Advanced
Python for Beginners

Table of content

- Tkinter
- MySQL
- Visual Studio – Flask
- HTML
- Kivy

Sl No	Type		
1	GUI	Tkinter	
2	Database	MySQL	
3	Web development	Flask	
4	Mobile app	Kivy	

Tkinter

Tkinter is Python's standard library for creating Graphical User Interfaces (GUIs). It is used to develop desktop applications with graphical elements, providing a way to build interactive programs that users can interact with visually, rather than solely through a command-line interface.

- **Creating Windows and Dialog Boxes:**
Tkinter allows the creation of main application windows and various types of dialog boxes (e.g., message boxes, input dialogs).
- **Managing Layouts:**
Tkinter offers tools to arrange and position widgets within a window, ensuring a well-organized and visually appealing interface.
- **Handling Events:**
It enables the association of actions with user interactions, such as button clicks, keyboard presses, or window resizing.
- **Developing Simple to Moderately Complex Desktop Applications:**
Tkinter is suitable for a wide range of applications, from basic utilities to more involved programs requiring a graphical front-end.

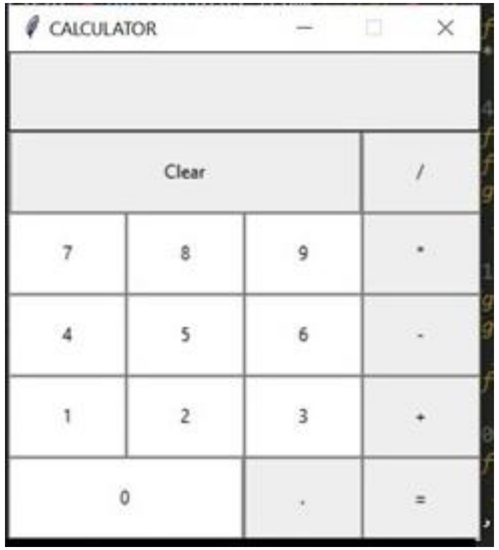
<u>Key advantages of using Tkinter:</u>

- **Included with Python:**
It comes as part of the standard Python distribution, eliminating the need for separate installation.
- **Ease of Use:**
It is relatively straightforward to learn and use, making it a popular choice for beginners in GUI programming.
- **Cross-Platform Compatibility:**
Applications built with Tkinter can run on various operating systems, including Windows, macOS, and Linux, with minimal code changes.

Its inbuilt software along with python

```
import tkinter as tk

def say_hello():
print("Hello from the button!")

root = tk.Tk()
root.title("Simple Tkinter Example")

label = tk.Label(root, text="Welcome!")
label.pack(pady=10) # Add some padding

button = tk.Button(root, text="Greet", command=say_hello)
button.pack()

root.mainloop()
```

- **Adding GUI Widgets:**
It provides a collection of pre-built "widgets" that can be added to windows, such as:
  - **Buttons:** For triggering actions.
  - **Labels:** For displaying static text.
  - **Entry fields:** For user input.
  - **Text areas:** For multi-line text input or display.
  - **Checkboxes and Radio buttons:** For selection options.
  - **Menus:** For navigation and options.
  - **Scrollbars, Sliders, Listboxes, etc.**



Tkinter Widget

Sl No	Types	Syntex	Example
	Label	*w=Label(master, option=value)*	w = Label(root, text='GeeksForGeeks.org!') tk.Label(w, text="8 is not a prime number!", font=("Arial", 18), bg="#f0f8ff")
	Button	*w=Button(master, option=value)*	button = tk.Button(r, text='Stop', width=25, command=r.destroy)
	CheckButton	*w = CheckButton(master, option=value)*	Checkbutton(master, text='male', variable=var1).grid(row=0, sticky=W)
	RadioButton	*w = RadioButton(master, option=value)*	Radiobutton(root, text='GfG', variable=v, value=1).pack(anchor=W)
	Listbox	*w = Listbox(master, option=value)*	Lb.insert(1, 'Python')

	Scrollbar	*w = Scrollbar(master, option=value)*	scrollbar = Scrollbar(root)
			scrollbar.pack(side=RIGHT, fill=Y)
			mylist = Listbox(root, yscrollcommand=scrollbar.set)

	Menu	*window.w = Menu(master, option=value)*	filemenu = Menu(menu)
			menu.add_cascade(label='File', menu=filemenu)
			filemenu.add_command(label='New')

	Entry	*w=Entry(master, option=value)*	Label(master, text='First Name').grid(row=0)

Contd…

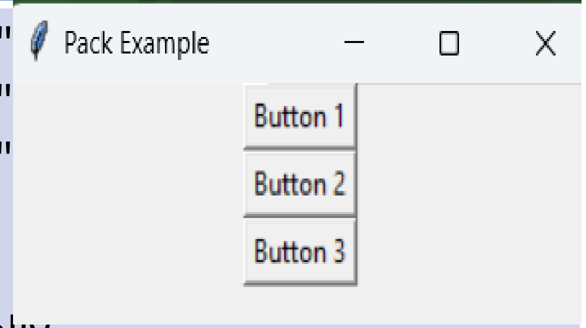
Sl No	Types	Syntex	Example
	Combobox	*combo = Combobox(master, values=[...], state='readonly')*	combo_box = ttk.Combobox(root, values=["Option 1", "Option 2", "Option 3"])

