



# VIRTUALIZACION

## TRABAJO PRACTICO INTEGRADOR ARQUITECTURA Y SISTEMAS OPERATIVOS

Alumnos: Gabriel Alejandro Ledesma - gledesma10@gmail.com

Tomas Lucas - tomasemmanuelucas96@gmail.com

Materia: Arquitectura y Sistemas Operativos

Profesor: Ariel Enferrel

Fecha de Entrega: 05/06/2025

# Índice

1. Introducción .....	pag 2
2. Marco teórico.....	pag 2
3. Caso práctico.....	pag 5
4. Metodología utilizada.....	pag 6
5. Resultados obtenidos.....	pag 9
6. Conclusiones.....	pag 9
7. Bibliografía.....	pag 11

## **Introducción**

Los estudiantes eligieron el tema "Virtualización y Máquinas virtuales" porque consideran que se trata de una herramienta fundamental en el entorno actual del desarrollo de software. Les resultó interesante la posibilidad de simular diferentes entornos de trabajo utilizando un solo equipo físico, lo que representa una gran ventaja tanto en términos de eficiencia como de seguridad.

La importancia de este tema en su formación como técnicos en programación radica en que le permite comprender y aplicar tecnologías que son ampliamente utilizadas en el mundo laboral. El conocimiento sobre virtualización les brinda la posibilidad de instalar distintos sistemas operativos, realizar pruebas sin comprometer el sistema principal, y entender el funcionamiento de servicios en la nube, que hoy son esenciales para el desarrollo de aplicaciones.

A través del desarrollo de este trabajo, los estudiantes se proponen alcanzar varios objetivos: aprender a instalar y configurar máquinas virtuales, diferenciar los tipos de virtualización existentes, y comprender en qué contextos resulta más conveniente utilizar cada uno. También busca adquirir la capacidad de explicar cómo estas herramientas contribuyen a optimizar recursos, facilitar la administración de entornos y aumentar la seguridad en el trabajo del programador.

## **Marco Teórico**

### **¿Qué es la virtualización?**

La virtualización es un proceso que permite una utilización más eficiente del hardware físico de la computadora y es la base de la computación en la nube.

El software de virtualización crea una capa de abstracción sobre el hardware de la computadora que permite que los elementos de hardware de una sola computadora (procesadores, memoria, almacenamiento y más) se dividan en múltiples computadoras virtuales, comúnmente llamadas máquinas virtuales (VM). Cada máquina virtual ejecuta su propio sistema operativo (SO) y se comporta como una computadora independiente, aunque se ejecuta solamente en una parte del hardware informático subyacente existente.

### **¿Qué es una máquina virtual (VM)?**

Una máquina virtual (virtual machine) es una representación virtual o emulación de un equipo físico que emplea software en lugar de hardware para ejecutar programas e implementar aplicaciones.

## ¿Cómo funcionan las máquinas virtuales?

La virtualización se basa en la tecnología de hipervisores. Esta capa de software ubicada en una computadora o servidor físico (también conocido como servidor bare metal) permite que la computadora física separe su sistema operativo y aplicaciones de su hardware.

Estas máquinas virtuales pueden ejecutar sus sistemas operativos y aplicaciones de forma independiente sin dejar de compartir los recursos originales (memoria, RAM, almacenamiento, etc.) del servidor, que gestiona el hipervisor. En esencia, el hipervisor actúa como un policía de tráfico, ya que asigna recursos a las máquinas virtuales y se cerciora de que no se interrumpan entre sí.

Existen dos tipos principales de hipervisores:

- **Hipervisores de tipo 1**

Se ejecutan directamente en el hardware físico (en general, un servidor), que reemplaza el SO. Normalmente, se emplea un producto de software independiente para crear y manipular VM en el hipervisor.

