

# 1. Fundamentos de Python para el Desarrollo Web

Inicio



# Fundamentos de Python el Desarrollo Web



# 1. Introducción a Python

---



SECRETARÍA DE  
INNOVACIÓN



## Introducción a Python



### Contenido

---

Python es un lenguaje de programación de alto nivel, versátil y ampliamente utilizado, conocido por su simplicidad y legibilidad. Su diseño promueve la escritura de código claro y eficiente, lo que lo hace ideal tanto para principiantes como para desarrolladores experimentados. A continuación, exploraremos algunos de los fundamentos más importantes de Python.

#### 1. Sintaxis Clara y Concisa

La sintaxis de Python es simple y directa, con un enfoque en la legibilidad. En lugar de utilizar llaves {} para delimitar bloques de código, se emplea la indentación, lo que obliga a una estructura limpia y uniforme.

Ejemplo:

```
if x > 0:  
    print("El número es positivo")  
else:  
    print("El número es negativo o cero")
```

#### 2. Tipos de Datos Básicos

Python admite varios tipos de datos básicos:

- **Numéricos:** int (enteros), float (decimales), complex (números complejos).
- **Cadenas de texto:** str, que representan secuencias de caracteres.
- **Booleanos:** bool, que pueden ser True o False.
- **Colecciones:** list, tuple, dict, set.

Ejemplo:

```
edad = 25 # int
nombre = "Ana" # str
es_estudiante = True # bool
promedio = 4.5 # float
```

### 3. Variables y Asignación

Las variables en Python no requieren declaración previa de tipo. Puedes asignarles un valor directamente, y Python determina su tipo automáticamente.

Ejemplo:

```
x = 10 # Variable de tipo entero
y = "Hola Mundo" # Variable de tipo cadena
```

### 4. Estructuras de Control

Python permite el uso de estructuras de control como condicionales (if, elif, else), bucles (for, while) y control de flujo (break, continue).

**Condicionales:**

```
if x > 0:
    print("Positivo")
elif x == 0:
    print("Cero")
else:
    print("Negativo")
```

**Bucles:**

```
for i in range(5):
    print(i)
```

## 5. Funciones

Las funciones se definen con la palabra clave `def` y permiten organizar el código en bloques reutilizables.

Ejemplo:

```
def saludar(nombre):
    return f"Hola, {nombre}!"

print(saludar("Juan"))
```

## 6. Manejo de Errores

El manejo de errores en Python se realiza mediante bloques `try`, `except` y, opcionalmente, `finally`.

Ejemplo:

```
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("No se puede dividir entre cero")
finally:
    print("Bloque finalizado")
```

## 7. Bibliotecas Estándar

Python incluye una biblioteca estándar rica que facilita tareas comunes, desde manipulación de archivos hasta cálculos matemáticos avanzados.

Ejemplo con `math`:

```
import math
print(math.sqrt(16)) # Imprime la raíz cuadrada de 16
```

## 8. Programación Orientada a Objetos (POO)

Python soporta paradigmas orientados a objetos, permitiendo la definición de clases, atributos y métodos.

Ejemplo:

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

    def saludar(self):
        return f"Hola, me llamo {self.nombre} y tengo {self.edad} años.

persona = Persona("Ana", 30)
print(persona.saludar())
```



## 9. Iterables y Comprensión de Listas

Python facilita el manejo de datos con iterables y permite crear estructuras como listas de forma compacta.

Ejemplo de comprensión de listas:

```
numeros = [x**2 for x in range(10)]
print(numeros) # Lista de cuadrados del 0 al 9
```

## 10. Versatilidad y Comunidad

Python es adecuado para múltiples áreas:

1. Desarrollo web con frameworks como Django o Flask.

2. Ciencia de datos con bibliotecas como NumPy, pandas y matplotlib.

3. Inteligencia artificial con TensorFlow y PyTorch.

4. Automatización y scripting.

Además, cuenta con una comunidad activa que contribuye a su mejora continua y a la creación de recursos educativos.

---

## 2. Framework y Bases de Datos Soportados por Python para la Web

---



### Contenido

---

Python es un lenguaje ampliamente utilizado en el desarrollo web gracias a su facilidad de uso, versatilidad y la disponibilidad de frameworks robustos y eficientes. Además, su soporte para una gran variedad de bases de datos permite construir aplicaciones web dinámicas y escalables. A continuación, exploraremos los frameworks más destacados y las bases de datos comúnmente utilizadas con Python para el desarrollo web.

#### 1. Frameworks Full-Stack

Estos frameworks ofrecen un conjunto completo de herramientas para desarrollar aplicaciones web, incluyendo soporte para la capa de datos, vistas, controladores y plantillas.

##### Django:

- Framework robusto y completo, conocido por seguir el principio "Don't Repeat Yourself" (DRY).
- Incluye ORM (Object-Relational Mapping), sistema de autenticación, administración automatizada y más.
- Ideal para aplicaciones grandes y complejas.

```
from django.shortcuts import render
```

```
def home(request):
    return render(request, 'home.html', {'mensaje': 'Hola Mundo'})
```

### Pyramid:

- Más flexible que Django, permite mayor personalización.
- Adecuado para proyectos en los que se requiera elegir herramientas específicas en lugar de un enfoque todo-en-uno.

## 2. Microframeworks

Los microframeworks son ligeros y no imponen una estructura específica. Son ideales para aplicaciones más pequeñas o si se requiere flexibilidad en la elección de herramientas.

### Flask:

- Sencillo y minimalista.
- No incluye un ORM ni herramientas específicas para la capa de datos; puedes integrar las que prefieras.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return 'Hola, Mundo!'

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

### Bottle:

- Extremadamente liviano y fácil de usar.
- Ideal para aplicaciones muy pequeñas o APIs REST.

### FastAPI:

- Diseñado para construir APIs rápidas y eficientes.
- Soporte nativo para OpenAPI y JSON Schema.

- Excelente para aplicaciones que priorizan rendimiento y compatibilidad con estándares modernos.

## Bases de Datos Soportadas por Python

Python tiene soporte para bases de datos relacionales y no relacionales a través de librerías y herramientas ORM.

### 1. Bases de Datos Relacionales

Las bases de datos relacionales estructuran los datos en tablas y utilizan SQL para gestionarlos.

#### MySQL/MariaDB:

- Soporte a través de librerías como mysql-connector-python o PyMySQL.
- Django incluye soporte nativo para MySQL.

#### PostgreSQL:

- Base de datos avanzada y robusta, conocida por su soporte a consultas complejas y extensiones como PostGIS.
- Django y SQLAlchemy tienen integración nativa con PostgreSQL.

#### SQLite:

- Base de datos liviana que no requiere servidor.
- Excelente para prototipos y aplicaciones pequeñas.
- Incluida por defecto en Python (sqlite3).

#### Oracle y Microsoft SQL Server:

Soporte a través de librerías específicas como cx\_Oracle y pyodbc.

### 2. Bases de Datos No Relacionales

Para aplicaciones que requieren flexibilidad en el esquema de datos o alto rendimiento en operaciones específicas.

#### MongoDB:

- Base de datos orientada a documentos.
- Soporte en Python a través de pymongo y frameworks como Flask o FastAPI.

### **Redis:**

- Base de datos clave-valor, ideal para caché y almacenamiento de sesiones.
- Usada frecuentemente junto con frameworks como Django y Flask.

### **Cassandra:**

- Base de datos distribuida para manejar grandes volúmenes de datos.
- Compatible con Python mediante cassandra-driver.

### **Firebase/Firestore:**

- Base de datos basada en la nube, muy utilizada en aplicaciones móviles.
- Acceso en Python mediante bibliotecas como firebase-admin.

## **3. ORM en Python**

Los ORM (Object-Relational Mappers) simplifican la interacción con bases de datos relacionales al permitir trabajar con objetos Python en lugar de escribir SQL manualmente.

**Django ORM:** Incluido en Django; permite definir modelos y relaciones directamente en Python.

```
from django.db import models

class Usuario(models.Model):
    nombre = models.CharField(max_length=100)
    correo = models.EmailField()
```

### **SQLAlchemy:**

- Librería ORM independiente, compatible con múltiples bases de datos.
- Ofrece mayor flexibilidad y control en comparación con Django ORM.

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class Usuario(Base):
```

```
__tablename__ = 'usuarios'  
id = Column(Integer, primary_key=True)  
nombre = Column(String)
```

### Tortoise ORM:

- Diseñado para aplicaciones asíncronas.
  - Popular en proyectos que usan FastAPI o Sanic.
-

### 3. Introducción a Flask

---



SECRETARÍA DE  
INNOVACIÓN



## Introducción a Flask



### Contenido

---

Flask es un microframework de Python que permite construir aplicaciones web de manera rápida y sencilla. Fue diseñado con un enfoque minimalista, ofreciendo solo lo esencial para desarrollar aplicaciones, mientras deja al desarrollador la libertad de elegir las herramientas adicionales necesarias.

