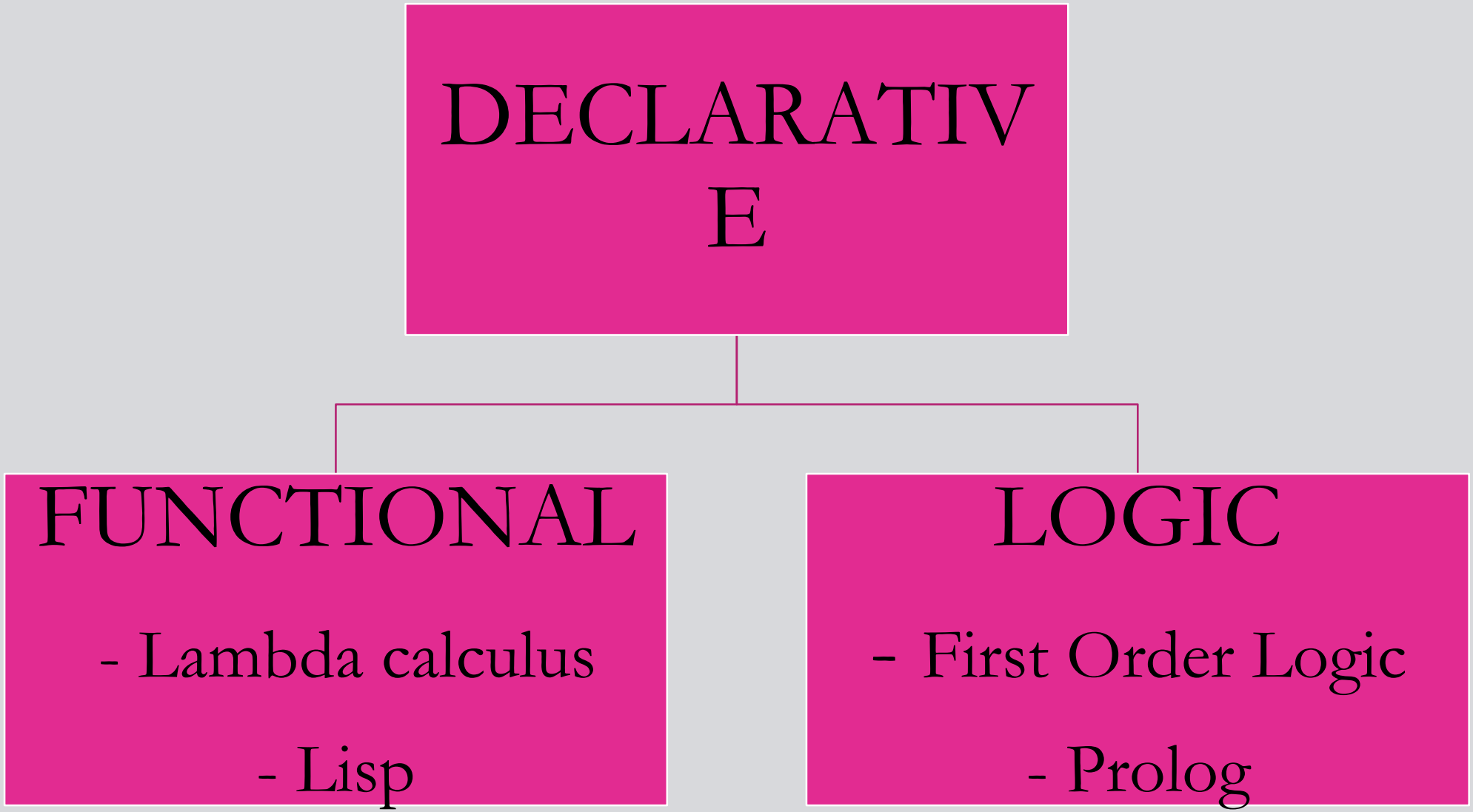
DECLARATIVE PROGRAMMING PARADIGM

INTRODUCTION

- Declarative programming is a programming paradigm that expresses the **logic of a computation** without describing its control flow.
- This paradigm often considers programs as theories of a **formal logic**, and computations as deductions in that logic space.
- Declarative programming is often defined as any style of programming that is **not imperative**.
- Common declarative languages include those of database **query languages** (SQL), **logic programming, functional programming**, etc.

