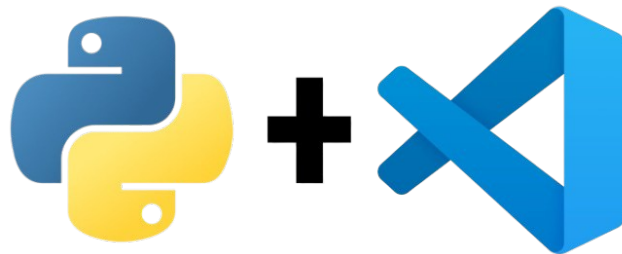


# CRIPTOGRAFÍA CON PYTHON



Preparar entorno para  
trabajar con Python

Esta obra está sujeta a la licencia Reconocimiento-  
CompartirIgual 4.0 Internacional de Creative Commons.

Para ver una copia de esta licencia, visitad  
<https://creativecommons.org/licenses/by-sa/4.0/>.





Autor: Enrique Melchor Iborra Sanjaime (em.iborrasanjaime@edu.gva.es)

## Contenido

1. Verificar si Python ya está instalado.....	3
2. Instalar Python (si es necesario).....	3
En distribuciones basadas en Debian/Ubuntu:.....	3
En distribuciones basadas en Fedora:.....	3
En distribuciones basadas en Arch Linux:.....	3
En Windows:.....	3
Comprueba la instalación con : \$ python3 --version.....	3
Entra en el interprete: \$ python3.....	3
Hola mundo en python.....	3
Explicación.....	3
Otra forma de hacer que python ejecute ordenes. Desde un fichero con extensión .py.....	3
3. Instalar pip (gestor de paquetes de Python).....	4
4. Configurar un entorno virtual.....	4
Crear un entorno virtual.....	4
Activar el entorno virtual.....	4
Desactivar el entorno virtual.....	4
5. Instalar Paquetes.....	5
6. Instalar un editor de texto o IDE.....	5
7. Configuración de linters y formateadores.....	6
8. Instalar herramientas de depuración.....	7
9. Control de versiones con Git.....	7
10. Configuración de entornos seguros.....	7
python-dotenv.....	7
11. Introducción a las bibliotecas de criptografía en Python.....	8
◦ cryptography.....	8
◦ PyCrypto => PyCryptodome.....	9
◦ hashlib.....	10
◦ ecdsa.....	10

Configurar un entorno de desarrollo para Python en Linux es un proceso sencillo, pero tiene varios pasos que puedes seguir para asegurarte de que tu entorno esté optimizado y listo para el desarrollo.

A lo largo del curso, verás de vez en cuando unos símbolos para indicar que te pares y que practiques.

 Instala o configura	\$     prompt del S.O. esperando orden C:\>
 Prueba o programa código	>>>   prompt de python esperando orden ...

## 1. Verificar si Python ya está instalado

La mayoría de las distribuciones de Linux ya vienen con Python instalado por defecto. Para comprobar si ya tienes Python, abre una terminal y escribe:

```
$ python3 --version      ( en windows      C:\> python --version )
```

Esto debería devolver la versión de Python que tienes instalada. Si no lo tienes, sigue los siguientes pasos para instalarlo.

## 2. Instalar Python (si es necesario)

Si no tienes Python o necesitas una versión más reciente, puedes instalarlo. En la mayoría de las distribuciones de Linux, puedes usar el gestor de paquetes para instalar Python.

### En distribuciones basadas en Debian/Ubuntu:

```
$ sudo apt update
$ sudo apt install python3
```



### En distribuciones basadas en Fedora:

```
$ sudo dnf install python3
```

### En distribuciones basadas en Arch Linux:

```
$ sudo pacman -S python
```

### En Windows:

Con el instalador de la pagina oficial <https://www.python.org/downloads/windows/>

**Comprueba la instalación con :** `$ python3 --version`

**Entra en el interprete:** `$ python3`

Deberia salir un prompt `>>>` esperando una orden

### Hola mundo en python

El programa más básico en Python es el clásico "Hola Mundo".