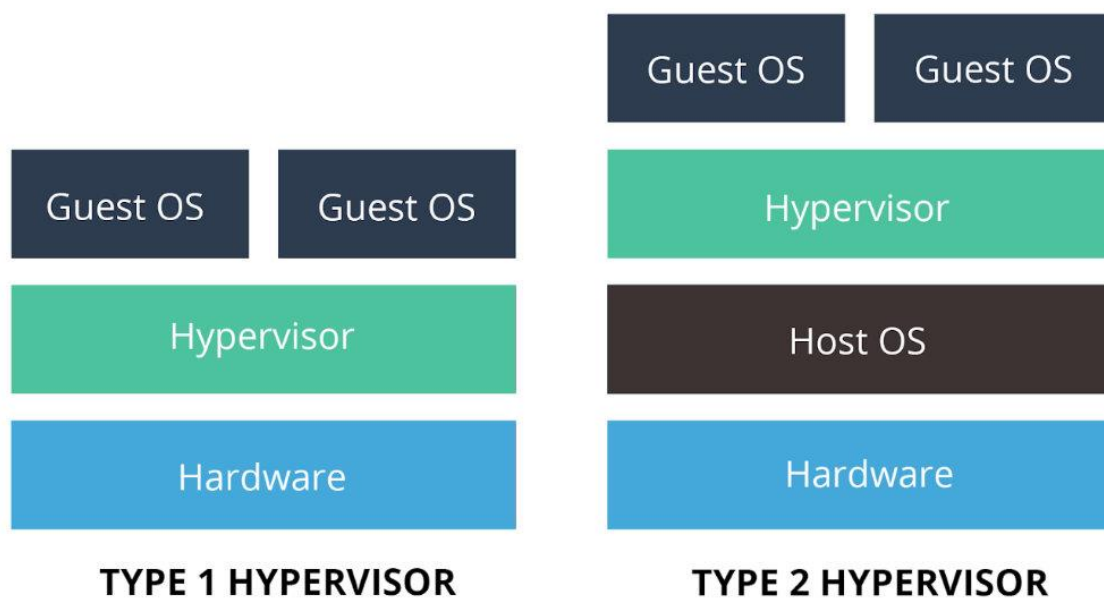
Algunas herramientas de gestión, como vSphere de VMware, le permiten seleccionar un sistema operativo invitado para instalarlo en la VM. Puede usar una VM como plantilla para otras y duplicarla para crear otras nuevas. Según sus necesidades, puede crear varias plantillas de VM para diferentes propósitos, como pruebas de software, bases de datos de producción o entornos de desarrollo. Una máquina virtual basada en kernel (KV) es un ejemplo de hipervisor de tipo 1.

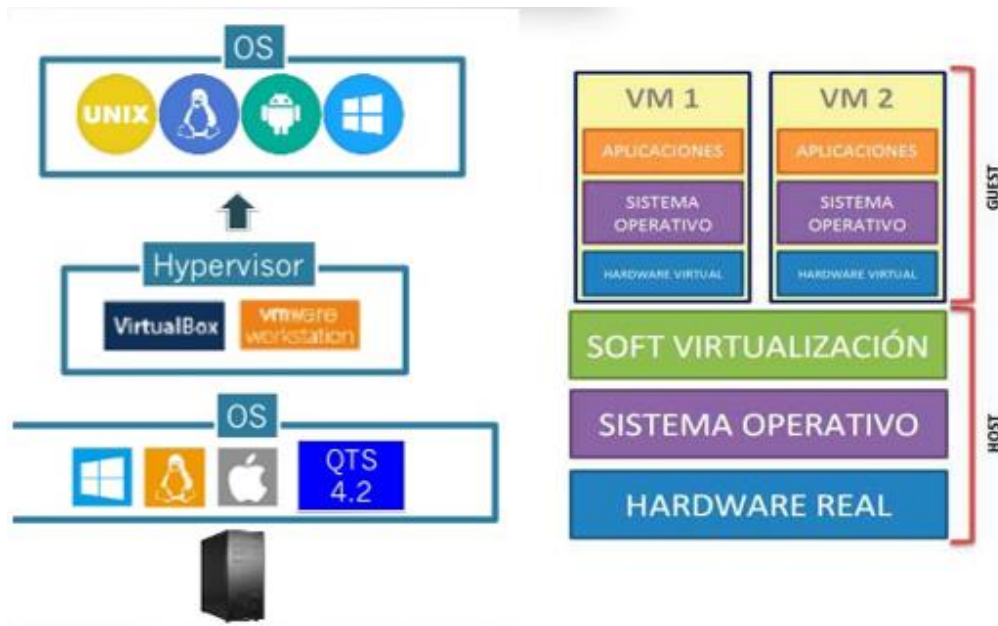
- **Hipervisores de tipo 2**

Se ejecutan como una aplicación dentro de un sistema operativo host y, por lo general, se dirigen a plataformas de escritorio o portátiles de un solo usuario.

