



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”**

**DESARROLLADO Y APROBADO POR LA ACADEMIA DE
PROGRAMACIÓN**

noviembre 2023

Versión 1.0



ELABORADO POR

MI. Jesús Alejandro Flores Hernández

MCC. Damaris Pérez Cruz

MCC. José Ángel Pérez Rejón

MCC. Juan Carlos Canto Rodríguez



INDICE DE TEMAS Y PRÁCTICAS

Tema del curso	Práctica	Pag.
Plataforma de trabajo	Práctica I	4
HTML	Práctica II	10
CSS	Práctica III	21
W3CSS	Práctica IV	35
JS	Práctica V	43
MySql/SQLite	Practica VI	73
Node Js	Practica VII	80



Contenido

INDICE DE TEMAS Y PRÁCTICAS	2
INTRODUCCIÓN	6
PRACTICA 1. Plataforma de trabajo	6
1.1. Objetivo	6
1.2. Equipo	6
1.3. Materiales	6
1.4. Descripción.....	6
1.4.1 Editor	7
1.4.2 Gestor de base de datos Xamp/Wamp.....	7
1.4.3 Base de datos SQLite.....	7
1.5. Procedimiento de la práctica	8
1.6. Práctica.....	8
PRACTICA 2. Base de datos.....	8
2.1. Objetivo	8
2.2. Equipo	8
2.3. Materiales	8
2.4. Descripción.....	8
2.4.1 MySql	8
2.4.2 SQLite	14
2.4.3 SQLiteAdministrator	22
2.5. Procedimiento de la práctica.....	25
2.6. Prácticas.....	25
PRACTICA 3. HTML.....	25
3.1. Objetivo.....	26
3.2. Equipo	26
3.3. Materiales	26
3.4. Descripción.....	26
3.4.1 Tags HTML.....	26
3.4.2 Encabezado	27
3.4.3 Cuerpo <body>.....	28
3.4.4 Atributos	29
3.4.5 Agrupadores	31
3.4.6 <form>	32
3.5. Procedimiento.....	37
3.6. Prácticas.....	37
PRACTICA 4. CSS	37
4.1. Objetivo	37
4.2. Equipo	37
4.3. Materiales	38
4.4. Descripción.....	38
4.4.1 Sintaxis CSS	38
4.4.2 Almacenamiento de reglas	40
4.4.3 Definiendo reglas	40

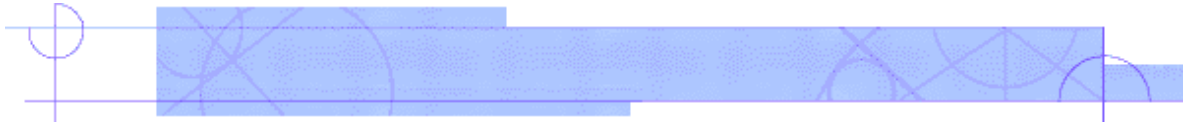


4.4.4 Selectores	41
3.5. Procedimiento.....	51
3.6. Prácticas.....	51
PRACTICA 5. W3CSS	51
5.1. Objetivo.....	51
5.2. Equipo	51
5.3. Materiales	52
5.4. Descripción.....	52
5.5. Procedimiento.....	54
5.6. Prácticas.....	54
PRACTICA 6. JS	54
6.1. Objetivo.....	54
6.2. Equipo	54
6.3. Materiales	54
6.4. Descripción.....	55
Introducción:	55
1 Java Script y HTML.	55
2 Donde codificar el JS.....	56
3 Entrada Salida.....	57
3 Entrada JS.....	59
4 Variables, tipos y expresiones.....	59
5 Funciones.....	62
6 Eventos.	64
7 Objetos (opcional).	70
8 prototipos de objetos (opcional).	71
8.1 Creando Objetos	71
8.2 Agregar propiedades y métodos a objetos.	73
8.3 Agregando propiedades y métodos al prototipo	74
9 Formato JSON	75
10 Recorre JSON	76
11 Acceso al DOM	80
V.5. Procedimiento.....	84
V.6. Prácticas.....	84
PRACTICA 7. Node Js	85
VII.1. Objetivo	85
VII.2. Equipo	85
VII.3. Materiales.....	85
VII.4. Descripción	85
1 Introducción.....	85
2 Bases de datos.....	86
3 Conexión a la base de datos.	89
4 JS-fetch	94
5 Insertar registro con fetch.....	97
6 Recibir datos con fetch desde php	100
7 Curl/ThunderClient	103

Universidad Autónoma del Carmen



7.1 Curl.....	103
7.2 Thunder	107
FUENTES CONSULTADAS	108



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

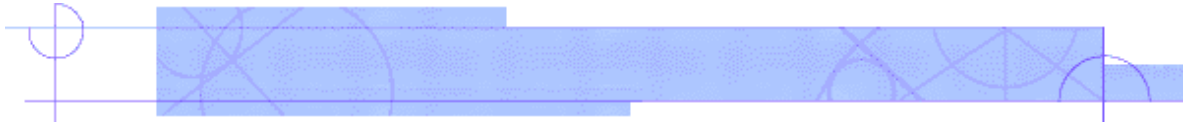
Para gestionar nuestras base de datos de MySQL usaremos XAMP/WAMP. WAMP es un acrónimo que significa Windows, Apache, MySQL y PHP. Es un stack o conjunto de soluciones de software que significa que cuando instalas WAMP, estás instalando Apache, MySQL y PHP en tu sistema operativo (Windows en el caso 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 usará 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 y útil para aplicaciones con pocos datos.



Un administrador para SQLite lo puede descargar desde:

<https://sqliteadmin.orbmu2k.de/>

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 XAMP en su computadora.
2. Instale Heidi Sql en su computadora desde: (<https://www.heidisql.com/>) como administrador de Mysql.
3. Descargue/instale SQLiteAdmin.
4. 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 en MySQL.

2.2. Equipo

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

2.3. Materiales

Internet, Computadora, Manuales.

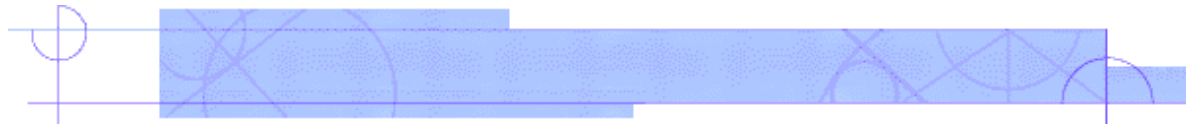
2.4. Descripción

2.4.1 MySQL

Asumiendo que tiene instalado XAMP server, arranque el servidor de MySQL.

HeidiSQL es un software libre y de código abierto que permite conectarse a servidores MySQL, MariaDB, Microsoft SQL Server y PostgreSQL.

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

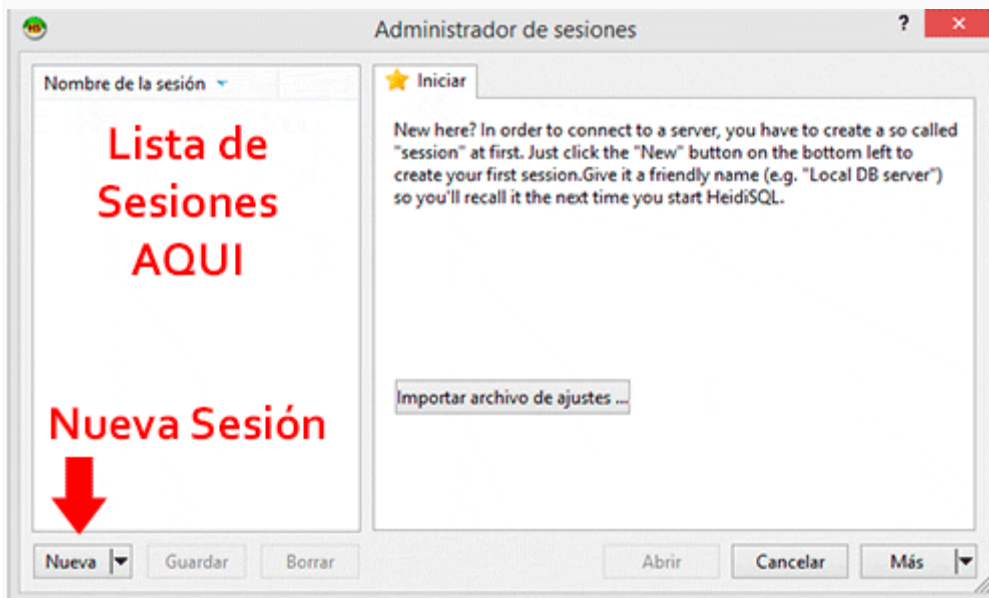


Iniciar MySQL

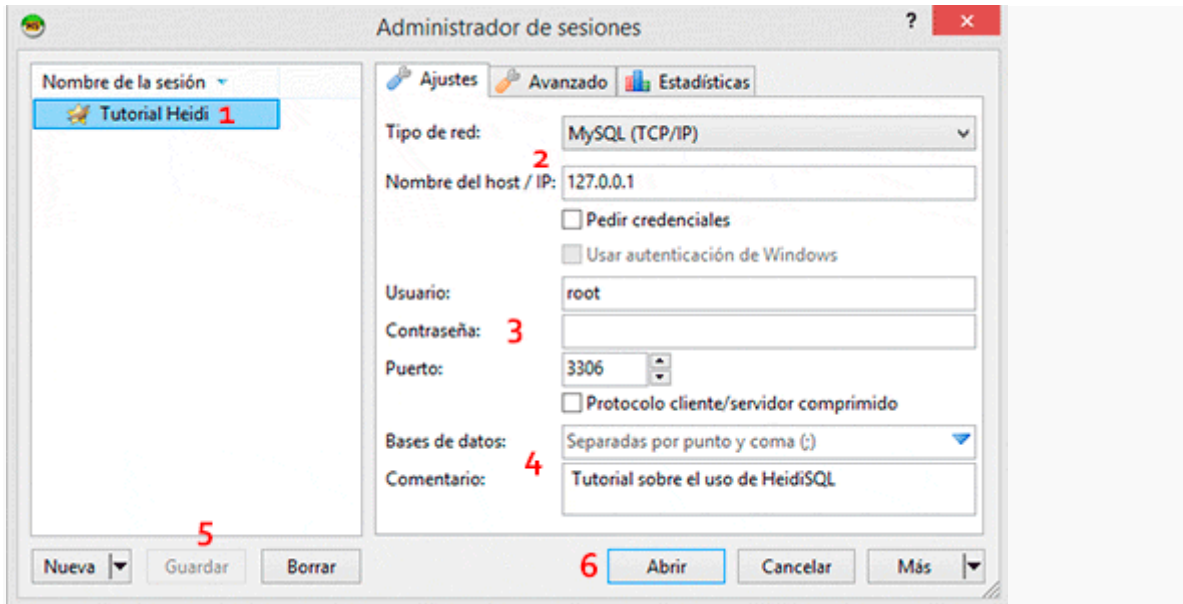
Si no tiene iniciado MySQL, inícielo.

Sesión en Heidi

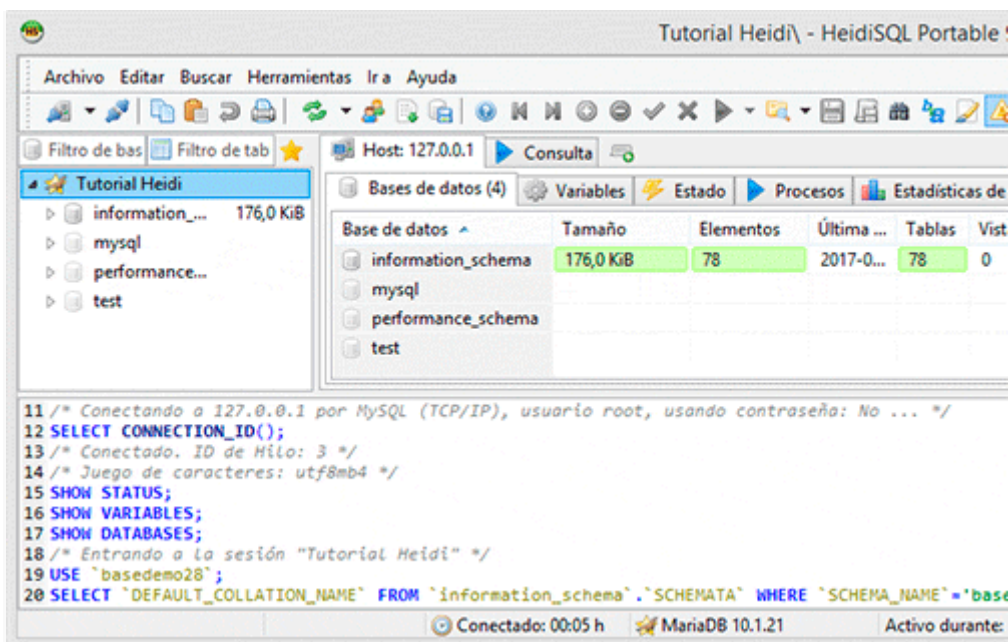
Arranque el HeidiSql



HeidiSQL 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**».