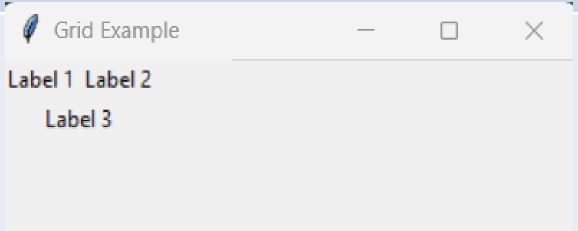
	Scale	*w = Scale(master, option=value)*	w = Scale(master, from_=0, to=42)
			w = Scale(master, from_=0, to=200, orient=HORIZONTAL)

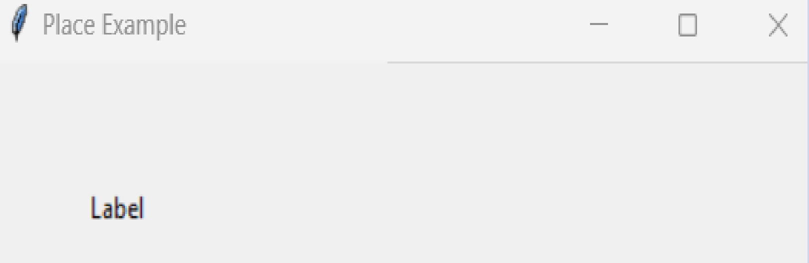
	TopLevel	*w = TopLevel(master, option=value)*	top = Toplevel()
	Message	*w = Message(master, option=value)*	messageVar = Message(main, text=ourMessage)
	MenuButton	*w = MenuButton(master, option=value)*	mb = Menubutton (top, text = "GfG")
	Progressbar	*Progressbar(parent, orient, length, mode)*	
	SpinBox	*w = SpinBox(master, option=value)*	w = Spinbox(master, from_=0, to=10)
	Text	*w =Text(master, option=value)*	T = Text(root, height=2, width=30)
	Canvas	*w = Canvas(master, option=value)*	w = Canvas(master, width=40, height=60)
	PannedWindow	*w = PannedWindow(master, option=value)*	m2 = PanedWindow(m1, orient=VERTICAL)

Tkinter Geometry Managers

Sl No	Types	Syntex	Example

	pack()	It organizes the widgets in blocks before placing in the parent widget. Widgets can be packed from the top, bottom, left or right. It can expand widgets to fill the available space or place them in a fixed size.	button1 = tk.Button(root, text="Button 1"
			button2 = tk.Button(root, text="Button 2"
			button3 = tk.Button(root, text="Button 3"
			button1.pack()
			button2.pack()
			button3.pack() # Pack the buttons vertically
			

	grid()	It organizes the widgets in grid (table-like structure) before placing in the parent widget. Each widget is assigned a row and column. Widgets can span multiple rows or columns using rowspan and columnspan.	label1 = tk.Label(root, text="Label 1")
			label2 = tk.Label(root, text="Label 2")
			label3 = tk.Label(root, text="Label 3")
			label1.grid(row=0, column=0)
			label2.grid(row=0, column=1) *# Grid the labels in a 2x2 grid*
			label3.grid(row=1, column=0, columnspan=2)
			

	place()	It organizes the widgets by placing them on specific positions directed by the programmer. Widgets are placed at specific x and y coordinates. Sizes and positions can be specified in absolute or relative terms.	
			
			root.title("Place Example")
			label = tk.Label(root, text="Label") # Create a label
			label.place(x=50, y=50) # Place the label at specific coordinates

SQL

SQL stands for **Structured Query Language** which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL was developed in the 1970s by IBM, it is a language to operate databases. It includes Database Creation, Database Deletion, Fetching Data Rows, Modifying & Deleting Data rows, etc.

RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

the most popular RDBMS are listed below –

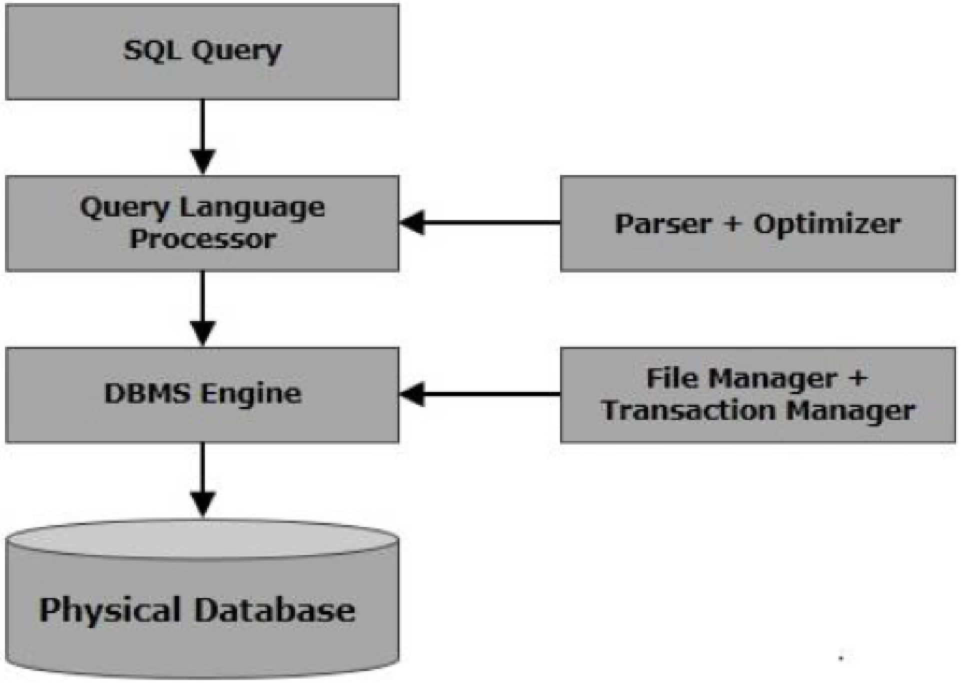
⍰ MySQL
⍰ MS SQL Server
⍰ ORACLE
⍰ MS ACCESS
⍰ PostgreSQL
⍰ SQLite

SQL Applications

SQL is one of the most widely used Query Language over the databases.

SQL provides following functionality to the database programmers −

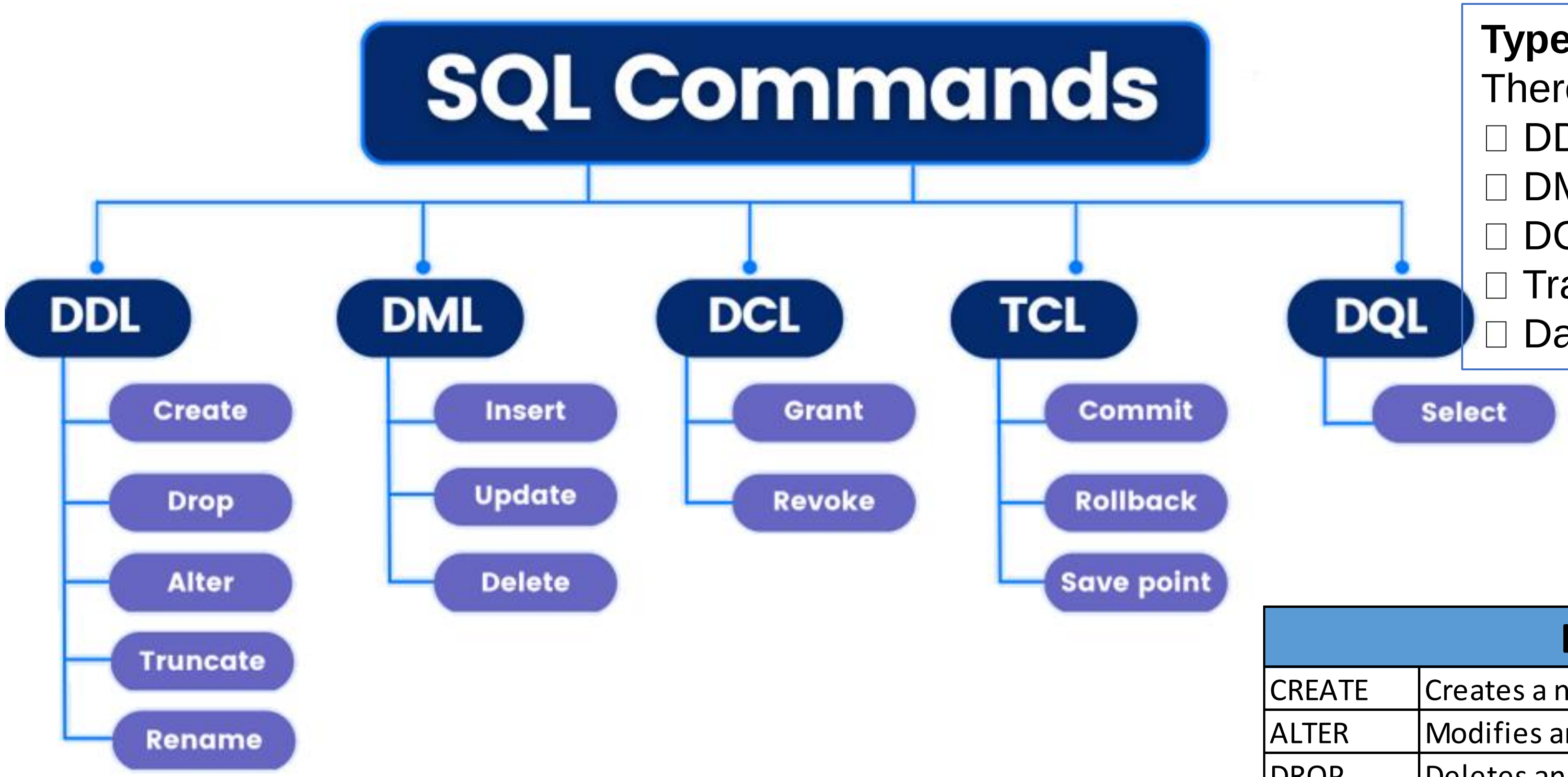
□ Executes different database queries against a database.
□ Defines the data in a database and manipulates that data.
□ Creates data in a relational database management system.
□ Accesses data from the relational database management system.
□ Creates and drops databases and tables.
□ Creates and maintains database users.
□ Creates views, stored procedures, functions in a database.
□ Sets permissions on tables, procedures and views.