Con un hipervisor de tipo 2, se crea manualmente una VM y se instala un SO invitado en su interior. Puede usar el hipervisor para asignar recursos físicos a su VM, lo cual configura manualmente la cantidad de núcleos de procesador y memoria que puede usar. Según las capacidades del hipervisor, puede establecer opciones, como el aceleramiento 3D para gráficos. Los hipervisores de tipo 2 incluyen VMware Workstation y Oracle VirtualBox.

	HYPERVISORES		
	Tipo 1 (Bare-metal)		Tipo 2 (Hosted)
Semejanzas	1. Permiten virtualizar sistemas operativos: Ambos permiten crear y ejecutar máquinas virtuales (VMs) con diferentes sistemas operativos sobre un hardware físico. 2. Comparten recursos físicos: Tanto tipo 1 como tipo 2 gestionan el uso de CPU, RAM, disco y red entre las máquinas virtuales. 3. Aislamiento de VMs: Ambos tipos proporcionan cierto nivel de aislamiento entre máquinas virtuales, aumentando la seguridad y estabilidad. 4. Administración de VMs: Ambos permiten crear, configurar, iniciar, detener y eliminar máquinas virtuales.		
diferencias	Ubicación	Corre directamente sobre el hardware físico	Corre sobre un sistema operativo anfitrión (host)
	Ejemplos	VMware ESXi, Microsoft Hyper-V, Xen, KVM	VirtualBox, VMware Workstation, Parallels
	Rendimiento	Más alto, con mejor acceso directo al hardware	Más bajo, debido a la capa del sistema operativo host
	Uso principal	Servidores, centros de datos, entornos empresariales	Escritorio, pruebas, desarrollo
	Instalación	Se instala como un sistema operativo	Se instala como una aplicación dentro del SO host
	Dependencia del sistema operativo	No depende de un SO anfitrión	Depende del SO anfitrión para funcionar
	Seguridad y Estabilidad	Más seguros, menor superficie de ataque	Menos seguros, afectados por fallos del SO host





## **Caso Práctico**

### Problema planteado

Para aplicar los conceptos de virtualización, se planteó como objetivo simular un entorno de desarrollo controlado en el cual ejecutar un programa en lenguaje *Python*. El problema consistió en desarrollar una aplicación simple que permitiera realizar cálculos con números ingresados por el usuario a través de la consola validando además que el entorno de ejecución fuera completamente funcional desde una máquina virtual.

### Descripción de la solución

Se utilizó el software *Oracle VirtualBox* para crear una máquina virtual con el sistema operativo *Ubuntu*. Una vez instalado el sistema operativo, se procedió a instalar *Python* desde la terminal utilizando el gestor de paquetes *apt*. Dentro de la máquina virtual, se creó un script en *Python* que solicita una lista de números al usuario, elegir una operación a realizar, y luego muestra el resultado por consola.

### Validación del funcionamiento

El funcionamiento del sistema fue validado exitosamente. Al iniciar la máquina virtual, se pudo ejecutar sin inconvenientes el sistema operativo *Ubuntu*, acceder a la terminal, instalar *Python* y correr el programa. Se realizaron múltiples pruebas con diferentes operaciones y valores para verificar la estabilidad del programa. El entorno virtualizado se comportó como un sistema físico, permitiendo el desarrollo, ejecución y prueba del código de manera aislada y controlada.

## **Metodología Utilizada**

### Instalación inicial

Descargar VirtualBox y la imagen ISO de Ubuntu en su versión 24.04.

