ACTIVIDADES

Contenido

1. Instalación de Python

Objetivo

Aprender a instalar Python en distintos sistemas operativos y verificar que está correctamente configurado para su uso desde la terminal.

Paso 1: Descargar el instalador de Python

1. Abre tu navegador web y ve al sitio oficial de Python: python.org.
2. En el menú superior, haz clic en **Downloads**.
3. El sitio web detectará tu sistema operativo. Verás un botón grande que dice **Download Python X.Y.Z** (donde X.Y.Z es la versión más reciente). Haz clic en ese botón para descargar el instalador.

Paso 2: Ejecutar el instalador

- **Para Windows:**
    1. Abre el archivo .exe que acabas de descargar.
    2. Verás una ventana de instalación. **IMPORTANTE:** Marca la casilla Add python.exe to PATH antes de hacer clic en Install Now. Esto es crucial para que puedas usar Python desde la terminal.
    3. Acepta los permisos de administrador y espera a que la instalación se complete.
- **Para macOS:**
    1. Abre el archivo .pkg descargado.
    2. Sigue las instrucciones del instalador. La configuración por defecto es suficiente para este curso.
    3. Al finalizar, es probable que se abra una ventana con el IDLE de Python, lo cual confirma que la instalación ha sido exitosa.
- **Para Linux:**
    1. En la mayoría de las distribuciones, Python ya está preinstalado. Sin embargo, para asegurarte de que tienes la versión más reciente, usa el gestor de paquetes de tu sistema.

2. Abre la terminal y ejecuta uno de los siguientes comandos, según tu distribución:
    - **En Ubuntu/Debian:** sudo apt-get install python3
    - **En Fedora/CentOS:** sudo dnf install python3

Paso 3: Verificar la instalación

Ahora, vamos a confirmar que Python está accesible desde la terminal.

1. Abre la terminal o el Símbolo del Sistema:
    - **Windows:** Presiona Win + R, escribe cmd y presiona **Enter**.
    - **macOS:** Busca Terminal en el Launchpad.
    - **Linux:** Abre la terminal desde el menú de aplicaciones.
2. En la ventana de la terminal, ejecuta el siguiente comando:

python --version

(En algunos sistemas, si tienes varias versiones, puede que necesites usar python3 --version).

3. **Resultado esperado:** Si la instalación fue exitosa, la terminal te mostrará la versión de Python que has instalado (por ejemplo: Python 3.12.2).

RESULTADO: toma una captura del resultado anterior.

Paso 4 (Opcional): Solución de problemas en Windows

Si en el paso 3 Windows muestra un error como python no se reconoce como un comando interno o externo..., es porque la opción de PATH no se marcó durante la instalación. Puedes solucionarlo de la siguiente manera:

1. Vuelve a ejecutar el instalador de Python.
2. En lugar de Install Now, haz clic en **Modify**.
3. Asegúrate de que la casilla de Add Python to PATH esté marcada y completa el proceso.
4. Cierra la terminal y abre una nueva. Ahora el comando python --version debería funcionar correctamente.

2. Consola del Sistema

Objetivo

Practicar la creación y ejecución de scripts Python desde la consola, observando cómo se muestran los resultados directamente en el terminal.

Guía paso a paso para el ejercicio con la consola

Abre la terminal (o Símbolo del Sistema en Windows).

Navega hasta una carpeta de trabajo, por ejemplo: cd Escritorio.

Crea un archivo llamado hola.py con cualquier editor de texto simple (Notepad, TextEdit, etc.).

Escribe el siguiente código y guarda el archivo: print("¡Hola, mundo!").

Vuelve a la terminal y ejecuta el script con el comando: python hola.py.

RESULTADO: toma una captura del resultado de ejecutar el paso anterior.

3. IDLE de Python

Objetivo

Aprender a usar IDLE para crear, guardar y ejecutar scripts Python, interactuando con la consola y observando el resultado de la ejecución del programa.

Paso 1: Abrir el Editor de IDLE Primero, necesitas dos ventanas: el editor (donde escribes el código) y la consola o *shell* (donde ves el resultado).

- Abre IDLE. Automáticamente se abrirá una ventana llamada "Python Shell".
- En esa ventana, ve a File en el menú superior y selecciona New File.
- Esto abrirá una segunda ventana. Esta es tu **ventana del editor**, donde escribirás el código.