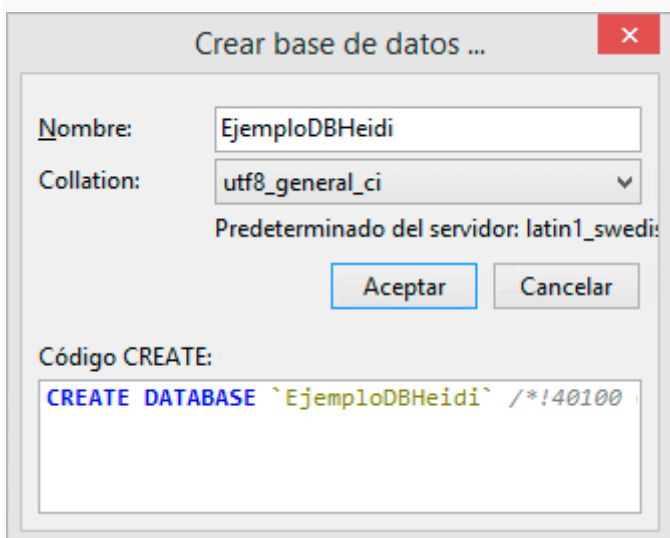
1. Escribimos el nombre de la sesión, por ejemplo «**Tutorial Heidi**»
2. Seleccionamos el Tipo de Red (MySQL TCP/IP) y el nombre del Host lo dejamos en: 127.0.0.1
3. Escribimos el nombre de usuario (*root por defecto en MariaDB*) y contraseña (*si tuviera*) en blanco si no tiene. Podemos indicar también 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*»



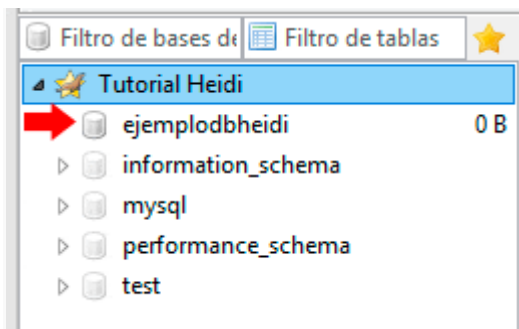
La interfaz de Heidi es intuitiva y además esta en español.

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).



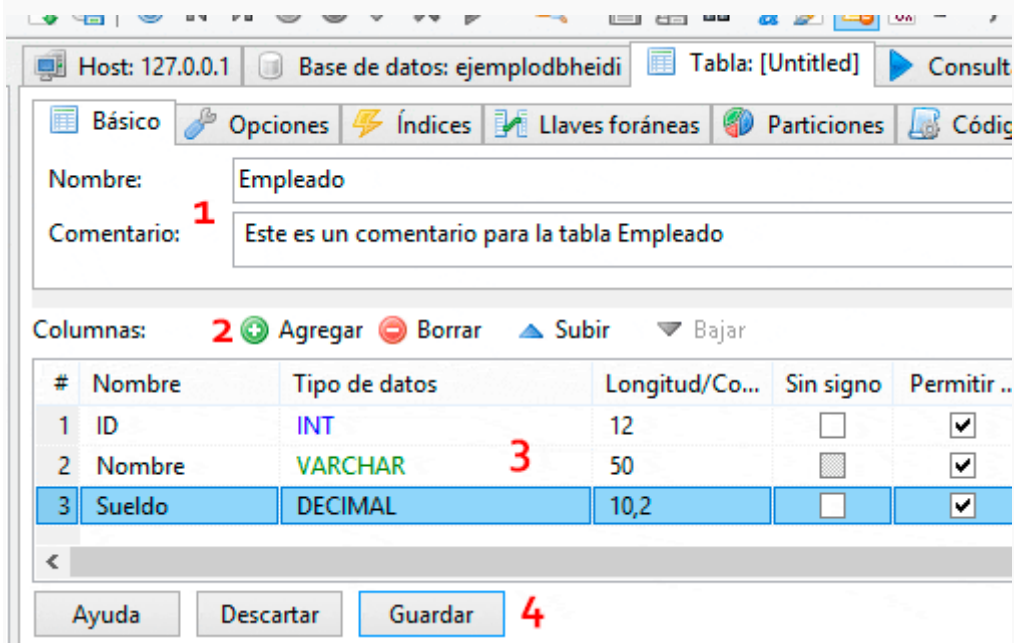
Para terminar la creación, presionamos «*Aceptar*»



Crear Tabla

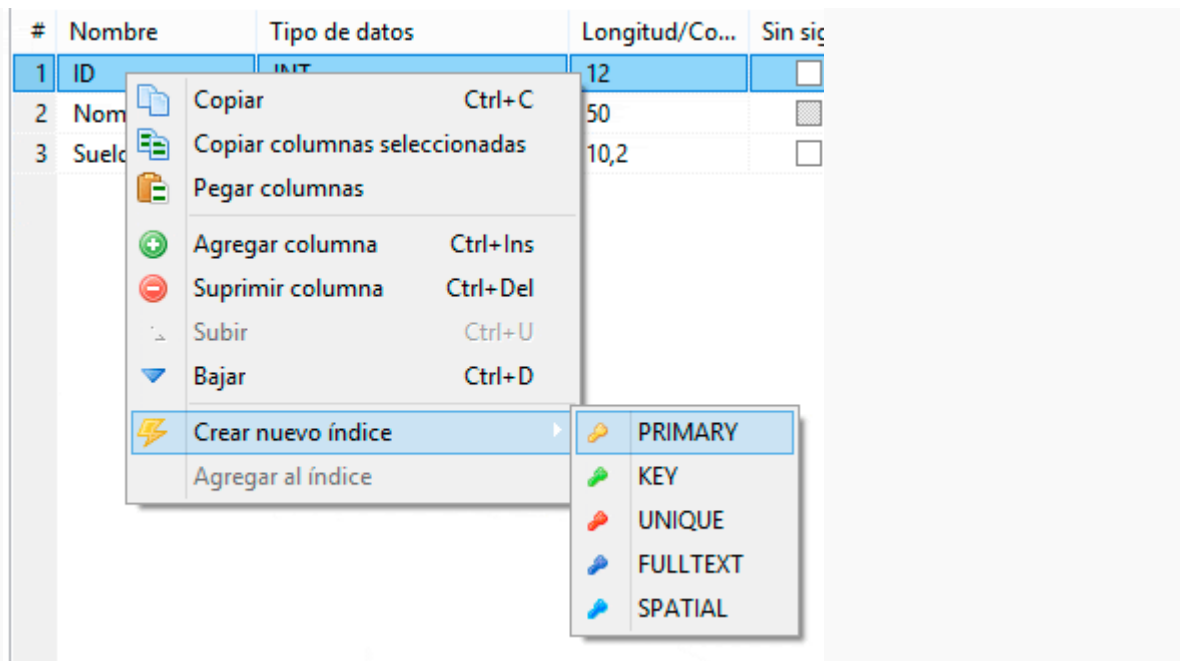
Clic derecho sobre la base de datos donde se quiere crear la tabla, seleccionar «Crear nuevo» → «Tabla».

- 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»

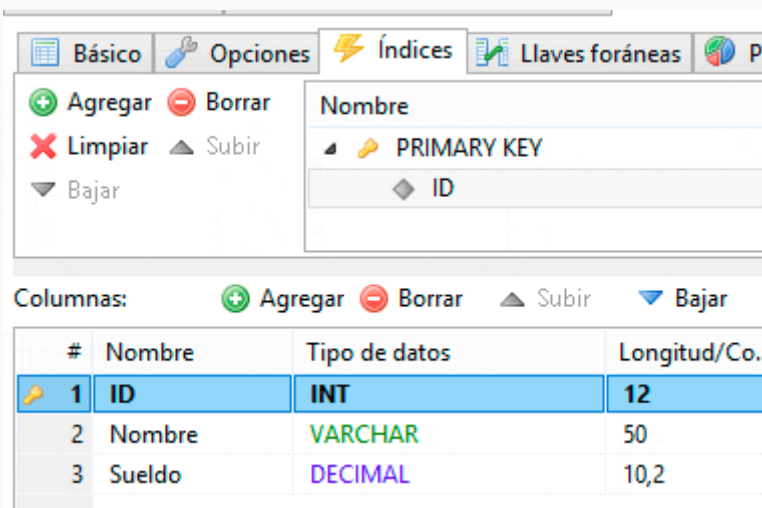


Llave primaria

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



El campo será marcado con una llave para indicar que corresponde a una **PRIMARY KEY**, además en la pestaña «Índices» 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 los botones (+) y (-)
 B: Para agregar datos a cada campo, doble clic y escribir su contenido



Host: 127.0.0.1 Base de datos: ejemplodbheidi

ejemplodbheidi.empleado: 3 filas en total (aproximadamen

ID	Nombre	Sueldo
1	Aquiles Brinco	5.000,00
2	Soytu Burla	7.500,50
3	B	(NULL)

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

Host: 127.0.0.1 Base de datos: ejemplodbheidi Tabla: empleado Consulta*

```
1 Select * from empleado where sueldo > 4000
```

empleado (3x2)

ID	Nombre	Sueldo
1	Aquiles Brinco	5.000,00
2	Soytu Burla	7.500,00

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 para VS code.

Universidad Autónoma del Carmen



- Usando un complemento para Firefox.
- Usando el shell sqlite3.

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.

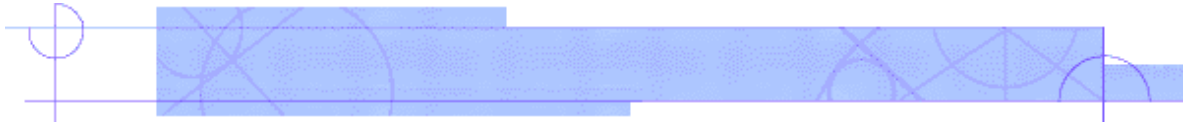
Ejecute la aplicación **sqlite3.exe** verá:

```
C:\trabajo\sqlite\sqlite-tools-win32-x86-3410000\sqlite-tools-win32-x86-3410000\sqlite3.exe
SQLite version 3.41.0 2023-02-21 18:09:37
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

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



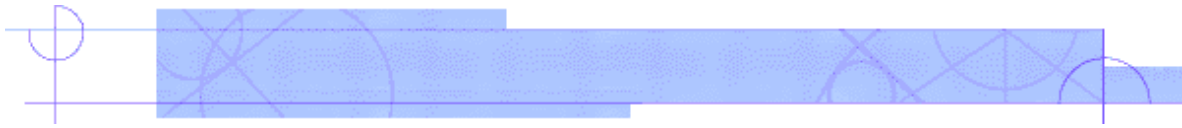
```
//mostrar estructura de tabla
sqlite >.schema partes
//crear una tabla
sqlite>sqlite> CREATE TABLE tbl2 (
...>  f1 varchar(30) primary key,
...>  f2 text,
...>  f3 real
...> );
sqlite>
ejecutar sentencia sql
sqlite>select * from partes;
//opcionalmente antes de ejecutar sus select puede activar
//un 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 'provp-ar.s3db' hacemos:



```
C:\trabajo2023\sqlite-tools-win32\sqlite3.exe
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> .open c:/trabajo2023/prov-par.s3db
sqlite> _
```