#### Características Principales de Flask

**Ligero y Extensible:** Flask no incluye un ORM ni un sistema de autenticación por defecto, lo que lo hace ideal para aplicaciones donde el desarrollador quiere tener control total sobre las herramientas utilizadas.

**Basado en Werkzeug y Jinja2:** Werkzeug: Una biblioteca WSGI que maneja solicitudes HTTP y enrutamiento. Jinja2: Un motor de plantillas para crear HTML dinámico.

**Diseño Modular:** Permite agregar extensiones según las necesidades del proyecto, como autenticación, bases de datos y gestión de sesiones.

**Fácil de Aprender y Usar:** Su sintaxis clara y concisa lo convierte en una excelente opción para principiantes y proyectos pequeños.

**Soporte para RESTful:** Ideal para construir APIs REST gracias a su diseño flexible y modular.

#### Primeros Pasos con Flask

**Instalación:** Antes de comenzar, asegúrate de tener instalado Python y luego instala Flask mediante pip:

```
pip install flask
```

**Estructura Básica de una Aplicación Flask:** Una aplicación Flask básica puede definirse en unos pocos pasos.

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "¡Hola, Mundo!"

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

- **Flask:** Se importa la clase principal del framework.
- **app = Flask(\_\_name\_\_):** Se crea una instancia de la aplicación.
- **Decorador @app.route:** Define las rutas (URLs) que manejará la aplicación.
- **app.run(debug=True):** Inicia el servidor web en modo de desarrollo.

## Manejo de Rutas

Las rutas en Flask son las URLs que una aplicación puede procesar. Se definen utilizando el decorador @app.route.

### Definiendo una Ruta

```
@app.route('/saludo')
def saludo():
    return "¡Bienvenido a Flask!"
```

### Rutas Dinámicas

Puedes usar variables en las rutas para personalizar la respuesta.

```
@app.route('/usuario/<nombre>')
def usuario(nombre):
    return f"Hola, {nombre}!"
```

## Métodos HTTP

Flask admite métodos HTTP como GET, POST, PUT y DELETE. Puedes especificar el método permitido para una ruta.

```
@app.route('/formulario', methods=['POST'])
def formulario():
    return "Procesando datos enviados por POST"
```

## Renderizado de Plantillas

Flask usa Jinja2 para renderizar HTML dinámico.

**Creando una Plantilla:** Guarda un archivo llamado index.html en un directorio llamado templates:

```
<!DOCTYPE html>
<html>
<head>
    <title>Inicio</title>
</head>
<body>
    <h1>¡Hola, {{ nombre }}!</h1>
</body>
</html>
```

## Renderizando la Plantilla

Usa render\_template para pasar datos a la plantilla.

```
from flask import render_template

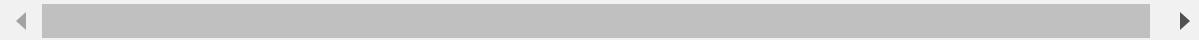
@app.route('/inicio/<nombre>')
def inicio(nombre):
    return render_template('index.html', nombre=nombre)
```

## Manejo de Formularios

Flask permite manejar formularios web y recuperar los datos enviados.

### Formulario HTML

```
<form action="/procesar" method="post">
    <input type="text" name="usuario" placeholder="Nombre de usuario">
    <button type="submit">Enviar</button>
</form>
```



### Procesar Datos del Formulario

Usa request para obtener los datos enviados.

```
from flask import request

@app.route('/procesar', methods=['POST'])
def procesar():
    usuario = request.form['usuario']
    return f"Usuario recibido: {usuario}"
```

## Extensiones Útiles y Uso de Flask

Flask tiene muchas extensiones que amplían sus capacidades. Algunas populares son:

- **Flask-WTF:** Manejo avanzado de formularios.
- **Flask-RESTful:** Construcción de APIs REST.
- **Flask-Migrate:** Migraciones para bases de datos.

- **Flask-Login:** Gestión de autenticación de usuarios.

Flask es ideal en los siguientes casos:

- Proyectos pequeños o medianos que requieren flexibilidad.
- Aplicaciones donde el control sobre las herramientas y la arquitectura es importante.
- APIs RESTful y servicios en la nube.
- Prototipos rápidos y pruebas de concepto.

Para proyectos más grandes o con requerimientos avanzados, podrías considerar un framework más completo como Django. Sin embargo, la versatilidad y la simplicidad de Flask lo hacen una excelente opción para muchos desarrolladores.

---

## 4. Instalación de los Recursos Necesarios

---



SECRETARÍA DE  
INNOVACIÓN



# Recursos Necesarios



## Contenido

---

A lo largo de la unidad trabajaremos con Python y Flask, antes de iniciar, primero instalaremos los recursos necesarios para crear nuestra WebApp:

1. Instalación de Python versión 3.11 (se utilizará esta versión para evitar problemas de compatibilidad).
2. Instalación de Visual Studio Code como entorno de desarrollo.

### 1. Descargando el interprete de Python

Visitamos el sitio oficial de Python ([puedes dar clic aquí](#)) <https://www.python.org/downloads/windows/>, también puedes descargar la versión 3.11.9 del interprete de forma directa haciendo clic aquí <https://www.python.org/ftp/python/3.12.3/python-3.12.3-amd64.exe> para versión de 64 bits, y haciendo clic aquí <https://www.python.org/ftp/python/3.11.9/python-3.11.9.exe> para versión de 32 bits (recuerda que esta depende de la versión de tu equipo).

Ejemplo de tipo de sistema:

## Sistema > Información

DESKTOP-MIPRGBM  
Latitude 3420

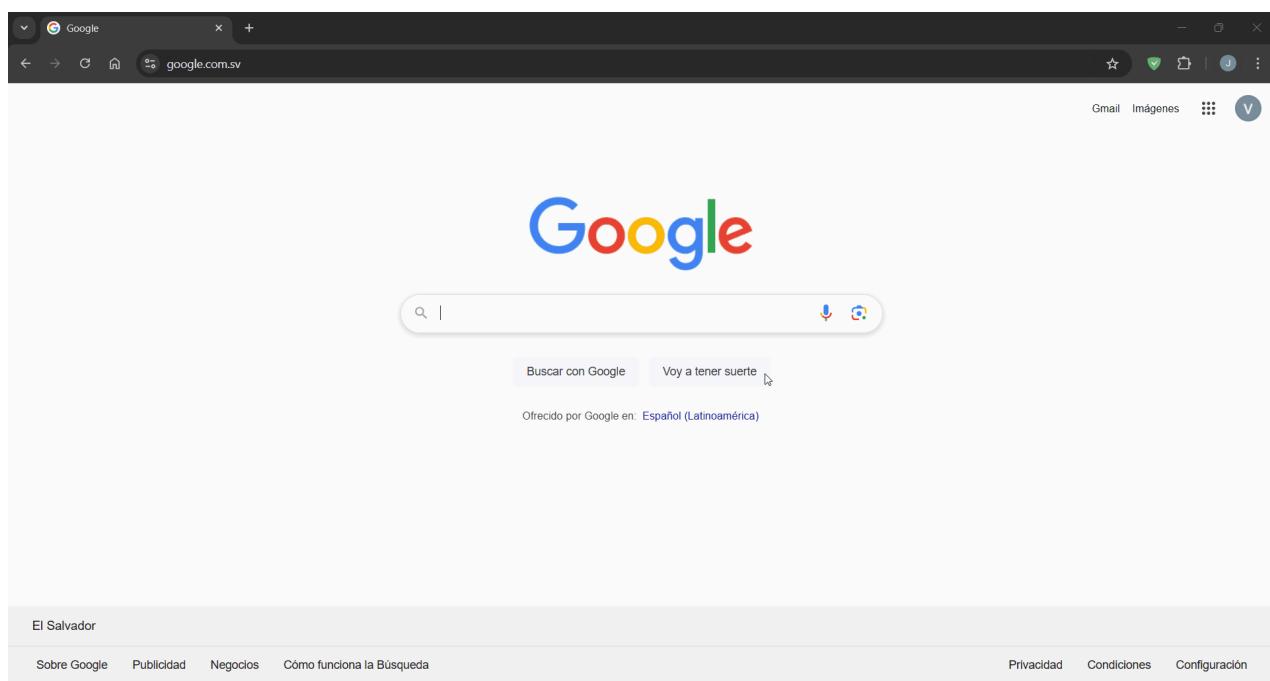
Cambiar el nombre de este equipo

Especificaciones del dispositivo

Copiar ^

Nombre del dispositivo	DESKTOP-MIPRGBM
Procesador	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 1.38 GHz
RAM instalada	8.00 GB (7.74 GB utilizable)
Id. del dispositivo	4F2D478C-9247-4880-87E2-FE824ECC6351
Id. del producto	00330-53950-66545-AAOEM
Tipo de sistema	Sistema operativo de 64 bits, procesador x64
Lápiz y entrada táctil	Compatibilidad con entrada táctil con 10 puntos táctiles

