UD01.02: Entornos de desarrollo

Contenido

1. Instalación de Python 2
   1.1. Obtención y disponibilidad del intérprete de Python 2
   1.2. Verificación de la instalación 2
   1.3. Configuración de la variable PATH (en Windows) 3
   1.4. ¿Qué es el intérprete de Python? 3
2. ¿Qué es un entorno de desarrollo? 5
   2.1. Funcionalidades clave de un IDE 5
   2.2. Criterios para elegir la herramienta correcta 6
   2.3. Ejemplos de herramientas populares para Python 6
3. El problema de las dependencias 7
   3.1. ¿Qué es un entorno virtual y para qué sirve? 7
   3.2. Herramientas para la gestión de entornos: venv y pip 7
   3.3. Herramientas avanzadas: conda y poetry 8
   3.4. Primer contacto con librerías populares en Python 9

1. Instalación de Python

Este es el paso fundamental para poder empezar a programar. Aunque Python puede venir preinstalado en algunos sistemas operativos (como en macOS o Linux), es crucial asegurarse de que tenemos la versión correcta y de que está correctamente configurada.

1.1. Obtención y disponibilidad del intérprete de Python

- **¿De dónde se descarga?** La fuente oficial y más segura para descargar el intérprete de Python es el sitio web oficial: python.org.
- **Versiones:** En 2025, la versión principal activa es Python 3. Siempre debemos utilizar la última versión estable de la rama 3.x para aprovechar las nuevas funcionalidades y mejoras de seguridad.
- **Sistemas Operativos:** Existen instaladores para los principales sistemas operativos:
    - **Windows:** Se descarga un ejecutable (.exe) que guía el proceso. Es crucial marcar la opción de "Add Python to PATH" durante la instalación.
    - **macOS:** Se descarga un instalador (.pkg).
    - **Linux:** Python suele venir preinstalado, pero la versión puede ser antigua. Se recomienda usar el gestor de paquetes del sistema (apt en Debian/Ubuntu, yum en RHEL, etc.) para instalar una versión más reciente.

1.2. Verificación de la instalación

Una vez instalado, es importante verificar que el intérprete es accesible desde la terminal.

- **Abre la terminal** (Símbolo del sistema en Windows, Terminal en macOS/Linux).
- **Para Windows ejecuta el comando:**
    - python –version
    - python -V
- **Resultado:** Si la instalación fue correcta, deberías ver la versión de Python que has instalado (por ejemplo: Python 3.12.2). Si no aparece, es probable que la variable de entorno PATH no se haya configurado bien.

1.3. Configuración de la variable PATH (en Windows)

La variable PATH es una lista de directorios que el sistema operativo busca para encontrar archivos ejecutables. Si Python no está en el PATH, no podrás ejecutarlo desde cualquier directorio de la terminal.

- **Problema común:** Al ejecutar python --version, el sistema muestra un error como "python" no se reconoce como un comando interno o externo....
- **Solución:** Aunque lo ideal es marcar la opción "Add Python to PATH" durante la instalación, si se te olvida, puedes añadir la ruta manualmente. Esto se hace en la configuración avanzada del sistema de Windows, añadiendo las rutas de la carpeta de Python y la de Scripts a la variable de entorno PATH.
- En caso de necesidad ver manual al final del capítulo.

1.4. ¿Qué es el intérprete de Python?

El corazón del lenguaje El intérprete de Python es el programa que lee y ejecuta tu código. A diferencia de lenguajes como C++ o Java, que requieren un paso previo de "compilación" para traducir todo el código a un lenguaje máquina, Python es un lenguaje *interpretado*. Esto significa que el intérprete lee tu código línea por línea y lo ejecuta de forma directa y en tiempo real.

Funcionamiento básico Cuando ejecutas un script de Python (python mi_script.py), en realidad estás invocando al intérprete de Python. El intérprete realiza las siguientes acciones:

- **Lectura:** Lee tu archivo de código (mi_script.py).
- **Análisis:** Analiza la sintaxis de cada línea.
- **Ejecución:** Ejecuta las instrucciones una por una.

Esta forma de trabajar facilita el desarrollo rápido y la depuración, ya que puedes ver el resultado de cada línea de código de forma inmediata en la terminal.

Diferencia con un compilador

- **Intérprete (Python):** Lee y ejecuta el código fuente directamente, línea por línea.
- **Compilador (C++, Java):** Traduce todo el código fuente a un archivo ejecutable (código máquina) de una sola vez, antes de poder ser ejecutado.

Es importante diferenciar el intérprete de Python de las herramientas que lo utilizan, como IDLE, Thonny, VS Code o PyCharm. Estas herramientas son solo interfaces que facilitan la escritura, la gestión y la ejecución del código, pero todas ellas dependen del intérprete de Python que has instalado en tu sistema para funcionar.

2. ¿Qué es un entorno de desarrollo?

