# Introducción

# Instalación y configuración

Crear el entorno virtual:

**py -3.12 -m venv .venv**

Activar el entorno virtual:

**.\.venv\Scripts\activate**

Instalar Flask:

**python -m pip install --upgrade pip setuptools wheel**

**pip install flask**

Ejecutar nuestro proyecto:

**code .**

Creamos un archivo e importamos la clase **Flask** perteneciente el módulo **flask**:

Texto

El contenido generado por IA puede ser incorrecto.

from flask import Flask

Lo siguiente es, usando esta clase **Flask** crear una aplicación.

Creamos una variable que representa nuestra aplicación de Flask:

from flask import Flask

app = Flask(\_\_name\_\_)

Con esta instrucción hemos creado nuestra aplicación.

Ahora necesitamos crear una ruta que estará asociada a una función que representa una vista en nuestra aplicación.

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def inicio():

    nombre = "Angel"

    return "Hola mundo!!!"

Ahora para ejecutar nuestra aplicación desde la consola ejecutaremos:

**flask --app main run**

Sustituyendo **main** por el nombre qu le hayamos puesto a nuestro proyecto.

**from flask import Flask**: Importas la clase Flask desde el paquete flask. Esa clase te permite crear una aplicación web WSGI ligera.

**app = Flask(\_\_name\_\_)**: Creas una instancia de Flask. **\_\_name\_\_** le dice a Flask dónde se encuentra tu módulo para localizar recursos (plantillas, archivos estáticos, etc.). Es decir, *“Enciende mi servidor Flask”*.

**@app.route('/')**: Este decorador registra la URL raíz **/**. Cada vez que el navegador pida esa ruta, Flask llamará a la función justo debajo. *“Si alguien pide la página raíz* ***/****, usa la función de abajo para contestar”*.

**def inicio()**: Defines la vista (view function). Debe devolver algo que el navegador pueda entender: normalmente una cadena (HTML) o un objeto Response. La función **inicio** es lo que Flask llama **vista** (view). No es más que código Python que decide qué ponemos en la respuesta.

**return 'Hola Mundo'**: Devuelves la respuesta al cliente. Devolvemos la frase **Hola Mundo**. Ese texto viaja al navegador como la respuesta. Aquí es texto plano, pero podría ser HTML, JSON, etc.

# Modo Debug

**flask --app hello run --debug**

**¿Qué es exactamente “debug mode” en Flask?**

Al activar **debug mode** Flask cambia tres cosas a la vez:

1. **app.debug = True**
   * Se registran mensajes más verbosos y se habilitan ayudas internas.
2. **Recargador automático (reloader)**
   * Arranca el servidor en un sub-proceso y vigila tu código; cuando detecta cambios reinicia la app sin que tengas que parar nada.​[flask.palletsprojects.com](https://flask.palletsprojects.com/en/stable/quickstart/?utm_source=chatgpt.com)
3. **Depurador interactivo (Werkzeug debugger)**
   * Si una vista lanza una excepción, verás una página con el traceback y una consola para ejecutar Python en cada frame. Está protegida con un *PIN*, pero supone un gran riesgo de ejecución remota: **nunca uses debug en producción**.

​

**Cómo encenderlo (Flask ≥ 2.3)**

|  |  |  |
| --- | --- | --- |
| Método | Comando / código | Cuándo usarlo |
| CLI recomendado | flask --app app.py run --debug | Siempre que lances tu app con la CLI.​[flask.palletsprojects.com](https://flask.palletsprojects.com/en/stable/debugging/?utm_source=chatgpt.com) |
| Variable entorno | export FLASK\_DEBUG=1 flask --app app.py run | Útil en contenedores o *docker-compose*. |
| En código | python<br>if \_\_name\_\_ == '\_\_main\_\_':<br> app.run(debug=True)<br> | Scripts que arranques con python app.py.​[flask.palletsprojects.com](https://flask.palletsprojects.com/en/stable/debugging/?utm_source=chatgpt.com) |

# Jinja2

**Jinja2** (hoy llamado simplemente **Jinja**) es un *motor de plantillas* escrito en Python.  
Permite mezclar texto estático (por ejemplo HTML, XML, YAML o cualquier otro formato) con “huecos” que se rellenan dinámicamente con datos cuando el programa lo ejecuta. Nació en 2008 inspirado en el sistema de plantillas de Django, pero con una sintaxis más parecida a Python y mayor extensibilidad.