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

```
C:\trabajo2023\sqlite-tools-win32\sqlite3.exe
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> .cd c:/trabajo2023
sqlite> .open prov-par.s3db
sqlite> _
```

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

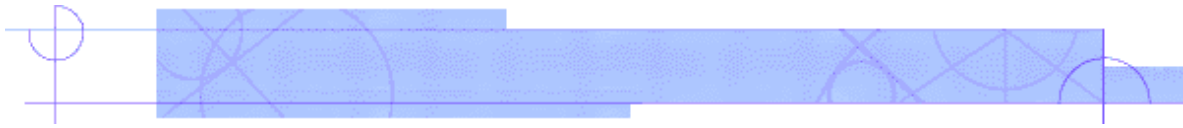


```
C:\trabajo2023\sqlite-tools-win32\sqlite3.exe
SQLite version 3.41.2 2023-03-22 11:56:21
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open c:/trabajo2023/prov-par.s3db
sqlite> .tables
parte
sqlite> .schema parte
CREATE TABLE parte(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    anio INTEGER NOT NULL,
    nombre TEXT NOT NULL,
    costo REAL NOT NULL
);
sqlite>
```

Ejemplo salida a 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



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

```
sqlite> .output salida.json
sqlite> select * from parte;
sqlite> .output
sqlite> select * from parte;
[{"id":1,"anio":34,"nombre":"juan Perez","costo":22.999999999999999},
{"id":2,"anio":44,"nombre":"33","costo":55.0},
{"id":3,"anio":2022,"nombre":"tuerca 1/2","costo":10.0},
{"id":4,"anio":2022,"nombre":"tuerca union 1/2","costo":12.0},
{"id":5,"anio":2022,"nombre":"tuerca 1/2","costo":10.0},
{"id":6,"anio":2022,"nombre":"tornillo 3/4","costo":15.0},
{"id":7,"anio":2022,"nombre":"arandela 1/2","costo":9.0},
{"id":8,"anio":2022,"nombre":"pija 3/8","costo":8.0},
{"id":9,"anio":2022,"nombre":"tuerca union 1/2","costo":12.0},
{"id":10,"anio":2020,"nombre":"llave allen #4","costo":45.0}]
sqlite> _
```

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 diseñó 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, credits)
values
(
  'ico',
  'ingenieria en computación',
  100
);

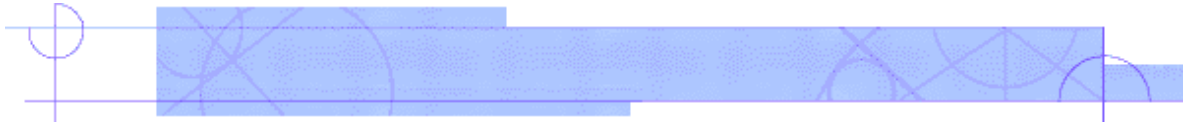
insert into
  carreras(alias, nombre, credits)
values
(
  'isc',
  'ingenieria en sistemas computacionales',
  110
);

insert into
  carreras(alias, nombre, credits)
values
(
  'itcc',
  'ingenieria en tcnologias 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

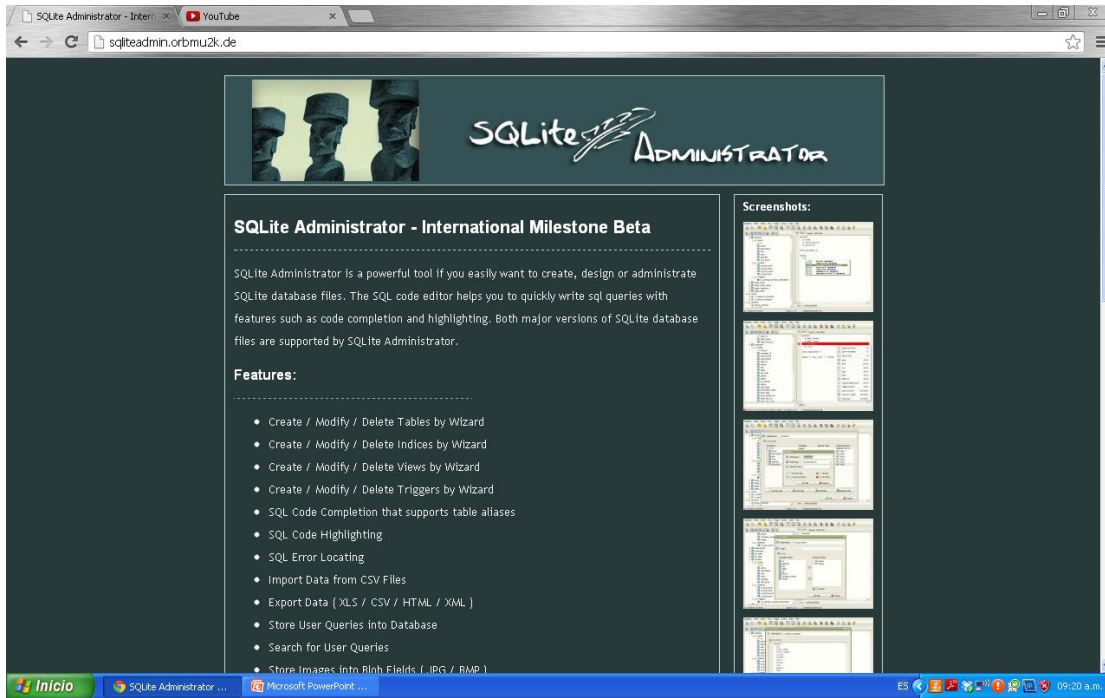
```
.mode list
select * from carreras;
select * from alumnos;
```

Pantalla de ejecución del shell:

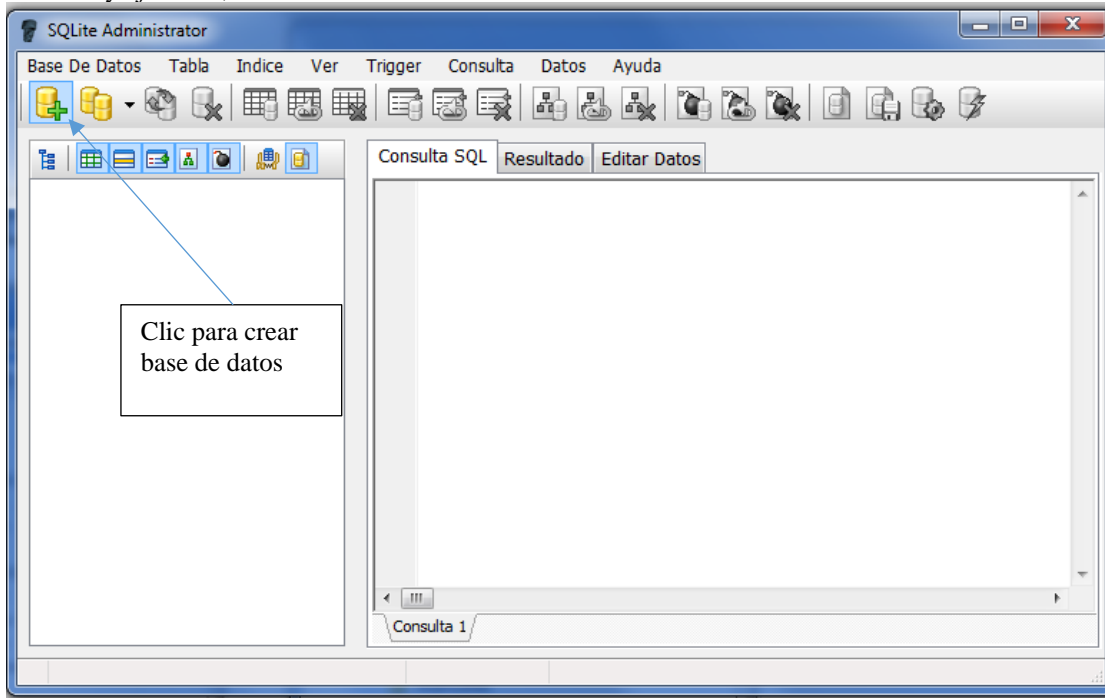
```
SQLite version 3.41.2 2023-03-22 11:56:21
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .read alumnos.sql
sqlite> .mode list
sqlite> select * from carreras;
1|ico|ingeniería en computación|100
2|isc|ingeniería en sistemas computacionales|110
3|itcc|ingeniería en tecnologías de cómputo 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

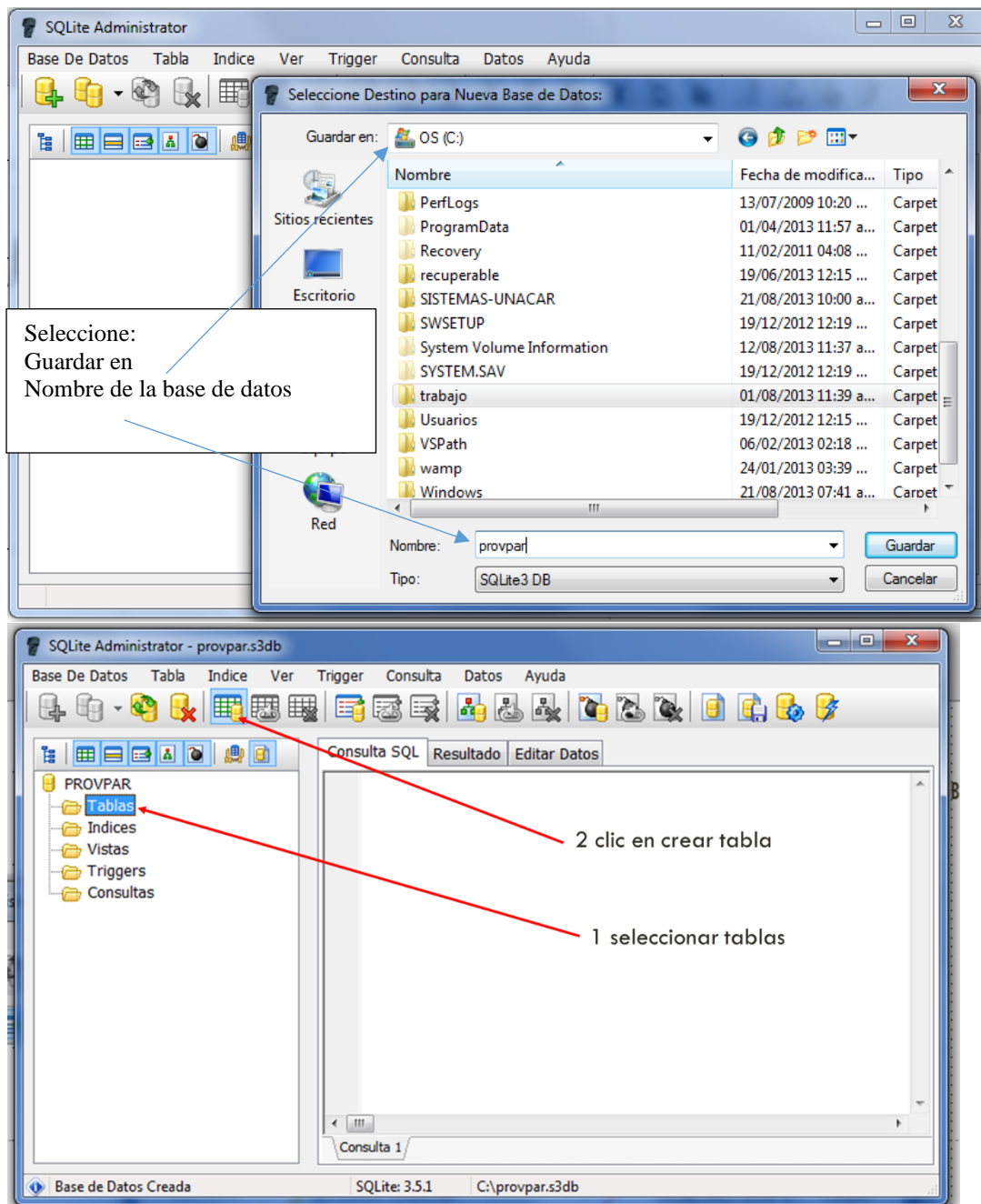
Aunque es útil, puede ser frustrante trabajar con línea de comandos, como solución usaremos SQLite-Admin, es una aplicación visual para crear y manipular base de datos de SQLite. Lo puede descargar desde: <http://sqliteadmin.orbmu2k.de/>

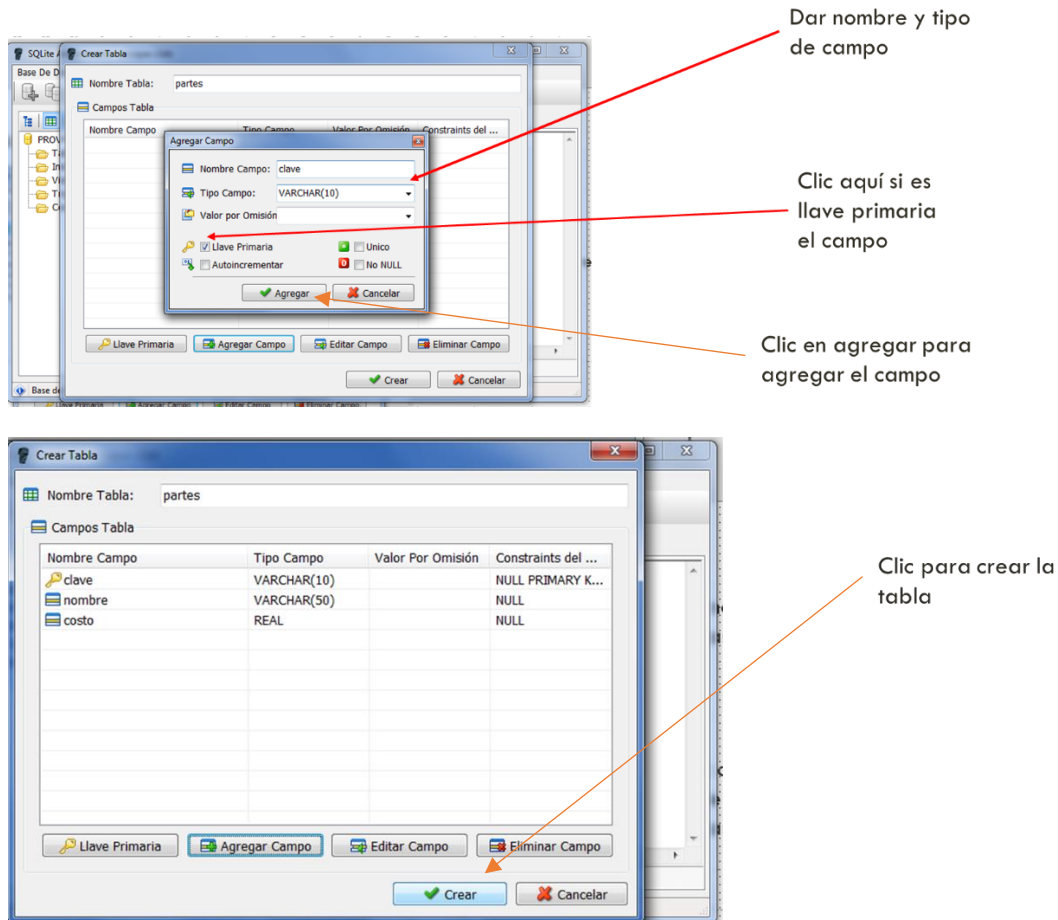


Instalelo y ejecutelo, vera la ventana:



Universidad Autónoma del Carmen





2.5. Procedimiento de la práctica

1. Lea la introducción de este tema y Con una base de datos que diseño en la práctica 1 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 diseño en la práctica 1.
3. Valide la base de datos y las consultas con el profesor.

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>, <div>, , , <aside>, <audio>, <canvas>, <datalist>, <details>, <embed>, <nav>, <output>, <progress>, <video>, , , , 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:

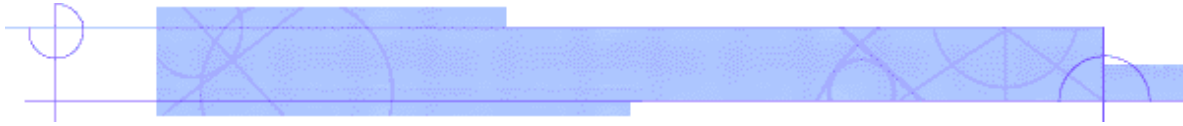
Archivo HTML: titulo.html

