

## Sesión de Aprendizaje N° 14-A

**Desarrollo Web BackEnd**

**Stack: Python - Flask**

**Python / Django**

## 1. Servidor Web: Apache HTTP Server

El **Apache HTTP Server**, comúnmente conocido simplemente como "**Apache**", es el servidor web de código abierto más popular y utilizado en el mundo. Su función principal es entregar contenido web a los usuarios.

Imagina que cuando tú escribes una dirección web (URL) en tu navegador (Chrome, Firefox, etc.) y presionas Enter, estás enviando una "**solicitud**" a un servidor. Apache es el software que reside en ese servidor y se encarga de:

1. Recibir **request** HTTP/HTTPS del navegador.
2. Buscar el contenido solicitado HTML (y imágenes, videos, archivos PDF, etc.) en el servidor.
3. Enviar **response** (ese contenido) de vuelta al navegador del usuario.

### Apache Lounge

**Apache Lounge** es un sitio web y una comunidad que se especializa en **proporcionar versiones compiladas** (binarios) del Apache HTTP Server, principalmente para sistemas operativos Windows.

#### Instalación del Servidor Portable

Descarga: <https://www.apachelounge.com/download/>

The screenshot shows the Apache Lounge website's download section for the Apache 2.4 VS17 Windows Binaries and Modules. The page features a header with the Apache logo and navigation links for Home, VS17, and Additional. A sidebar on the left lists recent releases: 20 June 2025 (New C++ Redistributable), 28 February 2025 (mod\_evasive Update), 07 February 2025 (httpd 2.4.63 Fix Crash Windows7, see here), 23 January 2025 (httpd 2.4.63), 17 November 2024 (New C++ Redistributable), and 06 November 2024 (mod\_security Update). The main content area contains text about the binaries being up-to-date and compatible with VS17. It also mentions VS17 is backward compatible and provides instructions for redistributables. A red box highlights a download link for "Apache 2.4.63-250207 Win64".

Puedes ejecutar Apache Lounge en Windows directamente desde la versión ZIP sin instalarlo como servicio.

#### 1. Extrae el ZIP

Descomprime el archivo descargado (por ejemplo, httpd-2.4.xx-win64-VS17.zip) en una carpeta como D:\Server\Apache24.

#### 2. Verifica dependencias

Asegúrate de tener instalado el paquete de Visual C++ Redistributable correspondiente (por ejemplo, VC15 o VC17). Si no, puedes descargarlo desde el sitio de Apache Lounge.

#### 3. Configura el archivo httpd.conf

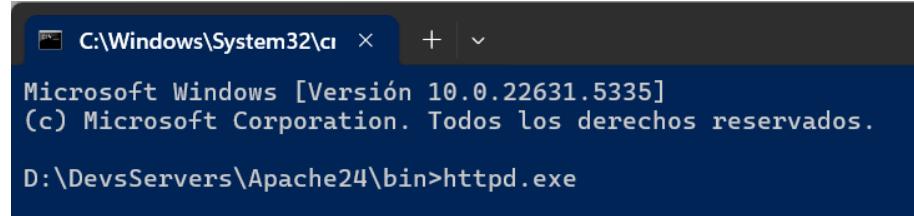
Abre D:\server\Apache24\conf\httpd.conf y revisa que tengas:

Listen 8080

ServerName localhost:8080

Define SRVROOT "D:\server\Apache24"