|  |  |
| --- | --- |
| Ventaja | Qué implica |
| Velocidad | Genera el texto final muy rápido gracias a que compila la plantilla a byte-code Python. [Jinja](https://jinja.palletsprojects.com/?utm_source=chatgpt.com) |
| Expresividad | Puedes usar variables, filtros, bucles, condicionales, macros y herencia de plantillas con una sintaxis breve (parece Python, pero sin riesgo de ejecutar código arbitrario). [Jinja](https://jinja.palletsprojects.com/en/stable/intro/?utm_source=chatgpt.com) |
| Extensible | Admiten plugins: filtros propios, tests personalizados, tags y extensiones. [Jinja](https://jinja.palletsprojects.com/en/stable/api/?utm_source=chatgpt.com) |
| Seguro | Auto-escape de HTML para prevenir XSS, entorno sandbox para renderizar plantillas no confiables. [Jinja](https://jinja.palletsprojects.com/en/stable/intro/?utm_source=chatgpt.com) |

**¿Jinja o Jinja2?**

* El paquete en PyPI se sigue llamando **Jinja2**, pero desde la versión 3 (el esquema semántico “3.x”) la documentación y los módulos internos hablan simplemente de **Jinja**.
* El import no cambió:

**from jinja2 import Template, Environment, FileSystemLoader**

Flask te expone todo lo necesario. No es necesario importar Jninja. Para renderizar HTML basta con importar **render\_template**:

**from flask import render\_template**

**Delimitadores básicos**

|  |  |  |
| --- | --- | --- |
| Delimitador | Uso | Ejemplo |
| {{ … }} | Expresiones (se imprime el resultado) | {{ user.name }} |
| {% … %} | Sentencias de control | {% for item in cart %} |
| {# … #} | Comentarios (no aparecen en salida) | {# solo para autores #} |
| {{ … upper }} | Filtros | {{ … upper }} |

# Vistas

Una **vista** (view function) es la pieza de código que Flask ejecuta cuando la URL de una petición coincide con la ruta registrada. La función produce un valor (cadena, dict, tupla, objeto Response, *stream*, etc.) y Flask lo convierte en la respuesta HTTP que devuelve al navegador.

Podemos crear tantas vistas como queramos pero tienen que tener diferente nombre.

Flask asigna a cada vista un **endpoint** (por defecto, el nombre de la función).

**No puede repetirse**: si tienes dos funciones llamadas **index**, la segunda sobrescribe a la primera y Flask avisa en consola.

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def inicio():

    nombre = "Angel"

    return "Hola mundo!!!"

@app.route('/saludo')

def saludo():

    return "Hola mundo"

Podemos incluso agregar varias rutas a una vista:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

@app.route('/home')

@app.route('/inicio')

def inicio():

    nombre = "Angel"

    return "Hola mundo!!!"

Por lo que, como podrás imaginar, un proyecto de Flask acabará teniendo varias rutas según nuestra necesidad:

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

@app.route('/home')

def inicio():

    return f'<h1>Esta es la página de inicio</h1>'

@app.route('/servicios')

def servicios():

    aficiones = None

    return render\_template('servicios.html', aficiones = aficiones)

@app.route('/contacto')

def contacto():

    return render\_template('contacto.html')

Podemos enviar variables:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

@app.route('/home')

@app.route('/inicio')

def inicio():

    nombre = "Angel"

    return "Hola mundo!!!"

@app.route('/saludo')

def saludo():

    return "Hola mundo"

@app.route('/saludo\_personal/<nombre>')

def saludo\_personal(nombre):

    return f'<h1>Bienvenido {nombre}</h1>'

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Por defecto el tipo de dato de una variable pasada por url es **String**. Este es el tipo por defecto, pero se puede cambiar:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def inicio():

    nombre = "Angel"

    return "Hola mundo!!!"

@app.route('/saludo\_personal/<nombre>')

def saludo\_personal(nombre):

    return f'<h1>Bienvenido {nombre}</h1>'

@app.route('/saludo\_edad/<nombre>/<int:edad>')

def saludo\_edad(nombre = None, edad = None):

    return f'<h1>Hola, {nombre}, tienes {edad} años.</h1>'

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Podemos contestar de diferente forma según los datos que se nos aporten:

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

@app.route('/home')

def inicio():

    return f'<h1>Esta es la página de inicio</h1>'

@app.route('/saludo\_personal/<nombre>')

def saludo\_personal(nombre):

    return f'<h1>Bienvenido {nombre}</h1>'

@app.route('/saludo\_edad/<nombre>/<int:edad>')

def saludo\_edad(nombre = None, edad = None):

    nombre = 'Angel'

    edad = 50

    if edad<18 :

        return f'<h1>Hola {nombre}, tienes {edad} años, eres menor de edad.</h1>'

    elif edad == 18:

        return f'<h1>Hola {nombre}, tienes {edad} años, acabas de cumplir la mayoría de edad.</h1>'

    else:

        return f'<h1>Hola, {nombre}, tienes {edad} años, eres mayor de edad.</h1>'

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Word

El contenido generado por IA puede ser incorrecto.