## Proceso de Descarga:



## Instalando interprete de Python

Procedemos a instalar a versión:

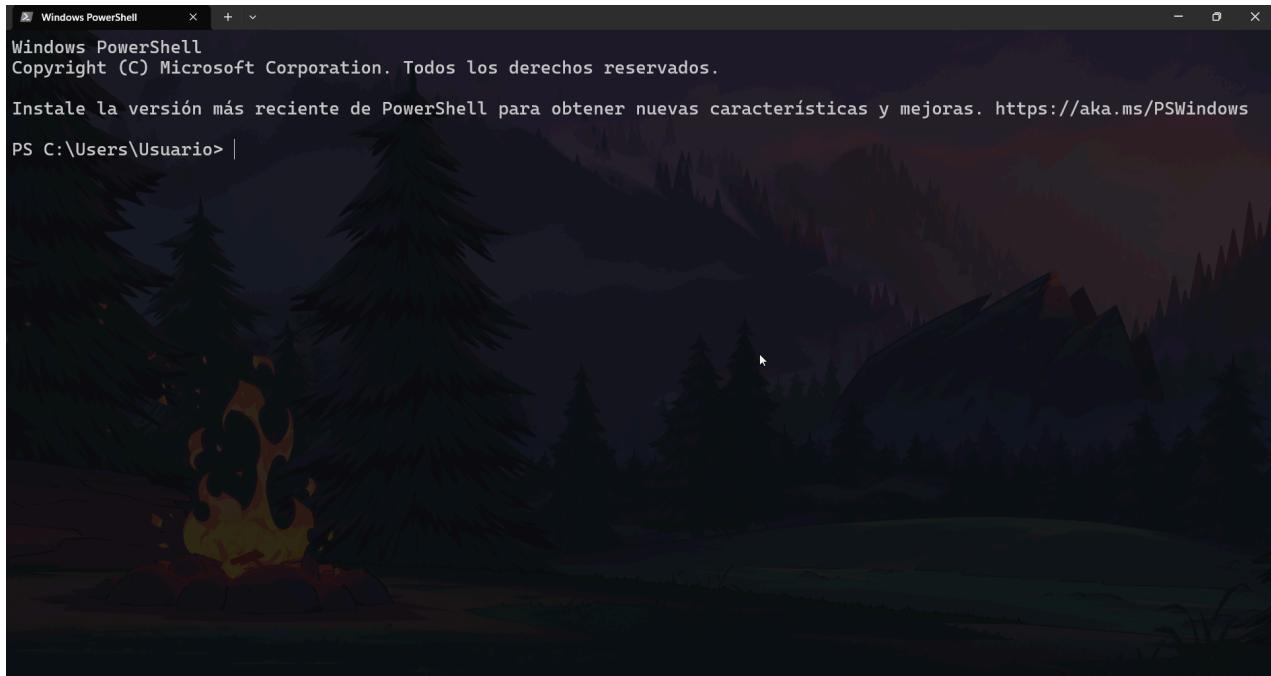
The screenshot shows a list of Python releases for Windows, organized by release date. Each entry includes a link to the Windows installer and embeddable package for both 64-bit and 32-bit architectures, as well as ARM64 for some versions.

- Python 3.12.3 - April 9, 2024
  - Download Windows installer (64-bit)
  - Download Windows installer (32-bit)
  - Download Windows installer (ARM64)
  - Download Windows embeddable package (64-bit)
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (ARM64)
- Python 3.11.9 - April 2, 2024
  - Note that Python 3.11.9 cannot be used on Windows 7 or earlier.
  - Download Windows installer (64-bit)
  - Download Windows installer (32-bit)
  - Download Windows installer (ARM64)
  - Download Windows embeddable package (64-bit)
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (ARM64)
- Python 3.10.14 - March 19, 2024
  - Note that Python 3.10.14 cannot be used on Windows 7 or earlier.
  - No files for this release.
- Python 3.9.19 - March 19, 2024
  - Note that Python 3.9.19 cannot be used on Windows 7 or earlier.
  - No files for this release.
- Python 3.8.19 - March 19, 2024

**Nota: recuerda marcar la casilla de ADD PATH, para que puedas usar los comandos de Python y PIP.**

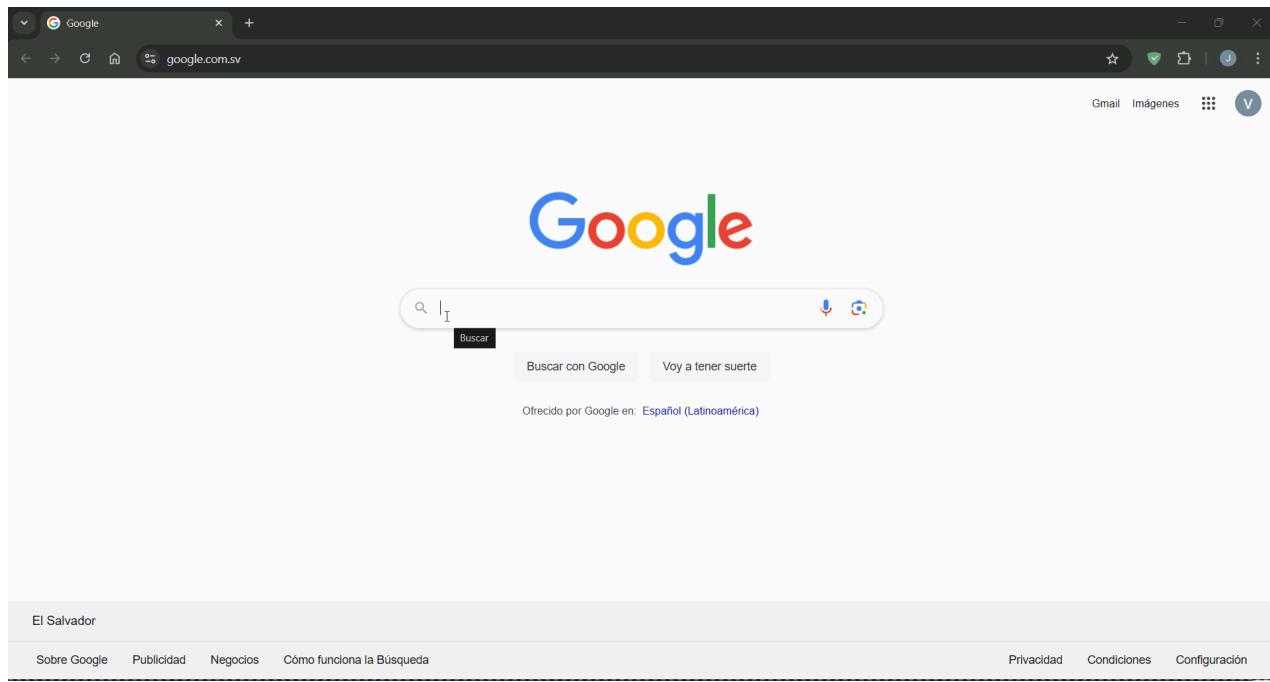
## Verificando la versión del interprete

Abrimos una terminal de Windows y escribimos python --version:



## Descargando e Instalando VSCode

Ahora procedemos a descargar e instalar Visual Studio Code:



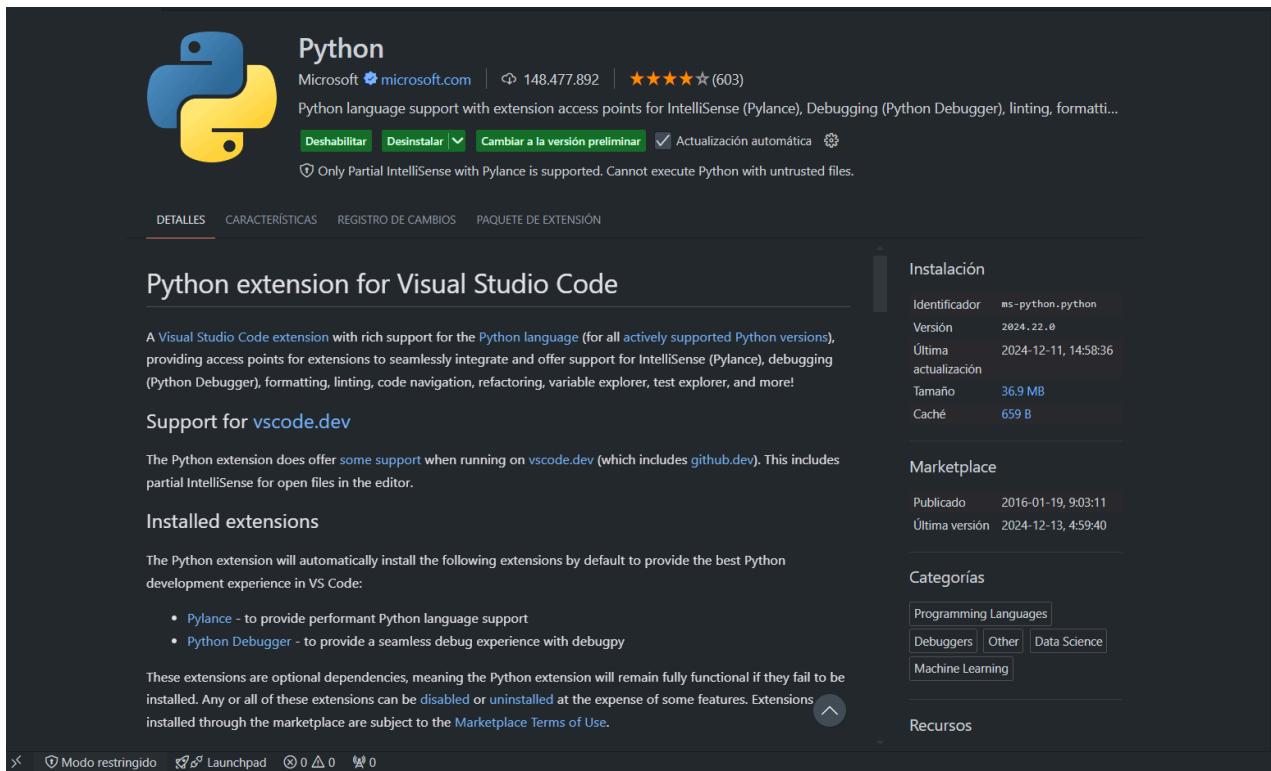
# 4.1 Extensiones para VSCode

## Contenido

Te compartimos extensiones que nos serán de utilidad para trabajar con Python y Flask:

### Python (Obligatoria)

Soporte completo para el lenguaje Python en Visual Studio Code. Incluye IntelliSense, depuración, formateo, linting, navegación y refactorización de código.



### Jinja (Obligatorio)

Es una herramienta que mejora la experiencia al trabajar con plantillas Jinja en el editor de código Visual Studio Code. Estas plantillas son comúnmente usadas en frameworks como Flask, Django, y otros entornos que utilizan el motor de plantillas Jinja2.

Extensión: **Jinja**

The screenshot shows the Jinja extension page in the Visual Studio Code Marketplace. At the top, there's a large icon of a traditional Japanese torii gate. Below it, the extension name "Jinja" is displayed in a large font, followed by the developer name "wholroyd" and a rating of 11.558.787 stars (17 reviews). A brief description states "Jinja template language support for Visual Studio Code". There are buttons for "Deshabilitar" (Disable) and "Desinstalar" (Uninstall), and a checked checkbox for "Actualización automática" (Automatic updates). The main content area is titled "Jinja for Visual Studio Code" and contains a "gitter" button and a "join chat" button. It describes the extension as adding language colorization support for the Jinja template language. Below this is a screenshot of VS Code showing a file named "example.j2" containing Jinja template code. To the right, there are sections for "Instalación" (Installation) with details like Identifier, Version, Last update, and Size; "Marketplace" history showing publication and update times; and "Categorías" (Categories) with "Languages" and "Snippets" selected.

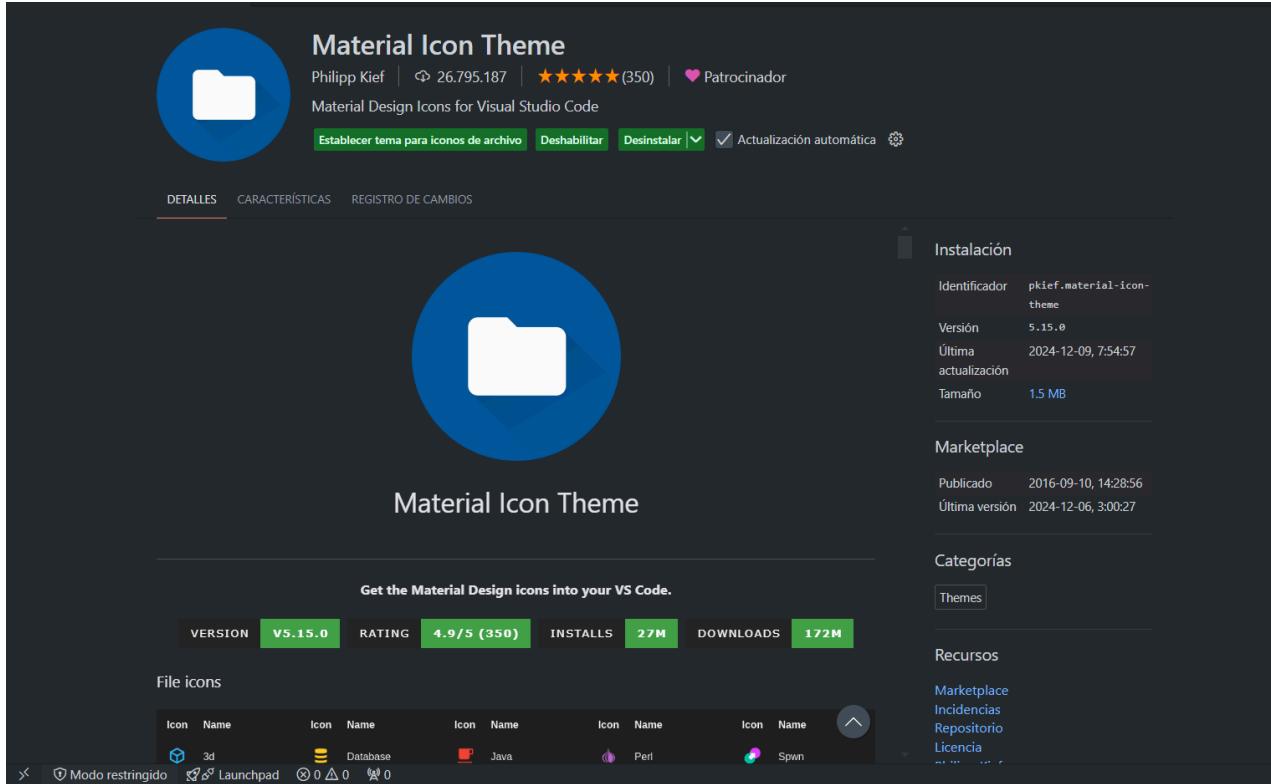
## Spanish Language Pack (Opcional)

Permite cambiar el idioma de la interfaz de VS Code al español.

The screenshot shows the Spanish Language Pack extension page in the Visual Studio Code Marketplace. It features a globe icon. The extension name is "Spanish Language Pack for Visual Studio Code" by Microsoft, with a rating of 8.676.191 stars (15 reviews). A brief description says it's a language pack extension for Spanish. There are buttons for "Desinstalar" (Uninstall) and "Actualización automática" (Automatic updates). The main content area is titled "Paquete de idioma español para VS Code" and explains that the Spanish language pack provides localized UI for VS Code. It includes sections for "Uso" (Usage), "Cómo contribuir" (How to contribute), "Licencia" (License), and "Reconocimientos" (Acknowledgments). To the right, there are sections for "Instalación" (Installation) with details like Identifier, Version, Last update, and Size; "Marketplace" history; "Categorías" (Categories) with "Language Packs" selected; and a "Recursos" (Resources) section listing "Marketplace", "Incidencias", "Repositorio", and "Microsoft".