HISTORY


DECLARATIVE
FUNCTIONAL
- Lambda calculus
- Lisp
LOGIC
- First Order Logic
- Prolog

OVERVIEW

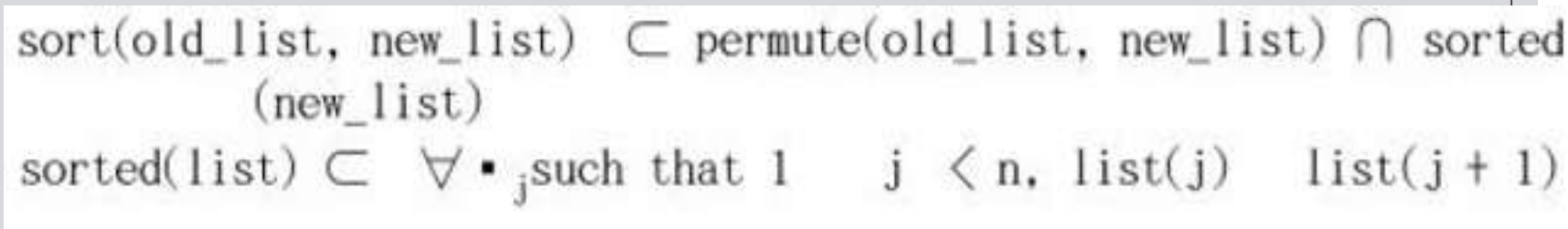
- A program that describes **what** computation should be performed and **not how** to compute it. Non-imperative, non-procedural.
- Any programming language that **lacks side effects**(example: a function might modify a global variable or static variable, modify one of its arguments, raise an exception,).
- A language with a clear correspondence to **mathematical logic.**

OVERVIEW – Logic Paradigm

- Computing takes place overthe domain of all terms defined over a **"universal"** alphabet.
- Values are assigned to variables by means of automatically generated substitutions, called **most general unifiers**. These values may contain variables, called **logical variables**.
- The control is provided by a single mechanism: **automatic backtracking**.

Declarative Semantics Vs Imperative Semantics

- In declarative semantics the meaning of a given proposition in a logic programming language can be concisely determined from the **statement itself**.
- In an imperative language, the semantics of a simple assignment statement requires examination **local declarations**, **scoping** rules of the language, **types** of variables in the assignment statement, depends on its **run-time context**.



SQL - Structured Query Language

- SQL is the standard language used to communicate with a relational database.
- It can be used to retrieve data from a database using a query but it can also be used to create , destroy as well as modify their structure.

SQL - ELEMENTS

- The language is subdivided into several language elements, including:
  - Clauses
  - Expressions
  - Predicates
  - Queries
  - Statements

types of programming:

procedural (imperative)

object-oriented

SQL declarative (nonprocedural)

functional

procedural (imperative)

how

declarative (nonprocedural)