# Plantillas

Una **plantilla** (template) es un archivo de texto –normalmente HTML– que contiene fragmentos de lógica declarativa (sintaxis **Jinja**).  
Cuando llamas a **render\_template()**, Flask:

1. Localiza el archivo en la carpeta templates/.
2. Lo compila a byte-code Python (caché en memoria/disco).
3. Sustituye las expresiones **{{ … }}** y evalúa las sentencias de control

**{% … %}**

1. Devuelve la cadena final al navegador como parte de la respuesta HTTP.

Podemos crear plantillas html para llamarlas como rutas:

Para ello, creamos una plantilla, en nuestro caso la llamaremos **home.html**:

<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Esta es la página de inicio!!!</h1>

</body>

</html>

Y tendremos que importar el módulo **render-template** y usarlo:

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

    return render\_template('index.html')

De modo que nuestro proyecto quedaría así:

Texto

El contenido generado por IA puede ser incorrecto.

Estructura recomendada para un proyecto en Flask sería algo como esto:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Y ahora si llamamos al a ruta por defecto:

Texto

El contenido generado por IA puede ser incorrecto.

Podemos pasar variable por plantilla.

Para ello, en la ruta creamos una variable:

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

    name = "Angel"

    return render\_template('index.html', name = name)

@app.route('/hello')

@app.route('/hello/<name>')

@app.route('/hello/<name>/<int:age>')

def hello(name=None, age=None):

    if name is None and age is None:

        return '<h1>Hola Mundo!</h1>'

    elif age is None:

        return f'<h1>Hola, {name}!</h1>'

    else:

        return f'<h1>Hola, {name}, el doble de tu edad es {age \* 2}!</h1>'

Y luego en la plantilla la usamos:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Bienvenido!!!</h1>

    <h2>Hola, {{name}}</h2>

</body>

</html>

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Podemos incluir estructuras de control en nuestra plantilla. Es este ejemplo incluiremso una sentencia **IF**:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Bienvenido!!!</h1>

    {% if name %}

    <h2>Hola, {{name}}</h2>

    {% else %}

    <h2>Hola, desconocido</h2>

    {% endif %}

</body>

</html>

De esta forma, si la variable no nos llega, se ejecutará lo que haya en el **else**:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Y si la variable tiene contenido se ejecuta el **if**:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

## Pasar listas

Vamos a trabajar ahora con una lista que recorreremos con un **for**:

En el **main.py**:

@app.route('/clientes')

def clientes():

    clientes = ['Manolo garcía', 'Ana Jiménez', 'Andrés González', 'Sonia Pérez']

    return render\_template('clientes.html', clientes = clientes)

En el html, que llamaremos **clientes.html**:

<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Lista de clientes</h1>

    <ul>

        {% for cliente in clientes%}

        <li>{{cliente}}</li>

        {% endfor %}

    </ul>

</body>

</html>

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

## Herencia de plantillas

Flask usa **Jinja2** como motor de plantillas.  
Jinja2 incorpora un sistema de *herencia* muy parecido al de los lenguajes orientados a objetos: defines un “esqueleto” común (plantilla base) y luego los demás archivos *.html* lo **extienden** y rellenan sólo las partes que cambian.

Esto evita repetir el mismo <head>, las barras de navegación, los scripts, etc. —­­el famoso principio **DRY** (*Don’t Repeat Yourself*).

Crearemos una plantilla que normalmente se llama **base.html**:

<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Mi sitio web - {% block title %}{% endblock %}</title>

</head>

<body>

    {% block content %}

    {% endblock %}

</body>

</html>

Luego en nuestra plantilla hija **clientes.html**:

{% extends 'base.html' %}

{% block title %}Página de inicio{% endblock %}

{% block content %}

    <h1>Bienvenido a mi sitio web</h1>

    {% if name %}

        <p>Hola, {{ name }}</p>

    {% else %}

        <p>Hola, desconocido</p>

    {% endif %}

    <h2>Lista de amigos</h2>

    <ul>

        {% for cliente in clientes %}

            <li>{{ cliente }}</li>

        {% endfor %}

    </ul>

{% endblock %}

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

A efectos prácticos no vamos a notar la diferencia, pero hemos hecho nuestra plantilla mucho más dinámica.

### Include

El tag **include** sirve para **insertar** el contenido renderizado de otra plantilla dentro de la plantilla actual. Es la forma más directa de **reusar fragmentos** (cabeceras, barras de navegación, formularios, tarjetas, etc.) sin copiar y pegar código HTML.

**Sintaxis básica**

**{% include "navbar.html" %}**

En nuestro caso vamos a incluir una barra de navegación y un footer a nuestro proyecto mediante **include**.