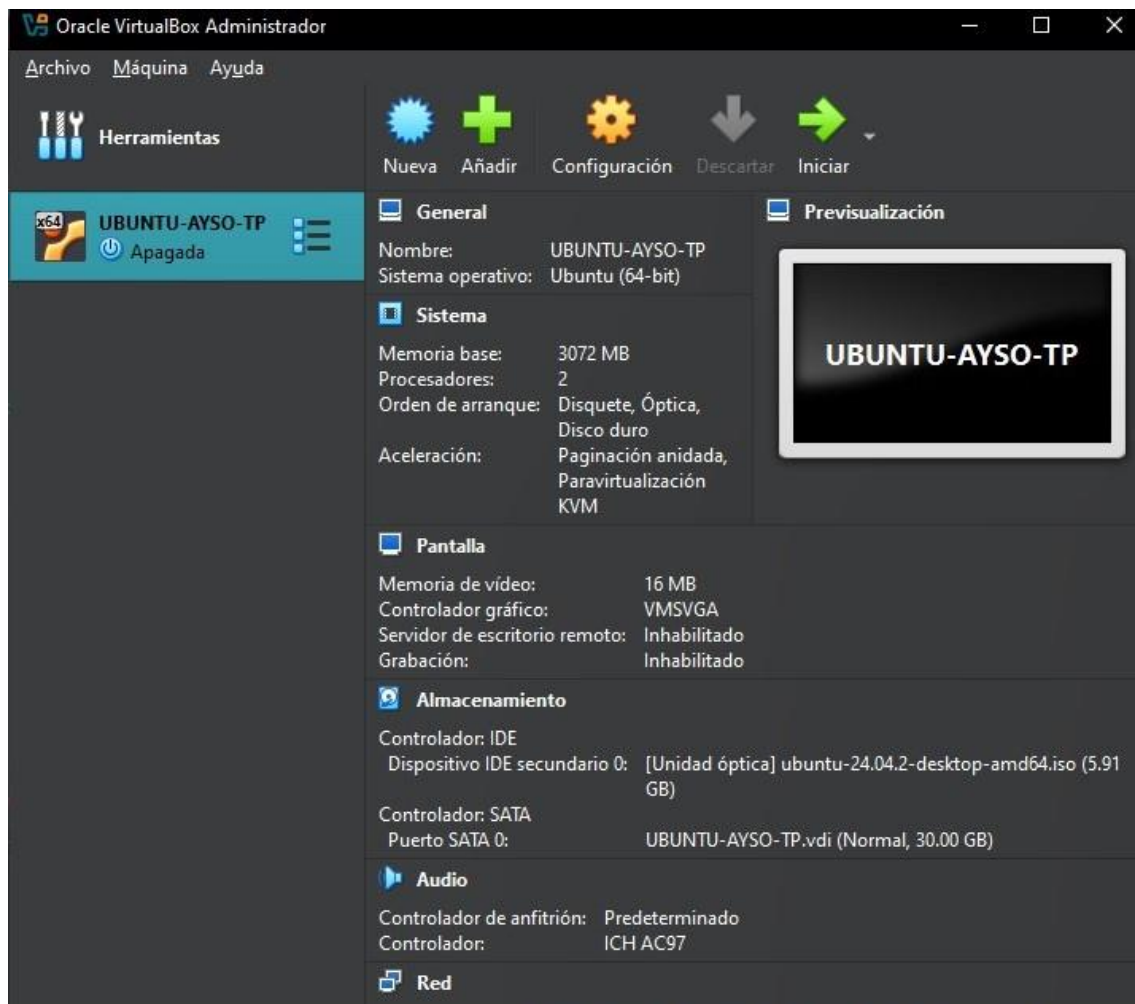
### Configuración de la máquina virtual

Se crea la VM en Virtual box con los siguientes parámetros:

Memoria RAM: 3072MB

Procesadores: 2

Almacenamiento: 30GB



### Preparación del entorno

Instalar Python en Ubuntu ya que no está disponible:

