Event Driven Programming Paradigm

- Event-driven programming is a programming paradigm in which the flow of program execution is determined by events - for example a user action such as a mouse click, key press, or a message from the operating system or another program.
- An event-driven application is designed to detect events as they occur, and then deal with them using an appropriate event-handling procedure.
- In a typical modern event-driven program, there is no discernible flow of control. The main routine is an event-loop that waits for an event to occur, and then invokes the appropriate event-handling routine.
- Event call-back is a function that is invoked when something significant happens like when click event is performed by user or the result of database query is available.

Event Handlers: Event handlers is a type of function or method that run a specific action when a specific event is triggered. For example, it could be a button that when user click it, it will display a message, and it will close the message when user click the button again, this is an event handler.

Trigger Functions: Trigger functions in event-driven programming are a function that decide what code to run when there are a specific event occurs, which are used to select which event handler to use for the event when there is specific event occurred.

Events: Events include mouse, keyboard and user interface, which events need to be triggered in the program in order to happen, that mean user have to interacts with an object in the program, for example, click a button by a mouse, use keyboard to select a button and etc.

Tkinter is a Python library which is used to create GUI-based application. Tkinter comes with many inbuilt features and extensions which can be used to optimize the application performance and behaviour. Tkinter Events are generally used to provide an interface that works as a bridge between the User and the application logic. We can use Events in any Tkinter application to make it operable and functional.

Here is a list of some common Tkinter events which are generally used for making the application interactive.

<Button> − Use the Button event in a handler for binding the Mouse wheels and Buttons.

<ButtonRelease> − Instead of clicking a Button, you can also trigger an event by releasing the mouse buttons.

<Configure> − Use this event to change the widgets properties.

Destroy − Use this event to kill or terminate a particular widget.

<Enter> − It actually works like <return> event that can be used to get the focus on a widget with mouse Pointer

<Expose> − The event occurs whenever a widget or some part of the application becomes visible that covered by another window in the application.

<Focus In> − This event is generally used to get the focus on a particular widget.

<Focus Out> − To move the focus from the current widget.

<Key Press> − Start the process or call the handler by pressing the key.

<KeyRelease> − Start the process or call an event by releasing a key.

<Leave> − Use this event to track the mouse pointer when user switches from one widget to another widget.

<Map> − Use Map event to show or display any widget in the application.

<Motion> − Track the event whenever the mouse pointer moves entirely within the application.

<Unmap> − A widget can be unmapped from the application. It is similar to hiding the widget using grid_remove().

<Visibility> − An event can happen if some part of the application gets visible in the screen.

Graphical User Interface

A graphical user interface allows the user to interact with the operating system and other programs using graphical elements such as icons, buttons, and dialog boxes.

- GUIs popularized the use of the mouse.
- GUIs allow the user to point at graphical elements and click the mouse button to activate them.
- GUI Programs are Event-Driven
- A graphical user interface allows the user to interact with the operating system and other programs using graphical elements such as icons, buttons, and dialog boxes.
- GUIs allow the user to point at graphical elements and click the mouse button to activate them.
- GUI Programs are Event-Driven
- User determines the order in which things happen
- GUI programs respond to the actions of the user, thus they are event driven.

What is Tkinter?

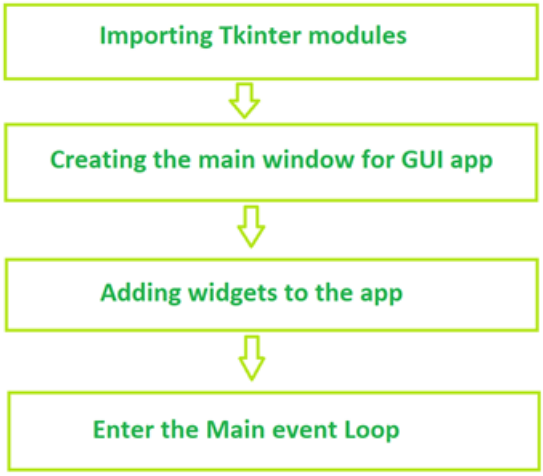
Tkinter is the inbuilt python module that is used to create GUI applications. It is one of the most commonly used modules for creating GUI applications in Python as it is simple and easy to work with.

Fundamental structure of tkinter program

1. Import tkinter package and all of its modules.

2. Create a root window. Give the root window a title (using title ()) and dimension (using geometry ()). All other widgets will be inside the root window.

3. Use mainloop() to call the endless loop of the window. If you forget to call this nothing will appear to the user. The window will wait for any user interaction till we close it.



What are Widgets?

Widgets in Tkinter are the elements of GUI application which provides various controls (such as Labels, Buttons, ComboBoxes, CheckBoxes, MenuBars, RadioButtons and many more) to users to interact with the application.

These are 19 widgets available in Python Tkinter module.