Nuestra estructura de archivos quedaría de la siguiente forma:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Y los archivos con el siguiente contenido:

**\_navbar.html**

<!-- El subrayado inicial \_ es solo una convención para indicar que es un parcial. -->

<!-- Los enlaces usan url\_for() para que sigan funcionando aunque cambies las rutas más adelante. -->

{#  Este archivo es un “parcial” (fragmento reutilizable).              #}

{#  El guion bajo inicial (\_) es solo una convención para decir         #}

{#  “no se invoca directamente; se incluye en otras plantillas”.        #}

<nav>

    <!--

        Menú de navegación horizontal:

        - <ul> sin viñetas (list-style:none)

        - flexbox para que los elementos queden en línea

        - gap para separar los enlaces

    -->

    <ul style="list-style:none; display:flex; gap:1rem; padding:0">

        <!-- Cada enlace usa url\_for(nombre\_función) para generar la     -->

        <!-- URL correcta incluso si cambias las reglas de ruta en Flask -->

        <li><a href="{{ url\_for('home') }}">Inicio</a></li>

        <li><a href="{{ url\_for('clientes') }}">Clientes</a></li>

        <li><a href="{{ url\_for('servicios') }}">Servicios</a></li>

        <li><a href="{{ url\_for('contacto') }}">Contacto</a></li>

    </ul>

</nav>

<hr> <!-- Separador visual entre el menú y el contenido principal -->

**Base.html**

  <!DOCTYPE html>

  <html lang="es">

  <head>

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1">

    <title>Proyecto de Flask</title>

  </head>

  <body>

    {# menú de navegación reutilizable #}

    {% include "\_navbar.html" %}

    <main>

      {% block content %}{% endblock %}

    </main>

  </body>

  </html>

**Clientes.html**

{% extends "base.html" %}

{% block title %}Clientes{% endblock %}

{% block content %}

  <h1>Clientes</h1>

  <!-- Saludo opcional (llega desde la vista ‘clientes’ en main.py) -->

  {% if name %}

    <p>Hola, {{ name }} 😄</p>

  {% endif %}

  <h2>Lista de clientes</h2>

  <ul>

    {# Recorremos la lista ‘clientes’ enviada por la vista #}

    {% for cliente in clientes %}

      <li>{{ cliente }}</li>

    {% endfor %}

  </ul>

{% endblock %}

**Contacto.html**

{% extends "base.html" %}

{% block title %}Contacto{% endblock %}

{% block content %}

  <h1>Contacto</h1>

  <!-- Enlace mailto para abrir el cliente de correo -->

  <p>Puedes escribirnos a

     <a href="mailto:hola@example.com">hola@example.com</a></p>

{% endblock %}

**Home.html**

{# Indicamos que esta plantilla hereda de base.html #}

{% extends "base.html" %}

{% block title %}Inicio{% endblock %}

{% block content %}

  <h1>Bienvenido a mi sitio Web</h1>

  <!--

      Condicional Jinja:

      - Si llega la variable name se saluda al usuario.

      - Si no, mensaje genérico para visitantes.

  -->

  {% if name %}

    <p>Hola, {{ name }} 👋</p>

  {% else %}

    <p>Hola, visitante</p>

  {% endif %}

{% endblock %}

**Servicios.html**

{% extends "base.html" %}

{% block title %}Servicios{% endblock %}

{% block content %}

  <h1>Nuestros servicios</h1>

  <ul>

    <li>Consultoría</li>

    <li>Desarrollo a medida</li>

    <li>Soporte 24 × 7</li>

  </ul>

{% endblock %}

**Main.py**

"""

main.py

========

Aplicación Flask sencilla con:

- Página de inicio

- Ejemplos de rutas dinámicas

- Páginas basadas en plantillas Jinja

- Listado de clientes

Para lanzar en consola:

    python main.py

o bien

    flask --app main run --debug

"""

from flask import Flask, render\_template  # importamos Flask y helper para plantillas

app = Flask(\_\_name\_\_)  # creamos la aplicación

# --------------------------------------------------

# 1️⃣  Página de inicio

# --------------------------------------------------

@app.route("/")        # ruta raíz

@app.route("/home")    # alias /home → misma función

def home():

    """

    Renderiza la página de inicio.

    Pasamos la variable 'name' para saludar al usuario.

    """

    return render\_template("home.html", name="Ángel")

# --------------------------------------------------

# 2️⃣  Saludo simple

# --------------------------------------------------

@app.route("/saludo\_personal/<nombre>")

def saludo\_personal(nombre):

    """

    Saluda al usuario con el nombre capturado en la URL.

