01",
            "nombre":"juan"
        },{
            "id":"02",
            "nombre":"toño"
        },
        {
            "id":"03",
            "nombre":"sánchez"
        },
        {
            "id":"04",
            "nombre":"rodriguez"
        },
        {
            "id":"05",
            "nombre":"lopitos"
        }
    ];


    window.onload=function(){
        document.getElementById("btnLlenar").addEventListener("click", function(){
            var content="";
            personas.forEach(function(element) {
                content+="<tr>";
                //se accede al json con:element["id"] o con:element.id
```



```
                    content+="<td>"+element["id"]+"</td><td>"+element.nombre+"</td>";
                    content+="</tr>";
                });
                document.getElementById("datosTabla").innerHTML=content;
            });

        }
    </script>
</html>
```

11 Acceso al DOM

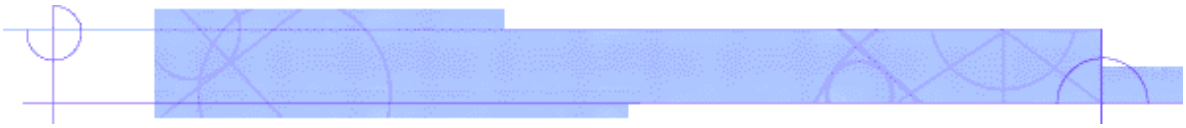
El modelo de objeto de documento (DOM) es una interfaz de programación para los documentos HTML y XML. Facilita una representación estructurada del documento y define de qué manera los programas pueden acceder, al fin de modificar, tanto su estructura, estilo y contenido. El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. Esencialmente, conecta las páginas web a scripts o lenguajes de programación.

Una página web es un documento. Éste documento puede exhibirse en la ventana de un navegador o también como código fuente HTML. Pero, en los dos casos, es el mismo documento. El modelo de objeto de documento (DOM) proporciona otras formas de presentar, guardar y manipular este mismo documento. El DOM es una representación completamente orientada al objeto de la página web y puede ser modificado con un lenguaje de script como JavaScript.

El W3C DOM estándar forma la base del funcionamiento del DOM. Varios navegadores ofrecen extensiones más allá del estándar W3C, hay que ir con extremo cuidado al utilizarlas en la web, ya que los documentos pueden ser consultados por navegadores que tienen DOMs diferentes.

Por ejemplo, el DOM de W3C especifica que el étodo ***getElementsByTagName*** en el código de abajo debe devolver una lista de todos los elementos ***<p>*** del documento:

```
paragraphs = document.getElementsByTagName ("p");
// paragraphs[0] es el primer elemento <p>
```

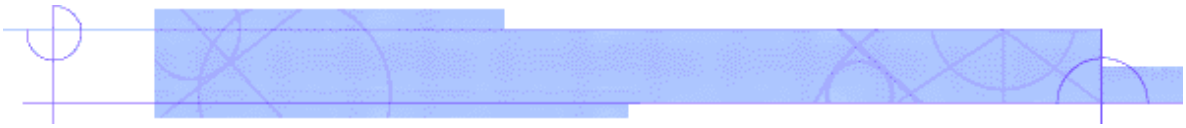


```
// paragraphs[1] es el segundo elemento <p>, etc.
alert (paragraphs [0].nodeName);
```

Todas las propiedades, métodos y eventos disponibles para la manipulación y la creación de páginas web están organizados dentro de objetos. Un ejemplo: el objeto ***`document`*** representa al documento mismo, el objeto ***`table`*** hace funcionar la interfaz especial ***`HTMLTableElement`*** del DOM para acceder a tablas HTML, y así sucesivamente. Ésta documentación procura una relación objeto-por-objeto del DOM.
Dentro de un script JS tiene acceso a objetos de DOM, por ejemplo los objetos ***document*** o ***window***.

Algunos objetos importantes para manipular el DOM son:

`document`	Cuando un miembro devuelve un objeto del tipo `document` (por ejemplo, la propiedad **ownerDocument** de un elemento devuelve el documento "`document`" al cual pertenece), este objeto es la raíz del objeto documento en sí mismo.
`element`	`element` se refiere a un elemento o a un nodo de tipo de elemento "`element`" devuelto por un miembro del API de DOM. Dicho de otra manera, por ejemplo, el método `document.createElement()` devuelve un objeto referido a un nodo, lo que significa que este método devuelve el elemento que acaba de ser creado en el DOM. Los objetos `element` ponen en funcionamiento a la interfaz `Element` del DOM y también a la interfaz de nodo "`Node`" más básica, las cuales son incluidas en esta referencia.
`nodeList`	Una "`nodeList`" es una serie de elementos, parecido a lo que devuelve el método `document.getElementsByTagName()`. Se accede a los items de la `nodeList` de cualquiera de las siguientes dos formas: list.item (1) lista [1] Ambas maneras son equivalentes. En la primera, **item()** es el método del objeto `nodeList`. En la última se utiliza la típica sintaxis de acceso a listas para llegar al segundo ítem de la lista.
`attribute`	Cuando un atributo ("`attribute`") es devuelto por un miembro (por ej., por el método **createAttribute()**), es una referencia a un objeto que expone una interfaz particular (aunque limitada) a los atributos. Los atributos son nodos en el DOM igual que los elementos, pero no suelen usarse así.
`NamedNodeMap`	Un `namedNodeMap` es una serie, pero los ítems son accesibles tanto por el nombre o por un índice, este último caso es meramente una conveniencia para enumerar ya que no están en ningún orden en particular en la lista. Un `NamedNodeMap` es un método de ítem() por esa razón, y permite poner o quitar ítems en un `NamedNodeMap`



A continuación, una lista de métodos comunes de acceso al DOM:

document.getElementById(id)

Devuelve una referencia aun elemento por si id, ejemplo:

Archivo: getElementById.html

```
<html>
<head>
   <title>Ejemplo getElementById</title>
</head>
<body>
   <p id="para">Cualquier texto acá</p>
   <button onclick="changeColor('blue');">Azul</button>
   <button onclick="changeColor('red');">Rojo</button>
</body>
<script>
    function changeColor(newColor) {
        var elem = document.getElementById('para');
        elem.style.color = newColor;
    }
</script>
</html>
```

element.getElementsByTagName(name)

Devuelve una lista de elementos con un nombre determinado.

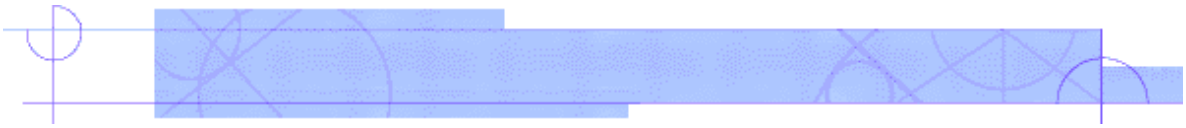
Archivo: getElementByTagName.html

```
<html>
<head>
<title>ejemplo de getElementsByTagName</title>
<script>
    function getAllParaElems(){
        var allParas = document.getElementsByTagName("p");
        var num = allParas.length;
        alert("Hay " + num + " <p> elementos en este documento");
    }
    function div1ParaElems(){
        var div1 = document.getElementById("div1")
        var div1Paras = div1.getElementsByTagName("p");
```

```
        var num = div1Paras.length;
        alert("Hay " + num + " <p> elementos en el elemento div1");
    }
    function div2ParaElems(){
        var div2 = document.getElementById("div2")
        var div2Paras = div2.getElementsByTagName("p");
        var num = div2Paras.length;
        alert("Hay " + num + " <p> elementos en el elemento div2");
    }
</script>
</head>
    <body style="border: solid green 3px">
        <p>Algo de texto</p>
        <p>Algo de texto</p>
        <div id="div1" style="border: solid blue 3px">
            <p>Algo de texto en div1</p>
            <p>Algo de texto en div1</p>
            <p>Algo de texto en div1</p>
            <div id="div2" style="border: solid red 3px">
            <p>Algo de texto en div2</p>
            <p>Algo de texto en div2</p>
            </div>
        </div>
        <p>Algo de texto</p>
        <p>Algo de texto</p>
        <button onclick="getAllParaElems();">
        muestra todos los elementos p en el documento</button><br />
        <button onclick="div1ParaElems();">
        muestra todos los elementos p en div1</button><br />
        <button onclick="div2ParaElems();">
        muestra todos los elementos p en div2</button>
    </body>
</html>
```

V.5. Procedimiento

1. En la sección de ejercicios se describen una serie de problemas en los que es necesario aplicar lenguaje JS para darles solución.



2. Elaborar el análisis del problema codificar adecuadamente una solución.
3. Codificar el programa en lenguaje JS.
4. Probar el programa en 3 navegadores diferentes.

V.6. Prácticas

1. Esciba una página web que muestre un saludo en una ventana emergente al iniciar.
2. Repita el ejercicio anterior, pero el saludo debe mostrarse al presionar un boton.
3. Defina una cadena JSON para representar la información de los alumnos del curso de cliente servidor 2, los datos requeridos son matricula, nombre, sexo y tres calificaciones por alumno.
4. Defina una cadena JSON para representar la información de los medicamentos en una farmacia, con datos id, nombre, descripción, caducidad, existencia, laboratorio que lo produce, costo de compra y costo de venta.
5. Defina una cadena JSON para representar la información del plan de estudio de las carreras de la FCI (clave y nombre de carrera, gestor y lista de materias por semestre).
6. Escriba una página web que capture los datos de 3 alumnos con los siguientes campos: matricula, nombre y edad, los datos deberá almacenarlos en una variable de memoria JSON y mostrarlos en una tabla en la misma página al presionar un botón.
7. En la carpeta “ejemplos\js-puro\json” pruebe los ejercicios que ahí se encuentran y que muestran el uso de JSON.

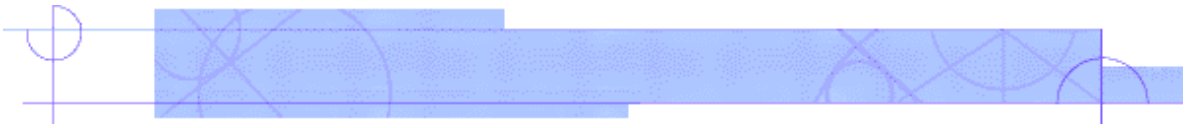
PRACTICA 7. PHP

VII.1. Objetivo

El alumno será capaz de elaborar rutinas que enlacen la interfaz de una aplicación web con una base de datos.

VII.2. Equipo

Computadora personal, dispositivo de almacenamiento secundario (disquete, memoria USB, etc.), lenguaje de programación pascal, c, java, etc.



VII.3. Materiales

Hojas de papel bond o cuaderno, lápiz o boligrafo.

VII.4. Descripción

1 Introducción.

En PHP vamos a comprobar qué tipo de navegador está utilizando el usuario visitante. Para hacerlo, vamos a comprobar el string del agente de usuario que el navegador envía como parte de la petición HTTP. Esta información es almacenada en una variable. En PHP, las variables siempre comienzan con un signo de dólar. La variable que nos interesa ahora es *$_SERVER['HTTP_USER_AGENT']*.

$_SERVER es una variable especial reservada por PHP que contiene toda la información del servidor web. Es conocida como una superglobal. Antes se podían usar en su lugar los antiguos arrays *$HTTP_*_VARS*, tales como *$HTTP_SERVER_VARS*. A partir de PHP 5.4.0, estos antiguos arrays han sido eliminados.

pruebe el programa:

index.php