Un **entorno de desarrollo** es el conjunto de herramientas que un programador usa para escribir, ejecutar y depurar código. Aunque lo más básico es el intérprete de Python y un editor de texto, a medida que los proyectos crecen, las herramientas se vuelven más complejas. La principal diferencia radica en si la herramienta es un **editor de código** o un **IDE**.

Característica	Editor de código	IDE (Entorno de Desarrollo Integrado)
Definición	Una aplicación para escribir código, con funciones básicas.	Un conjunto completo de herramientas para todas las fases del desarrollo.
Funcionalidades	Resaltado de sintaxis, numeración de líneas.	**Todo lo anterior más:** depurador, terminal, gestor de proyectos, control de versiones.
Ventajas	Ligero, rápido, flexible y minimalista.	Aumenta la productividad, reduce errores y facilita el trabajo en equipo.
Uso ideal	Proyectos pequeños, scripts sencillos y edición rápida.	Proyectos complejos, desarrollo profesional y colaboración.

2.1. Funcionalidades clave de un IDE

Un IDE va más allá de un editor simple. Sus herramientas integradas están diseñadas para ayudarte a ser más eficiente y a mantener la calidad de tu código.

- **Edición avanzada de código:** La mayoría de los IDEs ofrecen **resaltado de sintaxis** y un **autocompletado inteligente**. Esto no solo acelera la escritura, sino que también ayuda a detectar errores tipográficos en tiempo real.
- **Depurador visual:** Es una herramienta esencial para encontrar errores lógicos. El depurador te permite ejecutar el código paso a paso e **inspeccionar el valor de las variables** en cada momento, para que puedas ver exactamente por qué tu programa no se comporta como esperas.
- **Gestión de proyectos y control de versiones:** Los IDEs organizan los archivos de un proyecto en una estructura unificada, lo que facilita la navegación. Además, su integración nativa con herramientas como **Git** te permite gestionar cambios y colaborar con otros desarrolladores sin salir del programa.
- **Herramientas integradas:** Para evitar que tengas que cambiar constantemente de ventana, los IDEs incorporan una **terminal**, un gestor de **entornos virtuales** y otras utilidades que agilizan el flujo de trabajo.

2.2. Criterios para elegir la herramienta correcta

La elección de la herramienta depende de tus necesidades específicas. No hay una respuesta única, pero sí hay una herramienta ideal para cada situación.

- **Tu nivel de experiencia:** Si eres principiante, la simplicidad es clave. Un entorno con una interfaz limpia te ayudará a centrarte en aprender Python, no en la herramienta.
- **La complejidad del proyecto:** Para un *script* sencillo que procesa datos una vez, un editor ligero es suficiente. Para un proyecto con varias carpetas, librerías y un equipo de desarrollo, un IDE es indispensable.
- **Preferencias personales:** Algunos programadores prefieren la potencia de un IDE profesional, mientras que otros valoran la flexibilidad y la ligereza de un editor de código.

2.3. Ejemplos de herramientas populares para Python

- **Para principiantes:**
    - **IDLE:** Es el entorno básico que viene con la instalación de Python. Sencillo y perfecto para aprender a ejecutar scripts y usar el *shell* interactivo.
    - **Thonny:** Un IDE diseñado específicamente para la enseñanza. Su depurador visual es muy intuitivo y ayuda a entender cómo fluye el código.
- **Para uso general y proyectos flexibles:**
    - **Visual Studio Code (VS Code):** Es un editor de código muy potente que, con las extensiones adecuadas, se convierte en un IDE completo. Es muy popular por su flexibilidad, velocidad y el gran ecosistema de *plugins* que tiene.
- **Para desarrollo profesional y proyectos complejos:**
    - **PyCharm:** Un IDE profesional y completo, creado específicamente para Python. Ofrece las herramientas de depuración y análisis más avanzadas, lo que lo hace ideal para proyectos grandes y de nivel empresarial.

3. El problema de las dependencias

¿Por qué necesitamos entornos virtuales?

Imaginad la siguiente situación: estáis trabajando en dos proyectos de Python a la vez.

- El **Proyecto A** es una aplicación web que requiere una librería llamada requests en su versión 2.28.1.
- El **Proyecto B** es un script de automatización que solo funciona con una versión más antigua de esa misma librería, la 2.20.0.

Si instaláis las librerías directamente en vuestro sistema, de forma global, se producirá un conflicto. Un proyecto puede dejar de funcionar cuando instaléis la versión que necesita el otro.

El objetivo de los entornos virtuales es resolver este problema. Son la solución a este tipo de conflictos y nos permiten trabajar de forma limpia, organizada y profesional.

3.1. ¿Qué es un entorno virtual y para qué sirve?

Un entorno virtual es una **carpeta que contiene una instalación de Python y todas sus librerías de forma aislada**. Imagínalo como una "caja" para cada proyecto. Dentro de esa caja, puedes instalar las librerías que necesites, en las versiones que requieras, sin afectar a otros proyectos ni a la instalación principal de Python de tu sistema.