```
<!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 ñ ñ Ñ Ñ Ñ</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, recomendando 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 seleccione 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 **<p></p>**:

Universidad Autónoma del Carmen



`<p> mi parrafo </p>`

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:

`` y `` 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
<pre><!DOCTYPE html> <html> <body> Esta es mi liga </body> </html></pre>

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 único 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 <div>

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>
```

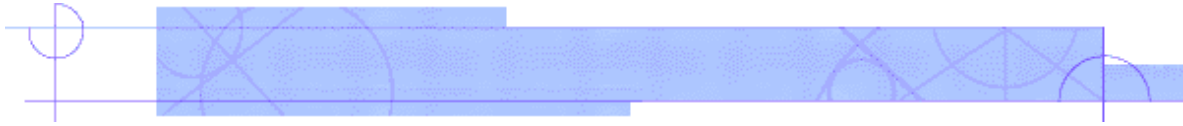
```
<LI> Un elemento
```

```
<LI> Otro elemento
```

```
<LI> Otro más
```

```
<LI> etc.
```

```
</UL>
```



Es decir, toda la lista está dentro de la etiqueta `` y ``, y luego cada elemento va precedido de la etiqueta `` (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



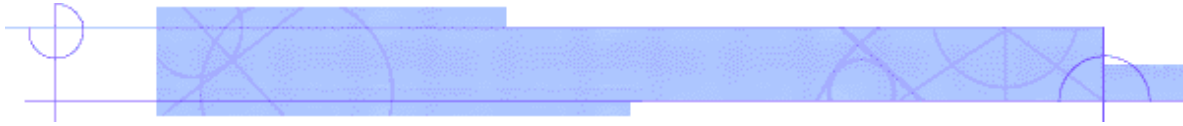
Vamos a ver ordenadamente (de fuera hacia dentro) las etiquetas necesarias para confeccionar las tablas.

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

Tabla 1.2-3

Ejemplo:

Tabla en HTML
<pre> <!DOCTYPE html> <html> <body> <table> <thead> <tr> <th>Month</th> <th>Savings</th> </tr> </thead> <tfoot> <tr> </pre>



```
<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 ahora 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 debrá agregar funcionalidad ni estilos.

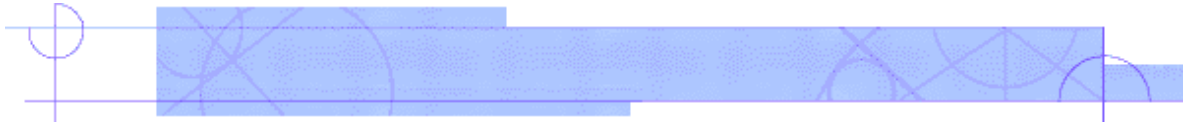
PRACTICA 4. CSS

4.1. Objetivo

El alumno será capaz de elaborar programas que den solución a problemas en los que se requiere el uso de sentencias de bifurcación de código.

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 bolígrafo.

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.

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:

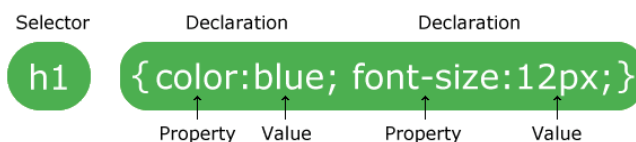
```
css-inline-01.html
<!DOCTYPE HTML>
<html>
<body>
  <p style="color:blue;font-size:46px;">
    letras grandes color azul
  </p>
  <p style="background-color: lightblue;">este es un texto con estilo</p>
</body>
</html>
```

Tabla 4.1 ejemplo uso de css en línea.

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

Una segunda y mejor opción, es agrupar todos los estilos entre los tag **<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 ahora en un archivo externo con extensión .css:

Archivo: ejcss03.html	Archivo: miEstilo.css
<pre><!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></pre>	<pre>p { color: red; text-align: center; }</pre>

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
<pre><!DOCTYPE html> <html> <head> <style> body { background-color: linen; } h1 { color: maroon; margin-left: 40px; } </style> </head> <body></pre>



```
<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;
}
```

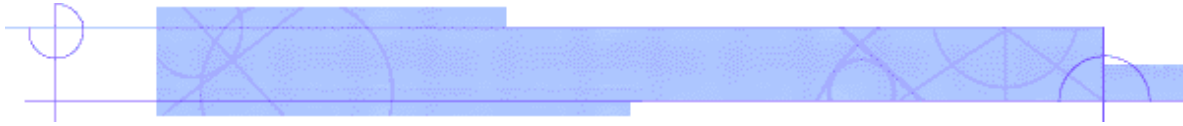


Tabla 4.7

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

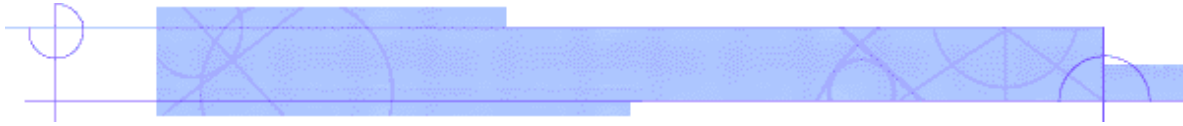
3.4.6 Selector de clase

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
<u><i>.class</i></u>	.intro	Selecciona todos los elementos con class="intro"
<u><i>#id</i></u>	#firstname	Selecciona los elementos con id="firstname"
<u><i>*</i></u>	*	Selecciona todos los elementos
<u><i>element</i></u>	p	Selecciona todos los elementos <p>
<u><i>element,element</i></u>	div, p	Selecciona todos los elementos <div> y todos los elementos <p>
<u><i>element element</i></u>	div p	Selecciona todos los elementos <p> dentro de los elementos <div>
<u><i>element>element</i></u>	div > p	Selecciona todos los elementos <p> que tienen como padre a un elemento <div>
<u><i>element+element</i></u>	div + p	Selecciona todos los <p> que se encuentran inmediatamente después de un elemento <div>



<u>element1~element2</u>	p ~ ul	Selecciona cada elemento que esta precedido por un elemento <p>
<u>[attribute]</u>	[target]	Selecciona todos los elementos con atributo target
<u>[attribute=value]</u>	[target=_blank]	Selecciona todos los elementos con target="_blank"
<u>[attribute~=value]</u>	[title~=flower]	Selecciona todos los elementos con atributo title que contenga la palabra "flower"
<u>[attribute =value]</u>	[lang =en]	Selecciona todos los elementos con atributo lang que comienzan con "en"
<u>[attribute^=value]</u>	a[href^="https"]	Selecciona todos los elementos <a> cuyo valor de atributo href inicia con "https"
<u>[attribute\$=value]</u>	a[href\$=".pdf"]	Selecciona todos los elementos <a> cuyo valor de atributo href termina con ".pdf"
<u>[attribute*=value]</u>	a[href*="w3schools"]	Selecciona todos los elementos <a> cuyo valor de atributo href contiene la subcadena "w3schools"
<u>:active</u>	a:active	Selecciona los link activos
<u>::after</u>	p::after	Inserta algo después del contenido de cada <p>
<u>::before</u>	p::before	Inserta algo antes del contenido de <p>
<u>:checked</u>	input:checked	Selecciona cada elemento <input> seleccionado
<u>:disabled</u>	input:disabled	Selecciona todos los elementos <input> no seleccionados
<u>:empty</u>	p:empty	Selecciona todos los elementos <p> sin hijos (incluyendo nodos de texto)
<u>:enabled</u>	input:enabled	Selecciona todos los elementos <input> activos
<u>:first-child</u>	p:first-child	Selecciona cada elemento <p> que es el primer hijo de su padre
<u>::first-letter</u>	p::first-letter	Selecciona la primera letra de cada elemento <p>
<u>::first-line</u>	p::first-line	Selecciona la primera línea de cada elemento <p>
<u>:first-of-type</u>	p:first-of-type	Selecciona cada elemento<p> que es el primer elemento <p> de su padre