Types of Data

	Structured data	Semi-Structured data	Unstructured data
Characteristics	Defined is well organized,	Data is organized to some extent	Data is fully non organized
	Organized means – relational database,	Partially organized, e.g by XML/RDF	Based on character and binary data
	Matured transaction, multiple concurrency techniques	Transaction is adapted from DBMS, but data concurrency can pose problems	Difficult but achievable transaction management and data concurrency
	Tuples, rows and tables	Tuples or graphs are possible	Versioning usually on whole data or chunks
	Schema dependent and less flexible	Data is more flexible than structured	The most flexible
	Query performance is the highest, structed query can be performed allowing complex joins	Queries over anonymous nodes are possible	Schema on-read so query performance is the lowest
Examples	Transactional information, Names, Dates and Addresses •Survey •Questionnaires •Tests •Claim Forms	XML/JSON Data, HTML, Emails, Web pages •Invoices •Purchase Orders •Bills of Lading •Explanation of Benefits	Documents - PDFs, Text files •ViContracts •Letters •Articles •Memos deos, Audio, Images files,

SQL Commands



Types of SQL commands

There are 5 main types of commands Listed below:

□ DDL (Data Definition Language) commands

□ DML (Data Manipulation Language) commands

□ DCL (Data Control Language) commands

□ Transaction Control Language(TCL) commands

□ Data Query Language(DQL) commands

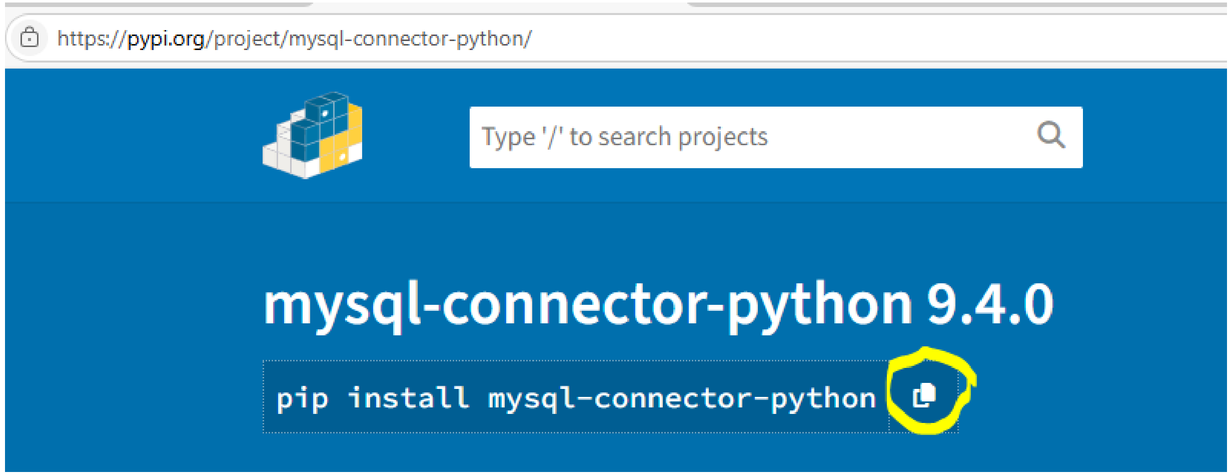
Transaction Control Language(TCL) commands	
START TRANSACTION / BEGIN:	All subsequent DML (Data Manipulation Language) statements
COMMIT:	committed, the changes are irreversible and visible to other transactions.
ROLLBACK	changes made within the current transaction that have not yet been committed
SAVEPOINT	This allows for partial rollbacks, where you can undo changes only up to a specific
ROLLBACK TO SAVEPOINT	This command rolls back the transaction to the specified savepoint

Data Definition Language (DDL)	
CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects in the database.
TRUNCATE	Truncates the entire table in a go.