```
<?php
echo $_SERVER['HTTP_USER_AGENT'];
?>
```

Tabla 7.1 Código

Para el código de la tabla 7.1, captúrelo en su editor, guárdelo en la carpeta “C:\wamp\www\pruebas" o “C:\wamp64\www\pruebas" según haya instalado el WAMPO de 32 o 64 bits. Arranque el servidor y:

ejecútelo así: En su navegador escriba: http://localhost/pruebas/index.php

deberá ver lo siguiente:

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36

2 Bases de datos

Para los ejemplos de este apartado requerirá:

1. Tener instalado alguna versión de MySql con Apache, en estos ejemplos se usa WAMP server. Si no lo ha instalado vea el video:
    - https://www.youtube.com/watch?v=5X7symb1QdE&list=PLExyZ7hD-J5b8ftq57xTwBceLXP6ErJzc&index=1
2. Tener instalado Heidi SQL
3. Tener implementada la base de datos "provpar2", vea el código en la carpeta “ejercicios/baseDeDatos/provPar2.sql", puede generar la base de datos ejecutando este código en una ventana de consulta de HEIDI. Tamnbien puede crear esta base de datos a partir de los videos:
    - https://www.youtube.com/watch?v=40YYmufKXus&list=PLExyZ7hD-J5b8ftq57xTwBceLXP6ErJzc&index=2
    - https://www.youtube.com/watch?v=5byoF-qF86U&list=PLExyZ7hD-J5b8ftq57xTwBceLXP6ErJzc&index=3
4. Si desea saber cómo instalar Wamp y Heidi, vea el archivo de apoyo instalar WAMP y Heidi.

Base de datos provpar2:

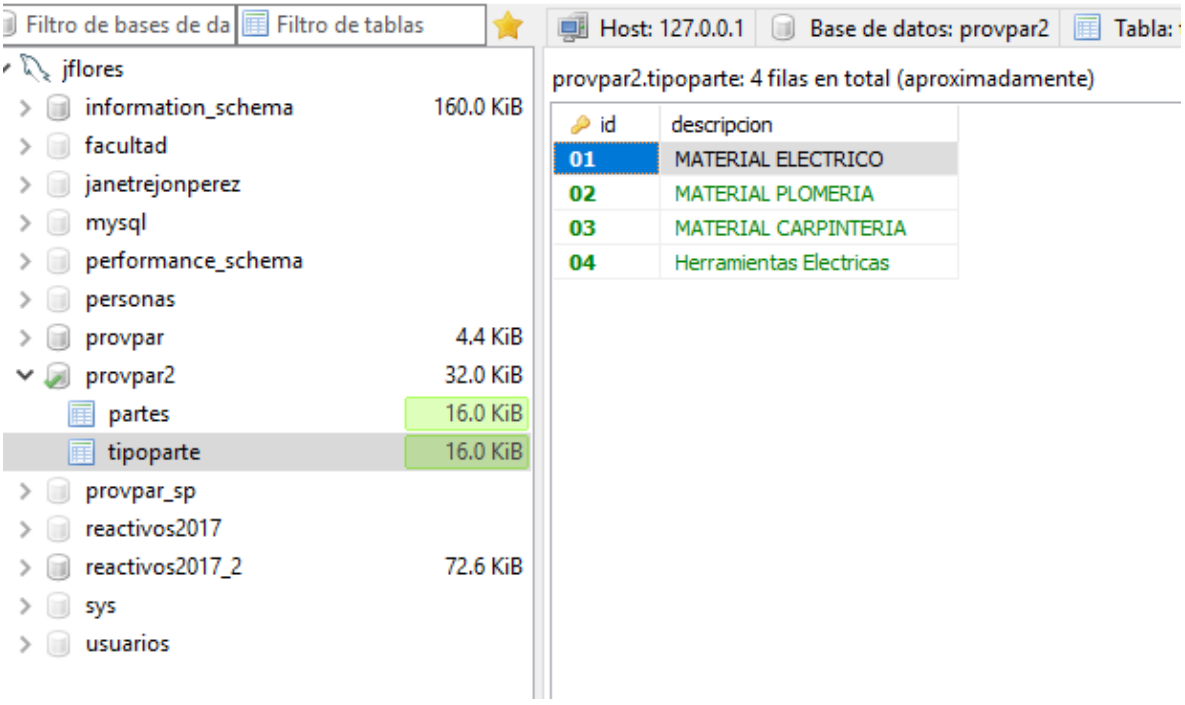


Tabla tipoparte de la base de datos provpar2

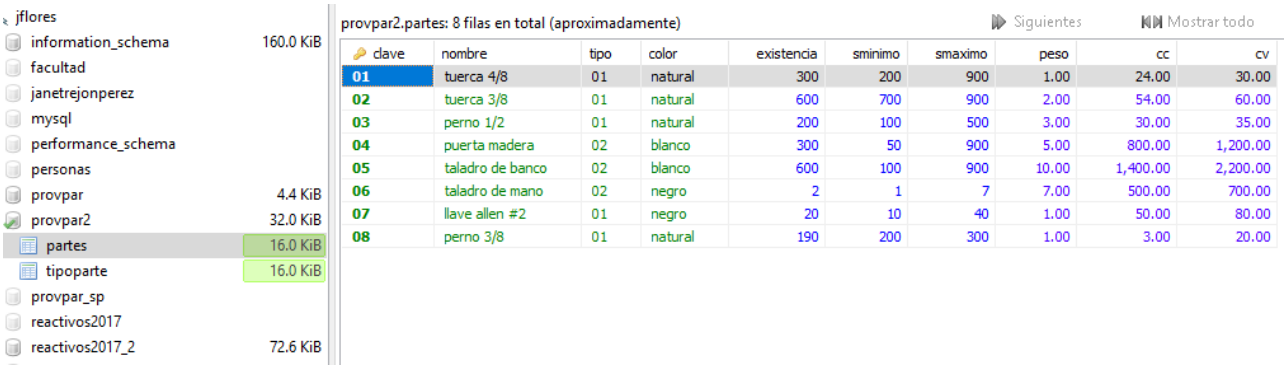


Tabla partes de la base de datos provpar2.

Aquí el código para crear la base de datos:

Copie este código en una ventana de consulat de heidisql y ejecútelo, esto creará su base de datos provpar2.

```
-- --------------------------------------------------------
-- Host:                         127.0.0.1
-- Versión del servidor:         5.7.14 - MySQL Community Server (GPL)
-- SO del servidor:              Win64
-- HeidiSQL Versión:             9.4.0.5125
-- --------------------------------------------------------

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET NAMES utf8 */;
/*!50503 SET NAMES utf8mb4 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;


-- Volcando estructura de base de datos para provpar2
CREATE DATABASE IF NOT EXISTS `provpar2` /*!40100 DEFAULT CHARACTER SET latin1 COLLATE
latin1_spanish_ci */;
USE `provpar2`;

-- Volcando estructura para tabla provpar2.partes
CREATE TABLE IF NOT EXISTS `partes` (
  `clave` varchar(10) COLLATE latin1_spanish_ci NOT NULL DEFAULT '',
  `nombre` varchar(50) COLLATE latin1_spanish_ci DEFAULT NULL,
  `tipo` varchar(10) COLLATE latin1_spanish_ci DEFAULT NULL,
  `color` varchar(20) COLLATE latin1_spanish_ci DEFAULT NULL,
```

```
  `existencia` int(11) DEFAULT NULL,
  `sminimo` int(11) DEFAULT NULL,
  `smaximo` int(11) DEFAULT NULL,
  `peso` decimal(10,2) DEFAULT NULL,
  `cc` decimal(10,2) DEFAULT NULL,
  `cv` decimal(10,2) DEFAULT NULL,
  PRIMARY KEY (`clave`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_spanish_ci;

-- Volcando datos para la tabla provpar2.partes: ~8 rows (aproximadamente)
/*!40000 ALTER TABLE `partes` DISABLE KEYS */;
INSERT INTO `partes` (`clave`, `nombre`, `tipo`, `color`, `existencia`, `sminimo`,
`smaximo`, `peso`, `cc`, `cv`) VALUES
  ('01', 'tuerca 4/8', '01', 'natural', 300, 200, 900, 1.00, 24.00, 30.00),
  ('02', 'tuerca 3/8', '01', 'natural', 600, 700, 900, 2.00, 54.00, 60.00),
  ('03', 'perno 1/2', '01', 'natural', 200, 100, 500, 3.00, 30.00, 35.00),
  ('04', 'puerta madera', '02', 'blanco', 300, 50, 900, 5.00, 800.00, 1200.00),
  ('05', 'taladro de banco', '02', 'blanco', 600, 100, 900, 10.00, 1400.00, 2200.00),
  ('06', 'taladro de mano', '02', 'negro', 2, 1, 7, 7.00, 500.00, 700.00),
  ('07', 'llave allen #2', '01', 'negro', 20, 10, 40, 1.00, 50.00, 80.00),
  ('08', 'perno 3/8', '01', 'natural', 190, 200, 300, 1.00, 3.00, 20.00);
/*!40000 ALTER TABLE `partes` ENABLE KEYS */;

-- Volcando estructura para tabla provpar2.tipoparte
CREATE TABLE IF NOT EXISTS `tipoparte` (
  `id` varchar(10) COLLATE latin1_spanish_ci NOT NULL DEFAULT '',
  `descripcion` varchar(80) COLLATE latin1_spanish_ci DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_spanish_ci;

-- Volcando datos para la tabla provpar2.tipoparte: ~3 rows (aproximadamente)
/*!40000 ALTER TABLE `tipoparte` DISABLE KEYS */;
INSERT INTO `tipoparte` (`id`, `descripcion`) VALUES
  ('01', 'MATERIAL ELECTRICO'),
  ('02', 'MATERIAL PLOMERIA'),
  ('03', 'MATERIAL CARPINTERIA'),
  ('04', 'Herramientas Electricas');
/*!40000 ALTER TABLE `tipoparte` ENABLE KEYS */;

/*!40101 SET SQL_MODE=IFNULL(@OLD_SQL_MODE, '') */;
```

```
/*!40014 SET FOREIGN_KEY_CHECKS=IF(@OLD_FOREIGN_KEY_CHECKS IS NULL, 1,
@OLD_FOREIGN_KEY_CHECKS) */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

Tabla 7.2

3 Conexión a la base de datos.

Para conectarnos a la base de datos en PHP. Un programa en php se escribe en un archivo entre las marcas:

<?php

?>

Construyamos un código php para conectarnos a nuestra base de datos provpar:

Archivo: abre_provpar.php

```php
<?php
    $hostname="localhost";
    $dbname="provpar";
    $username="root";
    $pw="";
    try{
        $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username, $pw);
        echo '{"conexión abierta"}';
    }catch(PDOException $e){
        $error="conexion fallida: ".$e;
        echo '{"error":'.$error.'}';
        //echo "conexion fallida:".$e;
    }
    $dbh = null;
?>
```

Tabla 7.3

Este código abre la base de datos, y después la cierra con la sentencia ***$dbh=null,*** se agrega el try para capturar errores. Si todo va bien deberá ver en la ventana del

navegador el mensaje “conexión abierta”. En caso de error se muestra en la ventana el motivo del error.

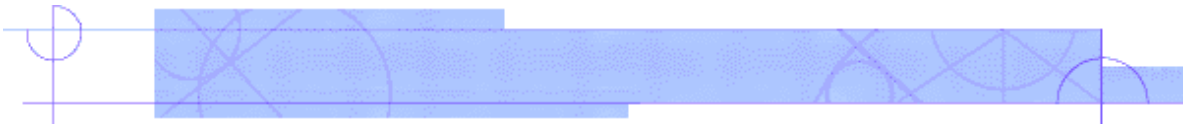
Como se ejecuta el programa:

1. Debe tener en ejecución el WAMP server.
2. Dentro de la carpeta ***wamp64/www/*** cree una nueva carpeta llamada: ***cursoapcnjs***
3. Abra su editor de código Miocrosoft Visual studio code (o programersNotepad si lo desea).
4. Escriba o copie el código del programa ***abre_provpar.php*** que aquí se muestra, en el editor, guardelo en la carpeta ***wamp64/www/cursoapcnjs,*** con el nombre ***abre_provpar.php.***
5. Abra una ventana de su navegador (Chrome, mozilla, Edge u opera)
6. En su ventana del navegador se ejecuta así:
    - En su navegador escriba: http://localhost/cursoapcnjs/abre_provpar.php
    - Deberá ver un mensaje en su ventana

Veamos ahora una consulta a la tabla ***tipoparte***:

Archivo: traeDatos.php

```php
<?php
    $hostname="localhost";
    $dbname="provpar2";
    $username="root";
    $pw="";
    try{
        $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username, $pw);
        // la consulta
        $sql = 'select * from tipoparte';
        // usar la sentencia prepara para verificar la sentencia
        $stmt = $dbh->prepare( $sql );
        // ejecuta la sentencia
        if($stmt->execute()){
            // poner el resultado de la consulta en el arreglo result
            $result = $stmt->fetchAll( PDO::FETCH_ASSOC );
            // convertir a formato json
            $json = json_encode( $result );
            // mandar la salida
            echo $json;
        }else{
            echo '{"error":"no se pudo realizar la consulta"}';
        }
    }catch(PDOException $e){
```



```
        $error="conexion fallida: ".$e;
        echo '{"error":'.$error.'}';
        //echo "conexion fallida:".$e;
    }
    $dbh = null;
?>
```

Tabla 7.4

Aquí se abre la base de datos como en la tabla 7.3 y se agrega el código para realizar la consulta, la sentencia:

$sql = 'select * from tipoparte';

Almacena la consulta en la variable $sql

La sentencia:

```
$stmt = $dbh->prepare( $sql );
```

No es obligatoria, pero es recomendable para verificar algunos errores.