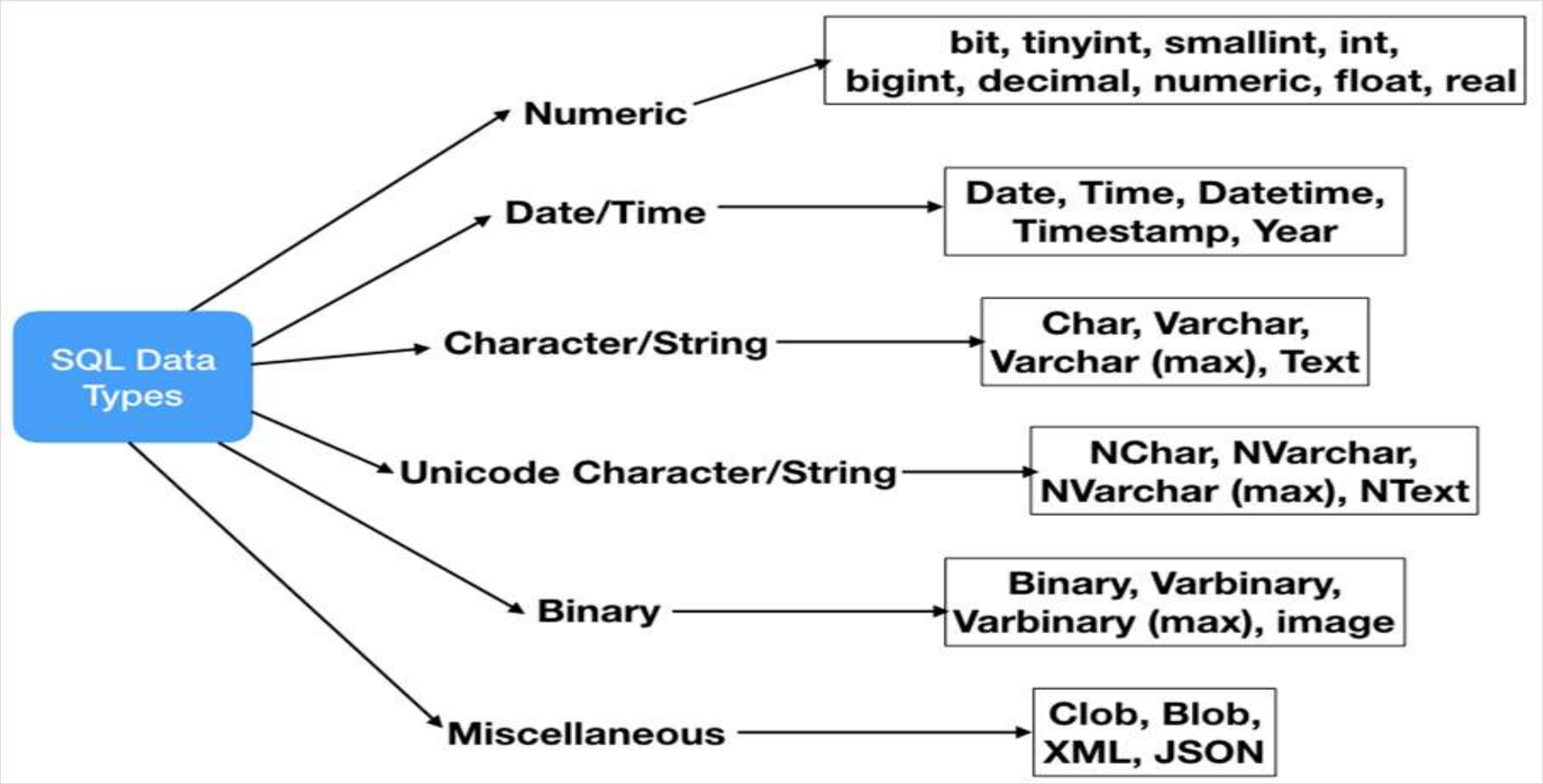
WHAT

1. Please, open the door.
2. Go outside.
3. Take the bucket I forgot there.
4. Bring it back to me

1. Fetch the bucket, please.

OPTIMIZER

SQL – DATA TYPES



SQL Numeric Data Types

Datatype	From	To
bit	0	1
tinyint	0	255
smallint	-32,768	32,767
int	-2,147,483,648	2,147,483,647
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

SQL Date and Time Data Types

Datatype	Description
DATE	Stores date in the format YYYY-MM-DD
TIME	Stores time in the format HH:MI:SS
DATETIME	Stores date and time information in the format YYYY-MM-DD HH:MI:SS
TIMESTAMP	Stores number of seconds passed since the Unix epoch ('1970-01-01 00:00:00' UTC)
YEAR	Stores year in 2 digit or 4 digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069.