Widgets	Description
Button	The Button widget is used to display buttons in your application.
Canvas	The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
Checkbutton	The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
Entry	The Entry widget is used to display a single-line text field for accepting values from a user.
Frame	The Frame widget is used as a container widget to organize other widgets.
Label	The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
Listbox	The Listbox widget is used to provide a list of options to a user.
Menubutton	The Menubutton widget is used to display menus in your application.
Menu	The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
Message	The Message widget is used to display multiline text fields for accepting values from a user.
Radiobutton	The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
Scale	The Scale widget is used to provide a slider widget.

Scrollbar	The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
Text	The Text widget is used to display text in multiple lines.
Toplevel	The Toplevel widget is used to provide a separate window container.
Spinbox	The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
PanedWindow	A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
LabelFrame	A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
tkMessageBox	This module is used to display message boxes in your applications.

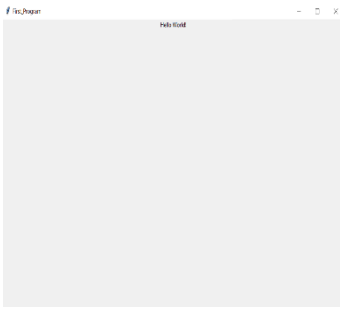
Standard attributes

1. Dimensions
2. Colors
3. Fonts
4. Anchors
5. Relief styles
6. Bitmaps
7. Cursors

Example

```
from tkinter import *
from tkinter.ttk import *

# writing code needs to
# create the main window of
# the application creating
# main window object named root
root = Tk()

# giving title to the main window
root.title("First_Program")

# Label is what output will be
# show on the window
label = Label(root, text ="Hello World !").pack()

# calling mainloop method which is used
# when your application is ready to run
# and it tells the code to keep displaying
root.mainloop()
```

Output:



Geometry Management

Tkinter exposes the following geometry manager classes: pack, grid, and place.

1. The pack() Method − This geometry manager organizes widgets in blocks before placing them in the parent widget.

2. The grid() Method − This geometry manager organizes widgets in a table-like structure in the parent widget.

3. The place() Method − This geometry manager organizes widgets by placing them in a specific position in the parent widget.

Pack - pack method, as literally indicated, packs the widget to the window frame after it is created.

Syntax: `widget.pack( pack_options )`

Here is the list of possible options −

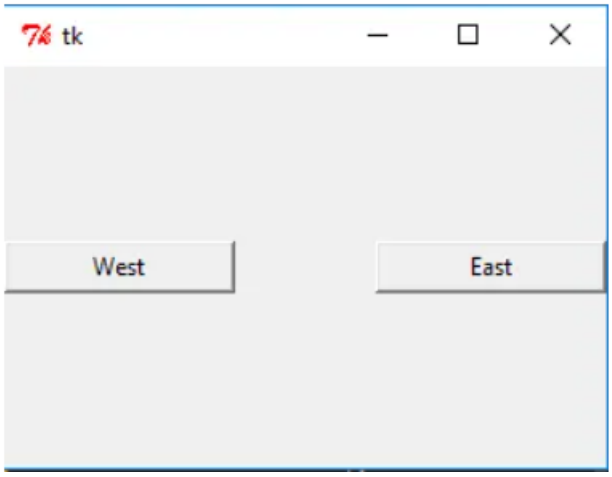
1. expand − When set to true, widget expands to fill any space not otherwise used in widget's parent.

2. fill − Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally and vertically).
3. side − Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT.

Example

```
import tkinter as tk
app = tk.Tk()
app.geometry('300x200')
buttonW = tk.Button(app, text="West", width=15)
buttonW.pack(side='left')
buttonE = tk.Button(app, text="East", width=15)
buttonE.pack(side='right')
app.mainloop()
```

Output:



side has four options -
top
bottom
left
right
It places the widget on the side of the window.
Grid - grid is often used in dialog boxes, and we could place the widgets based on the position coordinates of the grid. grid layout method could have stable relative positions of all widgets. This geometry manager organizes widgets in a table-like structure in the parent widget.

Syntax
widget.grid(grid_options)

Here is the list of possible options −

column − The column to put widget in; default 0 (leftmost column).
columnspan − How many columns widgetoccupies; default 1.
ipadx, ipady − How many pixels to pad widget, horizontally and vertically, inside widget's borders.
padx, pady − How many pixels to pad widget, horizontally and vertically, outside v's borders.
row − The row to put widget in; default the first row that is still empty.
rowspan − How many rowswidget occupies; default 1.
sticky − What to do if the cell is larger than widget. By default, with sticky='', widget is centered in its cell. sticky may be the string concatenation of zero or more of N, E, S, W, NE, NW, SE, and SW, compass directions indicating the sides and corners of the cell to which widget sticks.

Example

```
import Tkinter
root = Tkinter.Tk( )
for r in range(3):
   for c in range(4):
      Tkinter.Label(root, text='R%s/C%s'%(r,c),
        borderwidth=1 ).grid(row=r,column=c)
root.mainloop( )
```

Output

Place:

This geometry manager organizes widgets by placing them in a specific position in the parent widget.