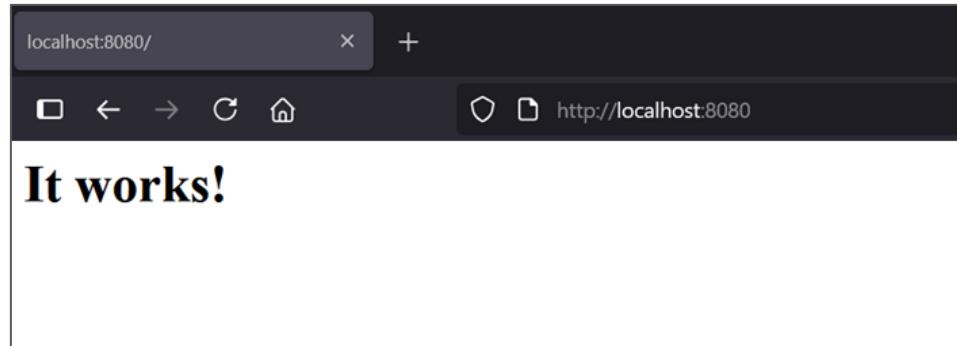
4. Ejecutar el Servidor httpd.exe



```
C:\Windows\System32> + 
Microsoft Windows [Versión 10.0.22631.5335]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DevsServers\Apache24\bin>httpd.exe
```

5. Visualizar el Servidor



2. INSTALACIÓN Y CONFIGURACIÓN DE MOD\_WSGI

¿Qué es MOD\_WSGI?

MOD\_WSGI (el acrónimo WSGI significa Web Server Gateway Interface) es un módulo para el servidor Apache que permite ejecutar aplicaciones web escritas en Python. Esta integración facilita que frameworks como Flask o Django funcionen directamente dentro de Apache sin necesidad de servidores intermedios.

**Instalación de mod\_wsgi**

**Instalación vía pip**

Primero, asegúrate de que tienes Python instalado y agregado al PATH del sistema. Luego abre la terminal y ejecuta:

```
pip install mod_wsgi
```

Esto instalará el paquete y creará el archivo del módulo .pyd necesario para Apache.

**Configurar variable MOD\_WSGI\_APACHE\_ROOTDIR**

Antes de extraer la configuración del módulo, debes indicar (variable de entorno temporal sólo para CMD) dónde está instalado Apache:

CMD

```
set MOD_WSGI_APACHE_ROOTDIR=C:/Apache24
powershell
$env:MOD_WSGI_APACHE_ROOTDIR="C:/Apache24"
```

Ajusta la ruta si Apache está instalado en otro directorio.

### Obtener líneas de configuración con mod\_wsgi-express

Permite obtener la configuración

Ejecutar:

```
mod_wsgi-express module-config
```

Esto generará líneas como:

```
LoadFile "C:/PythonXX/pythonXY.dll"  
LoadModule wsgi_module "C:/Users/TuUsuario/AppData/Local/Programs/Python/  
PythonXY/Lib/site-packages/mod_wsgi/server/mod_wsgi.cpXY-win_amd64.pyd"
```

Estas líneas deben copiarse al archivo httpd.conf de Apache.

### Configuración de Apache

Abre el archivo httpd.conf, ubicado típicamente en: C:/Apache24/conf/httpd.conf

Añade las líneas generadas por mod\_wsgi-express justo antes del bloque <IfModule mime\_module>.

### Inclusión de LoadFile y LoadModule

Ejemplo:

```
LoadFile "C:/Python39/python39.dll"  
LoadModule wsgi_module "C:/Python39/Lib/site-packages/mod_wsgi/server/mod_wsgi.  
.cp39-win_amd64.pyd"
```

### Configuración de WSGIScriptAlias y <Directory>

Agrega algo como:

```
WSGIScriptAlias / "C:/webapps/miapp/app.wsgi"  
  
<Directory "C:/webapps/miapp">  
    Require all granted  
</Directory>
```

- WSGIScriptAlias: define la entrada principal WSGI.
- <Directory>: permite el acceso al directorio de la app.

## 3. Introducción al Lenguaje Python

### ¿Qué es Python?

**Python** es un lenguaje de programación interpretado **multiparadigma** (la programación orientada a objetos, funcional e imperativa) y de **tipado dinámico**. Trabaja con una **compilación intermedia** (bytecode): que transforma el código fuente (.py) en bytecode (.pyc). Este bytecode es **interpretado por la máquina virtual** de Python (como CPython), lo que permite una ejecución rápida sin necesidad de recompilar cada vez.