<u>:focus</u>	input:focus	Selecciona el elemento input que tiene el foco
<u>:hover</u>	a:hover	Selecciona los link con el ratón sobre ellos
<u>:in-range</u>	input:in-range	Selecciona los elementos input con un valor dentro de un rango específico
<u>:invalid</u>	input:invalid	Selecciona a todos los elementos input con un valor invalido
<u>:lang(<i>language</i>)</u>	p:lang(it)	Selecciona cada elemento <p> con un atributo lang igual a "it" (Italian)
<u>:last-child</u>	p:last-child	Selecciona cada elemento <p> que es el último hijo de su padre
<u>:last-of-type</u>	p:last-of-type	Selecciona cada elemento <p> que es el último elemento <p> de su padre
<u>:link</u>	a:link	Selecciona todos los links no visitados
<u>:not(<i>selector</i>)</u>	:not(p)	Selecciona cada elemento que no es un elemento <p>
<u>:nth-child(<i>n</i>)</u>	p:nth-child(2)	Selecciona cada elemento <p> que es el segundo hijo de su padre
<u>:nth-last-child(<i>n</i>)</u>	p:nth-last-child(2)	Selecciona cada elemento <p> que es el segundo hijo de su padre contado desde el último
<u>:nth-last-of-type(<i>n</i>)</u>	p:nth-last-of-type(2)	Selecciona cada elemento <p> que es el segundo elemento <p> de su padre, contado desde el último
<u>:nth-of-type(<i>n</i>)</u>	p:nth-of-type(2)	Selecciona cada elemento <p> que es el segundo elemento <p> de su padre
<u>:only-of-type</u>	p:only-of-type	Selecciona cada elemento <p> que es el único elemento <p> de su padre
<u>:only-child</u>	p:only-child	Selecciona cada elemento <p> que es el único hijo de su padre
<u>:optional</u>	input:optional	Selecciona todos los elementos input sin atributo "required"
<u>:out-of-range</u>	input:out-of-range	Selecciona los elementos input con un valor fuera del rango especificado
<u>:read-only</u>	input:read-only	Selecciona los elementos input con atributo "readonly" especificado



<u>:read-write</u>	input:read-write	Selecciona los elementos input con atributo "readonly" NO especificado
<u>:required</u>	input:required	Selecciona los elementos input con atributo "required" especificado
<u>:root</u>	:root	Selecciona el elemento raiz del documento
<u>::selection</u>	::selection	Selecciona la porción de un elemento que esta seleccionado or el usuario
<u>:target</u>	#news:target	Selecciona elemento corriente activo #news (cuando se hace clic en el URL que contiene su ancla)
<u>:valid</u>	input:valid	Selecciona todos los elementos input con un valor valido
<u>:visited</u>	a:visited	Selecciona todos los 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>

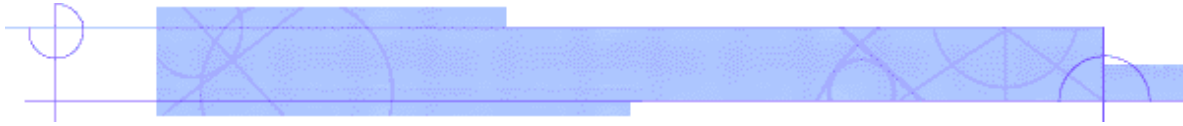
</html>
```

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

3.5. 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.

3.6. 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, dispositivo de almacenamiento secundario (disquete, memoria USB, etc.), editor.



5.3. Materiales

Internet, navegador, manuales.

5.4. Descripción

W3css es una alternativa de calidad a Bootstrap. W3.CSS es Más Pequeño, Más Rápido y Más Fácil de Usar.

Para usar W3.CSS en su sitio web, simplemente agregue un enlace a "w3.css" desde su web páginas:

```
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
```

Elementos

La Clase Contenedor

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

<div>, <article>, <section>, <header>, <footer>, y <form>.

```
<div class="w3-container">
  <p>The w3-container class is an important w3.CSS class.</p>
</div>
```

La Clase panel

El panel agrega un margen superior e inferior de 16px y un relleno izquierdo y derecho de 16px a cualquier elemento HTML.

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

Menú

```
<!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>
  <!-- pruebe cambiar la clase w3-blue-grey por: w3-blue-grey o por w3-blue -->
  <div class="w3-bar w3-blue-grey">
    <a href="#" class="w3-bar-item w3-button">Home</a>
    <a href="#" class="w3-bar-item w3-button">Liga 1</a>
    <a href="#" class="w3-bar-item w3-button">Liga 2</a>
    <a href="#" class="w3-bar-item w3-button">Liga 3</a>
  </div>

</body>
</html>
```

Grid

```
<!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">
  <h3>Mobile First Fluid Grid</h3>
  <p class="w3-large">Try to resize the window!</p>
</div>

<div class="w3-row">
  <div class="w3-col m1 w3-center w3-grey">1</div>
  <div class="w3-col m1 w3-center">2</div>
  <div class="w3-col m1 w3-center w3-grey">3</div>
  <div class="w3-col m1 w3-center">4</div>
  <div class="w3-col m1 w3-center w3-grey">5</div>
  <div class="w3-col m1 w3-center">6</div>
  <div class="w3-col m1 w3-center w3-grey">7</div>
  <div class="w3-col m1 w3-center">8</div>
  <div class="w3-col m1 w3-center w3-grey">9</div>
  <div class="w3-col m1 w3-center">10</div>
  <div class="w3-col m1 w3-center w3-grey">11</div>
  <div class="w3-col m1 w3-center">12</div>
</div>

<div class="w3-row">
  <div class="w3-col w3-container m4 l3 w3-yellow">
    <p>This part will occupy 12 columns on a small screen, 4 on a medium screen, and 3 on a large screen.</p>
  </div>
</div>
```



```
</div>
<div class="w3-col w3-container m8 l9">
  <p>This part will occupy 12 columns on a small screen, 8 on a medium screen, and 9 on a large
screen.</p>
  <p>This part will occupy 12 columns on a small screen, 8 on a medium screen, and 9 on a large
screen.</p>
</div>
</div>

</body>
</html>
```

5.5. Procedimiento

1. En la sección de practicas 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 W3css.
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 bolígrafo.



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 extencion html introduce el código de la tabla 5.1 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
<pre><!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></pre>

Tabla 5.1.

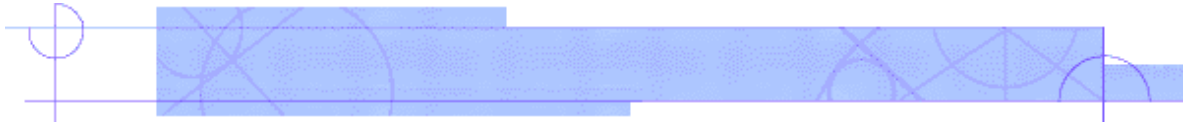
Aquí el trabajo lo hace la función onclick que esta como atributo del boton:

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:

```
<p id="demo">JavaScript puede cambiar el contenido de HTML.</p>
```

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.

También puede cambiar los atributos del HTML

Cambiado atributos, archivo: atributo.html
<pre><!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></pre>

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 especial dentro del archivo HTML (ver tabla 5.3) en la tabla se muestra el uso de una función de JS asociada a un Button, donde la función está definida en el área <script>...</script>

Ejemplo JS
<pre><!DOCTYPE html> <html> <head> <script> function myFunction() { document.getElementById("demo").innerHTML = "Parrafo cambiado."; } </script> </head></pre>



```
<body>
  <h1>JavaScript in Head</h1>
  <p id="demo">Mi parrafo.</p>
  <button type="button" onclick="myFunction()">clíc 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
<pre><!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()">clíc aqui</button> </body> </html></pre>
Archivo: myscript.js
<pre>function myFunction() { document.getElementById("demo").innerHTML = "Parrafo cambiado."; }</pre>

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:

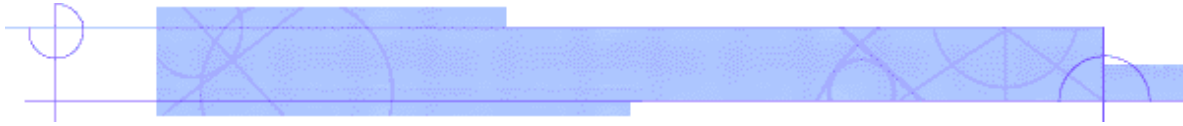
var mivariable;

var miotravariable=23;

Si son variables globales

O asi si son locales:

let mivariable;



let miotravariabale=23;

No se declarar el tipo, el tipo de una variable se asocia 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.

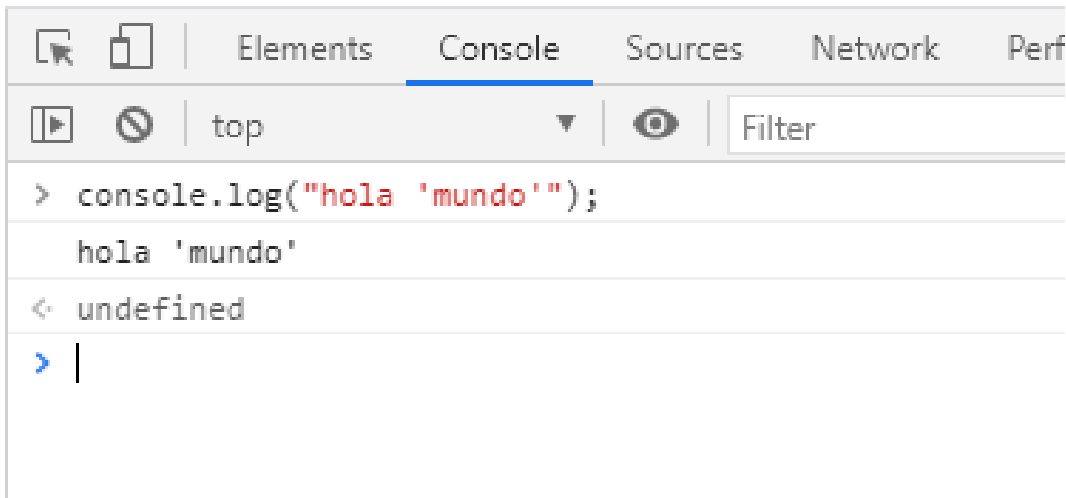
"esta es una cadena"

'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";
```

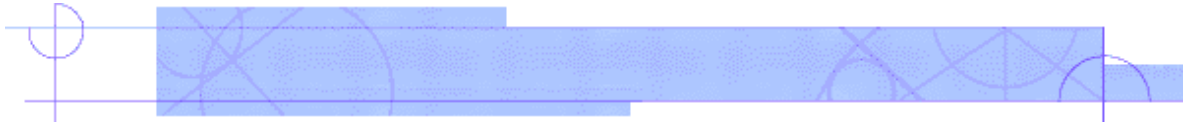
Operadores

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

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

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
<pre><!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></pre>

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')
}
```

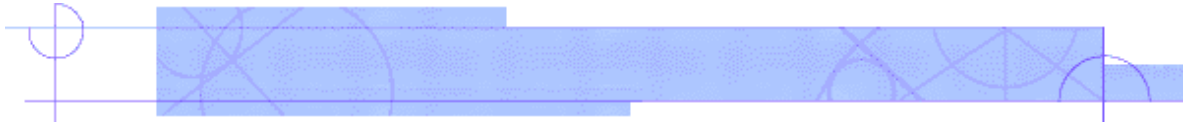
No parece tener sentido ya que al no tener nombre no puedes llamarla, pero se usan cuando se quiere una función autoinvocada o enviar una función como parámetro.