Syntax

widget.place(place_options)

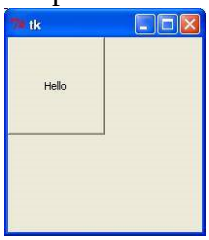
Here is the list of possible options −

1. anchor − The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget)
2. bordermode − INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise.
3. height, width − Height and width in pixels.
4. relheight, relwidth − Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
5. relx, rely − Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.
6. x, y − Horizontal and vertical offset in pixels.

Example:

```
from Tkinter import *
import tkMessageBox
import Tkinter
top = Tkinter.Tk()
def helloCallBack():
   tkMessageBox.showinfo( "Hello Python", "Hello World")
B = Tkinter.Button(top, text ="Hello", command = helloCallBack)
B.pack()
B.place(bordermode=OUTSIDE, height=100, width=100)
top.mainloop()
```

Output



Tkinter Events

Tkinter also lets we set callable to handle a variety of events. Tkinter does not let we create custom events: We are limited to working with events predefined by Tkinter itself.

The Event Object

General event callbacks must accept one argument event that is a Tkinter event object. Such an event object has several attributes that describe the event:

1.char :A single-character string that is the key's code (only for keyboard events)
2. keysym: A string that is the key's symbolic name (only for keyboard events)
3.num:Button number (only for mouse-button events); 1 and up
4. x, y :Mouse position, in pixels, relative to the upper-left corner of the widget
5. x_root y_root: Mouse position, in pixels, relative to the upper-left corner of the screen
6.widget : The widget in which the event has occurred

Bind:

The binding function is used to deal with the events. We can bind Python's Functions and methods to an event as well as we can bind these functions to any particular widget.

- To bind a callback to an event in a widget w, call w.bind and describe the event with a string, usually enclosed in angle brackets ('<...>').

1. Binding mouse movement with tkinter Frame.

```
# Import all files from
# tkinter and overwrite
# all the tkinter files
# by tkinter.ttk
from tkinter import *
from tkinter.ttk import *

# creates tkinter window or root window
root = Tk()
root.geometry('200x100')

# function to be called when mouse enters in a frame
def enter(event):
        print('Button-2 pressed at x = % d, y = % d'%(event.x, event.y))

# function to be called when when mouse exits the frame
def exit_(event):
        print('Button-3 pressed at x = % d, y = % d'%(event.x, event.y))

# frame with fixed geometry
frame1 = Frame(root, height = 100, width = 200)

# these lines are showing the
# working of bind function
# it is universal widget method
```

```
frame1.bind('<Enter>', enter)
frame1.bind('<Leave>', exit_)
frame1.pack()
mainloop()
```

2. Binding Mouse buttons with Tkinter Frame

```
# Import all files from
# tkinter and overwrite
# all the tkinter files
# by tkinter.ttk
from tkinter import *
from tkinter.ttk import *

# creates tkinter window or root window
root = Tk()
root.geometry('200x100')

# function to be called when button-2 of mouse is pressed
def pressed2(event):
        print('Button-2 pressed at x = % d, y = % d'%(event.x, event.y))

# function to be called when button-3 of mouse is pressed
def pressed3(event):
        print('Button-3 pressed at x = % d, y = % d'%(event.x, event.y))

## function to be called when button-1 is double clocked
def double_click(event):
        print('Double clicked at x = % d, y = % d'%(event.x, event.y))

frame1 = Frame(root, height = 100, width = 200)

# these lines are binding mouse
# buttons with the Frame widget
frame1.bind('<Button-2>', pressed2)
frame1.bind('<Button-3>', pressed3)
frame1.bind('<Double 1>', double_click)

frame1.pack()

mainloop()
```

3. Binding keyboard buttons with the root window (tkinter main window).

```
# Import all files from

# tkinter and overwrite
# all the tkinter files
```

```
# by tkinter.ttk
from tkinter import *
from tkinter.ttk import *
# function to be called when
# keyboard buttons are pressed
def key_press(event):
        key = event.char
        print(key, 'is pressed')

# creates tkinter window or root window
root = Tk()
root.geometry('200x100')
# here we are binding keyboard
# with the main window
root.bind('<Key>', key_press)
mainloop()
```

Declarative Programming Paradigm

The Declarative Paradigm is the code should describe what the desired outcome of a program should be, rather than how to obtain the outcome.

- The declarative paradigm focuses on expressing logic without using control flows, i.e., if/else and loop statements.
- Declarative languages often use a style that results in few side-effects, if any. This is usually a natural consequence of not needing to specify explicit steps to solve a problem.

Examples of declarative languages are database query languages such as SQL, markup languages like HTML and CSS, and in functional and logic programming languages.

Side Effects

A function is known to cause a side-effect if it modifies a state that exists outside the immediate function scope.

For example, changing the global state of our program is considered a side effect, in addition to changing the state of a variable passed by reference. I/O operations to read and write from files are also known as side effects.

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, manage, destroy as well as modify their structure and contents.