## Material Icon Theme (Opcional)

Personaliza los íconos de archivos y carpetas en VS Code con un diseño moderno basado en Material Design.



## Prettier - Code Formatter (Opcional)

Formatea automáticamente el código según reglas de estilo predefinidas. Compatible con lenguajes como JavaScript, TypeScript, HTML, CSS, y más.

**Prettier - Code formatter**

Prettier [prettier.io](#) | ⚡ 52.124.005 | ★★★★★(467) | ❤ Patrocinador

Code formatter using prettier

[Deshabilitar](#) [Desinstalar](#)  Actualización automática

Only the built-in version of Prettier will be used when running in untrusted mode.

**DETALLES** [CARACTERÍSTICAS](#) [REGISTRO DE CAMBIOS](#)

## Prettier Formatter for Visual Studio Code

Prettier is an opinionated code formatter. It enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary.

JavaScript · TypeScript · Flow · JSX · JSON  
CSS · SCSS · Less  
HTML · Vue · Angular · HANDLEBARS · Ember · Glimmer  
GraphQL · Markdown · YAML  
Your favorite language?

[Main](#) no status | [downloads](#) 244M | [installs](#) 52M | [code style](#) prettier | [follow prettier](#)

### Installation

Install through VS Code extensions. Search for Prettier - Code formatter

Visual Studio Code Market Place: Prettier - Code formatter

Can also be installed in VS Code: Launch VS Code Quick Open (Ctrl+P), paste the following command, and press enter.

```
ext install esbenp.prettier-vscode
```

> ⌂ Modo restringido ⌂ Launchpad ⌂ 0 △ 0 ⌂ 0

**Instalación**

Identificador	esbenp.prettier-vscode
Versión	11.0.0
Última actualización	2024-08-15, 13:33:50
Tamaño	9.6 MB

**Marketplace**

Publicado	2017-01-10, 13:52:02
Última versión	2024-08-14, 9:16:38

**Categorías**

Formatters
------------

**Recursos**

Marketplace
Incidencias
Repositorio

## TODO Highlight (Opcional)

Resalta palabras clave como TODO y FIXME en el código para identificar tareas pendientes o notas importantes.

**TODO Highlight**

Wayou Liu | ⚡ 5.065.801 | ★★★★★(113)

highlight TODOs, FIXMEs, and any keywords, annotations...

[Deshabilitar](#) [Desinstalar](#)  Actualización automática

**DETALLES** [CARACTERÍSTICAS](#) [REGISTRO DE CAMBIOS](#)

## VSCODE-TODO-HIGHLIGHT

License [MIT](#) build [Unknown](#)

Highlight TODO, FIXME and other annotations within your code.

Sometimes you forget to review the TODOs you've added while coding before you publish the code to production. So I've been wanting an extension for a long time that highlights them and reminds me that there are notes or things not done yet.

Hope this extension helps you as well.

**NOTICE**

Many report that the `List highlighted annotations` command is not working, make sure you have the file types included via `todohighlight.include`.

**Preview**

- with `material night` color theme:

> ⌂ Launchpad ⌂ 0 △ 0 ⌂ 0

**Marketplace**

Identificador	wayou.vscode-todo-highlight
Versión	1.0.5
Publicado	2016-12-22, 8:16:28
Última versión	2021-10-28, 21:31:46

**Categorías**

Other
-------

**Recursos**

Marketplace
Incidencias
Repositorio
Licencia
Wayou Liu

## Todo Tree (Opcional)

Busca y organiza tareas TODO y FIXME en una vista de árbol dentro de la barra de actividad. Permite navegar fácilmente a las anotaciones destacadas.

## Todo Tree

Gruntfuggly | ⌂ 5.397.588 | ★★★★★(184)

Show TODO, FIXME, etc. comment tags in a tree view

[Deshabilitar](#) [Desinstalar](#)  Actualización automática

[DETALLES](#) [CARACTERÍSTICAS](#) [REGISTRO DE CAMBIOS](#)

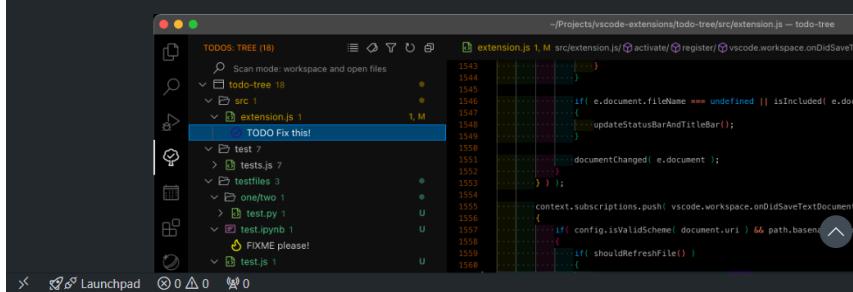
### Todo Tree

This extension quickly searches (using [ripgrep](#)) your workspace for comment tags like TODO and FIXME, and displays them in a tree view in the activity bar. The view can be dragged out of the activity bar into the explorer pane (or anywhere else you would prefer it to be).

Clicking a TODO within the tree will open the file and put the cursor on the line containing the TODO.

Found TODOs can also be highlighted in open files.

Please see the [wiki](#) for configuration examples.



Instalación

Identificador	gruntfuggly.todo-tree
Versión	0.0.226
Última actualización	2024-12-16, 11:33:40
Tamaño	1.6 MB