Ventajas clave:

- **Aislamiento:** Evita conflictos de versiones entre diferentes proyectos.
- **Control y reproducibilidad:** Permite documentar las librerías y sus versiones (requirements.txt), asegurando que cualquier otro desarrollador pueda replicar tu entorno exacto.
- **Buenas prácticas:** Es un estándar de la industria y una señal de profesionalidad.

3.2. Herramientas para la gestión de entornos: venv y pip

Python ya viene con las herramientas que necesitamos para gestionar entornos virtuales y librerías.

- **venv (módulo de Python):** Es el módulo oficial para crear entornos virtuales.
    - **Creación del entorno:** Para crear un nuevo entorno virtual, ejecutamos el siguiente comando en la terminal, dentro de la carpeta de nuestro proyecto.

python -m venv .venv

Esto creará una carpeta llamada .venv (nombre estándar) con la instalación aislada de Python.

- **Activación:** Antes de instalar cualquier librería, debemos "entrar" en el entorno virtual.
  - **Windows:** .venv\Scripts\activate
  - **Linux/macOS:** source .venv/bin/activate
- **Desactivación:** deactivate

- **pip (gestor de paquetes):** Es la herramienta principal para instalar, actualizar y eliminar librerías de Python.
  - **Instalación de librerías:** Con el entorno virtual activado, puedes instalar librerías con pip install [nombre_librería]. Por ejemplo: pip install requests.
  - **Registro de dependencias:** Para guardar las librerías que usas en el proyecto, utilizamos el siguiente comando. Esto crea el archivo requirements.txt, que se debe compartir con el proyecto.

pip freeze > requirements.txt

  - **Instalación desde un archivo:** Si quieres replicar un entorno, solo necesitas el archivo requirements.txt.

pip install -r requirements.txt

3.3. Herramientas avanzadas: conda y poetry

Aunque venv y pip son la combinación estándar, existen otras herramientas más especializadas:

- **conda:** Muy popular en ciencia de datos. Gestiona no solo paquetes de Python, sino también dependencias de otros lenguajes.
- **poetry:** Una herramienta moderna que simplifica la gestión de proyectos, dependencias y publicación de paquetes en un solo archivo de configuración.

3.4. Primer contacto con librerías populares en Python

Una vez que sepamos cómo gestionar un entorno, estaremos listos para usar las librerías que hacen de Python un lenguaje tan potente. Aquí tienes algunas de las más utilizadas, agrupadas por su campo de aplicación.

- **Desarrollo Web:**
    - **Flask:** Para crear un servidor web sencillo.
    - **Django:** Un *framework* completo para proyectos web robustos.
- **Análisis de Datos:**
    - **pandas:** Para manipular y analizar datos en tablas.
    - **NumPy:** Para operaciones matemáticas eficientes con grandes conjuntos de datos.
- **Automatización y *Scraping*:**
    - **requests:** Para hacer peticiones web y descargar contenido de internet.
    - **BeautifulSoup:** Para analizar y extraer datos de páginas web.

Configuración de la variable PATH en Windows

1. Comprobar si Python funciona

Abre el **Símbolo del sistema** o **PowerShell**. Escribe: python --version

Si aparece un error como: 'python' no se reconoce como un comando interno o externo... entonces Python **no está agregado al PATH** y debemos configurarlo manualmente.

2. Localizar la carpeta de instalación de Python

Busca la carpeta donde instalaste Python. Por defecto suele estar en:

C:\Users\TU_USUARIO\AppData\Local\Programs\Python\Python3XX

(donde **3XX** depende de la versión, por ejemplo Python312 para Python 3.12).

Dentro de esa carpeta, localiza también la subcarpeta **Scripts**, que suele estar aquí:

C:\Users\TU_USUARIO\AppData\Local\Programs\Python\Python3XX\Scripts

Importante: Necesitamos **ambas rutas**.

3. Abrir la configuración de variables de entorno

Haz **clic derecho** en **Este PC** o **Mi PC** en el Escritorio o en el Explorador. Selecciona **Propiedades**. En la nueva ventana, haz clic en **Configuración avanzada del sistema** (columna izquierda). En la ventana **Propiedades del sistema**, ve a la pestaña **Opciones avanzadas**. Pulsa el botón **Variables de entorno**.

4. Editar la variable PATH

En la sección **Variables del sistema**, busca la variable llamada **Path**. Selecciónala y haz clic en **Editar**. En la ventana que se abre:

Haz clic en **Nuevo**. Agrega la **primera ruta** (la carpeta principal de Python).

Haz clic en **Nuevo** de nuevo. Agrega la **ruta de la carpeta Scripts**.

Pulsa **Aceptar** en todas las ventanas para guardar los cambios.

5. Verificar que Python funciona

Cierra la terminal y ábrela de nuevo. Escribe: python --version

Si la configuración fue correcta, verás la versión de Python, por ejemplo: Python 3.12.2