The language is subdivided into several language elements, including:

- Clauses
- Expressions
- Predicates
- Queries
- Statements

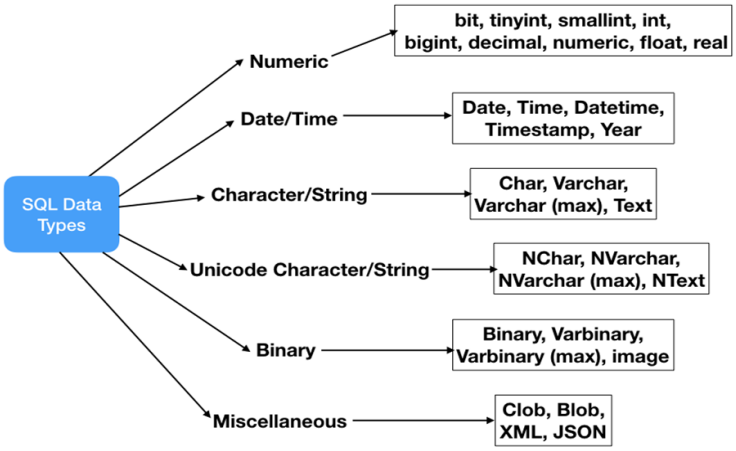
SQL Commands

The SQL commands help in creating and managing the database. The most common SQL commands which are highly used are mentioned below:

1. CREATE command: used to create the new database, new table, table view, and other objects of the database.
2. UPDATE command: used to update or change the stored data in the database.
3. DELETE command : used to remove the saved records from the database tables. It erases single or multiple tuples from the tables of the database.
4. SELECT command : used to access the single or multiple rows from one or multiple tables of the database. We can also use this command with the WHERE clause.
5. DROP command :used to delete the entire table, table view, and other objects from the database
6. INSERT command : used to insert the data or records into the database tables.

SQL Data Types

Data types are used to represent the nature of the data that can be stored in the database table. For example, in a particular column of a table, if we want to store a string type of data then we will have to declare a string data type of this column.



Data types mainly classified into three categories for every database.

1. String Data types
2. Numeric Data types

3. Date and time Data types

String Datatypes

CHAR(Size)	It is used to specify a fixed length string that can contain numbers, letters, and special characters. Its size can be 0 to 255 characters. Default is 1.
VARCHAR(Size)	It is used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters.
BINARY(Size)	It is equal to CHAR() but stores binary byte strings. Its size parameter specifies the column length in the bytes. Default is 1.
VARBINARY(Size)	It is equal to VARCHAR() but stores binary byte strings. Its size parameter specifies the maximum column length in bytes.
TEXT(Size)	It holds a string that can contain a maximum length of 255 characters.
TINYTEXT	It holds a string with a maximum length of 255 characters.
MEDIUMTEXT	It holds a string with a maximum length of 16,777,215.
LONGTEXT	It holds a string with a maximum length of 4,294,967,295 characters.
ENUM(val1, val2, val3,...)	It is used when a string object having only one value, chosen from a list of possible values. It contains 65535 values in an ENUM list. If you insert a value that is not in the list, a blank value will be inserted.
SET(val1,val2,val3,....)	It is used to specify a string that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values at one time in a SET list.
BLOB(size)	It is used for BLOBs (Binary Large Objects). It can hold up to 65,535 bytes.

Numeric Data Types

BIT(Size)	It is used for a bit-value type. The number of bits per value is specified in size. Its size can be 1 to 64. The default value is 1.
INT(size)	It is used for the integer value. Its signed range varies from -2147483648 to 2147483647 and unsigned range varies from 0 to 4294967295. The size parameter specifies the max display width that is 255.
INTEGER(size)	It is equal to INT(size).
FLOAT(size, d)	It is used to specify a floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal point is specified by **d** parameter.

FLOAT(p)	It is used to specify a floating point number. MySQL used p parameter to determine whether to use FLOAT or DOUBLE. If p is between 0 to24, the data type becomes FLOAT (). If p is from 25 to 53, the data type becomes DOUBLE().
DOUBLE(size, d)	It is a normal size floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal is specified by d parameter.
DECIMAL(size, d)	It is used to specify a fixed point number. Its size parameter specifies the total number of digits. The number of digits after the decimal parameter is specified by **d** parameter. The maximum value for the size is 65, and the default value is 10. The maximum value for **d** is 30, and the default value is 0.
DEC(size, d)	It is equal to DECIMAL(size, d).
BOOL	It is used to specify Boolean values true and false. Zero is considered as false, and nonzero values are considered as true.

MySQL Date and Time Data Types

DATE	It is used to specify date format YYYY-MM-DD. Its supported range is from '1000-01-01' to '9999-12-31'.
DATETIME(fsp)	It is used to specify date and time combination. Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1000-01-01 00:00:00' to 9999-12-31 23:59:59'.
TIMESTAMP(fsp)	It is used to specify the timestamp. Its value is stored as the number of seconds since the Unix epoch('1970-01-01 00:00:00' UTC). Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
TIME(fsp)	It is used to specify the time format. Its format is hh:mm:ss. Its supported range is from '-838:59:59' to '838:59:59'
YEAR	It is used to specify a year in four-digit format. Values allowed in four digit format from 1901 to 2155, and 0000.

SELECT Statement

This SQL statement reads the data from the SQL database and shows it as the output to the database user.