Marketplace

Publicado	2017-11-23, 16:58:27
Última versión	2023-04-12, 12:43:52

Categorías

- Other

Recursos

- Marketplace
- Incidencias
- Repositorio
- Licencia

# 5. Creando el Espacio de Desarrollo

---

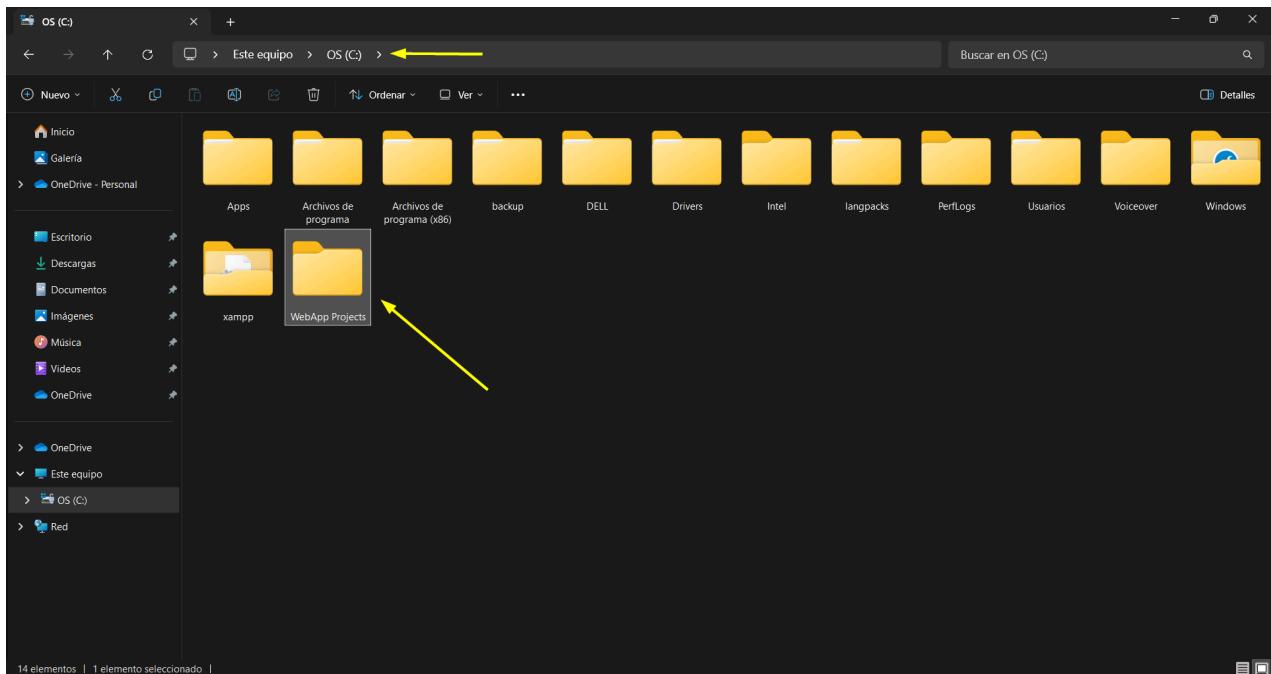


## Contenido

---

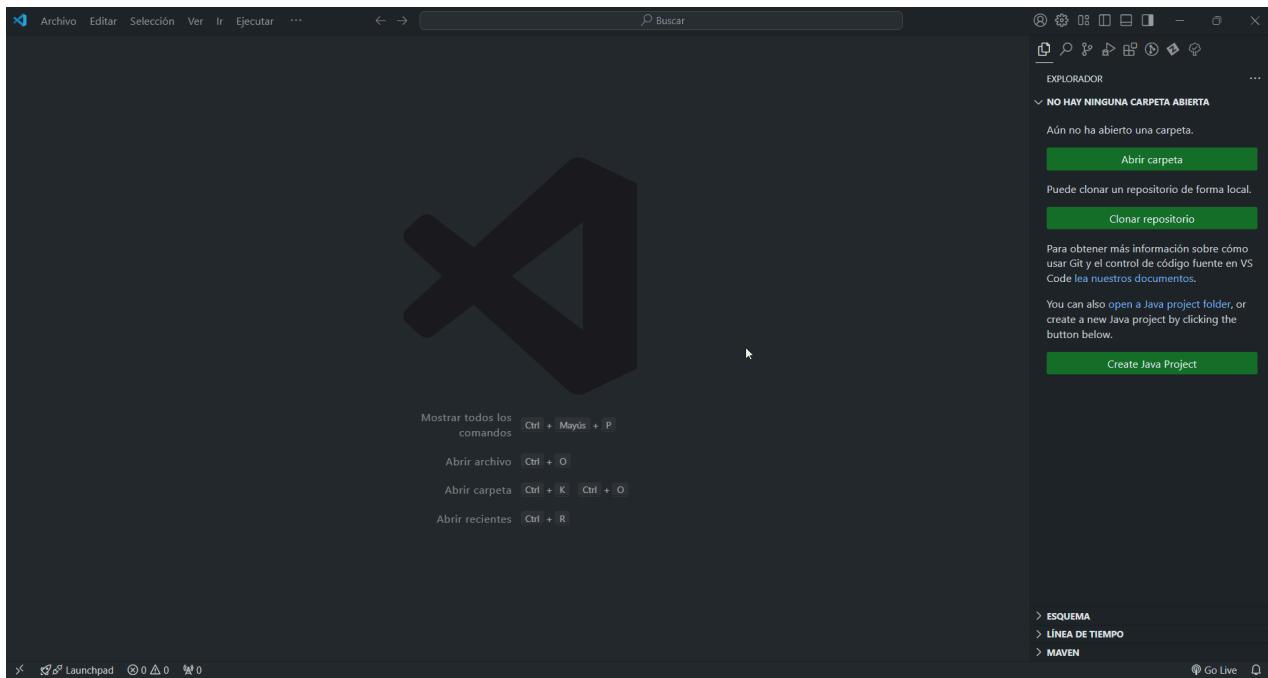
### Creando el directorio de los proyectos

Vamos a crear una carpeta llamada WebApp Projects, no es obligatorio usar el mismo nombre, para garantizar el uso correcto, la creamos dentro del disco local C, pero puedes crear el directorio en el lugar donde puedas ubicarlo siempre:



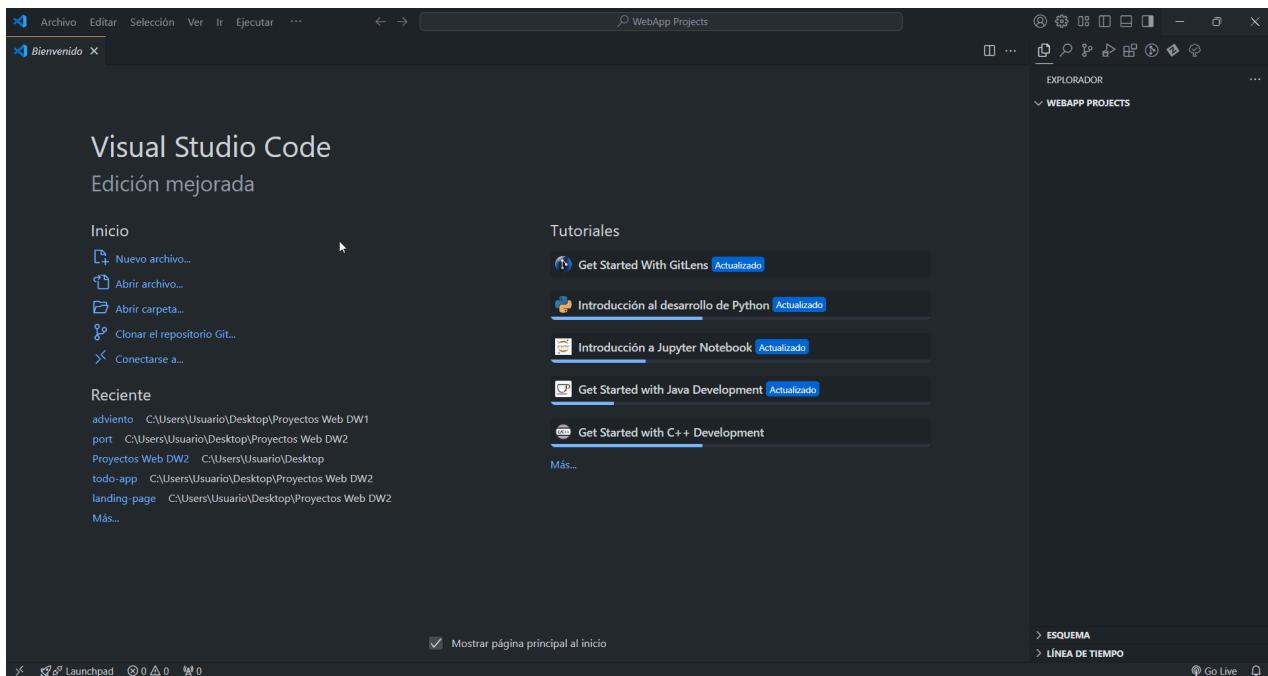
### Abriendo el directorio en VSCode

Abrimos la carpeta creada en Visual Studio Code:



## Instalando Flask

Procedemos a instalar el framework Flask:



## 6. "Hola Mundo" en Flask

---

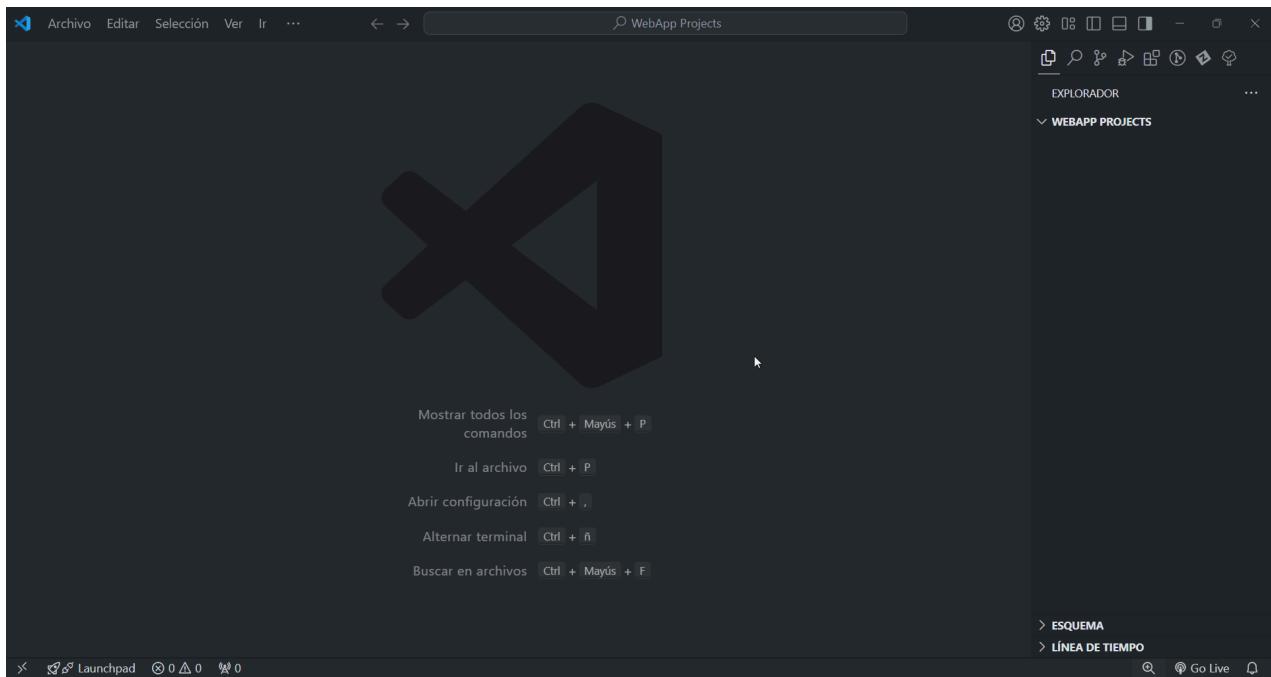


### Contenido

---

#### Creando primer fichero

Vamos a crear nuestro primer fichero llamado holamundo.py (siempre debe llevar la extensión .py para identificar que es código escrito en Python):