Data Manipulation Language (DML)	
SELECT	Retrieves certain records from one or more tables.
INSERT	Creates a record.
UPDATE	Modifies records.
DELETE	Deletes records.

Data Control Language (DCL)	
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user.

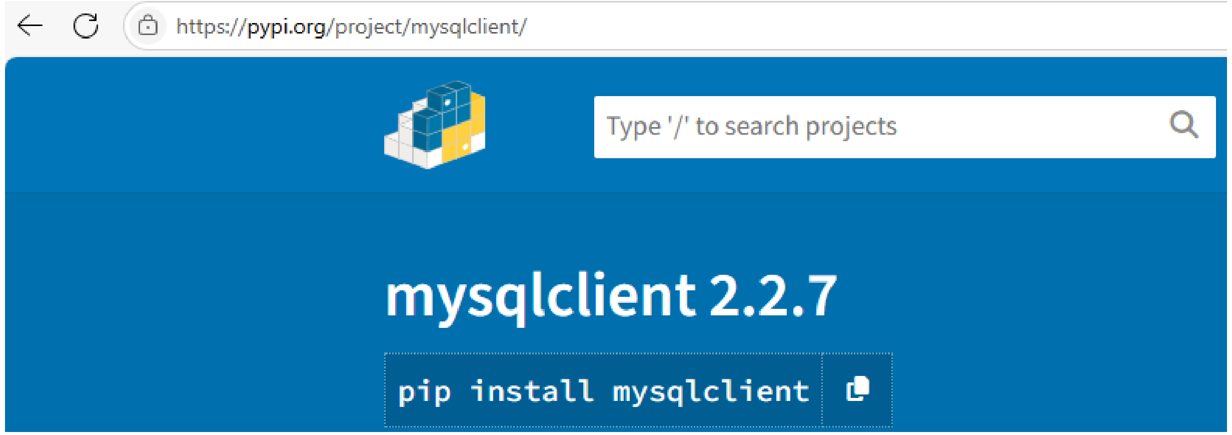
Install and Server connect



https://pypi.org/project/mysql-connector-python/
Type '/' to search projects

mysql-connector-python 9.4.0

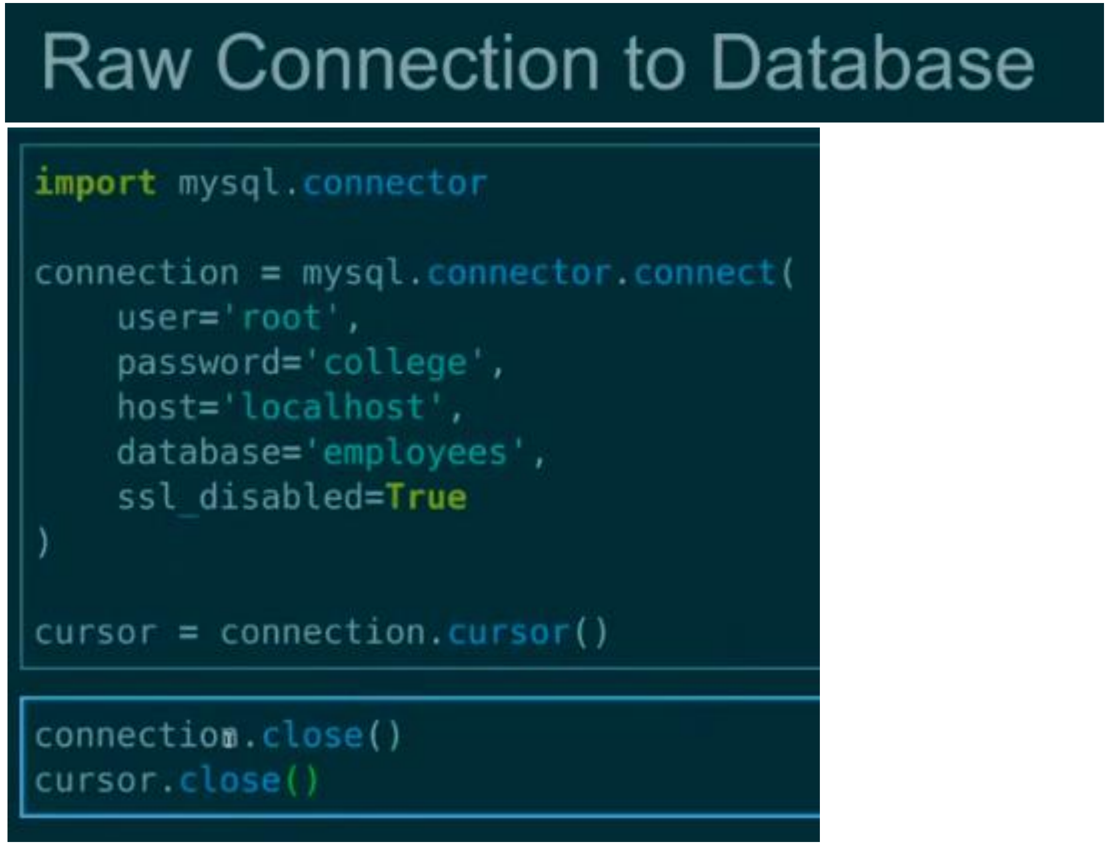
```
pip install mysql-connector-python
```



https://pypi.org/project/mysqlclient/
Type '/' to search projects

mysqlclient 2.2.7

```
pip install mysqlclient
```



Raw Connection to Database

```
import mysql.connector

connection = mysql.connector.connect(
    user='root',
    password='college',
    host='localhost',
    database='employees',
    ssl_disabled=True
)

cursor = connection.cursor()
```

```
connection.close()
cursor.close()
```