La sentencia:

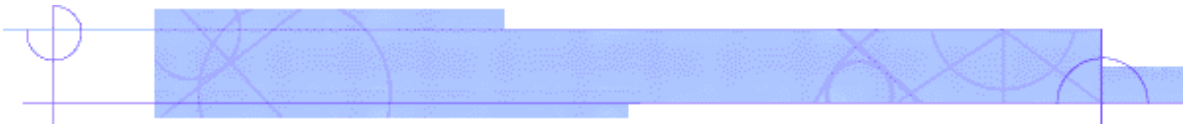
$stmt->execute()

Ejecuta la sentencia, devuelve null si no pudo ejecutarla o asocia un arreglo asociativo de php a la variable $stmt con los datos encontrados. Por tanto, al meterla al if verificamos si hay datos al momento de ejecutarla. Importante si la consulta esta vacia no devuelve null, devuelve un arreglo asociativo vacio, si no hay conexión o no existe la tabla o esta mal la consulta devuelve null y se ejecuta el else.

```
if($stmt->execute()){
            // poner el resultado de la consulta en el arreglo result
            $result = $stmt->fetchAll( PDO::FETCH_ASSOC );
            // convertir a formato json
            $json = json_encode( $result );
            // mandar la salida
            echo $json;
        }else{
            echo '{"error":"no se pudo realizar la consulta"}';
        }
```

Dentro del if, si trae datos la consulta, con la sentencia:

```
$result = $stmt->fetchAll( PDO::FETCH_ASSOC );
```



Devuelve los datos del arreglo asociativo y los almacena en la variable $result.
Finalmente, la sentencia:

```
$json = json_encode( $result );
echo $json;
```

Codifica el arreglo asociativo en formato json y con ***echo*** lo enviamos al navegador.

Aunque no lo usaremos, por curiosidad podemos ver como se muestran los valores sin convertirlos a json, cambie las líneas de código:

```
$json = json_encode( $result );
echo $json;
```

Por estas:

```
var_dump($result);
```

Tabla 7.5

Ahora debe ver los datos en el formato de arreglo asociativo.
Deje el código como antes para que devuelva un arreglo json.

Insertar datos

Para insertar datos primero veremos como enviar datos desde un archivo HTML, primera forma:

Necesitamos 2 archivos, el primero un html con un formulario, el segundo un php que reciba los datos:

Archivo: inserta.html

```
<!DOCTYPE html>
<html>
    <body>
        <form action="inserta.php" method="POST">
            id<input type="text" name="id">
            descripcion<input type="text" name="descripcion">
            <input type="submit">
        </form>
    </body>
</html>
```

Archivo: inserta.php

```
<?php
    //require_once('php/clsPartes.php');
    //$objpartes=new clsPartes();
```

```
    if(!empty($_POST)){
        $id = $_POST['id'];
        $descripcion = $_POST['descripcion'];
        echo "id=".$id." desc=".$descripcion;
    }else{
        echo "sin datos";
    }
?>
```

Tabla 7.6

El artchivo html tiene un formulario con 2 campos, con atributo name: ***id*** el primero y ***descripción*** el segundo. El formulario está sociando en su atributo action al archivo inserta.php, de manera que al hacer clic en el botón submit, se ejecutara el archivo ***inserta.php*** enviando los parametros ***id*** y ***descripción*** que se obtienen del contenido de los cuadros de texto.

En el archivo inserta.php con el ***if(empty($_POST))*** verifica que se hayan recibido datos con el método post, de ser así, con las sentencias:

```
$id = $_POST['id'];
$descripcion = $_POST['descripcion'];
```

Obtiene los datos y con ***echo*** los muestra en la ventana.

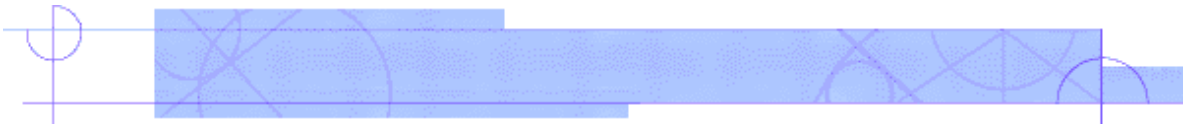
Copie estos archivos a su navegador, guárdelos con el nombre indicado en la carpeta ***wamp64/www/cursoapcnjs*** , pruebe así:

En su navegador ponga la sentencia: http://localhost/cursoapcnjs/inserta.html , ponga información en los dos campos y haga clic en el botón, deberá ver lainformación de los campos en una ventana nueva. Con esto esta comprobando que el PHP recibe y devuelve bien la información desde/hacia JS.

4 JS-fetch

En la insercion de datos, en el ejemplo de la tabla 7.6 se muestra la forma clásica de usar php, sin embargo, nosotros no lo usaremos de esta forma, trabajaremos así:

1. Craremos la interface en html+css+bootstrap.
2. Craremos las consultas de inserción de datos en php.



3. Crearemos rutinas en JS que tomen los datos de inserción de los formularios de html y los envíen a php.
4. Crearemos rutinas JS que tomarán los resultados de la consulata y los enviaremos a html.

Para el punto 3 usaremos la sentencia fetch que es muy eficiente a la hora de enviar y recibir datos desde el back end, la sentencia fetch trabaja de manera asíncrona usando un concepto en JS llamado promesas (Promise), para entender mejor las promesas vea una explicación en:

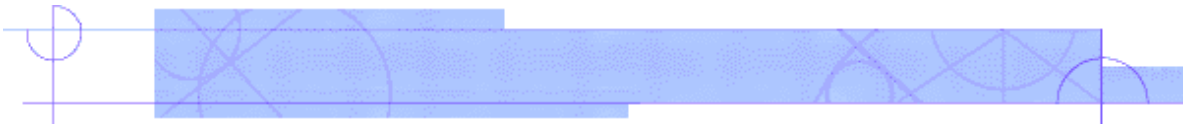
- Promesas: https://docs.google.com/document/d/1JZg8oCON5LG7j42r9l2sKdfRqvz4qf9DWKsOL2XEKVk/edit?usp=sharing
- Una explicación de fetch y php: https://docs.google.com/presentation/d/1WNqsq268kCPe4XUBkIKx5f-Dx4S7yorjiVyGnpnltKA/edit?usp=sharing
- fetch envía datos a php: https://docs.google.com/presentation/d/1Q3nFXiDohNoaFkwbcs7LSncihtR1DHFGwOaITnO-CJ0/edit?usp=sharing

- fetch envía formulario: https://docs.google.com/presentation/d/1ukDY8TcTNGk5F1AnX1MEPLyFC0sJKsjr9SqlpnMNoaE/edit?usp=sharing
- Fetch enviar parámetros en url:https://docs.google.com/presentation/d/1HMD4jTeg3zOOiKsrEqDlsvKUmEQfMMHGcksR9OUoL4o/edit?usp=sharing
- fetch leer json: https://docs.google.com/presentation/d/1PsWqwIe---sbqWFGBd4ormqIjZNJQS0U5k1CNmMphMA/edit?usp=sharing

Trabajaremos con fetch así:
HTML ←→ JS-fetch ←→PHP←→ Base de datos

Veamos como tomar datos de HTML para enviarlos a php.

HTML → JS-fetch →PHP



Para esto dberemos:

Crear el archivo html para capturar datos:

Archivo: fetch-basico-envia-formulario.html

```
<!DOCTYPE html>
<html>
    <head>
            <meta charset="utf-8">
    </head>
    <body>
        <form id="miformulario">
            <input type="number" name="movimiento" placeholder="1">
            <input type="text" name="id" placeholder="id">
            <input type="text" name="nombre" placeholder="nombre">
            <input type="text" name="apellido" placeholder="apellido">
            <input type="number" name="edad" placeholder="18">
            <input type="button" id="mi_boton" onclick="enviarDatos()" value="enviar
datos">
        </form>
        <div id="div1"></div>
    </body>
    <script>
        function enviarDatos(){
            const data = new FormData(document.getElementById('miformulario'));
            var url = 'recibe_Form_devuelveTXT.php';
            fetch(url, {
                method: 'POST',
                body: data
            }).then(res => res.text())
            .catch(error => console.error('Error:', error))
            .then(response => {
                console.log('Success:', response);
                document.getElementById("div1").innerText=response;
            });
        }
    </script>
</html>
```

Archivo php que recibe los datos del formulario y los envía a la base de datos, en este ejemplo para probar el funcionamiento, no se envían los datos a la base de datos, si no que se devuelven al JS para verificar la comunicación:

Archivo: recibe_Form_devuelveTXT.php

```php
<?php
            header("Content-Type: text/html;charset=utf-8");
            if(! empty($_POST)){
                //se enviaron parametros por el método _GET

$movimiento=(isset($_POST["movimiento"])?$_POST["movimiento"]:"ninguno");
                $id=(isset($_POST["id"])?$_POST["id"]:"");
                $nombre=(isset($_POST["nombre"])?$_POST["nombre"]:"");
                $apellido=(isset($_POST["apellido"])?$_POST["apellido"]:"");
                $edad=(isset($_POST["edad"])?$_POST["edad"]:0);
                echo "recibi mov:".$movimiento.", id:".$id.",nombre: ".$nombre.",
apellido:".$apellido.", edad:".$edad;
            }else{
                echo "sin datos recibidos";
            }
            exit();
?>
```

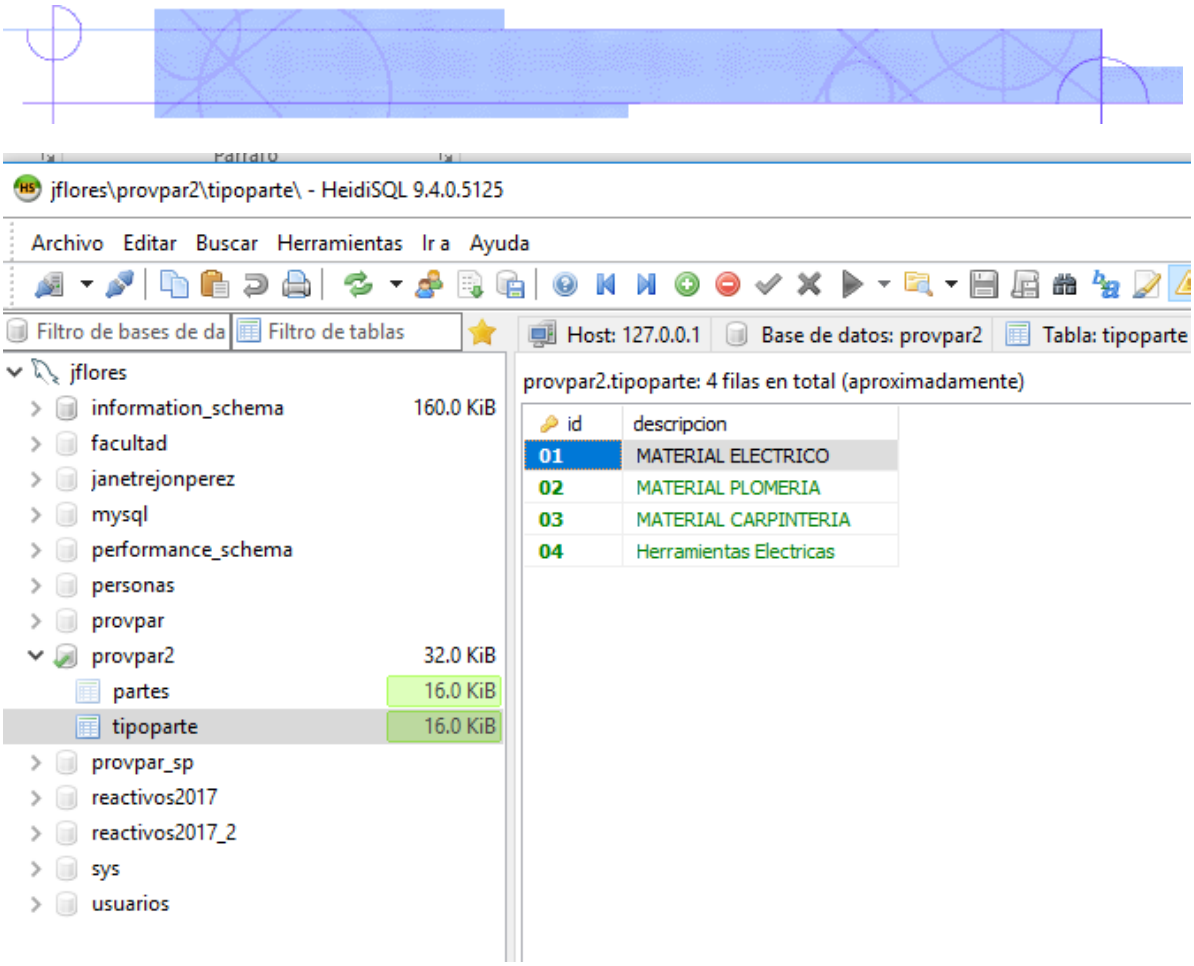
Pruebelo así:

1. Guarde los dos archivos en la carpeta: C:\wamp64\www\cursoapcnjs
2. En su navegador ejecute así: http://localhost/cursoapcnjs/fetch-basico-envia-formulario.html
3. Como resultado debe ver que el html devuelve la misma información que recibe.