SQL Character and String Data Types

Datatype	Description
CHAR	Fixed length with maximum length of 8,000 characters
VARCHAR	Variable length storage with maximum length of 8,000 characters
VARCHAR(max)	Variable length storage with provided max characters, not supported in MySQL
TEXT	Variable length storage with maximum size of 2GB data

Some of The Most Important SQL Commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

SQL EXAMPLES – INSERT INTO

```sql
CREATE TABLE Student (
    StudID int,
    LastName varchar(255),
    FirstName varchar(255),
    Gender varchar(255),
    DOB DATE
    );

INSERT INTO Student (StudID, Lastname, Firstname, Gender, DOB)
VALUES ('1000', 'H', 'John', 'Male','1990-01-01');

select * from Student;
```

```
1000|H|John|Male|1990-01-01
```

```sql
CREATE TABLE Scientist (
        SciID int,
        LastName varchar(255),
        FirstName varchar(255),
        Gender varchar(255),
        DOB DATE
        );

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1000', 'Albert', 'Einstein', 'Male','1879-03-14');

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1005', 'Marie', 'Curie', 'Female','1867-11-07');

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1010', 'Isaac', 'Newton', 'Male','1643-01-04');

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1015', 'Ada', 'Lovelace', 'Female','1815-12-10');

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1020', 'Charles', 'Darwwin', 'Male','1809-02-12');

select * from Scientist;
```

```
1000|Albert|Einstein|Male|1879-03-14
1005|Marie|Curie|Female|1867-11-07
1010|Isaac|Newton|Male|1643-01-04
1015|Ada|Lovelace|Female|1815-12-10
1020|Charles|Darwwin|Male|1809-02-12
```

WHERE CLAUSE

```
select * from Scientist where Gender = 'Male';
```

```
1000|Albert|Einstein|Male|1879-03-14
1010|Isaac|Newton|Male|1643-01-04
1020|Charles|Darwwin|Male|1809-02-12
```

ORDER BY

```
select * from Scientist ORDER BY DOB;
```

```
1010|Isaac|Newton|Male|1643-01-04
1020|Charles|Darwwin|Male|1809-02-12
1015|Ada|Lovelace|Female|1815-12-10
1005|Marie|Curie|Female|1867-11-07
1000|Albert|Einstein|Male|1879-03-14
```

UPDATE

```
UPDATE Scientist SET Firstname = 'Sundar',
    Lastname = 'Pitchai', DOB = '1972-06-10'
    where SciID = 1020;
select * from Scientist;
```

```
1000|Albert|Einstein|Male|1879-03-14
1005|Marie|Curie|Female|1867-11-07
1010|Isaac|Newton|Male|1643-01-04
1015|Ada|Lovelace|Female|1815-12-10
1020|Pitchai|Sundar|Male|1972-06-10
```

DELETE

```
delete from Scientist where SciID = 1015;
select * from Scientist;
```

```
1000|Albert|Einstein|Male|1879-03-14
1005|Marie|Curie|Female|1867-11-07
1010|Isaac|Newton|Male|1643-01-04
1020|Pitchai|Sundar|Male|1972-06-10
```

ALTER – ADD COLUMN

```
ALTER TABLE Scientist ADD Country varchar(100);
Update Scientist set Country = 'India' where
    SciID = 1020;
select * from Scientist;
```

```
1000|Albert|Einstein|Male|1879-03-14|
1005|Marie|Curie|Female|1867-11-07|
1010|Isaac|Newton|Male|1643-01-04|
1020|Pitchai|Sundar|Male|1972-06-10|India
```

ALTER – DELETE COLUMN

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

- In SQLite V3.20.1 it is not possible to rename a column, remove a column, or add or remove constraints from a table.