1. Abrir una Terminal

2. Actualizar las Listas de Paquetes:

se usa *sudo apt update* para actualizar



3. Instalar Python 3:

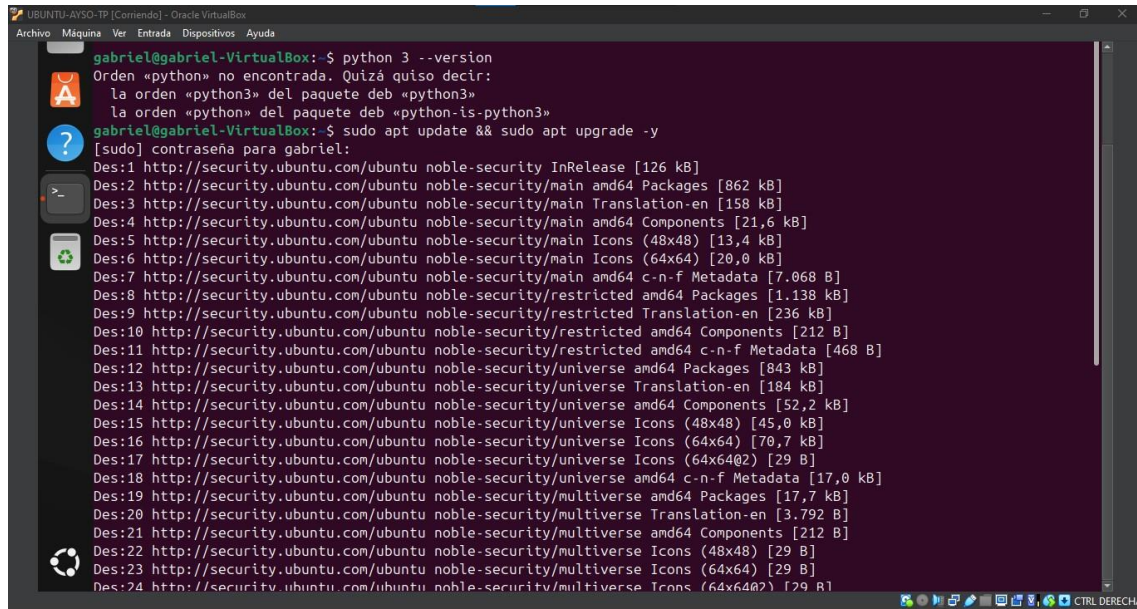
`sudo apt install python3`

4. Verificar la Instalación:

`python3 --version`

5. Instalacion de Paquetes para Python:

`sudo apt install python3-pip`



```
gabriel@gabriel-VirtualBox:~$ python 3 --version
Orden «python» no encontrada. Quizá quiso decir:
  la orden «python3» del paquete deb «python3»
  la orden «python» del paquete deb «python-is-python3»
gabriel@gabriel-VirtualBox:~$ sudo apt update && sudo apt upgrade -y
[sudo] contraseña para gabriel:
Des:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Des:2 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [862 kB]
Des:3 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [158 kB]
Des:4 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [21,6 kB]
Des:5 http://security.ubuntu.com/ubuntu noble-security/main Icons (48x48) [13,4 kB]
Des:6 http://security.ubuntu.com/ubuntu noble-security/main Icons (64x64) [20,0 kB]
Des:7 http://security.ubuntu.com/ubuntu noble-security/main amd64 c-n-f Metadata [7.068 B]
Des:8 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [1.138 kB]
Des:9 http://security.ubuntu.com/ubuntu noble-security/restricted Translation-en [236 kB]
Des:10 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Des:11 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 c-n-f Metadata [468 B]
Des:12 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [843 kB]
Des:13 http://security.ubuntu.com/ubuntu noble-security/universe Translation-en [184 kB]
Des:14 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [52,2 kB]
Des:15 http://security.ubuntu.com/ubuntu noble-security/universe Icons (48x48) [45,0 kB]
Des:16 http://security.ubuntu.com/ubuntu noble-security/universe Icons (64x64) [70,7 kB]
Des:17 http://security.ubuntu.com/ubuntu noble-security/universe Icons (64x64@2) [29 B]
Des:18 http://security.ubuntu.com/ubuntu noble-security/universe amd64 c-n-f Metadata [17,0 kB]
Des:19 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [17,7 kB]
Des:20 http://security.ubuntu.com/ubuntu noble-security/multiverse Translation-en [3.792 B]
Des:21 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [212 B]
Des:22 http://security.ubuntu.com/ubuntu noble-security/multiverse Icons (48x48) [29 B]
Des:23 http://security.ubuntu.com/ubuntu noble-security/multiverse Icons (64x64) [29 B]
Des:24 http://security.ubuntu.com/ubuntu noble-security/multiverse Icons (64x64@2) [29 B]
```