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 elemento 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>
```

También se pueden asignar eventos en forma dinámica dentro del código html:

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

    <style>
      table{
        width:300px;
      }
    </style>
  </head>
  <body>
    <table border="1">
      <tr>
        <td>
          <button onclick="displayDate()">The time is?</button>
        </td>
      </tr>
    </table>
  </body>
</html>
```



```
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:

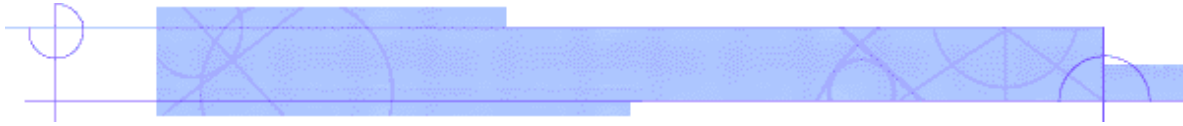
```
"click"
```

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 algo un poco más complejo, mostraremos un menú y asignaremos clic a las opciones del menú de manera que al hacer clic en alguna opción, se muestre en la consola, un mensaje que indique a qué opción se hizo clic: Este ejemplo lo puede ver en la carpeta de ejercicios: ***ejemplos/js-puro/leventos/levent-mnu-0.html***

Y en la tabla siguiente:

Archivo: event-mnu-0.html
<pre><!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></pre>



```
</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 clic en alguna opción en la consola se verá un mensaje.

Tabla 6.4

Explicación:

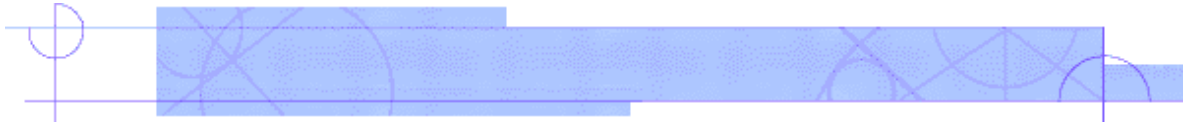
La sentencia

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

Obtiene una lista de elementos, en este caso la lista de elementos cuyo 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 argumento (de manera automática, usted no tiene que enviarlo) un 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 un 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

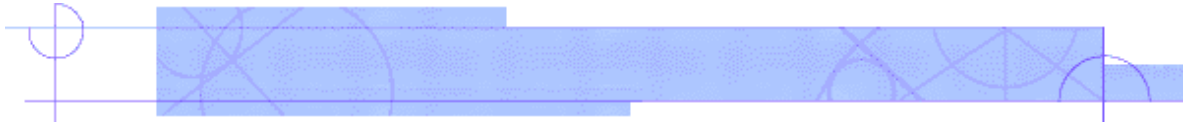
8 prototipos de objetos (opcional).

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:

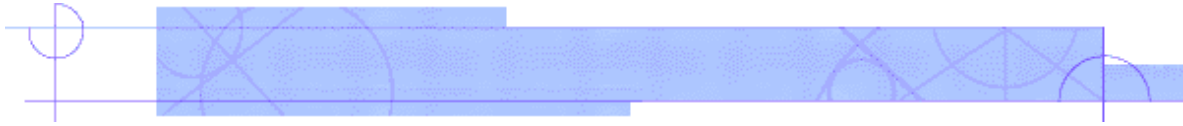
```
var person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";
```

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;
      };
    </script>
  </body>
</html>
```



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

Tabla 7.3

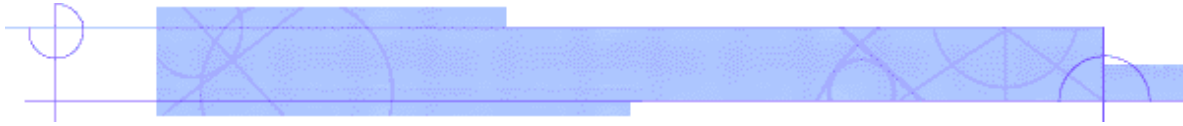
En este ejemplo se agregó la propiedad `nationality` y el método `name` al objeto `myFather` y no al objeto `myMother`.

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"  
    }  
  ]  
</script>
```



```
    }  
  };  
  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 json, y los objetos json tienen asociada cierta 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 en cada iteración del for e index que contiene el índice o posición del elemento dentro del json, console.log ya sabemos que escribe a consola. 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

en seguida dos ejemplos más 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>

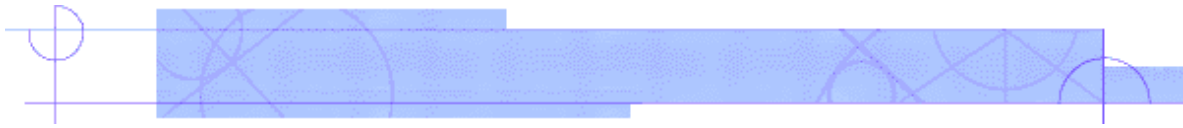
    let personas=[
        {
            nombre : "anita pacheco",
            familia : {
                padre : "anastacio",
                madre : "nastasia"
            },
            edad : 22
        },
        {
            nombre : "bety",
            familia : {
                padre : "beto",
                madre : "beta"
            },
        },
    ]
</script>
```

```
        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

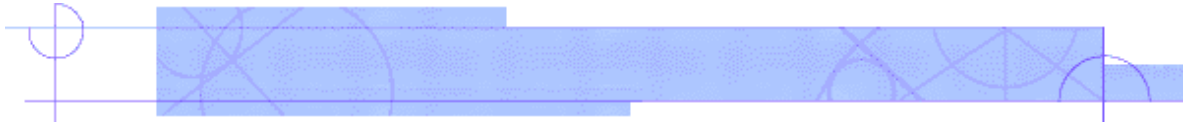
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <style>
      table{
        width:300px;
      }
      th{
        background-color:#008080;
        color:#ffffff;
      }
      tr:nth-child(even){background-color:#f2f2f2;}
      tr:nth-child(odd){background-color:#C0C0C0;}
      th, td{
        text-align:center;
        padding:10px;
      }
      td:hover{
        background-color:#424242;
        color:#ffffff;
      }
    </style>
  </head>
  <body>
    <table>
      <tr>
        <th>Nombre</th>
        <th>Apellido</th>
        <th>Edad</th>
      </tr>
      <tr>
        <td>cesilio</td>
        <td>cesilio</td>
        <td>33</td>
      </tr>
      <tr>
        <td>ceci</td>
        <td>cecilia</td>
        <td>44</td>
      </tr>
    </table>
  </body>
</html>
```



```
#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>
        <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+="|  |
| --- |
|";
            //se accede al json con:element["id"] o con:element.id
            content+=" "+element["id"]+"</td><td>"+element.nombre+"</td>";             content+=" |

```

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 método ***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 <i>ownerDocument</i> 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 <code>document.createElement()</code> 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	<p>Una "nodeList" es una serie de elementos, parecido a lo que devuelve el método <code>document.getElementsByTagName()</code>. Se accede a los items de la nodeList de cualquiera de las siguientes dos formas:</p> <p><code>list.item (1)</code></p> <p><code>lista [1]</code></p> <p>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.</p>
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	<p>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.</p> <p>Un NamedNodeMap es un método de ítem() por esa razón, y permite poner o quitar ítems en un NamedNodeMap</p>