    Ej.: /saludo\_personal/Ada -> “Bienvenido Ada”

    """

    return f"<h1>Bienvenido {nombre}</h1>"

# --------------------------------------------------

# 3️⃣  Saludo con edad

# --------------------------------------------------

@app.route("/saludo\_edad/<nombre>/<int:edad>")

def saludo\_edad(nombre, edad):

    """

    Devuelve un mensaje distinto según la edad recibida:

    - < 18  : menor de edad

    - == 18 : acaba de cumplir la mayoría

    - > 18  : mayor de edad

    """

    # Determinamos el texto adecuado

    if edad < 18:

        titulo = "eres menor de edad"

    elif edad == 18:

        titulo = "acabás de cumplir la mayoría de edad"

    else:

        titulo = "eres mayor de edad"

    # Nota: devolvemos HTML en crudo porque es un ejemplo rápido

    return f"<h1>Hola {nombre}, tienes {edad} años, {titulo}.</h1>"

# --------------------------------------------------

# 4️⃣  Páginas “estáticas” (renderizadas con Jinja,

#     pero sin lógica adicional)

# --------------------------------------------------

@app.route("/contacto")

def contacto():

    """Muestra la página de contacto."""

    return render\_template("contacto.html")

@app.route("/servicios")

def servicios():

    """Muestra la página de servicios."""

    return render\_template("servicios.html")

# --------------------------------------------------

# 5️⃣  Lista de clientes

# --------------------------------------------------

@app.route("/clientes")

def clientes():

    """

    Genera una lista de clientes (hard-coded por ahora) y se la

    pasa a la plantilla 'clientes.html' para que Jinja la recorra.

    """

    lista\_clientes = [

        "Manolo García",

        "Ana Jiménez",

        "Andrés González",

        "Sonia Pérez",

    ]

    return render\_template(

        "clientes.html",

        name="Ángel",           # Para el saludo opcional de la plantilla

        clientes=lista\_clientes  # La lista que se mostrará

    )

# --------------------------------------------------

#  Punto de entrada cuando ejecutas “python main.py”

# --------------------------------------------------

if \_\_name\_\_ == "\_\_main\_\_":

    # debug=True recarga el servidor al guardar cambios y

    # muestra trazas completas si algo falla

    app.run(debug=True)

### Super()

Cuando una plantilla hija **sobrescribe** **({% block … %})** un bloque que ya tiene contenido en la plantilla padre, la llamada **{{ super() }}** te permite **insertar ese contenido original** y después añadir (o preceder) lo que quieras.  
Es la forma limpia de “heredar y extender” sin copiar-pegar.

A continuación, verás un ejemplo práctico **integrado en tu proyecto**:

Dentro del bloque **content** vamos a poner un pequeño mensaje genérico.  
(Así habrá “algo” que una hija pueda recuperar con **super()**.)

**Base.html**

<!-- templates/base.html -->

<!DOCTYPE html>

<html lang="es">

<head>

  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width, initial-scale=1">

  <title>{% block title %}Mi sitio{% endblock %}</title>

</head>

<body>

  {% include "\_navbar.html" %}

  <main>

    {% block content %}

      <!-- ✅  Contenido por defecto visible si la página hija NO lo

               sobrescribe —o accesible con super() si lo hace. -->

      <p style="color:gray">Contenido genérico proveniente de la plantilla base.</p>

    {% endblock %}

  </main>

</body>

</html>

Ahora, modificaremos, por ejemplo, **home.html** para que conserve ese texto y añada más:

<!-- templates/home.html -->

{% extends "base.html" %}

{% block title %}Inicio{% endblock %}

{% block content %}

  {{ super() }}  {# 👈  Inserta el párrafo gris definido en base.html #}

  <h1>Página de inicio</h1>

  {% if name %}

    <p>Hola, {{ name }} 👋</p>

  {% else %}

    <p>Hola, visitante</p>

  {% endif %}

{% endblock %}

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

# Filtros

Un **filtro** en las plantillas de Flask (Jinja2) es una función pequeñita que **recibe un dato, lo “pasa por un colador” y devuelve el mismo dato transformado**.  
Básicamente recibe datos, los transforma y los devuelve transformados.

En el HTML se usa con una “tubería” (|):

**{{ nombre | upper }} {# pone el nombre en MAYÚSCULAS #}**

**¿Qué filtros vienen “de fábrica”?**

|  |  |  |
| --- | --- | --- |
| Lo que hace | Ejemplo en plantilla | Resultado |
| upper – mayúsculas | `{{ "hola" | upper }}` | HOLA |
| lower – minúsculas | `{{ "HoLa" | lower }}` | hola |
| length – cuenta | `{{ [1,2] | length }}` | 2 |
| default – valor  por defecto | `{{ usuario.email | default("-sin email-") }}` | SI no se recibe email:  -sin email- |
| join – unir lista | `{{ ["a","b"] | join("‐") }}` | a-b |

Piensa en ellos como “efectos rápidos” que aplicas sobre el dato antes de imprimirlo.