### Histórica y Evolución de Python

- **1989–1991: Guido van Rossum desarrolla Python** como sucesor del lenguaje ABC. La versión 0.9.0 incluye manejo de excepciones, módulos, y estructuras como listas y diccionarios.
- **1994–2000 (Python 1.x):** Se lanza Python 1.0 con soporte para **programación funcional** (lambda, map, filter) y módulos de extensión. Se consolida como lenguaje legible y flexible.
- **2000–2010 (Python 2.x):** Python 2.0 introduce recolección de basura, Unicode, y comprensiones de listas. Aunque potente, genera problemas de compatibilidad con futuras versiones.
- **2008–presente (Python 3.x):** Python 3.0 rompe compatibilidad para mejorar el diseño. Se incorpora print() como función, Unicode por defecto, y mejoras en iteradores. Versiones recientes (3.5–3.12) incluyen f-strings, data classes, walrus operator (:=) y pattern matching.

### Ambios de uso de Python

Python se emplea ampliamente en múltiples áreas de la informática por su versatilidad y sintaxis limpia:

**Educación y enseñanza:** por ser accesible para principiantes y fomentar buenas prácticas. Ofrece recursos didácticos como notebooks interactivos (Jupyter y Google Colab). Favorece el aprendizaje progresivo de algoritmos y estructuras de datos.

#### Ciencia de datos e inteligencia artificial:

Es compatible con librerías como NumPy, Pandas, TensorFlow y Scikit-learn. Facilita análisis estadístico, manipulación de datos y creación de modelos predictivos. Su ecosistema permite visualización, entrenamiento y despliegue de modelos IA.

#### Desarrollo de software:

Se usa para crear aplicaciones de escritorio con Tkinter, PyQt o Kivy. Python permite construir CLIs, módulos reutilizables y APIs internas. Soporta pruebas automatizadas con PyTest y unittest para control de calidad.

#### Desarrollo web:

Utiliza frameworks como Django, Flask y FastAPI para construir aplicaciones escalables. Permite integración con bases de datos, APIs RESTful y autenticación. Su sintaxis clara facilita el mantenimiento de aplicaciones complejas.

#### Automatización, IOT y scripting:

- Python permite controlar dispositivos electrónicos usando placas como Raspberry Pi, Arduino (a través de Firmata) o ESP32. Se accede a pines GPIO, sensores y actuadores fácilmente usando librerías como RPi.GPIO, gpiozero o pyserial.
- Python se utiliza para automatizar tareas repetitivas, como manejo de archivos, operaciones en la red o ejecución de comandos del sistema. Gracias a su sintaxis clara y a módulos como os, shutil, subprocess o schedule, permite escribir scripts eficientes y mantenibles. Es ideal para mejorar la productividad en entornos administrativos, DevOps, testing automatizado y control de procesos.

### Recursos y Documentación de Python

Recursos

<https://www.python.org/>

Documentación

<https://docs.python.org/es/3.13/>

## 4. Instalación y Ejecución de Python

### Instalar Python en Windows

- Visita el sitio oficial: Ve a <https://www.python.org/downloads>

The screenshot shows the Python.org homepage. At the top, there are navigation links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header is a search bar with a magnifying glass icon and a 'GO' button. To the right of the search bar is a 'Socialize' button. The main content area features a large Python logo. Below the logo is a navigation bar with links for About, Downloads, Documentation, Community, Success Stories, News, and Events. A code snippet is displayed in a box:

```
# Python 3: List comprehensions
>>> fruits = ['Banana', 'Apple', 'Lime']
>>> loud_fruits = [fruit.upper() for fruit in
fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'LIME']

# List and the enumerate function
>>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

A yellow callout box highlights the first line of the code. To the right of the code, a section titled "Compound Data Types" is shown with a brief description and a link to "More about lists in Python 3". At the bottom of the page, a blue banner says "Python is a programming language that lets you work quickly".

- Descarga Python: Haz clic en el botón de descarga para Windows. Se descargará un archivo .exe.
- Instala Python:
  - Ejecuta el archivo descargado.
  - Marca la opción “**Add Python to PATH**” antes de hacer clic en “Install Now”.
  - Espera a que finalice la instalación.
- Verifica la instalación:
  - Abre la terminal (CMD) y escribe: **python --version**
  - Deberías ver algo como Python 3.12.0

### Tu primer programa en Python

Crea un archivo llamado hola.py con este contenido:

```
# Mi primer programa en Python
print("¡Hola, mundo!")
```

Luego, desde la terminal, ejecuta el archivo:

```
python hola.py
```

Y verás la salida: ¡Hola, mundo!

## 5. Gestión de Dependencias con PIP

### Herramienta PIP

**PIP (acrónimo de "Python Installs Packages")** es el gestor de paquetes oficial de Python, utilizado para instalar, actualizar y desinstalar bibliotecas externas desde el repositorio PyPI (Python Package Index).

Es una herramienta de línea de comandos que permite a los desarrolladores ampliar las capacidades de Python incorporando módulos de terceros de forma sencilla.

Por ejemplo, puedes instalar la biblioteca requests con el comando:

```
pip install requests
```

### PIP también permite:

- Listar paquetes instalados (pip list)
- Actualizar paquetes (pip install --upgrade nombre)
- Instalar dependencias desde un archivo requirements.txt (pip install -r requirements.txt)

## 6. Python Estilos PEP-8

### Indentación

Reglas principales:

- Usa 4 espacios por nivel de indentación (¡no uses tabulaciones!)
- Las estructuras como if, for, def, class deben ir indentadas correctamente para que el flujo del código funcione.

```
def saludar(nombre):
    if nombre:
        print("Hola", nombre)
```

Si no se respeta la indentación, Python generará un error tipo `IndentationError`.

### Estilo general PEP-8

#### Espacios

- No pongas espacios alrededor de signos de asignación en argumentos de funciones:

```
# Correcto
def suma(a, b=2):
```

```
# Incorrecto
def suma(a, b = 2):
```

- Sí poner espacios alrededor de operadores en expresiones:

```
x = 3 + 4
```

#### Longitud de línea

- Máximo 79 caracteres por línea para legibilidad.

#### Nombres de variables y funciones

- Funciones y variables: estilo SnakeCase minúsculas separadas por guiones bajos → `saludar_amigo`
- Clases: estilo PascalCase → `MiClaseBonita`

#### Importaciones

- Organízalas en tres bloques: estándar, de terceros, y locales. Ejemplo:

```
# Estandar
import os
import sys
# Tercero
import requests
# Local
from mi_paquete import modulo_local
```

### 7. Sintaxis Python

La sintaxis de Python se caracteriza por ser limpia, intuitiva y muy cercana al lenguaje humano. *Utiliza la indentación con espacios para definir bloques de código*, en lugar de llaves como otros lenguajes.

Las **instrucciones se escriben en líneas separadas**, y no es necesario terminar con punto y coma. Las estructuras condicionales y de bucles siguen un formato claro con dos puntos al final de la línea de apertura.

#### 1. Comentarios

Sintaxis:

Comentario de una línea: # comentario

Comentarios multilínea: triple comillas

```
# Esto es un comentario
```

```
"""
Esto es un comentario
multilínea
"""
```

#### 2. Variables

Sintaxis:

Las variables no necesitan declaración previa de tipo.

```
python
nombre = "Ana"
edad = 30
altura = 1.65
```

#### 3. Salida de datos

Sintaxis:

print() muestra datos en consola.

```
print("Hola, mundo!")
print("La edad es", edad)
```

#### 4. Control de flujo condicional

Sintaxis:

if, elif, else

```
edad = 20
if edad >= 18:
    print("Eres mayor de edad")
elif edad == 17:
    print("Estás cerca de ser mayor")
else:
    print("Eres menor de edad")
```

#### 5. Estructuras de repetición (bucles)

Sintaxis:

for y while

for

```
for i in range(3):
    print("Repetición", i)
```

```
while
    contador = 0
    while contador < 3:
        print("Contador:", contador)
        contador += 1
```

### 6. Manejo de archivos e inclusión

Sintaxis:

Leer archivo