5 Insertar registro con fetch

Veamos como insertar un registro.
HTML → JS-fetch →PHP→ Base de datos

Insertaremos un nuevo tipo de partes en la tabla tipoparte de provpar que tiene los campos id y descripción:



Necesitaremos 2 archivos:

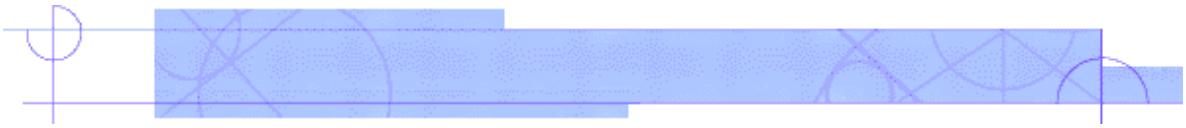
Archivo: inserta_tipo_articulo.html

```
<!DOCTYPE html>
<html>
    <head>
            <meta charset="utf-8">
    </head>
    <body>
        <p>Inserción de datos a la tabla tipo de articulo</p>
        <form id="miformulario">
            <input type="text" name="id" placeholder="id">
            <input type="text" name="descripcion" placeholder="nombre">
            <input type="button" id="mi_boton" onclick="enviarDatos()" value="enviar datos">
        </form>
        <div id="div1"></div>
    </body>
    <script>
```

```
        function enviarDatos(){
            //http://localhost/ejercicios/json-fetch/fetch-basico/fetch-basico-envia-
formulario.html
            const data = new FormData(document.getElementById('miformulario'));
            var url = 'inserta_tipo_articulo.php';
            fetch(url, {
                method: 'POST',
                body: data
            }).then(res => res.text())
            .catch(error => console.error('Error:', error))
            .then(response => {
                console.log('Success:', response);
                document.getElementById("div1").innerText=response;
            });
        }
    </script>
</html>
```

Archivo: inserta_tipo_articulo.php

```
<?php

    header('Content-type: text/html; charset=UTF-8') ;
    //tabla a la que se le hacen los movimientos
    $mi_tabla='tipoparte';

    $hostname="localhost";
    $dbname="provpar2";
    $username="root";
    $pw="";
    if (isset($_POST) && count($_POST)>0){
        $id=(isset($_POST["id"])?$_POST["id"]:"");
        $descripcion=(isset($_POST["descripcion"])?$_POST["descripcion"]:"");
        //
        $sql="insert into $mi_tabla(id,descripcion)";
        $sql.=" values('$id','$descripcion');";
        try{
            $dbh = new PDO("mysql:host=$hostname;dbname=$dbname", $username, $pw);
            // usar la sentencia prepare para verificar la sentencia
            $stmt = $dbh->prepare( $sql );
            // ejecuta la sentencia
```



```
            if($stmt->execute()){
                echo '{"error":"insersion sin error"}';
            }else{
                echo '{"error":"no se pudo realizar la consulta"}';
            }
        }catch(PDOException $e){
            $error="conexion fallida: ".$e;
            echo '{"error":'.$error.'}';
            //echo "conexion fallida:".$e;
        }
        $dbh = null;
    }
    exit();

?>
```

Explicación

En el archivo html. Tenemos un formulario y un botón asociado en su evento clic a la función enviarDatos(). Al dar clic se ejecuta la función de JS, el la función de JS enviarDatos() se realízalo siguiente, la sentencia:

```
const data = new FormData(document.getElementById('miformulario'));
```

Crea la variable data con la información del formulario, la sentencia:

var url = 'inserta_tipo_articulo.php';

declara una variable con el nombre del archivo php a ejecutar.

La sentencia:

```
fetch(url, {
                method: 'POST',
                body: data
            }).then(res => res.text())
            .catch(error => console.error('Error:', error))
            .then(response => {
                console.log('Success:', response);
                document.getElementById("div1").innerText=response;
```

```
        });
```

Realiza todo el trabajo, Indica que se enviarán los datos por el método post, en el body pone los datos obtenidos del formulario, recuerde que el url tiene la dirección del archivo php a ejecutar.

A continuación, una lista de documentos que explica a mas detalle el uso de fetch y las promesas:

6 Recibir datos con fetch desde php

HTML ← JS-fetch ←PHP←Base de datos

A continuación, unejemplo para recibir datos desde php en JS y ponerlos en una tabla de html:

Archivo: muestraPersonas.html

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            table{ width:50%; }
            td{ text-align:center; }
            tr:hover{ background-color:#edd; }
            tr:nth-child(even){background-color:#f2f2f2;}
        </style>
    </head>
    <body>
        <div id="area_trabajo">
            <table>
                <thead>
                    <tr>
                        <th>id</th>
                        <th>nombre</th>
                        <th>edad</th>
                        <th>ciudad</th>
                        <th>trabajo</th>
                    </tr>
```

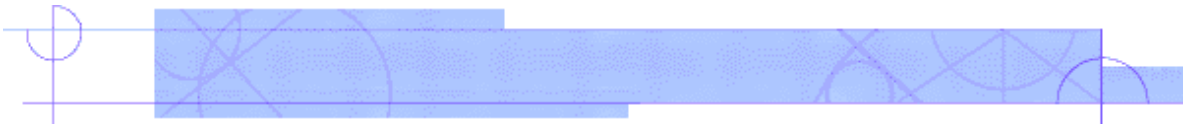
```
                </thead>
                <tbody id="datosTabla">
                </tbody>
            </table>
        </div>
    </body>
    <script>
        //http://localhost/ejercicios/json-
fetch/ejemploPersonas/muestraPersonas.html
        var personas={};
        //fetch('traePersonas-mysqli.php')
        fetch('traePersonasPDO')
        .then(function(response) {

            return response.json();
        })
        .then(function(data) {
            //console.log('data = ', data);
            personas=data.salida;
            //console.log(personas);
            var content="";
            personas.forEach(function(element) {
                    content+="<tr>";
                    //nota: como se ve a continuación, se accede al json con:
element["id"] o con:element.id

content+="<td>"+element["id"]+"</td><td>"+element.nombre+"</td>";

content+="<td>"+element["edad"]+"</td><td>"+element['ciudad']+"</td>";
                    content+="<td>"+element.trabajo;
                    content+="</tr>";
            });
            document.getElementById("datosTabla").innerHTML=content;
        })
        .catch(function(err) {
            console.error(err);
        });
    </script>
</html>
```

Archivo:traePersonasPDO.php

```php
<?php
    //http://localhost/ejercicios/json-
fetch/ejemploPersonas/traePersonasPDO.php
    //crear la conexion a la base de datos
    try{
        //intenta la conexión
        $conexion = new
PDO("mysql:host=localhost;dbname=personas;charset=utf8", "root", "");
    } catch (PDOException $e) {
        //atrapa el error
        echo "Falla al obtener un manejador de BD: ".$e->getMessage() . "\n";
        exit();
    }

    // realizar la consulta y llenar la tabla
    $query = $conexion->prepare("SELECT * FROM persona;");
    $query->execute();

    if($query->rowCount() > 0){
        $userData = $query->fetchAll(PDO::FETCH_ASSOC);
        $datos["status"]="ok";
        $datos["salida"]=$userData;
    }else{
        $datos["status"]="ok";
        $datos["salida"]=NULL;
    }
    unset($conexion);
    echo json_encode($datos);
    //var_dump($userData);

?>
```

PRACTICA 8. Back end con Node JS

8.1 introduccion



Node JS es un entorno de ejecución que permite usar el lenguaje JS del lado del servidor, lo pude descargar desde: https://nodejs.org/en/download/current

Una vez instalado, vamos a crear una nueva carpeta llamada proyecto1, a continuación, abra Visual Studio Code (VSC) en esa carpeta, y en VSC abra una consola de Bash (dispone de bash si ya instalo GIT) o también puede abrir una ventana de CMD (aquí usaremos la consola de bash, puede abrir la consola, presionando ctrl+ñ). En la cosnola ejecute el comando (node --version):

```
jflor@DESKTOP-MCN7GVO MINGW64 /c/trabajo2024/proyecto1
$ node --version
v19.7.0

jflor@DESKTOP-MCN7GVO MINGW64 /c/trabajo2024/proyecto1
$
```

Verá la versión de node que instaló.

En VSC creé un archivo nuevo “saludo.js” escriba un código en JS, por ejemplo.

Archivo: saludo.js
console.log("hola mundo")

El “;” al final de cada línea en JS es opcional.

Ejecute el programa con el siguiente comando en la consola (sin el $):

$ node saludo.js
Podríamos seguir probando el lenguaje, pero nuestro interés en hacer una API, vayamos directamente a los pasos a seguir para construirla.

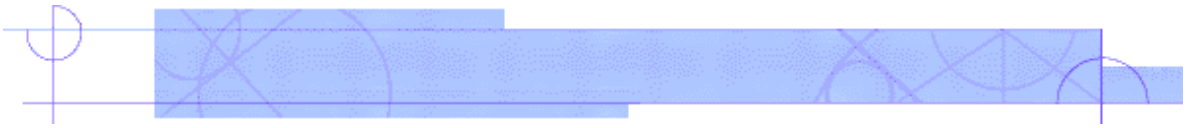
NPM

Node Package Manager, manejador de paquetes de node, es una aplicación que se instala con node y sirve para gestionar paquetes e iniciar proyectos, los paquetes en Node son como las bibliotecas de código en otros lenguajes.

NPM también me permite iniciar proyectos, esto es muy útil pues me permite tener registro y control sobre los paquetes de los que depende mi aplicación, lo que llamamos dependencias del proyecto.

Puede ver la versión instalada de npm así:

$npm --version



Creación de un proyecto con npm.

Vamos a crear un nuevo proyecto para programar una api muy básica.

1. Creamos una nueva carpeta llamada pry-personas.
2. Abrimos esa carpeta con Code (clic derecho en la carpeta y clic en abrir como).
3. Iniciamos el proyecto ejecutado en la consola
    a. (***npm init -y***).
4. Cargamos la biblioteca Express que es un framework para la creación de nuestro backend, hacemos esto ejecutando el siguiente comando en la consola
    a. (***npm install express --save***)

Hasta ahora tiene una carpeta que contiene:

- Una carpeta node_modules (donde se descargaron los módulos necesarios para usar Express).
- Un archivo package-lock.json que contiene un json con la descripción de las dependencias de su proyecto.
- Un archivo package.json que contiene la descripción de nuestro proyecto.

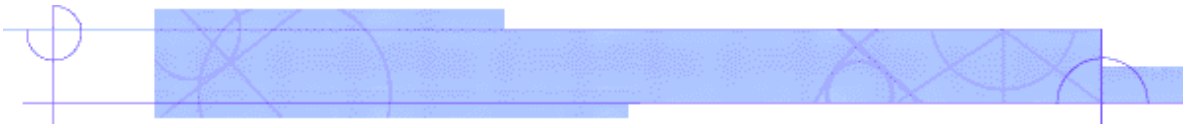
Si abre el archivo *package.json*, vera la entrada: "main":"index.js"
Que indica que el programa principal de nuestra aplicación es el archivo index.js.

Hola mundo con Express

1. En la carpeta raíz de nuestro proyecto, creamos un archivo llamado index.js.
2. Agregamos el código:

```
const express = require('express')
const app = express()
const port = 3000
app.get('/', (req, res) => {
        res.send('Hello World!')
})

app.listen(port, () => {
        console.log(`Example app listening on port ${port}`)
})
```

3. Para ejecutarlo:
    a. Ejecute en su consola (***node index.js***), esto arranca un servidor en http://localhost:3000
    b. Ponga esta liga en su navegador, deberá ver el letrero "Hello World!"



c. También puede probarlo en una nueva consola de gitbash (puede abrir una consola de bash dando clic derecho en su carpeta y eligiendo la opción gitbash here), en su nueva consola pruebe nuestra api, usando el comando:
    i. ***curl http://localhost:3000/***
d. También puede probarlo abriendo una ventana de Thunder Client, poniendo la dirección antes mencionada y seleccionando el método get (vea el uso de la herramienta Thunder Client e este mismo manual).
e. Para detener el servidor, presione ***ctrl-c*** en su terminal de VSC.

Explicación del código:

La línea: ***const express = require ('express')***
Carga la biblioteca Express.

La línea: ***const app=express()***
Crea una nueva instancia de express en la constante app.

```
app.get('/', (req, res) => {
        res.send('Hello World!')
})
```

Define una función a ejecutar cuando se reciba una solicitud get, aquí indicamos que al recibir una solicitud get en la raíz de nuestra ruta ('/'), se ejecute la función callback:

```
(req, res) => {
        res.send('Hello World!')
}
```

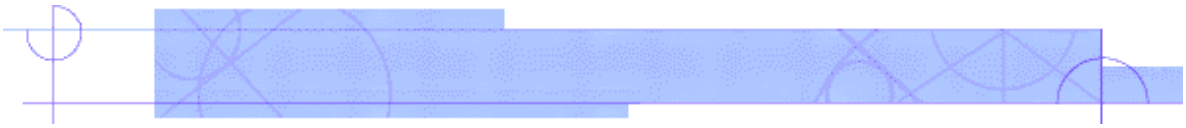
A esta función callback se le envían los parámetros res y req, req tiene la información de la petición: datos como su encabezado y el cuerpo de la solicitud, y res es un objeto mediante el cual enviaremos la respuesta de la solicitud, al cliente.

En este caso estamos enviando al cliente la respuesta: 'Hello World!'

En resumen estamos diciendo que cuando se reciba una solicitud a la ruta: http://localhost:3000/

Debemos devolver 'Hello World!' al cliente que hizo la solicitud, por esta razón al poner esta dirección en su navegador, verá el letrero 'Hello World!'.

Devolver html.



También podemos devolver html y de paso veremos como recibir parámetros enviados desde la solicitud http.