## Resumen del programa desarrollado

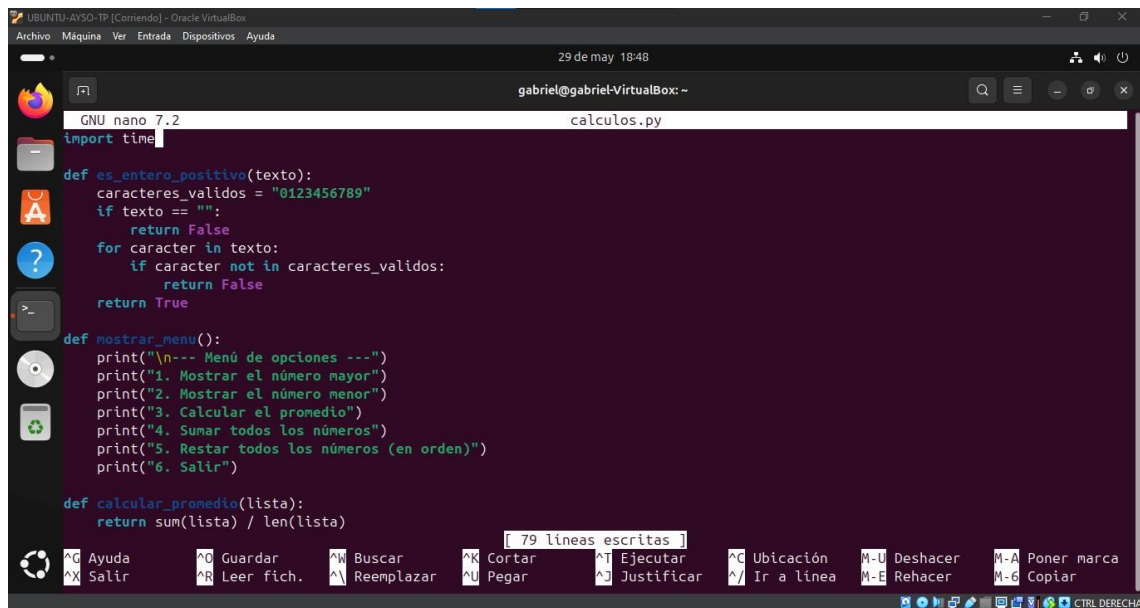
Se desarrolló un programa en lenguaje Python por fuera del entorno virtual cuyo objetivo principal es permitir al usuario ingresar una serie de números enteros positivos y, posteriormente, realizar diferentes operaciones matemáticas sobre los datos ingresados. El funcionamiento del programa se divide en dos etapas: en la primera, el usuario ingresa números por consola hasta que introduce un valor no válido (letra o número negativo). En la segunda etapa, se presenta un menú interactivo con distintas opciones: mostrar el número mayor o menor, calcular el promedio, sumar o restar los valores ingresados, y salir del programa.

## Prueba en Máquina Virtual

Para la implementación del programa dentro de la máquina virtual, se utilizó el editor de texto *nano*, accesible desde la terminal de Ubuntu. Una vez instalado Python se creó un archivo con extensión `.py` en el cual se escribió el código del programa directamente desde la consola.



El uso de nano permitió editar y guardar el script de manera sencilla dentro del entorno virtualizado.