#### Creando el código para el "Hola Mundo"

Escribiremos lo siguiente:

```
from flask import Flask, render_template_string

app = Flask(__name__)

@app.route('/')
def hola_mundo():
    contenidoHtml = "<h1>Hola Mundo</h1>"
    return render_template_string(contenidoHtml)

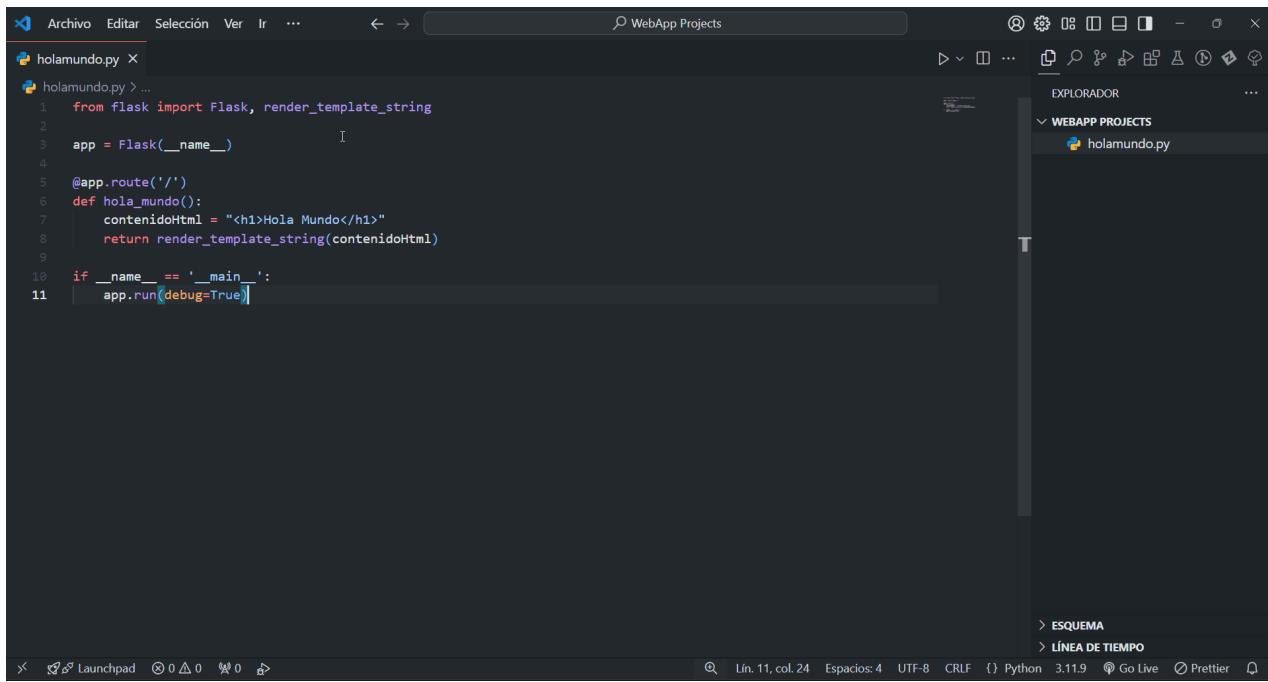
if __name__ == '__main__':
    app.run(debug=True)
```

Explicando:

- Flask: Es el núcleo del framework que permite crear aplicaciones web.
- render\_template\_string: Una función que permite renderizar contenido HTML directamente desde una cadena de texto en lugar de un archivo HTML.
- Flask(\_\_name\_\_): Aquí se instancia la aplicación Flask. El argumento \_\_name\_\_ indica el nombre del módulo principal, lo cual ayuda a Flask a localizar recursos y plantillas.
- @app.route('/'): Define una ruta en la aplicación. En este caso, la ruta raíz / es la dirección principal del sitio web (por ejemplo, http://127.0.0.1:5000/).
- La función hola\_mundo se ejecutará cuando alguien visite la ruta /.
- contenidoHtml: Es una variable que contiene el código HTML en forma de cadena. Aquí, el HTML solo tiene un encabezado <h1> con el texto "Hola Mundo".
- render\_template\_string: Esta función envuelve el contenido HTML y lo envía como respuesta al navegador del usuario.
- if \_\_name\_\_ == '\_\_main\_\_': Asegura que la aplicación solo se ejecutará si el archivo es ejecutado directamente, no si es importado como módulo en otro archivo.
- app.run(debug=True): Inicia el servidor web de Flask en modo de depuración (debug=True), lo que permite detectar errores fácilmente y recargar el servidor automáticamente si se realizan cambios en el código.

## Ejecutando el archivo

Abrimos la terminal de Visual Studio Code y digitamos python holamundo.py:



```
holamundo.py > ...
1  from flask import Flask, render_template_string
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def hola_mundo():
7      contenidoHtml = "<h1>Hola Mundo</h1>"
8      return render_template_string(contenidoHtml)
9
10 if __name__ == '__main__':
11     app.run(debug=True)
```

## Usando esquema básico de HTML

Ahora vamos a cambiar el contenido

```
from flask import Flask, render_template_string

app = Flask(__name__)

@app.route('/')
def hola_mundo():
    contenidoHtml = """
        <!DOCTYPE html>
        <html lang="en">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <title>Página con Flask</title>
        </head>
        <body>
            <h1>Hola Mundo desde Flask 😊</h1>
        </body>
    </html>
"""

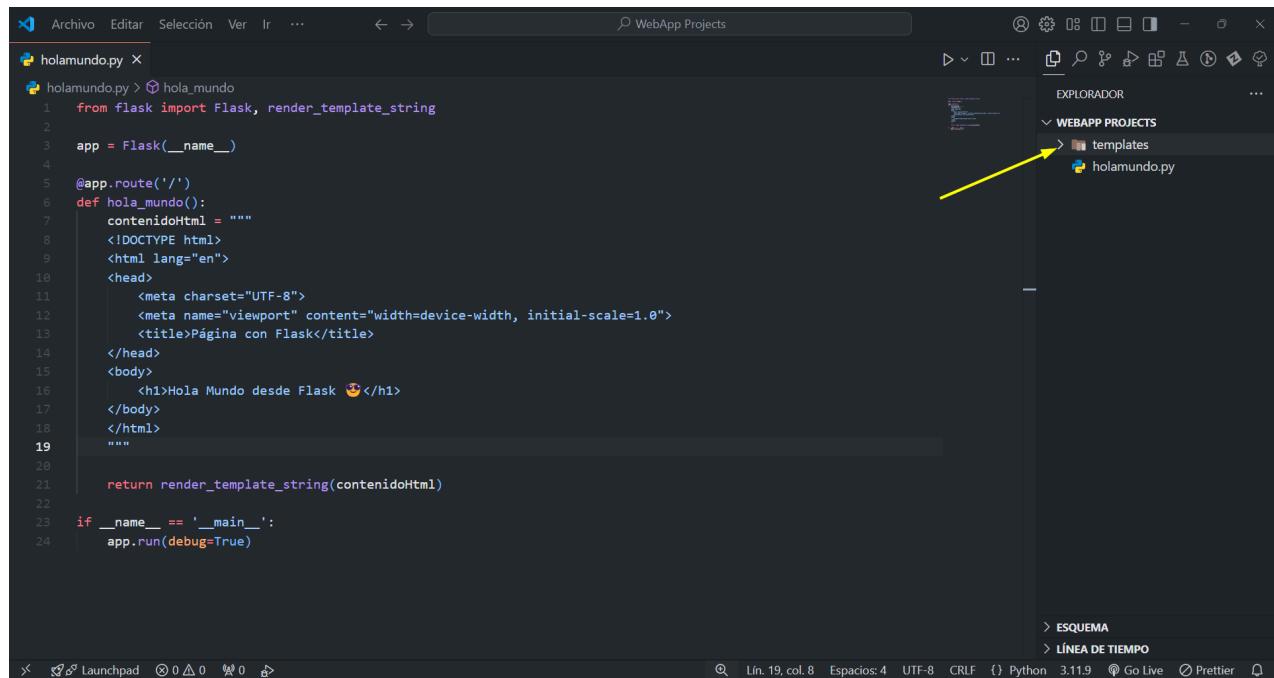
    return render_template_string(contenidoHtml)
```

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

Volvemos a ejecutar el comando `python holamundo.py` para correr el servidor y revisamos los cambios.