```
const express = require('express')
const app = express()
const port = 3000

//se llamará con: http://localhost:3000/user/juan perez
app.get('/user/:name', (req, res) => {
  res.send(`
  <h1>binevenidos a express nombre=${req.params.name}</h1>
  `)
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Pruébelo desde su navegador escribiendo la ruta:
`http://localhost:3000/user/ana pacheco`

o poniendo en una nueva consola el comando:

```
curl http://localhost:3000/user/ana pacheco
```

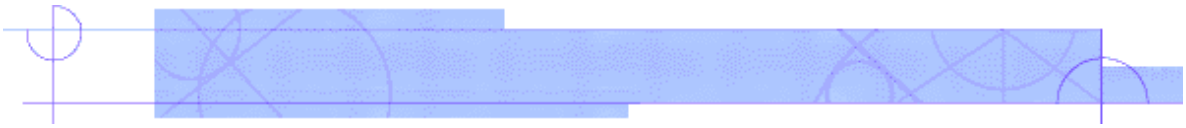
Enviar solicitud POST

Para obtener los datos del body de una solicitud, agregue esta línea de código al inicio de su código después de cargar sus bibliotecas.

app.use(express.urlencoded({extended:false}));

Esta línea decodificara los parámetros del body enviados mediante 'urlencoded', debe ponerla antes de sus rutas.
Agregue este código a su archivo index.js

```
//llamada a post con un body
app.post('/user',(req,res)=>{
  const {id,nombre,correo,edad}=req.body
  console.log({id,nombre,correo,edad})
  res.set({"content-type":"application/json; charset=utf-8"})
```



```
  res.status(200).send({id,nombre,correo,edad})
})
```

Este código asigna la ruta /user enviada con post a esta función, la función hace:

1. Destructura los datos enviados en el body: ***const {id,nombre,correo,edad}=req.body***
2. Indica que la respuesta se envía en formato json***:*** ***res.set({"content-type":"application/json; charset=utf-8"})***
3. Envia de regreso los mismos datos para verificar que se recibieron bien: ***res.status(200).send({id,nombre,correo,edad})***

Pruébelo así:
curl -X POST -d 'id=23' -d 'nombre=ana perez' -d 'correo=ana@mail' -d 'edad=32' http://localhost:3000/user

ejemplo: archivo index.js

Archivo: .env

```
PORT=3000
DB_PLATFORM="sqlite"
SQLITE3_DB_NAME="dataBase.sqlite"
SQLITE3_DB_MEMORY=":memory:"
MYSQL_HOST=""
MYSQL_DB_NAME=""
MYSQL_USER=""
MYSQL_PASSWORD=""
URL="http://localhost"
```

Archivo: index.js

```
import express from "express"
import dotenv from 'dotenv'

dotenv.config() //cargar las variables de entorno del archivo
.env en process.env
const PORT=process.env.PORT

const app=express()
```

```
//middleware para  aceptar json y texto
app.use(express.json())
app.use(express.text())
app.use(express.urlencoded({extended:false}));
//parametros en la url
//espera una url asi: http://localhost:3000/user/779-jflores-59
app.get("/user/:id-:name-:edad",(req,res)=>{
    res.set({"content-type":"text/html; charset=utf-8"})
    //params almacena los  parametros enviados por la url
    console.log(req.params)
    res.end(
        `
        <h1>${req.params.name} binevenidos a express
id=${req.params.id}</h1>
        `
    )
})
//parametros query
//espera:http://localhost:3000/search?id=779&nombre=alejandro
flores&edad=59
app.get('/search',(req,res)=>{
    console.log(req.query)
    res.send()

})
//parametros en el body
//espera:http://localhost:3000/account  , con un body en formato
json o text
app.post('/account',(req,res)=>{
    console.log(req.body)
    res.send()
})
app.patch('/account',(req,res)=>{

})
```

```
app.delete('/account',(req,res)=>{

})

app.listen(PORT,()=>{
    console.log(`iniciando express desde:
http://localhost:${PORT}/`)
})
```

Ejercicio:

1. Pruebe este ejercicio usando Thunder Client.

8.2 base de datos

8.3 Ejercicio

Inicie un proyecto nuevo:

- Creé una carpeta nueva: 'pry-bd-1'
- Dentro de esa carpeta, en una ventana de consola. Creé el proyecto: ***npm init -y***
- Instale la dependencia de sqlite: ***node install sqlite3***
- Cree un archivo llamado *index.js* y ponga el código: ***console.log("hola mundo")***
- En la consola. Ejecute así: ***node index.js***

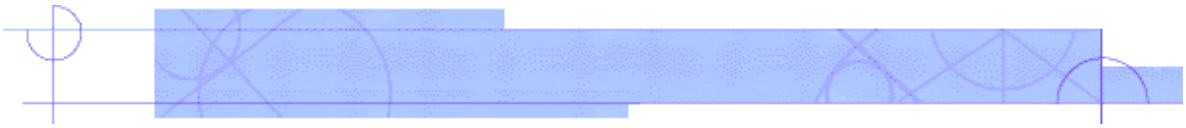
También puede crear una nueva entrada en script de Package.json, así:

```
"scripts": {
    "test": "node ./index.js"
  }
```

Y ejecutar así: ***npm run test***

Usando la interfaz que quiera creé una base de datos ('escolar.sqlite') con la única tabla carreras con campos:

```
CREATE Table
   if not exists carreras(
      id integer PRIMARY KEY autoincrement,
      alias VARCHAR(10),
```



```
    nombre VARCHAR(30),
    creditos integer
);
```

Agregue un registro:

```
Insert into carreras(alias,nombre,creditos)
Values('isc','ingenieria en sistemas computacionales',110);
```

A continuación

- En la carpeta raíz del proyecto. Creé una carpeta llamada 'src'.
- Dentro de esta nueva carpeta creé el archivo: 'accesdb.js'
- En la carpeta 'src' copie la base de datos que creo anteriormente: (' escolar.sqlite')

A continuación, vamos a crear una clase para el manejo de la base de datos de sqlite:

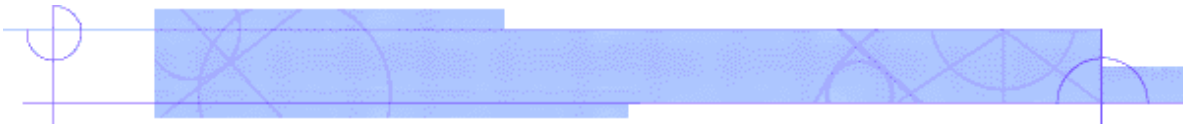
Archivo:accesdb.js

```
const sqlite3 = require('sqlite3').verbose()
class AppSqlite{
    constructor(dbFilePath){
        this.dbFilePath=dbFilePath
        this.dbOpen=false
        this.db=null
    }
    open(){
        return new Promise((resolve, reject)=>{
            this.db = new sqlite3.Database(this.dbFilePath, (err) => {
                if (err) {
                    console.log('No se pudo conectar a la database: ', err)
                    this.dbOpen=false
                    reject("La base de datos no se pudo abrir")

                } else {
                    console.log('Connectado a la database')
                    this.dbOpen=true
                    resolve(true)
                }
            })
        })
    }
    run(sql, params = []) {
        // console.log(sql,params)
        return new Promise((resolve, reject) => {
            this.db.run(sql, params, function (err) {
                if (err) {
                    console.log('Error corriendo  sql ' + sql)
                    console.log(err)
                    reject("en run ",err)
                } else {
                    //si el statemet se ejcuta satisfactoriamente
                    //el objeto this contiene 2 propiedades:
```

```
            //lastID contiene el inidce del ultimo registro insertado.
            //changes contiene el indice del ultimo renglon afectado por la consulta.
            //console.log("en run ",{ id: this.lastID, changes: this.changes })
            // console.log({ id: this.lastID, changes: this.changes })
            resolve({ id: this.lastID, changes: this.changes })
          }
        })
      })
    }
    get(sql, params = []) {
      return new Promise((resolve, reject) => {
        this.db.get(sql, params, (err, result) => {
          if (err) {
            console.log('Error running sql: ' + sql)
            console.log(err)
            reject(err)
          } else {
            //console.log(`consulta ${sql} correcta `,result)
            resolve(result)
            //resolve({ id: this.lastID, changes: this.changes })
          }
        })
      })
    }

    all(sql, params = []) {
      return new Promise((resolve, reject) => {
        this.db.all(sql, params, (err, rows) => {
          if (err) {
            console.log('Error running sql: ' + sql)
            console.log(err)
            reject(err)
          } else {
            resolve(rows)
          }
        })
      })
    }
    close(){
      console.log("base de datos cerrada")
      if(this.dbOpen) this.db.close()
    }
}

module.exports=AppSqlite
```

- Agregue este archivo (‘accesdb.js’) en su carpeta src

- Agregue el siguiente código a su archivo index.js:



```
const AppSqlite = require('./src/accesdb.js')

const dbm=new AppSqlite('./src/escolar.sqlite')

dbm.open()
.then(()=>{
   return dbm.all('select * from carreras')
})
.then((data)=>{
   console.log(data)
})
```

Ejecute la aplicación, verá la salida:

```
[
  {
    id: 1,
    alias: 'isc',
    nombre: 'ingenieria en sistemas computacionales',
    creditos: 110
  }
]
```

A continuación, uniremos nuestro servidor con la consulta a la base de datos:

Agregue la referencia a express en su proyecto:

- Ejecute en la terminal ***(npm install express)***
- Sustituya su código en su archivo index.js, por el siguiente:

```
const AppSqlite = require('./src/accesdb.js')
const express=require('express')

const dbm=new AppSqlite('./src/escolar.sqlite')
const app=express()

app.get("/",(req,res)=>{
   res.set({"content-type":"application/json; charset=utf-8"})
   dbm.open()
   .then(()=>{
      return dbm.all('select * from carreras')
   })
```

```
    .then((data)=>{
        console.log(data)
        res.status(200).send(data);
    })
    .catch((err)=>{
        res.status(400).send(err);
    })

})

app.listen(3000,()=>{
    console.log(`iniciando express desde: http://localhost:3000/`)
})
```

Referencias:

Node sqlite: https://www.npmjs.com/package/sqlite3

Express: https://expressjs.com/es/starter/installing.html

PRACTICA 9. Front end JS

Introducción

fetch

El método fetch() lanza el proceso de solicitud de un recurso de la red. Esto devuelve una promesa que resuelve al objeto Response que representa la respuesta a la solicitud realizada.