```
with open("archivo.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

Escribir en archivo

```
with open("archivo.txt", "w") as archivo:
    archivo.write("Hola desde Python")
```

### 7. Importar código con import:

```
import math
print(math.sqrt(16))
```

### 8. Funciones

Sintaxis:

```
def saludar(nombre):
    print("Hola", nombre)
```

```
saludar("Lucía")
```

### 9. Programación Orientada a Objetos (POO)

Sintaxis:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        print(f"Hola, soy {self.nombre} y tengo {self.edad} años.")
```

Crear objeto

```
personal = Persona("Carlos", 25)
personal.saludar()
```

## 8. Flask Framework



**Flask** es un *microframework de desarrollo web* escrito en Python, diseñado para ser ligero, flexible y fácil de usar. Su enfoque minimalista permite construir aplicaciones web rápidamente sin necesidad de herramientas complejas o configuraciones pesadas.

### Características principales:

- Utiliza el patrón MVC (Modelo-Vista-Controlador) para organizar el código.
- Incluye un servidor de desarrollo integrado y soporte para pruebas unitarias.
- Permite definir rutas (@app.route) para manejar peticiones HTTP.
- Se puede ampliar fácilmente con extensiones como Flask-SQLAlchemy, Flask-Login, entre otras.
- Compatible con Python 3 y el protocolo WSGI para desplegar en servidores web.

### Recursos y Documentación:

Flask framework pequeño.

Link de Laravel Framework <https://flask.palletsprojects.com/en/stable/>

### Terminología Flask

**Flask:** El núcleo del framework, utilizado para crear aplicaciones web.

**Route (`@app.route`):** Decorador que define las URL que activan funciones específicas.

**Request:** Objeto que representa datos enviados por el cliente (como formularios, parámetros, etc.).

**Response:** Objeto que contiene los datos que la app envía de vuelta al cliente.

**Render\_template:** Función que renderiza archivos HTML usando Jinja2.

**Jinja2:** Motor de plantillas que permite incrustar lógica en archivos HTML.

**Debug mode:** Modo que facilita el desarrollo al recargar la app automáticamente y mostrar errores detallados.

**Blueprint:** Componente para dividir una app en partes reutilizables y organizadas.

**Session:** Sistema para almacenar información específica del usuario en el navegador.

**Flask-WTF:** Extensión para facilitar el manejo de formularios con validación.

**Flask-SQLAlchemy:** Extensión que integra SQLAlchemy para trabajar con bases de datos relacionales.

**Flask-Migrate:** Herramienta que gestiona migraciones de base de datos.

**Flask-Login:** Extensión que gestiona la autenticación y el inicio de sesión.

**Contexto de aplicación:** Entorno donde viven ciertas variables de configuración y estado.

### Primera Aplicación Web en Flask

#### 1. Instala Flask

Abre tu terminal y ejecuta:

```
pip install flask
```

#### 2. Crea el archivo principal

Crea un archivo llamado app.py y escribe lo siguiente:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hola():
    return '¡Hola Mundo desde Flask!'

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

#### 3. Ejecuta la aplicación

En la terminal, ingresa a la carpeta donde guardaste app.py y ejecuta:

```
python app.py
```

Verás algo como:

Se ejecuta en <http://127.0.0.1:5000/>

## 9. Jinja2

### ¿Qué es Jinja2?

**Jinja en Python** es un **motor de plantillas** que permite generar contenido dinámico, especialmente útil en aplicaciones web. Se utiliza para separar la lógica del programa (**backend**) de la presentación visual (**frontend**), como el HTML.

### ¿Qué hace Jinja?

- **Renderiza plantillas:** Inserta datos dinámicos en archivos HTML u otros formatos.
- **Usa estructuras de control:** Puedes usar condicionales (if, else) y bucles (for) como en Python.
- **Permite herencia de plantillas:** Puedes crear una plantilla base y extenderla en otras.

- **Utiliza filtros y macros:** Para modificar datos o reutilizar fragmentos de código.

### Instalación de jinja2