Paso 2: Escribir el código en el editor Ahora, escribe el código exacto en la ventana del editor que acabas de abrir.

```
# guardar como saludar.py

nombre = input("¿Cómo te llamas? ")

print(f"¡Hola, {nombre}! Bienvenido a IDLE.")
```

Paso 3: Guardar el archivo Esto es muy importante. IDLE necesita que guardes el archivo antes de poder ejecutarlo.

- En la ventana del editor, ve a File y selecciona Save.
- IDLE te pedirá que elijas un nombre y una ubicación. Guarda el archivo con el nombre **saludar.py** en tu escritorio o en una carpeta que encuentres fácilmente.

Paso 4: Ejecutar el script. Al presionar F5, el código se ejecuta.

- Con el archivo saludar.py aún abierto en el editor, presiona la tecla **F5**.
- Verás que el foco de la pantalla se moverá automáticamente a la ventana de la **consola de IDLE**.

Paso 5: Interactuar con el programa En la ventana de la consola, tu programa estará esperando tu respuesta.

- Verás el mensaje ¿Cómo te llamas? .
- Escribe tu nombre y presiona **Enter**.
- El programa te saludará con el mensaje completo, y el programa habrá terminado su ejecución.

RESULTADO: toma una captura del resultado anterior.

4. Thonny

Objetivo

Practicar el uso del depurador de Thonny para identificar y corregir errores lógicos en un programa Python, observando cómo cambian los valores de las variables durante la ejecución.

Parte 0. Instalación de Thonny

Acceder a la página oficial de Thonny: https://thonny.org

Descargar la versión correspondiente al sistema operativo del ordenador (Windows, macOS o Linux).

Ejecutar el instalador y seguir los pasos de instalación por defecto.

Abrir Thonny. Deberías ver una ventana con el editor de código a la izquierda y la consola de Python a la derecha.

Parte 1. Preparación del código

Crear un nuevo archivo en Thonny: **Archivo → Nuevo**.

Copiar el siguiente código y guardarlo como depurar_thonny.py:

```
def calcular_suma_par(numeros):
    resultado = 0
    for numero in numeros:
        if numero % 2 == 0:
            resultado = numero
    return resultado

lista = [1, 2, 3, 4, 5, 6]
print(calcular_suma_par(lista))  # Salida esperada: 12, pero el resultado será 6
```

Parte 2. Uso del depurador de Thonny

Activar el depurador: Pulsar el **botón con el icono de un bug** en la barra de herramientas.

Ejecutar el código paso a paso:

Pulsar **“Paso a paso”** (o Step) para que el depurador recorra el código línea por línea.

Observar cómo cambia el valor de las variables:

En la ventana de la derecha, se puede ver la variable resultado.

Notar que resultado cambia de 0 → 2 → 4 → 6, en lugar de ir sumando los números.

RESULTADO: toma una captura con el valor de la variable que estás depurando en la penúltima iteración.

Parte 3. Corrección del error

Identificar la línea problemática: resultado = numero

El operador = **sobrescribe** el valor de resultado en cada iteración.

Corregir el código para que sume los números pares:

resultado += numero

Guardar el archivo y ejecutar de nuevo.

La salida ahora debe ser 12, que es el resultado correcto.

Vuelve a usar el depurador para ver los valores de la variable.

RESULTADO: toma una captura con el valor de la variable que estás depurando en la penúltima iteración.

5. Visual Studio Code

Objetivo

Configurar un entorno de desarrollo profesional con Visual Studio Code, crear un entorno virtual para un proyecto Python, instalar librerías externas y documentar las dependencias.

Paso 0: Instalación de Visual Studio Code

Visual Studio Code se descarga desde https://code.visualstudio.com según el sistema operativo. Se ejecuta el instalador siguiendo las indicaciones por defecto. Una vez finalizada la instalación, se abre la aplicación para comprobar su correcto funcionamiento.

Paso 1: Configurar el entorno de trabajo

- Abrir Visual Studio Code.
- Desde el menú superior, seleccionar **Archivo → Abrir carpeta...** y elegir una carpeta vacía para el nuevo proyecto.
- Una vez abierta la carpeta, la barra lateral mostrará el nombre del proyecto.