Resumen

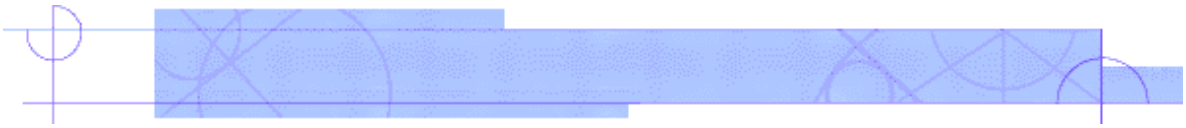
```
async function postData(url = "", data = {}) {
  // Default options are marked with *
  const response = await fetch(url, {
    method: "POST", // *GET, POST, PUT, DELETE, etc.
    mode: "cors", // no-cors, *cors, same-origin
    cache: "no-cache", // *default, no-cache, reload, force-cache, only-if-cached
    credentials: "same-origin", // include, *same-origin, omit
    headers: {
      "Content-Type": "application/json",
      // 'Content-Type': 'application/x-www-form-urlencoded',
    },
    redirect: "follow", // manual, *follow, error
    referrerPolicy: "no-referrer", // no-referrer, *no-referrer-when-downgrade, origin, origin-when-cross-origin, same-origin, strict-origin, strict-origin-when-cross-origin, unsafe-url
    body: JSON.stringify(data), // body data type must match "Content-Type" header
```

```
  });
  return response.json(); // parses JSON response into native JavaScript objects
}
postData("https://example.com/answer", { answer: 42 })
.then((data) => {
  console.log(data); // JSON data parsed by `data.json()` call
});
```

Ejemplos

```
//solicitar datos con get
const url="http://localhost:3000"
let request = new Request(url,
    {
        method: 'get',
        headers: { 'Content-Type': 'application/x-www-form-urlencoded' }
    });
    fetch(request)
    .then(function (returnedValue) {
        if (returnedValue.ok) {
            return returnedValue.json()
        } else {
            console.log("no se pudo recuperar los datos")
            return Promise.reject("no se pudo recuerar datos")
        }
    })
    .then((data)=>{
        //hacer algo con los datos
        console.log("Datos en formato json",data)
    })
```

```
//enviar datos de un formulario
const data = new FormData(document.getElementById('miformulario'));
fetch(url, {
    method: 'post',
    headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
    },
    body: new URLSearchParams(data)
})
.then(function (response) {
    if (response.ok) {
        return response.text()
    } else {
        console.log("no se pudieron actualizar los datos")
        return Promise.reject("no se pudo insertar el dato")
    }
})
.then(resp => {
    console.log(resp);
});
.catch(function (err) {
    console.error(err);
```



})
Enviar json ***//poner en el header*** ***headers: {*** ***"Content-Type": "application/json",*** ***}*** ***// y en el body*** ***body: JSON.stringify(Object.fromEntries(new FormData(document.getElementById('miformulario'))))***

10 Herramientas

10.1 Curl

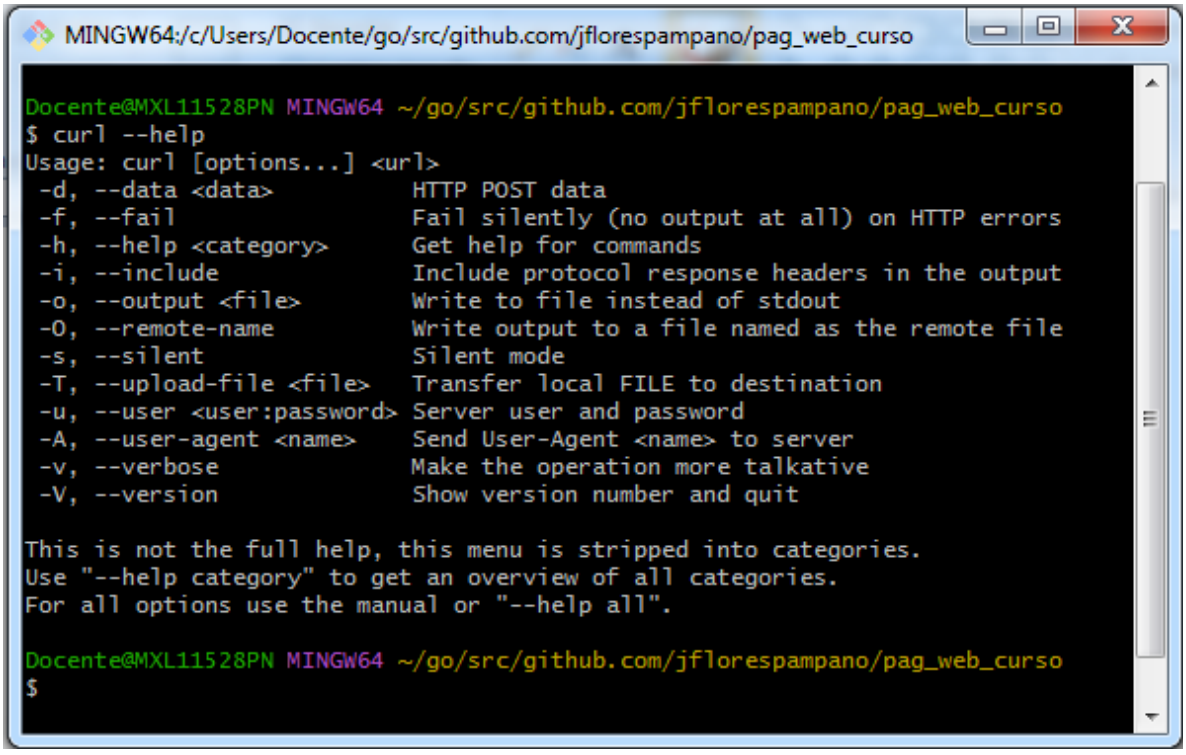
Resumen básico sentencia curl:
#estructura básica curl: ***curl -X POST [url]*** ***-H "Content-Type: [content type]"*** ***-d "[request data]"*** ***-v/-i***
Ejemplos GET
curl -X GET https://jsonplaceholder.typicode.com/posts #por default si no se pone -X GET se asume que es un get ***curl https://jsonplaceholder.typicode.com/posts*** #Al enviar parámetros, se debe poner \ para escapar el & en la consola de linux ***curl http://localhost:3000/?id=23\&nombre=juan perez\&correo=juan@mail\&edad=23***
Ejemplos POST Cuando pasa parámetros en curl con -d x default se usa la codificación: 'Content-Type: application/x-www-form-urlencoded' de no usar esa se debe especificar otra con -H
curl -X POST http://localhost:3000/user -d 'id=23&nombre=ana perez&correo=ana@mail&edad=32'
#Post enviando json
curl -X POST http://localhost:3000/user -H "Content-Type: Application/json" -d '{"id":"23","nombre":"ana perez","correo":"ana@mail","edad":32}'

Una API REST es básicamente el ***back end*** de nuestras aplicaciones, son funciones y protocolos que nos permiten realizar operaciones: *create*, *read*, *update*, *delete* (CRUD) a un almacén de datos y nos devuelve la información el algún formato, el más usado es JSON, por ejemplo:
Pruebe en su navegador las siguientes ligas:

- https://mdn.github.io/learning-area/javascript/oojs/json/superheroes.json
- https://catfact.ninja/docs/api-docs.json
- https://randomuser.me/api
- https://randomuser.me/api?inc=name,email,picture
- https://jsonplaceholder.typicode.com/users
- https://reqres.in/api/user

En cada una, vera en su navegador un conjunto de datos json, este es el funcionamiento de una API REST, usted envía solicitudes y recibe datos, para probar un ***api*** mientras la programamos, o para revisar su funcionamiento mientras implementamos un **front end** para ella, es bastante útil el comando **curl** de **Linux**, que puede usar desde la consola **Bash**.

Ejercicio 1:
Abra una ventana de **Bash**.
En la ventana ejecute el comando:
$curl --help



Vera una ayuda sobre el uso de curl.

Pruebe el comando:

$ curl https://catfact.ninja/docs/api-docs.json

Aquí con curl estamos haciendo una solicitud get de datos al sitio. Deberá ver un conjunto de datos en formato json que vienen desde la página: https://catfact.ninja/docs/api-docs.json.

Para el siguiente ejercicio probaremos enviar datos a una API, para esto usaremos el sitio: https://reqres.in/

Pruebe esto en curl:
$curl https://reqres.in/api/user

Pruebe también poner esa dirección: *https://reqres.in/api/user* en su navegador, se ven los mismos datos.

Ahora vera la ventaja de usar curl, escriba el comando:

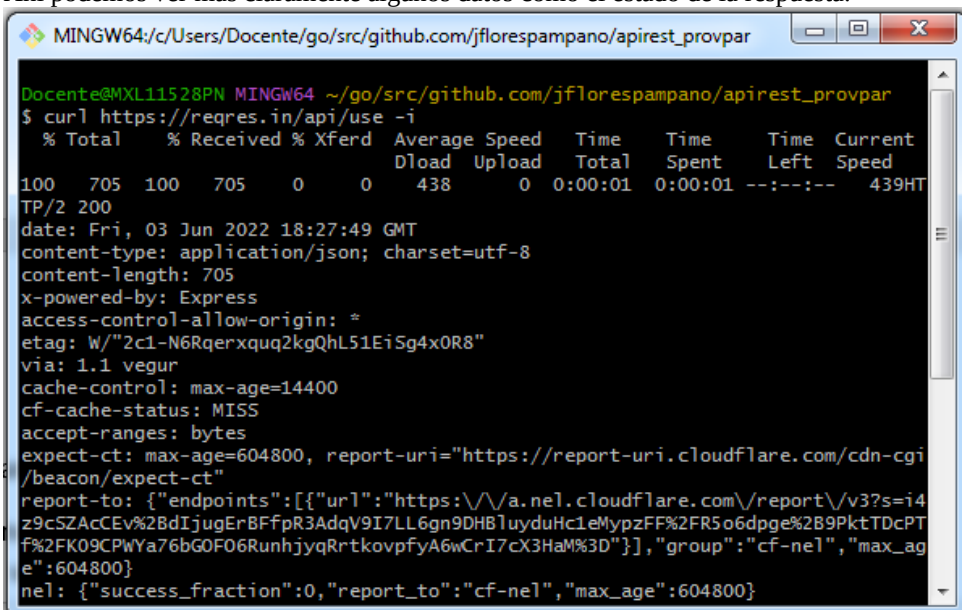
```
$curl https://reqres.in/api/user -v
```

Deberá ver ahora los mismos datos, pero con la cabecera HTTP enviada por el servidor.

Si le parece demasiada información, pruebe con el comando siguiente:

```
$curl https://reqres.in/api/user -i
```

Ahí podemos ver más claramente algunos datos como el estado de la respuesta:



La línea **http/2 200, 200:** indica que no hubo ningún problema. En otra línea, Puede ver que el contenido de la respuesta viene en formato:

content-type: application/json; charset=utf-8.

Si escribe el comando:

```
$curl https://jsonplaceholder.typicode.com/posts/1
```

Vera que esta liga nos envía un solo post.

Veamos ahora como solicitar un recurso y almacenarlo en un archivo:

```
curl -o datos.txt https://jsonplaceholder.typicode.com/posts/1
```

```
curl https://jsonplaceholder.typicode.com/posts/1 > datos.txt
```

Ambas líneas hacen lo mismo pero la segunda almacena el resultado en el archivo datos.txt.

```
$ curl -I https://jsonplaceholder.typicode.com/posts/1
```

Muestra el encabezado del sitio.
Ahora enviaremos datos desde curl:
Enviaremos un post al api ***https://jsonplaceholder.typicode.com/posts***:
Para insertar datos: nuestra página de ejemplo espera que enviemos *title* y *body*, el *userid* y el *id* lo genera la misma página.

Entonces en curl enviamos:
$curl --data "title=hola mundo&body=mi primer post" https://jsonplaceholder.typicode.com/posts

Si todo va bien nos devuelve:

```
Docente@MXL11528PN MINGW64 ~/go/src/github.com/jflorespampano/pag_web_curso
$ curl --data "title=hola mundo&body=mi primer post" https://jsonplaceholder.typ
icode.com/posts
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   104  100    68  100    36    145     77 --:--:-- --:--:-- --:--:--   223{
  "title": "hola mundo",
  "body": "mi primer post",
  "id": 101
}
```

Nos devuelve el mismo dato enviado que significa que todo estuvo bien.

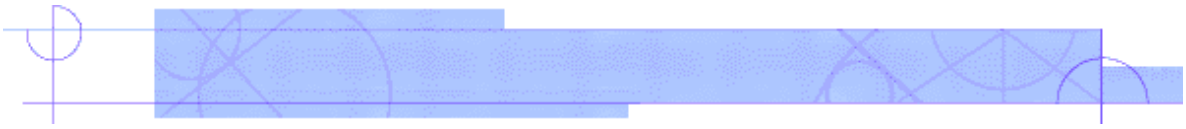
Enviar datos de formulario con curl:

Si tiene una aplicación que espera recibir datos desde un formulario en formato ***multipart/form-data***:
curl -X POST -F 'name=noviello' -F 'email=noviello@example.com' https://example.com/contact.php
Cuando se usa la opción -F curl envía los datos usando el tipo de contenido multipart/form-data
Otra forma de realizar una solicitud POST es utilizar la opción ***-d*** Esto hace que los datos se envíen con curl usando la opción: ***Content-Type application/x-www-form-urlencoded***.
curl -X POST -d 'name=noviello' -d 'email=noviello@example.com' https://example.com/contact.php

Una explicación más detallada la puede encontrar en el video:
https://www.youtube.com/watch?v=9u2qDQc6KWg&list=PLExyZ7hD-J5bigoqeZq1G_AfD4K4cWPn8&index=17
de donde se tomaron estos ejemplos.