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>
```

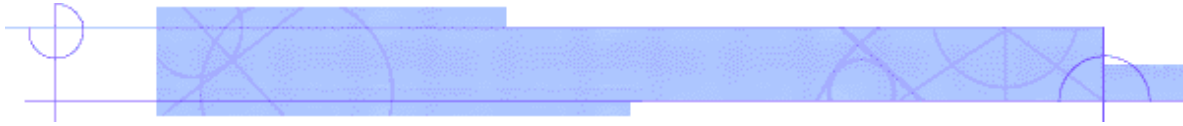


element.getElementsByTagName(name)

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>
```



```
</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. Escriba 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 botón.
3. Defina una cadena JSON para representar la información de los alumnos del curso de cliente servidor 2, los datos requeridos son matrícula, 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. Node Js

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 bolígrafo.

VII.4. Descripción

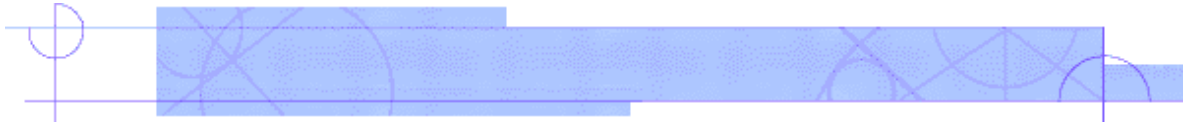
1 Introducción.

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

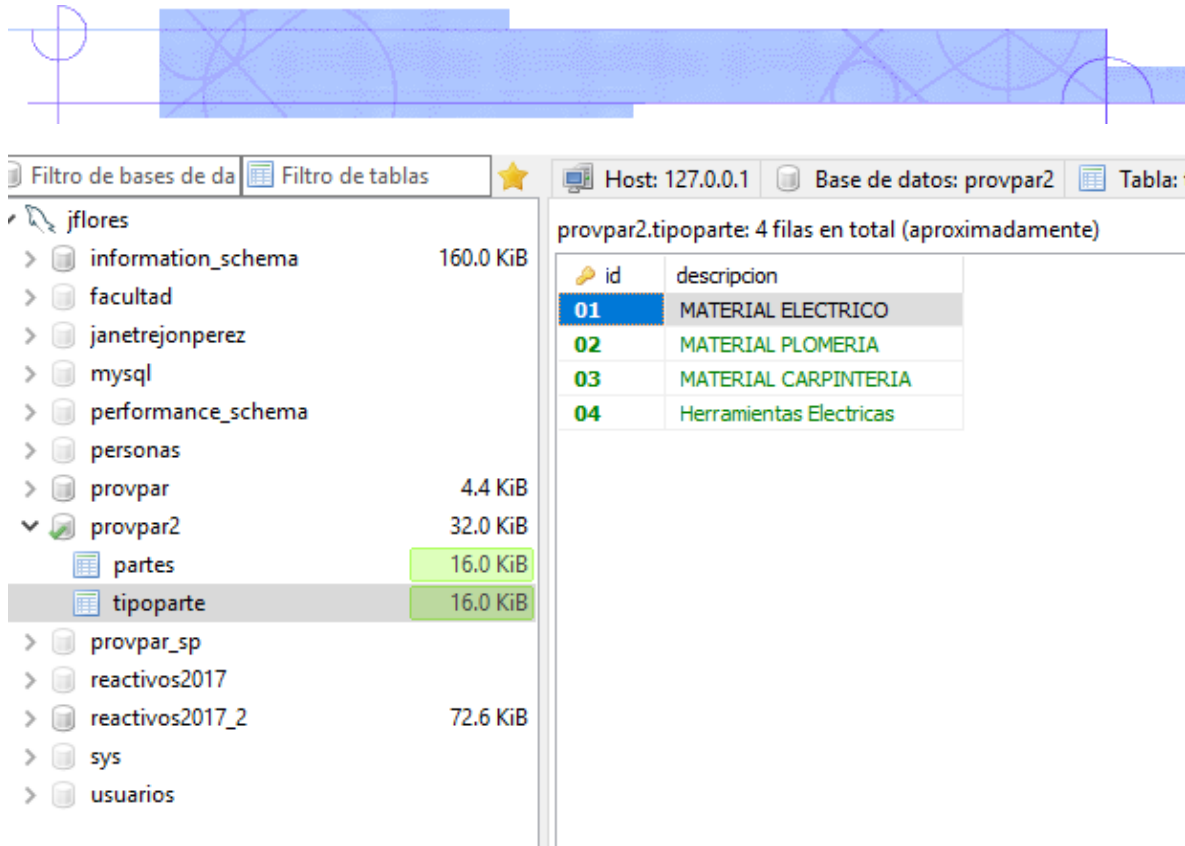
2 Bases de datos

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. También 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:

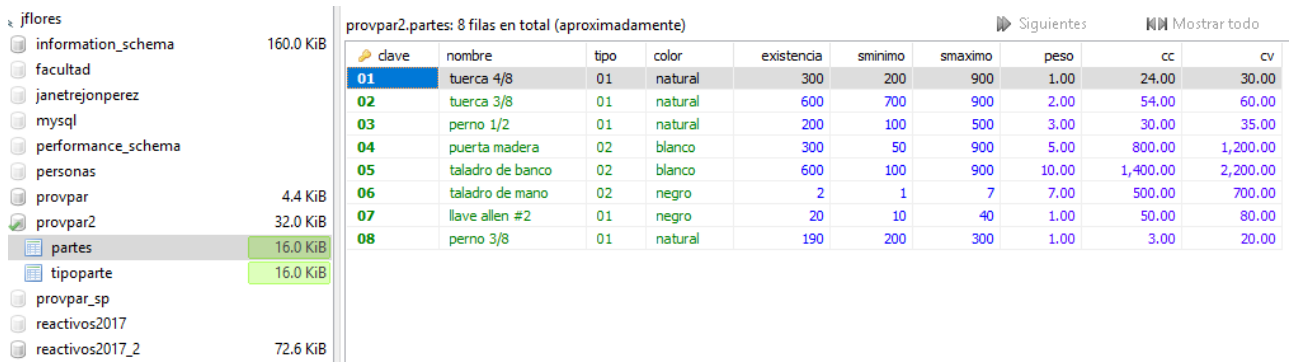
Universidad Autónoma del Carmen



The screenshot shows a database management interface. On the left, a tree view lists databases and tables. The 'provpar2' database is selected, and the 'tipoparte' table is highlighted. On the right, the 'tipoparte' table is displayed with 4 rows. The table has two columns: 'id' and 'descripcion'.

id	descripcion
01	MATERIAL ELECTRICO
02	MATERIAL PLOMERIA
03	MATERIAL CARPINTERIA
04	Herramientas Electricas

Tabla tipoparte de la base de datos provpar2



The screenshot shows the same database management interface. The 'partes' table is selected and displayed with 8 rows. The table has 11 columns: 'clave', 'nombre', 'tipo', 'color', 'existencia', 'sminimo', 'smaximo', 'peso', 'cc', and 'cv'.

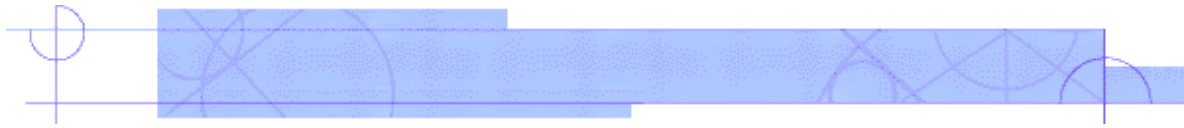
clave	nombre	tipo	color	existencia	sminimo	smaximo	peso	cc	cv
01	tuerca 4/8	01	natural	300	200	900	1.00	24.00	30.00
02	tuerca 3/8	01	natural	600	700	900	2.00	54.00	60.00
03	perno 1/2	01	natural	200	100	500	3.00	30.00	35.00
04	puerta madera	02	blanco	300	50	900	5.00	800.00	1,200.00
05	taladro de banco	02	blanco	600	100	900	10.00	1,400.00	2,200.00
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

Tabla partes de la base de datos provpar2.

Aquí el código para crear la base de datos:

Copie este código en una ventana de consola 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
    $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 Microsoft Visual studio code (o programmersNotepad 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
```



```
$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:

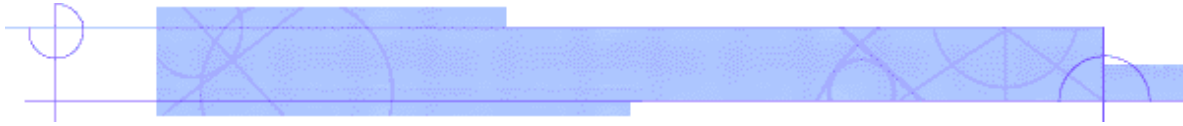
```
$sql = 'select * from tipoparte';
```

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
<pre><!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></pre>
Archivo: inserta.php
<pre><?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"; } ?></pre>

Tabla 7.6

El archivo html tiene un formulario con 2 campos, con atributo name: **id** el primero y **descripción** el segundo. El formulario está asociando 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 la información de los campos en una ventana nueva. Con esto está comprobando que el PHP recibe y devuelve bien la información desde/hacia JS.

4 JS-fetch

En la inserción 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. Crearemos la interface en html+css+bootstrap.
2. Crearemos 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 consulta 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-Dx4S7yoriVYGnpnlTKA/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/1HMD4jTeg3zOOiKsrEqDIsvKUmEQfMMHGcksR9OUoL4o/edit?usp=sharing>
- fetch leer json: <https://docs.google.com/presentation/d/1PsWqwle---sbqWFGbD4ormqljZNJQS0U5k1CNmMphMA/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 deberemos:

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
    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:

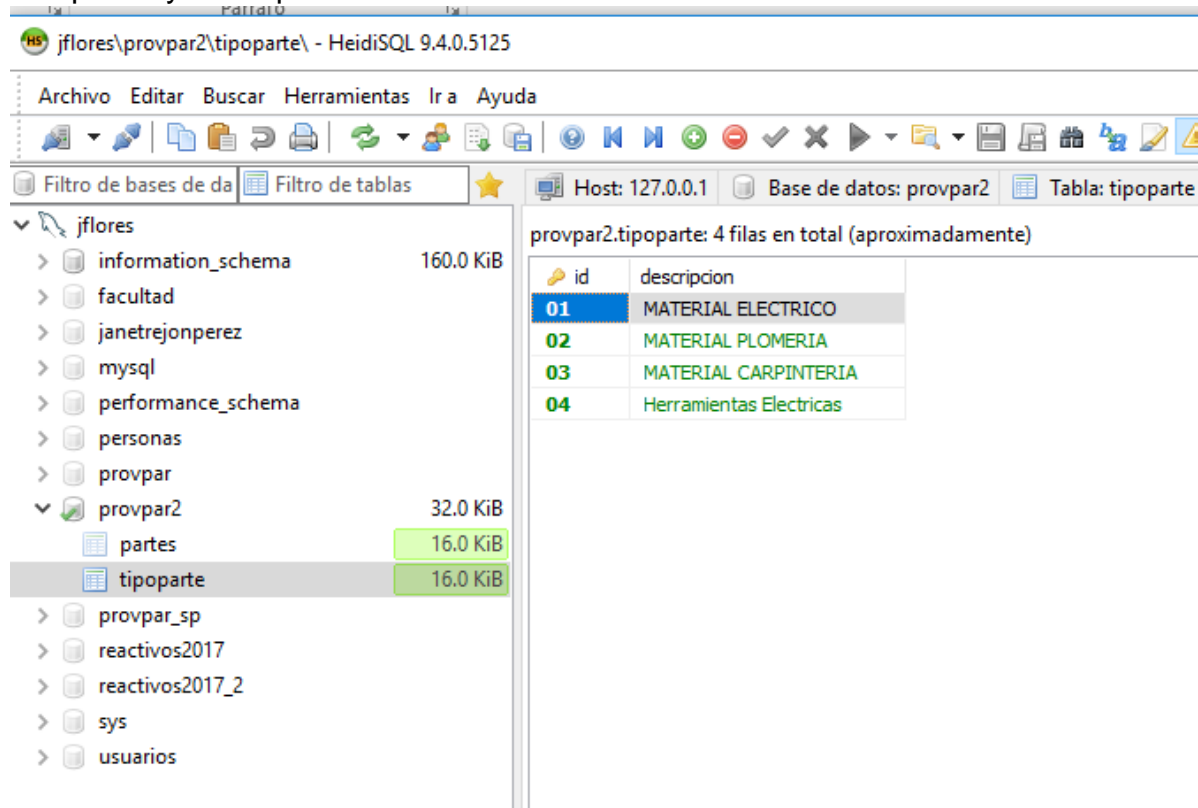
1. Guarde los dos archivos en la carpeta: C:\wamp64\www\cursoapcnjs
2. En su navegador ejecute así: <http://localhost/cursoapcnjs/fetch-basico-envia-formulario.html>
3. Como resultado debe ver que el html devuelve la misma información que recibe.

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:



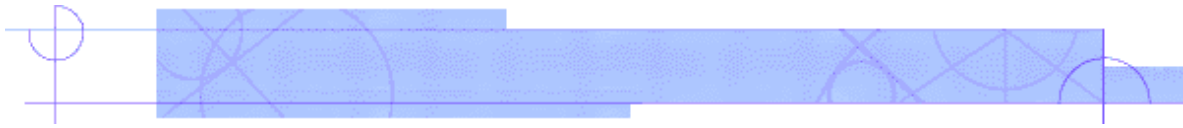
Archivo: inserta_tipo_articulo.html

```
<!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":"insercion 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 realizó la 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, recuerda que el url tiene la dirección del archivo php a ejecutar.

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

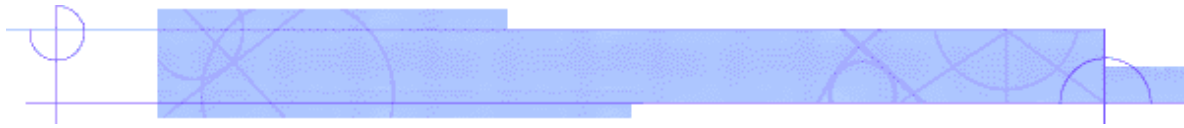
6 Recibir datos con fetch desde php

HTML ← JS-fetch ← PHP ← Base de datos

A continuación, un ejemplo 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+="
```

```
//nota: como se ve a continuación, se accede al json con:
element["id"] o con:element.id

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

Archivo:traePersonasPDO.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);

?>
```

7 Curl/ThunderClient

7.1 Curl

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 en algún formato, el más usado es JSON, por ejemplo:

Pruebe en su navegador las siguientes ligas:

- <https://mdn.github.io/learning-area/javascript/ojs/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://regres.in/api/user>

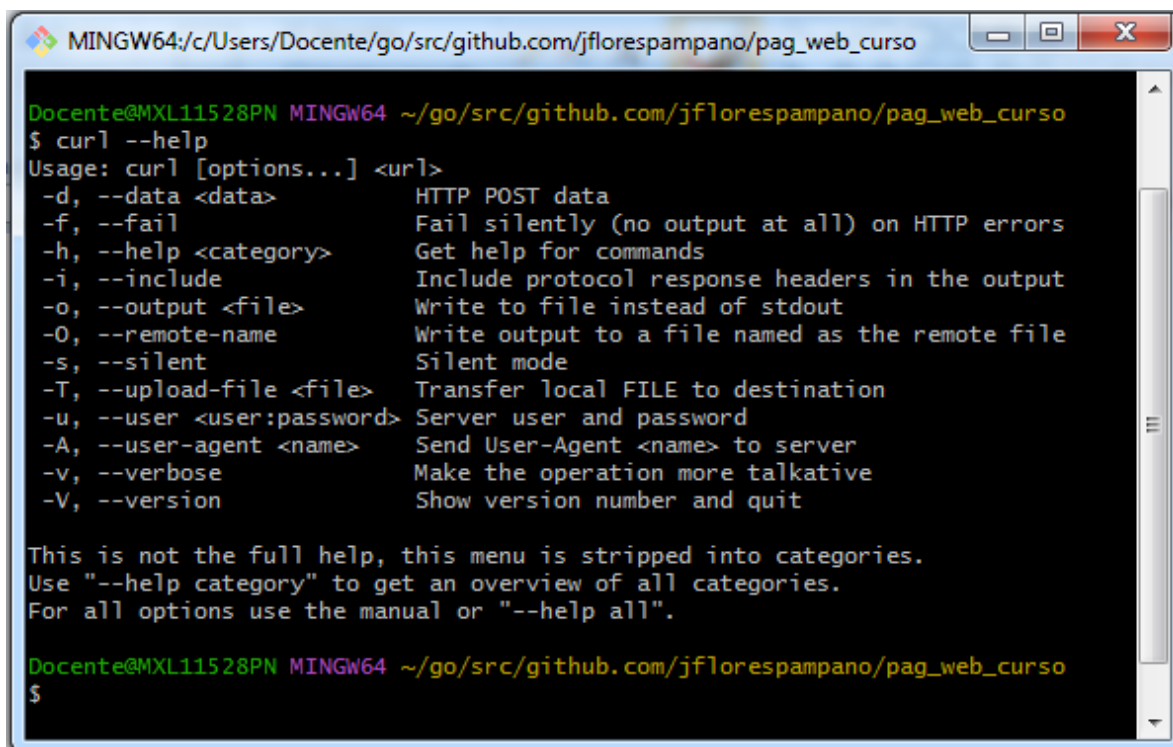
En cada una, vera en su navegador un conjunto de datos json, este es el funcionamiento de una API REST, usted envía solicitudes y recibe datos, para probar un **api** mientras la programamos, o para revisar su funcionamiento mientras implementamos un **front end** para ella, es bastante útil el comando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



```
MINGW64:/c/Users/Docente/go/src/github.com/jflorespampano/pag_web_curso
Docente@MXL11528PN MINGW64 ~/go/src/github.com/jflorespampano/pag_web_curso
$ curl --help
Usage: curl [options...] <url>
  -d, --data <data>           HTTP POST data
  -f, --fail                  Fail silently (no output at all) on HTTP errors
  -h, --help <category>      Get help for commands
  -i, --include               Include protocol response headers in the output
  -o, --output <file>        Write to file instead of stdout
  -O, --remote-name          Write output to a file named as the remote file
  -s, --silent                Silent mode
  -T, --upload-file <file>   Transfer local FILE to destination
  -u, --user <user:password> Server user and password
  -A, --user-agent <name>    Send User-Agent <name> to server
  -v, --verbose               Make the operation more talkative
  -V, --version               Show version number and quit

This is not the full help, this menu is stripped into categories.
Use "--help category" to get an overview of all categories.
For all options use the manual or "--help all".