Paso 2: Instalar la extensión de Python

- Acceder a las extensiones mediante el icono en la barra lateral izquierda (un cuadrado con tres cuadrados más pequeños).
- Buscar "Python" en el buscador.
- Seleccionar la extensión oficial de Python de Microsoft (la primera de la lista) y pulsar **Instalar**. Esto habilita todas las funcionalidades de un IDE para Python.

Paso 3: Crear el entorno virtual

- Abrir la terminal integrada de Visual Studio Code desde **Terminal → Nueva terminal**.
- Ejecutar el siguiente comando para crear un entorno virtual llamado .venv:

```
python -m venv .venv
```

- Tras la ejecución, aparecerá una carpeta .venv dentro del proyecto, que contiene la instalación de Python y las librerías específicas del proyecto.

Paso 4: Activar el entorno virtual

- Activar el entorno virtual desde la terminal. Los comandos dependen del sistema operativo:
    - **Windows:**
- .\.venv\Scripts\activate
    - **Linux/macOS:**
- source .venv/bin/activate
- Si se produce un fallo en PowerShell, existen dos soluciones posibles:

Solución A:

- Abrir PowerShell como administrador.
- Ejecutar:

Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

- Confirmar la acción y abrir una nueva terminal en Visual Studio Code.

Solución B:

- Abrir la terminal cmd.
- Navegar a la carpeta del proyecto.
- Ejecutar .venv\Scripts\activate.

- Si la activación se realiza correctamente, el nombre (.venv) aparecerá al inicio de la línea de comandos.

Paso 5: Escribir un script y gestionar una librería externa

- Crear un archivo llamado web.py en la carpeta del proyecto.
- Copiar el siguiente código:

```
import requests


url = "https://www.google.com"

response = requests.get(url)


print(f"La petición a {url} ha devuelto el código de estado: {response.status_code}")
```

- Inicialmente, la librería requests puede aparecer marcada como error porque aún no está instalada.
- Con el entorno virtual activado, instalar la librería ejecutando en la terminal:

pip install requests

- Una vez instalada, el error desaparece y se puede ejecutar el script con:

python web.py

- La salida esperada será similar a:

La petición a https://www.google.com ha devuelto el código de estado: 200

Paso 6: Guardar las dependencias del proyecto

- Para documentar las librerías utilizadas, ejecutar en la terminal:

pip freeze > requirements.txt

- Esto genera un archivo requirements.txt dentro del proyecto, con la lista de librerías y sus versiones.

RESULTADO: toma una captura de la ejecución del código y del contenido de requirements.txt

6. PyCharm

Objetivo

Comprender la potencia de un IDE profesional como PyCharm para gestionar proyectos con múltiples archivos, facilitando la organización y la colaboración. Se utilizará PyCharm, un IDE diseñado específicamente para Python, con el objetivo de simplificar la gestión de proyectos grandes y complejos.

Paso 0: Instalación de PyCharm

PyCharm se descarga desde la página oficial: https://www.jetbrains.com/pycharm/ según el sistema operativo. Se ejecuta el instalador siguiendo las indicaciones por defecto. Al finalizar, se abre PyCharm para comprobar que funciona correctamente.

Paso 1: Crear un nuevo proyecto en PyCharm

Abrir PyCharm.

En la pantalla de inicio, seleccionar **New Project**.

Asignar un nombre al proyecto (por ejemplo, proyecto-modular).

En la sección **Interpreter**, asegurarse de que PyCharm está creando un Virtualenv nuevo para el proyecto; el nombre por defecto es venv.

Pulsar **Create**. PyCharm tardará unos segundos en configurar el entorno virtual y el proyecto.

Paso 2: Crear una estructura de carpetas modular

En la barra lateral izquierda, hacer clic derecho sobre el nombre del proyecto (proyecto-modular).

Seleccionar **New → Directory** y nombrar la nueva carpeta como src, que se utilizará para almacenar todo el código fuente.

Hacer clic derecho en la carpeta src y seleccionar **New → Python File**, nombrando el archivo como utils.py.

Paso 3: Escribir la función en el módulo utils.py

- Abrir el archivo utils.py y copiar el siguiente código:

```
# src/utils.py


def saludar(nombre):
    """Devuelve un saludo formal."""
    return f"Hola desde el módulo utilidades, {nombre}!"
```

- Guardar el archivo (PyCharm guarda automáticamente, pero se puede usar Ctrl+S).