Resumen comandos curl:

#solicitar recurso: ***$curl https://jsonplaceholder.typicode.com/posts/1***
#solicitar recurso y almacenarlo en un archivo: ***curl -o datos.txt https://jsonplaceholder.typicode.com/posts/1***
#solicitud con get , para poner el & en una consola de linux, necesitara anteponer la barra invertida (ok) ***curl http://localhost:8080/tipoparte/get?id=45\&nombre=juan\&edad=34***
#enviar archivo de datos json ***$curl -d "@data.json" -X POST http://misitio/post***
#enviar con encabezado ***$curl -d '{"title":"tit1","body":"body1","author":{"name":"juan","email":"juan@mail"}}' -H "Content-Type: application/json" -X POST http://misitio/post***

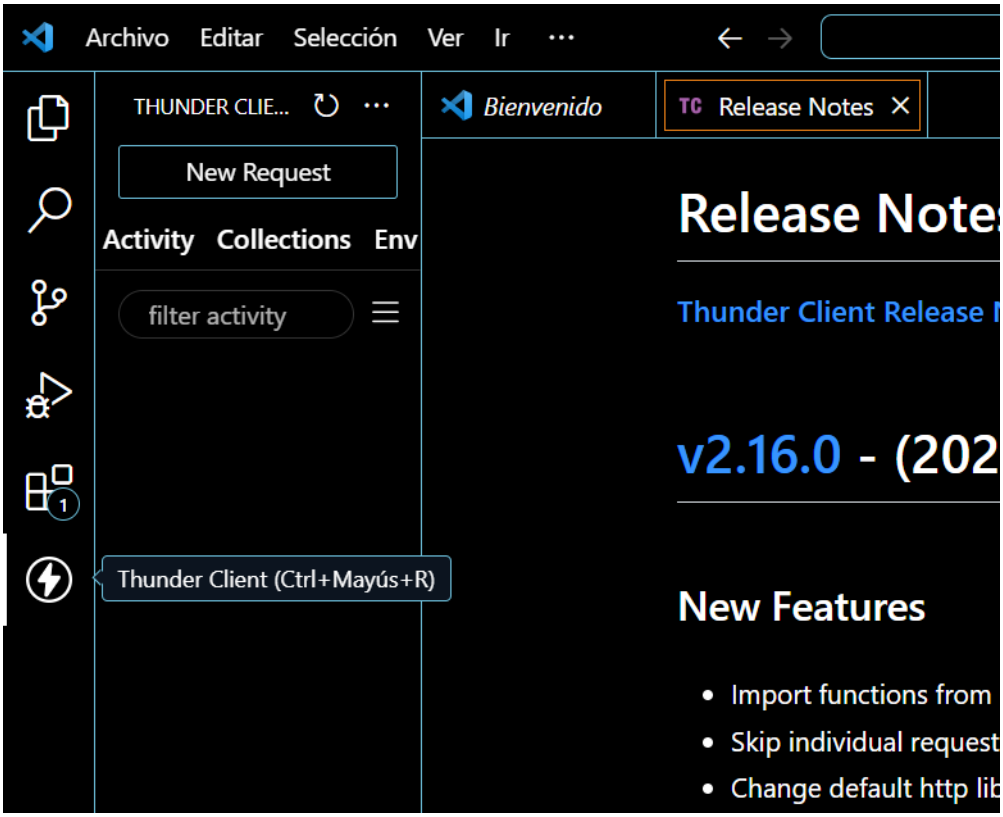


#enviar con encabezado *$ curl --header "Content-type: Application/json" -d '{"title":"titulo1", "body":"b1","userId":2}' https://jsonplaceholder.typicode.com/posts*
#probar ok *curl "http://misitio" -X POST -d '{"nombre":"juan"}' -H "Content-Type:application/json"*
#probar –F usa: multipart/form-data *curl -X POST -F 'name=noviello' -F 'email=noviello@example.com' https://example.com/contact.php*
probar –d usa: Content-Type application/x-www-form-urlencoded (ok) *curl -X POST -d 'name=noviello' -d 'email=noviello@example.com' https://example.com/contact.php*
#por default –d usa POST (ok) *$ curl -i -d "id=55" -d "nombre=juan" -d "carrera=isc" http://localhost/ejemplos/php/recibePeticion.php*
#por default –F usa post (ok) *$ curl -i -F "id=55" -F "nombre=juan" -F "carrera=isc" http://localhost/ejemplos/php/recibePeticion.php*
por ejemplo para un sitio que espera contenido en formato json: *$ curl -H "Content-Type:application/json" -d "@entrada.json" -X POST http://localhost:8080/tipoparte/crear* # y donde el archivo entrada.json contiene: { "tipo":"02", "descripcion":"material plomeria" }

10.2 Thunder

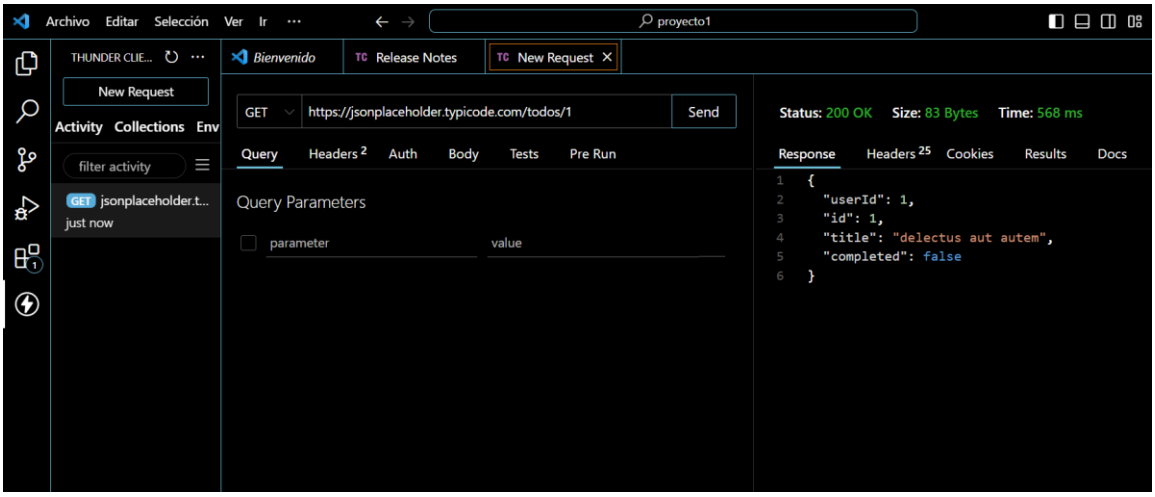
Thunder Client es una extensión de code, instálela.

Trabajar con **Thunder Client**
Abra VC
Fr clic en el ícono de Thunder Client:
Trabajar con **Thunder Client**:



De clic en el botón “new request”

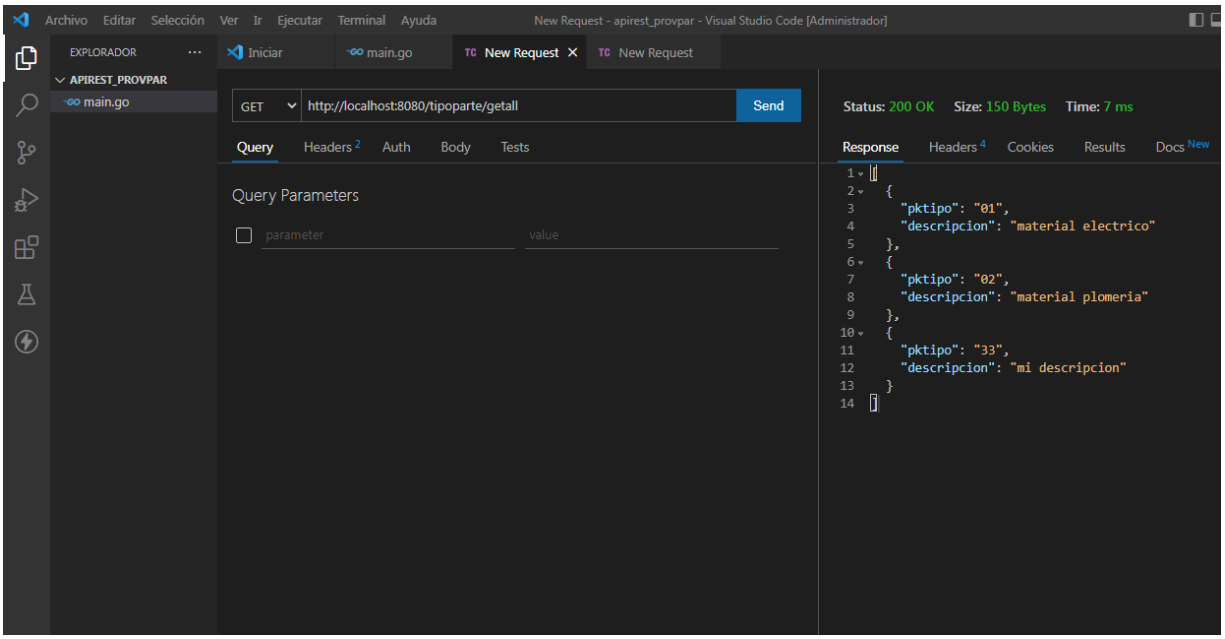
Seleccione la opción get y ponga esta dirección:



Trabajar con **Thunder Client** con una api propia:

Para este ejemplo, programe en node una api que acepta la ruta: /tipoparte/getall y que devuelve una lista de partes en formato json, con campos pktipo y descripción.

1. Ejecute su programa.
2. En su editor **code**,
3. Abra una ventana de **Thunder haciendo clic en la barra derecha del editor, en la parte de abajo vera un icono de un rayo, ese es Thunder**:

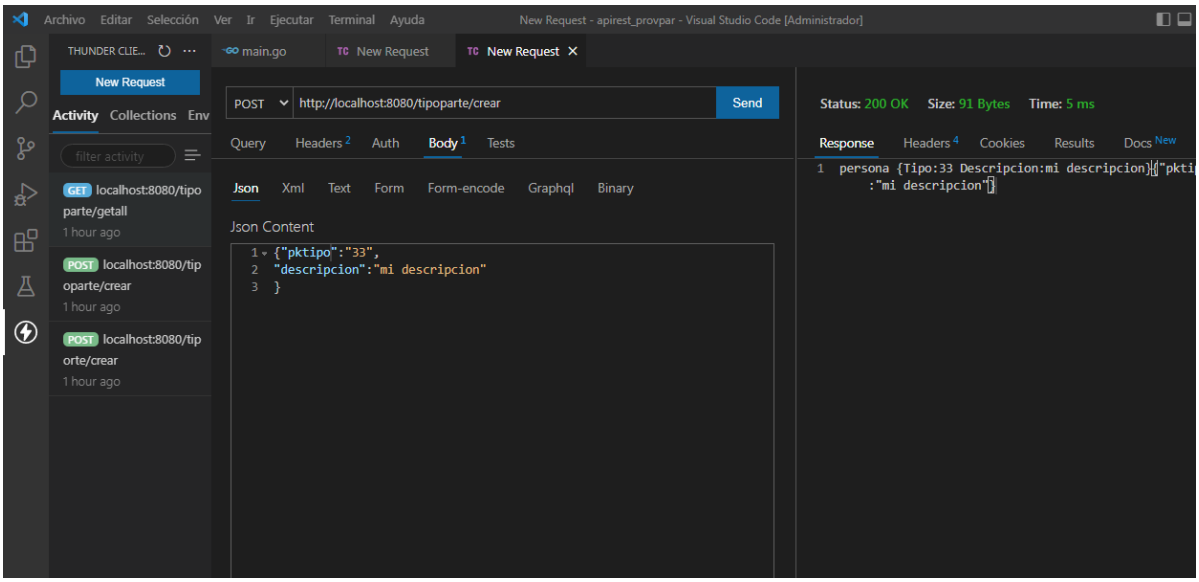


En **Thunder** haga una solicitud **get** a la dirección: **http://localhost:8080/tipoparte/getall**

Debe ver como respuesta los datos que tiene la variable **ListaTipoParte**, y que se ven en la ventana de arriba del lado derecho y el estado 200 ok que es la respuesta del servidor.

Para probar envío de datos mediante post con **Thunder**, haga:

1. Haga clic en el botón ***new request*** de la ventana de **thunder** para crear una nueva petición
2. Configure la petición así:



Seleccione la opción **POST**, ponga la petición: **http://localhost:8080/tipoparte/crear**

Seleccione la pestaña **body**

Dentro de **body** seleccione json

Dentro de la ventana json escriba los datos a enviar en formato json:

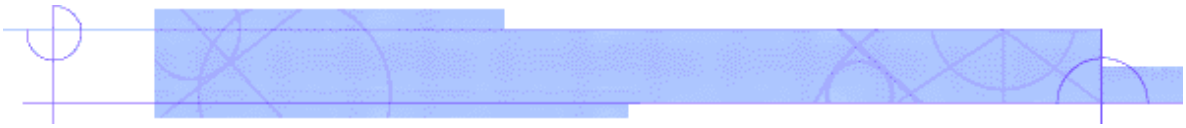
{
"pktipo":"33",
"descripcion":"mi descripcion"
}

Haga clic en el botón ***send***, deberá ver los datos de regreso y el código 200 del lado derecho como se ve en la imagen.

Solicitar un registro

Leer datos desde http con una llamada get:

#Agregue esta función a su código
func LisaPartesGet(w http.ResponseWriter, r *http.Request) { id := r.URL.Query().Get("id") nombre := r.URL.Query().Get("nombre") fmt.Fprintf(w, "desplegando dato...%s y %s", id, nombre) }
Pruébelo con Thunder lo puede probar también con el navegador: http://localhost:8080/tipoparte/get?id=45&nombre=juan # o también lo puede probar con curl: curl http://localhost:8080/tipoparte/get?id=45\&nombre=juan –i



Esta función solo recibo los datos de la llamada GET, agregue el código para devolver el dato.