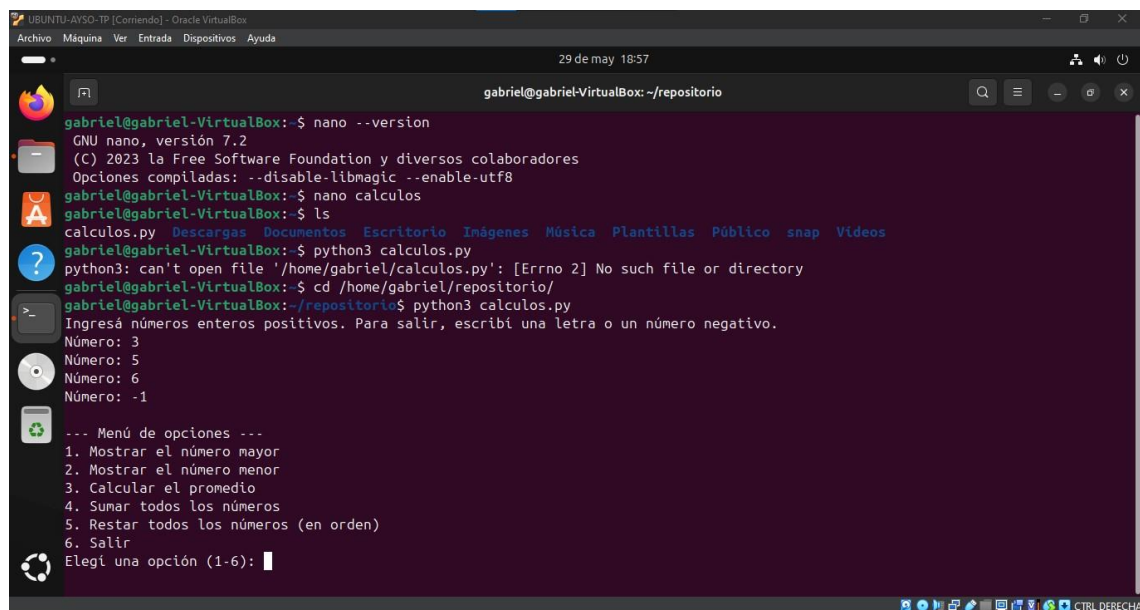
```
GNU nano 7.2 calculos.py
import time

def es_entero_positivo(texto):
    caracteres_validos = "0123456789"
    if texto == "":
        return False
    for caracter in texto:
        if caracter not in caracteres_validos:
            return False
    return True

def mostrar_menu():
    print("\n--- Menú de opciones ---")
    print("1. Mostrar el número mayor")
    print("2. Mostrar el número menor")
    print("3. Calcular el promedio")
    print("4. Sumar todos los números")
    print("5. Restar todos los números (en orden)")
    print("6. Salir")

def calcular_promedio(lista):
    return sum(lista) / len(lista)
```

Posteriormente, el programa fue ejecutado desde la terminal utilizando el comando *python3*, lo que permitió comprobar su correcto funcionamiento dentro de la máquina virtual.



```
gabriell@gabriel-VirtualBox:~$ nano --version
GNU nano, versión 7.2
(C) 2023 la Free Software Foundation y diversos colaboradores
Opciones compiladas: --disable-libmagic --enable-utf8
gabriell@gabriel-VirtualBox:~$ nano calculos
gabriell@gabriel-VirtualBox:~$ ls
calculos.py Descargas Documentos Escritorio Imágenes Música Plantillas Público snap Videos
gabriell@gabriel-VirtualBox:~$ python3 calculos.py
python3: can't open file '/home/gabriel/calculos.py': [Errno 2] No such file or directory
gabriell@gabriel-VirtualBox:~$ cd /home/gabriel/repositorio/
gabriell@gabriel-VirtualBox:~/repositorio$ python3 calculos.py
Ingresá números enteros positivos. Para salir, escribí una letra o un número negativo.
Número: 3
Número: 5
Número: 6
Número: -1

--- Menú de opciones ---
1. Mostrar el número mayor
2. Mostrar el número menor
3. Calcular el promedio
4. Sumar todos los números
5. Restar todos los números (en orden)
6. Salir
Elegí una opción (1-6):
```

## **Resultados obtenidos**

La ejecución del programa dentro de la máquina virtual permitió alcanzar los objetivos propuestos en el trabajo. Se logró simular un entorno de desarrollo funcional y aislado, en el cual fue posible escribir, guardar y ejecutar código en lenguaje Python sin inconvenientes.

Durante la prueba del programa, se ingresaron distintos valores numéricos y se verificaron todas las funciones disponibles en el menú. Cada una de las operaciones arrojó resultados correctos, incluyendo la validación de entradas, el cálculo del número mayor y menor, el promedio, la suma total y la resta secuencial. Además, se comprobaron los mecanismos de salida del programa y el tratamiento de errores ante entradas no válidas.