Paso 4: Crear el archivo principal y usar la función

- Hacer clic derecho en la carpeta src y seleccionar **New → Python File**, nombrando el archivo como main.py.
- Abrir main.py y copiar el siguiente código:

```
# src/main.py


from utils import saludar


# Esta es una buena práctica para que el código se ejecute solo si es el archivo principal
if __name__ == "__main__":
    print(saludar("Ana"))
```

Paso 5: Ejecutar el programa

- En la ventana del editor, hacer clic derecho en cualquier parte del archivo main.py.
- Seleccionar **Run 'main'** en el menú contextual.
- En la parte inferior de la pantalla, la terminal de PyCharm mostrará el resultado:

```
Hola desde el módulo utilidades, Ana!
```

RESULTADO: toma una captura del árbol de carpetas del proyecto.

7. Entornos Virtuales en Python

Objetivo del ejercicio
El objetivo de este ejercicio es **crear, configurar y comprobar** dos entornos virtuales independientes en Python, instalando diferentes versiones de un mismo paquete en cada uno, y verificando que no interfieren entre sí.

Escenario
Se trabajará con dos proyectos distintos, cada uno con su propio entorno virtual. En cada proyecto se instalará una **versión diferente** del paquete requests.
La estructura final deseada será la siguiente:

```
C:\TrabajoPython\
├─ ProyectoA\
│  ├─ .venv\
│  ├─ check_env.py
│  └─ requirements.txt
└─ ProyectoB\
   ├─ .venv\
   ├─ check_env.py
   └─ requirements.txt
```

Parte 1. Preparación del entorno de trabajo

1. Crear carpeta principal y proyectos
En **Windows (CMD o PowerShell):**
mkdir C:\TrabajoPython
cd C:\TrabajoPython
mkdir ProyectoA ProyectoB
Linux/macOS:
mkdir -p ~/TrabajoPython/ProyectoA ~/TrabajoPython/ProyectoB

Parte 2. Creación de entornos virtuales

1. Crear entorno para ProyectoA
cd C:\TrabajoPython\ProyectoA
python -m venv .venv
En Linux/macOS: python3 -m venv .venv
2. Crear entorno para ProyectoB
cd C:\TrabajoPython\ProyectoB
python -m venv .venv

Parte 3. Configuración y pruebas

1. ProyectoA — Instalar requests 2.31.0

a) Activar el entorno
cd C:\TrabajoPython\ProyectoA
.venv\Scripts\activate
Linux/macOS: source .venv/bin/activate
b) Actualizar pip e instalar paquete
python -m pip install --upgrade pip
pip install requests==2.31.0
c) Verificar instalación
python -c "import sys,requests; print(sys.executable); print(requests.__version__)"
pip list
d) Guardar dependencias
pip freeze > requirements.txt

2. ProyectoB — Instalar requests 2.32.3

a) Activar el entorno
Si estás en la misma terminal que ProyectoA, primero desactiva:
deactivate
Ahora activa el entorno de B:
cd C:\TrabajoPython\ProyectoB
.venv\Scripts\activate
b) Actualizar pip e instalar paquete
python -m pip install --upgrade pip
pip install requests==2.32.3
c) Verificar instalación
python -c "import sys,requests; print(sys.executable); print(requests.__version__)"
pip list
d) Guardar dependencias
pip freeze > requirements.txt

Parte 4. Creación del script de comprobación

Dentro de cada proyecto, crea el archivo **check_env.py** con el siguiente contenido:

```
import sys, subprocess

print("Ruta de Python:", sys.executable)

try:
  import requests
  print("Versión de requests:", requests.__version__)
except ImportError:
  print("El paquete 'requests' no está instalado en este entorno.")

print("\nPaquetes instalados:")
subprocess.run([sys.executable, "-m", "pip", "list"])
```

Ejecutar el script

```
python check_env.py
```

Parte 5. Demostración de independencia

1. **Comprobar rutas de Python**
   Ejecuta en cada proyecto:
2. python -c "import sys; print(sys.executable)"

Los resultados deben ser distintos:

```
...\ProyectoA\.venv\Scripts\python.exe
...\ProyectoB\.venv\Scripts\python.exe
```

3. **Comprobar versiones diferentes del mismo paquete**
   En ProyectoA:
4. python -c "import requests; print(requests.__version__)"

→ Debe mostrar 2.31.0.