Visual Studio – Flask python

Download Visual Studio Code - Mac, Linux, Windows

Create a Project Folder:

Create a Virtual Environment (Recommended):

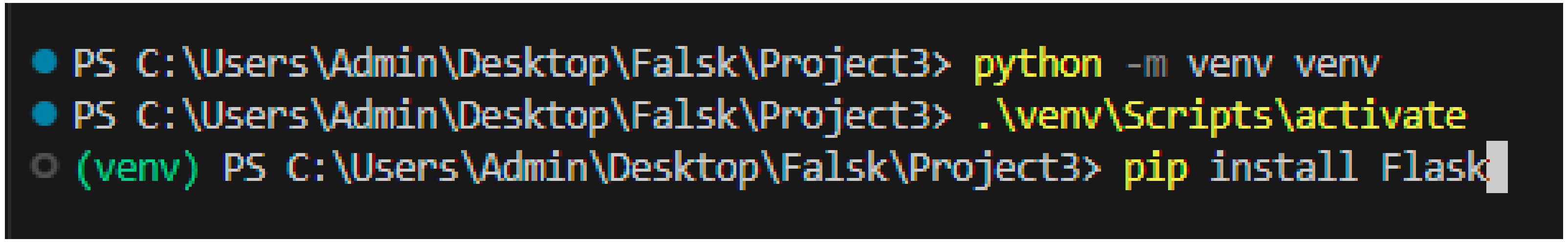
- Open the integrated terminal in VS Code (Terminal > New Terminal or Ctrl+Shift+`).
- Create a virtual environment: `python -m venv venv`
- Activate the virtual environment: `.\venv\Scripts\activate`

Install Flask: `pip install Flask`

Create app.py:

- In your project folder, create a new file named app.py.

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def             ():
    return



if __name__ == "__main__":
    app.run(debug=True)
```



```
● PS C:\Users\Admin\Desktop\Falsk\Project3> python -m venv venv
● PS C:\Users\Admin\Desktop\Falsk\Project3> .\venv\Scripts\activate
○ (venv) PS C:\Users\Admin\Desktop\Falsk\Project3> pip install Flask
```

Visual Studio - Flask

01	render_template	This function is used to render HTML templates and send them as a response to the client's browser. It typically works with a templating engine like Jinja2 (default in Flask). You can pass variables from your Python code to the template for dynamic content generation.	
02	request	This object provides access to incoming request data, such as form data, query parameters, headers, and JSON payloads. It allows your application to interact with the data sent by the client.	
03	redirect	This function is used to redirect the client's browser to a different URL. It is often used after a successful form submission or to navigate to a different part of the application.	
04	url_for	This function dynamically generates a URL for a given endpoint (view function name). It helps in maintaining clean and flexible URLs, as you don't hardcode paths. It can also include variable parts of the URL.	
05	json	When dealing with APIs or AJAX requests, you might need to send and receive data in JSON format. The jsonify function (from Flask) converts Python dictionaries or lists into JSON responses, setting the appropriate Content-Type header. request.json can be used to parse incoming JSON data from the request body.	

HTML

<title>
<h> headings
<p> paragraphs
<a> links

2. HTML Syntax:

HTML (Hypertext Markup Language) is the standard markup language for creating web pages. Its syntax involves:

- **Tags:**
Enclosed in angle brackets (e.g., <p>, <h1>, <div>), often paired with closing tags (e.g., </p>).
- **Elements:**
Consist of an opening tag, content, and a closing tag.
- **Attributes:**
Provide additional information about elements, placed within the opening tag (e.g., <a href="url">).
- **Structure:**
Documents begin with <!DOCTYPE html> and are structured with <html>, <head>, and <body> tags.