Practica →

```
>>> print("Hola, Mundo!")
```



### Explicación

- `print()` es una función en Python que imprime el texto dentro de los paréntesis en la consola.

### Otra forma de hacer que python ejecute ordenes. Desde un fichero con extensión .py

Para ejecutar un programa:

1. Guarda el código en un archivo con extensión `.py`, por ejemplo, `hola_mundo.py`.
2. Abre la terminal y navega hasta la carpeta donde guardaste el archivo.
3. Ejecuta el programa con el siguiente comando:

```
$ python3 hola_mundo.py
```

Esto debería mostrar:

```
Hola, Mundo!
```

¡Es un excelente punto de partida para comenzar a explorar Python!

### 3. Instalar pip (gestor de paquetes de Python)

pip es el gestor de paquetes oficial para Python y se usa para instalar y gestionar bibliotecas y dependencias externas.

Comprobar si esta instalado con `pip3 --version`

Esto debería devolver la versión de pip que tienes instalada. Si no lo tienes, sigue los siguientes pasos para instalarlo

Para instalar pip para Python 3, ejecuta:

```
$ sudo apt update           # En sistemas basados en Debian/Ubuntu
$ sudo apt install python3-pip # En sistemas basados en Debian/Ubuntu
$ sudo dnf install python3-pip # En Fedora
$ sudo pacman -S python-pip  # En Arch Linux
```



--Antes de empezar a utilizar pip, para evitar problemas, se instalará el entorno virtual--

### 4. Configurar un entorno virtual

Para evitar problemas con dependencias entre proyectos y tener un entorno aislado para cada uno de tus proyectos, es recomendable usar entornos virtuales. Además, en Linux, si no configuras un entorno virtual, tendrás problemas con `pip`. (si estás en Windows, te puedes saltar este paso)

#### Crear un entorno virtual

1. **Instalar el módulo venv (si no lo tienes):** En muchas distribuciones, el módulo `venv` ya está instalado por defecto. Si no lo tienes, instala el paquete correspondiente.

En Ubuntu/Debian:

```
$ sudo apt install python3-venv
```



2. **Crear un entorno virtual:** Navega hasta la carpeta de tu proyecto y crea un entorno virtual usando el siguiente comando:

```
$ python3 -m venv nombre_del_entorno   C:\> python -m venv nombre_del_entorno
```

Esto creará una carpeta llamada `nombre_del_entorno` que contendrá el entorno virtual.

#### Activar el entorno virtual

Para activar el entorno virtual, usa el siguiente comando (deberás hacer esto cada vez que trabajes en tu proyecto):

```
$ source nombre_del_entorno/bin/activate
C:\> nombre_del_entorno\Scripts\activate (cmd)
PS C:\> Set-ExecutionPolicy Unrestricted -Scope Process (Powershell)
PS C:\> nombre_del_entorno\Scripts\activate (Powershell)
```

Verás que el nombre del entorno aparecerá entre paréntesis en tu terminal, indicando que estás trabajando dentro del entorno virtual.

```
servora@servora-VirtualBox:~$ source Enrique/bin/activate
(enrique) servora@servora-VirtualBox:~$
```

#### Desactivar el entorno virtual

Cuando termines de trabajar, puedes desactivar el entorno virtual con el siguiente comando:

```
$ deactivate
```

## 5. Instalar Paquetes

Una vez que tengas el entorno virtual activado, puedes instalar los paquetes necesarios para tu proyecto utilizando `pip`.

```
$ pip3 install nombre_paquete
```

```
ejemplo $ pip3 install pyfiglet      (https://pypi.org/project/pyfiglet/)
```

```
uso      $ pyfiglet hola python
```

```
$ man pyfiglet
```



uso desde python

```
>>> from pyfiglet import Figlet
>>> f = Figlet(font='slant')
>>> print(f.renderText('hola python'))
```

Si tienes un archivo `requirements.txt` con todas las dependencias de tu proyecto, puedes instalar todas las dependencias de una sola vez con:

```
$ pip3 install -r requirements.txt
```

El archivo `requirements.txt` en Python se usa para gestionar las dependencias de un proyecto. Contiene una lista de paquetes y sus versiones necesarias para que el código funcione correctamente.