En ProyectoB:

```
python -c "import requests; print(requests.__version__)"
```

→ Debe mostrar 2.32.3.

5. **No se pueden usar dos entornos en la misma terminal**
   Si activas ProyectoA y, sin desactivarlo, activas ProyectoB, el prompt cambiará, pero solo el **último entorno** quedará activo.
   Para trabajar con ambos entornos **simultáneamente**, usa **dos terminales distintas**.

Parte 6. Exportar e importar dependencias

1. Exportar dependencias

En cada proyecto:

pip freeze > requirements.txt

2. Crear un clon de ProyectoA

mkdir C:\TrabajoPython\ProyectoA_Clonado

cd C:\TrabajoPython\ProyectoA_Clonado

python -m venv .venv

.venv\Scripts\activate

copy ..\ProyectoA\requirements.txt .

pip install -r requirements.txt

python -c "import requests; print(requests.__version__)"

El resultado debe coincidir con **ProyectoA**.

RESULTADO: Ejecutar el script check_env.py, para cada proyecto y tomar una captura con el resultado.

8. Explorando el ecosistema de Python

Ejercicio Autónomo – Trabajo en Grupo

Objetivo

Investigar las herramientas más relevantes y utilizadas dentro del **ecosistema de desarrollo de Python** (IDE y editores de código) y preparar una **presentación** para exponer los resultados a la clase.

Instrucciones

1. Organización de los grupos

La clase se dividirá en **equipos de trabajo**, y cada equipo se encargará de investigar de forma detallada una de las siguientes herramientas de desarrollo:

- **Grupo 1:** IDLE y Thonny
- **Grupo 2:** Visual Studio Code (VS Code)
- **Grupo 3:** PyCharm

2. Investigación detallada

Cada equipo deberá convertirse en **experto** de la herramienta que se le haya asignado. Para ello, se deberá realizar una investigación exhaustiva que incluya los siguientes aspectos:

- **Interfaz principal**:
    - Es fundamental explicar de forma clara la **estructura y organización de la interfaz de la herramienta**, identificando las partes más relevantes, como el **editor de código**, la **terminal integrada**, el **explorador de archivos**, el **depurador** y otros paneles o secciones importantes.
    - Se recomienda incluir **capturas de pantalla** para ilustrar la distribución y facilitar la comprensión.
- **Extensiones o plugins esenciales**:
    - Identificar las **extensiones** o **complementos** imprescindibles para trabajar de manera eficiente con **Python** en la herramienta seleccionada.
    - Ejemplos: la extensión oficial de Python para VS Code, los **plugins de linters**, o complementos para la depuración avanzada.

- **Gestión de entornos virtuales**:
    - Analizar cómo la herramienta permite **crear, configurar y administrar entornos virtuales**, aspecto clave para el desarrollo profesional en Python.
    - Describir si la integración es **nativa** (como en PyCharm o Thonny) o si requiere configuraciones adicionales (como en VS Code).
- **Funcionalidades destacadas**:
    - Identificar las características más útiles o diferenciales de la herramienta.
    - Ejemplos: el **depurador visual** de Thonny, la **integración automática de entornos virtuales** en PyCharm, o las **amplias opciones de personalización** de VS Code.
- **Casos de uso recomendados**:
    - Indicar en qué tipo de proyectos o contextos resulta más adecuado utilizar la herramienta, por ejemplo:
        - Proyectos pequeños o de aprendizaje.
        - Desarrollo profesional a gran escala.
        - Ciencia de datos e inteligencia artificial.

3. Preparación de la presentación

Cada equipo deberá preparar una **presentación de entre 5 y 10 minutos** que resuma de forma clara y estructurada los resultados obtenidos.

- La presentación debe incluir **capturas de pantalla** que muestren:
    - Las principales secciones de la interfaz.
    - Las funcionalidades más destacadas investigadas.
    - Ejemplos de uso relevantes.
- Para la presentación también se podrá realizar una **demostración en vivo** de la herramienta.

RESULTADO: Se debe entregar el documento de la presentación.