```
html_content = """
<!DOCTYPE html>
<html>
<head>
      <title>My Python-Generated Page</title>
</head>
<body>
      <h1>Hello from Python!</h1>
      <p>This paragraph was created by a Python script.</p>
      <a href="https://www.google.com" target="_blank"></a>
 </body>
</html>
"""
with open("index.html", "w") as f:
f.write(html_content)
```

```
<h2>Tuple Data</h2>
   <p>{{ my_tuple[0] }}, {{ my_tuple[1] }}, {{ my_tuple[2] }}</p>

<h2>List Data</h2>
   <p>{{ my_list[0] }}, {{ my_list[1] }}, {{ my_list[2] }}</p>

<h2>Dictionary Data</h2>
   <p>ID: {{ my_dict["id"] }}</p>
```

HTML

- <input type="button">
- <input type="checkbox">
- <input type="color">
- <input type="date">
- <input type="datetime-local">
- <input type="email">
- <input type="file">
- <input type="hidden">
- <input type="image">
- <input type="month">
- <input type="number">
- <input type="password">
- <input type="radio">
- <input type="range">
- <input type="reset">
- <input type="search">
- <input type="submit">
- <input type="tel">
- <input type="text">
- <input type="time">
- <input type="url">
- <input type="week">

Connecting Python with HTML and CSS for web development is primarily achieved through web frameworks and libraries, as browsers directly execute JavaScript, not Python for front-end interactions.

1. **Web Frameworks (Recommended for dynamic websites):**

- **Django and Flask:** These are popular Python web frameworks that allow you to build dynamic web applications.
    - **Templating Engines:** They utilize templating engines (like Jinja2 in Flask or Django's built-in templating system) to render HTML files. This involves embedding Python variables and logic within your HTML templates, which are then processed by the Python backend before being sent to the browser.
    - **Backend Logic:** Python handles the backend logic, such as data processing, database interactions, and routing requests, while HTML and CSS are used for the front-end presentation.

2. **Client-Side Python (for limited front-end execution):**

- **PyScript:** This library allows you to embed and run Python code directly within an HTML file using the <py-script> tag.
    - **Browser Execution:** PyScript leverages WebAssembly to enable Python execution within the browser, allowing for direct manipulation of HTML elements and client-side scripting.
    - **CSS Integration:** You can apply CSS styles to elements manipulated by PyScript just like any other HTML element.

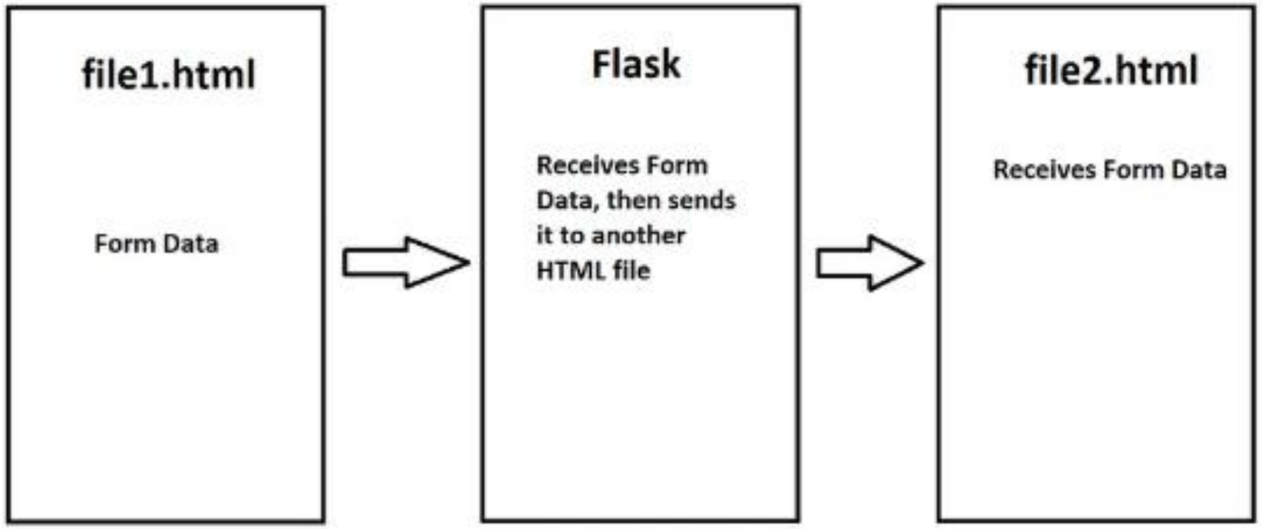
3. External CSS:

- **Linking CSS:** The most common and recommended way to connect CSS with HTML (and by extension, Python-generated HTML) is by creating a separate .css file and linking it to your HTML document using the <link> tag in the <head> section. This promotes modularity and maintainability.

In summary:

- For building full-fledged dynamic web applications, use Python web frameworks like Django or Flask, which utilize templating engines to combine Python logic with HTML and CSS.
- For direct, client-side Python execution within the browser, explore PyScript.
- Always use external CSS files linked to your HTML for better organization and styling management.

HTML



file1.html

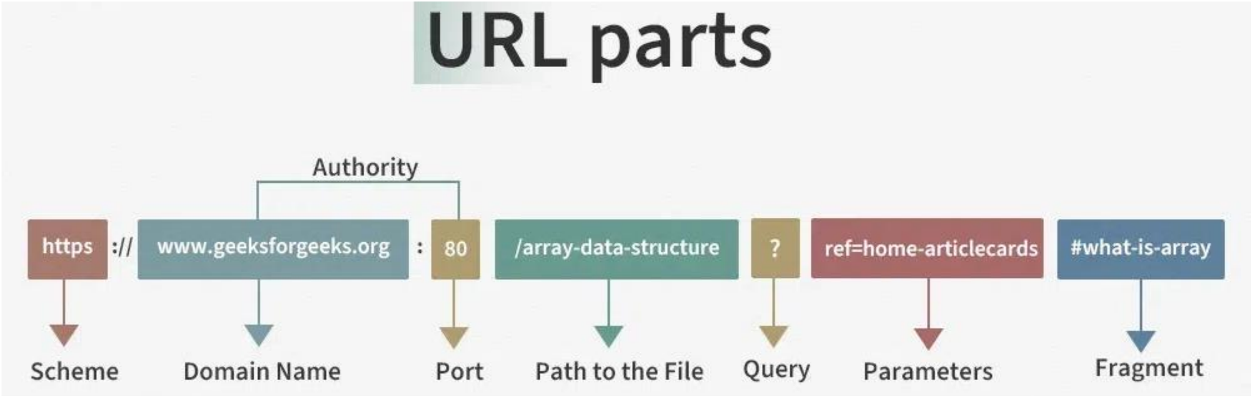
Form Data

Flask

Receives Form Data, then sends it to another HTML file

file2.html

Receives Form Data



URL parts