Para generar un archivo `requirements.txt`, puedes usar el siguiente comando dentro de tu entorno virtual:

```
pip freeze > requirements.txt
```

Este comando guarda todas las dependencias instaladas en el entorno virtual dentro del archivo.

Es recomendable usar `requirements.txt` junto con un entorno virtual (`venv` o `virtualenv`) para evitar conflictos de dependencias entre proyectos.

## 6. Instalar un editor de texto o IDE

Para escribir código en Python, necesitarás un editor de texto o un IDE (entorno de desarrollo integrado). Algunas de las opciones más populares son:

- **Visual Studio Code (VS Code):** Un editor muy popular con soporte para Python. Instálalo desde la terminal:

```
$ sudo apt install code # En distribuciones basadas en Debian/Ubuntu
$ sudo apt install code-oss # en kali
sudo dnf install code # En Fedora
sudo pacman -S code # En Arch Linux
```



- En kubuntu no esta en repositorio. Descargamos el `.deb` de la web e instalamos con la orden `$ sudo dpkg -i nombre_del_archivo.deb`

- 0 con snap    \$ sudo snap install code --classic

\*\*\* Una vez el Visual Studio Code descargado e instalado, ejecuta el editor y agrega la extensión de Python para una mejor experiencia de desarrollo.

- **PyCharm:** Un IDE dedicado a Python. Puedes instalar la versión Community de PyCharm desde [su sitio oficial](#).
- **Sublime Text:** Otro editor de texto popular. Instálalo desde su sitio web [Sublime Text](#).
- **Vim o Emacs:** Si prefieres usar editores de texto basados en terminal, ambos son muy potentes para programar en Python.

## 7. Configuración de linters y formateadores

Para tener un código limpio y ordenado, puedes usar herramientas como **flake8** para linters y **black** para formateo automático.

- **Instalar ruff**

```
$ pip3 install ruff
```

```
uv add --dev ruff
```

```
ruff check
```

```
ruff check --fix
```

```
ruff format
```

- **Instalar flake8:** ( **autoflake pyflakes**)

```
$ pip3 install flake8
```

- **Instalar pylint :**

```
$ pip3 install pylint
```

- **Instalar black (formateador de código):**

```
$ pip3 install black
```

- Para usar **flake8**, puedes ejecutar:

```
$ flake8 nombre_del_archivo.py
```

- Para usar **black**, puedes ejecutar:

```
$ black nombre_del_archivo.py
```

## 8. Instalar herramientas de depuración

Para depurar tu código Python, puedes usar herramientas como **pdb**, que es el depurador integrado en Python. Solo necesitas incluir la línea: `import pdb; pdb.set_trace()` en tu código para comenzar a depurar.

## 9. Control de versiones con Git

Es recomendable usar Git para el control de versiones de tu proyecto. Si no tienes Git instalado, puedes hacerlo con:

```
$ sudo apt install git # En Ubuntu/Debian
sudo dnf install git # En Fedora
sudo pacman -S git # En Arch
```



Inicia un repositorio Git en tu proyecto con:

```
$ git init
```

Y haz tus primeros commits con:

```
$ git add .
$ git commit -m "Primer commit"
```

## 10. Configuración de entornos seguros

Es importante configurar entornos de desarrollo seguros para trabajar con bibliotecas de criptografía, ya que manejar claves y datos sensibles requiere precauciones adicionales.