10.3 Git

Git es un software de control de versiones diseñado por Linus Torvalds. Git tiene 3 áreas de trabajo:

- working directory //mi directorio de trabajo
- staging area //archivos preparados para commit
- git directory(repositorio) //archivos respaldados (commited)

```
git config --global user.name "mi nombre en git hub"
git config --global user.email "miemail@mail"
#muestra la configuración actual
git config --list
```

10.3.1 Configuración de git

Debe configurar git la primera vez con el usuario y el email, esta información la usa git en los commit.

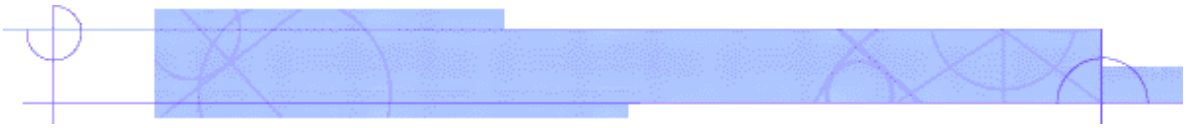
Si pone la opción --system esta configuración se usa para todos los usuarios del equipo, se almacena (en el caso de Windows) en la carpeta:(C:\msys64\etc).

Si pone la opcion *--global*, esta información se usará en todos los proyectos de su usuario y se almacena en $HOME, si desea ver donde esta esté directorio ejecute:
$echo $HOME
en bash.

Si desea sobrescribir esta información para un proyecto en específico, ejecute estos comandos en ese proyecto, sin la opción *--global* ni *--system*. Esta información se guarda en el archivo **.git/config**
Instale git desde: https://git-scm.com/download/win

10.3.2 Para trabajar con git

- Creamos una carpeta para nuestro proyecto
- Agregamos el proyecto (o puede iniciar con una carpeta vacía)



- En la carpeta damos clic derecho y elegimos (gitbash here), si esta opción no aparece, reinstale git, y en la opción (Windows explorer integration, seleccione Git Bash Here).
- Ahora podemos empezar a trabajar con nuestro proyecto.

Esta carpeta que creo es el workin directory, es una carpeta normal de Windows. Una vez que tenga un conjunto de archivos que quiera guardar, por ejemplo, por que ya programo hasta un punto en que tiene una funcionalidad ya probada y que no quiere perder, lo primero que puede hacer es:

10.3.3 Pasar sus archivos de proyecto al staging área.

Desde la consola de bash ejecute el comando:

$git add .

O puede enviar un solo archivo al staging

$git add archivo

Agrega todos o uno de sus archivos al staging área, este es un área de trabajo temporal, ahora puede seguir editando su proyecto y tendrá 2 versiones la de las nuevas modificaciones y la almacenada en el staging área.

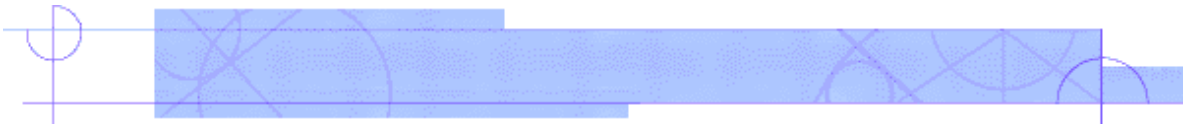
Suponga que un archivo que acaba de editar no le gusta porque perdió código en la edición o su nueva funcionalidad ya no la quiere, para recuperar un archivo desde el staging área al área de trabajo ejecute:

$git restore archivo

Solo hay un staging área, una vez que **todos** sus archivos están el staging área lo siguiente que puede hacer es tomar una snap shot (una instantánea, también decimos un commit), esto almacena una copia de su proyecto en el estado actual, puede sacar las instantáneas que requiera cada una tendrá una versión de su proyecto en el estado que se encontraba al sacar la instantánea.

Para ver el estado de su proyecto escriba:

$git status



Debe mostrar todo en verde lo que indica que todos los archivos de su proyecto están en el área temporal (staging), de no ser así, si aparece alguno en rojo debe agregarlo con: git add . o git add archivo.

Enseguida para sacar la instantánea escriba:

$git commit -m "letrero"

Esto crea la instantánea y la rotula con el letrero dado.

10.3.4 Recuperar un archivo de alguna instantánea (snap shot)

Primero debe mostrar todos los commit escribiendo:

$git log –oneline

Mostrará una lista de los commit con una clave y su letrero

Suponga que quiere recuperar el archivo texto.txt desde el commit con clave 55dfac4, para hacer esto escriba:

$git restore --source=55df4c2 prueba.txt

o Tambien puede hacer:

$git checkout 55df4c2 prueba.txt

10.3.5 Bajar repositorio desde github

```
#crear carpeta proyecto
#arrancar gitbash ahí
#
mkdir proyecto
cd proyecto
git init
#cambiar el nombre de la rama principal a main
git branch -m main
git remote add origin https://github.com...
#bajar el proyecto
```

```
git pull origin main
# si es nuestro y hacemos cambios
git add .
git commit -m "comentario"
git pull origin main #por si hubo algún cambio en el remoto
git push origin main #cargar mis cambios al remoto
```

11.- ANEXO 1 LENGUAJE DE CONSULTAS.

OBJETIVO DE LA PRÁCTICA. 11.

Diseñar los movimientos de consultas, actualizaciones e inserción a la base de datos.

1.- INTRODUCCIÓN.

1.- SENTENCIA CREATE TABLE.

CREATE TABLE table_name (

column1 datatype,

column2 datatype,

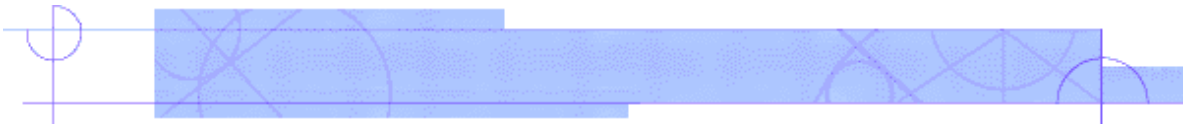
column3 datatype,

....

);

EJEMPLO:

```
CREATE TABLE Persons (
    id int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
```



```
    Primary key(id)
);
```

2.- Sentencia Alter Table.

3.- SENTENCIA SELECT.

Select [lista de campos | *] from tabla;

Devuelve una tabla que contiene una lista de registros desde una tabla dada, ejemplos:

Suponga la tabla:

Tabla: Costo	
Ciudad	Costo
Frontera	50
Sabancuy	60
Champoton	100
Frontera	50
Campeche	145
Puebla	800
DF	1200
Mérida	250

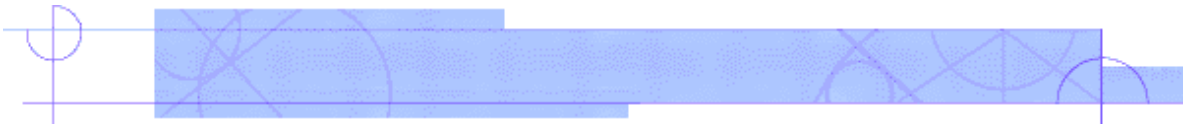
A continuación, se dan ejemplos de consultas y su resultado:

Todos los datos de la tabla costo

```
Select * from costo;
Ciudad   Costo
----------  --------
Frontera        50
Sabancuy        60
Champoton       100
Frontera        50
Campeche        145
Puebla  800
DF      1200
Mérida  250
```

Todos los nombres de las ciudades

```
Select ciudad from costo;
Ciudad
----------
Frontera
```



Sabancuy
Champoton
Frontera
Campeche
Puebla
DF
Mérida

Los datos de la tabla costo para las ciudades con costo mayor que 200

```
Select * from costo where costo>200;
Ciudad   Costo
---------- --------
Puebla   800
DF       1200
Mérida   250
```

4.- SENTENCIA INSERT.

Suponga que queremos insertar un registro en la tabla:

Tabla: Alumnos	
Matricula	Nombre
110967	Juan Puc
118967	Rosa Chi

Insert into alumnos (matricula, nombre) values ('125634','Erika Uc');

Nota*:* Debe tomar en cuenta que los valores de tipo texto como (varchar, char) deben ir entre apostrofes, los valores numéricos deben ir sin apostrofes, las fechas deben ir entre apóstrofes.

Tabla: Alumnos	
Matricula	Nombre
110967	Juan Puc
118967	Rosa Chi
125634	Erika Uc

5.- SENTENCIA UPDATE.

Suponga que queremos modificar los datos de la Tuerca de ½ porque su peso debe ser 12 y su costo 15.

Tabla Partes			
clave	Parte	peso	costo
110967	Tornillo ¾	6	1
118967	Tuerca ½	9	4
129087	Arandela ¾	4	3

Update Partes set peso=12, costo=15 where clave='118967';

Nota: Debe tomar en cuenta que los valores de tipo texto como (varchar, char) deben ir entre apostrofes, los valores numéricos deben ir sin apostrofes, las fechas deben ir entre apóstrofes.

6.- SENTENCIA LIKE.

Suponga la tabla:

Tabla: Alumnos	
Matricula	Nombre
110967	Juan Puc Alba
118967	Rosa Chi Cobos
125634	Erika Uc Paz
122534	Ross Díaz Montt
102354	Juan Ríos Perez
114523	José Juan Morales Rojas

La siguiente sentencia muestra todos los registros cuyo nombre empiece con 'Juan';

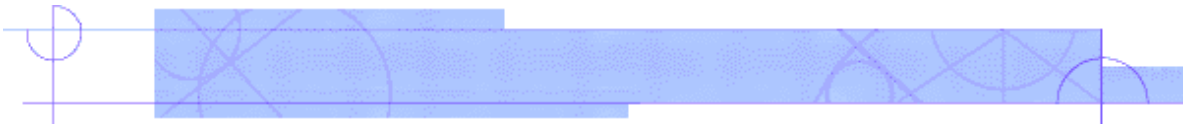
```
Select * from Alumnos where Nombre like 'Juan%';
Matricula Nombre
------------ --------
110967     Juan Puc Alba
102354     Juan Ríos Perez
```

Muestra todos los registros que contengan la palabra 'Juan' en el nombre:

```
Select * from Alumnos where Nombre like '%Juan%';
Matricula Nombre
------------ --------
110967     Juan Puc Alba
102354     Juan Ríos Perez
114523     José Juan Morales Rojas
```

MATERIALES DE LA PRÁCTICA. 11.

- Computadora.



- Internet
- Suite de oficina

DESARROLLO DE LA PRÁCTICA. 11.

Para la narrativa de proveedores y partes se forma un equipo de trabajo donde hay desarrolladores de la interfaz y fue diseñada ya la base de datos, usted es el encargado de diseñar las funciones PHP para hacer los movimientos a la base de datos, para lo que las funciones reciben datos en formato JSON los datos a insertar o actualizar por ejemplo y para las consultas devuelvan arreglos JSON con el resultado de las consultas. Apóyese en el material de introducción a PHP que proporcionó el profesor. Y las operaciones que se desean manejar son:

- Altas/ bajas/ modificaciones de partes.
- Altas/ bajas/ modificaciones de proveedores.
- Altas/ bajas/ modificaciones de tiposPartes.
- Altas/ bajas/ modificaciones de pedidos.
- Lista de partes.
- Lista de partes ordenada por tipo de parte junto con el nombre del tipo de la parte.
- Lista de partes con stock menor o igual al mínimo.
- Lista de pedidos de las partes para un tipo de parte dado.
- Lista de pedidos de las partes con un stock menor al mínimo.

ENTREGABLE DE LA PRÁCTICA. 11.

- Archivo pdf con el código php con sus respectivas sentencias sql.

FUENTES CONSULTADAS

[Joyanes1998] Joyanes Aguilar, Luis y Zahoneri Martínez, Ignacio, (1998) Estructura de datos. Algoritmos, abstracción y objetos. Ed. McGrawHill, España.

[Joyanes2003] Joyanes Aguilar, Luis (2003) Fundamentos de la programación algoritmos y estructuras de datos. Tercera edición, Ed. McGrawHill. España.

[Oviedo2003] Oviedo Regino, Efrain (2003) Lógica de programación. Ed. Ecoediciones. España.

[Tenembaum1996] Tenembaum, Aaron M. Tenembaum, Aarón M., Langsam, Yedidhah y Augenstein, Moshe A., (1996) Estructuras de datos con C y C++. Segunda Edición – Prentice hall, México.

[Goodrich2002] Goodrich, Michael T., Tamassia, Robert, (2002). Estructura de datos y algoritmos en Java. Segunda Edición – ed. CECSA, México.

[www3schools/php] https://www.w3schools.com/php/DEFAULT.asp

[mozila/html] https://developer.mozilla.org/es/docs/Web/HTML

[mozila/js] https://developer.mozilla.org/es/docs/Web/JavaScript

[www3schools/bootstrap] https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp

[php] https://www.php.net/manual/es/intro-whatis.php