Lista de filtros de la documentación oficial:

<https://jinja.palletsprojects.com/en/latest/templates/#list-of-builtin-filters>

Por ejemplo, en nuestro proyecto vamos a utilizar dos filtros, **upper** para pasar a mayúsculas y **reverse** para revertir la lista de clientes:

**Home.html**

{% extends "base.html" %}

{% block title %}Inicio{% endblock %}

{% block content %}

  <h1>Bienvenido a mi sitio Web</h1>

  {# ──────────── USO DE upper ──────────── #}

  {% if name %}

    {#  'name' llega desde main.py → lo mostramos en MAYÚSCULAS  #}

    <p>Hola, {{ name | upper }} 👋</p>

  {% else %}

    <p>Hola, visitante</p>

  {% endif %}

{% endblock %}

**Clientes.html**

{% extends "base.html" %}

{% block title %}Clientes{% endblock %}

{% block content %}

  <h1>Clientes</h1>

  {% if name %}

    <p>Hola, {{ name }} 😄</p>

  {% endif %}

  <h2>Lista de clientes</h2>

  <ul>

    {# ──────────── USO DE reverse ──────────── #}

    {#  El filtro muestra la lista en orden inverso                #}

    {% for cliente in clientes | reverse %}

      <li>{{ cliente }}</li>

    {% endfor %}

  </ul>

{% endblock %}

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

## Encadenar filtros (uno después de otro)

En caso de que necesitemso aplicar varios filtros sobre un elemento, se pueden encadenar uno detrás de otro para ahorrar código.

**Clientes.html**

{% extends "base.html" %}

{% block title %}Inicio{% endblock %}

{% block content %}

  <h1>Bienvenido a mi sitio Web</h1>

  {# ──────────── USO DE upper ──────────── #}

  {% if name %}

    {#  'name' llega desde main.py → lo mostramos en MAYÚSCULAS  #}

    <p>{{ name | upper | reverse}} 👋</p>

  {% else %}

    <p>Hola, visitante</p>

  {% endif %}

{% endblock %}

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

## Crear tu propio filtro paso a paso

En nuestro ejemplo vamos a incluir la fecha para que salga en pantalla.

En el **main.py**:

Lo primero será importar datetime para mostrar la fecha actual:

from datetime import datetime

Luego creamos una instancia de **datetime**

# --------------------------------------------------

# 1️⃣  Página de inicio

# --------------------------------------------------

@app.route("/")        # ruta raíz

@app.route("/home")    # alias /home → misma función

def home():

    """

    Renderiza la página de inicio.

    Pasamos la variable 'name' para saludar al usuario.

    """

    date = datetime.now()

    return render\_template("home.html", name="Ángel", date=date)

Ahora la mostramos en el **home.html**:

{% extends "base.html" %}

{% block title %}Inicio{% endblock %}

{% block content %}

<h1>Bienvenido a mi sitio Web</h1>

{# ──────────── USO DE upper ──────────── #}

{% if name %}

{# 'name' llega desde main.py → lo mostramos en MAYÚSCULAS #}

<p>{{ name | upper | reverse}} 👋</p>

{% else %}

<p>Hola, visitante</p>

{% endif %}

<hr>

<p>La fecha de hoy es: {{date}}</p>

{% endblock %}

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Pero nosotros no queremos este formato. Lo que queremos es simplemente día, mes, año. Así que vamos a crear un filtro personalizado que nos formatee la fecha.

Primero necesitamos crear la función que va a formatear la fecha y luego registrarla en nuestro proyecto como filtro.

Creamos la función en el **main.py**:

def today(date):

    return date.strftime('%d-%m-%Y')

Y ahora la registramos:

@app.add\_template\_filter

def today(date):

    return date.strftime('%d-%m-%Y')

Y ya podemos usar nuestro filtro personalizado al que hemos llamado **today**:

En el **home.html**:

{% extends "base.html" %}

{% block title %}Inicio{% endblock %}

{% block content %}

<h1>Bienvenido a mi sitio Web</h1>

{# ──────────── USO DE upper ──────────── #}

{% if name %}

{# 'name' llega desde main.py → lo mostramos en MAYÚSCULAS #}

<p>{{ name | upper | reverse}} 👋</p>

{% else %}

<p>Hola, visitante</p>

{% endif %}

<hr>

<p>La fecha de hoy es: {{date | today}}</p>

{% endblock %}

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Hemos registrado nuestro filtro usando un decorador:

@app.add\_template\_filter

def today(date):

    return date.strftime('%d-%m-%Y')

También podríamos haberlo hecho de la siguiente forma:

def today(date):

    return date.strftime('%d-%m-%Y')

app.add\_template\_filter(today, "today")

Poniendo primero el nombre de la función y luego cómo vamos a llamar a nuestro filtro. En nuestro caso de igual forma.