1. **Uso de herramientas de gestión de secretos:** Es importante manejar claves y otros datos sensibles de manera segura. Puedes utilizar herramientas como **Python-dotenv** para cargar las variables de entorno desde archivos `.env`, o utilizar servicios externos como **AWS Secrets Manager** o **Azure Key Vault**.
2. **Instalación de herramientas adicionales:** Si vas a trabajar en un entorno seguro o en producción, asegúrate de utilizar herramientas de análisis de seguridad para verificar que tu código es seguro y que no hay vulnerabilidades conocidas.

### python-dotenv

#### ¿Qué es?

Es una librería de terceros que permite cargar variables de entorno desde un archivo `.env` a tu aplicación en Python.

#### ¿Para qué sirve?

- Manejar de forma segura las **credenciales sensibles**, como claves API, contraseñas, o configuraciones de base de datos.
- Evitar incluir información sensible directamente en el código fuente.
- Facilitar la configuración de entornos diferentes (desarrollo, pruebas, producción) sin modificar el código.

## ¿Cuándo usarlo?

- Cuando necesitas trabajar con configuraciones que varían según el entorno, como claves de API o nombres de servidores.
- Para mantener las credenciales fuera del repositorio de código.

## Ejemplo de uso:

### 1. Instalación

```
pip3 install python-dotenv
```

### 2. Archivo .env:

```
API_KEY=abc123
DATABASE_URL=mysql://user:password@localhost/db
DEBUG=True
```

### 3. Código en Python: En tu código, puedes acceder a estas variables de la siguiente manera:

```
from dotenv import load_dotenv
import os

# Cargar variables del archivo .env
load_dotenv()

# Acceder a las variables
api_key = os.getenv('API_KEY')
db_url = os.getenv('DATABASE_URL')

print(f"API Key: {api_key}")
print(f"Database URL: {db_url}")
```

## 11. Introducción a las bibliotecas de criptografía en Python

### ◦ cryptography

La biblioteca **cryptography** es una de las más populares y completas en Python para la implementación de criptografía moderna. Ofrece una amplia variedad de algoritmos criptográficos tanto para la protección de datos como para la autenticación y firma digital.

Algunas de las características principales de la biblioteca son:

- **Cifrado simétrico:** AES, ChaCha20, etc.
- **Cifrado asimétrico:** RSA, DSA, ECDSA.
- **Firmas digitales:** Usando RSA y ECDSA.
- **Funciones de hash:** SHA-256, SHA-512, y más.
- **Gestión de claves:** Generación de claves, almacenamiento seguro y manejo de certificados X.509.
- **Protocolos seguros:** Soporte para crear y verificar certificados X.509.

Instalación:

```
pip install cryptography
```





API de cryptography

<https://cryptography.io/en/latest/>

API de modulo secrets de python

<https://docs.python.org/es/3.13/library/secrets.html>

Ejemplo de uso básico:

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
import os

key = os.urandom(32) # Generar una clave aleatoria de 32 bytes
iv = os.urandom(16) # Generar un vector de inicialización de 16 bytes

cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
encryptor = cipher.encryptor()
ciphertext = encryptor.update(b"mensaje a cifrar") + encryptor.finalize()
print(ciphertext)
```

### ◦ PyCrypto => PyCryptodome

**PyCrypto** es una antigua biblioteca para criptografía en Python que ha sido descontinuada. Sin embargo, su sucesora, **PyCryptodome**, se sigue manteniendo y se considera una excelente alternativa para trabajar con criptografía en Python.

PyCryptodome es una biblioteca más completa que incluye:

- **Cifrado simétrico:** AES, DES, Triple DES, Blowfish, etc.
- **Cifrado asimétrico:** RSA, ElGamal, etc.
- **Generación de claves y firmas digitales.**
- **Funciones de hash:** SHA-1, SHA-256, MD5, etc.
- **Autenticación:** HMAC, que combina un algoritmo de hash con una clave secreta.

Instalación:

`pip install pycryptodome`

API



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

Ejemplo básico