Syntax of SELECT Statement:

```
SELECT column_name1, column_name2, …., column_nameN
   [ FROM table_name ]
   [ WHERE condition ]
   [ ORDER BY order_column_name1 [ ASC | DESC ], .... ];
```

Example of SELECT Statement:

```
SELECT Emp_ID, First_Name, Last_Name, Salary, City
FROM Employee_details
WHERE Salary = 100000
ORDER BY Last_Name
```

This example shows the Emp_ID, First_Name, Last_Name, Salary, and City of those employees from the Employee_details table whose Salary is 100000. The output shows all the specified details according to the ascending alphabetical order of Last_Name.

UPDATE Statement

This SQL statement changes or modifies the stored data in the SQL database.

Syntax of UPDATE Statement:

```
UPDATE table_name
SET column_name1 = new_value_1, column_name2 = new_value_2, ...., column_nameN = new_value_N
[ WHERE  CONDITION ];
```

Example of UPDATE Statement:

```
UPDATE Employee_details
SET Salary = 100000
WHERE Emp_ID = 10;
```

This example changes the Salary of those employees of the Employee_details table whose Emp_ID is 10 in the table.

3. DELETE Statement

This SQL statement deletes the stored data from the SQL database.

Syntax of DELETE Statement:

```
DELETE FROM table_name
[ WHERE CONDITION ];
```

Example of DELETE Statement:

```
DELETE FROM Employee_details
WHERE First_Name = 'Sumit';
```

This example deletes the record of those employees from the Employee_details table whose First_Name is Sumit in the table.

CREATE TABLE Statement

This SQL statement creates the new table in the SQL database.

Syntax of CREATE TABLE Statement:

```
CREATE TABLE table_name
(
column_name1 data_type [column1 constraint(s)],
column_name2 data_type [column2 constraint(s)],
.....
.....,
column_nameN data_type [columnN constraint(s)],
PRIMARY KEY(one or more col)
);
```

Example of CREATE TABLE Statement:

```
CREATE TABLE Employee_details(
    Emp_Id NUMBER(4) NOT NULL,
    First_name VARCHAR(30),
    Last_name VARCHAR(30),
    Salary Money,
    City VARCHAR(30),
    PRIMARY KEY (Emp_Id)
);
```

This example creates the table Employee_details with five columns or fields in the SQL database. The fields in the table are Emp_Id, First_Name, Last_Name, Salary, and City. The Emp_Id column in the table acts as a primary key, which means that the Emp_Id column cannot contain duplicate values and null values.

5. ALTER TABLE Statement

This SQL statement adds, deletes, and modifies the columns of the table in the SQL database.

Syntax of ALTER TABLE Statement:

```
ALTER TABLE table_name ADD column_name datatype[(size)];
```

The above SQL alter statement adds the column with its datatype in the existing database table.

```
ALTER TABLE table_name MODIFY column_name column_datatype[(size)];
```

The above 'SQL alter statement' renames the old column name to the new column name of the existing database table.

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

The above SQL alter statement deletes the column of the existing database table.

Example of ALTER TABLE Statement:

ALTER TABLE Employee_details

ADD Designation VARCHAR(18);

This example adds the new field whose name is Designation with size 18 in the Employee_details table of the SQL database.

6. DROP TABLE Statement

This SQL statement deletes or removes the table and the structure, views, permissions, and triggers associated with that table.

Syntax of DROP TABLE Statement:

DROP TABLE [IF EXISTS]

table_name1, table_name2, ……, table_nameN;

The above syntax of the drop statement deletes specified tables completely if they exist in the database.

Example of DROP TABLE Statement:

DROP TABLE Employee_details;

This example drops the Employee_details table if it exists in the SQL database. This removes the complete information if available in the table.

7. CREATE DATABASE Statement

This SQL statement creates the new database in the database management system.

Syntax of CREATE DATABASE Statement:

CREATE DATABASE database_name;

Example of CREATE DATABASE Statement:

CREATE DATABASE Company;

The above example creates the company database in the system.

8. DROP DATABASE Statement

This SQL statement deletes the existing database with all the data tables and views from the database management system.

Syntax of DROP DATABASE Statement:

DROP DATABASE database_name;

Example of DROP DATABASE Statement:

DROP DATABASE Company;

The above example deletes the company database from the system.

9. INSERT INTO Statement

This SQL statement inserts the data or records in the existing table of the SQL database. This statement can easily insert single and multiple records in a single query statement.

Syntax of insert a single record:

```
INSERT INTO table_name
(   column_name1, column_name2, ….,   column_nameN  )
VALUES
(value_1,  value_2, ..…,  value_N  );
```

Example of insert a single record:

```
INSERT INTO Employee_details
(  Emp_ID, First_name,  Last_name,  Salary,  City  )
VALUES
(101,   Akhil,  Sharma,  40000,  Bangalore  );
```

This example inserts 101 in the first column, Akhil in the second column, Sharma in the third column, 40000 in the fourth column, and Bangalore in the last column of the table Employee_details.