Ambas opciones son igualmente válidas.

# Funciones

Podemos crear nuestras propias funciones. Para lo cual sólo tenemos que definir la función y luego añadirla al **return**:

"""

main.py

========

Aplicación Flask sencilla con:

- Página de inicio

- Ejemplos de rutas dinámicas

- Páginas basadas en plantillas Jinja

- Listado de clientes

Para lanzar en consola:

    python main.py

o bien

    flask --app main run --debug

"""

from flask import Flask, render\_template  # importamos Flask y helper para plantillas

from datetime import datetime  # importamos datetime para mostrar la fecha actual

app = Flask(\_\_name\_\_)  # creamos la aplicación

# --------------------------------------------------

# 1️⃣  Página de inicio

# --------------------------------------------------

@app.route("/")        # ruta raíz

@app.route("/home")    # alias /home → misma función

def home():

    """

    Renderiza la página de inicio.

    Pasamos la variable 'name' para saludar al usuario.

    """

    date = datetime.now()

    return render\_template("home.html", name="Ángel", date=date, repeat=repeat)

# --------------------------------------------------

# 2️⃣  Saludo simple

# --------------------------------------------------

@app.route("/saludo\_personal/<nombre>")

def saludo\_personal(nombre):

    """

    Saluda al usuario con el nombre capturado en la URL.

    Ej.: /saludo\_personal/Ada -> “Bienvenido Ada”

    """

    return f"<h1>Bienvenido {nombre}</h1>"

# --------------------------------------------------

# 3️⃣  Saludo con edad

# --------------------------------------------------

@app.route("/saludo\_edad/<nombre>/<int:edad>")

def saludo\_edad(nombre, edad):

    """

    Devuelve un mensaje distinto según la edad recibida:

    - < 18  : menor de edad

    - == 18 : acaba de cumplir la mayoría

    - > 18  : mayor de edad

    """

    # Determinamos el texto adecuado

    if edad < 18:

        titulo = "eres menor de edad"

    elif edad == 18:

        titulo = "acabás de cumplir la mayoría de edad"

    else:

        titulo = "eres mayor de edad"

    # Nota: devolvemos HTML en crudo porque es un ejemplo rápido

    return f"<h1>Hola {nombre}, tienes {edad} años, {titulo}.</h1>"

# --------------------------------------------------

# 4️⃣  Páginas “estáticas” (renderizadas con Jinja,

#     pero sin lógica adicional)

# --------------------------------------------------

@app.route("/contacto")

def contacto():

    """Muestra la página de contacto."""

    return render\_template("contacto.html")

@app.route("/servicios")

def servicios():

    """Muestra la página de servicios."""

    return render\_template("servicios.html")

# --------------------------------------------------

# 5️⃣  Lista de clientes

# --------------------------------------------------

@app.route("/clientes")

def clientes():

    """

    Genera una lista de clientes (hard-coded por ahora) y se la

    pasa a la plantilla 'clientes.html' para que Jinja la recorra.

    """

    lista\_clientes = [

        "Manolo García",

        "Ana Jiménez",

        "Andrés González",

        "Sonia Pérez",

    ]

    return render\_template(

        "clientes.html",

        name="Ángel",           # Para el saludo opcional de la plantilla

        clientes=lista\_clientes  # La lista que se mostrará

    )

# --------------------------------------------------

#  Punto de entrada cuando ejecutas “python main.py”

# --------------------------------------------------

if \_\_name\_\_ == "\_\_main\_\_":

    # debug=True recarga el servidor al guardar cambios y

    # muestra trazas completas si algo falla

    app.run(debug=True)

# --------------------------------------------------

# 💻  Filtro personalizado

# --------------------------------------------------

@app.add\_template\_filter

def today(date):

    return date.strftime('%d-%m-%Y')

# --------------------------------------------------

# ✍️  Funciones propias

#     Repetirá un String N veces

# --------------------------------------------------

def repeat(s, n):

    return s \* n

Y ya podemos usar nuestra nueva función en nuestras plantillas:

En el **index.html**:

{% extends "base.html" %}

{% block title %}Inicio{% endblock %}

{% block content %}

<h1>Bienvenido a mi sitio Web</h1>

{# ──────────── USO DE upper ──────────── #}

{% if name %}

{# 'name' llega desde main.py → lo mostramos en MAYÚSCULAS #}

<p>{{ name | upper | reverse}} 👋</p>

{% else %}

<p>Hola, visitante</p>

{% endif %}

<hr>

<p>La fecha de hoy es: {{date | today}}</p>

<p>La frase repetida es: {{repeat("Hola mundo ", 3)}}</p>

{% endblock %}

Texto

El contenido generado por IA puede ser incorrecto.