```
from Crypto.Random import get_random_bytes
# Generar un número aleatorio de 4 bytes
random_bytes = get_random_bytes(4)
# Convertir los bytes a un entero si es necesario
random_number = int.from_bytes(random_bytes, byteorder="big")
print(f"Número aleatorio generado (entero): {random_number}")
random_in_range = random_number % 1001
print(f"Número aleatorio entre 0 y 1000: {random_in_range}")
```

### ◦ **hashlib**

La biblioteca **hashlib** es parte de la librería estándar de Python y se utiliza para generar funciones de hash criptográficas. Las funciones de hash son fundamentales en criptografía, especialmente para la verificación de integridad de datos y la autenticación.

Las funciones más comunes que soporta son:

- **SHA-256** y otras variantes de SHA.
- **MD5**, aunque no es recomendado para nuevas aplicaciones por ser vulnerable.
- **Blake2**, considerado seguro y eficiente.

No hace falta instalación. Python la lleva incorporada

API

<https://docs.python.org/es/3/library/hashlib.html>

Ejemplo de uso:

```
import hashlib

# Crear un objeto hash SHA-256
hash_obj = hashlib.sha256()
hash_obj.update(b"mensaje a hashear")
print("Hash:", hash_obj.hexdigest())
```

### ◦ **ecdsa**

Este paquete es una implementación en Python de la criptografía de curva elíptica, que incluye algoritmos como ECDSA (Elliptic Curve Digital Signature Algorithm), EdDSA (Edwards-curve Digital Signature Algorithm) y ECDH (Elliptic Curve Diffie-Hellman). La documentación proporciona información detallada sobre la generación de claves, firma, verificación y derivación de secretos compartidos

Instalación. `pip3 install ecdsa`

API : <https://ecdsa.readthedocs.io/en/latest/>

Ejemplo de uso:

```
from ecdsa.keys import SigningKey
key = SigningKey.generate()
```

Si estás interesado en una alternativa más ligera y rápida, puedes considerar el paquete starkbank-ecdsa, cuya documentación y código fuente están disponibles en GitHub.

<https://github.com/starkbank/ecdsa-python>

Para comenzar a trabajar con criptografía en Python, es necesario instalar las bibliotecas que se mencionaron previamente. Dependiendo de la biblioteca que elijas, la instalación será diferente.

#### 1. **cryptography**:

```
pip3 install cryptography
```

#### 2. **PyCryptodome** (sucesor de PyCrypto):

```
pip3 install pycryptodome
```

#### 3. **hashlib**: Esta biblioteca ya viene incluida con Python, por lo que no es necesario instalarla.

## Otras bibliotecas

```
pip3 install M2Crypto          pip3 install pynacl
pip3 install simplecrypt      pip3 install pysodium
pip3 install pyopenssl        pip3 install pycrypt
pip3 install ecdsa            pip3 install bcrypt
```

## Resumen de bibliotecas

- **PyCryptodome** y **cryptography** son opciones completas para la mayoría de los proyectos que requieren cifrado y autenticación.
- **PyNaCl** y **ecdsa** son excelentes si necesitas criptografía moderna con **clave pública**.
- **bcrypt** es ideal para almacenar contraseñas de forma segura.
- **PyOpenSSL** y **M2Crypto** son útiles cuando necesitas trabajar con **SSL/TLS** o **certificados**.
- **SimpleCrypt** y **Sodium** son buenas opciones si buscas implementaciones más simples pero seguras de cifrado simétrico.

## Resumen de los pasos:

1. Verifica si Python ya está instalado: `python3 --version`.
2. Instala Python si es necesario: `sudo apt install python3`.
3. Instala pip: `sudo apt install python3-pip`.
4. Crea y activa un entorno virtual con `python3 -m venv nombre_del_entorno`.
5. Instala dependencias con `pip (pip3)`.
6. Instala un editor de texto o IDE (VS Code, PyCharm, Sublime, etc.).
7. Configura linters y formateadores (flake8, black).
8. Usa Git para control de versiones.
9. Instala la librería `python-dotenv` para evitar el hardcoding
10. Instala la librerías orientadas a la criptografía