Syntax of inserting a multiple records in a single query:

```
INSERT INTO table_name
( column_name1, column_name2, …., column_nameN)
VALUES (value_1, value_2, ..…, value_N), (value_1, value_2, ..…, value_N),….;
```

Example of inserting multiple records in a single query:

```
INSERT INTO Employee_details
( Emp_ID, First_name, Last_name, Salary, City )
VALUES (101, Amit, Gupta, 50000, Mumbai), (101,  John, Aggarwal, 45000, Calcutta), (101, Sidhu, Arora, 55000, Mumbai);
```

This example inserts the records of three employees in the Employee_details table in the single query statement.

10. TRUNCATE TABLE Statement

This SQL statement deletes all the stored records from the table of the SQL database.

Syntax of TRUNCATE TABLE Statement:

```
TRUNCATE TABLE table_name;
```

Example of TRUNCATE TABLE Statement:

```
TRUNCATE TABLE Employee_details;
```

This example deletes the record of all employees from the Employee_details table of the database.

DESCRIBE Statement

This SQL statement tells something about the specified table or view in the query.

Syntax of DESCRIBE Statement:

DESCRIBE table_name | view_name;

Example of DESCRIBE Statement:

DESCRIBE Employee_details;

This example explains the structure and other details about the Employee_details table.

12. DISTINCT Clause

This SQL statement shows the distinct values from the specified columns of the database table. This statement is used with the SELECT keyword.

Syntax of DISTINCT Clause:

SELECT DISTINCT column_name1, column_name2, ...

FROM table_name;

Example of DISTINCT Clause:

SELECT DISTINCT City, Salary

FROM Employee_details;

This example shows the distinct values of the City and Salary column from the Employee_details table.

13. COMMIT Statement

This SQL statement saves the changes permanently, which are done in the transaction of the SQL database.

Syntax of COMMIT Statement:

COMMIT

Example of COMMIT Statement:

DELETE FROM Employee_details

WHERE salary = 30000;

COMMIT;

This example deletes the records of those employees whose Salary is 30000 and then saves the changes permanently in the database.

14. ROLLBACK Statement

This SQL statement undo the transactions and operations which are not yet saved to the SQL database.

Syntax of ROLLBACK Statement:

ROLLBACK

Example of ROLLBACK Statement:

```
DELETE FROM Employee_details
WHERE City = Mumbai;
ROLLBACK;
```

This example deletes the records of those employees whose City is Mumbai and then undo the changes in the database.

15. CREATE INDEX Statement

This SQL statement creates the new index in the SQL database table.

Syntax of CREATE INDEX Statement:

```
CREATE INDEX index_name
ON table_name ( column_name1, column_name2, …, column_nameN );
```

Example of CREATE INDEX Statement:

```
CREATE INDEX idx_First_Name
ON employee_details (First_Name);
```

This example creates an index idx_First_Name on the First_Name column of the Employee_details table.

16. DROP INDEX Statement

This SQL statement deletes the existing index of the SQL database table.

Syntax of DROP INDEX Statement:

```
DROP INDEX index_name;
```

Example of DROP INDEX Statement:

```
DROP INDEX idx_First_Name;
```

This example deletes the index idx_First_Name from the SQL database.

17. USE Statement

This SQL statement selects the existing SQL database. Before performing the operations on the database table, you have to select the database from the multiple existing databases.

Syntax of USE Statement:

```
USE database_name;
```

Example of USE DATABASE Statement:

```
USE Company;
```

This example uses the company database.

Creating Events without describing the flow of execution

SQL | Triggers

Trigger is a statement that a system executes automatically when there is any modification to the database. In a trigger, we first specify when the trigger is to be executed and then the action to be performed when the trigger executes. Triggers are used to specify certain integrity constraints and referential constraints that cannot be specified using the constraint mechanism of SQL.

Example –

Suppose, we are adding a tuple to the 'Donors' table that is some person has donated blood. So, we can design a trigger that will automatically add the value of donated blood to the 'Blood_record' table.

Types of Triggers –

We can define 6 types of triggers for each table:

1. AFTER INSERT activated after data is inserted into the table.
2. AFTER UPDATE: activated after data in the table is modified.
3. AFTER DELETE: activated after data is deleted/removed from the table.
4. BEFORE INSERT: activated before data is inserted into the table.
5. BEFORE UPDATE: activated before data in the table is modified.
6. BEFORE DELETE: activated before data is deleted/removed from the table.

Examples showing implementation of Triggers:

1. Write a trigger to ensure that no employee of age less than 25 can be inserted in the database.

```
delimiter $$
CREATE TRIGGER  Check_age  BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
IF NEW.age < 25 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'ERROR:
      AGE MUST BE ATLEAST 25 YEARS!';
END IF;
END; $$
delimiter;
```

Explanation: Whenever we want to insert any tuple to table 'employee', then before inserting this tuple to the table, trigger named 'Check_age' will be executed. This trigger will check the age attribute. If it is greater then 25 then this tuple will be

inserted into the tuple otherwise an error message will be printed stating “ERROR: AGE MUST BE ATLEAST 25 YEARS!”
2. Create a trigger which will work before deletion in employee table and create a duplicate copy of the record in another table employee_backup.