```
pip install Jinja2
```

### Estructura recomendada de proyecto

```
mi_proyecto/
└── app.py
└── templates/
    ├── base.html
    └── index.html
```

### Principales Sentencias de Jinja2

#### 1. Variables {{ ... }}

**Definición:** Se usan para mostrar el valor de una variable en la plantilla.

**Ejemplo:**

```
<p>Hola {{ nombre }}</p>
```

Si nombre = "Luis", se mostrará: Hola Luis

#### 2. Bucle for

**Definición:** Recorre una lista o secuencia y repite contenido.

**Ejemplo:**

```
<ul>
    {% for fruta in frutas %}
        <li>{{ fruta }}</li>
    {% endfor %}
</ul>
```

Si frutas = ["manzana", "pera"], se mostrará una lista con esos elementos.

#### 3. Condicional if, elif, else

**Definición:** Muestra contenido según condiciones.

**Ejemplo:**

```
{% if edad >= 18 %}
    <p>Eres mayor de edad.</p>
{% elif edad >= 13 %}
    <p>Eres adolescente.</p>
{% else %}
    <p>Eres niño.</p>
{% endif %}
```

#### 4. Herencia de plantillas extends y block

**Definición:** Permite reutilizar una plantilla base y definir bloques que se pueden sobrescribir.

**Ejemplo base.html:**

```
<html>
  <body>
    {% block contenido %}{% endblock %}
  </body>
</html>
```

**Ejemplo hijo.html:**

```
{% extends "base.html" %}
{% block contenido %}
  <p>Bienvenido a mi sitio</p>
{% endblock %}
```

**5. Filtros |**

**Definición:** Modifican el valor de una variable.

**Ejemplo:**

```
<p>{{ nombre|upper }}</p>
```

Convierte nombre a mayúsculas.

**6. Comentarios {# ... #}**

**Definición:** No se muestran en el HTML generado.

**Ejemplo:**

```
{# Este es un comentario interno #}
```

**7. Asignación set**

**Definición:** Crea una variable dentro de la plantilla.

**Ejemplo:**

```
{% set saludo = "Hola" %}
<p>{{ saludo }} {{ nombre }}</p>
```

**8. Control de bucle loop**

**Definición:** Proporciona información sobre el estado del bucle.

**Ejemplo:**

```
{% for item in lista %}
  <p>{{ loop.index }} - {{ item }}</p>
{% endfor %}
```

Segunda Unidad: Desarrollo Backend  
GUÍA DE LABORATORIO N° 14-1:  
DESARROLLO PYTHON – FLASK - DJANGO

**Docente:** Jaime Suasnabar Terrel

**Fecha:** .....

**Duración:** 30 minutos

**Instrucciones:**

**OBJETIVOS**

- Conocer el uso de los diversos elementos del lenguaje python

**EJERCICIO 01. CREACIÓN DE PROYECTO PYTHON**

**1. Crear una Aplicación Orientado a Objetos de la Clase Persona y Estudiante**

**1. Crear una carpeta y su archivo principal**

```
laravel new estudiantes-app  
cd estudiantes-app
```

**2. Crear la Clase base: Persona**

```
class Persona:  
    def __init__(self, nombre, edad):  
        self.nombre = nombre  
        self.edad = edad  
    def presentarse(self):  
        return f"Hola, soy {self.nombre} y tengo {self.edad} años."
```

**3. Clase derivada: Estudiante (hereda de Persona)**

```
class Estudiante(Persona):  
    def __init__(self, nombre, edad, carrera):  
        super().__init__(nombre, edad) # Llama al constructor de Persona  
        self.carrera = carrera  
  
    def estudiar(self):  
        return f"{self.nombre} está estudiando {self.carrera}."  
  
    def presentarse(self):  
        # Sobrescribe el método de Persona  
        return f"Hola, soy {self.nombre}, tengo {self.edad} años y estudio {self.carrera}."
```

**4. Ejemplo de uso**