MARKUP LANGUAGE

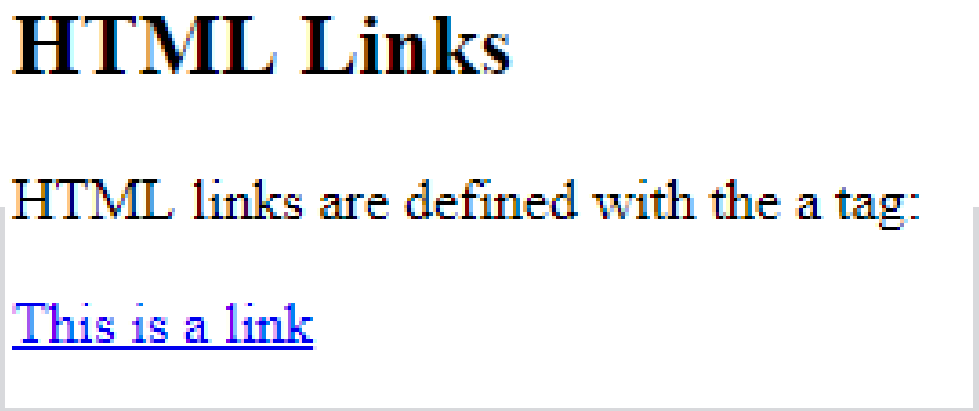
- Many markup languages such as HTML, MXML, XAML are often declarative.
- XAML (Extensible Application Markup Language) is a declarative XML-based language that is used for initializing structured values and objects. It is used extensively in .NET Framework 3.0 and .NET Framework 4.0 technologies, particularly WPF (Windows Presentation Foundation), Silverlight, Windows Store Apps…
- In WPF, XAML forms a user interface markup language to define UI elements, data binding, eventing, and other features. Anything that is created or implemented in XAML can be expressed using a more traditional .NET language such as C# or Visual Basic .NET.

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

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

My First Heading

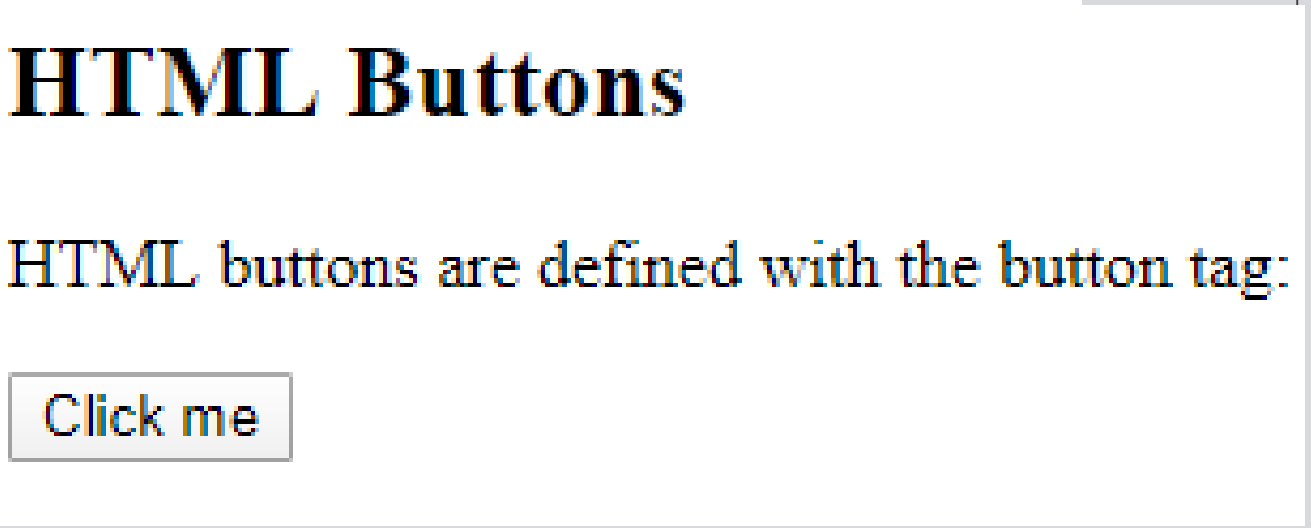
My first paragraph.