El entorno virtualizado respondió de manera estable, sin interrupciones ni errores en el sistema operativo anfitrión. Esto confirmó la eficacia de la virtualización como herramienta para el desarrollo y prueba de software en entornos seguros, flexibles y fácilmente configurables.

## **6. Conclusiones**

Durante la realización de este trabajo, como grupo pudimos aplicar conocimientos de programación en Python y además exploramos el funcionamiento de diferentes sistemas operativos, específicamente Windows y Ubuntu en su versión 24.04.

Inicialmente desarrollamos el programa en Python dentro del entorno de Windows para asegurar que la lógica del código funcionara correctamente. Una vez validado, decidimos instalar Ubuntu 24.04 en una máquina virtual para ejecutar el mismo código en un entorno distinto, y así ampliar nuestra experiencia con software multiplataforma.

### ***Dificultades que surgieron y cómo se resolvieron.***

Uno de los primeros inconvenientes que encontramos fue que no podíamos pegar contenido desde Windows hacia la máquina virtual de Ubuntu. Para solucionar esto, habilitamos el portapapeles compartido desde **Dispositivos > Portapapeles compartido > Bidireccional** en VirtualBox e intentamos instalar las *Guest Additions*. Sin embargo, al insertar el CD desde **Dispositivos > Insertar imagen de CD de las Guest Additions**, surgieron errores al ejecutar el instalador.

Para resolverlo, fue necesario instalar varios paquetes adicionales desde la terminal, ya que Ubuntu 24.04 no tenía todo lo necesario por defecto. Ejecutamos la siguiente serie de comandos:

```
sudo apt update
sudo apt install build-essential dkms linux-headers-$(uname -r)
cd /media/$USER/VBox_GAs_*
sudo ./VBoxLinuxAdditions.run
```

Estos pasos instalaron correctamente los headers del kernel y otras herramientas esenciales para compilar los módulos requeridos. Además, instalamos utilidades como bzip2, que también eran necesarias para el proceso. Una vez finalizada la instalación de *Guest Additions*, el portapapeles bidireccional funcionó correctamente, y pudimos pegar el código fuente desde Windows a Ubuntu sin problemas.

Luego utilizamos el editor de texto **Nano**, ya que es una herramienta liviana y fácil de usar en terminal. Con él creamos el archivo calculo.py, donde pegamos el código y lo ejecutamos.

Otro problema importante fue que la máquina virtual no tenía conexión a internet. Para resolverlo, apagamos la VM y modificamos su configuración de red en VirtualBox, cambiando el adaptador de red a **Adaptador puente**, lo que le permitió usar la conexión del sistema anfitrión. Al reiniciar Ubuntu y comprobar conectividad con ping google.com, ya pudimos continuar con las pruebas.

En resumen, este trabajo no solo reforzó nuestros conocimientos de programación — especialmente en el uso de conjuntos, bucles y condicionales—, sino que también nos permitió aprender sobre la instalación, configuración y solución de problemas en entornos Ubuntu. Como mejora futura, podríamos incorporar validación de datos, una interfaz gráfica para el usuario o convertir este proyecto en una pequeña aplicación web multiplataforma.

Este ejercicio demostró cómo la virtualización permite simular un entorno real de desarrollo sin afectar el sistema operativo principal del equipo físico, y validó el uso de máquinas virtuales como herramienta educativa y práctica en la formación como técnico en programación.

## **Bibliografía**

Oracle. (s. f.). *VirtualBox manual*. <https://www.virtualbox.org/manual/>

Canonical Ltd. (s. f.). *Ubuntu server guide*. <https://ubuntu.com/server/docs>

ArchWiki. (s. f.). *VirtualBox*. <https://wiki.archlinux.org/title/VirtualBox>

IBM. (s. f.). *Máquinas virtuales*. <https://www.ibm.com/mx-es/think/topics/virtual-machines>

IBM. (s. f.). *¿Qué es la virtualización?* <https://www.ibm.com/mx-es/topics/virtualization>