```
alumno = Estudiante("Laura", 20, "Ingeniería de Sistemas")
```

```
print(alumno.presentarse()) # Hola, soy Laura, tengo 20 años y estudio  
Ingeniería de Sistemas.  
print(alumno.estudiar())      # Laura está estudiando Ingeniería de  
Sistemas.
```

## EJERCICIO 02. CREACIÓN UNA APLICACIÓN WEB EN FLASK

### 1. Requisitos previos

- Python instalado
- MySQL corriendo localmente
- Instalar dependencias:

```
pip install flask pymysql
```

### 2. Crear la base de datos y tabla

```
CREATE DATABASE escuela;  
  
USE escuela;  
  
CREATE TABLE estudiantes (  
    IdEstudiante INT PRIMARY KEY AUTO_INCREMENT,  
    nomEstudiante VARCHAR(100),  
    dirEstudiante VARCHAR(150),  
    ciuEstudiante VARCHAR(100)  
)
```

### 3. Conexión a MySQL (conexion.py)

```
import pymysql  
  
def obtener_conexion():  
    return pymysql.connect(  
        host='localhost',  
        user='root',  
        password='tu_contraseña',  
        db='escuela',  
        cursorclass=pymysql.cursors.DictCursor  
)
```

### 4. Aplicación Flask (app.py)

```
from flask import Flask, render_template, request, redirect  
from conexion import obtener_conexion  
  
app = Flask(__name__)  
  
@app.route('/')  
def formulario():  
    return render_template('formulario.html')
```

```
@app.route('/guardar', methods=['POST'])
def guardar():
    nombre = request.form['nombre']
    direccion = request.form['direccion']
    ciudad = request.form['ciudad']

    conexion = obtenerConexion()
    with conexion.cursor() as cursor:
        cursor.execute("INSERT INTO estudiantes(nomEstudiante,
dirEstudiante, ciuEstudiante) VALUES (%s, %s, %s)",
                    (nombre, direccion, ciudad))
    conexion.commit()
    conexion.close()
    return redirect('/')
```

## 5. Plantilla HTML (templates/formulario.html)

```
<!DOCTYPE html>
<html>
<head>
    <title>Registro de Estudiantes</title>
</head>
<body>
    <h1>Agregar Estudiante</h1>
    <form action="/guardar" method="post">
        <label>Nombre:</label><br>
        <input type="text" name="nombre"><br>
        <label>Dirección:</label><br>
        <input type="text" name="direccion"><br>
        <label>Ciudad:</label><br>
        <input type="text" name="ciudad"><br><br>
        <input type="submit" value="Guardar">
    </form>
</body>
</html>
```

## 6. Ejecutar la app

```
python app.py
```

Abre tu navegador en <http://localhost:5000> y prueba el formulario 

## **PREGUNTAS DE REPASO**

### **Preguntas esenciales de Python**

1. ¿Qué diferencia existe entre una lista (list) y una tupla (tuple) en Python, y cuándo conviene usar cada una?
2. ¿Cómo funciona el manejo de excepciones con try, except, finally y raise, y cómo se crean excepciones personalizadas?
3. ¿Qué es una función lambda y cómo se compara con una función tradicional definida con def?
4. ¿Qué son los diccionarios (dict) y cómo se recorren con métodos como .items() para trabajar con claves y valores?
5. ¿Cómo se define una clase en Python y cuál es el rol del método \_\_init\_\_ dentro de la estructura orientada a objetos?

### **Preguntas básicas de Flask**

6. ¿Qué es Flask y por qué se le considera un microframework ideal para construir aplicaciones web ligeras?
7. ¿Cómo se define una ruta en Flask con el decorador @app.route, y cómo funcionan las rutas dinámicas como /usuario/<id>?
8. ¿Cómo se usan plantillas HTML con render\_template y qué función cumple Jinja2 en la carpeta templates/?
9. ¿Qué es request en Flask y cómo se obtiene información que el usuario envía con formularios (usando GET o POST)?
10. ¿Cómo se guarda información temporal entre peticiones usando session, y cómo se protege esa información del lado del cliente?