Docente@MXL11528PN MINGW64 ~/go/src/github.com/jflorespampano/pag_web_curso
$
```

Vera una ayuda sobre el uso de curl.

Pruebe el comando:

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

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.

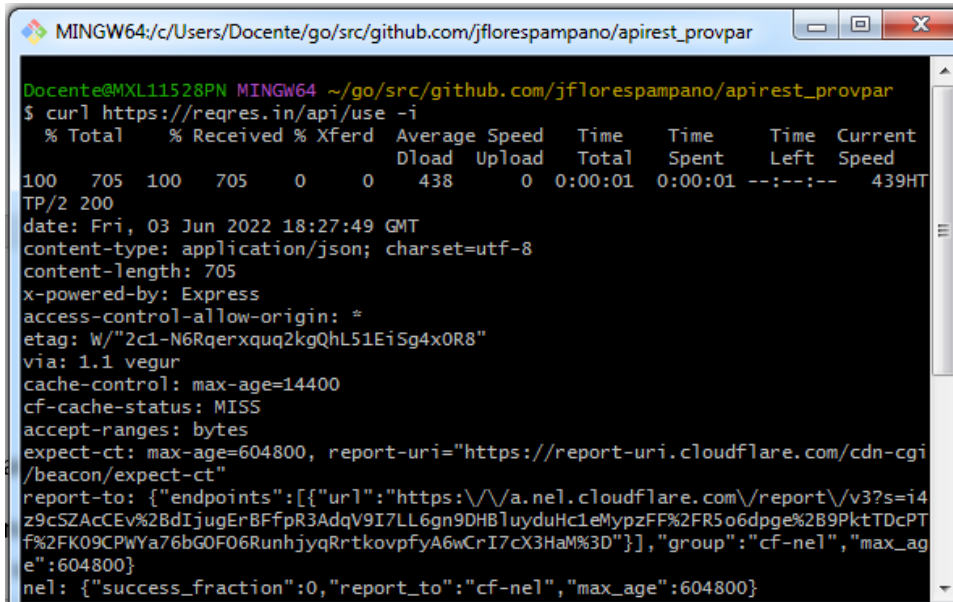
Ahora vera la ventaja de usar curl, escriba el comando:

```
$ 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:



```
MINGW64/c/Users/Docente/go/src/github.com/jflorespampano/apirest_provpar
Docente@MXL11528PN MINGW64 ~/go/src/github.com/jflorespampano/apirest_provpar
$ curl https://reqres.in/api/user -i
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
         Dload  Upload   Total   Spent    Left     Speed
100  705    100  705    0    0   438    0  0:00:01  0:00:01 --:--:--  439HT
TP/2 200
date: Fri, 03 Jun 2022 18:27:49 GMT
content-type: application/json; charset=utf-8
content-length: 705
x-powered-by: Express
access-control-allow-origin: *
etag: W/"2c1-N6Rqerxquq2kgQhL51EiSg4x0R8"
via: 1.1 vegur
cache-control: max-age=14400
cf-cache-status: MISS
accept-ranges: bytes
expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
report-to: {"endpoints":[{"url":"https://a.ne1.cloudflare.com/report/v3?s=i4z9cSZAcCEv%2BdIjugErBFfpR3AdqV9I7LL6gn9DH8luyduHc1eMypzFF%2FR5o6dpge%2B9PktTDcPTf%2FK09CPWYa76bG0F06RunhjyqRrtkovpfyA6wCrI7cX3HaM%3D"}],"group":"cf-nel","max_age":604800}
nel: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
```

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.typicode.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-J5bigoeQz1G_AfD4K4cWPn8&index=17

de donde se tomaron estos ejemplos.

Resumen comandos curl:

#solicitar recurso:
<i>\$curl https://jsonplaceholder.typicode.com/posts/1</i>
#solicitar recurso y almacenarlo en un archivo:
<i>curl -o datos.txt https://jsonplaceholder.typicode.com/posts/1</i>
#solicitud con get , para poner el & en una consola de linux, necesitara anteponer la barra invertida (ok)
<i>curl http://localhost:8080/tipoparte/get?id=45\&nombre=juan\&edad=34</i>
#enviar archivo de datos json
<i>\$curl -d "@data.json" -X POST http://misitio/post</i>
#enviar con encabezado
<i>\$curl -d '{"title": "tit1", "body": "body1", "author": {"name": "juan", "email": "juan@mail"}}' -H "Content-Type: application/json" -X POST http://misitio/post</i>
#enviar con encabezado
<i>\$ curl --header "Content-type: Application/json" -d '{"title": "titulo1", "body": "b1", "userId": 2}' https://jsonplaceholder.typicode.com/posts</i>
#probar ok
<i>curl "http://misitio" -X POST -d '{"nombre": "juan"}' -H "Content-Type: application/json"</i>
#probar -F usa: multipart/form-data
<i>curl -X POST -F 'name=noviello' -F 'email=noviello@example.com' https://example.com/contact.php</i>
probar -d usa: Content-Type application/x-www-form-urlencoded (ok)
<i>curl -X POST -d 'name=noviello' -d 'email=noviello@example.com' https://example.com/contact.php</i>
#por default -d usa POST (ok)
<i>\$ curl -i -d "id=55" -d "nombre=juan" -d "carrera=isc" http://localhost/ejemplos/php/recibePetición.php</i>



```
#por default -F usa post (ok)
$ curl -i -F "id=55" -F "nombre=juan" -F "carrera=isc"
http://localhost/ejemplos/php/recibePeticion.php

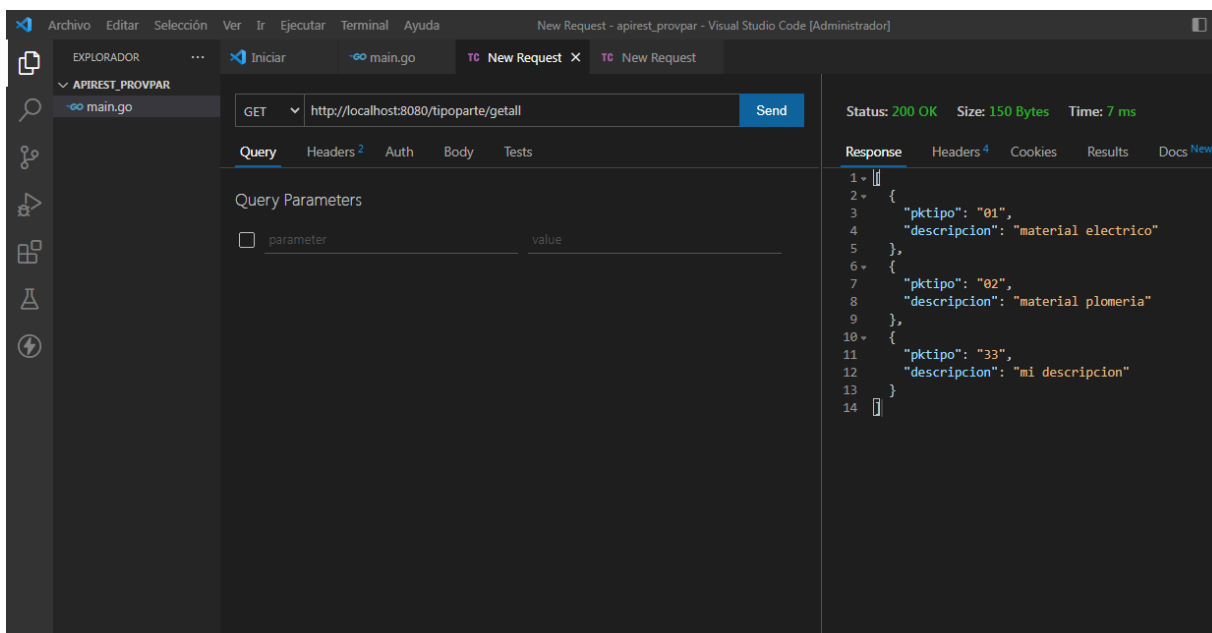
# por ejemplo para un sitio que espera contenido en formato json:
$ curl -H "Content-Type:application/json" -d "@entrada.json" -X POST
http://localhost:8080/tipoparte/crear

# y donde el archivo entrada.json contiene:
{
  "tipo": "02",
  "descripcion": "material plomeria"
}
```

7.2 Thunder

Probarlo con **Thuneder Client**:

1. Ejecute su programa.
2. En su editor **code**, abra una ventana de **Thunder**:



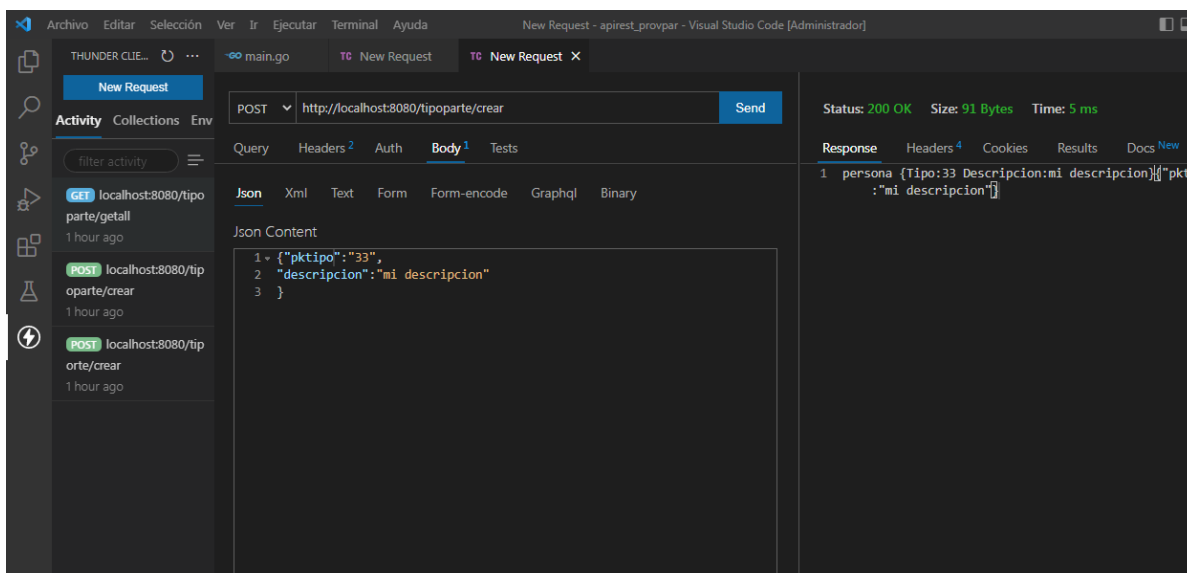
En **Thunder** haga una solicitud **get** a la dirección: **http://localhost:8080/tipoparte/getall**

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**, mientras está corriendo su aplicación, haga:

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

Universidad Autónoma del Carmen



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
<pre>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) }</pre>
<pre># Pruébalo 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</pre>

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

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.
- [Tenenbaum1996]** Tenenbaum, Aaron M. Tenenbaum, 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>
- [mozilla/html]** <https://developer.mozilla.org/es/docs/Web/HTML>
- [mozilla/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-what-is.php>