Before writing trigger, we need to create table employee_backup.

```
create table employee_backup (employee_no int,
    employee_name varchar(40), job varchar(40),
    hiredate date, salary int,
    primary key(employee_no));
delimiter $$
CREATE TRIGGER Backup BEFORE DELETE ON employee
FOR EACH ROW
BEGIN
INSERT INTO employee_backup
VALUES (OLD.employee_no, OLD.name,
      OLD.job, OLD.hiredate, OLD.salary);
END; $$
delimiter;
```

Explanation: We want to create a backup table that holds the value of those employees who are no more the employee of the institution. So, we create a trigger named Backup that will be executed before the deletion of any Tuple from the table employee. Before deletion, the values of all the attributes of the table employee will be stored in the table employee_backup.

3. Write a trigger to count number of new tuples inserted using each insert statement.

```
Declare count int
Set count=0;
delimiter $$
CREATE TRIGGER Count_tupples
        AFTER INSERT ON employee
FOR EACH ROW
BEGIN
SET count = count + 1;
END; $$
delimiter;
```

Explanation: We want to keep track of the number of new Tuples in the employee table. For that, we first create a variable ‘count’ and initialize it to 0. After that, we create a trigger named Count_tuples that will increment the value of count after insertion of any new Tuple in the table employee.

HTML, CSS as Declarative Programming Paradigm

Using CSS
CSS can be added to HTML documents in 3 ways:

Inline - by using the style attribute inside HTML elements
Internal - by using a <style> element in the <head> section
External - by using a <link> element to link to an external CSS file
The most common way to add CSS, is to keep the styles in external CSS files. However, in this tutorial we will use inline and internal styles, because this is easier to demonstrate, and easier for you to try it yourself.

Inline CSS
An inline CSS is used to apply a unique style to a single HTML element.

An inline CSS uses the style attribute of an HTML element.

The following example sets the text color of the <h1> element to blue, and the text color of the <p> element to red:

Example
```
<h1 style="color:blue;">A Blue Heading</h1>

<p style="color:red;">A red paragraph.</p>
```

Internal CSS
An internal CSS is used to define a style for a single HTML page.

An internal CSS is defined in the <head> section of an HTML page, within a <style> element.

The following example sets the text color of ALL the <h1> elements (on that page) to blue, and the text color of ALL the <p> elements to red. In addition, the page will be displayed with a "powderblue" background color:

Example
```
<!DOCTYPE html>
<html>
<head>
<style>
body {background-color: powderblue;}
h1   {color: blue;}
p    {color: red;}
</style>
</head>
<body>

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

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

External CSS

An external style sheet is used to define the style for many HTML pages.

To use an external style sheet, add a link to it in the <head> section of each HTML page:

Example

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

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

</body>
</html>
```

The external style sheet can be written in any text editor. The file must not contain any HTML code, and must be saved with a .css extension.

Here is what the "styles.css" file looks like:

"styles.css":

```
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```

Imperative Programming Paradigm

Imperative programming is a programming paradigm that uses statements that change a program's state. An imperative program consists of commands for the computer to perform.

A program based on this paradigm is made up of a clearly-defined sequence of instructions to a computer.

Therefore, the source code for imperative languages is a series of commands, which specify what the computer has to do – and when – in order to achieve a desired result. Values used in

variables are changed at program runtime. To control the commands, control structures such as loops or branches are integrated into the code.

Imperative programming languages are very specific, and operation is system-oriented. On the one hand, the code is easy to understand; on the other hand, many lines of source text are required to describe what can be achieved with a fraction of the commands using declarative programming languages.

- In a computer program, a variable stores the data. The contents of these locations at any given point in the program's execution are called the program's state. Imperative programming is characterized by programming with state and commands which modify the state.
- The first imperative programming languages were machine languages.
- Machine Language: Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory.

  MOV AL, BL

  MOV A, B

These are the best-known imperative programming languages:

Fortran,Java,Pascal,ALGOL,C,C#,C++,Assembler,BASIC,COBOL,Python,Ruby

The different imperative programming languages can, in turn, be assigned to three further subordinate programming styles – structured, procedural, and modular.

Advantages:

- Easy to read
- Relatively easy to learn
- Conceptual model (solution path) is very easy for beginners to understand
- Characteristics of specific applications can be taken into account.

Disadvantages

- Code quickly becomes very extensive and thus confusing
- Higher risk of errors when editing
- System-oriented programming means that maintenance blocks application development
- Optimization and extension are more difficult

Characteristics of the Imperative Paradigm

The definition of the imperative paradigm describes using statements and variables as the tools on our utility belt to write useful programs.

Mutable Variables and Assignment

Variables are the key element of imperative paradigm. Collectively, they make up what is known as the state of a program. Some variables have the ability to change their value after they've been set to an initial value. These variables are called mutable variables, which variables with values that cannot be changed once set are called immutable variables.

Consider the following program:

```
var first = 5
var second = 17
var result
result = first + second
print(result)
```

When we want a new variable within our program, we need to declare that variable. In our program, we've declared the variables first and second, and assigned their values to be 5, and 17 by using the assignment operator =. In common programming speech, we can say we've initialized a variable when we assign its value for the first time.

Now that we've covered the basics of variable declaration and assignment, consider the following:

```
var first = 5
var second
first + 5 = second // Error!
```

Clearly this code is wrong, but why? At first glance, it might appear that we're taking the output of first + 5 and assigning it to second, but this isn't so.

To understand why this code is wrong, we need understand the difference between what are known as l-values and r-values.

L-Values, R-Values, and the Assignment Operator

The terms l-value and r-value get their names because they appear on the left and right side of the assignment operator. In other words, a proper assignment must look like this:

```
l-value = r-value
```

This is simple enough, but it doesn't tell us much about what and l-value or an r-value is. To get a better grasp, we need to peel back the layers of our computer and take a look at memory.

If we look at memory in a simplified way, it is essentially a long line of compartments, known as addresses that contain values. Below are four addresses, with the values that currently reside within them:

Address	0x0000	0x0001	0x0002	0x0003
Value (binary)	0110 1001	1100 0010	1101 1111	1001 0010

Some types of data need more than 1 byte to express their possible values. For instance, an integer typically needs 4 bytes of memory to express whole numbers ranging from -2,147,483,648 through 2,147,483,648. If we wanted bigger numbers, we would need to use more memory, or a different way to represent the numbers, like floating point notation. Now that we have a basic understanding of the structure of memory, we can start to demystify l-values and r-values.

An l-value is a variable or pointer that refers to a specific, modifiable address in memory which we intend to write an r-value into. L-values appear on the left-hand side of the assignment operator. Some parts of memory are designated as non-modifiable, like when they being used by immutable variables, which is why we need this distinction.

An r-value is an expression that can be evaluated to a plain value in order to be read into the l-value. It is important to know that a variable or pointer can be both r-values and l-values. In other words, whether or not a variable is an l-value or an r-value depends on the context that it is being used in. Let's practice with some examples:

```
var counter // counter is an l-value
counter = 5 // counter is an l-value, 5 is an r-value.
// ...
counter = counter + 1 // counter is an l-value, AND an r-value!
```

How is it that counter is both an l-value and an r-value? This is where context is key! When we're declaring the variable and assigning it a value, it is an l-value because we are assigning the value 5 to the place in memory that the variable owns. However, on line 3 we need to read the value at the address that counter owns, evaluate the expression 5 + 1, and then finally write the result into the l-value.

Recall our code from earlier that was troublesome:

```
var first = 5
var second
first + 5 = second
```

With our new knowledge of l-values and r-values, we can see that there are actually two problems with the code on line 3:

second is an r-value and has not been initialized, so we can't read that value into the left-hand side.

More importantly, first + 5 doesn't refer to any modifiable point in memory, so it isn't a valid l-value!

Pay close attention to the phrase "doesn't refer to any modifiable point in memory", because it holds a subtle clue that often trips up many programmers, and it has to do with pointers.

If you're unfamiliar with pointers, or have never used languages like C or C++, this may be something to come back to later. Nonetheless, suppose we're writing a program in C and we have the following code:

```
int *first = 5; // *first is an l-value, 5 is an r-value
int *second = 10; //...

*(first + 1) = second // This is valid (but weird looking) code!
```

Wait a minute, how can we have an expression like that on the left-hand side when in the example before we couldn't? This has to do once again with context. The expression *(first + 1) evaluates to the address 4 bytes* ahead of the variable first, which we dereference in order to store a value at the address. When we evaluate this expression, we end up with a reference to a point in memory! Because it is referring to a place in memory, it is a valid l-value!

The reason it moves 4 bytes instead of 1 byte is because the size of an int is 4 bytes, and pointer arithmetic multiplies the number by the size of the datatype.

Imperative Programming	Declarative Programming
In this, programs specify how it is to be done.	In this, programs specify what is to be done.
It simply describes the control flow of computation.	It simply expresses the logic of computation.
Its main goal is to describe how to get it or accomplish it.	Its main goal is to describe the desired result without direct dictation on how to get it.
Its advantages include ease to learn and read, the notional model is simple to understand, etc.	Its advantages include effective code, which can be applied by using ways, easy extension, high level of abstraction, etc.
Its type includes procedural programming, object-oriented programming, parallel processing approach.	Its type includes logic programming and functional programming.
In this, the user is allowed to make decisions and commands to the compiler.	In this, a compiler is allowed to make decisions.
It has many side effects and includes mutable variables as compared to declarative programming.	It has no side effects and does not include any mutable variables as compared to imperative programming.
It gives full control to developers that are very important in low-level programming.	It may automate repetitive flow along with simplifying code structure.
Declarative Programs are *context-independent*. Because they only declare what the ultimate goal is, but not the intermediary steps to reach that goal, the same program can be used in different contexts.	This is hard to do with *imperative programs*, because they often depend on the context (e.g. hidden state).