```
<form action="{{url_for('second_page')}}" method="post">

</form>
```

```
<form action="{{url_for('second_page')}}" method="post">
<label for="enter_value">*Sample Text*:</label><br>
<input type="text" name="enter_value">
<input type="submit" value="Submit">
</form>
```

Text

```
<h2>{{ candidate_info.courseenrolled }}</h2>
```

List

```
<ul>
<li>Laptop - $1200 (Electronics)</li>
<li>Mouse - $25 (Electronics)</li>
<li>Keyboard - $75 (Electronics)</li>
</ul>
```

Dictionary

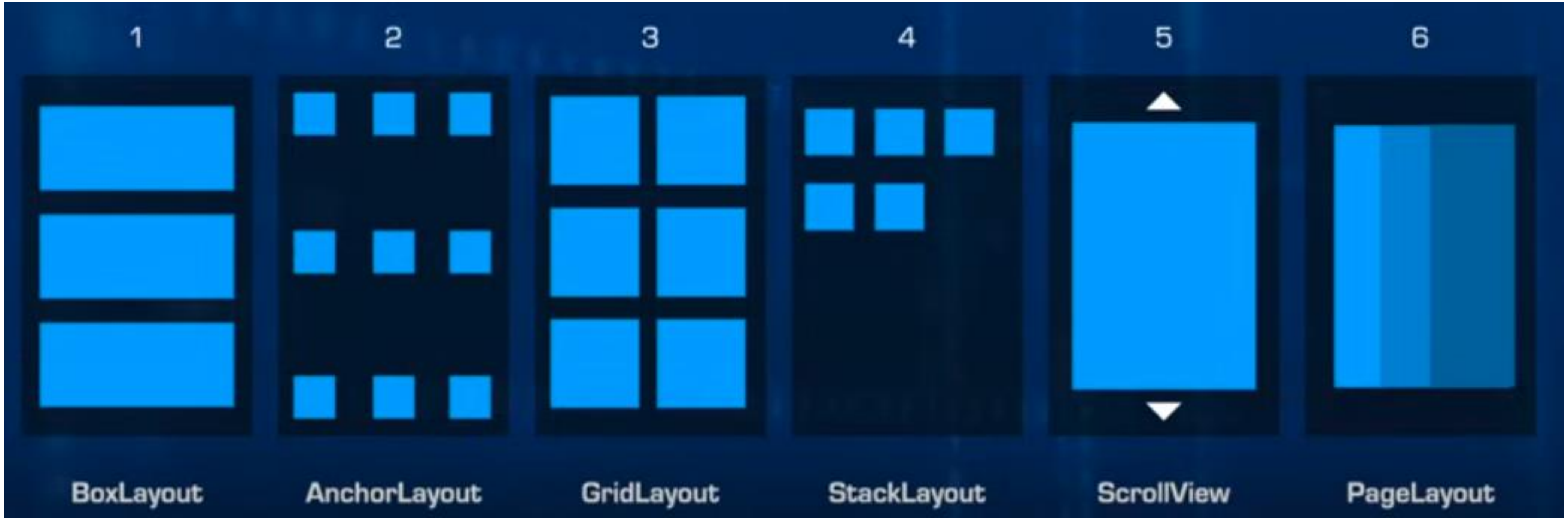
```
<h1>{product_data['name']}</h1> <p>Price:
${product_data['price']:.2f}</p>
```

Loop statement

```
{%else%}
```

Kivy

```
from kivy.uix.anchorlayout import AnchorLayout
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
from kivy.uix.stacklayout import StackLayout
```



Thankyou