## Creando la carpeta templates

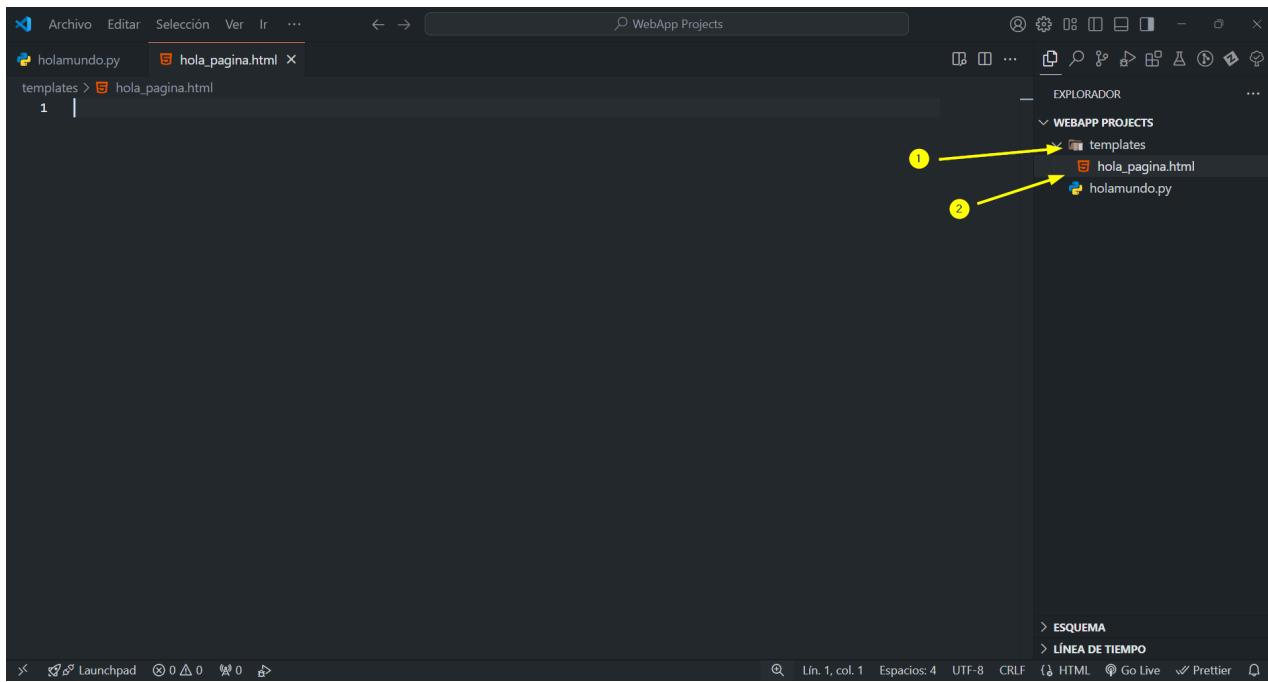
Creamos la carpeta `templates` dentro del proyecto:



```
holamundo.py
holamundo.py > hola_mundo
1   from flask import Flask, render_template_string
2
3   app = Flask(__name__)
4
5   @app.route('/')
6   def hola_mundo():
7       contenidoHtml = """
8           <!DOCTYPE html>
9           <html lang="en">
10              <head>
11                  <meta charset="UTF-8">
12                  <meta name="viewport" content="width=device-width, initial-scale=1.0">
13                  <title>Página con Flask</title>
14              </head>
15              <body>
16                  <h1>Hola Mundo desde Flask 🎉</h1>
17              </body>
18          </html>
19      """
20
21      return render_template_string(contenidoHtml)
22
23 if __name__ == '__main__':
24     app.run(debug=True)
```

## Creando primer archivo dentro de templates

Creamos un archivo llamado `hola_pagina.html`:



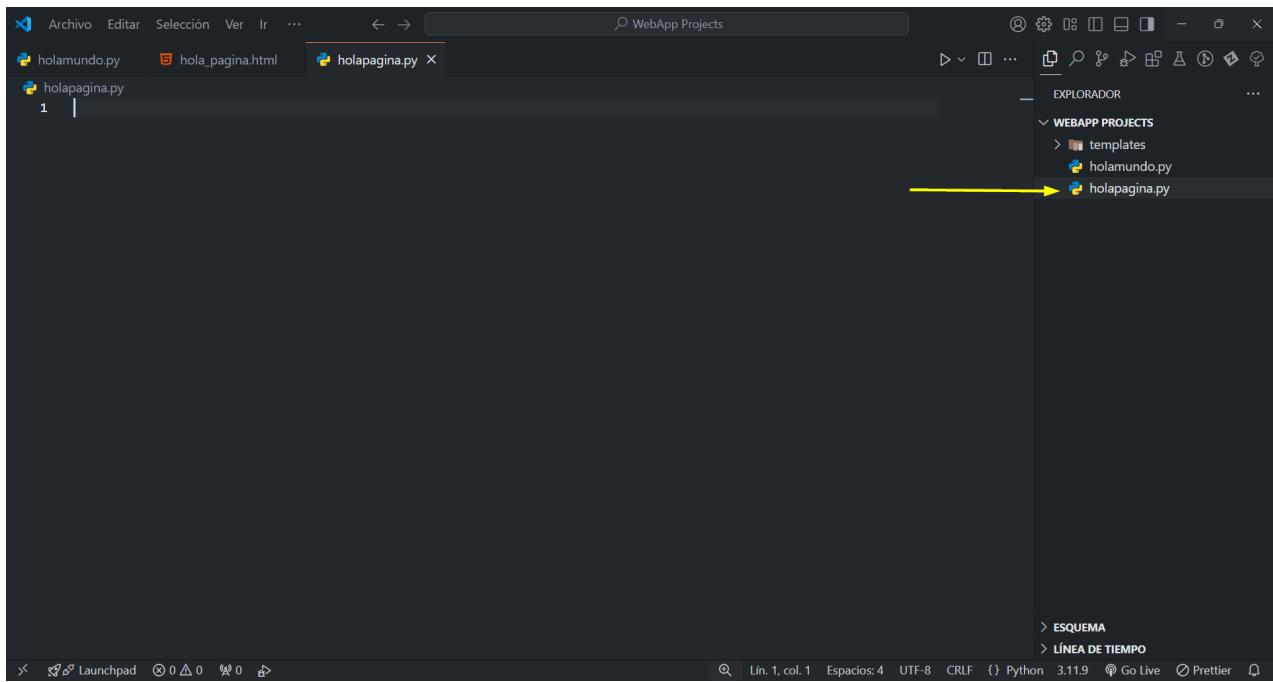
## Contenido en página de HTML

Agregamos el contenido creado en holamundo.py a hola\_pagina.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Página con Flask</title>
</head>
<body>
<h1>Hola Mundo desde Flask 🎉</h1>
</body>
</html>
```

## Creamos nuevo archivo de Flask

Creamos un nuevo archivo de Flask llamado holapagina.py:



Agregamos el siguiente código:

```
from flask import Flask, render_template

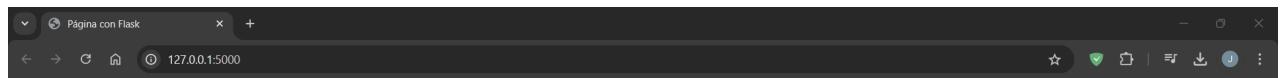
app = Flask(__name__)

@app.route('/')
def hola_mundo():
    return render_template('hola_pagina.html')

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

## Ejecutando servicio

Al ejecutar python holapagina.py, veremos lo siguiente:



---

Hola Mundo desde Flask 😊

---

Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0  
<<http://creativecommons.org/licenses/by-sa/4.0/>>