Hemos registrado una función mandándola a las plantillas mediante su inclusión en el **return**. Hay otro modo de hacerlo sin tener que hacer este paso, usando un decorador:

@app.add\_template\_global

def repeat(s, n):

    return s \* n

Si usamos esta opción no hay que incluir la función en el **return**.

E incluso hay una tercera forma de hacerlo parecida a la forma en que creábamos los filtros:

def repeat(s, n):

    return s \* n

@app.add\_template\_global(repeat, "repeat")

Las tres formas son igualmente válidas. Yo os recomiendo la del decorador, es más simple.

# Formularios

Los formularios son una parte importante de cualquier aplicción web, ya que permiten recibir y enviar información del usuario.

En Flask, los formularios se pueden crear usando HTML y CSS, pero también existen bibliotecas que facilitan el proceso y añaden características adicionales como la validación de datos y la protección contra ataques.

Podemos crear un formulario al modo tradicional, usando html y css:

Creamos una plantilla nueva:

**Formulario.html**

{% extends "base.html" %}

{% block title %}Servicios{% endblock %}

{% block content %}

<h1>Formulario de alta</h1>

<form action="" method="post">

  <label for="username">Nombre de usuario</label>

  <input type="text" name="username" id="username" required>

  <br><br>

  <label for="password">Contraseña</label>

  <input type="password" name="password" id="password" required>

  <br><br>

  <input type="submit" value="Registrar">

</form>

{% endblock %}

En el **main.py** necesitaremos importar el objeto **request** para capturar todo lo que introduzca el usuario en el formulario y crear una ruta nueva hasta la plantilla del formulario.

**Main.py**

@app.route("/formulario", methods=["GET", "POST"])

def formulario():

    """

    Muestra el formulario de alta de clientes.

    Si se envía el formulario, lo procesa y muestra un mensaje de éxito.

    """

    if request.method == "POST":

        # Aquí procesarías los datos del formulario

        # Por ejemplo, guardar en la base de datos

        username = request.form['username']

        password = request.form['password']

        return f"Nombre de usuario: {username}, Contraseña: {password}"

    return render\_template("formulario.html", success=True)

    # Si no se envía el formulario, simplemente lo mostramos

def formulario():

    """Muestra la página del formulario de alta."""

    return render\_template("formulario.html")

También podemos usar alguna de las librerías especializadas en formularios que nos brinda Flask. Por ejemplo, **WTForm**.

Primero deberamos instalar la biblioteca:

**pip install flask flask-wtf**

Volvemos a ejecutar nuestro proyecto:

**flask --app main run –debug**

Deberemos importar los módulos que necesitemos de la librería WTForm, en nuestro caso el formulario tendrá usuario, contraseña y botón submit.

También añadiremos las validaciones que nos interesen. WTForm contiene todas las validaciones que podamos necesitar:

from flask\_wtf import FlaskForm

from wtforms import StringField, PasswordField, SubmitField

from wtforms.validators import DataRequired, Length

class RegisterForm(FlaskForm):

    username = StringField("Nombre de usuario", validators=[

        DataRequired(message="Este campo es obligatorio"),

        Length(min=4, max=25, message="El nombre debe tener entre 4 y 25 caracteres")

    ])

    password = PasswordField("Contraseña", validators=[

        DataRequired(message="Este campo es obligatorio"),

        Length(min=6, max=40, message="La contraseña debe tener entre 6 y 40 caracteres")

    ])

    submit = SubmitField("Registrar")

Ahora tenemos que crear dentro de nuestro formulario una instancia del objeto **RegisterForm** que hemos creado:

@app.route("/formulario", methods=["GET", "POST"])

def formulario():

    form = RegisterForm()

    """

    Muestra el formulario de alta de clientes.

    Si se envía el formulario, lo procesa y muestra un mensaje de éxito.

    """

    if request.method == "POST":

        # Aquí procesarías los datos del formulario

        # Por ejemplo, guardar en la base de datos

        username = request.form['username']

        password = request.form['password']

        return f"Nombre de usuario: {username}, Contraseña: {password}"

    return render\_template("formulario.html", success=True, form = form)

    # Si no se envía el formulario, simplemente lo mostramos

def formulario():

    """Muestra la página del formulario de alta."""

    return render\_template("formulario.html")

**formulario.html**

<form action="" method="post">

  {{ form.username.label }} {{ form.username }} <br>

  {{ form.password.label }} {{ form.password }} <br>

  {{ form.submit }}

</form>