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

<h2>HTML Links</h2>
<p>HTML links are defined with the a tag:</p>

<a href="https://www.w3schools.com">This is a link</a>

</body>
</html>
```



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

<h2>An Unordered HTML List</h2>

<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>

<h2>An Ordered HTML List</h2>

<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>

</body>
</html>
```

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

<h2>HTML Buttons</h2>
<p>HTML buttons are defined with the button tag:</p>

<button>Click me</button>

</body>
</html>
```



DEMO

SQL EXAMPLES – INSERT INTO

```sql
CREATE TABLE Student (
    StudID int,
    LastName varchar(255),
    FirstName varchar(255),
    Gender varchar(255),
    DOB DATE
    );

INSERT INTO Student (StudID, Lastname, Firstname, Gender, DOB)
VALUES ('1000', 'H', 'John', 'Male','1990-01-01');

select * from Student;
```

```
1000|H|John|Male|1990-01-01
```

SQL in PYTHON

```
import sqlite3

# connecting to the database
connection = sqlite3.connect("myTable.db")

crsr = connection.cursor()

sql_command = """CREATE TABLE Student (
studID INTEGER PRIMARY KEY,
fname VARCHAR(20),
lname VARCHAR(30),
gender VARCHAR(30),
DOB DATE);"""

crsr.execute(sql_command)

sql_command = """INSERT INTO Student VALUES (1000, "H", "John", "Male", "1990-01-01");"""
crsr.execute(sql_command)

# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()

connection.close()
```

```
crsr.execute("SELECT * FROM Student")

# store all the fetched data in the ans variable
ans= crsr.fetchall()

# loop to print all the data
for i in ans:
    print(i)
```

```
(1000, 'H', 'John', 'Male', '1990-01-01')
```

```sql
CREATE TABLE Scientist (
        SciID int,
        LastName varchar(255),
        FirstName varchar(255),
        Gender varchar(255),
        DOB DATE
        );

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1000', 'Albert', 'Einstein', 'Male','1879-03-14');

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1005', 'Marie', 'Curie', 'Female','1867-11-07');

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1010', 'Isaac', 'Newton', 'Male','1643-01-04');

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1015', 'Ada', 'Lovelace', 'Female','1815-12-10');

INSERT INTO Scientist (SciID, Lastname, Firstname, Gender, DOB)
VALUES ('1020', 'Charles', 'Darwwin', 'Male','1809-02-12');

select * from Scientist;
```

```
1000|Albert|Einstein|Male|1879-03-14
1005|Marie|Curie|Female|1867-11-07
1010|Isaac|Newton|Male|1643-01-04
1015|Ada|Lovelace|Female|1815-12-10
1020|Charles|Darwwin|Male|1809-02-12
```

```
sql_command = """CREATE TABLE Scientist (
sciD INTEGER PRIMARY KEY,
fname VARCHAR(20),
lname VARCHAR(30),
gender VARCHAR(30),
DOB DATE);"""
crsr.execute(sql_command)

sql_command = """INSERT INTO Scientist VALUES (1000, "Albert", "Einstein", "Male", "1879-03-14");"""
crsr.execute(sql_command)
sql_command = """INSERT INTO Scientist VALUES (1005, "Marie", "Curie", "Female", "1867-11-07");"""
crsr.execute(sql_command)
sql_command = """INSERT INTO Scientist VALUES (1010, "Isaac", "Newton", "Male", "1643-01-04");"""
crsr.execute(sql_command)
sql_command = """INSERT INTO Scientist VALUES (1015, "Ada", "Lovelace", "Female", "1815-12-10");"""
crsr.execute(sql_command)
sql_command = """INSERT INTO Scientist VALUES (1020, "Charles", "Darwin", "Male", "1809-02-12");"""
crsr.execute(sql_command)
# execute the command to fetch all the data from the table emp
crsr.execute("SELECT * FROM Scientist")
# store all the fetched data in the ans variable
ans= crsr.fetchall()
# loop to print all the data
for i in ans:
    print(i)
connection.close()
```

```
(1000, 'Albert', 'Einstein', 'Male', '1879-03-14')
(1005, 'Marie', 'Curie', 'Female', '1867-11-07')
(1010, 'Isaac', 'Newton', 'Male', '1643-01-04')
(1015, 'Ada', 'Lovelace', 'Female', '1815-12-10')
(1020, 'Charles', 'Darwin', 'Male', '1809-02-12')
```